

7-4142 776

PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND. (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O.
C A PORUBCANSKY NOV 82

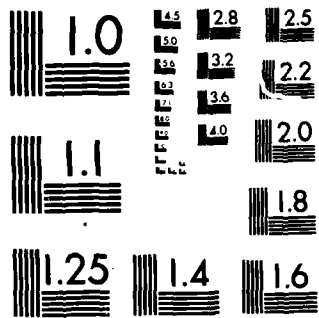
178

UNCLASSIFIED

F/B 9/3

NL

The table consists of a grid of approximately 12 columns and 15 rows. The top row contains a few white cells, while the rest of the grid is almost entirely filled with black redaction marks, obscuring any data that might have been present.



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

AD-A142 776

2nd AFSC STANDARDIZATION CONFERENCE

COMBINED PARTICIPATION BY:
DOD-ARMY-NAVY-AIR FORCE-NATO



30 NOVEMBER - 2 DECEMBER 1982

TUTORIALS: 29 NOVEMBER 1982

DAYTON CONVENTION CENTER
DAYTON, OHIO

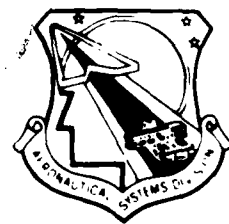
DTIC FILE COPY

SPONSORED BY:



DTIC

HOSTED BY



PROCEEDINGS

Approved for Public Release Distribution Unlimited

84 06 86 113

NOTICE

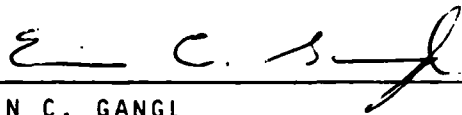
When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.



JEFFERY L. PESLER
Vice Chairman
2nd AFSC Standardization Conference



ERWIN C. GANGL
Chief, Avionics Systems Division
Directorate of Avionics Engineering

FOR THE COMMANDER



ROBERT P. LAVOIE, COL, USAF
Director of Avionics Engineering
Deputy for Engineering

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASD/ENAS, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD(ENA)-TR-82-5031, VOLUME I	2. GOVT ACCESSION NO. AD-A142776	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Proceedings Papers of the Second AFSC Avionics Standardization Conference	5. TYPE OF REPORT & PERIOD COVERED Final Report 29 November - 2 December 1982	
	6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s) Editor: Cynthia A. Porubcansky	8. CONTRACT OR GRANT NUMBER(s)	
9. PERFORMING ORGANIZATION NAME AND ADDRESS HQ ASD/ENAS Wright-Patterson AFB OH 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS HQ ASD/ENA Wright-Patterson AFB OH 45433	12. REPORT DATE November 1982	
	13. NUMBER OF PAGES	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same as Above	15. SECURITY CLASS. (of this report) Unclassified	
	15a. DECLASSIFICATION, DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) N/A		
18. SUPPLEMENTARY NOTES N/A		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Instruction Set Architecture, Multiplexing, Compilers, Support Software, Data Bus, Rational Standardization, Digital Avionics, System Integration, Stores Interface, Standardization, MIL-STD-1553, MIL-STD-1589 (JOVIAL), MIL-STD-1750, MIL-STD-1760, MIL-STD-1815 (ADA), MIL-STD-1862 (NEBULA).		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is a collection of UNCLASSIFIED papers to be distributed to the attendees of the Second AFSC Avionics Standardization Conference at the Convention Center, Dayton, Ohio. The scope of the Conference includes the complete range of DoD approved embedded computer hardware/software and related interface standards as well as standard subsystems used within the Tri-Service community and NATO. The theme of the conference is "Rational Standardization". Lessons learned as well as the pros and cons of standardization are highlighted.		

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This is Volume 1

Volume 1	Proceedings pp. 1-560
Volume 2	Proceedings pp. 561-1131
Volume 3	Governing Documents
Volume 4	MIL-STD-1553 Tutorial
Volume 5	MIL-STD-1589 Tutorial
Volume 6	MIL-STD-1679 Tutorial
Volume 7	MIL-STD-1750 Tutorial
Volume 8	MIL-STD-1815 Tutorial
Volume 9	Navy Case Study Tutorial

PROCEEDINGS OF THE

2nd AFSC

STANDARDIZATION CONFERENCE

30 NOVEMBER - 2 DECEMBER 1982

**DAYTON CONVENTION CENTER
DAYTON, OHIO**

Sponsored by:

Hosted by:

Air Force Systems Command

Aeronautical Systems Division



DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE SYSTEMS COMMAND
ANDREWS AIR FORCE BASE DC 20334

29 AUG 1982

REPLY TO
ATTN: OF CV

SUBJECT Second AFSC Standardization Conference

TO ASD/OC

1. Since the highly successful standardization conference hosted by ASD in 1980, significant technological advancements have occurred. Integration of the standards into weapon systems has become a reality. As a result, we have many "lessons learned" and cost/benefit analyses that should be shared within the tri-service community. Also, this would be a good opportunity to update current and potential "users." Therefore, I endorse the organization of the Second AFSC Standardization Conference.

2. This conference should cover the current accepted standards, results of recent congressional actions, and standards planned for the future. We should provide the latest information on policy, system applications, and lessons learned. The agenda should accommodate both government and industry inputs that criticize as well as support our efforts. Experts from the tri-service arena should be invited to present papers on the various topics. Our AFSC project officer, Maj David Hammond, HQ AFSC/ALR, AUTOVON 858-5731, is prepared to assist.

ROBERT M. BOND, Lt Gen, USAF
Vice Commander

2nd AFSC STANDARDIZATION CONFERENCE

ORGANIZATION COMMITTEE

EXECUTIVE CHAIRMAN

Erwin C. Gangl

EXECUTIVE VICE CHAIRMAN

Jeffery L. Pesler

PROGRAM CHAIRMAN

Jerry L. Duchene

CO-CHAIRMAN

Harold J. Alber

CO-CHAIRMAN

Maj Lee Cheshire

CO-CHAIRMAN

David J. Krile

CO-CHAIRMAN

John Slivinski

EXHIBITS CHAIRMAN

Lt C.W. (Bud) Meygaard

PUBLICATIONS CHAIRMAN

Cindy Porubcansky

SPECIAL ARRANGEMENTS CHAIRMAN

Lt Dennis A. Shoulders

PROTOCOL OFFICER

Capt Francis A. DeCurtis

ADMINISTRATIVE CHAIRMAN

Marie P. Jankovich

TREASURER

Richard H. McBride

CONFERENCE MANAGER

Systems Productivity & Management Corporation

Tuesday Luncheon

Keynote Speaker

Major General Marc C. Reynolds

Major General Marc C. Reynolds is Commander of the Air Force Acquisition Logistics Division, and Deputy Chief of Staff for Acquisition Logistics, Air Force Logistics Command, Wright-Patterson Air Force Base, Ohio.

General Reynolds was born in Chamberlain, S.D., on June 2, 1928, and graduated from Chamberlain High School in 1946. He subsequently attended Dakota Wesleyan University and the University of Denver until the outbreak of the Korean War. He holds a Bachelor's Degree in Political Science from the University of Rhode Island and is a graduate of the Air Command and Staff College and the Naval War College.

General Reynolds entered the Air Force as an aviation cadet in January 1951 at Perrin Air Force Base, Texas, and was commissioned upon graduation from pilot training at Vance Air Force Base, Okalahoma, in February 1952. He then attended jet interceptor training at Moody Air Force Base, Georgia, and Tyndall Air Force Base, Florida.

In July 1952, General Reynolds was assigned pilot duty with the 83rd Fighter-Interceptor Squadron at Hamilton Air Force Base, California, and in September he moved with the squadron to Paine Air Force Base, Washington. In March 1953, he was transferred to the 4th Fighter-Interceptor Squadron at Naha Air Base, Okinawa, where he continued to serve as a fighter-interceptor pilot, flying the F-94B.

His next assignment, in September 1954, was Otis Air Force Base, Mass., where he served with the 437th and 60th Fighter-Interceptor Squadrons as a tactical and training flight commander, flying the F-94C and F-101B, and with the 602d Consolidated Maintenance Squadron as a maintenance officer.

General Reynolds was transferred to Europe in November 1961, assigned to the 10th Tactical Reconnaissance Wing, with duty at RAF Station Bruntingthorpe, England, as a Flight Commander, and later at Toul-Rosieres Air Base, France, as Chief of the Wing Standardization Evaluation Branch.

After Command and Staff College at Maxwell Air Force Base, Alabama, General Reynolds was assigned to the 22d Tactical Reconnaissance Squadron, Mountain Home Air Force Base, Idaho. In November 1966, he moved to the 460th Tactical Reconnaissance Wing at Tan Son Nhut Air Base, Republic of Vietnam, and flew 230 combat missions over North and South Vietnam in RF-4C.

(over)

Following his Southeast Asia tour, he served in Japan as Deputy Chief of the Reconnaissance Division, Headquarters Fifth Air Force, Fuchu Air Station. In April 1970, he moved to Misawa Air Base as Commander of the 16th Tactical Reconnaissance Squadron.

General Reynolds returned to the United States in February 1971, assigned to Shaw Air Force Base, S.C., where he served as Assistant Deputy Commander for Operations in the 363d Tactical Reconnaissance Wing. He attended the Naval War College at Newport, R.I., in 1972-73 and was subsequently assigned to Ogden Air Logistics Center, Hill Air Force Base, Utah, initially as the Director of Distribution and later as Director of Maintenance. In July 1976, he was transferred to McClellan Air Force Base, California, as the Director of Materiel Management, Sacramento Air Logistics Center. In March 1978, he became the Center Vice Commander. He transferred to the Air Force Acquisition Logistics Division in May 1980, where he served as Vice Commander until October 1981, when he assumed his present duties.

General Reynolds is a command pilot with more than 5,200 hours flying time, including 475 combat hours. His military decorations and awards include the Distinguished Service Medal, Legion of Merit, Distinguished Flying Cross, Meritorious Service Medal with one oak leaf cluster, Air Medal with 15 oak leaf clusters, and Air Force Commendation Medal with two oak leaf clusters.

He was promoted to Major General Sept 8, 1980, with date of rank July 1, 1977.

General Reynolds was married to the former Judy Coppage of Falmouth, Mass., who died in February 1982. Their children are Barbara and Scott.

Wednesday Luncheon

Keynote Speaker

Dr. Alan M. Lovelace

Effective 1 Sep 82, Dr. Lovelace was named VP, Productivity and Quality Assurance.

Dr. Lovelace joined General Dynamics Corporation as Vice President, Science and Engineering in July 1981. He had served as Acting Administrator of the National Aeronautics and Space Administration since January of 1981.

Dr. Lovelace joined NASA in 1974 as Associate Administrator for the Office of Aeronautics and Space Technology. He was named Deputy Administrator in June 1976 by President Ford.

Since entering federal service with the U.S. Air Force in 1954, he has held many research management positions. He served at the Air Force Materials Laboratory, Wright-Patterson Air Force Base, Ohio, from 1954 through 1972, having been named Director in 1967.

From 1972 to 1973, he served as Director of Science and Technology with the Air Force Systems Command, Andrews AFB, Washington, D.C. From 1973 to 1974, Dr. Lovelace was Principal Deputy Assistant Secretary of the Air Force for Research and Development.

Dr. Lovelace retired as Deputy Administrator of NASA in December 1980, but stayed with the Administration through the first flight of the Space Shuttle Columbia and the appointment of a new Administrator.

Born in St. Petersburg, Florida, in 1929, Dr. Lovelace received Bachelor's, Master's and Doctoral Degrees in Chemistry from the University of Florida. Awards he has received include the Presidential Citizens Medal, the Department of Defense Exceptional Service Medal, the Air Force Decoration for Exceptional Service, the National Civil Service League Career Service Award, and the Office of Aerospace Research Award for Outstanding Contributions to Research.

He is a Fellow of the American Institute of Aeronautics and Astronautics and the American Astronautical Society, and is a member of the National Academy of Engineering, Air Force Association, Sigma XI and Phi Beta Kappa.

Thursday Luncheon

Keynote Speaker

Charles P. Lecht

Mr. Lecht is President of Lecht Sciences, Inc., a research and think-tank recently established in New York City.

Mr. Lecht is founder and former President/Chairman of the Board of Advanced Computer Techniques Corporation (ACT), a computer software consulting firm.

He holds a B.S. Degree in Mathematics from Seattle University and a M.S. Degree, also in Mathematics, from Purdue. His involvement in the computer field stretches back to 1951, making him an "old-timer" in a very young industry.

Among his earliest professional activities were programming for IBM's Service Bureau and for the MIT community's Lincoln Laboratory/MITRE organizations on a variety of scientific and military simulation projects.

From 1960 to 1962, Mr. Lecht served in the U.S. Army Ordnance Corps, first as Chief of its Programming Division and subsequently of its Mobilization Application Division; Ordnance Industrial Data Agency.

Mr. Lecht came to New York City in 1962, where he founded ACT. In the 17 intervening years, the Company has grown from a one-man show to an international complex employing over 450 persons and deriving more than 50% of its revenues from operations in Europe, Canada and the Middle East as well as the U.S.

In addition to building and presiding over ACT, Mr. Lecht has found time to hold a number of technical posts, author five books and innumerable articles and maintain a heavy schedule of speaking engagements in the U.S. and abroad. In addition to THE WAVES OF CHANGE, his books include three on computer languages and one on project management.

He is a member of the Young Presidents Organization, The Hudson Institute, the Data Processing Management Association, the Association for Computing Machinery and the New York Academy of Sciences.

In 1976, Mr. Lecht was designated by "The Gallagher Presidents' Report" as one of the "10 Best Businessmen in the USA" representing companies with income below \$1 billion. Profiles of Mr. Lecht have appeared in the New Yorker and Datamation, among other publications.

Table of Contents

Volume 1

MIL-STD-1589 Jovial (J-73) High Order Language

	Page
The Evolution of the Jovial/J73 Language from Definition to Use, James T. Pepe, SofTech, Inc.	3
Jovial Standardization, Austin J. Maher, The Singer Company - Kearfott Division	13
Management Overview of the Benefits of Efficient JOVIAL J73/1750A Software Tools, Joel Fleiss, Proprietary Software Systems, Inc.	15
Object Code Optimization in a Standard Compiler, Terence E. Devine, Software Engineering Associates	19
J73AVS: A JOVIAL J73 Automated Verification System, Carolyn Gannon, General Research Corporation	39
JOVIAL Language Control Procedures with a View Toward Ada, Patricia Knoop and Bobby R. Evans, ASD/ADOL, W-PAFB, OH	49
Feasibility Assessment of JOVIAL to Ada Translation, Daniel H. Ehrenfried, AFWAL/AAAF-2	65
The Multiple System OFP Support (MSOS) System, A Pre-PMRT Capability for Evaluating Tactical Software, Marjorie Kirchoff, Intermetrics, Inc. and Victor S. Vajo, Harold Lowery, Air Logistics Command, Warner Robbins AFB	85
Application of JOVIAL (J-73) to Digital Flight Controls, J. H. Robb and P. J. Boatman, General Dynamics, Data Systems Division, Central Center and P. H. Lang, General Dynamics, Fort Worth Division	93

Table of Contents

MIL-STD-1760 Standard Store Interface

	Page
The MIL-STD-1760 Development Program, Claude M. Connell and Bryce M. Sundstrom, Air Force Armament Laboratory, Eglin AFB	103
MIL-STD-1760 Implementation Strategy, Frank T. Woodall, Teledyne Brown Engineering	109
Aircraft-Store Electrical Interconnection System (AEIS) Functional Requirements, J. R. Perkins and D. E. Lautner, Vought Corporation	111
Signal Set Standardization for the Aircraft-Store Electrical Interconnection System, D. E. Lautner and J. R. Perkins, Vought Corporation	123
Consideration of MIL-STD-1760, Aircraft/Store Electrical Interface Standard on Stores Management System Architectures, John E. Sill, Fairchild Space and Electronics Company	139
Aircraft Multiplexing System Architecture and Stores Compatibility, A. DeRuggiero, Grumman Aerospace Corporation and B. Zempolich, NAVAIR	161

Table of Contents

MIL-STD-1862 Nebula 32 Bit Instruction Set Architecture

	Page
The NEBULA and MCF Standardization Programs, Edward Lieblein, US Army Communications - Electronics Command, Ft. Monmouth, NJ	177
The Nebula Standard Computer Architecture, William B. Dietz, Tartan Laboratories, Inc.	187
Transportability of Nebula Software, Guy L. Steele, Jr., Carnegie-Mellon University, Pittsburgh, Pennsylvania	207
The Impact of NEBULA, MCF and ADA on Real-Time Embedded Computer Systems, F. E. Wuebker, RCA Government Systems Division	219
PAS/NEB: A PASCAL Language Tool for the MCF, Elizabeth Souren, Guido Lonardo and Dr. Robert Couranz, Raytheon Company	225

MIL-STD-1553 Multiplex Data Bus

A Programmable Bus Control Interface, Robert M. Salter, Sperry Univac	241
Single Chip MIL-STD-1553B Bus Interface Unit Keeps Pace With Chip Sets, Scott Schaire, Grumman Aerospace Corporation	255
MIL-STD-1553B Marconi LSI Chip Set in a Remote Terminal Application, Albert DiMarino	269

Table of Contents

	Page
MIL-STD-1553 Interface Application Notes, Steve Friedman, ILC Data Device Corporation	277
Application of 1553B to MRASM - A Systems Look, John E. Leib, General Dynamics Convair Division	295
MIL-STD-1553B in MRASM - The Designer's Challenge - Vicki L. Elmore, General Dynamics Convair Division	305
Third Generation MIL-STD-1553B LSI Chip Set, Robin D. Beasley, Marconi Avionics, Ltd, Rochester, Kent, UK	315
Data Word Standardization, Francis E. Peter, Naval Avionics Center	327
MIL-STD-1553B Validation Testing, Duane J. Thorpe and Kumar V. Vakkalanka, ASD/ENAS (SEAFAC), W-PAFB, OH	337
MIL-STD-1553: Testing and Test Equipment, Leroy Earhart, Test Systems, Inc.	349
A Common 1553B I/O Channel for the F-16, Stephen Alford, General Dynamics - Fort Worth Division	367

MIL-STD-1750 16 Bit Instruction Set Architecture

MIL-STD-1750A Users Group, C. Ray Turner, The Boeing Company and Marlin L. Wagner, Sperry Univac Defense Systems	385
---	-----

Table of Contents

	Page
MIL-STD-1750A Microprocessor Chip Set Development, Dr. Thomas A. Longo and Dan Wilnai, Fairchild Camera and Instrument Corporation	395
A High Performance MIL-STD-1750A Microprocessor, T. L. Rasset and J. H. Lane, McDonnell Douglas Aeronautics Company	405
A MIL-STD-1750A Computer Employing the MDAC CMOS-SOS Chipset, Charles Frank and Larry Speelman, ROLM Corporation	413
A New Silicon-on-Sapphire MIL-STD-1750A Microprocessor, Joseph R. Burns, Henry Silcock, Gregory Portanova and Steven Nicholas, Mikros Systems Corporation	429
Delco Electronics Standard Architecture Military Computers, Dr. Clive D. Leedham, Delco Electronics Division, General Motors Corporation	445
The Use of Computer ISA and Software Standards at Westinghouse Defense Electronics Center, Dr. Marvel A. Geyer, John G. Gregory and Harvey R. Moran, Jr., Westinghouse Defense Electronics Center	447
Transportable GPU Chip Set Technology for Standard Computer Architectures, Robert E. Fosdick and Harvey C. Denison, Tracor Aerospace	457
MIL-STD-1750A As A Spaceborne Instruction Set Architecture, Robert N. Constant, Richard C. Fleming, Edwin M. Garcia Marvin Lubofsky, and Donald R. O'Bell, The Aerospace Corporation	479
MIL-STD-1750A Verification Testing, Luis E. Velez, ASD/ENASF (SEAFAC), W-PAFB, OH	493

Table of Contents

MIL-STD-1815 Ada High Order Language

	Page
Update of the State of Ada Language Standardization and Other Ada Related Standards, Col Lawrence E. Druffel, HQ USAF, DARPA Program Office	503
Navy Transition to Ada - Potential for a Fresh Start, Owen L. McOmber, HQ Naval Material Command	505
Introducing Ada into the USAF, Maj David A. Hammond and Capt Michael C. Vinyard, HQ AFSC	507
Ada as a Program Design Language - A Rational Approach to Transitioning Industry to the World of Ada Through a Program Design Language Criteria, Robert M. Blasewitz, RCA Government Systems Division	509
The KAPSE Interface Team, Patricia A. Oberndorf, Naval Ocean Systems Center	519
A Standard Run-Time Executive for Compiled Ada, Ben Hyde, Intermetrics, Inc.	527
The Ada Run-Time Environment, Dr. Joseph K. Cross, Sperry Univac, Defense Systems Division	533
The Ada Work Center - Its Features, Capabilities and Developments, David Babcock, Rolm Corporation	539
A Code of Practice to Constrain ADA, Dr. Tim Swann, Marconi Avionics Limited	541
Use of Ada in System Design: A Case Study, Michael B. Patrick and Hal C. Ferguson, General Dynamics Data Systems Division Central Center	543
Ada Training Considerations, Christine L. Braun, SofTech, Inc.	545

Table of Contents

Volume 2

Standardization Issues - Near Term

	Page
Standards and Integrated Avionic Digital System Architecture, Edward L. Griffin, Martin Marietta Orlando Aerospace	563
Achieving the Benefits of Modular Avionics Design, Stephen W. Behnen, Fred M. Lightfoot and Peter R. Metz, Boeing Military Airplane Company	583
Standard ISA's and VLSI: Two Interacting Trends, Dr. Peter M. Kogge, IBM Federal Systems Division	597
Successful Development/Supply of Standardized Computers in a Period of Rapid Technological Change, Keith B. Dixon, Ferranti Computer Systems Ltd	611
AN/UYK-43 (V) and AN/UYK-44 (V) Program Overview, Capt James P. O'Donovan, Naval Sea Systems Command	613
AN/AYK-14 (V) Program Overview, Henry H. Mendhall, Naval Air Systems Command	615
Navy Packaging Standardization Thrusts, John R. Kidwell, Naval Avionics Center, Indianapolis, Indiana	617
An Introduction to the Avionics Integrity Program, James E. Verdier, ASD/ENASA, W-PAFB, OH	633

Table of Contents

Advanced Systems Architecture

	Page
Avionic Architecture - Past and Future, T.V. McTigue, McDonnell Aircraft Company	641
Elements for Successful Implementation of Computing Standards, Gordon R. England, General Dynamics Corporation	643
B-1B Avionics Applications of Military Standards, L. M. Carrier and G. A. Kinstler, Rockwell International	655
Standards Application to B-1B Avionics Program, H. L. Ernst, Boeing Military Airplane Company	657
The Application of Standards to the TDY-750 (Tigershark) Mission Computer, David W. Geyer, Teledyne Systems Company	659
PAVE PILLAR: A Maturation Process for an Advanced Avionics Architecture, D. Reed Morgan and Lt Col R. Bellem, AFWAL/AAA, W-PAFB, OH	675
FACET - Integration and Standardization Thrusts in US Army AVRADA CNI Development Efforts, Arthur W. Lindberg, US Army Avionics R&D Activity, Fort Monmouth, New Jersey	691
Avionics Control Architecture of Army Helicopter Improvement Program (AHIP), Glenn P. Tomlin, Jr., US Army Avionics R&D Activity, St. Louis, MO	693
Advanced Cockpit - Systems Integration, Graham Roe, British Aerospace P.L.C., UK	695

Table of Contents

The Digital Interface Challenge

	Page
Defense Industry Attitudes About Air Force Interface Standards: Report of an Electronics Industries Association Survey, Peter N. Pocalyko, IBM Corporation and Chandler E. Swallow, Jr., Sperry Univac	721
Digital Avionics Design for Validation, Ellis F. Hitt, Battelle, Columbus Laboratories	729
HH-60D Advanced Avionics Architecture, Ira Glickstein, IBM Federal Systems Division	751
Westinghouse Uses US Air Force - Developed Standards, Carl S. Shyman, Westinghouse Defense Electronics Center	753
A General Purpose Computer Architecture Investigation Facility, Lt. Dean W. Gonzalez, RADC/COEA, Griffiss AFB, NY	767
An Integrated Approach to a Successful Embedded Computer Resource Project, Leo G. Egan, ITT/Federal Electric Corporation	777
Is A Federal Software Engineering Series Needed?, Gwendolyn Hunt, Data Processing Service Center/West	813
Concepts For LHX Avionics, LTC Russell H. Smith, US Army Aviation Center, Ft. Rucker, Alabama	815

Table of Contents

Standardization Issues of the Future

	Page
MIL-Prime Program System for Military Specifications and Standards, Frederick T. Rall, Jr., ASD/EN, W-PAFB, OH	823
Options and Opportunities for Standards - A NATO/AGARD Viewpoint, John T. Shepherd, Marconi Avionics Limited and Louis J. Urban, ASD/AX, W-PAFB, OH	843
Proposed MIL-STD for Avionics Installation Interfaces, Maj Gerald Schopf, ASD/XRX, W-PAFB, OH	861
Fiber Optics for the Future - Wavelength Division Multiplexing, J. Larry Spencer, NASA Langley Research Center	871
Integrated CNI Avionics and Future Standardization, Darlow Botha, AFWAL/AAAI, W-PAFB, OH	889
Architecture, Hardware and Software Issues in Fielding the Next Generation DOD Processors, Ole Golubjatnikov, General Electric Company	899
Standard Avionic Software - The Future Strategy for Cost-Effective Avionics, Edward C. Straub, ARINC Research Corporation	927
Application of Standard System Specification Techniques to the Design of Very Large Scale Integrated Circuits, Dave Jordan, Marconi Avionics Limited	947

Table of Contents

Advanced Standardized Systems/Subsystems

	Page
LANTIRN - Tomorrow's Software Development Today!, Kenneth B. Hawks, ASD/RWNM, W-PAFB, OH	951
AFTI/F-16 Digital Flight Control System Development, James K. Ramage, AFWAL, W-PAFB, OH	957
Quantum Leap in Avionics, W. E. Cantrell, General Dynamics Corporation	959
Integrated Digital Avionics System (IDAS), Peter Boxman, US Army Avionics R&D Activity, St. Louis, MO	975
Embedded Computer Standardization in the Submarine Advanced Combat System (SUBACS), Ronald L. Ticker, Naval Sea Systems Command	977
Standardized Computing System SDS 80, J. Olsson, Ericsson	979
Navy Real Time Signal Processor Development: Second Generation Planned Service Standard, Capt C. B. Robbins, Naval Sea Systems Command	991
Tri-Service Combined Attitude Radar Altimeter (CARA) The Army Perspective, William M. Gill, US Army Avionics R&D Activity, Fort Monmouth, NJ	993
MATE Standardization, Capt Richard E. Farmer, ASD/AEGB, W-PAFB, OH	1005
Digital Intercom, Capt Darian Ross, ASD/AEAC, W-PAFB, OH	1013

Table of Contents

Standardized Software Development

	Page
System Planning Tool to Measure Cost Avoidance Resulting from Independent Assessment Techniques Applied to Test Program Software Development, P. D. Kidd, Technology Development of California	1017
A Corporate Approach to an Embedded Software Development and Support Standard, D. D. Doe, D. E. Hilt and G. B. Wigle, Boeing Aerospace Company; J. P. Bateman, Boeing Commercial Airplane Company; L. L. Tripp, Boeing Computer Services Company and W. F. Jackson, Boeing Military Airplane Company	1029
MIL-STD Defense System Software Development, Deane F. Bergstrom, Rome Air Development Center, Griffiss AFB, NY	1043
Cost/Schedule Management for Software Development, Maj H. Wendt, DCAS PRO/Ford Aerospace Corporation and M. W. Evans, Ford Aerospace Corporation	1053
Software Configuration Management in a Project Environment, Maj H. Wendt, DCAS PRO/Ford Aerospace Corporation and M. W. Evans, Ford Aerospace Corporation	1071
Revising MIL-STD-1679, William J. Egan, Naval Material Command	1083
APSE Database User Scenario, Elizabeth S. Kean, Rome Air Development Center, Griffiss AFB, NY	1085
Architectural and Control Considerations for a High Speed Signal Processor Implemented with an Ada Executive, Steven E. Adams, Intermetrics, Inc. and Thomas R. Butler, Magnavox Company	1097
Avionic Systems Integration Facilities, Mark van den Broek and Paul M. Vicen, AFLC/LOE	1113
Planning of Operational Software Implementation Tool, Aaron Spinak, Proprietary Software Systems, Inc.	1115

EXHIBITORS

ARINC RESEARCH	RAYCHEM
BENDIX	RAYTHEON
BOMAR INSTRUMENTS	RCA
CIRCUIT TECHNOLOGY	ROLM
CONTROL DATA CORPORATION	SANDERS ASSOCIATES
DELCO ELECTRONICS	SCI SYSTEMS
FAIRCHILD CAMERA & INSTRUMENTS	SEAFAC
FAIRCHILD SPACE & ELECTRONICS	SINGER KEARFOTT
GARRETT	SMITH INDUSTRIES
GENERAL DYNAMICS, FT. WORTH	SOFTech
GENERAL ELECTRIC	SPECTRAL SYSTEMS
GRUMMAN AEROSPACE	SPERRY UNIVAC
HARRIS SEMICONDUCTOR	STC COMPONENTS
IEEE	SYSTEM PRODUCTIVITY & MANAGEMENT
ILC/DATA DEVICE CORPORATION	SYSTEMS RESEARCH LAB
INTELLIMAC	TELEDYNE SYSTEMS
INTERMETRICS	TELEFONAKTIEBOLAGET LM ERICSSON
KAISER	TEST SYSTEMS
LITTON	TROMPETER
LORAL	TRW
MCDONNELL DOUGLAS ASTRONAUTICS	UTC/MOSTEK
NORTHROP	WESTINGHOUSE DEFENSE

Authors Index

Adams, Steven E., 1097
Alford, Stephen, 367

Babcock, David, 539
Bateman, J. P., 1029
Beasley, Robin D., 315
Behnen, Steven W., 583
Bellem, R., 675
Bergstrom, Deane F., 1043
Blasewitz, Robert M., 509
Boatman, P. J., 93
Botha, Darlow, 889
Boxman, Peter, 975
Braun, Christine L., 545
Burns, Joseph R., 429
Butler, Thomas R., 1097

Cantrell, W. E., 959
Carrier, L. M., 655
Connell, Claude M., 103
Constant, Robert N., 479
Couranz, Robert, 225
Cross, Joseph K., 533

Denison, Harvey C., 457
DeRuggiero, A., 161
Devine, Terence E., 19
Dietz, William B., 187
DiMarino, Albert, 269
Dixon, Keith B., 611
Doe, Dennis D., 1029
Druffel, Lawrence E., 503

Earhart, Leroy, 349
Egan, Leo G., 777
Egan, William J., 1083
Ehrenfried, Daniel H., 65
Elmore, Vickie L., 305
England, Gordan R., 643
Ernst, H. L., 657
Evans, Bobby R., 49
Evans, M. W., 1053, 1071

Farmer, Richard E., 1005
Ferguson, Hal C., 543
Fleiss, Joel, 15
Fleming, Richard C., 479
Fosdick, Robert E., 457
Frank, Charles, 413
Friedman, Steven M., 277

Gannon, Carolyn, 39
Garcia, Edwin M., 479
Geyer, David W., 659
Geyer, Manvel, 447
Gill, William M., 993
Glickstein, Ira, 751
Golubjatnikov, Ole, 899
Gonzalez, Dean W., 767
Gregory, John G., 447
Griffin, Edward L., 563

Hammond, David A., 507
Hawks, Kenneth B., 951
Hilt, David E., 1029
Hitt, Ellis F., 729
Hunt, Gwendolyn, 813
Hyde, Ben, 527

Jackson, W. F., 1029
Jordan, D., 947

Kean, Elizabeth S., 1085
Kidd, Paul D., 1017
Kidwell, John R., 617
Kinstler, G. A., 655
Kirchoff, Marjorie, 85
Knoop, Patricia A., 49
Kogge, Peter M., 597

Lane, J. H., 405
Lang, P. H., 93
Lautner, D. E., 111, 123
Leedham, Clive D., 445
Leib, John E., 295
Lieblein, Edward, 177
Lightfoot, Fred M., 583
Lindberg, Arthur W., 691
Lonardo, Guido, 225
Longo, Thomas A., 395
Lowery, Harold, 85
Lubofsy, Marvin, 479

Maher, Austin J., 13
McOmber, Owen L., 505
McTigue T. V., 641
Mendhall, Henry H., 615
Metz, Peter R., 583
Moran, Harvey R., 447
Morgan, D. Reed, 675

Nicholas, Steven, 429

O'Bell, Donald R., 479
Oberndorf, Patricia A., 519
O'Donovan, James P., 613
Olsson, J., 979

Patrick, Michael B., 543
Pepe, James T., 3
Perkins, J. R., 111, 123
Peter, Francis E., 327
Pocalyko, Peter N., 721
Portanova, Gregory, 429

Rall, Frederick T., 823
Ramage, James K., 957
Rasset, T. L., 405
Robb, J. H., 93
Robbins, C. B., 991
Roe, Graham, 695
Ross, Darian, 1013

Salter, Robert M., 241
Schaire, Scott, 255
Schopf, Gerald, 861
Shepherd, John T., 843
Shyman, Carl S., 753
Silcock, Henry, 429
Sill, John E., 139
Smith, Russell H., 815
Souren, Elizabeth, 225
Speelman, Larry, 413
Spencer, Larry, 871
Spinak, Aaron, 1115
Steele, Guy L., 207
Straub, Edward C., 927
Sundstrom, Bryce M., 103
Swallow, Chandler E., 721
Swann, Tim Jr., 541

Thorpe, Duane J., 337
Ticker, Ronald L., 977
Tomlin, Glen P., 693
Tripp, L. L., 1029
Turner, C. Ray, 385

Urban, Louis J., 843

Vajo, Victor S., 85
Vakkalanka, Kumar V., 337
VandenBroek, Mark, 1113
Velez, Luis E., 493
Verdier, James E., 633
Vicen, Paul, 1113
Vinyard, Michael C., 507

Wagner, Marlin L., 385
Wendt, H., 1053, 1071
Wigle, G. B., 1029
Wilnai, Dan, 395
Woodall, Frank T., 109
Wuebker, F. E., 219

Zempolich, B., 161

MIL-STD-1589

JOVIAL (J-73) HIGH ORDER LANGUAGE

SESSION CHAIRMAN: Donna K. Gant
General Dynamics Corporation

MODERATOR: Melvin R. Barlow
VP & General Manager of General Dynamics
Data Systems Division

COMPONENT PART NOTICE

THIS PAPER IS A COMPONENT PART OF THE FOLLOWING COMPILATION REPORT:

(TITLE): Proceedings Papers of the AFSC (Air Force Systems Command) Avionics
Standardization Conference (2nd) Held at Dayton, Ohio on 30 November
2 December 1982. Volume 1.

(SOURCE): Aeronautical Systems Div., Wright-Patterson AFB, OH.

JUL 13 1984

TO ORDER THE COMPLETE COMPILATION REPORT USE AD-A142 776 **A**

THE COMPONENT PART IS PROVIDED HERE TO ALLOW USERS ACCESS TO INDIVIDUALLY AUTHORED SECTIONS OF PROCEEDINGS, ANNALS, SYMPOSIA, ETC. HOWEVER, THE COMPONENT SHOULD BE CONSIDERED WITHIN THE CONTEXT OF THE OVERALL COMPILATION REPORT AND NOT AS A STAND-ALONE TECHNICAL REPORT.

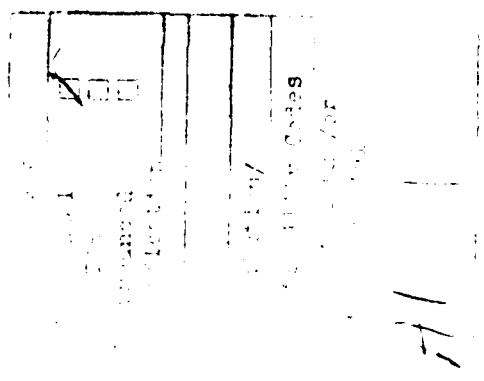
THE FOLLOWING COMPONENT PART NUMBERS COMPRISE THE COMPILATION REPORT:

AD#:	TITLE:
AD-PO03 518	The Evolution of the JOVIAL/J73 Language from Definition to Use.
AD-PO03 519	Management Overview of the Benefits of Efficient JOVIAL J73/1750A Software Tools.
AD-PO03 520	Object Code Optimization in a Standard Compiler.
AD-PO03 521	A JOVIAL J73 Automated Verification System.
AD-PO03 522	JOVIAL Language Control Procedures with a View Toward Ada (Trademark).
AD-PO03 523	Feasibility Assessment of JOVIAL to Ada (Trademark).
AD-PO03 524	The Multiple System OFF Support (MSOS) System, a Pre-PMRT Capability for Evaluating Tactical Software.
AD-PO03 525	Application of JOVIAL (J-73) to Digital Flight Controls.
AD-PO03 526	Mil-STD-1760 Development Program.
AD-PO03 527	Aircraft-Store Electrical Interconnection System (AEIS) Functional Requirements.
AD-PO03 528	Signal Set Standardization for the Aircraft-Store Electrical Interconnection System.
AD-PO03 529	Consederation of MIL-STD-1760, Aircraft/Store Electrical Interface Standard on Stores Management System Architectures.
AD-PO03 530	Aircraft Multiplexing System Architecture and Stores Compatibility.
AD-PO03 531	The Nebula and MCF (Military Computer Family) Standardization Programs,
AD-PO03 532	The Nebula Standard Computer Architecture.
AD-PO03 533	Transportability of Nebula Software.
AD-PO03 534	The Impact of Nebula, MCF (Military computer Family), and ada (Trademark) on Real-Time Embedded Computer Systems.
AD-PO03 535	PAS/NEB: APASCAL Language Tool for the MCF (Military Computer Family).

COMPONENT PART NOTICE (CON'T)

AD#:	TITLE:
AD-P003 536	A Programmable Bus Control Interface.
AD-P003 537	Single Chip MIL-STD 1553B Bus Interface Unit Keeps Pace with Chip sets.
AD-P003 538	MIL-STD-15538 Marconi LSI Chip Set in a Remote Terminal Application,
AD-P003 539	MIL-STD-1553 Interface Application Notes.
AD-P003 540	Application of 1553B of 1553B to MRASM Medium Range Air to Surface Missile-A systems Look.
AD-P003 541	MIL-STD-1553B in MRASM (Medium Range Air to Surface Missile) - The Designer's Challenge.
AD-P003 542	Third Generation MIL STD 1553B LSI Chip Set.
AD-P003 543	Data Word Standardization.
AD-P003 544	MIL-STD-1553B Validation Testing.
AD-P003 545	MIL-STD-1553: Testing and Test Equipment.
AD-P003 546	A Common 1553B I/O Channel for the F-16.
AD-P003 547	MIL-STD-1750A Users Group.
AD-P003 548	MIL-STD-1750A Microprocessor Chip Set Development.
AD-P003 549	A High Performance MIL-STD-175A Microprocessor.
AD-P003 550	A MIL-STD-1750A Computer Employing the MDAC (McDonnell Douglas Astronautics Corporation) CMOS-SOS Chipset.
AD-P003 551	A New Silicon-on-Sapphire MIL-STD-1750A Microprocessor.
AD-P003 552	The Use of Computer ISA and Software Standards at Westinghouse Defense Electronics Center.
AD-P003 553	Transportable GPU (General Processor Units) Chip Set Technology for Standard Computer Architectures.
AD-P003 554	MIL-STD-1750A as a spaceborne Instruction Set Architecture.
AD-P003 555	MIL-STD-1750A Verification Testing.
AD-P003 556	ADA (Trademark) as a Program Design Language. A Rational Approach to Transitioning Industry to the World of Ada through a Program Design Language Criteria.
AD-P003 557	The Kapse (Kernel Ada (Trademark) Programing Support Environment) Interface Team.
AD-P003 558	The Ada (Trademark) Run-Time Environment.
AD-P003 559	A Code of Practice to Constrain Ada (Trademark).
AD-P003 560	Ada (Trademark) Training Considerations.

This document has been approved for public release and distribution is unlimited.



THE EVOLUTION OF THE JOVIAL/J73
LANGUAGE FROM DEFINITION TO USE

James T. Pepe*

SofTech, Inc.
460 Totten Pond Road
Waltham, MA 02154

ABSTRACT

The development of a standard programming language is a multi-year effort involving many phases of activity starting with language requirements analysis, leading to language definition, production of compilers and programming utilities, and then configuration management of the support software and documentation. After a study of the requirements for a standard Air Force high order language, the JOVIAL/J73 language was defined by MIL-STD-1589A (later superceded by MIL-STD-1589B). Several years of compiler development has resulted in JOVIAL/J73 compilers hosted on three mainframe computers and targeted to several embedded architectures. Because of an embedded computer's limited resources considerable effort has been devoted to compiled object code optimization. The Air Force has also sponsored the development of the JOVIAL Compiler Validation System for validating JOVIAL compilers and JOVIAL programming utilities to assist programmers in writing and debugging JOVIAL code. The Language Control Facility has been established to control the definition of JOVIAL/J73, validate compilers, and provide support for JOVIAL programmers.

SECTION 1- INTRODUCTION

The development of JOVIAL/J73 has been supported by several government and industrial agencies. Many people have contributed valuable ideas to the language design, compiler implementation, and standardization efforts. In the course of developing JOVIAL some lessons were learned that are applicable in other language development efforts.

This paper will attempt to present the major events in the development of JOVIAL/J73 as well as the lessons learned. It will focus on the areas in which the author has been personally involved and will not dwell on areas covered by other presentations at this conference.

*Current Address: Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

SECTION 2 - LANGUAGE DEFINITION AND BACKGROUND

The JOVIAL programming language is one of our oldest high order languages. To be accurate, JOVIAL is a generic name encompassing several language dialects. The original JOVIAL specification was defined in 1959 and was followed by several years of language refinement until the first major dialect JOVIAL/J3 (MIL-STD-1588) was developed.

Several compilers for JOVIAL/J3 were implemented and used for major defense programs including SAGE and NORAD. Millions of lines of code have been written in J3 that are still in use and are being maintained.

In the mid-1970s in response to the needs of the avionics community (specifically the B-1 and F-16 programs), the J3 dialect of JOVIAL was revised in accordance with some new ideas of reliable programming that had emerged. Error prone constructs, such as automatic data type conversion, were removed and replaced with those whose use is less likely to result in programming errors. This new dialect of JOVIAL was named J3B and was used by the above two programs as well as the B-52 offensive avionics system.

During this same time period another group in the Air Force attempted to define a dialect of JOVIAL for a wide variety of Air Force applications. The resulting language definition was called JOVIAL/J73. Because of the complexity of this language design, the full language was never implemented. Only a simple subset J73 (Level 1) was implemented. The definition of JOVIAL/J73 (Level 1) is MIL-STD-1589. This language was used by the DAIS program and also as the initial implementation language for the current JOVIAL compilers that will be discussed below.

In the late-1970s the Air Force wanted to standardize on one dialect of JOVIAL that would be used for a wide variety of application areas. The two leading candidates were the J3B and J73 (Level 1) dialects. Each had its supporting agencies but no agreement could be reached on which should be chosen because each dialect had its strong points that no one wanted to sacrifice. To resolve the problem, a language study committee was formed to take the desirable features of J3B and J73 (Level 1) as well as other languages such as Pascal and define a single dialect of JOVIAL for standardization. The result was that in 1978, MIL-STD-1589A was published. This standard defined the JOVIAL/J73 (not to be confused with the previous attempt) programming language.

Two years later MIL-STD-1589A was superseded by MIL-STD-1589B. This later standard resolved some ambiguities in the earlier standard and added a few new language features. It is the later standard that is currently on the DoD list of approved high order languages. It is required to be used on all Air Force avionics applications. The list of programs using JOVIAL/J73 is extensive and includes F-16, F-111, MX, DIS, MCG, MRASM, CX, LANTIRN, MATE, AMRAAM, and Pershing II.

It is interesting to note that although JOVIAL is considered an Air Force language, the Army has also been a user. The J3 dialect was used for PATRIOT, J73 (Level 1) for MLRS, and J73 for Pershing II.

SECTION 3 - COMPILER DEVELOPMENT

A substantial effort has been devoted to implementing JOVIAL/J73 compilers over the last four years. Several software companies have been placed under contract to perform the work. The result is compilers hosted on three mainframes and targeted to about twelve computer architectures including both mainframes and embedded computers. A list of the available compilers is given in Table 3-1.

TABLE 3-1
AVAILABLE JOVIAL/J73 COMPILERS

HOST	TARGET
IBM 370	IBM 370 BENDIX 930 MECA Z8002 MIL-STD-1750A TI-990/9900 INTEL 8086 CP-2EX
DEC-10	DEC-10 MIL-STD-1750A AN/AYK-15 MAGIC 362F Z8002 INTEL 8086
VAX-11/780*	VAX-11/780 MIL-STD-1750A

The compilers were designed for economic retargeting and rehosting. They are organized into two main parts, a front end and a back end. See Figure 3-1. The front end is target machine independent. It performs J73 syntax and semantic analysis, global optimization, and production of a target machine independent intermediate language. The front end comprises 70-75% of the compiler's source code. The back end contains the target machine specific code generator and must be rewritten for each new target machine. The code generator contains the local optimizations for the target machine. The compilers are cross-compilers in that one front end can drive multiple code generators targeted to machines other than the compiler host machine. See Figure 3-2.

*VAX is a trademark of the Digital Equipment Corporation.

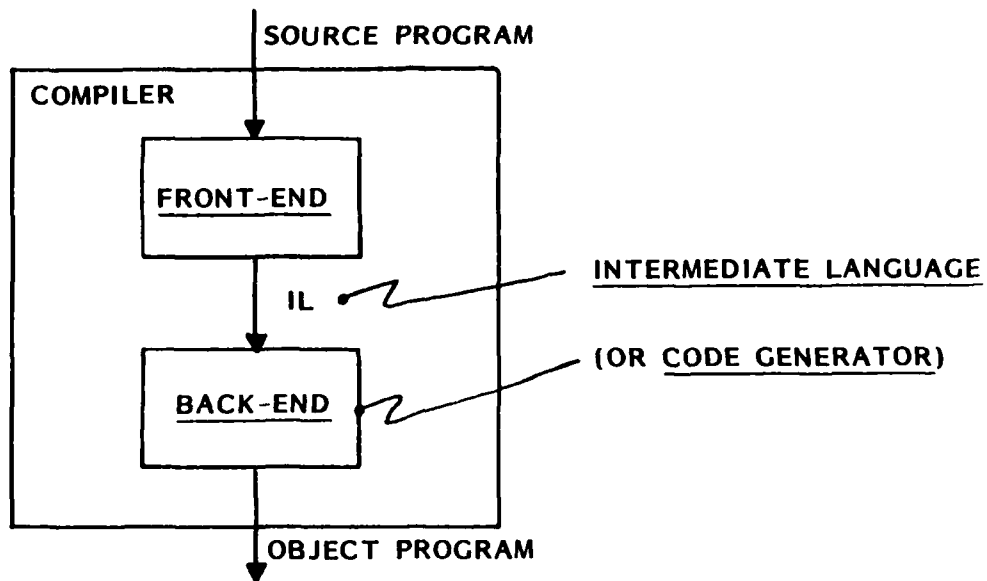


Figure 3-1. Compiler Overview

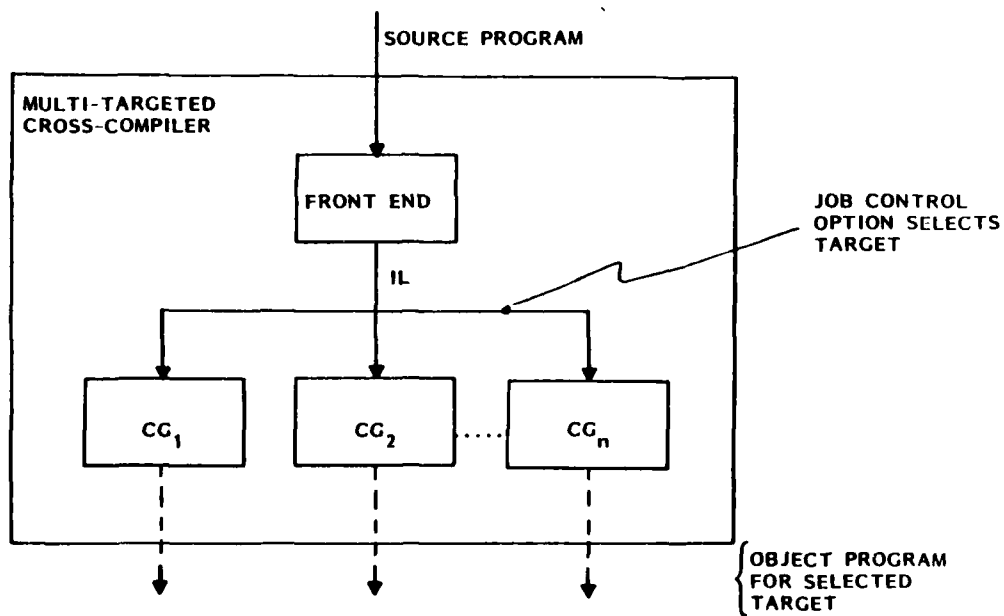


Figure 3-2. Compiler Retargetability

The majority of the compilers are implemented using JOVIAL/J73 as the implementation language. Rehosting these compilers requires first retargeting the compiler to the new host machine and then having this new cross-compiler compile itself. After linking the object code on the new host, this compiler is then in executable form on the new machine.

A recent effort for the F-16 avionics upgrade is sponsoring the development of a MIL-STD-1750A targeted compiler in FORTRAN. This choice for an implementation language will considerably simplify the rehosting process for this compiler. More will be said about this effort in other presentations at this conference.

Many of the JOVIAL/J73 compilers have been completed and are now being maintained for production use. Among the projects that have successfully used JOVIAL for parts of their application coding are MX, Pershing II, DIS, MCG, and MRASM.

Section 4 - OBJECT CODE OPTIMIZATION

The majority of the projects using JOVIAL/J73 are embedded computer applications. Because of the very stringent memory size and CPU speed constraints of embedded computers, the quality of the object code emitted by the JOVIAL compilers is of great concern. Each of the compilers has a set of local and global optimizations included but for certain applications these were not sufficient.

The MX project sponsored the enhancement of the local and global optimizations of the MECA targeted J73 compiler. An extensive set of optimizations has been provided which includes those listed in Table 4-1. The result is a compiler which emits object code with a 12% memory expansion factor and an 11% CPU speed overhead factor as compared to assembly language coding. The factors were measured by a government supplied benchmark having the same operational algorithms coded in both JOVIAL/J73 and assembler. The compiler's performance on the JOVIAL portion as compared with the size and speed of the assembler portion provides the above measurements.

This project demonstrated that JOVIAL/J73 compilers can produce very high quality object code. However, the cost of putting these optimizations in already existing compilers is high. It is best if they are included in a compiler from its design phase.

The optimizations given in Table 4-1 are applicable to almost all embedded applications. They should be especially useful in MIL-STD-1750A targeted compilers. These compilers will be used by a variety of projects and will have to produce efficient object code.

TABLE 4-1
DESIRABLE OPTIMIZATIONS FOR EMBEDDED TARGET

- Common Subexpression Elimination
- Algebraic Simplification
- Dead Code Elimination
- Expression Evaluation at Compile Time
- Value Folding
- Extraction of Invariant Expressions from Loops
- Strength Reduction
- Next Use Information for Register Manager
- Register History Merging
- Peephole Optimizations

SECTION 5 - PROGRAMMING UTILITIES

The current government efforts in developing Ada* compilers includes the development of programming environments as well. Entire systems are being designed and implemented in which the compiler is a utility among other utilities such as text editors, assemblers, linkers, code timing and frequency analyzers, etc. This concurrent design of all the utilities should result in cohesive, well engineered Ada programming systems.

JOVIAL/J73 development did not follow that course. The compilers were developed before the other utilities that now exist. There was never a design for a JOVIAL language system. Despite this fact each of the JOVIAL utilities developed has been well engineered and is a useful product for JOVIAL users.

There are four major utilities apart from the compilers. These are:

- a. Interactive Debugger - DEC-10 hosted symbolic debug package,
- b. Code Auditor - IBM 370 hosted utility to check conformance of JOVIAL/J73 source code to coding standards,
- c. Program Support Library - IBM370 hosted configuration management utility,

*Ada is a trademark of the Department of Defense (Ada Joint Program Office).

- d. JOVIAL Automatic Validation System - IBM 370 hosted utility to assist in automatic testing of JOVIAL object code.

Each of these utilities has been described elsewhere [1-4], and this paper will not go into detail.

SECTION 6 - LANGUAGE CONTROL FACILITY

The increasing importance of standardization in the DoD has led to the development of standard computer architectures and standard high order languages. It is generally accepted that the development and use of the standards will help decrease the high cost of software in tactical systems. The success of standardization depends upon the control of the definition of the standards. The JOVIAL/J73 language definition is controlled by the JOVIAL Language Control Facility (LCF).

The LCF is a branch of the Avionics Systems Division at Wright-Patterson AFB. The major job of the LCF is to control the JOVIAL/J73 language. The LCF has identified four elements of language control which it provides [5]:

- a. a policy for controlling language changes,
- b. a mechanism for making language changes,
- c. a database of information on the language definition, compilers, and other programming utilities,
- d. a mechanism to check conformance of a JOVIAL application with the language definition.

The need for this last element has led to the development of the JOVIAL Compiler Validation System (JCVS). The JCVS consists of about 1300 test cases organized into 6 categories of tests, given in Table 6-1. To ensure conformance with the MIL-STD-1589B specification, the Air Force requires all JOVIAL/J73 compilers to be validated.

TABLE 6-1
JCVS TEST CATEGORIES AND CHARACTERISTICS

TEST CATEGORY	TEST CHARACTERISTICS
A	clean compile
B	known errors to be recognized by compiler
C	implementation dependent tests
D	compiler capacity tests
E	clean compile and execute
F	clean compile and link

The compiler validation is performed by the LCF. Upon completion of testing, the LCF analyzes the results and prepares a report. This report then serves as the basis for judging the compiler's satisfactory compliance with the language definition.

The LCF maintains the JCVS test library. The JCVS is a dynamic set of test cases. As new tests are developed they can be incorporated into the library.

SECTION 7 - CONCLUSIONS

JOVIAL/J73 is in a useable state today. This was made possible by the hard work of several agencies. The language has been used successfully to program several important DoD embedded computer projects and more are in progress.

There are many facets to the development of a standard programming language. Those who have been involved with the evolution of JOVIAL/J73 have discovered the complexity of standardization. Many important lessons were learned in bringing JOVIAL to a useable state. These lessons are applicable to development of other languages, such as, Ada.

The four most important lessons are the following:

- a. Optimizing compilers for embedded targets are complex pieces of software. The same standards that are used for application coding should also be applied to compiler implementation. A sufficient design, coding, and test period should be allowed for a compilers development rather than have it driven by the schedule of the operational programs.
- b. A changing language specification during compiler development opens the door to an implementation disaster. If a major language change is necessary, be prepared to go back to the design phase of the compiler's implementation.
- c. A compiler for an embedded target must generate very efficient object code. Plan for this fact in the compiler's design phase rather than try to retrofit optimizations in later.
- d. A commonly available implementation language on mainframes, such as, FORTRAN (and perhaps later Ada) significantly decreases the cost of compiler rehosting.

REFERENCES

1. JOVIAL Language Control Facility Newsletter, Wright-Patterson AFB, Vol. 3, No. 5, Oct. 1981, pp 3-4.
2. JOVIAL Language Control Facility Newsletter, Wright-Patterson AFB, Vol. 3, No. 4, Aug. 1981, pp 3.
3. JOVIAL Language Control Facility Newsletter, Wright-Patterson AFB, Vol. 4, No. 2, Apr. 1982, pp 7.
4. JOVIAL Language Control Facility Newsletter, Wright-Patterson AFB, Vol. 3, No. 3, Jun. 1981, pp 3-4.
5. Knoop, P. and Evans, B., "JOVIAL Language Control Procedures with a View Toward Ada," NAECON Proceedings, 1982, pp 953-960.

BIOGRAPHY

JAMES T. PEPE

EDUCATION:

S.B., Mathematics, MIT, 1965
S.M., Mathematics, MIT, 1967
Ph.D., Applied Mathematics, MIT, 1972

PROFESSIONAL:

1. C.S. Draper Lab., 1965-1969
 - a. Designed and implemented avionics software for Poseidon Guidance Computer.
 - b. Designed and implemented support software for Poseidon flight simulations.
2. Intermetrics, Inc., 1971-1979
 - a. Project Manager for development of SPL/I language and support software for U.S. Navy.
 - b. Manager of DoD Language System Department, which produced support software for SPL/I, CMS-2, and Fortran languages.
3. SofTech, Inc., 1980-Present
 - a. Manager of JOVIAL Compiler Department which produced optimizing J73 and J3B compilers for MX, Pershing II, MGD, MRASM, B-1, F-16, and B-52 programs.
 - b. Presently Manager of Advanced Development Department, investigating Ada application areas.

JOVIAL STANDARDIZATION

Austin J. Maher
The Singer Company-Kearfott Division
150 Totowa Road
Wayne, New Jersey 07470
(201)785-6607

MIL-STD-1589B defines the JOVIAL (J73) high order programming language, the most recent dialect of a long line of AF languages in the JOVIAL family. Some highlights in the development of the JOVIAL language family and J73 in particular are reviewed as an introduction to this session.

At the present time, the J73 language is the only AF standard language for embedded computer systems. Some technical highlights of the language are presented as well as a review of its current use in AF systems. The key role played by the JOVIAL-Ada Users Group (JUG) in the evolution and use of J73 is reviewed as well as the procedures for handling proposed language changes and compiler validation.

With the apparent imminent availability of Ada, some have questioned the wisdom of continued JOVIAL development and use. To help clarify this situation, the relationship between JOVIAL (J73) and Ada is discussed in the framework of the recently released AF plan for phased introduction of Ada as the replacement for its current standard language, J73.

BIOGRAPHY

Mr. Maher has been active in the field of avionics for over 20 years. Currently, he is the Manager of the Computer Software Engineering Department at the Kearfott Division of the SINGER Company. This Department is the focal point for computer software technology at Kearfott, including the development of realtime software for avionic products and associated automatic test equipment, support software for computer products and general purpose computation facilities for Engineering applications.

Recent activities have included significant contributions to the support of DoD standardization activities, including the AF standard architecture (MIL-STD-1750A), AF standard language (MIL-STD-1589B) and the Ada standard language development. Mr. Maher was selected to participate in the Monterey Workshop sponsored by the Computer Software Management subgroup of the Joint Logistics Commanders' Joint Policy Coordinating Group on Computer Resource Management. He recently completed a two year term as Chairman of the Jovial-Ada Users Group (JUG), a very active organization of industry and government participants. Since its inception in 1978, the JUG has stimulated a high level of communication and cooperation among its participants on computer software standardization initiatives.

Mr. Maher received a BS Physics degree from Holy Cross College and an MS Computer Science degree from Stevens Institute of Technology. His professional affiliations include the IEEE Computer Society and the Ada TEC, SIGPLAN and SIGARCH organization within the Association for Computing Machinery.

MANAGEMENT OVERVIEW OF THE BENEFITS
OF EFFICIENT JOVIAL J73/1750A SOFTWARE TOOLS

Joel Fleiss

Proprietary Software Systems, Inc.
9911 West Pico Boulevard, Penthouse K
Los Angeles, California 90035
(213) 553-2997

Biographical Sketch

Mr. Fleiss is president of Proprietary Software Systems, Inc. He was the original founder of the company in 1969 and has been a project leader in numerous support software projects. He received his undergraduate and graduate degrees in economics at UCLA.

Abstract

The Air Force has invested a significant amount of funds in providing a set of standards for avionics applications. Support software tools for MIL-STD 1750A and MIL-STD 1589B are of critical importance to numerous avionics applications. This article discusses the tools being provided by PSS in support of MIL-STD 1750A and MIL-STD 1589B.

The development of sophisticated computer systems to meet the requirements of DoD applications has proven both costly and time consuming. Of critical importance in developing application software is the methodology utilized by management for controlling the implementation and the availability of efficient, flexible and user friendly software tools.

Software tools provide programmers a mechanism for implementing applications on a computer. A variety of products can be classified as software tools, including:

- o Operating Systems
- o Language Processors (compilers, assemblers, link editors)
- o Source Editors
- o Debug Systems
- o Data Base Systems
- o Utilities
- o Etc.

This talk addresses itself to the current set of tools provided JOVIAL J73 1750A application programmers by PSS. These tools are:

- o An optimizing JOVIAL J73 Compiler
- o A Macro Cross Assembler (DUAL)
- o A Link Editor (DUAL)
- o A 1750A Simulator
- o A Code Auditor

Prior to discussing the 1750A tools, it is best to prioritize what characteristics are the most desirable:

(1) Availability

The compiler and support tools must be operational on a convenient host computer.

(2) Reliability

All support tools must be as error free as possible. This implies that the tools and their interfaces have been thoroughly validated prior to release.

(3) Ease of Use

The tools should be user friendly. They must be compatible.

(4) Object Code Efficiency

The tools should be as efficient as possible. In particular, the compiler should minimize the memory requirements/execution time of the translated program.

(5) Minimal User Restrictions

No part of the tools set should restrict any other tool.

(6) Excellent User Documentation

The documentation should be thorough, concise, and comprehensive.

It is critical that the support software be available on a host that is readily accessible. The PSS J73 1750A compiler and support software is currently operational on the following host computers:

<u>HOST</u>	<u>OPERATING SYSTEMS</u>
IBM-370 (or equivalent)	OS, MVS, CMS
DEC-10	TOPS
DEC-20	TOPS
VAX-780	VMS

PSS in its latest rehosting effort (VAX-780) has made extensive modifications to the compiler's configuration (in particular the back end) in order to simplify future rehosting efforts.

Next to being available, the next most important characteristic is that the compiler be reliable. The Air Force has developed a set of test cases which are used as an integral part of verifying the reliability of a compiler. These test cases are grouped into six classes (A-F).

PSS is making significant improvements in providing its customers reliable, well documented support tools. In addition to the JCVS test suite, PSS also includes self-compiling, utilization of unclassified application procedures provided by customers, numerous special test cases, and test cases representing existing and old software problem reports (SPR's) as part of its validation process.

In addition, PSS has established a separate Q/A department from its compiler staff. The Q/A department's major function is to act as a buffer between PSS's development staff and the eventual users. All PSS products are delivered with exhaustive validation by the Q/A department.

In order to improve the ease of usage of the MIL-STD 1750A support tools, PSS has implemented a number of enhancements. Included in these enhancements are:

- o Symbolic Debug

- The JOVIAL compiler and the PSS provided support tools allow the user to access JOVIAL user symbols and their attributes. These symbols are output as part of the DUAL load module.

- o User Controlled Allocation

- The compiler separates each compilation unit into four distinct sections. The user can at link time concatenate these four sections from multiple compilation/assembly units or specify specific locations for any particular section. In addition, the assembler allows the user to specify up to 32 distinct sections per assembly. This simplifies placing constants and code into read only memories and placing all variables into a contiguous or specific set of memory locations.

- o Simpler Assembly Syntax

- Utilization of symbolic register assignments, consistent argument format, more meaningful mnemonics, and simpler argument delimiters.

- o Improved Compiler Expanded Listing

- More meaningful user names, reduction in listing of data variables with duplicate preset values, etc.

Nearly all command and control applications are concerned with efficiency. Both speed and memory utilization are of critical importance to users. The only important disadvantage of a JOVIAL implementation versus an assembly language implementation is efficiency. Therefore, it is critical that the JOVIAL compiler be as efficient as possible.

PSS has recently completed two major contracts which have vastly improved the efficiency of the existing JOVIAL J73 1750A compiler. In conjunction with our two customers, Boeing and General Dynamics, PSS has implemented the following optimization enhancements to the MIL-STD 1750A J73 code generator.

- o Subroutine Linkage

 - Usage of registers to pass arguments.

- o Automatic Allocation

 - Allocation of automatic data on stacks.

- o Base Register Emphasis

 - Better utilization of Base Registers causing a significant number of short instructions being generated.

- o Literal Pools

 - All literals across compilation units are pooled assuring only a single instance of each literal.

- o Machine Specific Procedures

 - Efficient utilization of MIL-STD 1750A instruction repertoire.

PSS initial test cases show an improvement of approximately 39% in memory utilization. That is, where the older version of the 1750A compiler generated a 1000 instructions, the new PSS version will generate approximately 610.

The JOVIAL language is a general purpose higher order language. It is important that the tool set provide an environment which simplifies the development and maintenance of command and control applications. For example, JOVIAL allows user name of 1 to 31 characters. It would be disastrous to use a tool set that supported names of 6 to 8 characters, thus forcing the JOVIAL programmer to utilize short names and eliminating a major feature of the language.

The PSS provided support tools permit symbolic names from 1 to 16,383 for either the assembler or link editor. In addition the PSS provided support tools allow a greater separation of assembly/compilation units, thus providing the user the mechanism for controlling the program's allocation.

In addition to developing support software PSS is also formalizing a new system for managing the development of large scale application software. While the software industry has emphasized recently better support tools, and programming methods (e.g., structured programming), very little has been done in the area of providing management a means of properly planning large scale software development.

OBJECT CODE OPTIMIZATION IN A STANDARD COMPILER

Terence E. Devine
Software Engineering Associates
23864 Hawthorne Blvd., Suite 200
Torrance, CA 90505
(213) 373-8901

Biographical Information

Terence E. Devine is a Senior Software Scientist with Software Engineering Associates. His research interests include compilers, programming languages, and programming environments. He received a BS in Computer Science from Michigan State University in 1971, an MS in Computer Science from Rutgers University in 1974, and is currently working on his doctoral dissertation, Automatic Code Generation, at UCLA.

Abstract

This paper discusses the generation of efficient object code. It covers ongoing work on JOVIAL compilers for various target machines. The cost and maintenance advantages of an optimizer which serves multiple targets are examined. Performance trade-offs with respect to single-target optimizers are considered.

The effects of JOVIAL, a military-standard language, and its application on optimizer characteristics are explored. Contrasts are made to optimizers for other languages, such as Pascal and Ada.*

The difficulties inherent in comparing the quality of compiled code to assembly language are discussed. These difficulties include finding a suitable basis for comparison, determining what is a "typical" application, and avoiding bias in measurement.

Introduction

The generation of efficient object code has long been of concern to compiler implementors. The developers of the original FORTRAN compiler felt that their compiler had to generate good code, or the compiler would be in danger of not being used [1].

* Ada is a registered trademark of the U.S. Government

In the intervening time numerous papers have been written on the subject of generating more efficient code. Optimization bibliographies can be found in [2,3].

Although hardware speeds have increased, and costs have decreased since that first FORTRAN compiler was built, there is still a need for efficient code generation for certain applications, particularly real-time systems.

It should be noted that the term "optimization" is a misnomer, because optimizing compilers do not generate optimal code, but rather code which is more efficient than that generated by non-optimizing ones. Nonetheless, because the term is used universally, it will be used here also.

"Standard" as used in the title of this paper has several meanings. First, the compiler under discussion compiles the (JOVIAL) J73 language which is specified by MIL-STD-1589B. Second, many compiler modules are standardized, in that the same module is used for a variety of hosts (machines on which the compiler runs) and targets (machines on which the compiled programs are run). In addition, the bulk of the compiler code is written in J73.

Background

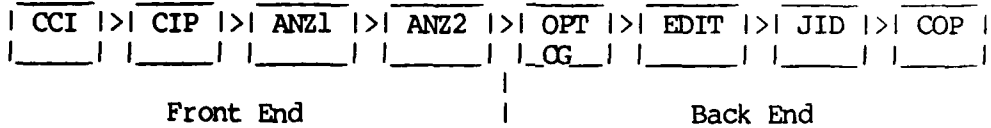
In order to provide a framework for the optimization discussion, a brief history of the compiler and a short description of its operation are provided here. More detailed descriptions of the algorithms used are contained in [4].

The original J73/I compiler for the DEC-10 host and target was developed in 1974-1975 by Computer Sciences Corporation. As part of the J73 JOCIT contract, Software Engineering Associates rehosted the compiler onto the IBM 370. This was done first by retargeting the compiler to the 370, recoding those parts of the compiler which were written in assembly language, and then transporting the compiler to the 370. One important fact to note is that, with the exception of the control card processor, the J73/I source was the same for the two hosts (although a number of modules differed for the two targets). The conditional compilation facilities of J73 were used to allow for differences in word size and other host dependencies, where this could not be done using machine parameters.

Later, the compiler was upgraded to handle the J73 language (MIL-STD-1589B). More recently the compiler itself was translated to J73 using the J73/I to J73 Translator. The translated compiler has also been rehosted to the VAX.

There is still a single set of source code for modules which are common to the various targets. A master copy of the source is maintained on one system, although other systems may be used for development work. When changes to common modules are made at other sites, they are integrated back into the master.

The general structure of the compiler is as follows:

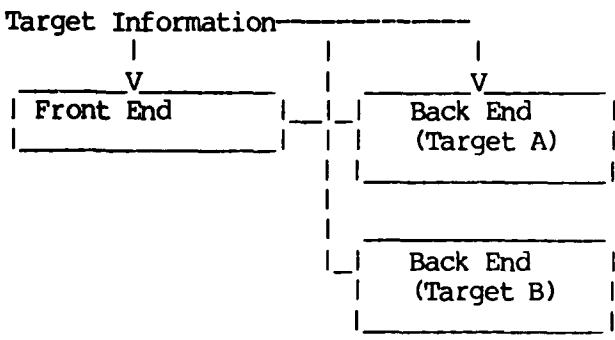


The front end consists of phases which are largely target-independent. The back end phases are largely target dependent. The Control Card Interpreter processes the compiler options. CIP performs Compool InPut. ANZ1 and ANZ2 are the analysis phases. They perform syntactic and semantic analysis of the source program to insure that it conforms to the standard, and produce an internal form of the program. The optimizer modifies this internal form to make the generated object code more efficient. The code generator selects instruction sequences to implement the source program. The editor produces a number of outputs including relocatable object (linker input), assembly language source and cross reference/attribute listings. JID produces tables for use by the JOVIAL Interactive Debugger. COP performs Compool OutPut.

Except for the phases which handle comppools, which are peculiar to JOVIAL, the compiler structure is typical of multi-pass compilers.

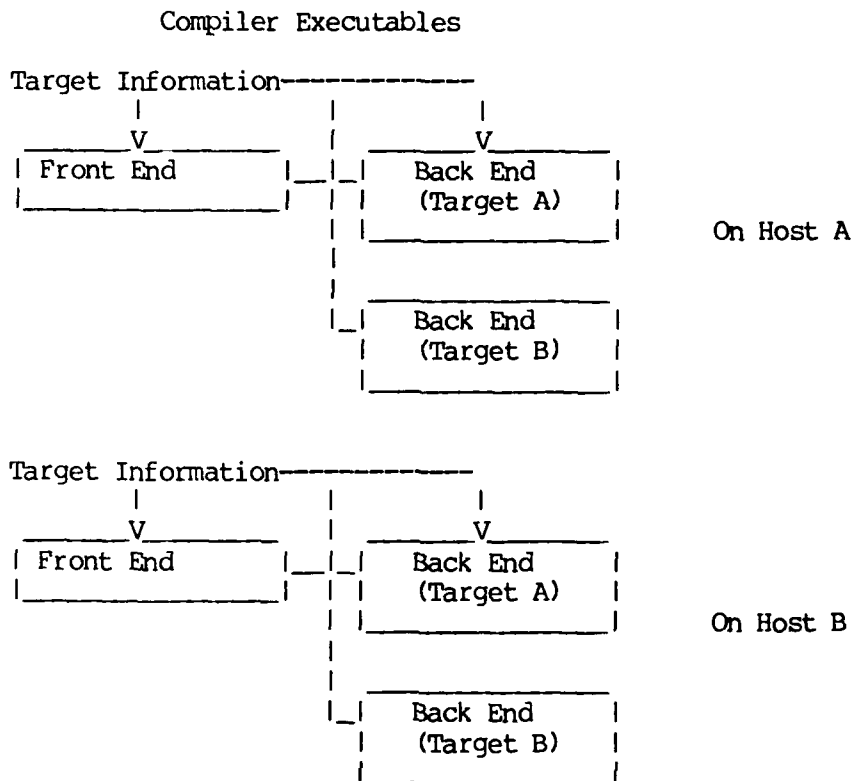
Although CCI is part of the front-end it is host-dependent. COP is target-independent, even though it is part of the back end.

At a more global level the compiler structure can be thought of as:

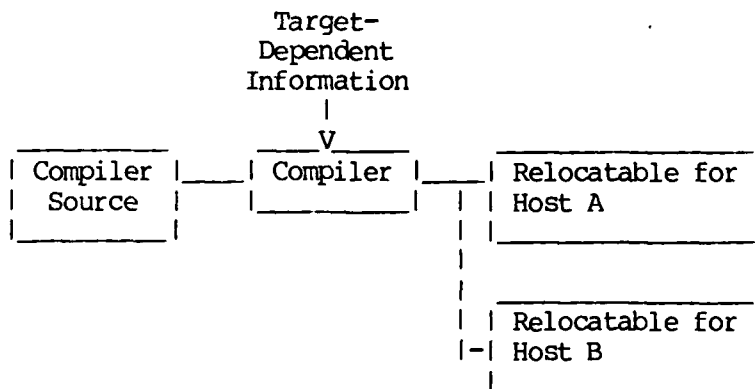


This depicts a compiler with several targets on a single host computer.

At a still more global level we have:



where the compilers for the different hosts are created by:



A common compiler source module is used to create relocatable object modules by selecting the appropriate target option and by supplying the appropriate !SKIP directives.

The present J73 compiler contains a regional optimizer phase, which performs various optimizations over a region of the program. A region is a

series of source statements terminated by a backward branch, or limited by the size of the optimizer's internal tables. Optimizations such as common subexpression elimination, and constant arithmetic are performed on the statements in the region. In the course of common subexpression elimination, reference counts are computed for use in register allocation. The optimizer also performs some local optimizations such as algebraic simplifications.

Register allocation, except for certain dedicated registers, is performed by the code generator. Values in registers are remembered across forward flow, within a region.

There is no peephole optimizer, per se, but some common peephole optimizations are performed by other phases, such as the code generator.

For those targets for which it is applicable the editor phase performs certain branch optimizations. In particular, if shorter branch sequences than the worst case sequences can be used for a particular branch, then the short sequences are used.

Advantages of Optimization

The primary advantage of optimization is that optimized programs use fewer resources at execution time. They normally will execute faster and require less space than unoptimized programs. An additional benefit which may be derived from an optimizer is enhanced diagnosing capability for the compiler. An optimizer which performs flow analysis may be able to determine that a given variable is set before it is used, that code can never be executed, or that a loop is non-terminating. By detecting and diagnosing such conditions, the compiler can aid the programmer in producing a correct program.

Optimization and Compiler Testing

The higher the degree of optimization present in a compiler, the more difficult it is to devise a comprehensive series of test cases. A compiler which does no optimization could be tested relatively well with a standard suite of tests, such as the JOVAL Compiler Validation System (JCVS) tests, which test language features, usually in isolation. Because optimization takes into account the interaction of various statements and language features, there are a greater number of possibilities for testing.

This problem can be mitigated somewhat by having a standard compiler, because this broadens the user base and provides the compiler with greater exposure to test programs and production programs.

Training a Compiler

One danger of using benchmarks to test compiler performance is that it is possible to train the compiler to do well on the benchmarks without improving its performance for real programs. This can happen if the benchmarks do not

accurately reflect the composition of application programs.

As an extreme example consider a test case which performed no input. It would then be possible for the compiler to interpret the program at compile-time and produce generated code only for those statements which performed some sort of output. The compiler's performance would be impressive for this particular test case, but it is highly unlikely that a real program of any significant size, which performed no input, would be written.

It should be noted also that the same is true with respect to validation tests. Validation suites typically test individual language features, rather than the interactions between features. Thus, a compiler may be trained to pass the tests, but may not be suitable for production use.

Effect of Optimization on Debugging

The presence of a global optimizer complicates debugging, whether it is being done at the source or machine level. Currently, there is no source-level interface in the J73 compiler for the 370 target. (Such interfaces exist in the DEC-10 targeted compiler, though.) There are a number of problems which arise from the optimization of code. At present, the problems have not been well-explored. There is a paper which deals with this subject [5]; some additional ideas are contained in [6,7].

The primary difficulties arise in relating what is happening at the machine level to what the programmer originally wrote. The loop optimizations, and those which delete code, such as dead store elimination, serve to obscure what is happening to the program. Optimizations are made assuming that there must be a match between the results of optimized and unoptimized programs, but do not necessarily preserve intermediate states. An operation such as loop fusion, for example, changes the order in which statements are executed. In an unoptimized program it would be possible to stop after the execution of the first loop to verify that it worked as intended. If loop fusion had been applied, there would be no point after which the first loop had finished, but before the second had started. Code motion is another optimization which could cause confusion. A programmer might like to set breakpoints before and after a statement is executed if he thinks that that particular statement is suspect. If part of the statement has been moved, what is the effect of setting a breakpoint before and after that statement? Where should the program actually be interrupted?

Lower-level debugging is more complicated, also. Patching is risky for unoptimized code, but virtually suicidal in the presence of global register allocation, code motion and strength reduction. In addition it would be extremely difficult to produce a reasonable interspersed source and assembly listing in the presence of code motion. Common subexpression elimination, folding, variable overlaying and dead store deletion complicate finding what the values of variables are at a given time.

At present there are many unanswered questions with respect to providing effective source-level debugging in the presence of optimized object code.

Additional research is required to determine what the best solutions are, and if reasonable results are possible.

Determining the Effectiveness of Optimizations

There are any number of optimizations which can reduce a program's resource requirements when they are applied to that program. However, not all optimizations are applicable to all programs. Ideally, it would be possible to predict the expected effect of various optimizations, alone and in combination, on real programs. This would require a knowledge of how much time or space a given optimization would save, as well as some indication of the probability of the optimization being applicable. It is possible to obtain a reasonable approximation of the former by explaining the cases to which the optimization is applicable. However, there are no good statistics available. Of the published work [8] comes closest, although his statistics deal primarily with improvements for some specific examples. One oft-quoted statistic from his work is that 4% of the program usually accounts for over 50% of the total execution time. An incomplete study at CMU indicates that there are a number of optimizations which provide a small, but appreciable, improvement [9].

An additional problem is that most of the currently available statistics for program characteristics are static ones. Even static statistics are not available for the applicability of optimizations. Furthermore, if they were, they would not necessarily be applicable to the problem at hand. For example, even if all possible statistics were available for commercial application, they probably would not accurately reflect the composition of military application programs.

There have been a number of studies of the characteristics of computer programs. A list of such studies can be found in [10]. The most famous is that of Knuth [8] which dealt with FORTRAN programs and possibilities for optimization. The studies which are most relevant to this report are [10,11,12]. [10] contains statistics for a number of characteristics for JOVIAL applications programs. [11] and [12] contain statistics for compilers. The latter contains figures for the J73 compiler for both the DEC-10 and IBM hosts. Although the figures are for the J73/I coded version of the compiler, they reflect the coding style used in the compiler.

Unfortunately, not much information is available regarding the expected payoff for various optimizations for real programs. Knuth does some work in this area and [13] provides some data on the occurrence of common subexpressions, but there are no good figures on how often strength reduction is applicable, for example.

The statistics in these various studies do point out some valuable facts, though. One is that statements are usually simple. Thus, a register allocation algorithm which was concerned with minimizing the height of the trees, would be nice in theory but would not produce a big savings in practice. Effort would be better invested in handling common subexpressions. Unfortunately, even with simplifying assumptions, the problem of register

allocation in the presence of common subexpressions is NP-complete. Thus, it is impossible to devise an algorithm which is both efficient and optimal.

Since there is insufficient information to be able to determine the frequency of applicability for a given optimization in real programs, it is not meaningful to make generalization that "this compiler produces code which is within 10% of the size (speed) for assembly coded programs". It may be possible to make such a claim for a specific set of test cases, however.

Even for a fixed set of test it may be difficult to derive a reliable figure. It is known that different programmers write programs which differ in efficiency. Suppose two programmers implemented the same algorithm in assembly language and that one program was 20% smaller than the other. Would this imply that one assembler was not within 10% of the other?

An additional problem in making an accurate assessment of the compiler's generated code is that programmers tend to deliberately avoid language features which are implemented inefficiently by the compiler which they happen to be using. This may cause a skewing of the data, particularly if an attempt is made to compare two compilers. One would expect that the compiler with which the programs were originally developed would have an advantage in this regard. The ideal test cases for such a comparison would be real programs developed in a vacuum, with the programmers knowing only the semantics of the language and not how efficiently a particular compiler implemented language features. This is impractical, however, if the system is being developed for a real application and not as an exercise in studying use of the language. If the system is to be usable, it must meet its efficiency constraints when it is compiled through an actual compiler. Thus, a particular implementation will influence the most effective way to achieve a given result. In addition, programmers often experiment to find out how a feature works if the specification is unclear.

Another factor to consider is the allocation of data. The J73 language specification does not specify whether local data is to be allocated statically or dynamically in the case of procedures which are neither recursive nor reentrant. Allocating the data on a runtime stack can save space, but costs time. In addition, more code is required to access dynamically allocated data in certain situations.

Judgements of effectiveness of the optimizations are somewhat subjective, therefore. Even if there was sufficient data available to form a judgement based on past programs, there is no guarantee that it would hold in the future. One example can be found in the statistics in [10]. Of all the procedures surveyed, none had more than six parameters. However, there are several routines in the DSM software that have between 15 and 20 parameters.

Thoroughness of Optimization

It should be noted that if two compilers are said to perform a given optimization it does not imply that the two perform the optimization to an equal degree. There are various scopes in which optimizations may be applied:

local, regional, global, inter-procedural and inter-modular. Local optimization is performed within a statement; regional optimization applies to a group of statements; global optimization refers to the optimization of a procedure as a whole and inter-procedural refers to optimization which takes into account the effects of other procedures, in addition to the one for which code is being generated. We introduce the term inter-modular optimization to distinguish between optimization which use information obtained from a single source module (compilation unit in Ada parlance) and those which make use of information from a number of different modules.

It would be possible, in principle, to perform inter-modular optimizations in a JOVIAL compiler, but to do so would require much more environmental information than is currently maintained. In addition, configuration management would become more complicated because there would be a requirement to recompile other modules as the result of internal changes to a given module. At present, only modification of compools or def procedures or data require other modules to be recompiled. Thus, we feel that the cost of inter-modular optimizations would be prohibitive.

A certain degree of inter-procedural analysis is desirable, however. In particular, it is useful to know what data may be affected by a call to a given procedure. This information can then be found to avoid having to make worst case assumptions whenever a procedure is called.

Even if the scope over which an optimization is applied is the same, there may be differences in the number of cases which are actually handled. It is possible to perform optimizations for special cases and to ignore the more general case. For example, it would be possible to perform redundant computation elimination only for subscripts. Thus, it is not sufficient just to say that a compiler performs an optimization; additional information is required before any intelligent comparisons can be made.

Effects of the J73 Language on Optimization

J73 is a relatively large language, so it presents more difficulties to an optimizer than would simpler languages such as FORTRAN or Pascal. It is not as large, however, as Ada, so J73 optimization is simpler than Ada optimization.

J73 has more data types (notably pointers and fixed-point) and operators than does FORTRAN. In addition, JOVIAL compilers must deal with nested procedures, recursiveness and reentrancy. The fact that there is no input/output in J73 (except for !TRACE) simplifies the compiler's job.

Pascal is a far simpler language than is J73. It does not provide fixed-point arithmetic, presets, and inline or external procedures (although some implementations do provide external procedures). The most significant features of Pascal not found in J73 are input/output and range checking. Variant records are not present in J73, but may be simulated with specified and LIKE tables. Because Pascal pointers may point only to data in the heap, a Pascal compiler does not have to worry about local data being altered

through the use of a pointer.

Ada, on the other hand, has a number of features which affect an optimizer, but which are not found in J73. Tasking, packages and exceptions all require the optimizer to do additional work. Generics could be ignored by a naive optimizer, but they do present additional opportunities for optimization. The only features of J73 which require additional work by an optimizer are label and proc parameters. Abort is analogous to the exception mechanism in Ada, but all aborts are explicit.

For the most part, optimizations which are applied to algebraic languages apply to J73, also. They will not be discussed here. The following paragraphs deal only with differences between optimizing JOVIAL as opposed to other languages.

JOVIAL has a number of features which affect the optimization of object code. Some, like inline procedures and defines, provide additional opportunities for optimization. Others, like procedure parameters and overlays, make optimization more difficult.

The existence of inline procedures and defines means that the compiler is likely to see code which may be more susceptible to optimization than that which is written directly by a programmer. These language features make it easier to parameterize code so that a single copy of a source program may be used to generate code for various versions of a system, or which is to be run on various target machines. This parameterization often results in opportunities for the compiler to perform constant arithmetic or dead code elimination. These factors must be taken into account in making a decision as to which optimizations are cost-effective.

There are a variety of kinds of aliasing which are possible in J73. These include:

1. Overlay statements.
2. Aliasing by means of pointers.
3. Aliasing by means of formal parameters.
4. Overlaying by means of specified tables.

The fact that procedures and functions may be passed as parameters means that it may be impossible at compile time to determine precisely which procedures may be invoked as the result of a given call.

The existence in the language of a means for users to specify data allocation means that the compiler cannot, in general, make convenient assumptions about the allocation of data (for instance, that an item is allocated to a storage unit for which the machine provides easy access). These can produce a large number of special cases, if high-quality code is to be generated.

They also provide an opportunity for optimizations which would not arise in other languages, such as FORTRAN or Pascal, in which the storage mapping is more machine-oriented. One such optimization, discussed in more detail below,

is finding storage units common.

The awkward syntax used for referencing data in type tables provides an opportunity to do a special kind of folding.

The !BASE and !ISBASE directives allow users to specify that certain registers are to be used for base registers. The register allocation algorithm must allow for this outside interference.

The compiler must honor other directives when it is attempting optimizations. !LEFTRIGHT affects any optimizations which would attempt to change evaluation order for expressions. !ORDER affects variable overlaying.

There are several directives which are designed to facilitate optimization. !REDUCIBLE allows the compiler to eliminate redundant function calls. !INTERFERENCE tells the compiler that it must assume that setting one variable may change another. In the absence of this directive the compiler is free to assume that different names refer to distinct storage locations, except where there are explicit overlay or specified table declarations.

J73 requires that boolean expressions be evaluated in short circuit form whenever possible. Thus, the compiler has no choice as to whether to implement these expressions using tests and branches or logical operations, if there may be any side-effects.

In addition to these factors which arise directly from the language definition, the intended use of the language has an effect on its compilers. Because JOVIAL is intended for embedded systems, there are more stringent requirements on the size and speed of generated code than there would be in a data processing environment, for instance.

Constraints

There are a number of constraints which must be observed in considering a standard optimization strategy including:

1. Adherence to the language specification.
2. Compatibility with J73 front-end.
3. Compatibility with present code generators.
4. Compatibility with runtime library and old generated code.
5. Compatibility with system calling conventions.
6. Compatibility with other optimizer development.
7. Expandability to accomodate architecture extensions.

One of the most important constraints on an optimizer is that it may not produce results which deviate from the language standard.

It would, of course, be possible to design a compiler from scratch in order to facilitate code optimization. This would, however, not be cost-effective when compared to adding additional capability to the present compiler. That being the case, compatibility issues become important.

Compatibility provides a number of advantages. First, it is possible to use existing compiler phases. These have already been developed and tested, so the effort to add an optimizer is considerably less than it would be to build an entire compiler. Second, work which is done for one target (e.g., an added optimization) can benefit other targets, provided that a common optimizer is used. Similarly, optimizer bugs which are fixed for a given target will be fixed for all targets if there is one common source for optimizer modules. Having separate source files for optimizers for different targets would necessitate a mechanism for distributing changes to all appropriate optimizers.

There are some disadvantages to maintaining compatibility, also. A common optimizer will be more complex than a single-target one if it is to achieve comparable results. Second, scheduling conflicts can arise with respect to changes. One project or target may require changes just before another project or target is due for a compiler release. A change to the optimizer at an inopportune time would require testing to be rerun to insure that no regression has occurred. Third, an optimizer tailored to a given machine may be able to employ strategies which would be inappropriate for another target. A common optimizer would be confined to the same basic algorithm, although portions may be table-driven or target-dependent.

Maintaining compatibility with other code generators allows the benefits of optimization to be obtained for those targets at a reduced cost. This is a significant constraint, however, since the code generator-optimizer interface is complex and is not ideally suited to a new design. The alternative to maintaining compatibility, however, is a major modification to approximately ten code generators.

It is important to maintain execution time compatibility, at least at the module level with code generated by the present version of the compiler. This minimizes the risks of using newly compiled modules. This is particularly important for a self-compiled compiler, but also allows a fall-back position for users. They can add newly-compiled modules a few at a time until confidence is gained in the new optimizer.

Failure to observe the system calling conventions can cause problems if other tools are to be used with JOVIAL programs. For instance, misleading dumps would be produced on an IBM system if the compiler deviated too far from the standard calling sequence. It would also make it impossible to call routines written in other languages, unless standard calling sequences could also be generated. The present IBM compiler generates calls which use non-standard calls for JOVIAL-JOVIAL calls, but allows the user to specify standard calling sequences by means of the !LINKAGE directive.

The new optimizer should allow for future target architectural changes. (One such example is IBM's forthcoming extended address architecture.) This appears to be primarily an implementation issue, rather than a design consideration. Note that this implies avoiding the addition of future impediments rather than removing ones presently in the compiler.

Conflicting Standards

Since J73 is the standard Air Force programming language (at least until the development of the first successful Ada compiler), J73 compilers exist for a variety of host and target machines. It is desirable that these compilers exhibit similar behavior to minimize the learning required for a programmer to move from one compiler to another. It is also desirable that the interface to the J73 compiler be like those for other compilers on the same host, so that programmers familiar with the host system may more easily adapt to using J73.

Unfortunately, this produces conflicts, because the conventions for different host systems tend to be dissimilar. The ways compilers are invoked under MVS and TOPS-20 bear no resemblance to each other. Thus, a choice must be made as to which convention to follow: that of the host machine or that of existing JOVIAL compilers. In the case of the SEA J73 compiler, the choice has been to follow the host system conventions. Thus, the JCL for invoking the J73 compiler on an IBM system is more similar to that used for a FORTRAN compilation on that system, than for JOVIAL compilations on a DEC-20.

Problems also arise in the interface between user programs and the operating system. A classic example is character allocation on the VAX. The natural way for characters to be allocated within a word on the VAX is right-to-left, but the J73 specification requires that characters be allocated left-to-right. Thus, when designing a JOVIAL compiler for the VAX, one is faced with a choice of creating a compiler which may produce different results than would be obtained on another machine, or of failing to take full advantage of the host hardware and creating incompatibilities with system conventions and with programs written in other languages.

Methods for Determining Potential Optimizations

There are a number of approaches which can be used to determine those areas in which generated code should be improved, including examining the current generated code for inefficiencies, comparison with generated code for other compilers for the same target, and inspection of the target instruction set for instructions which are unused or under-used in generated code. All of these approaches have been applied to the SEA compiler. In addition, standard optimization techniques not implemented in the compiler have been examined for applicability.

Examination of current code

The compiler supports at least two classes of programs: the compiler itself and application programs. Because these are likely to have different characteristics, samples of the generated code for both types were examined.

The compiler was originally coded in J73/I and then translated automatically to J73. This has had several effects on the code found in the compiler. First, the compiler is, in effect, coded in the J73/I subset of J73. Some enhancements have been added since the translation took place, but

their magnitude is small relative to the size of the entire compiler. Second, the database is a J73/I database, so that even new code will take on a J73/I flavor. For example, defines were used rather than types to declare data, because J73/I had no user-defined types. Thus, new code which declares items which are of the same (logical) type as data already in the compiler's database would use define calls rather than types to declare the new data.

On the other hand, the Satellite Control Facility (SCF) code is effectively Ada translated to J73. Thus, types are used more often, and operations on aggregates (real or simulated) are more common.

Another major difference is that applications software is reentrant. The only routines in the compiler which are reentrant are those from the runtime library. Several compiler routines are recursive, although the vast majority are not.

Comparison with other compilers

Some test cases were run using the present J73 compiler, the J73/I compiler, and compilers for various other languages, such as FORTRAN and PL/I. These tests were run using various levels of optimization.

Several interesting observations can be made. The first is that reduction of space requirements and execution time are conflicting. Interestingly enough, the generated code produced by J73 was more compact than that produced by either version of FORTRAN H, FORTRAN G or the PL/I optimizing compiler. The code produced by FORTRAN H was more compact at optimization level 1 than at optimization level 2, but the amount of code in the inner loop was 4 times greater for level 1.

Second, it is possible to make great improvements in the generated code for tight loops. J73 had 50 (hex) bytes of code in the inner loop, but FORTRAN H with full optimization had only 10. For comparison FORTRAN H without optimization generated 60 bytes for the inner loop and PL/I with optimization off generated 74.

Examination of the target instruction set

Comparing the set of instructions generated by a compiler against those in the target machine instruction set can be a means of improving the use of the target's instructions, particularly if the original code generator was restricted to a least common denominator approach for compatibility reasons, or to minimize development time. Clearly, not all machine instructions are suitable for use by JOVIAL compilers. COBOL-oriented edit instructions, for example, are unlikely to enhance JOVIAL code. However, if binary arithmetic or logical instructions were unused, it would be worth investigating to determine if they could be used to produce better code.

Consideration of known optimization techniques

There are a number of catalogs [14,15,16] of optimization techniques which can provide ideas for consideration. Since JOVIAL contains arithmetic operations which are conventional, techniques developed for other languages, such as FORTRAN, can be applied.

Optimizer Data Base for the MX Optimizer

Certain information is needed in order for the optimizer to be able to determine whether optimizations are applicable and profitable. In particular, set/use information is needed for variables and dominator information for the program graph is required. The way this database is derived is a major issue in the design of an optimizing compiler.

There are a number of algorithms for data flow analysis. Some are described in widely-available literature [17,18,19,20,21]. Others, such as the algorithm used in the current J73 compiler [4], are not as widely known.

Perhaps the best-known of these methods is interval analysis [17]. A program is divided up into a series of intervals, each of which contains, at most, one strongly connected region. Interval analysis is then applied on a graph in which each interval is condensed to a single node. This process repeats until the whole program has shrunk to one node or the graph is found to be irreducible. The principle disadvantage of this method is that it is applicable only to reducible programs. Because J73 allows goto statements, it is possible to write irreducible programs in this language.

Other methods also are restricted to reducible programs [20]. The method chosen for the MX J73 optimizer is P-graphing [22]. This method is completely general and can be applied to arbitrary program graphs. In addition SEA staff members are familiar with this method, having implemented two optimizers which use it. Further, a version of the algorithm, written in SYMPL is available. Thus, the cost of implementing the algorithms is one of translation, rather than development.

The P-graph of a variable contains information about the set and use information for that variable. All occurrences of a variable which represent the same value are assigned to the same "generation class". Each use is associated with a given operation. For those uses which can result from two or more actual generations (due to the merging of control flow), a pseudo-generation is created at the merge point.

The P-graph for each variable is constructed from the program graph. Initially, each node which contains a generation of the variable is tagged with its own node number. The tag is propagated from a node to each of its successors. If the successor is tagged with its own number, it is left unchanged. If the successor is tagged with a value which is neither its own number, nor that being propagated, there are several generations which can reach that node, so a pseudo-generation is created at that node. The algorithm terminates when all generations and pseudo-generations have been propagated. At this point each node is labeled with the number of the node which contains the generation which can reach that node.

The actual algorithm contains optimizations for special cases and must handle generations and uses which occur within basic blocks, but the general idea is the same. A very high level language description for the P-graph algorithm is contained in [22].

In addition to the generations and uses of variables, dominator information must also be known in order to perform such optimizations as code motion. A node n back-dominates node n' if $n \langle \rangle n'$ and every path from the entry node to n' passes through n . An analogous definition exists for forward dominators. This dominator information is necessary to insure that if a computation is moved from one block to another, that it will be performed under the same conditions as it would have prior to the code motion.

The algorithm for computing dominators is given in [23,24]. It computes dominators by doing a depth-first search of the program graph, constructing a spanning tree and partitioning the remaining nodes into forward, cycle and cross arcs.

Implementation Approach

A global optimizer is being implemented for the MX project. The compiler for MX is based on the same root as the J73 compiler used by SCF. Thus, the most reasonable approach for the SCF optimizer is to use the optimizations which become available as a result of the MX project and to add those optimizations which are particularly applicable to the 370 architecture and SCF needs. In addition, as optimizations are being designed for MX, issues of rehosting and retargeting to the 370 are to be addressed.

The rehosting issues are relatively easy to handle. Because the optimizer will be coded in J73, it will be rehostable if host-dependent coding is avoided. Fortunately, this does not require great effort, only foresight.

Retargeting issues have been mentioned in the preceding discussion on optimizations. The most important ones are base register handling and register allocation in general.

The following optimizations will be implemented in the MX compiler:

- a. Constant arithmetic
- b. Constant folding
- c. Common subexpression elimination
- d. Reordering of arithmetic expressions (including unary - optimizations)
- e. Elimination of jumps to jumps
- f. Unreachable code deletion
- g. Invariant code motion
- h. Strength reduction and test replacement

Development for the MX optimizer is already under way on a DEC-20. At the present time the dominators algorithm and the optimizer data base have been translated to J73 from SYMPL and have been compiled. Development will continue on the DEC-20. The most reasonable approach to take for the SCF optimizer is to do the development on the DEC-20, including the necessary

modifications to the IBM code generator. After the optimizer has been checked out by examining code produced, it can be rehosted to a 370, to facilitate the running of test cases.

Summary

Although optimization is not necessary for all applications, it is still important for those which have critical space and/or speed requirements. Costs associated with optimization include increased compiler development time and cost, requirements for additional compiler testing, lowered compilation speed and increased difficulty of creating a source-level debugging interface.

The fact that there is a standard Air Force language makes it possible to reuse large portions of the compiler. This allows compilers to be produced for a far lower cost than would be possible if it were necessary to develop compilers for different languages.

Acknowledgement

Portions of this paper were produced under Contract # F04690-81-C-0006 of the Air Force Satellite Control Facility.

References

1. Backus, J., "The History of FORTRAN I, II, and III", SIGPLAN 13:8, August 1978, pp. 165-180.
2. Allen, F.E., "Bibliography on Program Optimization", RC-5767, IBM Thomas J. Watson Laboratories, 1975.
3. Hansche, B., Hudson, S., and Huey, B., "Selected Bibliography of Compiler Optimization Topics", SIGPLAN 17:8, August 1982, pp. 74-83.
4. J73 JOVIAL Workbook for AFWAL/AAA MIL-STD-1589B Compiler, October 1981.
5. Hennessy, J., "Symbolic Debugging of Optimized Code", TOPLAS 4:3, July 1982, pp. 323-344.
6. Intermetrics/MCA, Ada Integrated Environment: Interim Technical Report, 1981.
7. CSC/SEA, Ada Integrated Environment: Computer Program Development Specification, 1981.
8. Knuth, D.E., "An Empirical Study of FORTRAN Programs", Software - Practice and Experience 1:2, 1971, pp. 105-134.
9. Wulf, W., private communication, 1981.
10. Softech, Survey and Measurement of Properties of High-Level Language Programs, 1980.
11. Bloom, B., Feldman, C., Clark M., and Coe, R., Criteria for Evaluating the Performance of Compilers, Rome Air Development Center Technical Report RADC-TR-74-259, October 1974, 348 p.
12. Devine, T., Dunbar, T., Littlejohn, M., and White, K., JOVIAL/ADA Microprocessor Study, Rome Air Development Center Technical Report, RADC-TR-82-61, April 1982, 190 p.
13. Lunde, A., "Empirical Evaluation of Some Features of Instruction Set Processor Architecture", CACM 20:3, March 1977, pp. 143-152.
14. Allen, F.E., "Program Optimization", Annual Review of Automatic Programming 5, 1969, pp. 239-307.
15. Loveman, D., "Program Improvement by Source-to-Source Transformation", 2nd FOPL, 1975, pp. 140-152.
16. Standish, T., Harriman, D., Kibler, D. and Neighbors, J., The Irvine Program Transformation Catalogue, University of California at Irvine, 1976, 82p.
17. Schaefer, M., A Mathematical Theory of Global Program Optimization.

Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

18. Wulf, Johnson, Weinstock, Hobbs and Geshke, The Design of an Optimizing Compiler, American Elsevier, New York, 1975.
19. Killdall, G., "A Unified Approach to Program Optimization", ROPL, 1973, pp. 194-206.
20. Graham, S. and Wegman, M., "A Fast and Usually Linear Algorithm for Global Flow Analysis", JACM 23:1, 1976, pp. 172-202.
21. Rosen, B., "High-Level Data Flow Analysis", CACM 20:10, October 1977, pp. 712-724.
22. Loveman, D. and Faneuf, R., "Program Optimization - Theory and Practice", SIGPLAN 10:3, 1975, pp. 97-102.
23. Tarjan, R., "Finding Dominators in Directed Graphs", SIAM Journal of Computing 3:1, 1974, pp. 62-89.
24. Tarjan, R., Edge-Disjoint Spanning Trees, Dominators, and Depth-First Search, Stanford University Computer Science Department Technical Report STAN-CS-74-455, Sept. 1974, 40p.

J73AVS: A JOVIAL J73 Automated Verification System

Carolyn Gannon

General Research Corporation
P.O. Box 6770
Santa Barbara, CA 93111
805-964-7724

ABSTRACT

→ The development of J73AVS reflects the commitment of the Air Force to facilitate JOVIAL J73 standardization. This paper describes software verification as it is automated by the J73AVS tool. The concept of software verification is discussed, as well as the capabilities and operation of J73AVS. J73AVS provides much of its payoff by detecting certain software errors and measuring the thoroughness of testing far more accurately and efficiently than could be achieved manually. While J73AVS operates as a standalone program on several host computers, it augments the JOVIAL J73 support environment when used with other Air Force-sponsored tools such as the code auditor [1], debugger [2], and Program Support Library [3].

INTRODUCTION

What is "software verification"? According to the IEEE committee on standardizing software terminology, it is

"the iterative evaluation of evolving software to ensure compliance with requirements"

Thus, verification differs from validation or certification in that it is an activity that is performed continuously throughout a software development cycle. It incorporates a variety of automated and manual techniques to determine consistency between the requirements, design, coding, testing, and documenting stages of software projects.

Our focus in designing and building Automated Verification Systems (AVS) for JOVIAL [4], FORTRAN [5], and COBOL [6] has been on static and dynamic code analysis. That is, each AVS reads source code as input for static analysis and uses the program's regular input data during dynamic analysis. Therefore, requirements and design are not verified directly by the AVS. However, because the AVS analyzes the actual code, the tool reports true program characteristics. This approach interferes very little with normal program development, since the AVS does not require additional information other than the source code and initial set of test data.

One very important, but often neglected, area of software verification is that of ensuring that the program documentation reflects the real code. The most time-consuming aspect of conforming to Mil-Std-483 or other documentation standards is generating program symbol, structure, and interface information. Because it maintains a database of intra- and inter-module characteristics, the AVS can generate some of this information automatically. As code is modified due to error correction or enhancements, the reports can be easily regenerated to reflect the changed code. In contrast, manually generated documentation is rarely up to date.

J73AVS CAPABILITIES

J73AVS should play a role in JOVIAL J73 software development as soon as some of the modules are compilable. The source code is generated based upon a design, which in turn is based on a set of requirements. It is recommended that the expected program output (acceptance criteria) and at least an initial set of input test data be generated concurrently with the program's design. The requirements, design, and acceptance criteria play an indirect role in J73AVS's analysis of the software.

The types of J73AVS analysis capabilities are:

- Static and data-flow analysis (symbol usage anomalies and dangerous coding)
- Reporting of program structure and characteristics
- Measuring execution coverage of statements, branches, and program units
- Execution tracing of variables, branches, and program units
- Execution timing
- Structural (branch) retesting assistance
- Test history reporting

Figure 1 shows how the requirements, design specification, acceptance criteria, and test data interact with J73AVS-supported testing. The acceptance criteria are used to judge the proper performance of the program. J73AVS provides detailed source analysis reporting which aids the analyst in determining that certain acceptance criteria are being met. The bold path marked number one in Figure 1 indicates the cycle of static code checking.

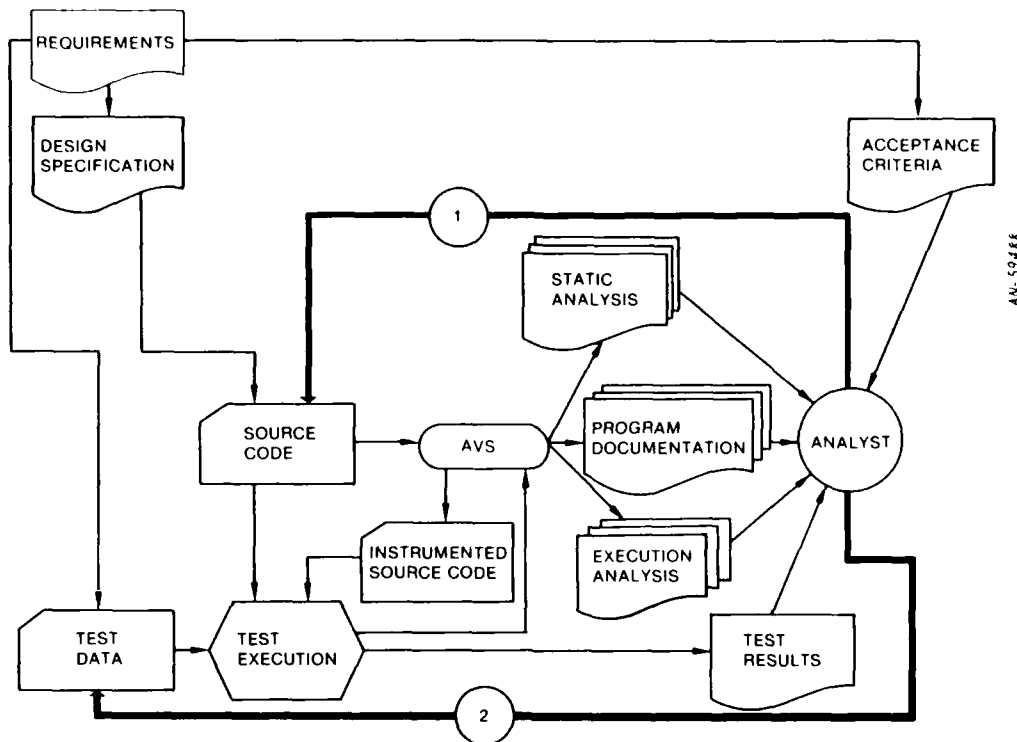


Figure 1. Using an AVS

Once the static errors are removed, the program can be analyzed dynamically by driving it with the initial set of test data. Dynamic analysis is indicated by bold path number two in Figure 1. J73AVS outputs execution coverage, timing, and tracing information, along with the normal program output, which aids the analyst in determining acceptable performance. Unexercised statements and branches are indicated by J73AVS so that additional test data can be generated to ensure that all parts of the code are tested. Dynamic analysis, therefore, is usually an iterative activity that continues until the desired level of exercise is achieved. J73AVS maintains the coverage levels for each test in its database.

As shown in Figure 1, compilable source code generally is first analyzed by J73AVS to detect semantic errors that are outside the scope of the compiler's static analysis capabilities. As each module is analyzed by J73AVS, a database is built that contains single and multi-module detailed characteristics. This database is used and augmented each time additional analysis (static or dynamic) is requested by the user. Thus, J73AVS is a partner in the development, testing, and documentation phases.

It should be pointed out that, because of the database feature, J73AVS supports top-down code development in the following way. High-level modules can be coded early with stubs (module skeletons) for lower-level modules. Both fully coded modules and stubs can be input to J73AVS for analysis and documentation. J73AVS includes the stubs in its database. Module interaction and interfaces (COMPOOL usage and parameter passing) will be analyzed and reported to the extent that they occur in the code. Then, as lower-level stubs are replaced with full source code, J73AVS replaces the modules on the database.

A typical sequence of J73AVS-supported verification of fully coded source modules is:

1. JOVIAL J73 source text, perhaps with assertions (Boolean expressions, recognized by J73AVS, that specify expected behavior), is read by J73AVS as one or more compilable modules.
2. J73AVS produces program analysis reports showing control structure, symbol usage, calling hierarchy, etc., as well as a static analysis report showing errors and dangerous programming practices.
3. Using the reports as a guide, the source modules are changed or new modules are added to the program.
4. J73AVS reports the interaction of the new or changed modules with the rest of the program. This information, in turn, may show the need to modify other modules.
5. For debugging, the program is instrumented by J73AVS and executed with an initial test case supplied by the user.
6. Assertion messages, variable, branch, and module tracing, and execution timing reports can be used for debugging.
7. Using the J73AVS reports, the user chooses to create more test data or instrument other modules.
8. For testing, the same cycle of instrumentation and execution is repeated, but for a different goal: rather than detecting and locating errors, testing aims to demonstrate that the entire program has been exercised to some degree. The J73AVS execution analysis reports show the thoroughness of execution coverage.
9. The user evaluates execution coverage reports, the program's own execution results, and the program specification to determine if testing is complete.

10. J73AVS provides branch sequence information to retest targets chosen by the user. A test history of execution coverage assists the user in choosing targets for retesting.

As just noted, J73AVS can be used to assist with several phases of software development. These phases can be grouped as:

- Program development and maintenance
- Debugging
- Testing
- Retesting assistance

Program Development and Maintenance

Executable assertions provide a means for a programmer to specify expected behavior. Assertions can be used for reporting execution-time exceptions, stress testing, and manual or automated test data generation. When assertions are left as comments in the source code they can be used as inline documentation of the program's specifications. An example of an executable assertions is:

```
". ASSERT (STACK'POINTER >= 0)"
```

To assist with reliable system development and maintenance, J73AVS provides substantial program analysis reporting on structural hierarchy, symbol usage, invocations, certain J73 constructs, and system characteristics. The user has control over obtaining high- or low-level information through the tool's command language.

Debugging

Normal compilation using JOVIAL J73 compilers can detect many syntax and semantic errors. Other errors, such as uninitialized variables, possible infinite loops, unreachable code, certain improper constructs, and dangerous coding practices (like transferring into CASE or IF statements) will be reported by J73AVS. The user can specify the degree of analysis to the error, warning, or message level.

Debugging is supported by assertion exceptions, variable and module execution tracing, and execution timing reports. When the program's execution behavior deviates from the acceptable logical behavior as specified by the assertions, it is immediately reported in the program's output. The user-embedded assertions have no effect on program control flow until they are violated; at that time the violation is reported with the source statement number of the assertions.

Testing

The primary purpose of program coverage analysis is to provide a measure of the level of testing. One measuring technique uses the program's control structure as a guide. Structure-based testing means that the program's control structures are analyzed for execution behavior; that is, whether the structures are exercised and in what order. Structure-based testing can uncover errors due to untested branches or improper sequences of branches. J73AVS provides program unit or branch tracing and analyzes execution coverage of program units, branches, and statements. Further, J73AVS assembles the timing information from program unit tracing and user-directed timing probes into an execution timing report.

Retesting Assistance

Software is retested when analysis shows that prior testing is inadequate (insufficient branch coverage, not all functions demonstrated, etc.) or when program changes have taken place. The proper approach to take in retesting is highly dependent upon the characteristics of the program being tested as well as the measures being used to evaluate testing completeness.

To determine the sequence of branches which lead to an untested branch or statement, the user can request that the "reaching set" be computed between two specified statements (or from a procedure's entry). After the flow of control is identified by J73AVS, the user can back-track through the program to the actual test data. New test data can be created by using J73AVS module interaction, invocation, and execution coverage reports. Unfortunately, automatic test data generators which use symbolic execution are not yet general enough, easy to use, or reliable. Therefore, J73AVS has no test data generation capability at this time.

The testing history maintained by J73AVS is useful in attaining testing coverage goals and for determining targets for retesting. Procedure invocation and coverage information is saved in a concise way for each test case. The results of subsequent execution runs can be added, providing a cumulative report of all tests.

J73AVS OPERATION

J73AVS operates in either batch or interactive mode on a host computer. If the JOVIAL J73 code being analyzed is destined for execution on a target computer, the J73AVS dynamic analysis operation is modified slightly, as shown in Figure 2.

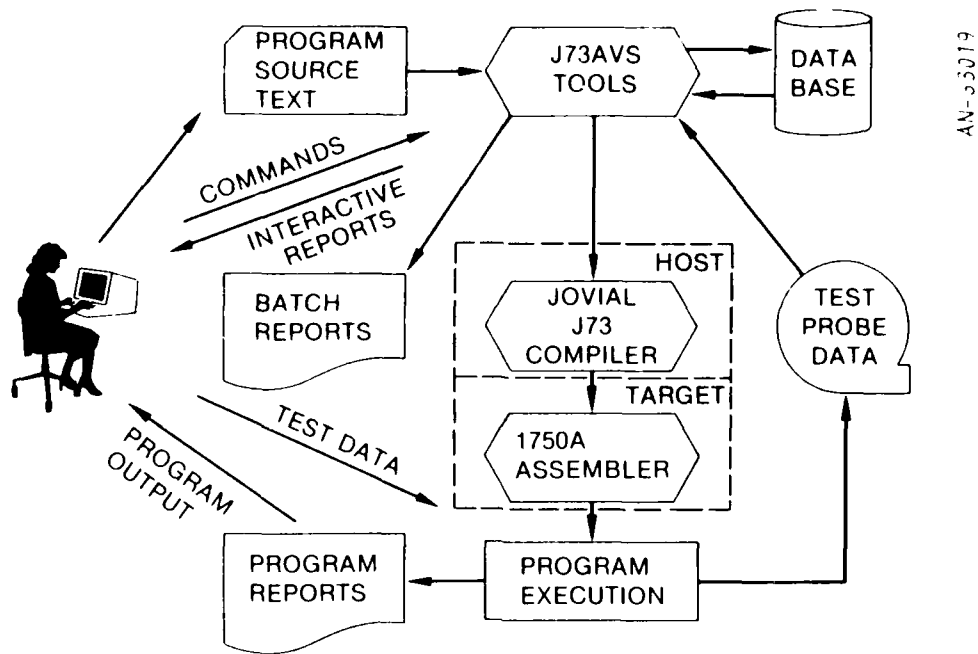


Figure 2. Using J73AVS in a Host-Target Environment

The user directs J73AVS analysis through a simple command language. The basic commands are verbs that select the type of analysis, followed by command parameters that specify the scope of the analysis or level of error reporting. As much as a whole program or as little as a single symbol can be analyzed.

J73AVS displays or prints reports during static analysis. For dynamic analysis, the instrumented source (augmented by expanded assertions and by probes for execution coverage, tracing, or timing) is passed to the JOVIAL J73 compiler. In a host execution environment, input data is read by the program and normal program output is accompanied by an execution data collection file required for the J73AVS post-execution analysis reports. J73AVS uses that file, along with its database, to provide readable, user-selected reports that describe execution behavior.

In a host-target environment, the target computer must have a sequential output device such as a disk or tape to transfer the data collected during execution back to the host. In the absence of any sequential output device, the J73AVS data collection routines can be modified to output test coverage information on the target in an abbreviated manner.

J73AVS was developed for operation on IBM 370 and DEC 20 computers. It is currently being rehosted to the VAX 11/780. J73AVS is written in JOVIAL J73, except for a few small input/output routines written in FORTRAN. The VAX version of J73AVS is written in a structured dialect of FORTRAN.

ACKNOWLEDGEMENT

The design, development, and current rehosting of J73AVS is being sponsored by Rome Air Development Center, Griffiss Air Force Base, New York, under Contract F30602-79-C-0265. The project officer is Frank LaMonica, RADC/COEE. The VAX rehosting effort is being funded by the Embedded Computer Standardization Program Office (ECSP0), ASD-AFALD/AXS, Air Force Wright Aeronautical Laboratory.

REFERENCES

1. L. Brownell and R. J. Gilinsky, JOVIAL (J73) Code Auditor User's Manual, Proprietary Software Systems, Inc., 16 March 1982.
2. Maj. D. Burton, S. May, and T. Fujawa, "A JOVIAL Interactive Debugger," NAECON 82, pp. 1121-1129.
3. JOVIAL J73 Programming Support Library User's Manual, SofTech, Inc., Jan. 1982.
4. C. Gannon and R. F. Else, JOVIAL J73 Automated Verification System User's Manual, General Research Corporation CR-4-947, November 1981.
5. RXVP80™ User's Manual, (Preliminary Draft) General Research Corporation RM-2419.
6. P. Roberson, R. Melton, and C. Andrews, COBOL Automated Verification System User's Manual, General Research Corporation CR-4-970, May 1982.

Biographical Sketch for Carolyn Gannon

Carolyn Gannon is the director of the Software Technology Department at General Research Corporation in Santa Barbara, California. She has been active in the design and development of automated verification systems for JOVIAL (J3 and J73) since 1974.

Mrs. Gannon earned a Master's degree in Computer Science from the University of California and has been involved in software testing and formal language research and, more recently, in command and control systems for public safety applications.

AD-P003 522

JOVIAL LANGUAGE CONTROL PROCEDURES WITH A VIEW TOWARD Ada*

Patricia A. Knoop
Bobby R. Evans

ASD/ADOL
Language Control Branch
Computer Operations Division
ASD Computer Center
Wright-Patterson Air Force Base, Ohio 45433
513-255-4472

ABSTRACT

JOVIAL is the interim standard language for Air Force avionics embedded computers until Ada becomes available. The JOVIAL Language Control Facility (LCF) has developed and fine-tuned the procedures of language control and defined them using a formal modeling technique. The resulting models promote tight administration of the control function by exposing the details of all tasks and forcing attention to their interrelationships. They also provide a basis for reconfiguring proven Air Force language control functions for Ada, and the LCF has identified some important considerations in accomplishing this. The Air Force's transition to Ada has a high probability of success because of their experience with JOVIAL, their systematic evolution and fine-tuning of language control procedures, and the extensibility of these procedures to encompass Ada.

A PRAGMATIC DEFINITION OF LANGUAGE CONTROL

High order language standardization is being successfully achieved in the Air Force. The Air Force currently requires JOVIAL (J73) to be used on all avionics embedded computers until the DoD-wide language, Ada, becomes available. To support this standardization effort, language control is essential. This paper is about the JOVIAL Language Control Facility (LCF)...what it does, how it operates, and the considerations it feels are most important as we extend language control to encompass Ada.

*Ada is a trademark of the United States Department of Defense (DoD)

This paper was published in the proceedings of the IEEE 1982 National Aerospace and Electronics Conference (NAECON), 18-20 May 1982, Institute of Electrical and Electronics Engineers, New York.

Copyright 1982 by the Institute of Electrical and Electronics Engineers, Inc.

After initial development and trial operation by the Rome Air Development Center (RADC), the LCF was moved to its present location in the ASD Computer Center's Computer Operations Division in March 1981. Our first year of running the LCF has provided us an opportunity to fine-tune the excellent procedures developed by RADC and determine the areas of user support in which to concentrate our resources. Through this experience, we have also developed a pragmatic definition of language control that not only describes our function but provides it a firm conceptual foundation:

Language control is the assurance of the integrity, stability, consistency, and usability of the language.

Assuring the integrity of the language means assuring its accuracy and completeness. While doing this, however, we must also allow changes to be made to the language to remove deficiencies and adapt it to new applications. This allowance for change is somewhat in conflict with the attending requirement for assuring language stability. Stability requires a resistance to change. We resolve this apparent conflict operationally through careful compromise and control of the language change criteria. Assuring the consistency of the language requires assuring that every implementation is in accordance with the standard language specification and, thus, that standardization itself is achieved. Finally, assuring usability of the language requires us to educate people about it and to promote its use by making data, documentation, and support tools readily available.

A. MAJOR ELEMENTS OF LANGUAGE CONTROL

Based on the above definition, the act of language control has some very specific requirements. To understand these requirements, it is necessary to consider both the elements or ingredients of the control process and the procedures for using them. We have identified four major elements necessary to make language control possible:

- (1) A well-defined and consistent policy for controlling changes to the language;
- (2) A mechanism for fielding, analyzing, and implementing language changes;
- (3) A mechanism for checking conformance of applications of the language to the official specification; and
- (4) A centralized, knowledgeable source of information about the language and associated compilers and tools.

The provision of a well-defined change policy and a mechanism for making changes allows us to assure language integrity and stability. The mechanism for conformance-checking supports the assurance of language consistency. Finally, the centralized knowledge source allows us to assure language usability. Thus, the elements of language control all relate to its definition. All that remains is to define the procedures for accomplishing control using these basic elements.

B. SYSTEMS ANALYSIS AND DESIGN TECHNIQUES (SADT*) MODELS

We have defined these language control procedures using an SADT modeling technique developed by SofTech, Inc. (Ref. 1). These definitions are documented in Reference 2, which we update annually to incorporate revisions and additions. The SADT models promote tight administration of language control because they force attention to details and identification of every important parameter of the control process. They also provide an organized means of reconfiguring the language control function to new languages or extending it to include new policies. This is facilitated by the models' provision for complete definition of all interrelationships among the various control activities.

The SADT modeling technique consists of drawing a series of diagrams to describe the control process in successive layers of increasing detail. In the sample diagram in Figure 1, boxes represent activities. Arrows entering a box on the left represent input data and those leaving on the right, output data. Arrows entering the top represent controlling functions or data. Those entering the bottom indicate the mechanism for implementing the activity. Thus, box 2 in Figure 1 shows that the Configuration Manager (in the LCF), the Language Control Agent (LCA), and the JOVIAL/Ada Users Group (JUG) all participate in maintaining the language standard under the guidance of the LCA and a configuration management plan. Language Issue Reports (LIRs) are the inputs and the revised language standard is the output of this activity. Diagram and node numbers are used to identify related diagrams that show additional details and to number the charts for easy reference. We took all the SADT charts in this paper from Reference 2.

JOVIAL LANGUAGE CONTROL PROCEDURES

In this Section, we present a detailed description of the procedures used to control the JOVIAL language. Figure 1 is an SADT chart for the total task of operating the LCF. First, we must establish and maintain policies that guide how we handle new requirements and respond to changing user needs. We must also maintain the language standard, currently

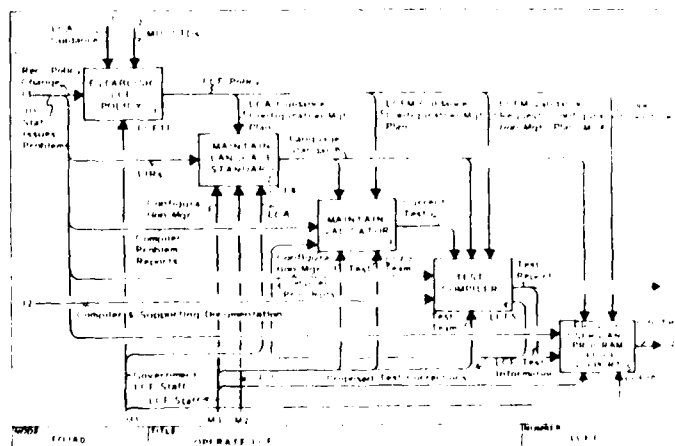


FIGURE 1: PROCEDURE TO OPERATE LCF

*SADT is a trademark of SofTech, Inc.

MIL-STD-1589B, by analyzing and incorporating changes necessary to enhance the utility of JOVIAL for Air Force programs. We must maintain the JOVIAL Compiler Validation System (JCVS) by designing and incorporating new tests when necessary and updating it to match new versions of the governing language standard. We must use the JCVS to perform validations of JOVIAL compilers as requested by the LCA. Finally, we must provide a variety of support services to assist users in learning and applying JOVIAL and related tools. In the following subsections, we expand on each of these tasks.

A. ESTABLISH LCF POLICY

The government LCF Manager establishes policy with guidance from the LCA and various military standards. This policy guides all operations of the LCF and, thus, control of the language. As existing policies are changed and new ones added, the LCF so informs the JOVIAL user community to keep it apprised of new services and activities.

All LCF policy and the policy-making process must be flexible to accommodate unique user requests, new language applications, and changing emphasis as we move toward Ada. One example of a recently formulated policy is that we now automatically inform holders of the JCVS about the existence of a new version, which they may order by dealing directly with the LCF. This assures users of access to the most recent version for use during compiler development activities.

An example of a policy extension we are in the process of formulating under the guidance of the LCA is to perform informal validations of industry-developed JOVIAL compilers. This is in addition to the formal validations we perform at the request of Air Force Program Offices. We believe this will promote JOVIAL by providing an additional measure of confidence on evolving compilers before they are applied to specific programs.

B. MAINTAIN LANGUAGE STANDARD

The military standard for JOVIAL describes the syntax, semantics, and constraints of the language. It is by assuring adherence to this standard at the compiler level that the Air Force enforces standardization requirements. While standardization inherently assumes language stability, a mechanism must exist for making controlled changes to the language when they are necessary to enhance or enable applications to Air Force programs. The LCF maintains the language specification by implementing this change mechanism and controlling the criteria for accepting changes.

Actually, the maintenance operation involves direct participation of the JUG, its associated Language Issues Committee, the LCA, and the Language Control Board (LCB). This is depicted in Figure 2, which shows the major steps involved in the operation. Any JOVIAL user may suggest a change by preparing a Language Issue Request (LIR) and submitting it to the

JUG or the LCF. This LIR describes the perceived problem and may also suggest a solution. To formally document it, the LCF or JUG prepares a corresponding Language Change Request (LCR) that describes precisely what changes are to be made to specific paragraphs or phrases of the language specification.

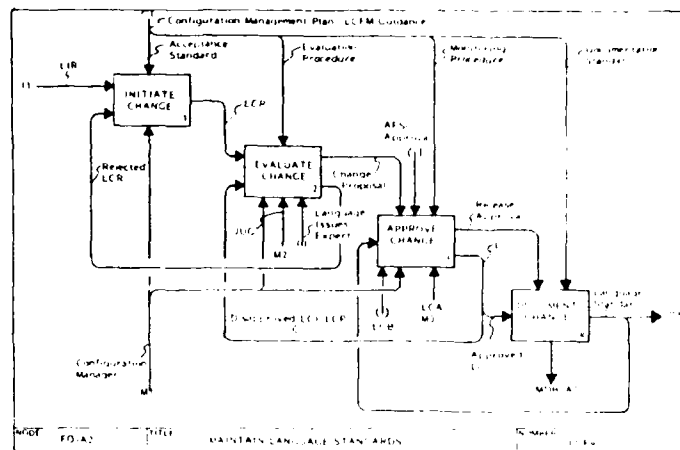


FIGURE 2: PROCEDURE TO MAINTAIN LANGUAGE STANDARD

Both the JUG Language Issues Committee and the LCF review and analyze each LCR. For those LCRs approved by the Committee, the LCF prepares a formal analysis according to a rigid set of criteria. These criteria include the following:

- (1) The proposed change must be upward compatible. This means that existing, sound JOVIAL code must continue to be compilable by compilers that implement the change;
- (2) The proposed change must correct a real deficiency (be necessary) and be complete (be sufficient);
- (3) The proposed change must not significantly increase the need to resort to assembly level coding; and
- (4) The proposed change must not have significant negative impact on the difficulty of compiler development.

It is by manipulating and tightening these criteria that the LCF can control change proliferation and nurture the stability of the language.

When both the JUG and LCF have completed their reviews and analyses, the LCR, suitably signed and approved, becomes a Language Change Proposal (LCP). A JUG Language Issues Committee spokesperson and the LCF language expert formally present this proposal to the LCB. The LCB votes on the proposal and, if it is passed, it becomes a Language Configuration Item (LCI) for entry by the LCF into the working version of a new specification.

Upon direction from AFSC through the LCA, the LCF produces a new version of the language specification or official changes thereto. Current

plans are to produce a Change Notice to MIL-STD-1589B in mid-1982 covering only benign changes such as typographical errors. In Mid-1983, the Air Force is considering a full update of the specification to MIL-STD-1589C.

C. MAINTAIN VALIDATOR

The JOVIAL Compiler Validation System (JCVS) (Ref. 3) is the official set of test programs used to check conformance of JOVIAL compilers to the language specification. The LCF maintains and enhances it under the guidance of a rigid configuration management plan. Figure 3 illustrates the steps in the JCVS maintenance process. We will describe related configuration management procedures in conjunction with the steps they affect in the following discussion.

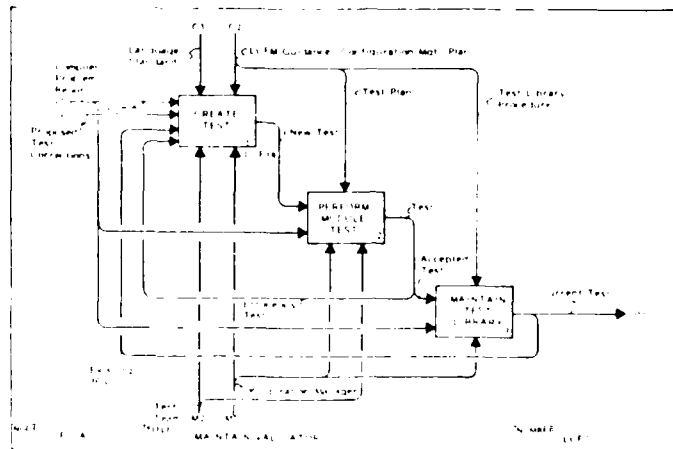


FIGURE 3 PROCEDURE TO MAINTAIN VALIDATOR

Any user of the JCVS may submit a problem report documenting a perceived inadequacy of the test-set. In practice, the LCF identifies most of the problems during either (a) the performance of a compiler validation or (b) a planned review of the JCVS to look for ways of improving it. The submitter documents the problem on a Validator Change Request (VCR) form.

The LCF verifies the validity of the problem and designs a proposed solution consisting of a new or revised program in the JCVS. The LCF implements and tests the solution by compiling the new program using one or more compilers in the LCF repository, which is purposely stocked to support testing JCVS changes. If the compiler used in the test appears to have failed the test, the LCF rechecks the test logic using code walk-throughs and similar procedures; and it reruns the test using an alternative compiler.

Upon successful completion of testing, the LCF prepares formal documentation of the proposed change to the JCVS by completing Part 1 of the VCR form. This form contains a description of the problem, proposed solution, and specific code used or affected and is appended by the output of test compilations. The LCF staff presents the VCR to a review team

appointed by the LCF Manager. This team analyzes the purpose of the change, symptoms of its necessity, and evidence of its soundness, more-or-less playing a devil's advocate role. The team indicates its decided disposition of the VCR by completing Part II, which includes evaluation comments and reasons for the enacted decision.

Upon direction by the LCF Manager, the LCF staff prepares a new version of the JCVS and a new Version Description Document. They archive all old versions for possible reuse in situations involving contention of earlier validation test results. The LCF notifies people currently holding the JCVS that a new version is available.

One important feature of the JCVS maintenance process is that the LCF very carefully reviews and screens any test revisions before implementing them. Test construction requires an in-depth knowledge of the language and careful attention to maintaining the purity of each test. Even with such care, we have learned that a good test set, like a compiler, should never be considered completely finished or guaranteed free of error. Therefore, we are continuously pursuing various avenues to reanalyze the JCVS, look for ways to improve it, and develop new compiler testing concepts.

One avenue is to carefully review the "Rigorous JCVS", an alternative test-set developed for RADC by TRW (Ref. 4). We want to determine whether any of the tests it contains can usefully be added to the JCVS. Another current activity is to carefully compare the JCVS with the language specification to assure that every paragraph has a corresponding test. Finally, at the request of the LCA, we are investigating the development of some avionics benchmarks with probes to extract compiler performance data. While not envisioned as a part of the JCVS, these benchmarks may be useful as adjuncts to provide an additional confidence factor for the compiler.

D. PERFORM VALIDATIONS

A very critical function of the LCF is to validate JOVIAL J73 compilers. It is important to note, and apparently easy to forget, that validation does not constitute compiler acceptance testing. Its major purpose is to assure that the compiler interprets and translates source code in accordance with the language specification. Acceptance testing and software verification are separate activities that have nothing to do with compiler validation.

Any JOVIAL J73 compiler to be used for a new Air Force application must be validated. To accomplish this, the Program Office requests validation services from the LCA, who initiates action to establish a Memorandum of Agreement (MOA). The MOA defines the mutual responsibilities of its three participants: (1) the Program Office, which must arrange with its appropriate contractor to assist with the Test Plan preparation by supplying the values of certain parameters unique to the implementation; and perform a trial execution of the JCVS to demonstrate the compiler's readiness for formal validation; (2) the LCF, which must accomplish the validation, analyze and interpret results, and prepare a report; and (3)

the LCA, which must review the report and submit it to the Program Office, with recommendations regarding the compiler's satisfaction of validation requirements. The LCA then tasks the LCF with conduct of the validation in accordance with the terms of the MOA.

Figure 4 describes the essential steps of validation. The LCF first accomplishes careful planning by working with the customer to obtain host and target specific information and, if required, designing special tests to supplement the JCVS. This activity culminates in development of a formal Test Plan containing all required data, the schedule, and proposed analysis methods. When approved by the LCF Manager, this plan guides the validation procedure.

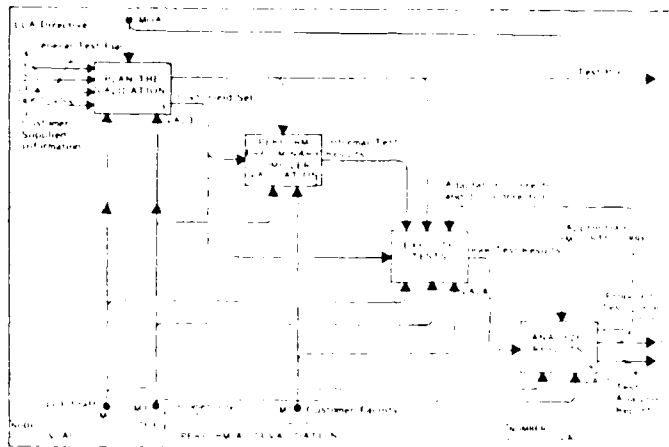


FIGURE 4. PROCEDURE TO PERFORM A J73 VALIDATION

The LCF then assures readiness of the compiler for validation by reviewing the printouts from informal runs of the JCVS accomplished by the customer. For these runs, it is permissible to use an earlier version of the JCVS because the major purpose is to assure that all classes of tests are executable on the host/target facilities. This is normally a straightforward task for the customer, who usually has already acquired and used the JCVS throughout program development to help debug the compiler. The LCF then accomplishes formal testing on-site using the most recent version of the JCVS.

The final step is to analyze results and prepare a formal Test Analysis Report (TAR). The LCF accomplishes the analysis by a careful, manual review of all printouts. For some of the tests, this analysis requires examination of the location and content of each compiler-produced error message to be sure the correct error has been flagged. For other tests, the analysis requires checking to see that all the code compiled without error and, if an error did occur, determining its probable cause. It is during this analysis that the LCF sometimes detects inadequacies in the JCVS that were not revealed by any previously validated compilers. In this case, the LCF develops a VCR and processes a recommended fix as described earlier.

The TAR contains an overview of the testing accomplished and identifies all areas of compiler nonconformance. It also contains the LCF's recommendation about the severity of any discrepancies; identification of those discrepancies that would not have occurred if LCB-approved LCRs were incorporated into the official specification; and identification of any JCVS tests which were discovered to be faulty and for which results are indeterminable. The LCF forwards this report to the LCA, who makes a final recommendation to the Program Office.

One highlight of the validation process is that it involves close coordination with the customer to be sure the JCVS is properly tailored. Each compiler implementation is unique, and this tailoring is always necessary. Also, no changes are ever made to the JCVS during a validation despite all temptations to correct a "small error". This assures good configuration control and allows us to recreate the exact conditions of testing if necessary. Finally, no part of the analysis is trusted totally to automation. Every result is manually analyzed to be sure compiler idiosyncracies are distinguished from real problems.

E. PROVIDE USER AND PROGRAM OFFICE SUPPORT

The final task performed by the LCF is to support the user community and, as requested through the LCA, provide consultation and assistance to Program Offices. Figure 5 identifies the various functions performed in carrying out this task. First, the LCF maintains a data base of user information, including a complete mailing list of JOVIAL users. We also keep information about active compiler and tool development efforts. This allows us to put users with common interests in contact with each other and notify users of important JOVIAL news.

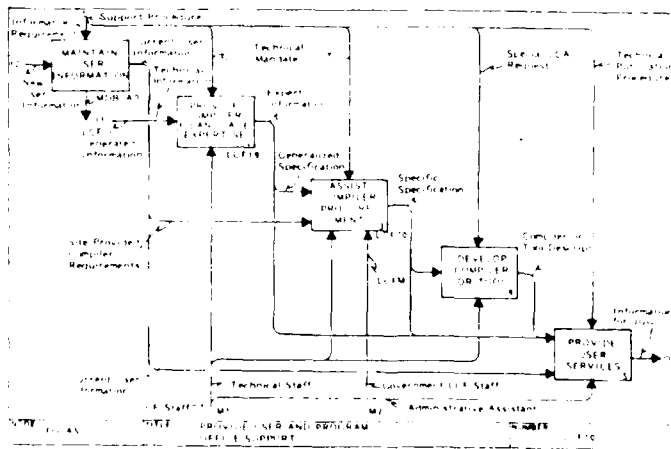


FIGURE 5:
PROCEDURE TO PROVIDE USER AND PROGRAM OFFICE SUPPORT

The LCF provides both compiler and language expertise by staffing itself with experts in these areas and collecting all available JOVIAL documentation for ready reference and referral. The LCF uses this expertise to answer any users' questions and prepare special documentation to discuss frequently raised issues. We also attend all JUG meetings and serve as language consultant there; and we publicize each meeting and edit and distribute minutes using our computer-implemented mailing list. We serve as office-of-record for the JUG, maintaining copies of all minutes, handouts, and briefing material. Finally, we provide training in JOVIAL J73 programming by teaching 4-6 two-week classes a year.

The LCF assists with JOVIAL compiler procurements in two ways. We have developed and periodically update a sample compiler specification and an accompanying guide that explains how to tailor it to unique requirements (Refs. 5 and 6). We also consult with Program Offices as JOVIAL experts and can assist with writing specifications and participate in design reviews as requested.

The LCF is responsible for maintaining JOVIAL tools. Currently in the LCF repository are the JOVIAL Code Auditor, Programming Support Library, and J73 Automated Verification System, all recently developed by RADC. To assist with the transition of these tools to users, we are providing user assistance and hands-on experience in our facility. After assuring each tool performs as described and is implementable on other systems, we begin distributing it. At that time, we establish a Software Problem Reporting (SPR) system for collecting user-reported problems and disseminating known fixes or work-arounds. We are planning full maintenance services for the tools as requirements dictate.

Finally, the LCF provides a variety of user services to promote the use of JOVIAL. On request, we present briefings on any JOVIAL-related topic. We either distribute JOVIAL documents directly or provide information on how to obtain them, and we maintain an extensive reference library and list of available documents. Through an MOA among ourselves, the LCA, and the Federal Software Exchange (FSE), we act as a clearinghouse for all JOVIAL compilers and major support tools enroute to FSE for distribution. This provides us the opportunity to assure these products are complete and ready for distribution and to establish a tracking system for use in keeping holders apprised of updates. We directly distribute selected tools and compilers on an interim basis as necessary. Finally, we keep the JOVIAL community informed about all on-going JOVIAL activities, available services, new products, important events and available software by publishing and distributing a newsletter every two months.

CONSIDERATIONS FOR Ada LANGUAGE CONTROL

Many of the foregoing procedures developed by the Air Force for controlling JOVIAL can be applied directly to Ada. Others will require tailoring, but that will be facilitated by the existing SADT models. The type of tailoring needed for some of these procedures is the topic of this Section, in which we point out some of the more obvious considerations to be made in preparing an LCF for Ada.

A. IMPACT OF DOD-WIDE LANGUAGE. Since Ada is a DoD-wide language, maintenance of the Ada language standard will require coordination among the Air Force, Army, and Navy through the Ada Joint Program Office (AJPO). This will result in a lengthy process unless efforts are made to establish an efficient screening procedure for proposed changes. One approach would be for each Service as well as DoD to have its own LCR analysis and screening procedures. LCRs that are approved at the Service level could then be forwarded to the DoD level for final review and analysis. In effect, the Services would propose changes based principally on criteria of language utility; and the DoD would dispose of or approve those changes based principally on criteria of language and compiler impact and the coordinated satisfaction of the needs of all the Services. The current JOVIAL language control mechanism could serve for the Air Force with adjustment of the criteria for analysis and acceptance.

B. GRADUAL TRANSITION TO Ada. One point that nearly everyone in the standardization community agrees with is, "We want to profit from our lessons learned in JOVIAL and not make the same mistakes in the Ada effort." With that point in mind, the trend we observe in the Air Force toward making the Ada transition a gradual one is readily understood. The transition could occur in four carefully planned phases that we might descriptively title JOVIAL (in effect now), JOVIAL/Ada, Ada/JOVIAL, and Ada. With the benefit of proven language control procedures on which to base the transition and a flexible number of computer resources from which to draw in implementing each phase, the Air Force would enjoy a high probability of success with such an approach.

C. Ada VALIDATION POLICY. The AJPO, staffed with Air Force, Navy, and Army personnel, has the responsibility for ensuring the appropriate validation of Ada compilers throughout the DoD. AJPO policy requires that before a compiler can use the name Ada, it must be fully validated, i.e. there must be a current certificate of validation issued for the compiler from the AJPO. They may also require renewal of the validation every two years. AJPO presently allows use of the trademark Ada in conjunction with partial implementations if a caveat is included in all associated advertisements. These policies mean that frequent retesting of full and partial implementations of Ada may be required, and therefore configuration management of the Ada Compiler Validation Capability (ACVC) test suite will be very important.

Existing configuration management procedures used for the JOVIAL test suite (JCVS) could be directly adapted to the ACVC. As in the case of JOVIAL, using VCRs approved at the Service level would need to be considered. Additionally, VCRs approved at the DoD level to screen for redundancies and implement centralized control.

A final consideration is that with the explosion of Ada implement on microprocessors, there is an attending requirement for the ACVC to adapted to the microprocessor environment. It is unlikely that these processors will host an Ada Programming Support Environment. This area presents additional new challenges for establishing validation and configuration management procedures and tools.

D. SIZE OF Ada USER COMMUNITY. The DoD standardization policy for Ada will obviously result in an Ada users community that exceeds the size of the JOVIAL users community by several orders of magnitude. User services as performed by the LCF for the JOVIAL/Ada Users Group, is already a full-time job, and that job will increase significantly for the Ada users community. We recommend a direct extension of current JOVIAL user services, with addition of a liaison function to interact with other user groups that exist. There is already talk about the JOVIAL/Ada Users Group transitioning to an Ada/JOVIAL Users Group, and by popular demand we have established an "Ada Corner" in the JOVIAL LCF Newsletter. Thus, the transition has already begun.

E. RAPID GROWTH OF Ada EXPERIENCE BASE. With Ada, we anticipate an emphasis on user support and coordination among the Services to assist and dispense a common knowledge base. Then, as the users emerge, we will see rapid growth of the Ada experience base and a high demand for compile and validation services. This means early preparation is essential to be familiar with the ACVC and to refine JOVIAL procedures for administration effectively. We cannot afford to wait too long to get started.

F. Ada AS AN ANSI STANDARD. DoD recognizes that to accomplish its long-term purpose, it must expose Ada to public review and obtain a national consensus. Therefore, DoD approached the American National Standards Institute (ANSI) about making Ada an ANSI standard. Of three possible avenues for accomplishing this, DoD chose the canvass approach. The canvass has been completed, and DoD expects Ada to become an ANSI standard by the summer of 1982.

As sponsor of Ada as an ANSI standard, DoD will be totally responsible for maintenance of the standard. Later, DoD intends to make Ada an international standard through the International Standards Organization (ISO). The degree to which the DoD, ANSI, and ISO standards are affected will be affected by the review process of the respective organizations.

Once Ada is an ANSI standard, it must comply with ANSI rules, which require that the standard must either be revised, reaffirmed, or dropped within a five year time period. This means that any changes to be made to MIL-STD-1815 will be reviewed by the ANSI technical committee before approval is given to implement those changes in the ANSI standard. Furthermore, if Ada becomes an ISO standard, another level of review is required by an international committee to approve changes to the ISO standard. Notice of plans to revise the ISO standard must be given to the international community at least a year ahead of the target date for revision of the standard. The impact of national and international standardization on language control functions, particularly at the D level, could be enormous and needs to be considered as we evolve the Force LCF to service Ada.

SUMMARY AND CONCLUSIONS

A. SUMMARY

JOVIAL is the interim standard language for Air Force avionics embedded computers until Ada becomes available. The LCF is responsible for controlling JOVIAL, and we have developed and fine-tuned the procedures of language control. We have described those procedures in this paper using SADT models and presented some important considerations for their revision as we move toward Ada.

Language control is the assurance of the integrity, stability, consistency, and usability of the language. The four major elements of language control are a well defined and consistent policy for controlling language changes, a mechanism for making these changes, a mechanism for checking for conformance to the language specification, and a centralized knowledge source. The principal control tasks are establishing and maintaining LCF policy, maintaining the language specification, maintaining the validator, performing validations, and providing user and Program Office support. The LCF has developed rigorous descriptions of procedures for these tasks using SADT models. These models promote tight administration of the control function and provide an organized basis for reconfiguring the language control function to new languages, such as Ada.

There are several readily recognized characteristics of Ada that need to be considered in establishing language control for it. First, since Ada is DoD-wide, maintenance of the specification will require inter-Service and AJPO coordination and will be a lengthy process. One approach to streamlining this task may be to establish both a component level and a DoD level of LCR analysis, and, in effect, set up a well-coordinated double-screening process. Second, the Air Force trend toward transitioning to Ada very gradually suggests we should build the Ada control function to operate in parallel with that for JOVIAL, then gradually phase-out the latter. Third, we anticipate a need for frequent testing and retesting of Ada compilers and a possible need for validating partial implementations, including those on microprocessors. This will make configuration management of the ACVC a very important factor in successful test administration, and it will pose many new challenges for language control. Fourth, the large size of the Ada user community will make user support a big job, and liaison among user groups will be necessary. Fifth, we envision a rapid growth of the Ada experience base and an equally rapid transition to a high demand for validation services. It is important to begin preparing as soon as possible. Finally, with Ada as a military (DoD), ANSI, and (potentially) ISO standard, coordination on changes to the language will be especially important and will affect control activities at all levels.

B. CONCLUSIONS

JOVIAL language control is highly structured, well defined, and tightly administered. This is facilitated by the maintainance of SADT models that define LCF operations in considerable detail. These models provide a basis for extending proven Air Force language control functions to Ada. This approach to the Ada transition has a high probability of success because of the Air Force experience with JOVIAL and their systematic evolution and fine-tuning of language control procedures.

ACKNOWLEDGEMENTS

We want to acknowledge the assistance of Mr Frederick D. Pitts, Chief, Computer Operations Division, ASD Computer Center, whose continuing, enthusiastic support of our Branch's function has made this paper possible. We also thank Ms Peggy Hayes for typing the original manuscript and Ms Naomi Byrd for typing final changes and developing the layout.

REFERENCES

1. Ross, D., Structured Analysis (SA): A Language for Communicating Ideas, IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, Jan 1977.
2. JOVIAL Language Control Facility Operating Procedures, SofTech, Inc., LCF-OPM-01.0981, Sep 1981. (Available from LCF)
3. J73 Validator Users' Manual, SofTech, Inc., LCF-VT-02.0381, Mar 1981. (Available from LCF)
4. Hart, R., and McClanahan, M., JOVIAL (J73) Compiler Validator, RADC-TR-81-128, Rome Air Development Center, Griffiss AFB, NY, Jun 1981.
5. General Procurement Specification, SofTech, Inc., LCF-PS-01.0681, Jun 1981. (Available from LCF).
6. Guide for General Procurement Specification, SofTech, Inc., LCF-PS-02.0681, Jun 1981. (Available from LCF).

Biographical Sketch for Patricia A. Knoop

Patricia A. Knoop is manager of the JOVIAL Language Control Facility (LCF) in the Language Control Branch, Computer Operations Division, ASD Computer Center, Wright-Patterson AFB, Ohio. Her work includes managing the LCF support contract that provides five on-site contractor personnel, overseeing the day-to-day operation of the LCF, and interfacing on behalf of the LCF with the JOVIAL/Ada Users Group. She is also responsible for developing plans for future Branch activities dealing with the Air Force's transition to Ada.

Ms Knoop has extensive background in contract management and research and development activities in the area of weapon system simulation. She was formerly manager of the Air Force's exploratory development project for flight simulation technology and has authored numerous technical reports in this area. She is a member of the Association for Computing Machinery and AdaTEC.

Ms Knoop received an AB in Mathematics in 1962 from MacMurray College, Jacksonville, IL, and an MS in Computer and Information Science in 1975 from the Ohio State University, Columbus, OH.

Biographical Sketch for Bobby R. Evans

Mr. Bobby R. Evans is Chief of the Language Control Branch of the ASD Computer Center. The Language Control Branch is responsible for the operation of the JOVIAL Language Control Facility.

His duties include managing the overall operation of the Branch and acting as consultant to the JOVIAL community in the area of JOVIAL J73 language control. Mr. Evans participated in the initial planning and organizing that facilitated the transition of the JOVIAL Language Control Facility from RADC to ASD.

Mr. Evans is a member of the Association of Computing Machinery (ACM) and serves on the board of the Dayton Chapter of the American Society for Public Administration (ASPA).

He holds an MPA degree from the University of Dayton and a BS degree in Mathematics from North Carolina A&T State University.

FEASIBILITY ASSESSMENT OF JOVIAL TO ADA TRANSLATION

Daniel H. Ehrenfried
AFWAL/AAAF-2
WPAFB, Ohio 45433
(513) 255-2446

ABSTRACT

The Ada program is part of a DOD policy change towards incrementally reducing the number of languages in use by the DOD from many to only a few and then eventually to just one; Ada. Currently, MIL-STD-1589B (JOVIAL - J73) is the Air Force standard language for use in the embedded application domain. One approach towards an earlier transition of all software written in Ada would be the development of an automatic J73 to Ada translation system. With the translation of all J73 software into Ada, J73 software development systems could be phased out of use, the cost of maintaining the J73 system could be recovered, and programmers would be freed earlier for their eventual transition to Ada. This paper will examine the feasibility and cost effectiveness of developing a J73 to Ada translation system.

1. Introduction

An initial impetus for standardization on a single High Order Language (HOL) for Department of Defense software applications was to reduce the overabundance of languages requiring support by the DOD. Each language requires a pool of people knowledgeable in the details of that particular language, a compiler, and a set of related support tools in order to write and maintain software in that language. Often, people and resources are not easily interchangeable between different language systems. This redundancy illuminated the need to consolidate to a smaller set of languages that could still satisfy DOD requirements in all application areas.

Currently, MIL-STD-1589B (JOVIAL - J73) is the Air Force standard language for use in the Embedded Computer System (ECS) domain. Software written today in J73 will probably last throughout the lifetime of the weapon system in which it is used. History shows that this life span is normally between 5 and 10 years, and often is more than 15. A J73 language system (compiler and support tools) and the necessary complement of trained personnel will also be required throughout this life cycle.

The Ada program is part of a DOD policy change towards incrementally reducing the number of languages in use from many to only a few, and then eventually to just one; Ada. One approach towards an earlier transition from all software written in J73 to software written in Ada would be the development of an automatic J73 to Ada translation system. With the translation of all J73 software into Ada, the J73 language system could be

phased out of use, the cost of maintaining the J73 system could be recovered, and programmers would be freed earlier for their eventual transition to Ada. The question is whether this is a rational step towards standardization on a single HOL.

Three issues must be addressed before this approach should be adopted by the Air Force. First, the feasibility of an automatic translation system must be demonstrated. It must be proven that a sufficient percentage of the J73 language can be correctly translated into Ada. Second, the impact of translation upon the quality of the resulting software must be measured. A translation system must not only produce a correct translation but also maintain the quality of the resulting software including its maintainability, reliability, and robustness. Within the highly constrained environment of real-time embedded software, it is critical that efficiency be preserved as well. Finally, the cost effectiveness of this approach must be shown. The cost of performing the translation must be weighed against the expected savings from an early transition to Ada.

2. Requirements

Section 2.1 will define a set of general requirements for performing source-to-source translation at the HOL level. Section 2.2 will refine these requirements to reflect the specific characteristics of J73, Ada, the embedded applications environment, and the state of the art in translation technology.

2.1 Source-to-Source Translation

Any translation system must preserve the characteristics of the original program in two major ways: 1) execution equivalency including functionality and efficiency and 2) source code quality.

2.1.1 Execution Equivalence

The original program and the resulting translation must be equivalent to the largest extent possible. Exact equivalence might be defined as two source code modules which, when compiled, produce the exact same load module for a given target. Even if this were possible to attain, exact equivalence is certainly not necessary. The critical measure is that the two pieces of software are functionally equivalent; that they perform the same task.

Thus, at a minimum, any functional requirements placed upon the original code must be preserved during translation. The resulting translation must produce the same "effect" on the outside world as the original. Other more specific restrictions might include:

1. External inputs should be interpreted and stored in the same manner.

2. For a given set of inputs, outputs should emanate with the same values and in the same order.
3. Critical Timing dependencies within the original program must be met by the resulting translation.

2.1.1.1 Efficiency

The discussion of efficiency often reduces to a question of tradeoffs between available resources and time. In programming terms, one major resource is storage space and time is measured in execution cycles. Space can be optimized by packing data into the smallest representation possible. Unfortunately, additional effort must then be exerted (and time expended) to extract the data when it is required, and replace it after it has been modified. Through the use of data redundancy, processing time can, in most cases, be reduced. All applications must strike an appropriate balance between space and time to fit their underlying hardware resources and meet any timing requirements.

Programs must not only be translated correctly, but also preserve the efficiency characteristics of the original program. Again, exact equivalence between two programs written in different HOLs would require that the same number of machine instructions be used to implement all functional aspects, and that the same amount of storage be used for any accompanying data. This goal is just as unattainable as exact functional equivalence and even more unnecessary. A more reasonable requirement might be to restrict the translation to be, on average, no less efficient than the original. The overall size of the code and execution speed of the translation should not exceed that of the original in any considerable way. Minor local aberrations may in some cases be tolerable, but approximate global parity must be preserved and local deviations must not be too large.

2.1.2 Source Code Quality

If the software is expected to have a reasonably long life cycle, it must be of high quality. Software systems are constantly being modified and updated to fix newly detected errors and reflect changing requirements. Quality software can make this process easier and more cost effective. Execution characteristics are not the only measure by which the quality of software is judged. Maintainability and reliability are critical metrics of quality. It should be as readable, easily understandable, and embrace the style and intent of the language in which it is coded. Translations should also result in robust implementations, using to the fullest extent possible the power of the target HOL. An equivalent or greater level of these qualities should be present in the software resulting from translation.

2.1.2.1 Maintainability and Reliability

"Maintenance" is a deceptive term when applied to software. It does not imply that one must apply constant tinkering to maintain a constant level of functioning as one might a piece of machinery. Except for the possibility of hardware error, programs should by definition execute in the same manner each time they are invoked with the same inputs. This term actually refers to the fixing of bugs in software that deviate from the original set of functional requirements or the modification of a program to reflect a new set. Several qualities make the maintenance of software easier to perform. Software should be well structured with separate modules for separate functions. It should exhibit clear data flow between modules and clear flow of control within. It should be well documented and commented in a manner that promotes the understanding of its intent. The specification or functional interface of a subprogram and data should be separate from its implementation.

"Reliability" is a related term that is probably misapplied as well. The definition used here is the quality of a program to isolate the effects of inevitable programming bugs. The following aspects of an HOL and its programming environment can improve the reliability of software. The first is the automatic detection of bugs. Through exact specification (typing) and redundant information about intent (declarations) many bugs can be found at compile time, and corrected immediately. A second aspect is a program's ability to localize the effect of modifications to a program. Subtle dependencies between two sections of software within one program can allow errors to occur in one section when changes are made to the other. (Global flags are a good example of such dependencies.) Dependencies other than those explicitly placed in the software in a clear manner should be avoided whenever possible. Third, access to the internal definition of data and functions should be restricted only to parts of the program that require it. This is often called Information Hiding. Finally, modularization and the grouping of similar functions together contribute to reliable code.

A translation system should preserve or improve the level of these qualities in the resulting program. This is often very difficult to do. The quality of software is frequently inherent in the design of the software and the features of the language used in the implementation. The original software may even violate some of the quality standards given above. We can reasonably attempt, therefore, to preserve only those qualities that are present in the original software.

2.1.2.2 Robustness

Robustness is a measure of how well a program utilizes the features of the language in which it is written. Software written in a particular HOL should take advantage of its powerful features whenever possible. This not only improves efficiency but also provides a clearer representation of what the program is intended to do. Features are placed in a language for a reason and should be used whenever the intent of the feature is applicable to the requirement at hand. A translation system should attempt to provide the highest possible utilization of the target HOL.

2.2 A J73 to Ada Translation System

This section will now consider translation system requirements when the source HOL is JOVIAL / J73, Ada is the target HOL, and the type of software to be translated is real-time embedded software.

2.2.1 Scope

A compiler is an example of a system that translates programs written in a high order language into equivalent programs "written" in machine language. A source-to-source translation system would have to perform many of the same "front end" analysis functions as a compiler for the same source HOL; the difference lying in the level of the target language. The capabilities of today's compiler technology can therefore be used as a baseline for analyzing the limits of a source-to-source translation system.

From the outset, we must remind ourselves of the limited capacity to which machines (or the programs that run on them) can understand programs written in one language and translate them into another. An ideal system would accept any legal J73 program and return the Ada equivalent. In practice, however, translation of J73 to Ada will require a mixture of both automatic translation and human augmentation. Two things can be stated with certainty:

- 1) All J73 programs have a functionally equivalent Ada implementation. J73 and Ada are alike in many ways. Although difficult to prove, neither can implement a function that the other cannot duplicate. Turing Equivalence should guarantee that a skilled programmer can design and code an algorithm in both languages that is functionally equivalent.

- 2) The translation from J73 to Ada cannot be a 100% automated process. J73 and Ada are also dissimilar in many ways. Several J73 features have no corresponding feature in Ada. When a particular feature cannot be translated directly, some alternative must be found. A translation system may not be able to understand the meaning of a whole program well enough to find an alternative combination of Ada features that provide the same effect. A human will have to step in and redesign sections in Ada, then integrate them with the rest of the program. The extent to which this must be done is the critical measure of a translation system's viability.

2.2.2 Semantic Equivalence

Section 2.1.1 stated the requirement that the original program and resulting translation be "functionally equivalent". Unfortunately, today's compilers can not "understand" at this high a level. They can recognize and understand most constructs at the statement level only. Thus, "functional equivalence" must also be applied at this level. (Both languages contain definitions of more than just "statements". Here, reference to a "statement" is intended to mean any separately defined construct of the language, including those constructs not formally defined as statements.)

In order to make a comparison between two similar statements from two

different languages, and arrive at a judgement of their equivalence or lack thereof, a precise definition must be available for both the "source" and "target" statements. Unfortunately, MIL-STD-1589B contains many constructs for which the definition is incomplete or ambiguous. This allows the compiler writer the freedom to interpret the definition to mean either what is most logical to him or easiest to implement. J73 also defines several parts of the language to be "implementation dependent", again allowing the compiler writer the freedom of choice. The same is true for the definition of Ada, but to a much more limited extent. Every attempt was made during the language definition to provide the most complete definition possible, to remove any ambiguities, and to isolate machine dependencies.

Incomplete, ambiguous, and implementation dependent definitions have dire consequences for general purpose J73 to Ada translation systems. In order to have the widest possible application, such a translation system would have to be flexible enough to adapt to the various interpretations that particular J73 and Ada implementations have adopted. Another option might be to have several translation systems, each tailored to a particular compiler pair, although the economics of such a solution would probably be prohibitive.

2.2.3 Efficiency

Real-time software systems are bound by very stringent efficiency requirements. Memory is usually limited and the computing power is seldom adequate to execute every desirable function. Software often has to be "shoehorned" into memory with little if any space to spare. A translating system must therefore minimize its impact on both time and space. There are several pitfalls awaiting our attempts to translate J73 to Ada. Numeric accuracy and data representation are defined differently in J73 and Ada. Differences between implementation dependent parameters and options are especially troublesome. And each compiler, both the original J73 compiler and the Ada compiler for the translated program, will make space/time tradeoff decisions in a different manner.

2.3 Requirements Summary

1. The semantic equivalence between J73 statements or blocks of statements and Ada must be guaranteed.
2. Data storage requirements must remain approximately equivalent. Induced processing overhead must be minimized. The exact toleration thresholds are application dependent.
3. The translation must produce readable code. It should be well structured in the style and intent of Ada.
4. The resulting code should be free of subtle underlying dependencies.
5. The translation system should utilize the features of Ada to the largest extent possible.

3. Analysis

3.1 Automatic J73 to Ada Translation

This section will mirror the structure of the MIL-STD-1589B definition of the J73 language. Chapter titles in the Standard match subsection titles here. All comments relating to the translation of J73 constructs will appear in the appropriate section. Due to the space restrictions of this publication, only comment relating to major incompatibilities shall be included in this discussion. (A non-abbreviated copy of this paper may be obtained from the author.) Any violations of the requirements given in 2.3 will be so noted. All references to the Ada Language Reference Manual (LRM) refer to the July 1982 version of that document.

3.1.0.1 Compool Modules (1.2.1)

Compools can be translated into Ada packages with some minor caveats. The Ada package is actually much more powerful than the J73 Compool. Packages can contain variables. Compools can declare variables, but only with the external DEF construct. Compools have nothing corresponding to private or limited private types. Package bodies may contain internal declarations not visible outside the package for use in the internal implementation of the package specification. Package bodies may also contain an executable part similar to the BEGIN ... END of a subprogram definition. Compools support none of these capabilities.

3.1.0.2 Implementation Parameters (1.4)

The J73 LRM contains the following statement: "The machine on which a J73 program runs contains an array of memory cells." Ada does not make this specific a statement about the hardware on which it will run.

This difference contains several implications. The most obvious one appears in this section; namely the existence of implementation parameters relating to linear memory. Programs which refer to these parameters will require the same value when translated into Ada. Other implementation parameters are J73 dependent; i.e., MAXTABLESIZE. Tables are obviously not in Ada. The name "MAXTABLESIZE" would make no sense in an Ada program. Parameters such as this will probably have to be hand translated.

3.1.1 Declarations

3.1.1.1 Integer Type Descriptions (2.1.1.1)

J73 and Ada differ in the manner in which they define the range of integers. Ada allows the range to be arbitrary (within the bounds of the SYSTEM INTEGER RANGE) and be expressed in decimal or as a based number. J73 requires that "... the minimum number of bits required to hold the maximum value of the integer (excluding the sign, if any)..." be given in the ITEM declaration of an integer. While this does allow for the implication of a range constraint, it offers them only with limits of powers of two. For example, the declaration:

ITEM X U 4

declares the unsigned integer X with 4 bits to hold its values. This implies a range of 0 to 15.

There are several problems with the J73 definition provided. It does not define what happens when an attempt is made to assign to an integer ITEM with a value larger than it is allowed to hold. Does rounding or truncation occur? Are the high order bits masked, performing something like modulo arithmetic? Is the size constraint ignored resulting in no effect. Does the execution of the program halt? MIL-STD-1589B simply does not say.

Ada contains the concept of a range constraint. An exception, CONSTRAINT_ERROR, will be raised if an attempt is made to assign a value outside the declared range. J73 has no concept of exceptions, exception handlers, or even of error conditions. One approach towards avoiding CONSTRAINT_ERROR exceptions would involve translating all integer definitions into the SYSTEM defined integer type ignoring any <integer-size> attributes. There are two problems with this. One, an exception cannot be avoided entirely if an attempt is made to assign a value outside INTEGER'RANGE. And two, it is hard to justify ignoring this attribute when the original programmer took the time to specify it, and must have done so for a reason.

The J73 <round-or-truncate> attribute is also troublesome. Either rounding or truncation invoked during type conversion in J73 (specified by the ITEM declaration of the target variable). The exact algorithms for truncation and rounding are not specified and so must be assumed to be implementation dependent. The J73 manual statement: "If the [<round-or-truncate>] attribute is omitted, truncation in an implementation-dependent manner will occur." further muddies the water.

3.1.1.2 Floating Type Descriptions (2.1.1.2)

The definition of J73 floating point numerics have many of the same problems as that of integers. The <precision> field again refers to the number of bits needed to represent the mantissa. It offers no ability to specify a range constraint. Ada requires precision to be defined as the number of decimal digits for the mantissa and allows for a range constraint. Anomalies between the representation of the mantissa as decimal digits and binary bits may cause problems. The definition of rounding and truncation is missing. If the attribute is omitted, truncation in an implementation dependent manner will again occur.

3.1.1.3 Fixed Type Descriptions (2.1.1.3)

Precision and range are again specified in numbers of bits. J73 fixed type declarations contain two attribute fields. The <scale-specifier> indicates the number of bits to the left of the decimal point including the sign bit. It is unclear whether this implies a range constraint similar to that for integers. The <fraction-specifier> indicates the number of bits to the right of the decimal point. Again, Ada differs in its specification semantics for this data type. Ada allows specification of a delta and a range constraint. It is unclear whether these two definitions are compatible in all cases. A definition for rounding and truncation is implementation dependent and not provided in the manual.

3.1.1.4 Status Type Descriptions (2.1.1.6)

In Ada, enumeration types must be named types. Objects can not be directly declared as enumeration types as with arrays, etc. Thus, a type declaration and type name must be generated when a status declaration is translated into an Ada enumeration type. The problems associated with generating an appropriate name will be discussed in section 3.1.7.1.

3.1.1.5 Pointer Type Descriptions (2.1.1.7)

J73 pointers appear to be equivalent to Ada access types. In fact, they are not.

Access types are included in Ada for two reasons. Their primary purpose is as the mechanism for naming dynamically created objects. Static objects are given a name reference at declaration time. Dynamically created objects are given an internal name by which to reference them. Access types hold these name values. Access values are typed in that they can only hold references to objects of one type. Access types also provide a convenient way to implement directed graph structures.

J73 pointers differ in the following respects. A minor difference is that pointers can be untyped in J73. Untyped pointers will not translate into Ada. The major difference is that pointers are actually defined to be the address of the object pointed to. The functions LOC and NEXT move pointers around the address space allowing access to the internal structure of all data objects. Pointers can also be converted into integers and bit strings. This allows manipulation with integer and bit string operators. The values can then be converted back into a pointer. In this way, all data (and possibly even instructions) is exposed to meddling from anywhere in the program. Ada was designed specifically to prevent programmers from accessing data in this manner. Pointers, therefore, can not be translated into access types. All code involving pointer types will have to be hand translated. This is a very serious violation of requirement 1.

WD-A142 776

PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND. (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O.
C A PORUBCANSKY NOV 82

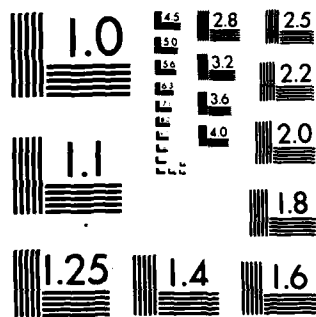
2/6

UNCLASSIFIED

F/G 9/3

NL

A large grid of blacked-out cells, likely representing redacted data or a table of contents. The grid consists of approximately 14 columns and 12 rows of solid black squares.



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3.1.1.6 Table Declarations (2.1.2)

There is no directly parallel structure for J73 TABLES in Ada. Table-like structures can be composed with an array of records. This works fairly well with several small problems. Most of the incompatibilities occur with the several special case rules connected with TABLES.

Ada requires record types to have a name in a similar manner to enumeration types. Objects cannot be directly declared as a record. They can be declared only as a record type declared elsewhere. This requires a name to be generated for the record type to match the internal structure of the table. Name generation is discussed further in section 3.1.7.1.

The following pointer related restriction in the J73 manual precludes the use table types. "Items in tables declared with a <table-type-name> can only be accessed using pointers to the tables." Since pointers cannot be translated, tables declared with a type name cannot either.

3.1.1.7 Table Structure (2.1.2.2)

In J73, the programmer can specify the layout of tables in memory. Ada allows the programmer no control over the manner in which arrays of records are laid out.

3.1.1.8 Ordinary Table Entries (2.1.2.3)

There are a couple problems here. An equivalent to the J73 <order-directive> is not available in Ada. Secondly, J73 provides for 3 levels of packing; some in an implementation dependent fashion. Ada allows for one level through the PRAGMA(PACK). It is unclear whether the mapping of both (M)edium and (D)ense packing will have any effect on translated programs or not.

3.1.1.9 Block Declarations (2.1.4)

"A <block-declaration> declares a group of items, tables, and other blocks that are to be allocated in a contiguous area of storage." Presumably blocks are used to improve the access efficiency to data contained within the block. Ada does not define an equivalent construct. Perhaps this can be ignored during translation, but programmers who specifically used a BLOCK construct probably did so for a reason. It is likely that some translations will be effected if block designations are ignored.

3.1.1.10 Allocation of Data Objects (2.1.5)

Ada does not explicitly provide a static allocation specifier. Variables contained in packages do remain allocated for the life of the package in which they are contained. Thus, data objects declared in packages are essentially static. One approach towards the translation of STATIC data might be to encapsulate all modules that define STATIC data inside a new package. The proliferation of packages each containing just one module for the sole purpose achieving STATIC data would have a

tremendous impact on the readability of programs. This is not the intended purpose of packages.

3.1.1.11 Define Declarations (2.4)

Ada has nothing equivalent to J73 DEFINE declarations. The concept of generics in Ada is close but does not have the same semantics. DEFINES can be expanded before they are run through the translator resulting in a correct program, but the modularity and structure of the DEFINES will have been destroyed. This will impact readability to the extent that DEFINES are used in the original program.

3.1.1.12 External Declarations (2.5)

The Ada mechanism for exporting and importing name references is the package construct coupled with the WITH clause. Several compatibility problems exist between the J73 DEF - REF mechanism and the Ada package / WITH.

Single names can be pulled out of compools through use of the DEF - REF mechanism. Ada has no such mechanism. The WITH clause imports all names declared within the referenced package. If REF specs are simply translated into WITH clauses with the compool/package name, some name conflicts may arise. Since Ada allows name overloading in some cases, the error may not be immediately apparent. In fact, REFERENCE to single names were likely made to avoid conflict with other names in the compool. For DEF specs that are not contained in compools, there is no corresponding Ada mechanism. The variable could be encapsulated within a package and then WITHed into the declaring module and all modules with a REF spec, but this is terribly cumbersome and results in poorly structured code. This further proliferation of packages should be avoided.

3.1.1.13 Overlay Declarations (2.6)

J73 allows entire objects or portions of objects (i.e. tables) to occupy the same storage space. The J73 manual states: "2) that certain objects are to occupy the same memory locations as other data objects." Ada strictly forbids the overlays. Section 13.5 of the Ada LRM states: "Address clauses should not be used to achieve overlays of objects or overlays of program units. Nor should a given interrupt be linked to more than one entry. Any program using address clauses to that effect is erroneous."

3.1.2 Procedures and Functions

3.1.2.1 Parameters of Procedures and Functions (3.3)

J73 allows the programmers to designate the actual binding mechanism to be used during parameter passing. Ada provides no such capability; allowing the compiler to make the appropriate choice. In fact, the Ada LRM states: "A program is erroneous if its effect depends on which mechanism is selected by the implementation." J73 programmers who do specify the type of binding mechanism will have done so for a reason and will likely

rely on the mechanism for the correct functioning of their program. This conflict in definitions cannot be resolved. Therefore, any subprograms that specify the parameter binding mechanism cannot be automatically translated. This is a serious violation of requirement 1.

J73 and Ada differ significantly in their methods of defining of formal parameters. J73 allows the type definitions of formal parameters to be given within the subroutine body. Type definitions may also be any type definition. Ada requires that formal parameters be given an immediate subtype indication. This means that the formal parameter must be declared as a subtype of some previously declared and visible type name. (Formal parameters cannot be directly declared as arrays or records or as enumeration types.) In order to be translated from J73 to Ada, the type definitions must be elevated to a level where both the subprogram definition and the module containing the subprogram call can see them. This elevation not only complicates the structure of the program but can also cause name conflicts.

3.1.3 Statements

3.1.3.1 Loop Statements (4.2)

J73 and Ada differ in the definition of loop statements.

Ada allows incrementation through a scalar range only by 1. Thus, J73 BY and THEN statements will have to be fabricated. An expression to calculate the correct value at each iteration must be formed as dictated by the original BY or THEN formula. A temporary variable to hold this value will also be required and all references to the original loop variable will have to be changed to reference the temporary variable. A <while-phrase> attached to a BY or THEN phrase will have to appear as an explicit test and EXIT. All this may require extra storage and additional computations though the impact should be minimal.

A major problem occurs when the <control-item> in a J73 loop is a <control-variable> (is declared as a variable in the local scope). J73 allows modifications to such variables within the loop and use of their value after the loop statements is terminated. Ada does not allow this. Temporary variables will not work in this case. One might try to assign the value of the loop parameter to the variable declared in the outer scope just before exit from the loop. But the generalized GOTO will prevent a guarantee that the assignment will happen in all cases. Loop statements that have <control-variables> cannot be translated.

3.1.3.2 GOTO Statements (4.7)

Ada does not allow labels as parameters or allow GOTOS to reference labels outside the scope of the GOTO statement itself. This type of GOTO cannot be translated. This J73 feature might be comparable to the Ada exception facility. Labels passed as parameters could designate "handlers" for errors within the subroutine. GOTOS to these labels could act as the Ada RAISE statement. Although a GOTO to a label passed by calling procedure may be used in this way, it can also be used in other ways that

do not map into the Ada exception facility. It is safe to say that GOTOs to labels cannot be translated and that the Ada exception facility will not be used by a translation system.

J73 does not allow GOTOs into statements within a loop referred to in the manual as a <controlled-statement>. The manual does not prohibit GOTOs into a <conditional-statement> or into IF statements. Ada does not allow this. Such "out of scope" analysis will have to be performed prior to the translation of a GOTO.

3.1.3.3 STOP Statements (4.9)

Stop statements have no parallel in Ada.

3.1.3.4 ABORT Statements (4.10)

Abort statements, similar to GOTOs to statement names passed as parameters, cannot be translated into Ada. They are in no way equivalent to the Ada abort statement that relates to the Ada tasking facility.

3.1.4 Formulas

3.1.4.1 Numeric Formulas (5.1)

The definitions for all numeric formulas are incomplete in that they do not specify what happens for error conditions. For example, the J73 manual specifies that: "The right operand of "/" and MOD must be non-zero." But it does not say what happens when it is zero. Range constraints are also specified but nothing is defined when they are violated. This is a serious semantic difference between J73 and Ada. An Ada exception is defined for all possible violations of language restrictions.

3.1.4.2 Integer Formulas (5.1.1)

The modulus operator is defined differently in J73 and Ada. Section 4.5.5 of the Ada LRM defines the modulus operator as:
 $A \text{ mod } B = (A + K*B) \text{ mod } B$; J73 gives: $A \text{ mod } B = A - (A/B) * B$.
These definitions do not give the same answer when A is negative and B is positive. Therefore, the MOD operator cannot be directly translated.

3.1.5 Data References

3.1.6 Type Matching and Conversions

In general, the rules governing type conversions are much less restrictive than they are in Ada. Ada allows (explicit) type conversions in three cases. The following is a summary of the rules for allowed conversions. Complete definitions appear in section 4.6 of the Ada LRM.

1. Numeric types can be converted to other numerics types. Conversions from a real value into an integer type involves rounding.

2. Conversion is allowed when the type of the operand is directly derived from the type mark of the conversion.
3. Array types can be converted when both the operand type and the type mark of the conversion have the same index and component types.

J73 allows many other legal conversions. The following is a list of incompatibilities:

- Numeric conversion seems to be ok. Questions of accuracy, rounding, and truncation are still unclear.
- By allowing any data object to be converted into a bit string and any bit string to be converted back to any other type, J73 completely destroys the concept of information hiding and data consistency. This capability allows anyone to access the "guts" of any data objects. Thus everything is available to anyone who can see it. This conflicts with one of the basic design tenets of Ada.
- The allowance for converting pointer types to integers and bit strings is a primary reason why pointers cannot be translated into access types.
- BIT_STRINGS are implemented as an array of BOOLEANS in Ada, and as such are governed by the rules for array conversion in Ada. J73 implicit conversions between bit strings of different sizes can therefore not be translated.

3.1.7 Basic Elements

3.1.7.1 Names (8.2.1)

As discussed in previous sections, there are instances when the translator will have to generate a name. To prevent conflict with other names declared within the same scope, the name must be unique. Identifier names should also be readable and imply something about the object which they denote. The combined requirements of uniqueness and readability are incompatible. In order to guarantee uniqueness, readable names cannot be used; they are likely to already exist. One possibility is to use a character allowed in Ada but not in J73 such as the underscore character. Names with an underscore anywhere would always be unique as long as the translator did not generate the same name twice. The key problems with this is making the generated name make sense in the local context. This requires a handle name already declared in the local scope and the attachment of an underscore and a suffix or prefix. Even this does not guarantee a suitable name. The best solution is to generate a definitely unique name and do the best possible with its actual content.

Ada allows the use of the underscore character in identifiers to make them more useful. Jovial does not allow this. Thus, since identifier names are translated verbatim, they will not appear in the same style as Ada identifiers. This conflicts with requirement 3.

3.1.7.2 Reserved Words (8.2.2)

Any name that is not in the J73 reserved word list can be used as an identifier in a J73 program. There are, however, some Ada reserved words that do not appear in the J73 reserved list. This poses the potential for name conflict.

3.1.7.3 Comments (8.4)

Although seemingly innocuous, comments pose a very serious problem. The syntax translation from the J73 "comment" or %comment% to the Ada --comment is obviously trivial. But the translation of the actual wording of the comments themselves is not.

Comments often refer to language constructs. Comments in a J73 program might read: "This table is used to store aircraft attitude vectors". Or: "Value-result binding is used here to ...". If these comments were to be translated verbatim, they would be confusing and self defeating. Comments may also refer to names which have disappeared during translation. The names of DEFINE constructs which have been expanded during translation is an example. J73 numeric type designators are another.

A complicating factor is our ability to recognize when comments are relevant and helpful and when they are not. Unless a translating system is prepared to solve the problem of deciphering the English language, it can safely be said that all comments must be suspect and therefore discarded. This is extremely damaging to the quality of the resulting code. Of course, a human could run through the code and fill in comments by looking at the original code, but this would substantially increase the percentage of work required after translation.

3.1.8 Directives

The J73 manual states: "<Directives> are used to provide supplemental information to a compiler about the <complete-program>, and to provide compiler control." This makes them comparable to Ada pragmas. Some of the predefined J73 directives match well with Ada predefined pragmas. These are primarily text and listing control directives such as COPY, SKIP, BEGIN, and END.

Some J73 directives violate the Ada language definition such as expression evaluation directives, initialization directives, and allocation order directives. The use of the !LEFTRIGHT directive in a J73 program has very serious consequences. This directive forces left to right evaluation of operators at the same precedence level. This is incompatible with the Ada LRM statement that "A program that relies on a specific order (for example because of mutual side effects) is therefore erroneous." The

reason that this is so serious is that Ada programs that contain dependencies in the evaluation order of operands will compile without error, but may not execute as intended. One solution to this problem might be to impose a left to right evaluation order through the use of explicit parentheses. Although possible, the resulting code will become less readable and subtly embed dependencies. Future modification can expect problems unless the exact format is preserved. Unless these side effect can be tolerated, all code under the directive !LEFTRIGHT must be suspect and can not be guaranteed to be semantically equivalent.

3.2 Summary of Untranslatable Features

J73 and Ada have a variety of incompatibilities. There are several basic design tenets of each language that do not match well. This results both in constructs that have no equivalent in Ada and ones for which a correct translation has a major impact on the quality of the resulting software. Still other J73 features are considered to have high risk for translation. These features have definitions that are very similar to Ada, but anomalies in their implementations may result in some incompatibilities in some translations. These classifications are summarized in figure 1.

3.3 Percentage Translatable

The percentage of J73 constructs that can be automatically translated into Ada can be measured in two ways. The first method is a straight ratio between those constructs that can be translated and those that cannot. This measure has limited utility, however, since our goal is to translate real J73 programs, and not just the reference manual. A more useful metric is the average percentage of real J73 programs that can be translated. This measure takes into consideration the relative frequency of constructs appearing in real programs. It also considers the amount of local translatable code that is "poisoned" by constructs that cannot be translated.

It is very difficult to estimate how much code will be poisoned by other local untranslatable statements. This can happen in several ways. 1) The construct may be an integral part of the local algorithm. Even though most of the algorithm can be translated, the lack of the untranslatable construct will likely prohibit the module from performing its assigned task. It is also unlikely that there is a quick, local patch. If there were, the translator would be able to substitute it as an equivalent construct. 2) Illegal declarations can invalidate references to those objects. 3) The LEFTRIGHT directive is very pervasive. Any code within the area affected by this directive must be suspect. 4) DEFINES are heavily used in J73 programs. If their negative impact on program modularity cannot be tolerated, large chunks of code will not be translated. 5) GOTOs into IF statements or to parameters. It is safe to say that most J73 constructs that violate the rules of Ada will poison much of the surrounding code.

What then is the average percentage translatable? With the above discussions in mind, 30% to 40% of all J73 programs should be achievable with a good system.

Figure 1: J73 - Ada Definition Conflicts

Conflicting Design Concepts:

- Information Hiding
- Type Composition
- Type Conversion
- Data Representation and Access
- Name Importation / Exportation
- Error Handling

Specific Untranslatable Constructs:

Declarations:

- Pointers
- Table Structure Specifiers
- Statement Name Declarations

Procedures and Functions:

- Formal Parameter Declarations
- Machine Specific Procedures

Statements:

- LOOPS with Control Variables
- GOTOs to Statement Names
- STOP and ABORT Statements

Type conversions:

- Primarily Conversions to and from INTEGER and BIT Types

Directives:

- Some COMPOOL Directives

Translations Impacting Quality:

- Static Allocation
- Define Declarations
- External Declarations
- LOOP temporary variables
- CASE FALLTHRU option
- Name Generation
- Comments
- LEFTRIGHT Directive

High Risk Constructs:

- Numeric precision
- Numeric truncation and rounding
- Blocks
- Bit string operators

3.4 Summary of Unused Ada Constructs

Several Ada features have no equivalent J73 constructs and will therefore be absent entirely from automatically translated programs. They include: tasking facilities, exceptions, generics, Ada I/O, access type and dynamic data allocation, and overloading. Still others, such as packages and the Ada typing system, are not utilized to their fullest extent. Programs translated into Ada will use a subset which does not include these features. (If translation is augmented by human redesign, some of these features may be used.)

3.5 Cost Effectiveness

The stated objective was to remove the need for maintaining a J73 programming environment by switching all code into Ada; thereby removing the cost of maintaining it. These cost savings must be weighed against the cost of developing a translation system, the cost of translating large amounts of complex software, and the differential cost, if any between maintaining the program in the J73 and Ada environments. This section will not attempt to attach actual figures to each cost but will outline the types of costs that can be expected. Estimates will be given when known.

3.5.1 Translation System Development Costs

As stated in section 2.2.1, the complexity and thus the cost of a translation system would be similar to the cost of a compiler. It is unclear, however, whether just one translator can handle all J73 translations. The analysis in section 3.1 provides several examples where MIL-STD-1589B is ambiguous and contains many implementation dependent features. Several interpretations of MIL-STD-1589B exist and are embodied in J73 compilers used today. Programs that work correctly when compiled on these systems will require the same interpretation set in the translator in order to be translated correctly. Each point of interpretation must be reflected in a translator option in order to provide a correct interpretation and translation.

A translation system would be a short-lived system. Once all J73 was translated into Ada, the system would have no further use. Thus, it would not require the normal maintenance to fix bugs. This is a blessing in disguise, however. It means that all (or an extremely high percentage) bugs must be removed before it can be successfully used at all.

3.5.2 Code Translation Costs

Once a translator is built and functioning correctly, the primary cost will be the labor of programmers skilled in both J73 and Ada. They would be required to clean up the translation to provide a full translation. These costs are directly proportional to the amount of human translation required.

Of course, the resulting translated code must be entirely retested to certify that the new program satisfies all of the functional requirements. This is very often non-trivial, expensive operation. At this stage the

program could be considered an Ada program and all modifications made in Ada.

3.5.3 J73 versus Ada

Ada was designed to reduce the cost of maintaining software through the use of new concepts in the structuring of programs and data. These concepts were not placed in J73. As we have seen in section 3.1, the features in Ada that were designed for this purpose could not be correctly utilized by a translator. We, therefore, cannot expect to realize the advantages of Ada. We can expect the resulting programs to require the same effort to maintain as the originals.

4. Conclusions and Recommendations

Several conclusions can be drawn from the discussion above:

- A high percentage of real J73 programs cannot be automatically translated.
- The translation of numerics is risky. Some precision errors must be anticipated in some translations.

A large amount of human supplement is required to fully translate J73 programs into Ada.

- The resulting code may be poorly structured. Several of the program structuring facilities in J73 and Ada have incompatibilities. Several Ada structuring facilities will not be utilized by a translator.
- The resulting code may be hard to read and understand. The lack of translatable comments constitutes the largest impact. Name translation and name generation also effect readability.
- The style of the resulting program will still be J73 style. A translator will only rewrap J73 style programs in Ada syntax.
- The resulting programs will not be as robust as they should be. Several Ada features are not used by the translator.

Due to the overwhelming number of negative conclusions, the development of a J73 to Ada translation system is not recommended at this time. The best solution to the problems is to leave the J73 programs as they are until their life cycle is terminated. If programs must be translated into Ada, it is recommended that they be redesigned in Ada and entirely hand translated into Ada.

One possible use for our capability to translate some J73 constructs might be the development of a "local" translator. A human could bracket off portions of code that can be translated effectively. If some portion of the bracketed code could not be translated, then no part would be translated. These segments of translation could be used in conjunction with an editor to hand translate programs. Such a "local" translator could remove the tedium of translating these portions of code.

BIOGRAPHY

Lieutenant Ehrenfried graduated in 1980 from the Massachusetts Institute of Technology with a Bachelor's Degree in Electrical Engineering and Computer Science. He entered the Air Force the following fall, taking a Project Engineering position in the System Avionics Division of the Air Force Aeronautical Laboratories at Wright-Patterson Air Force Base, Ohio. He worked for one year on the use of hardware monitoring technology for debugging real-time software on 1750A processors. In January 1982 he switched into the Ada arena where he has been studying the use of Ada.

AD-P003 524

THE MULTIPLE SYSTEM OFF SUPPORT (MSOS) SYSTEM,
A PRE-PMRT CAPABILITY FOR
EVALUATING TACTICAL SOFTWARE

Marjorie Kirchoff
Intermetrics, Inc.
5392 Bolsa Ave.
Huntington Beach, CA 92649
(714) 891-4631

Victor Vajo
Harold Lowery
Warner Robins Air Logistics Command
Robins, Air Force Base, GA 31098
(912) 926-5921

Marjorie Kirchoff is Manager of the Emulation Technology Group and Synthetic Environment Laboratory at Intermetrics, Inc., Huntington Beach, CA. She is currently Project Manager of the QM-1/VAX-11/780 integration contract which will provide the foundation for the Automated Generic Testing Facility at Warner Robins Air Logistics Command. Prior to her affiliation with Intermetrics, Ms. Kirchoff was a member of the Emulation Group at McDonnell Douglas Astronautics Company where she was Principal Investigator for the CAMEO project which was designed to control concurrent multiple emulations on the QM-1 computer. She has a total of 23 years experience in computing, mostly in software testing and tool development. She has a BSSE from the University of Wisconsin.

Victor S. Vajo received the B.S. degree in electrical engineering and the M.S. degree in electrical engineering from New Jersey Institute of Technology in 1963 and 1965, respectively. From 1965 to 1977 he was involved in several investigative studies for the Avionics Laboratory at the Army Research and Development Laboratories, Fort Monmouth, NJ. These studies included man-machine simulations of aircraft and flight control systems for helicopters in a rigid rotor stabilization investigation, development of simulations for terrain following and obstacle avoidance, and Project Manager for developing computer generated terrain contour maps for cockpit TV display to all pilots in low altitude night navigation. For the past five years he has been employed by the Air Force Logistics Command, Robins AFB, GA and is currently Lead Engineer on a project to develop an automated generic testing facility to be used for independent verification and validation.

Harold Lowery is an electronics engineer at the Warner Robins Air Logistics Center, Robins Air Force Base, GA. His current work is with independent verification and validation of operational flight programs for embedded avionics computers. Specifically, this includes development of emulations and of software testing tools for FORTRAN, JOVIAL J73, and assembly language. Mr. Lowery received his BSPCS (physics) and MS (agricultural engineering) from the University of Georgia, where his research involved instrumentation and field testing of forest fire fighting plows.

ABSTRACT

A generic software testing facility is presently under development at Warner-Robins Air Logistics Command. The Multiple-System OFF Support (MSOS) System will allow independent verification and validation of avionic software for a variety of systems to be conducted early in the development cycle, reducing costs to the Air Force.

Intermetrics, Inc. is linking via hardware and associated software the Nanodata QM-1 microprogrammable computer and the VAX-11/780. The QM-1 hosts emulations of tactical embedded computers and the VAX hosts simulations of real environments. Overlaying the emulation/simulation system is a UNIX-based monitor tailored to provide absolute control and complete visibility into the executing target machine software.

A variety of static test tools for analyzing JOVIAL and, eventually, Ada code is being hosted on the VAX. A primary function of these tools will be to verify the conformance of the code to the specific standards.

INTRODUCTION

The Air Force is faced with an increasing proliferation of unique avionics and weapons systems, causing associated increased costs of acquisition, operation, maintenance and continuing modification. Standardization is a strategy being developed by the Air Force to reduce the uniqueness of major system components, such as computer architectures, languages, multiplex bus, controls and displays. An additional step is to standardize the tools used to develop and maintain these systems.

The Warner-Robins Air Logistics Center (WR-ALC) at Robins Air Force Base, Georgia, is responsible for the support of digital avionics and weapons systems containing embedded computer systems.

This support normally involves maintaining and updating operational systems after Program Management Responsibility Transfer (PMRT) from the developing agency to the maintaining agency. However, pre-PMRT support is also required for the software verification and validation during the acquisition phase.

In order to support the pre-PMRT activity, WR-ALC must provide a technical base of test engineering expertise and facilities. To this end, a generic capability for software verification and validation is being developed. This capability, called the Multiple System OFF Support (MSOS) System, will provide a generic testing capability by the creation of a software test bench upon which software for a variety of systems can be rigorously and thoroughly tested.

WR-ALC's goal is to prevent the loss of readiness capability due to software errors in fielded weapons systems by removing the errors before the systems reach the field. The primary objective of MSOS is to provide the means for finding those errors during the development phase.

MSOS

MSOS consists of both hardware and software, configured to give test engineers the ability to perform end-to-end testing on software under development. It will contain a battery of tools from which the test engineer may select a set appropriate to the testing requirements at hand. They range from relatively simple tools, such as code auditors, to complex tools, such as an emulation/simulation system.

The proposed capabilities include the following: compilation/assembly, software code analysis, automatic software testing, software testing history, data reduction analysis, analysis report generation, verification and validation testing, on-line simulation, emulation system modeling, automatic data base generation, and system familiarization.

The key element normally used to support an operational system is an Avionics Integration Support Facility (AISF). The AISF enables the support organization to develop and evaluate changes/modifications to OFF's and perform IV&V. The AISF consists of the actual computers and/or digital processors embedded in the avionics, interacting through a Computer Monitor and Control (CMAC) unit. A dynamic system simulation of the aircraft in a tactical environment and the drives for the system's interfaces, controls, and associated displays are generated by a simulation host processor. However, during the early acquisition phase, the embedded computer is usually not available. MSOS provides the solution by creating a synthetic environment, using a software emulation of the embedded computer to execute the OFF which is driven by data generated by an environment simulation. This emulation/simulation system is controlled by a monitor that provides absolute control and visibility into the executing software.

By using virtual embedded computers instead of real ones, a common set of support hardware and software can be used to test many systems.

Hardware Configuration

The hardware consists of a Nanodata QM-1 microprogrammable computer¹, a Digital Equipment Corporation VAX-11/780, and an electronic interface linking the two machines. Each computer can operate in either stand-alone or communicating mode. Each computer is supported by a full set of peripherals, including tape and disk drives, CRT's and line printers. The QM-1 is a microprogrammable computer designed specifically to host emulations of digital devices. It has three levels of memory, an 18-bit 750 ns mainstore (core), an 18-bit 80 ns control store, and a 360-bit 80 ns nanostore (both semi-conductor). The WR-ALC

configuration consists of 256K words of main-store, 20K words of controlstore, and 512 words of nanostore. It has 32 18-bit registers for local store, 32 18-bit registers for external control, and 32 6-bit F-store registers which control the independent bus connections between the local store registers and the CPU, ALU and shifters. The QM-1 features both 36 and 32-bit shifters and operators in either a 16 or 18-bit mode. The rotate, mask and index (RMI) unit is a programmable device which extracts and justifies up to three fields of a word (usually a target software instruction) as it is fetched from mainstore, an invaluable timesaving aid in instruction cracking for the emulator.

The high speed data transfer hardware consists of two links between the QM-1 and the VAX. One link is a programmable I/O device which transfers data between the control store in the QM-1 and VAX memory. It effects a pseudo-DMA transfer via a special microinstruction executing in nanostore that reads or writes a block of data into or out of a QM-1 control store buffer one word at a time until the transfer is complete. The other link is a DMA device which transfers a block of data between QM-1 mainstore and VAX memory in true DMA fashion. A low-speed link (9600 baud) link allows the QM-1 to appear as a terminal to the VAX. The links are built and installed by Hale Associates Research Corp. (HARC)².

Support Software Configuration

The support software for MSOS is based on the VAX operating system, VMS, and its utilities, and a QM-1 operating system, UNIX, Version 7³, coupled with Nanodata's micro-operating system, TCP2⁴. Intermetrics, Inc. has modified and enhanced UNIX to specifically interact with emulation systems on the QM-1; has developed a microcoded emulator control system for efficient management of emulator activities; and has developed a generic environmental simulation interface which manages the data transfers between emulators on the QM-1 and simulators on the VAX, making the transfer mechanics transparent to the emulator and simulator writer.

This software complex is the foundation for the basic MSOS capability. To it are added all of the actual tools that are used by the test engineer, and any tools used to generate those test tools.

Emulation/Simulation Software Configuration

The software necessary for executing target processor code is:

1. An emulation of the target processor.
2. An emulation interface which is application dependent.
3. A simulation of the environment in which the target software operates in real configurations.

Emulators can be coded in SMITE⁵, a high-level machine description language, in MULTI⁶, a vertical microcode instruction set which resides in control store or coded directly in nano assembly language, a highly horizontal microcode instruction set (which also implements MULTI). Emulators can be SMITE or MULTI first, followed by a migration into nanocode of heavily used and/or time-consuming functions. The emulator is a faithful reproduction in software of the instruction set, interrupt system, memory accessing and management system, registers, stacks, and other architecture features of the target processor. The target memory is emulated in the QM-1 main memory. Each target instruction is fetched from mainstore and executed by the emulator in the fast memory of control store or in nanostore.

The emulator interface provides communication between the emulation and the simulation, and between the emulation and the human user. Its primary tasks are to:

1. Manage the simulation data, moving data into or out of I/O locations to simulate bus or port I/O.
2. Manage the debugging and monitoring processes specified by the user as he tests the target software.

Environment simulators are hosted on the VAX 11/780, and generate environment data as required by the OFP.

A generic environmental simulation interface (ESI) resides on either side of the drivers, part on the VAX and part on the QM-1.

Most I/O operations involve the simulator and are handled by the emulator interface. It is necessary for the simulator to know the exact data requirements of the software. If the code is cyclic in nature, the I/O transfers can usually be pre-defined with both the simulator and emulator knowing the data requirements for each transfer a priori. However, if data requirements are asynchronous, then the request must identify the data needed, and the simulator must be able to decode the request and generate the required data. The programmable I/O link is used for transferring requests for simulation data to the VAX and for transferring data back to an emulator-controlled buffer in the QM-1 control store for distribution to the target memory as appropriate. If the simulation data is destined for contiguous locations in target memory, the transfer from the VAX may be made using the DMA link to the target memory locations in QM-1 mainstore.

An emulation system, once built, is stored on the QM-1 disk. Any number of scenarios can be generated off-line on the GPC and stored on the GPC disk for later input to the on-line simulation.

As stand-alone systems, the two computers are independently controlled. However, when an integrated emulation/simulation system is to be run, control is delegated to the QM-1. The user loads both computers with the desired software configuration via a command through the

QM-1 console. Control remains there until the run is terminated and the GPC detached (logically). The user has complete control of the emulation process via an extensive command set that can start, stop, continue, single step and terminate execution of the target software. He may request any of the debugging aids which include sequential traces, boundary traces, snap dumps and breakpoints. He may examine, modify, dump to the printer, or display on the CRT the contents of any target memory locations registers or stacks.

Of particular value is the checkpoint and restore capability which allows the user to save on disk the entire state of the emulation/ simulation system at any point he chooses, then restore it at any later time and continue as if it had not been interrupted. This allows a segment of code to be checked out and a checkpoint taken. Downstream code segments can then be tested without rerunning the checked out segments by starting with a restore of the checkpointed state. It is also helpful in checking multiple paths through the code from a single branch point.

Data captured by the emulation system during a test can be output to tape for later reduction, or can be sent across the link to a post processor on the VAX to be reduced on-line, with the results saved and/or displayed on a CRT, printer, or plotter while the test is still in progress.

When a test has been completed, the emulation/simulation system is terminated by command from the QM-1 console, and all termination processes for both machines are performed.

At this time, a new test case involving the same emulator but different simulation scenario, or involving both a different emulator and simulator may be loaded, a process taking only a few seconds.

Tools

Software test tools are often divided into two categories: static and dynamic. Static test tools analyze the source code of a program; thus the program is never executed. Such tools as code auditors, data flow and interface analyzers fall into this category. Dynamic test tools actually execute the code to be tested, usually after adding software probes to "instrument" the program being tested. Path-flow tracers, assertion checkers, etc., are tools belonging to this category.

Software Research Associates is investigating what tools are already available and what will have to be developed for testing software for embedded computer systems. Early indications are that there are relatively few tools applicable to Air Force systems. A major reason for this is the move to standardize to JOVIAL, and eventually Ada. These languages have not received as much test tool development effort as more widely-used, established languages, such as FORTRAN.

Among the tools available are a JOVIAL code auditor and J73AVS (JOVIAL J73 Automated Verification System). The code auditor is a static test tool that checks for compliance with established programming practices ("precepts") and warns of possibly dangerous code. J73AVS is a tool for both static and dynamic testing of JOVIAL J73. It not only provides static analysis but also allows for such functions as assertion checking, variable tracing, and execution coverage.

Although test tools for languages used by the Air Force are limited at present, it is expected that as JOVIAL J73 becomes more widely used and with the carefully planned transition to Ada, sufficient tools will be developed. When completed the SRA report will provide direction for rehosting and developing tools on the VAX.

Another type of tool is the SMITE compiler which generates emulators for the QM-1. The time and effort required to develop the microcode (and possibly nanocode, as well) to emulate a given processor threatens to be prohibitive. To try to reduce this time and effort and to improve the maintainability of emulations once they are written, TRW is under contract to rehost to the VAX a compiler for SMITE, a high-order Pascal-like, hardware description language. SMITE compiles to microcode for the QM-1 and has a number of features which facilitate emulation development. Eventually, a library of emulations for various processors will exist, raising the possibility that standard processors could be "off-the-shelf" packages and used for several systems. SMITE routines to emulate typical processor functions would be available to further reduce the need to write each emulation from scratch.

To go with this, tools are needed for the development of simulations of the target processor's environment. The choices in this area are less clear. A wide variety of languages is available for complex modeling. These range from such general purpose languages as FORTRAN and Pascal to specialized simulation languages such as GPSS. Each of these has certain advantages and at this point it is too early to say if any one language is clearly better suited to the need. The most likely candidate for a standard simulation language may be Ada. Not only is it a state-of-the-art language with several features useful for simulation purposes, but standardization within the DoD community will yield obvious benefits.

As relevant tools of all kinds become available, they will be added to the MSOS repertoire.

The Software Test Management System (STMS)

STMS is a system, whose requirements are presently being defined, that will provide the user with a simple means to install the software to be tested, choose and execute various test tools, generate meaningful reports of the results, and maintain test configuration control over various versions of the software. The intent is that the user

will be able to test his software in a user-friendly environment without having to learn all the details of each tool or method.

SUMMARY

To assure the maintaining agency has a thorough knowledge of the delivered system and associated support system equipment, experienced embedded computer engineering and technical support is required prior to Program Management Responsibility Transfer to the ALC. Since even a minimum complement of software support equipment is normally not available before PMRT, an alternative approach was required. To meet this requirement, WR-ALC has undertaken to establish the MSOS capability for diagnostic emulation and generic software testing of Embedded Computer Systems.

MSOS will provide the capability to perform testing that will support not only independent assessment of contractor developed software by Independent Verification and Validation (IV&V) techniques, but engineering and logistics trade-off studies, evaluation of support concepts and technical evaluation of software techniques.

REFERENCES

1. Nanodata, "The QM-1 System: Design, Software and Applications," Nanodata Computer Corporation, Buffalo, New York, 1979.
2. Hale, J., "QM-1/VAX-11/780 Integration Hardware Configuration Item Development Specification," Hale Associates Research Corp., Stony Brook, New York, 1982.
3. Bell Laboratories, Inc., "UNIX Time-Sharing System: UNIX Programmer's Manual," Bell Laboratories, Inc., Murray Hill, New Jersey.
4. Nanodata, "Task Control Program - TCP 2.0," Nanodata Computer Corporation, Buffalo, New York, 1979.
5. TRW, "Advanced SMITE Compiler/SASS Interface Control Document," TRW Defense and Space Systems Group, Redondo Beach, California, 1978.
6. Nanodata, "MULTI Micromachine Description," Nanodata Computer Corporation, Buffalo, New York, 1976.

APPLICATION OF JOVIAL (J-73) TO DIGITAL FLIGHT CONTROLS

J. H. Robb
General Dynamics/Data Systems Division/Central Center
P. O. Box 748, MZ 5940
Fort Worth, TX 76101
(817) 732-4811, ext. 5681

P. J. Boatman
General Dynamics/Data Systems Division/Central Center
P. O. Box 748, MZ 5940
Fort Worth, TX 76101
(817) 732-4811, ext. 5645

P. H. Lang
General Dynamics/Fort Worth Division)
P. O. Box 748, MZ 2834
Fort Worth, TX 76101
(817) 732-4811, ext. 2830

ABSTRACT

This paper presents a brief outline of the history of the General Dynamics/Fort Worth Division Production Digital Flight Controls Program. Two flight demonstration programs will be discussed: the single channel digital F-16 and the four channel digital F-16. These were the first flights ever of a microprocessor-based digital fly-by-wire system in a relaxed static stability aircraft. Both demo programs use a pseudo Higher Order Language (HOL) tailored for flight control system applications. This paper describes the techniques used in applying the JOVIAL J-73 HOL to Digital Flight Control System Operational Flight Program (OFP) mechanization. Results are presented from testing the OFP on flight hardware. Included in this paper are JOVIAL J-73 language issues such as subroutine implementation and their associated efficiencies, the test set, and testing methods.

Copyright © 1982 by General Dynamics Corporation
All rights reserved

TEXT

With the advancement of computers in this technological era, General Dynamics realizes that the advantages of digital flight control systems over conventional analog systems are manifold. These advantages include: increased flexibility; improved lifecycle costs; higher accuracy; ease of maintenance; and reduced size, weight, and power consumption. Before the first flight of an F-16 aircraft using a digital flight control system could be made, one critical question had to be answered. Could a digital computer control an unstable aircraft such as the F-16 over the entire flight/store envelope? The F-16 achieves its ranking as the world's most maneuverable fighter by incorporating the principles of relaxed static stability. Being a relaxed stability aircraft, the F-16 could not tolerate a single sustained failure in its flight control system without causing the aircraft to radically depart from a stable flight path. The approach taken, therefore, has been one of low risk, that is, the most conservative approach in getting digital flight controls on the F-16.

In conjunction with Lear Siegler Incorporated, a digital emulation of the production F-16 analog flight control system was developed based on a 4MHZ Z8002 microprocessor. The OFP for this digital channel was developed by General Dynamics and Lear Siegler Incorporated. This baseline OFP was funded by both companies for a demonstration of digital flight control systems on the F-16 aircraft. The baseline OFP was coded using a pseudo Higher Order Language, developed by Lear Siegler Incorporated, for flight control applications and was tested, during and after development, on the Digital Development System (DDS) also developed by Lear Siegler Incorporated. The DDS is comprised of an interactive program hosted on a PDP-11/34 which is linked to the Z8002 Flight Control Computer (FLCC). The DDS provided a means of testing the Operational Flight Program (OFP) in execution on actual flight hardware, allowing the Z8002 to execute the OFP and halt it at any point to monitor data flow and insure program validity. The DDS has an extensive menu of commands available that make it a very powerful tool. Up to four separate digital channels can be monitored, breakpoints set, and memory contents examined and changed. Dynamic command files can be created, and a programming language is provided that allows additional features to be added to the DDS test software. A manual test set, previously used with F-16 analog production systems, was used to insure that the digital system gave the same

commands to the aircraft control surfaces as that of the production analog system. After development, the single channel Z8002 completed an extensive testing phase which consisted of verification and validation (V & V), Safety Of Flight (SOF), and qualification and flight certification by automatic testing. A modified production analog flight control computer, with one of four of the analog channels replaced by the digital Z8002 channel, was flown at Edwards Air Force Base on an F-16, the first flight was October 20, 1981. The single channel had 99 flights with no software errors. Satisfied with the performance of the single channel, four digital channels identical to the previous single channel, were configured to provide a Quad digital flight control system. The Quad system was configured with an automatic engaging, manual override Independent Back-up Unit (IBU) for additional safety, although it has never been needed. The first flight of the Quad system was May 11, 1982. The Quad system at the writing of this paper has had 69 flights, as of October 14, 1982, with absolutely no software problems. These two successful programs mark the first digital flight control system on a fly-by-wire, relaxed static stability aircraft with no mechanical back-up, and the first application of a microprocessor technology to flight controls. The handling qualities of the F-16 aircraft with the digital flight control system has been found to be identical to the production analog system.

The advantages of a HOL became obvious during the development of the previously mentioned digital system. The ease of developing the OFP, the lower maintenance costs, high portability and other factors generated a great deal of interest in using a powerful HOL for flight controls. In compliance with MIL-STD's 1750A and 1589A it was decided to investigate the use of a J-73 targeted to a 1750A instruction set. Working in concert with other programs using J-73 at General Dynamics, most notably the F-16 MSIP (Multi-national Staged Improvement Plan) avionics program, a J-73 OFP has been developed for F-16 digital flight controls, the Production Digital Flight Control System (PDFCS) J-73 OFP.

The PDFCS J-73 OFP consists of 113 well structured-modules which are designed in such a way that any change in the OFP would minimize the needed coding change. To increase portability, the OFP almost entirely uses J-73. Of interest is that the executive is completely coded in J-73, without loss of throughput. This is the first application of an executive, using only HOL, in an F-16 OFP. This executive is as efficient as an executive employing assembly

language and obviously more portable. Also of interest is that all mathematical operations in the PDFCS OFP utilize a floating point representation. On analysis of the control laws, it was felt that floating point representation would have several advantages over a fixed point representation, namely:

- (i) Faster and easier development,
- (ii) Greater flexibility,
- (iii) More meaningful test data, and
- (v) Improved maintainability

Scaling problems, inherent in control laws, are minimized using a floating point representation, making development much easier. If additional flight control modes should be added, a fixed point implementation would require extensive scaling to incorporate these new modes; additional scaling would be minimal with a floating point system. Test data using floating point representation would be expressed in meaningful values, whereas test data using a fixed point representation usually requires a number of scaling factors to make the data comprehensible. Code written in floating point is more understandable than that written in fixed point and is therefore easier to maintain. The governing factor in using floating point, however, is throughput. Findings indicate that with a 1750A architecture and the instruction mix the OFP uses, a floating point representation would only decrease throughput by some 8 to 10 percent. Considering the previously mentioned advantages, a floating point representation is well worth this minor decrease in throughput.

The J-73 OFP is currently targeted to two processors, the Z8002, an interim processor, and a 1750A architecture processor. The OFP is developed using the Harris H800 based Software Engineering System (SES) (see figure 1). The SES is an on-site computer system for developing high quality embedded software products. Once each module is coded, it is sent via a data link to the IBM 370 and compiled using either the DIS (Digital Integrated Subsystems - Z8002 target) or the SEA (Software Engineering Associates - 1750A target) compilers. If the module is DIS compiled, the source listing and an assembly listing are then sent via the same data link back to the SES where it is assembled and linked. This load module can be sent to a PDP-11/34 by

either generating a tape and loading it or via a data link. From the PDP-11/34 it is down-loaded to the Z8002 flight hardware and tested using the latest version of the DDS interactive program. For a SEA compiled module, the source listing is sent from the IBM 370 via a data link back to the SES. The relocatable object code resides in a file on the IBM 370. This object code file is then sent via another data link to the VAX 11/780 where it is linked and tested using the McDonnell Douglas 1750A load module or a Delco 1750A bit slice processor. Finally, resident on the VAX, there is a DDS emulator for the Z8002 processor, and in the near future an emulator for a 1750A architecture.

There are two basic philosophies in testing software: a top-down approach, and a bottom-up approach. The top-down approach first tests the modules that are the top of the hierarchical scheme and works downward until the lowest are tested. The bottom-up approach first tests the lowest modules in the hierarchical scheme and works upward until the highest are tested. The PDFCS testing approach taken is generally a top-down approach. This tends to keep changes from restructuring the OFP as a bottom-up approach tends to do. However, many of the modules that are lower on the hierarchical scheme are tested in parallel with the top-down testing. This allows for full utilization of personnel and aids the top-down testing effort. These modules are tested in isolation using every possible combination of the input parameters. If the module's size is small, the output is simply checked by hand; otherwise, it is compared with a valid model for accuracy. The person who codes a module is responsible for testing that same module. This tends to expedite the testing phase as well as boost morale.

As tool sets for both the Z8002 and the 1750A are becoming more powerful, a method to incorporate them in some dynamic testing setup is being devised. For the short term, data sets will be created to simulate actual flights to allow testing of the OFP. Long term plans are to get data from test flights and incorporate this in a dynamic test set-up. With the advancements being made on the tool sets we now use, this process should not be very difficult and could even lead to a development system for future programs.

The following issues are those of the J-73 language that affect the PDFCS OFP and that have been resolved by testing or some other manner. The first issue concerns procedure declaration. There are three ways to declare a procedure in J-73:

- (i) Define declaration (the same as a macro),
- (ii) A procedure declared inline, or a
- (iii) Normally declared procedure

The define declaration is of either global or local scope and the generated assembly code is inserted inline. This results in savings since the procedure setup is not needed, which in some cases takes longer than the actual executable code. However, more assembly code is introduced since it is generated inline. The define is very awkward in appearance and does not seem to resemble a HOL feature. A procedure declared inline is locally defined and the generated assembly code is inserted inline. Again, the overhead of procedure setup is eliminated, although code expansion occurs. The inline procedure is identical in appearance to a regular procedure, the only difference being the INLINE instructive. Lastly, the normally declared procedure can be defined either globally or locally, but requires the necessary overhead for set-up. A normally declared procedure is easier to test and validate due to its modularity. Define or inline declared procedures require testing every time they are invoked and therefore require quite a bit more testing. The setup, or invocation time, for each normally declared procedure takes 25 usecs for the 1750A. Considering that the PDFCS OFP invokes many procedures, a large amount of time is spent doing procedure setup. As a general heuristic, an inline procedure is used when the time required to set up a procedure is greater than the executable code it contains, although code expansion must be considered.

Another J-73 issue affecting the PDFCS OFP is bit manipulation. For processors that have single bit manipulation instructions, for example, a set bit instruction, the most efficient way of manipulating bits, appears to be through the use of the POS declaration, which assigns a name to a particular bit or bits in a word, and subsequently referencing the bit by its assigned name. For single bit manipulation, this method generates one assembly instruction. For bit strings, more assembly instructions are generated but they seem to be less than any other method. Since both the 28002 and 1750A architecture processors have single bit manipulation instructions, this method is used.

The last issue concerns case statements. When making multiple decisions on a variable, either a nested 'if' or a case statement can be employed. The nested 'if' is generally employed when the probability of a single condition is much greater than the rest. This condition is then tested first. Since the probability of this first condition is much greater than the rest, this method should be more efficient, on the average, than a case statement due to the case statement's additional instructions to perform the computation for the jump table. If the probabilities are not known or are fairly similar, then a case statement is generally employed; a case statement makes its decision based on a jump table and, on the average, should be the most efficient under these circumstances.

The PDFCS J-73 OFP is under analysis and testing to provide the most efficient OFP possible. Depending upon the semiconductor family the 1750A chip set utilizes, the OFP should have a throughput between 40 and 60 percent. The memory requirements should be about 12K, depending upon how many procedures are coded using the INLINE declaration. There is substantial room for growth for additional modes or systems to be added to the OFP. Two demonstration programs are planned for the PDFCS OFP. The first demonstration program, in June of 1983, will be flight testing of a single channel of the OFP with three channels of the baseline OFP. Once this flight testing is completed, a Quad system using the PDFCS OFP will begin flight testing in the first quarter of 1984.

CONCLUSIONS

Digital flight controls have a perfect record on the F-16 aircraft in a production environment with no software problems. JOVIAL J-73 can be applied to digital flight controls in a production environment with substantial room for growth.

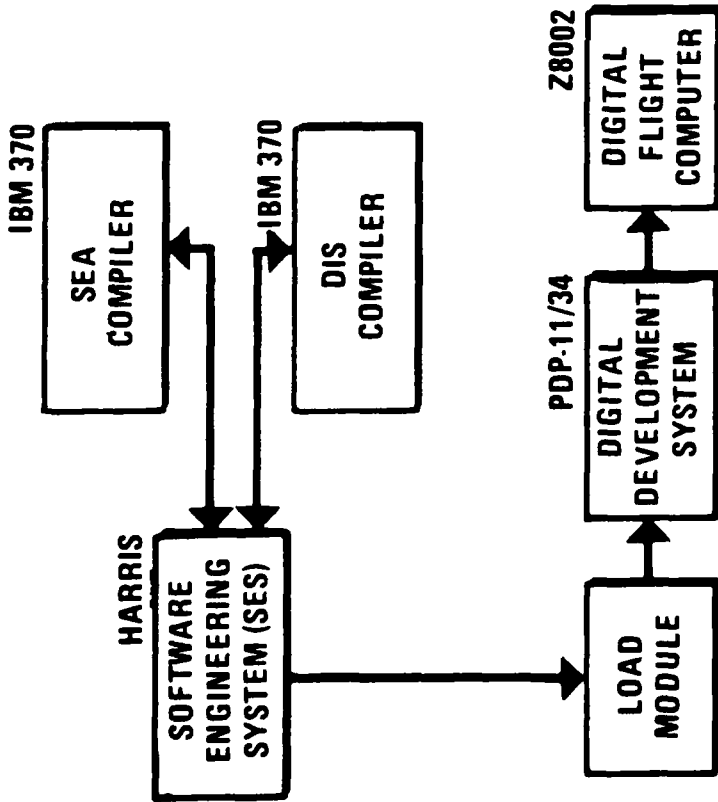
AUTHORS

J. H. Robb is one of the lead engineers in charge of implementing the Digital Flight Controls OFP in MOE at General Dynamics/ Fort Worth Division.

P. J. Boatman - Production Digital Flight Controls

P. H. Lang - Production Digital Flight Controls

OFFP GENERATION & TESTING



1750A SIMULATOR & EMULATOR TESTING

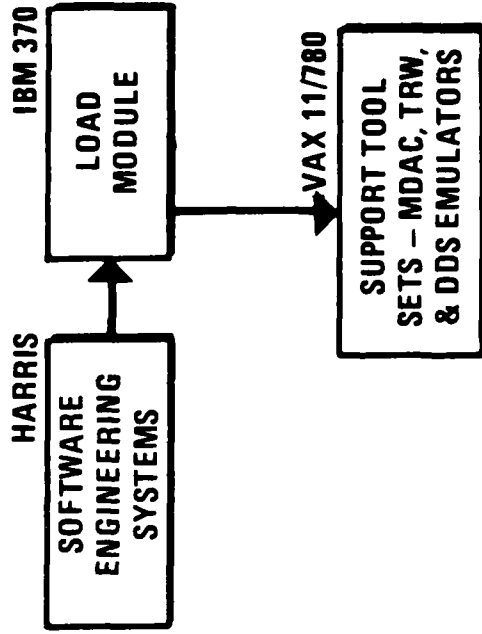


Figure 1 Current OFFP Development Methods

MIL-STD-1760

STANDARD STORE INTERFACE

SESSION CHAIRMAN: Claude Connell
AD/DLJA
Eglin AFB

MODERATOR: Major Lewellen Dougherty
HQ USAF/RDPV

AD-P003 526

MIL-STD-1760 DEVELOPMENT PROGRAM

Claude M. Connell

Air Force Armament Laboratory

Mun/Acft Interface Branch

Eglin AFB, Florida

(904) 882-2201

ABSTRACT

>The joint Air Force/Navy program to develop MIL-STD-1760, Aircraft and Store Electrical Interconnection System, is described. The rationale for the program and the program approach and status is presented. Each of the three interface elements of MIL-STD-1760 are discussed with emphasis on how the elements enhance aircraft/store interoperability and reduce future aircraft modification cost. Implementation of MIL-STD-1760 is advocated for all future US and NATO aircraft and stores.

BIOGRAPHICAL SKETCH OF AUTHOR

Mr. Connell is the Air Force Program Manager for the MIL-STD-1760 development effort. For the last ten years, he has been employed at the Air Force Armament Laboratory and has been actively involved in all phases of aircraft/store electrical integration and stores management system development. Mr. Connell obtained his BSEE from Mississippi State University in 1972 and his MSEE from the University of Florida in 1978.

THE MIL-STD-1760 DEVELOPMENT PROGRAM

Mr. Claude M. Connell and Mr. Bryce M. Sundstrom

Air Force Armament Laboratory
Mun/Acft Interface Branch
Eglin AFB, Florida
(904) 882-2201

INTRODUCTION.

MIL-STD-1760, Aircraft/Store Electrical Interconnection System, is being developed under the joint Air Force/Navy Aircraft Armament Interoperable Interface (A²I²) program. Primary program objectives are to promote interoperability between future aircraft and stores and to significantly reduce the modification cost of adding a new weapon to existing aircraft. This is being approached through interface standardization.

STATEMENT OF PROBLEM.

Interoperability between aircraft and stores is presently precluded by a set of obstructions. Within this set, a primary obstruction is the nonstandard aircraft-to-store and store-to-aircraft electrical interface. Interfaces between aircraft and stores are becoming increasingly sophisticated and complex. At the same time, there is an increasing desire on the part of the Department of Defense to increase service and allied nation interoperability between aircraft and stores.

The number of different types of stores is large (more than 100) and continues to grow as a result of development and acquisition programs. Stores include conventional general purpose bombs, guided bomb dispensers, missiles (air-to-air and air-to-ground), nuclear weapons, sensor pods, dropped sensors, camera pods, countermeasure pods, fuel tanks, dispensers, guns, rockets, etc. Interfaces between aircraft and stores are only partially guided by standards and, therefore, have tended to evolve into system peculiar mechanical adapters/connectors, electronic signals, power connections, and other armament assemblies which make interoperability impossible without major modifications to aircraft and/or stores on a case-by-case basis. The trend toward more complex store functions which require increasing amounts of avionics data from aircraft systems is causing the problem to become increasingly acute. Examples of this situation are AMRAAM, HARPOON, PHOENIX, HELLFIRE, ATLAS POD, ALCM, etc.

On the aircraft side of the interface, stores management systems are unique to each aircraft type and sometimes each model. Old aircraft Stores Management Systems (SMS) are generally hardwired, not integrated, not

automated, and reflect outmoded, obsolescent electronics and electronic design. Although new aircraft SMS designs reflect current technologies in electronics and communications, they are still tailored to a specific store list and are not designed for growth. Invariably, the changing stores list requires modifications almost as soon as the aircraft begins its operational life. The adoption of acquisition methods which result in aircraft systems which are tailored to handle specified lists of stores has limited weapon system capability, growth, and flexibility. These methods yield weapon systems which are well defined within themselves, but are inflexible and costly to modify.

Concept of a Solution.

The intent behind developing MIL-STD-1760 is to support achievement of interoperability between independently designed stores and aircraft by imposing specific interface design requirements applicable to each. To accomplish this, the interface characteristics of the aircraft and of the stores must be controlled so that each unit of a given kind, e.g., a carriage store, is functionally interchangeable with any other unit of the same kind.

The overall goal of the standard is to remove the non-standard electrical interface as an obstruction to interoperability. Application of the standard will result in a wide range of stores being interoperable with a wide range of aircraft. Modification of aircraft and store hardware to allow individual combinations to operate together will be minimized. The use of adapter modules will be discouraged. In this way, the effort and cost necessary to integrate aircraft and stores will also be minimized.

MIL-STD-1760 is designed to be flexible enough to accommodate individual system peculiarities. In particular, implementation may change with technology advances as long as the interface characteristics are maintained. The MIL-STD addresses only the electrical interface between aircraft and stores. Compatibility parameters such as size, weight, aerodynamics, avionics capabilities, etc., must be satisfied in addition to the electrical interface in order to realize interoperability. The electrical, or MIL-STD-1760, portion of the aircraft/store integration effort will ultimately be limited to developing software modifications necessary to accommodate new stores.

The Major Elements of MIL-STD-1760.

To achieve the program objectives, it was decided that the Aircraft/Store Electrical Interconnection System (MIL-STD-1760) would consist of three hierarchical elements: electrical, physical, and logical. Each element is described below:

a. Electrical: The electrical element quantitatively specifies the signal set the aircraft must provide and that the store must utilize. The signal set for the Aircraft Station Interface was published in July of 1981 and is described in detail in the following paper entitled "Signal Set Standardization for the Aircraft/ Store Electrical Interconnection System" by D. E. Lautner and J. R. Perkins.

b. Physical: The physical element of the standard defines the intermateability characteristics of a set of armament connectors. It is envisioned that the characteristics of the following three classes of connectors will be specified:

. An umbilical connector for gravity release stores employing the MIL-STD-1760 signal set.

. A low cost connector for simple stores employing a limited subset of the 1760 signal set.

. A blind mating connector for rail launched stores employing the 1760 signal set.

To achieve the goal of interoperability, it is not necessary to completely describe the interconnection component as one would, for example, by calling out a particular part number. The physical element of the standard must define only those characteristics essential to intermateability. Essentially this means that a particular set of physical dimensions has to be defined. The method of achieving this definition for gravity release and most eject launch stores was to select a set from an existing state of the art connector. Several manufacturers are currently designing similar connectors for MIL-STD-1760 employment under the constraint that each must employ the selected set of intermateability dimensions. The problem of intermateability also includes defining the connector insert physical and functional layouts, particular contacts, crimping tools, and etc. In all, some ten or twelve piece-part specifications are required to completely define a connector as a functionally intermateable system. Most of these have been developed, coordinated, and published for the lanyard release or so called umbilical connector for gravity release weapons.

The umbilical connector described above is intended for relatively sophisticated weapons and as such is complex. There is an effort under the SAE AE-9 Aerospace Avionics Equipment and Integration committee to define a signal set for simple low cost stores (SLCS). To date, a strawman configuration employing only a single channel MIL-STD-1553 data link, 28 volt dc power, addressing lines, and associated ground returns, has been proposed for review. The development of the SLCS connector definition will follow closely the process employed for the complex umbilical connector. The major difference will probably be in the method of selecting the intermateability aspects. For the complex connector, a decision was made to limit competition to a small number of existing devices. It is anticipated that for the SLC connector that manufacturers will respond to a request for proposal with innovative approaches; such as employment of composite materials, to hold costs down. As such, there will be no attempt to limit competition.

Rail-launched weapons pose particular interconnection problems such as the necessity for blind mating. There is also the problem of rocket or jet blast burning of connector contacts. Because of these considerations and others, the definition of the physical interface for rail-launched stores was deferred to following that for gravity release weapons. In the interim, a fortuitous development was taking place.

Initiation of the A²I² and Advanced Medium Range Air-to-Air Missile programs and their respective progress were sufficiently concurrent to hypothesize that the connector selected for AMRAAM could become a defacto standard for MIL-STD-1760. It was recognized that the interface requirements specified for the AMRAAM program were going to impose very difficult and complex interconnection problems. Since the AMRAAM launcher must meet certain interface requirements unique to each of the F-14, -15, -16, and -18 aircraft, internal space allocation for the connector and its release mechanism was critical. To the credit of the designers, it appears that multiple interfacing will be achieved. The method of coupling the missile receptacle to the launcher connector appears to be readily adaptable to other rail-launched weapons. That possibility in itself drives the AMRAAM connector towards a standard device.

It would have been desirable to undergo a long-term systematic development program for these three classes of connectors. However, the requirement is for interoperability now. The approach MIL-STD-1760 has taken is to select and standardize on the best which is available or can be made available in the near term.

c. Logical: The Logical element of MIL-STD-1760 is primarily concerned with the utilization of the MIL-STD-1553 multiplex data bus. Although this multiplex standard defines word types and protocols for general types of data transfers, further definition would be helpful to optimally apply MIL-STD-1553 in the aircraft/store environment.

It is envisioned that the MIL-STD-1760 logical element will be comprised of two primary areas; Standard Data Words and Aircraft/Store Protocols. Standard Data Words are MIL-STD-1553 data words which have been assigned specific bit patterns to represent particular functions, commands, or values. As such, they provide the same information to all users. If data words are not standardized, implementers will by necessity derive their own. Unique words, in turn, complicate aircraft or store interpretive hardware and software. The A²I² program is being closely coordinated with the SAE-AE9 Data Word Standardization Task Group effort towards solving this problem. The Aircraft/ Store Protocol area provides a definition of rules to transfer data between aircraft and stores. Additional protocols are necessary in such areas as user application data, store addressing, message routing, block data transfer, message encoding, encryption, and fault handling. A draft of the Logical element will be distributed in early 1983 with the final version published in 1984.

SUMMARY

MIL-STD-1760 implements a new philosophy in aircraft/store electrical integration. No longer will aircraft be restricted to designs for unique sets of store requirements and, conversely, stores will not be constrained to interfacing with aircraft peculiar electrical configurations. Through MIL-STD-1760, aircraft will offer a standard electrical capability and stores will electrically integrate in a prescribed and orderly manner. Through MIL-STD-1760, interoperability can be enhanced and aircraft modification costs reduced.

MIL-STD-1760 IMPLEMENTATION STRATEGY

Frank T. Woodall

Teledyne Brown Engineering

133 Hospital Drive

Ft. Walton Beach, Florida

ABSTRACT

A common assumption is that the merits of a particular standard are sufficient justification for enthusiastic endorsement and implementation by effected program managers. In the case of MIL-STD-1760, however, many of the benefits of the standard are related to downstream aircraft/store interoperability and to reduction in the cost of new store integration efforts. The immediate impact is an increase in the technical complexity and, therefore, costs of current programs. As an example, aircraft may have to maintain both standard and non-standard interfaces for the foreseeable future.

The Air Force Armament Laboratory/DLJA is the Air Force Agent for development of MIL-STD-1760. DLJA is also sponsoring a study to identify and address implementation issues such as the aircraft requirement for dual interface capability. Other issues being addressed include:

- Funding
- Tri-Service management of the standard
- Rules for subsets
- Implementation targets
- Schedules
- Government support for the implementation process

The results of this effort will be an implementation strategy to be used as a planning aid or tool by individual program managers.

BIOGRAPHICAL SKETCH OF AUTHOR

Frank Woodall is a Senior Systems Analyst with Teledyne Brown Engineering. He holds BS and MS degrees in Electrical Engineering and has over 25 years professional experience with Teledyne Brown, Booz-Allen Applied Research and the U.S. Navy. His areas of expertise include avionics system development and test, software engineering, airborne intelligence systems, ballistic missile defense and command control.

Since 1978, Mr. Woodall has been closely associated with aircraft/store interface technology programs. He supported the development and subsequent flight testing of a Stores Management System (SMS) developed by the Armament Laboratory at Eglin AFB, Florida. He participated in the definition of the initial electrical signal set for MIL-STD-1760. He is currently under contract to the Air Force to develop an implementation strategy for MIL-STD-1760.

A paper describing the software verification effort for the Air Force SMS program was presented by Mr. Woodall at an IEEE conference on Software Testing.

AD-P003 527



AIRCRAFT-STORE ELECTRICAL INTERCONNECTION SYSTEM (AEIS)
FUNCTIONAL REQUIREMENTS

J.R. Perkins
D.E. Lautner
Vought Corporation
Dallas, Texas
(214) 266-5097

BIOGRAPHICAL SKETCH OF AUTHORS

Mr. James (Jim) Perkins, Vought Corporation, Dallas, Texas - Technical Project Manager for the Advanced Armament System and Advanced Electrical System technology programs at Vought. Mr. Perkins has been employed with LTV based companies for approximately 25 years and has been involved in development of aircraft, missiles, automatic controls for aerospace vehicles, geophysical exploration equipment, and military electronic equipment. Mr. Perkins obtained his BSEE from Texas Technological University in 1958 and MSEE from Southern Methodist University in 1962.

Mr. Don Lautner, Vought Corporation, Dallas, Texas - Engineering Specialist in Electrical Systems and Circuit Design. Mr. Lautner's primary responsibility is principal investigator for aircraft/store electrical compatibility analysis and development. Recent experience includes consultation assistance to the Air Force and Navy for MIL-STD-1760 development. Education: BSEE, University of Texas at Arlington.

ABSTRACT

Air Force and Navy are upgrading and improving all aspects of aircraft weapon systems. The armament system, which includes stores, carriage equipment and stores management elements, has been a major target for this upgrading. As a result, substantial improvements and new capabilities have been developed and placed into service. These improvements include development of highly sophisticated "smart" stores that employ or interface with complex sensors and guidance systems for target acquisition, lock-on and neutralization. However, these weapons and sensors were developed without consistent interfacing constraints and standardization requirements. Consequently, a very severe compatibility problem exists in adapting these new stores to all aircraft and in achieving interoperability between aircraft and stores in general. The causes of these problems are the lack of flexibility

and standardization of the armament system, subsystems and equipment. Air Force and Navy Laboratories are addressing these shortcomings through the joint A2I2 (Aircraft Armament Interoperable Interface) Program.

The electrical interface is a major contributor to the non-standardized and non-interoperable problems. Immediate corrective action is needed by the military to circumvent continuing escalation of the problem. Thus, development of MIL-STD-1760 was set into motion. In the process of evolving MIL-STD-1760, it became apparent that issues driving this interface were very complex, broad in scope and consequently difficult to defend in terms of establishing a single interface standard. It was more specifically determined that definitive documented requirements were an essential ingredient for providing direction and substantiation during MIL-STD-1760 development.

This paper provides a summary of the work performed under an A2I2 contract sponsored by the Naval Weapon Center (NWC) and Air Force Armament Laboratory (AFATL) for developing the aircraft-store electrical functional requirements which will be principally implemented by MIL-STD-1760. The paper provides an overview of the overall requirement drivers and then focuses on three principal electrical areas of the AEIS: (1) The power interface, (2) high bandwidth signaling, and (3) digital data transfer. The paper provides insight on derivation of these requirements and supporting rationale in terms of drivers from existing store requirements, developmental store and technology trends, and "traditional" engineering approaches.

INTRODUCTION

The Problem

Variations in the interfaces (electrical, mechanical and operational) between stores and aircraft have complicated the inter- and intra-service exchange of stores among different aircraft. These variations have also complicated the introduction of new stores into service. This complication is due primarily to the cost and schedule associated with modifying existing aircraft for compatibility with the new stores or specifically with the new store interfaces. The variations in aircraft/store interfaces are caused by a number of factors, including: The lack of physical, electrical, and information compatibility, old technological performance characteristics, excessive logistical demands, and variable maintenance requirements. Many of these problems are due to a proliferation of aircraft/stores equipment - racks, umbilical cables, store management mechanizations, and functional needs of the stores. This proliferation has occurred because of such factors as rapid technological advances, trends toward higher store sophistication, acquisition processes dominated by cost and schedule concerns, and downward compatibility requirements of new equipment to existing stores and/or aircraft.

These current problems, coupled with trends in aircraft, in stores, and in operational and procurement environments, make it increasingly important to

achieve the maximum degree possible of aircraft-store interoperability. Interoperability will minimize development and implementation costs for new weapon systems, maximize combat capabilities of the force structure, allow quicker response to changing threats, and minimize logistic support problems, among many other advantages. This interoperability must be achieved, however, in a technological environment of rapidly increasing complexity and sophistication of both aircraft and stores. The approach used for achieving the desired interoperability must support technological advances over a significant span of time.

Interface standardization is a viable approach for increasing interoperability and for providing future solutions to these current problems. This approach to achieve interoperability must be accomplished through an optimum balance between standardized and customized features.

Since the electrical interface between aircraft and stores is a major contributor to compatibility obstacles, timely corrective action is needed. This timely action is particularly warranted based on the accelerating trends of more complex store system designs and the resultant variety of new interface implementations which evolve with these designs.

The Solution

These accelerating trends and the resultant impact on aircraft Stores Management Systems (SMS), set into motion the development of MIL-STD-1760. This standard development is the first inter-service coordinated effort to evolve a standardized Aircraft-Store Electrical Interconnection System (AEIS).

In the process of evolving MIL-STD-1760, it became apparent that the functional requirements and issues driving this interface were very complex, and consequently, should be well defined prior to finalizing the aircraft-store electrical interface standard.

This paper summarizes a portion of the AEIS functional requirements evolved during an Aircraft Armament Interoperable Interface activity sponsored by the Naval Weapons Center, China Lake and the Air Force Armament Laboratory, Eglin AFB. This effort primarily involved stepping back from the present issue of MIL-STD-1760 (July 1981) and re-evaluating the overall AEIS functional requirements. This effort was deemed advisable prior to starting a completion (revision) cycle on the MIL-STD-1760. As a reminder, the present MIL-STD-1760 only defines the signal set at the aircraft station interface. As stated in the standard, the electrical elements will be refined, logical and physical elements added, and interface definitions added for other points in the AEIS - such as electrical interfaces at mission stores and at carriage stores.

But First, Some Terminology

Prior to delving into the details of this paper, definitions are presented for extensively used terminology. This terminology is consistent with MIL-STD-1760 and is repeated here as a refresher.

To begin, two general types of stores are defined. While both types are "stores" in the general armament sense, the two have significantly different functions and significantly different interface functional requirements. The first type is referred to as a "Carriage Store". As the name infers, this store is, in reality, Suspension and Release Equipment used for the carriage of one or more other stores. Examples include existing Multiple Ejector Racks (e.g., MER-10), Triple Ejector Racks (e.g., TER-7), Missile Launchers (e.g., LAU-117), etc.

As might be expected, the second type of store is the actual store that directly supports the mission function. This type is defined as a "Mission Store" and includes such items as bombs, missiles, electronic pods, gun pods, etc.

To further refine AEIS terminology, the electrical interfaces at these two store types plus at the aircraft store stations are also assigned specific identifiers. These interfaces are illustrated in Figure 1 and are defined as:

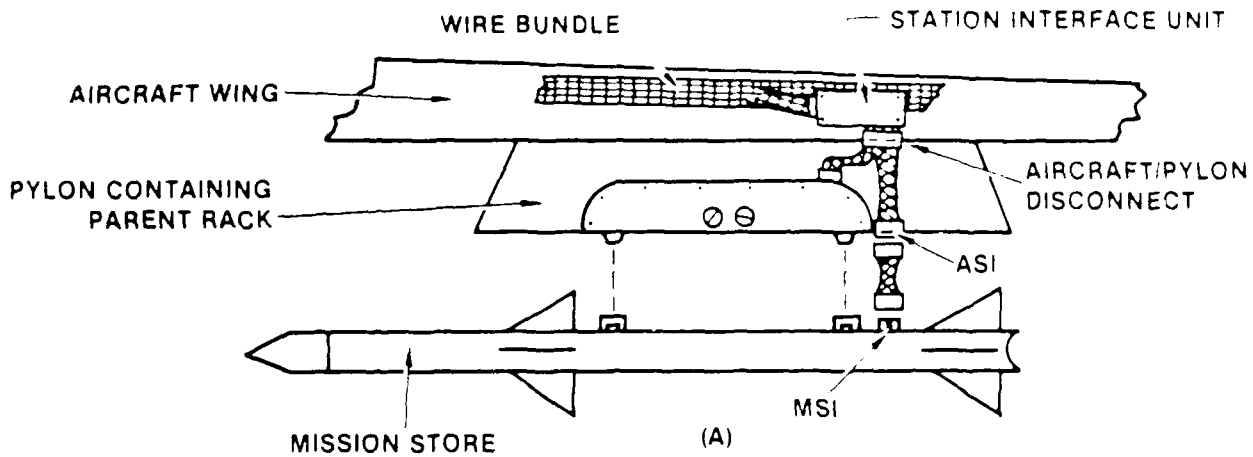
- (1) Aircraft Station Interface (ASI) - The electrical interface (e.g., at connector) located at the "lowest" point of aircraft dedicated store interfacing equipment. The ASI is the "aircraft side" of the AEIS.
- (2) Carriage Store Interface (CSI) - The electrical interface (e.g., at connector) on the carriage store through which the carriage store interfaces with the aircraft (i.e., with the ASI).
- (3) Carriage Store Station Interface (CSSI) - The electrical interface (e.g., at connector) on the carriage store through which the carriage store interfaces with a mission store. A multiple station carriage store will have associated multiple CSSIs.
- (4) Mission Store Interface (MSI) - The electrical interface (e.g., at connector) through which a mission store interfaces a carriage store (CSSI) or an aircraft (ASI). Mission stores may connect directly to an ASI (no intervening carriage store) or to a CSSI.

With this terminology outlined, the primary discussions will begin.

AEIS FUNCTIONAL REQUIREMENTS

Overview of AEIS Requirement Drivers

For most stores to be operated by aircraft, two very basic electrical functions must be performed. First, the aircraft must deliver to the store sufficient electrical energy (or power) to allow the store to operate. Various stores require various levels of the power. Second, information must be transferred between the aircraft and store and in some cases, between stores. Again, various stores and aircraft require various levels of information exchange. To enhance store interoperability among a large number



ASI - AIRCRAFT STATION INTERFACE
 CSI - CARRIAGE STORE INTERFACE
 CSSI - CARRIAGE STORE STATION INTERFACE
 MSI - MISSION STORE INTERFACE

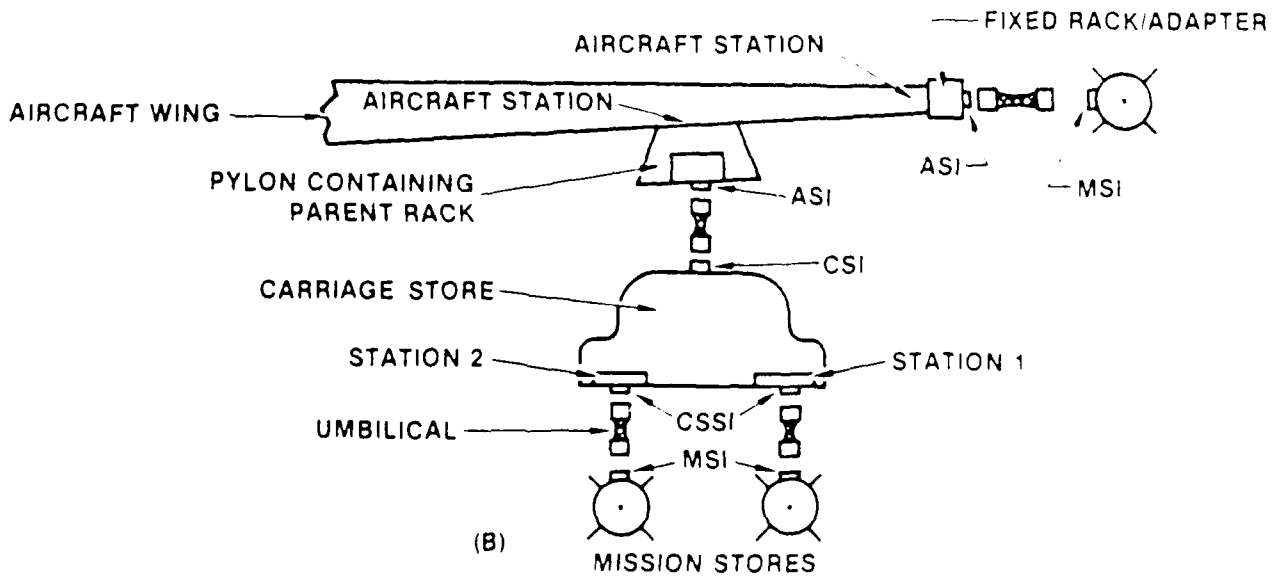


FIGURE 1 AIRCRAFT/STORE INTERFACE LOCATIONS

of aircraft types, standardization of these information and power transfers is suggested.

Many weapon system issues influence the AEIS (MIL-STD-1760) requirements. Table 1 lists eight of these general issues. Each of these eight issues are discussed below as a sampling of AEIS functional requirement drivers. Other issues (in addition to functional requirements) will also impact the AEIS Standard. An example is the temptation for traditional aircraft or store design implementations to influence the AEIS Standard even though these traditional approaches are no longer technically necessary.

TABLE 1

GENERAL ISSUES WHICH IMPACT AEIS REQUIREMENTS

- o Types of Information Transfer
- o Types of Power
- o Store Loadouts on Aircraft
- o Level of AEIS Standardization
- o Trends in Aircraft and Store Systems
- o Nuclear Weapon Compatibility
- o Multiple Carriage Compatibility
- o "Efficient" Interfaces for Low Cost Simple Stores

The information exchange between aircraft and stores can be categorized in several ways. One meaningful grouping is oriented toward information transfer implementation methods. A survey of current and projected aircraft/store information transfers identified three basic information transfer groupings. The first group consisted of relatively low transfer rate information which could easily be transmitted over conventional multiplexed serial digital data bus networks. This information group is referred to as "Digital Transfers". A second group contained data with information bandwidths sufficiently high that a time-shared, limited bandwidth multiplex bus network is not particularly practical. This second information group is categorized as "High Bandwidth Transfers". The final grouping applies to information transfers of extremely low data transfer rates. This third group, however, is also noted for peculiar information transfer requirements concerning: (1) The need to transfer the information under extreme conditions such as emergencies, loss of primary information channels, etc; (2) the need to transfer the information independent of other transfer channels; or (3) the need for safety critical functions to be implemented reliably. To achieve these three requirements essentially requires dedicated signals. This third information transfer group is referred to as "Discrete Transfers".

All information transfers between aircraft and stores can be divided into one of these three information types. The vast majority of information transfers uncovered during the aircraft/store survey could be practically implemented as "Digital Transfers". In contrast, a very limited number of high bandwidth and discretely were found. A large percentage of existing stores contain discrete based interfaces. However, these existing discretely should not be categorized as "Discrete Transfers" as described above, but converted to "Digital Transfer" implementations.

The aircraft/store survey also produced data revealing the types, or more meaningfully, the quantity of power required for store operation. The survey indicated that standardizing several characteristics of electrical power would yield significant payoffs in interoperability. These standards will be discussed in more detail later. At this time, it will be simply stated that store designers should be aware of realistic power limits and characteristics to which stores should be designed. Similarly, aircraft designers would be required to provide certain power levels and characteristics at the ASIs for store use. These levels and characteristics should, however, be realistic with respect to other aircraft design issues - such as weight and cost minimization. For example, simply because a store which required 15 kilowatts of electrical power could be identified, does not imply that the AEIS should require a 15 kilowatt capability at any or all aircraft store stations.

Another critical issue which impacts the AEIS requirements - i.e., impacts the information and power transfer - concerns realistic store loadouts. The number and mix of stores which can practically be installed on any specific aircraft determine a set of composite or concurrent information/power transfer requirements for that aircraft. Potential store loadouts for various Air Force/Navy missions were identified in an earlier A2I2 sponsored study. From these existing and projected loadouts, a set of information and power transfer stressing cases were selected. These stressing cases formed one set of drivers of the AEIS functional requirements.

An additional issue which affects the AEIS requirements deals with the level or degree that interface characteristics and performance are to be standardized. Once this standardization level crosses some gray boundary, two conflicting AEIS requirements clash. The first or prime directive of the AEIS is to define an aircraft/store interface which promotes "interoperability" among a broad class of stores and aircraft. This interoperability is achieved by standardizing all (or most) interface characteristics between aircraft and stores. The conflict arises, however, as more details are specified in a standard. The conflict occurs because operating, performance, and design flexibility are degraded as standardization is increased. As an oversimplified example, if all power interfaces were standardized as a single 28 VDC, 10 ampere line, interoperability would be high (with respect to the power interface). However, the type of store performance or operations which could be implemented with 280 watts would be restricted significantly. For this reason, a median must be reached between interoperability enhancements and operation flexibility restrictions. Those areas where meaningful aircraft/store flexibility can be maintained while interface standardization is pursued, should be prime targets for the AEIS Standard.

As the interface standard is evolved, maintaining compatibility with the perceived trends in aircraft and store designs is extremely important. An AEIS Standard, which becomes technically obsolete after a few years, has no value. Examples of these perceived trends which could impact the AEIS requirements are:

- (1) Expanded use of serial digital (multiplex bus based) data transfers. The promulgation of MIL-STD-1553 multiplex techniques through

avionics is certainly indicative of this trend. The serial digital multiplex information transfer technique is representative of a standardization step which not only enhances interoperability but in a practical sense simultaneously enhances system flexibility.

- (2) Reduction of the number of different electrical characteristics by which power is supplied to stores. This reduction, as an initial step, should eliminate all voltages other than those identified in MIL-STD-704. Standardizing to a smaller subset of MIL-STD-704 should also be considered.
- (3) Increased autonomous store operation. This trend should reduce the amount of information transferred between aircraft and stores. This tends to increase store interoperability but will also likely increase store procurement cost. Specific areas in which information reductions are anticipated are in the "High Bandwidth Transfer" category.
- (4) Increased use of sophisticated electronic pods such as target acquisition systems could occur at some aircraft stations. While the external pod application could conflict with attempts to minimize radar cross-section, the functional interface needs of these pods must also be considered.

Another area to consider when establishing requirements for the AEIS concerns the drivers imposed by nuclear weapons. A specification (System 2) for the next generation electrical interface for nuclear weapons is being developed. This specification imposes particular requirements on the Digital Transfer, Discrete Transfer and Power Transfer interfaces. Most of these System 2 requirements are not significant drivers on the AEIS. The primary area of significance concerns the level of quality and electrical isolation imposed on the Power Transfer function.

An additional complication on the AEIS results from insertion of carriage stores between the aircraft and mission stores. This addition has two effects. First, the insertion of a carriage store modifies the characteristics of the electrical signals from the aircraft (ASI) prior to reaching the mission store (MSI). Second, a multiple station carriage store conceptually changes the ASI from a single store interface point to a multiple store interface point. To complicate the issue even further, the number of stores that might be simultaneously loaded on carriage stores changes with different types of carriage stores.

One final AEIS requirement driver area is briefly mentioned. This issue deals with how an efficient and cost effective interface should be implemented to service the large number of low cost (and relatively simple) stores. The addition of store hardware to support a redundant MIL-STD-1553 serial digital line could have a significant cost impact on low cost stores such as freefall bombs.

The eight issues just discussed are representative of aspects which influence the interface requirements imposed on the AEIS. These eight issues do not, however, encompass the total set of AEIS drivers.

The next several sections highlight representative AEIS issues for power, high bandwidth and digital transfers. These issues are presented at a higher level of technical detail than the general discussion presented above.

Power Transfer Requirements

The present MIL-STD-1760 divides the power interface (at the ASI) into two parts (implemented as two connectors). This division is by power quantity. The primary connector has approximately 6.7 kilowatts of "power capacity" identified. This power is partitioned among three voltage characteristics and ten connector contacts. Each contact is rated for ten amperes. Similarly, the auxiliary MIL-STD-1760 connector contains the same three voltage characteristics distributed among eight contacts with a composite rating of nearly 20 kilowatts. One important point should be emphasized, however. Simply because the primary or auxiliary connector contains sets of power functions, each rated for a particular power capacity, it should not be interpreted that rated power can be simultaneously extracted through all of these contacts simultaneously. Likewise, it should not be interpreted that aircraft will contain sufficient power capacity to simultaneously energize all aircraft station interfaces at their maximum power ratings.

These restrictions should be identified in MIL-STD-1760. As an example, at the time of designing the store's interface to the aircraft, a design decision is required on which voltage characteristic (or group of voltage characteristics) to select. Any limitations imposed by the aircraft or carriage stores on power implementations should be highlighted to the store designer to maximize the specific store's interoperability.

A second power requirement issue concerns the quality or characteristic tolerances of delivered power. The present MIL-STD-1760 specifies that MIL-STD-704 voltage characteristics will be delivered to the ASI. The level of characteristic quality degradation from ASI to MSI (directly or through a carriage store) is not defined. More subtly, however, in-service aircraft are designed to various issues of MIL-STD-704. In fact, there are actually several versions of MIL-STD-704 which are presently active. Since the power quality must be defined at all AEIS interface levels and reference to MIL-STD-704 does not sufficiently identify the ASI power characteristics, the AEIS Standard must specify the power characteristics which a store should expect to be supplied and which an aircraft should be required to provide.

High Bandwidth Transfer Requirements

The electrical interfaces to a number of existing and projected stores include a variety of analog or high bandwidth signals. These signals include video, audio, time correlation pulses, variable voltages used for scaling or "synchro" control, plus modulated Radio Frequency (RF). Analysis indicated

that the response/timing and information content of some of these analog signals is sufficiently low to permit conversion to a digital format and transfer on a MIL-STD-1553 one megabit/second data bus. In contrast, some of these present analog formatted information transfer functions are grossly inappropriate for transfer over the MIL-STD-1553 data bus network. Three analog signal types definitely fall into this latter category. These signal types are real-time video, time correlation pulses and RF.

A number of existing stores transfer video information to the aircraft (i.e., to cockpit displays). Since this video transfer is expected to continue, a high bandwidth capability for routing video across the AEIS interfaces must be provided.

Likewise, Time Correlation Pulse (TCP) information transfer is presently used for accurate and efficient "blanking" of on-board RF receivers from on-board RF transmitters. These receivers or transmitters could be installed in stores. In addition, TCP applications include transferring precision clocking or synchronization signals between aircraft and stores.

The driver on the AEIS to include an RF transfer capability in MIL-STD-1760 is based primarily on projected Global Positioning System (GPS) applications for missiles. The missile receiver may require access to the RF signal (e.g., from GPS satellite transmitter) while the missile is installed on the aircraft. Since the aircraft may obstruct the missile antenna's line-of-sight access to the RF transmitter, an alternate access path to an airborne antenna through the AEIS interface is required.

Digital Transfer Issues

Prior to concluding this paper, a few comments are presented on several digital information transfer issues.

The vast majority of information transfers between aircraft and stores can be implemented with conventional serial digital multiplex methods. Since MIL-STD-1553 is the DoD designated serial digital bus communication method for military avionics, since MIL-STD-1553 hardware is off-the-shelf, and since MIL-STD-1553 can provide the performance levels identified for the AEIS digital transfers, selection of a MIL-STD-1553 serial digital link is justifiable. Although selection of MIL-STD-1553 provides an immediate level of maturity for AEIS digital communication, several additional communication aspects should be defined in MIL-STD-1760.

The intent of the MIL-STD-1553 standard was to supply a common or standard hardware communication interface which is to be supplemented by additional protocol, data word, and message specifications for a specific system implementation - such as the avionic suite of a particular aircraft. In a similar manner, the AEIS is a specific implementation with one subtle difference. While a specific aircraft system implementation and the associated communication protocol for applying MIL-STD-1553 may be limited to one aircraft type (e.g., F-16A, F-15C), the AEIS implementation must be common to all interoperable aircraft and stores. To enhance this interoperability, a

higher level of communication standardization is recommended for the AEIS above that defined in MIL-STD-1553. This higher level should approach that level defined for the specific system implementations discussed above. Although total definition (by the AEIS Standard) of the digital communication will contribute to interoperability, it will certainly not assure interoperability. Realistically, some communication aspects such as message timelines will not likely be appropriate for specification in the AEIS Standard. This, in turn, implies some level of software modification to an aircraft Stores Management System (SMS) must be performed whenever new AEIS compatible stores enter the services. What is definitely desired, however, is to eliminate any aircraft hardware modifications for adding compatibility with these new stores and to minimize any required SMS software modifications.

At the minimum level, the digital communication protocol aspects should be defined by MIL-STD-1760. These aspects include the message formats for identifying the position of specific communication control data and of end-user information in a MIL-STD-1553 contiguous message (i.e., a "command word" and up to 32 "data words"). Communication control data includes items such as (1) error correction/detection codes, (2) routing instructions across hierarchical data buses (such as SMS bus to internal carriage store bus), (3) data encryption tags, (4) provisions for linking large blocks of data which cannot be transferred as a single contiguous MIL-STD-1553 message, and (5) methods for transferring the message and/or data word identification(s).

In addition to standardizing the basic communication protocol - or as an analogy, the sentence structure and rules of grammar - a set of standard data words is highly recommended for the AEIS. The AEIS data words are envisioned as adopting the avionics system standard data words being developed for MIL-STD-1553. Additions to this avionics data word set are anticipated to cover aircraft/store peculiar data requirements.

A number of other digital transfer issues must be considered during development of the logical elements for MIL-STD-1760. Two of these issues are: (1) the impact on digital transfer requirements when a carriage store is inserted between an ASI and one or more MSIs; and (2) the cost implications of a dual redundant MIL-STD-1553 interface on low cost stores and potential methods for cost reduction.

CONCLUSION


This paper provided an overview of some functional requirements imposed on a standard Aircraft-Store Electrical Interconnection System (AEIS). The primary goal of this AEIS standard is to curtail the proliferation of different interfaces between various aircraft and various stores. This will be accomplished by standardizing as many important electrical interface aspects as is practical without significantly restricting aircraft/store performance or impeding incorporation of technology advances. This standardization is being accomplished via MIL-STD-1760.

This paper also discussed several technical issues which influence the AEIS functional requirements. These issues must be considered as MIL-STD-1760 evolves to include the logical and physical elements of the AEIS in addition to expanding the electrical characteristics presently defined by the standard.

With the completion of MIL-STD-1760, a giant step will have been taken toward minimizing aircraft modifications required to achieve electrical compatibility with new stores. In addition, the interface development effort required for new stores and new aircraft will be significantly simplified over that required with today's methods and procedures. These improvements lead to both reducing development and service incorporation costs and enhancing the ability to share stores among the U.S. services and allied nations for an improved military response.



AD-P003 528



SIGNAL SET STANDARDIZATION
FOR THE
AIRCRAFT-STORE ELECTRICAL INTERCONNECTION SYSTEM

D.E. Lautner
J.R. Perkins
Vought Corporation
Dallas, Texas
(214) 266-3035

BIOGRAPHICAL SKETCH OF THE AUTHORS

Mr. Don Lautner, Vought Corporation, Dallas, Texas - Engineering Specialist in Electrical Systems and Circuit Design. Mr. Lautner's primary responsibility is principal investigator for aircraft/store electrical compatibility analysis and development. Recent experience includes consultation assistance to the Air Force and Navy for MIL-STD-1760 development. Education: BSEE, University of Texas at Arlington.

Mr. Jim Perkins, Vought Corporation, Dallas, Texas - Technical Project Manager for the Advanced Armament System and Advanced Electrical System technology programs. Mr. Perkins has been employed with LTV based companies for approximately 25 years and has been involved in development of aircraft, missiles, automatic controls for aerospace vehicles, geophysical exploration equipment, and military electronic equipment. Mr. Perkins obtained his BSEE from Texas Technological University in 1958 and MSEE from Southern Methodist University in 1962.

ABSTRACT

The Air Force and Navy are conducting a joint program (Aircraft Armament Interoperable Interface [A²I²]) to standardize interfaces between aircraft and stores. One product of this joint A²I² program is a military standard for the Aircraft-Store Electrical Interconnection System (AEIS). This standard, released in July 1981 as MIL-STD-1760, defines the electrical interface between aircraft and stores. As mentioned in the MIL-STD-1760 foreword, the complete AEIS is comprised of electrical, logical, and physical elements. The present MIL-STD-1760 issue addresses only the electrical signal set element. This paper provides an overview on the background for the selected MIL-STD-1760 electrical signal set. Following this overview, application restrictions, application guidelines and various technical issues

are discussed for each of the power, digital, high bandwidth and discrete signals of MIL-STD-1760. The discussion covers the electrical signal set characteristics presently defined in MIL-STD-1760 plus clarifications and more rigorous definitions of the electrical signal characteristics expected in a future revision to MIL-STD-1760.

INTRODUCTION

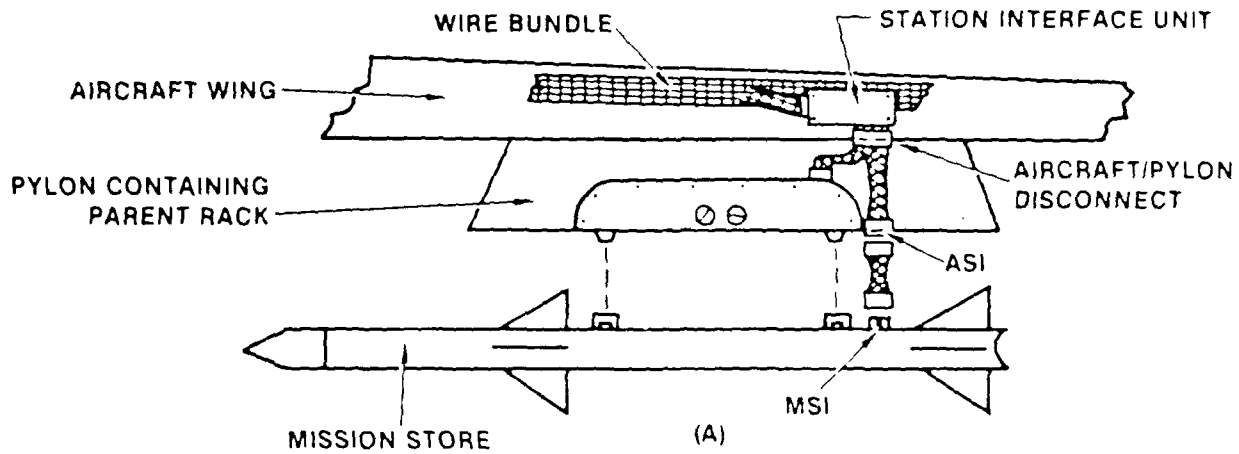
Other papers in this session present a broad range of views on MIL-STD-1760 - from Air Force and Navy perspectives to actual (or planned) system implementations. This paper introduces the signal set which comprises the Aircraft-Store Electrical Interconnection System (AEIS) standard. This signal set forms the basis of the AEIS standard (i.e., MIL-STD-1760). Logical and physical interface elements will be merged with the defined signal set in a later revision to the AEIS standard.

Although the logical elements (primarily software) and the physical elements (primarily connectors) have not been formally defined, identification of the signal set across the aircraft-station interface is a giant step toward implementing an interface standard. To this end, a signal set for the Aircraft Station Interface (ASI) was formally defined with the July 1981 release of MIL-STD-1760. The intent is to refine this same signal set (in MIL-STD-1760A) for application to the remaining AEIS interfaces, i.e., the Mission Store Interface (MSI), the Carriage Store Interface (CSI) and the Carriage Store Station Interface (CSSI). These various interface points are illustrated in Figure 1.

This paper provides an overview of the defined AEIS signal set and presents perceived restrictions and guidelines for signal set application to next generation stores and to new and modified aircraft.

OVERVIEW OF SIGNAL SET

The AEIS signal set was selected after several years of requirements analysis, comprehensive military and industry reviews and open forum discussions. These signals were selected to support next generation equivalents to current stores, present stores in the development cycle and stores in the concept stage. The number of signals in the selected set (see Figure 2) are limited in number and type but sufficient to support the projected interface requirements. Some people have even argued that too many signals are included in the AEIS interface and that the vast majority of stores could be operated from a smaller signal complement. While this is generally true, the signal set must also include (1) a level of growth capability to deter technical obsolescence, and (2) sufficient signals to allow a degree of flexibility in store interface designs and operational capability. A brief overview of the selected AEIS signals follows.



ASI - AIRCRAFT STATION INTERFACE
 CSI - CARRIAGE STORE INTERFACE
 CSSI - CARRIAGE STORE STATION INTERFACE
 MSI - MISSION STORE INTERFACE

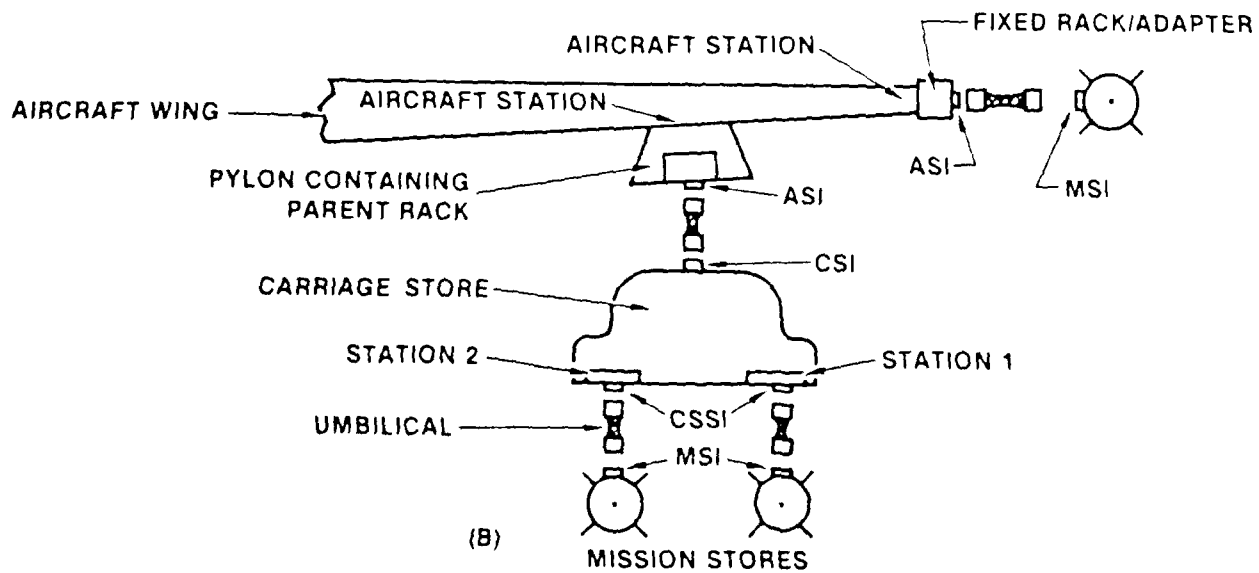


FIGURE 1 AIRCRAFT/STORE INTERFACE LOCATIONS

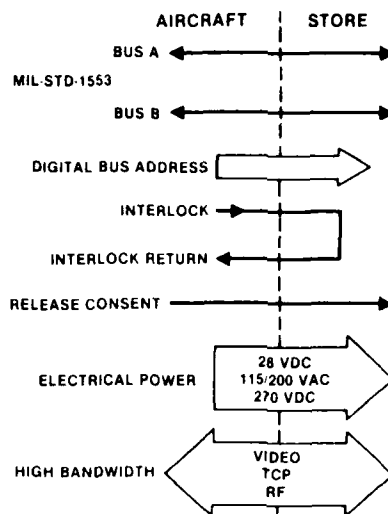


FIGURE 2 AEIS PRIMARY SIGNAL SET

The AEIS interface evolved around a dual redundant MIL-STD-1553 serial digital data link. This redundant link is illustrated in Figure 2 by Bus A and Bus B. Through this serial digital interface, a flexible communication link exists between aircraft and stores. This military/industry accepted multiplexed data bus standard provides sufficient data capacity and speed for most aircraft-to-store and store-to-store information transfer requirements.

To support the MIL-STD-1553 communication link and to provide energy for accomplishing the store functions, several types of electrical power are supplied at the AEIS interfaces. Basically, power is supplied at the AEIS interfaces with characteristics defined by MIL-STD-704. The power types provided include three phase, 115 volt ac and 28 volt dc. In addition, growth provisions are reserved in the signal set for 270 volt dc power sources. To minimize the size of the interface connectors, the power quantity was divided among two connectors. The primary connector was sized with sufficient power capacity to service most stores. An auxiliary power connector was also defined for powering those stores which require larger quantities of energy such as ECM pods.

Since the MIL-STD-1553 communication link is bus oriented, a method is required for assigning data bus addresses to the multiplex terminal in each store. Since several stores of the same type could be simultaneously loaded on a given aircraft, the address can not be assigned to the store at manufacture. For maximum flexibility, the store's multiplex terminal address should be automatically assigned during store installation onto the aircraft. Several methods could be used to accomplish this assignment. The selected method relies on five binary encoded discrete lines plus a parity discrete and return lines.

A simple continuity check (interlock) through each AEIS interface was also defined. This allows the aircraft to sense that the electrical interface with the store is properly mated without energizing the store and then interrogating the store over the MIL-STD-1553 bus. This interlock function has historically been labeled "store present" or "ground interlock". Not only does this interlock signal permit a quick check of the interface mate condition without store power-up, but also allows continuous monitoring of the interface without repeated interrogations on the MIL-STD-1553 bus. In addition, the same interlock signal set can be used by the store as an indication that the store has not separated from the aircraft.

To enhance safety during store operation, and to allow the aircraft (i.e., pilot) to maintain ultimate control of store safety related functions (firing, ejection, arming, etc.), a release consent discrete signal is included in the AEIS interface. This discrete is basically a low power, 23 volt dc signal which is activated during the time window that an armed release is desired. The signal was originally intended for (1) multiple carriage bomb racks which could erroneously release mission stores connected below the multiple rack and for (2) rail launched missiles which could ignite their own rocket motors as a result of commands on the MIL-STD-1553 data bus. The discrete could, however, be used by a broader class of weapons as a general "intent to launch" command. It should be noted that the discrete is not a launch command or a release signal. The signal is instead a release enable which is required before an armed release of the store is permitted. Secondly, it is intended that application of release consent not cause activation of an irreversible process within the store - such as firing an electroexplosive device, activating a thermal battery, etc.

Finally, the AEIS interface includes several "high" bandwidth analog signal transmission links for transferring information with bandwidths beyond MIL-STD-1553 data rates. A review of present and projected stores revealed that all interoperable stores could be serviced with five or fewer high bandwidth lines - two video lines, two Time Correlation Pulse (TCP) or RF lines, and one audio line.

Video information is the most predominate example of the high bandwidth signal type in existing stores. The intent of MIL-STD-1760 is to zero-in on STANAG 3350 for these video signal characteristics. The latest STANAG draft includes a video class with signal characteristics comparable to present U.S. video based stores, such as Maverick, Hobo, Lantirn, etc. Since the worst case video application projected required two simultaneous video transfers across an AEIS interface, MIL-STD-1760 included a dual video transfer capability. In addition, the video transmission link was specified to include growth capability for high resolution video.

A second high bandwidth signal class found in and projected for several store types was a time correlation or clocking pulse. Time correlation pulses are typically used for "blanking" RF receivers or for synchronizing time critical systems. While these signals are primarily DC pulses, the rise time and propagation delay requirements on the pulse mandate a high quality transmission line. In contrast with the video standardization efforts, no

attempts have previously been made to standardize Time Correlation Pulse (TCP) signal characteristics.

The third high bandwidth class is for transferring signals in the Radio Frequency (RF) bands. The primary application identified was for future stores using the Global Positioning System (GPS) as a mid-course guidance aid. The missile's antenna would likely have its view of one or more GPS satellites obstructed by the carrying aircraft. Therefore, access to an aircraft installed antenna via the AEIS interface is projected. This application generated an interface requirement for an RF transmission line with an approximate two Gigahertz capability.

The final "high" bandwidth signal class was based on a perceived need for an audio transfer capability. The information quantity in most store "audio" signals is sufficiently low that transfer on a MIL-STD-1553 bus is generally feasible. However, since real-time transfer of audio on the MIL-STD-1553 data bus is marginal for some applications, a twisted, shielded wire pair was included in MIL-STD-1760 for audio.

APPLICATION ISSUES

With the completion of a general overview of the AEIS interface signal set, several application issues are presented. These issues are grouped by interface signal type, i.e., power, digital, discrete and high bandwidth transfers.

Power Transfer

The power demand imposed by most existing and in development stores typically varies with time, i.e., the load is not at a constant "steady-state" level. These time varying load limits should be defined with one of two methods. The first method is to define a steady state or continuous power load limit and not permit a store to exceed that limit even for short time intervals. The second method takes advantage of short-term power delivery capacity of conductors, switching components, and power sources by allowing a store load demand to exceed the continuous rating for short time intervals. Limits on acceptable short-term or transient load conditions should, therefore, be defined in MIL-STD-1760 in addition to the steady-state load limits. The steady-state and transient load considerations are more fully discussed in subsequent paragraphs.

Figure 3 illustrates representative total steady-state power demands for a number of production and development mission stores. The figure segregates the power demands by four types of mission stores: Unguided freefall, air-air guided, air-surface guided and miscellaneous pods (typically electronic pods). The figure also identifies those mission stores which either are presently installed on multiple carriage stores or are potential candidates for multiple carriage. This figure defines the summation or composite steady-state load demand of all voltage levels supplied to the stores. No division by voltage type, i.e., 28 VDC, 115 VAC, etc., is shown.

One point to note in Figure 3 is that if the electronic (or special) pod category is excluded, the remaining stores could be operated with approximately two kilowatts or less of power. Since conductor, contact and switching hardware size increases as power (or more precisely current) capacity increases, a cost and weight pay-off was envisioned by identifying two levels of interface power capacities - one at approximately two kilowatts and one at approximately ten kilowatts. The ten kilowatt capacity interface could then be restricted to a smaller number of interface locations for more "efficient" aircraft and carriage store designs.

The transient or short-term mission store load limit is depicted in Figure 4. The load profile for several production stores is shown in the figure. In addition, the figure identifies a typical power capacity profile available from the aircraft. This profile, normalized to the continuous rating (100 percent), illustrates the higher power levels available on power lines for short time intervals. The shaded area in the figure identifies the band within which aircraft circuit protection hardware will likely sense an overload condition and remove power. Any store whose load penetrates this current-time boundary will likely have its power removed, e.g., the sourcing circuit breaker will trip. To adequately identify the operational load limits for the mission store and the short-term sourcing capacity of the aircraft, the AEIS standard (i.e., MIL-STD-1760) should define the transient as well as continuous load limits.

Another power requirement issue concerns the quality or characteristic tolerances of delivered power. The present MIL-STD-1760 specifies that MIL-STD-704 voltage characteristics will be delivered to the ASI. The level of characteristic quality degradation from ASI to MSI (directly or through a carriage store) is not defined. More subtly, however, in-service aircraft are designed to various issues of MIL-STD-704. In fact, there are actually several versions of MIL-STD-704 which are presently active. Since the power quality must be defined at all AEIS interface levels and reference to MIL-STD-704 does not sufficiently identify the ASI power characteristics, the AEIS standard must specify the power characteristics which a store should expect to be supplied and which an aircraft should be required to provide.

The final power issue discussed in this paper deals with the specific MIL-STD-704 voltages selected for implementing the AEIS power transfer. One goal of AEIS standardization was to minimize the number of voltage levels at which power is supplied to stores. Present stores receive power at MIL-STD-704 voltage levels, i.e., 28 VDC and 115 VAC, 400 Hertz. In addition, however, several of these existing stores also receive power at non-standard voltage levels such as 30 volts, 95 volts, +15 volts, etc. These non-standard levels should not be promulgated at the interface. The basic requirement for power transfer is transparent to the selected voltage level(s). All that is required is to transfer power at a defined voltage level(s) and specify the associated characteristics. Reducing the number of power types at which this power is transferred contributes to interface simplification. Due to the relatively high quantity of power transfer required (2-10 kilowatts), a power source with a relatively high voltage level is desirable as the primary power medium. Use of a high voltage lowers the conducted current level which, in turn, reduces the size of conductor and switching hardware.

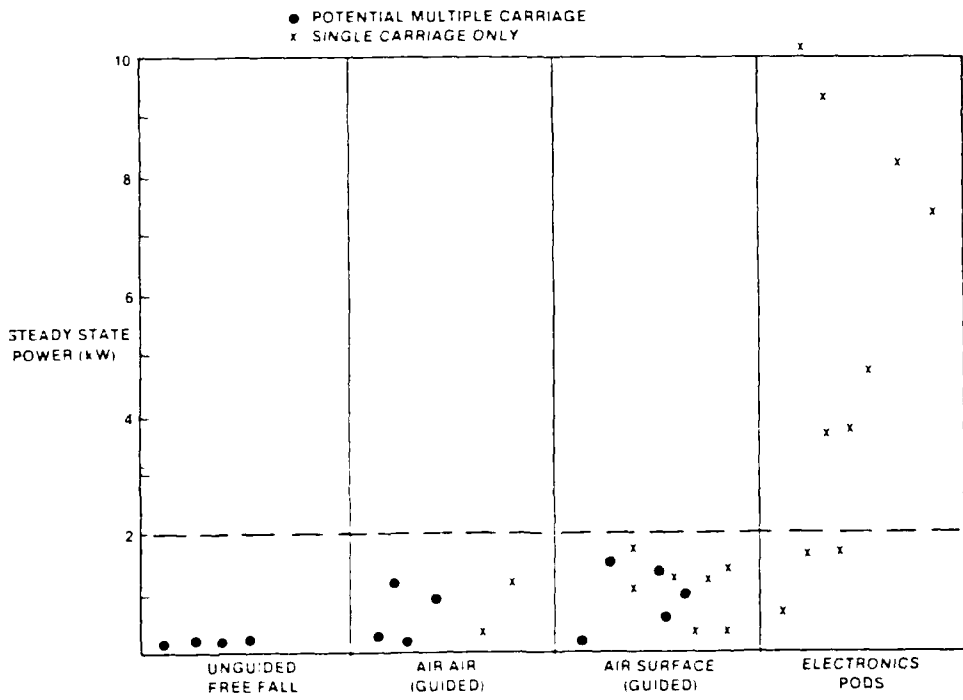


FIGURE 3 TOTAL (COMPOSITE) MSI LOAD DEMAND

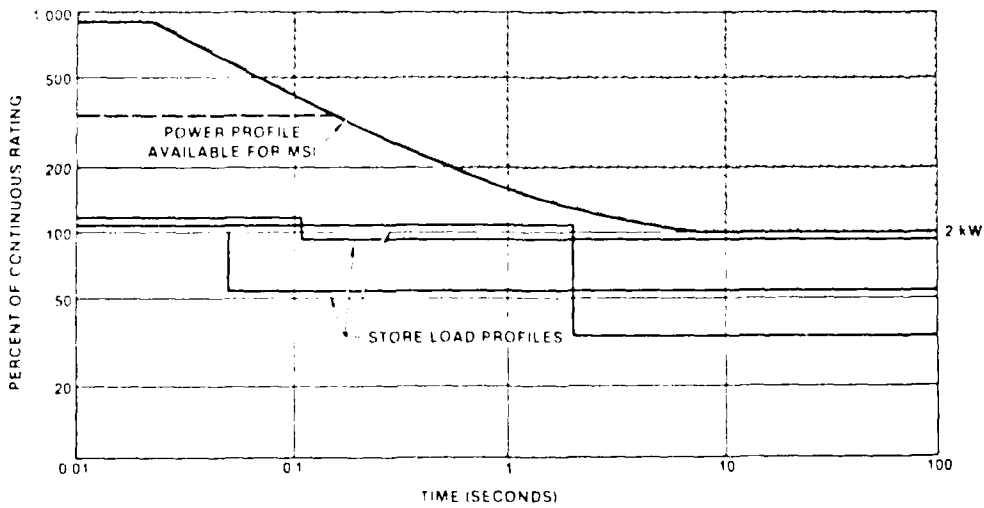


FIGURE 4 COMPOSITE LOAD PROFILE FOR MSI

The power transfer requirements in MIL-STD-1760 specifies three-phase, 115 volts ac as the primary power medium. Two sources of 28 volts dc are also included for implementing simple power supply functions in low power stores. Finally, growth provisions for 270 volt dc power transfer medium are reserved for compatibility with future aircraft power system trends. It should be noted, however, that inclusion of all three of these voltage characteristics in MIL-STD-1760 was based in part on historical precedence and not on AEIS functional requirements.

Digital Transfer

Since the AEIS digital interface incorporates MIL-STD-1553 redundant buses - or more specifically - bus stubs, the electrical characteristics and application issues are fairly well defined. The primary subtlety in the AEIS digital transfer evolves around two issues: (1) Stub lengths vary as different stores are installed on different aircraft or different aircraft stations; and (2) accommodation of carriage stores results in unique architectural requirements on the Stores Management System and its associated data bus.

The first issue (i.e., stub length variations) occurs because of the segmentation of the data bus stub between aircraft, umbilical cable and store. The main data buses and the associated bus couplers are installed on the aircraft side of the AEIS interfaces. Similarly, the MIL-STD-1553 remote terminals are installed in the store side of the AEIS interfaces. The data bus stub connecting bus coupler to remote terminal is therefore divided into the three segments mentioned above - i.e., aircraft, umbilical cable and store. Since different manufacturers build these segments, fine tuning of data bus performance is more complex than that required for "fixed" bus structures totally contained within the aircraft. In addition, as stores are released, the stores' remote terminals and a portion of the bus stubs disappear one at a time. Thus, the physical and electrical characteristics of the data bus are time varying.

The second digital transfer issue concerns the change in data bus architecture when carriage stores are installed at the Aircraft Station Interface (ASI). This application is illustrated in Figure 5. Conceptually, the addition of a carriage store converts a single port ASI into a connection point for multiple MIL-STD-1553 remote terminals. Since multiple Remote Terminals (RTs) can not be connected to a single MIL-STD-1553 stub, electronics must be added in a carriage store to implement a fan-out capability.

Figure 5 depicts one likely approach for achieving this fan-out. In essence, a second tier or hierarchical bus is installed in the carriage store. The carriage store contains an RT which properly terminates the data bus stub from the aircraft and extracts data for transfer to an associated Bus Controller (BC). The BC then passes this data down onto the carriage store's data bus and through a new set of data bus stubs into the mission stores. In a similar but more complex manner, data can also be transferred from mission stores up through a carriage store to the aircraft data bus.

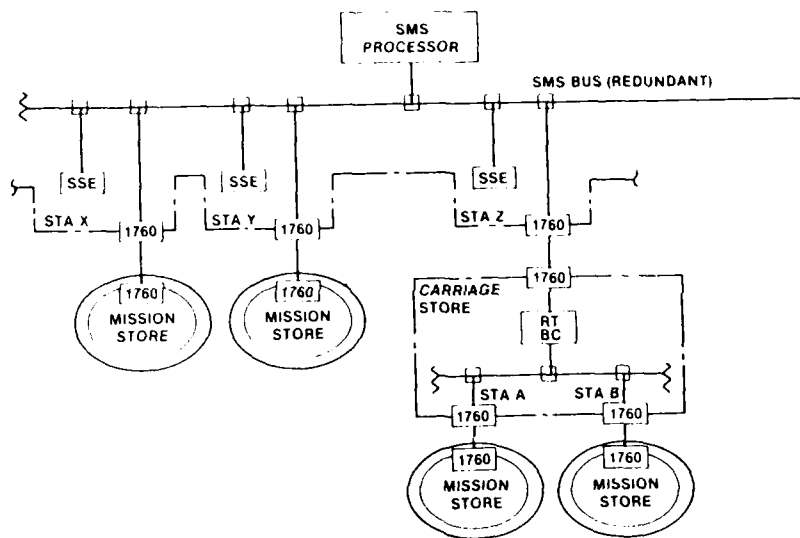


FIGURE 5 AEIS DATA BUS ARCHITECTURE

Discrete Transfers

The signal set overview included background on the three discrete signal types - data link address, aircraft/store interlock and release consent. The discussion which follows clarifies misconceptions on two of these discrete signals - the data link address and the interlock function.

Although careful reading of the address line requirements in MIL-STD-1760 should not result in any misunderstandings, the intent and application of these address lines are frequently misinterpreted. The intent of the address lines was discussed earlier in the signal set overview. To summarize, the MIL-STD-1553 Remote Terminal (RT) in the store must be informed of the address to which it is to respond. Since this address is expected to vary among store stations on an aircraft and among aircraft, the RT's address can not be fixed during store manufacture. Instead, the store contains a set of discrete interface lines which, when mated to the aircraft, result in transferring the address associated with that aircraft station. To minimize interface incompatibilities, these discrete lines are defined as electrically isolated from other aircraft circuits including ground. As illustrated in Figure 6, the aircraft provides "open circuits" or "continuity" between each address line and the associated address return. The aircraft side of the address discrettes is totally isolated from all other aircraft circuits. This allows a large degree of flexibility in the store circuit design.

Additionally, a degree of flexibility for address assignments should also be available to the aircraft. The aircraft should be permitted to use active components (such as the optical couplers shown in Figure 6) to implement the "open/short" address encoding. The use of active components permits assignment of store address by the SMS as a function of the aircraft's store load-out. This active address assignment is not presently identified in MIL-STD-1760.

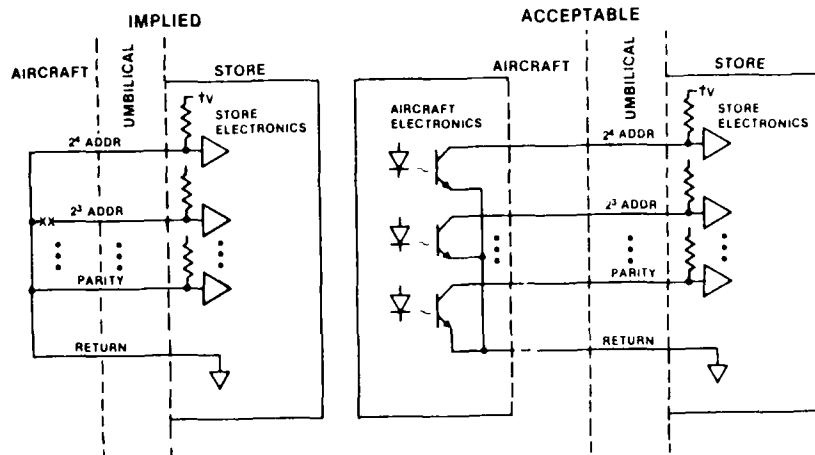


FIGURE 6 ADDRESS TRANSFER IMPLEMENTATIONS

The second discrete transfer issue discussed in this paper concerns the aircraft/store interlock discrete. As mentioned in the signal set overview, this interlock discrete is comparable to the "ground interlock" discrete provided by many existing stores. The primary differences in the AEIS interlock signal are twofold. First, a dedicated interlock return is included in the interface for the interlock signal. This reduces signal errors due to noise pickup if a power return is shared for the interlock function. Secondly, the dedicated interlock return also allows the store to sense that its interface is mated with the aircraft.

Figure 7 illustrates two implementations of the interlock function on both aircraft and store sides of the interface. The diodes associated with the interlock lines are recommended to isolate the aircraft and store internal electronic power supplies - particularly if the interlock return line opens (e.g., return wire breaks). The interlock discrete is specified by the AEIS standard in a manner that gives the aircraft and store designers the option of monitoring or not monitoring the interlock. All stores must, however, provide continuity between interlock and interlock return. Likewise, all aircraft must provide continuity between interlock return and power (or structure) return.

One final comment on the interlock discrete is presented. The interlock discrete monitors the mating status of the AEIS connectors. Historically, however, the ground interlock signal has been used to imply the physical presence or absence of the store. The interlock signal should not be used as a "store gone" indication for safety related functions. This application restriction is strongly recommended because too many failure modes exist for the interlock signal which yield an erroneous "store gone" indication. Likewise, use of the interlock as a "store present" indication would also need to be evaluated with respect to the impact of signal failures.

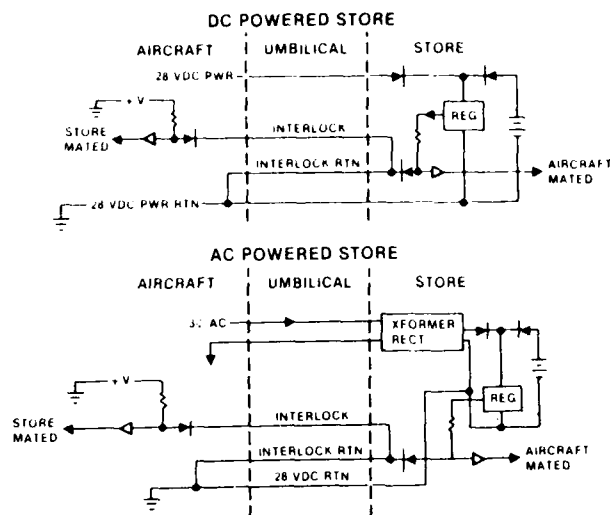


FIGURE 7 INTERLOCK IMPLEMENTATIONS

High Bandwidth Transfer

As described in the signal set overview, three types of true high bandwidth signals are included in the AEIS standard - video, Time Correlation Pulses (TCPs), and RF.

While the technology is readily available to convert analog composite video signals to a digital format, real-time digital transfer of this information requires a multi-megabit per second digital communication channel. Real-time store video transfer cannot be implemented by a MIL-STD-1553 type serial digital bus. This transfer function must be performed over a high bandwidth media in either an analog format or a higher speed digital format. The present MIL-STD-1760 defines the video signal characteristics for compatibility with a modified EIA RS-170 standard (similar to Maverick, Hobo, and GBU-15 video), with EIA RS-343 and with STANAG 3350 Class A and B. However, the STANAG is presently being revised to include a Class C video comparable to existing U.S. video based stores. As a result, the next MIL-STD-1760 revision will likely define the video signal characteristics by reference only to the STANAG. The extended bandwidth of the transmission media to 20 Megahertz is expected to be maintained for a high resolution video growth capability.

Figure 8 illustrates the video signal flow expected through the AEIS. This flow is representative of present applications with one variation. Routing of video information between two stores is projected. The application of Lantirn type electronic pods which receive and process video information from other mission stores is representative of future store-to-store video transfers.

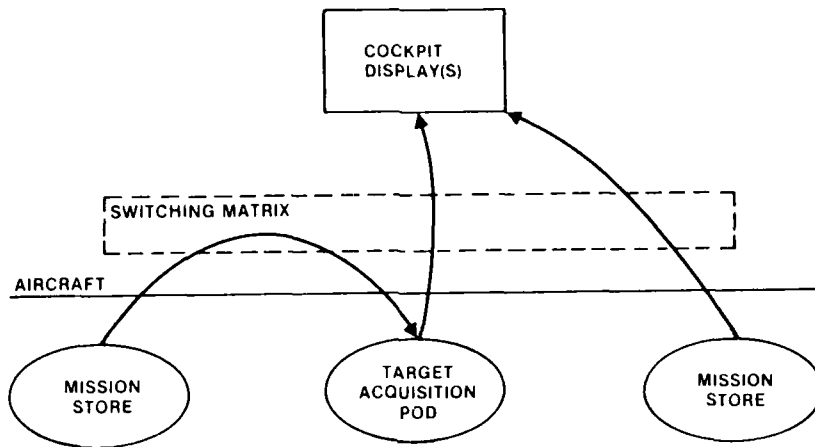


FIGURE 8 VIDEO SIGNAL FLOW

Time Correlation Pulse (TCP) transfer includes two general information types. The first TCP signal type provides a look-through or "blinking" capability in RF receivers from RF emissions by transmitters on-board the aircraft. The approach is to inform the receiver(s) of the precise time-of-arrival of the RF emission. The RF transmitter is the source for the TCP signal and the RF receiver is the ultimate signal sink. The primary "peculiar" requirement drivers for this critical timing signal are the fast pulse rise-time and low propagation delays required through the transfer network.

The second type of TCP is derived from a clocking or synchronization signal between two systems. For many of these applications, the synchronization tolerance requirement is tighter than that achievable with MIL-STD-1553 based synchronization.

There are no existing standards for TCP signal characteristics. The characteristics vary with each TCP signal application. Standardization of several pertinent characteristics is, however, realistic for the AEIS. Characteristics which can be standardized include Logic 1 and Logic 0 voltage thresholds, rise/fall time limits, propagation delay limits on the network, load impedance, etc.

The signal flow for TCP is comparable to the video signal flow. The TCP flow could be from aircraft-to-store, store-to-aircraft, and store-to-store. For blinking type TCP, the signals typically flow into or out of a centralized aircraft interference blanking unit. This unit coordinates the simultaneous transfer of blanking signals among the various aircraft subsystems and stores.

The functional requirement for RF signals across the AEIS is based on several potential applications. The first RF application is the perceived

need for RF transfers from aircraft antennas/preamplifier to store receivers. These RF transfers include a class of navigation aids from systems such as Global Positioning System (GPS) and Joint Tactical Information Distribution System (JTIDS). For various reasons, access to the RF signal may be required while the store is attached to or installed within the aircraft. Due to RF obstructions from the aircraft, a store installed antenna may not have direct line-of-sight access to the RF transmitter (which is on the GPS satellite, for example). An alternate access path through the AEIS interface to an aircraft installed antenna could, therefore, be required.

The dominate requirement for transferring this signal type is for a cable that is capable of transferring up to two Gigahertz at a reasonable VSWR (less than 1.5).

A second RF application is derived from supplying a growth capability for future high bandwidth information transfers. Conceptual examples include very high data rate digital networks and multi-channel or multiplexed video buses. However, no stores presently exist which implement these high bandwidth digital concepts. It should also be noted that the AEIS includes a growth capability for fiber optic based information transfer. The fiber optic link could, for example, accommodate future high speed digital data buses, high resolution video, etc.

Figure 9 illustrates the signal flow projected for RF applications. Only store-to-aircraft and aircraft-to-store RF signal flows are anticipated. While store-to-store transfer of RF signals is possible, no applications could be projected - particularly since a fiber optic growth capability is available in the AEIS for future high speed digital transfers.

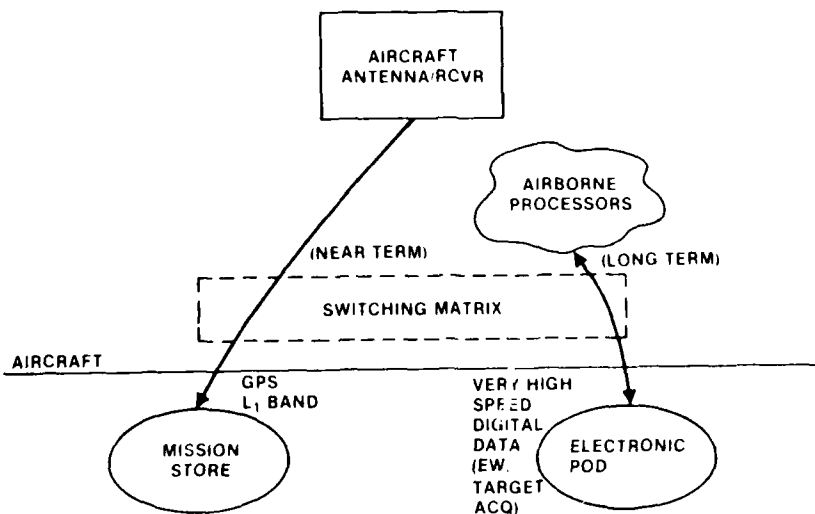


FIGURE 9 RF SIGNAL FLOW

One of the significant requirement drivers for the high bandwidth transfers concerns the composite network topology or total signal routing requirements. For a MIL-STD-1553 data bus network, access to the data bus by various aircraft subsystems can be implemented relatively easily by tapping the data bus. Similar access to the high bandwidth transfer network is more technically complicated due to the higher operating frequencies and the associated sensitivity to impedance mismatch, signal distortion, etc. Also, the end-user equipments in the aircraft for the high bandwidth signals are expected to vary considerably in terms of signal characteristics. The quantity of aircraft equipment which requires access to the high bandwidth network will vary among aircraft resulting in different levels of network switching complexity. The location of end-user equipment will also vary, affecting the potential signal losses and distortion caused by the aircraft. Figures 10 and 11 illustrate two levels of aircraft-store high bandwidth networking variations. These two variations certainly are not all inclusive.

Due to these issues plus others, the scope of high bandwidth network specification for eventual incorporation into the AEIS standard is projected to include:

- (1) Defining topology through the carriage store, i.e., CSSIs to CSI.
- (2) Defining topology in the aircraft from ASIs to an aircraft high bandwidth "switching matrix" including transfer paths between ASIs through the "switching matrix". (The switching matrix function in the aircraft can be implemented in several ways including coaxial switches, aircraft high bandwidth multiplex systems such as a video bus network, electronic repeater network, etc.).

CONCLUSION

This paper presented an overview of the signal set selected for the standard Aircraft-Store Electrical Interconnection System (AEIS) interface. This overview included background on signal derivation, application guidelines and restrictions, and several comments on areas in MIL-STD-1760 which are frequently misunderstood.

The signal set defined by MIL-STD-1760 is one of the most critical elements for developing a standard AEIS. Definition of a common set of signals for the interface between all future stores and aircraft will result in significantly lower modification costs associated with introducing new stores into the services. With the addition of a standard physical interface (e.g., connectors and cables) and standard logical interface (e.g., multiplex bus protocol and messages) to the AEIS in the near future, most remaining hardware modification costs and many software modification costs will also be eliminated.

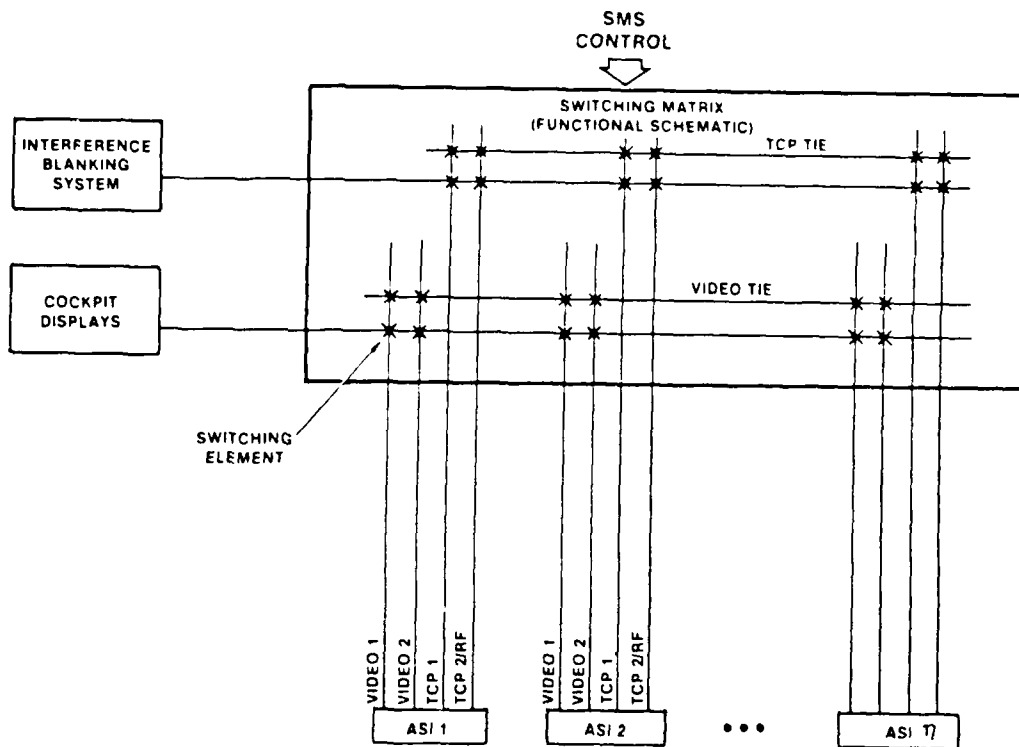


FIGURE 10 ANALOG SWITCHING MATRIX - EXAMPLE 1

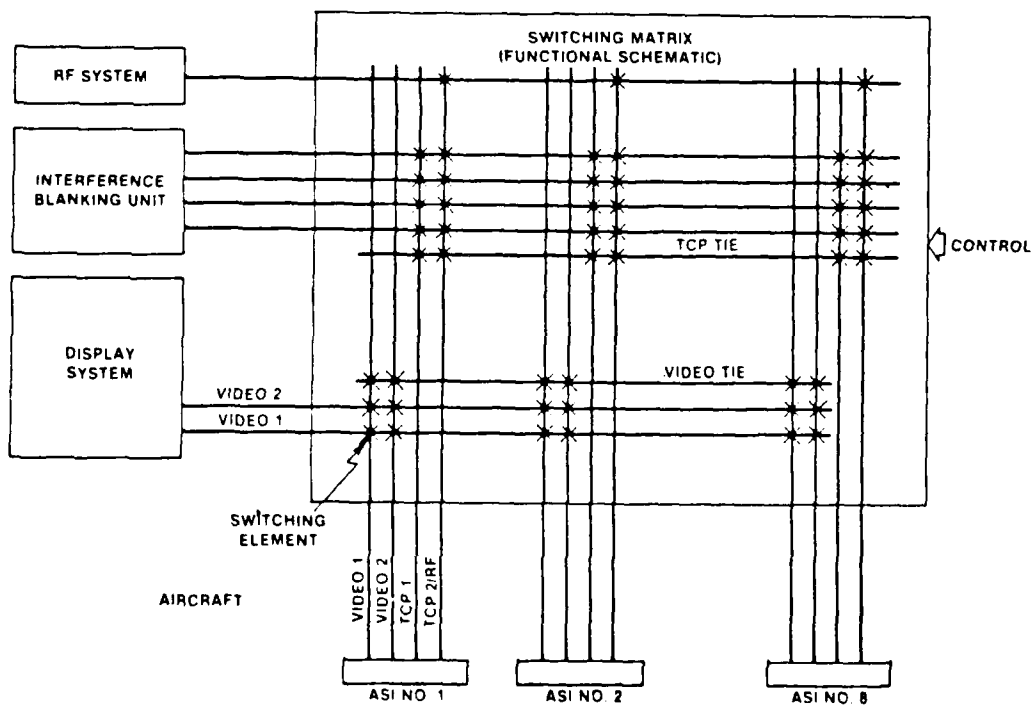



FIGURE 11 ANALOG SWITCHING MATRIX - EXAMPLE 2



Consideration of MIL-STD-1760, Aircraft/Store Electrical
Interface Standard on Stores Management System Architectures

John E. Sill

Fairchild Space & Electronics Company

Abstract

Stores Management Systems (SMS) have usually been developed to satisfy point design needs of a particular store suite on a particular military aircraft. Typically during aircraft service life the carriage of additional store types, either newly developed or existing, becomes desirable. This frequently requires major surgery to make the aircraft compatible. MIL-STD-1760 is under development to ease that problem at the aircraft/store connection point. Several implications exist for the Stores Management System (which controls that connection point on the aircraft side) if the interoperability objectives of MIL-STD-1760 are to be fully exploited. The analysis presented takes a four step approach to defining an advanced SMS that can provide the interoperability and flexibility desired.

First a top-down look into how an SMS fits into an advanced avionics system is presented. The second step is a bottoms-up look into what a multiplicity standard store interfaces imply for the SMS. Third is an analysis of other SMS requirements which must be considered. Finally, an SMS system is projected which can fulfill many of the derived requirements. The flexibility and interoperability objectives of DOD in the armament system area can be achieved but they require an integrated analysis which must address mechanical, electrical, and functional compatibility issues for both stores and aircraft. This paper addresses an important part of that analysis, the Stores Management System.

THE AVIONICS SYSTEM

Integrated Avionics is a most popular term used to define trends in avionics systems. Stripped to its barest essentials, the term recognizes the need for subsystems to be highly interactive on future aircraft. Many of these subsystems have in the past been significantly independent in function. Now we recognize that significantly improved performance can be obtained by correlating the information developed and the control processes applied by these subsystems. Three areas are particularly important.

- (1) Sensor information should be correlated from multiple devices, thereby improving the overall understanding achieved by observing the results from any one subsystem. For example, radar, infrared, and stored map information when correlated can produce a much clearer picture of what is "out there".
- (2) Control processes must also be correlated to improve performance. For example, survivability of an attack aircraft can only be maximized by integrating the flight and fire control processes. In fact, the operation of one seriously affects the capabilities and operations of the other.
- (3) Crew monitoring/control of the system has become increasingly difficult due to the amount of information presented and options possible for each of the subsystems. Unless the information is presented in an integrated, simplified form and unless the options are prioritized and kept at a high level, the flight crew is easily overloaded. Consequently, controls and displays systems are appearing which sort and correlate information before presentation to the crew and conversely assist the crew in making decisions, while minimizing presentation of options that are contradictory or undesirable.

To the SMS (as to other subsystems) Integrated Avionics implies a subsystem that is subservient to the overall avionics system rather than independently controllable by the crew. It also implies a greater capability to communicate with other subsystems and a greater capability to process information, correlating SMS performance with other subsystems, and automatically simplifying the control decisions required of the crew. It further implies that the SMS/crew interface will be less and less direct and will occur only after integration/prioritization with other data.

The primary vehicle for providing top level avionics integration is the mission computer(s), with its communication medium being the avionics data bus. Currently the bus is being implemented with a MIL-STD-1553 shielded-twisted pair technique. However, regardless of technology involved, this is a limited resource and in any top-down design a priority must be established for the information handled at this high level. The natural priority for the mission computer/avionics bus should be:

- (1) System level command and control coordinating system level shared resources (multi-purpose subsystems) and high level control of special purpose subsystems.

- (2) Exchanging data commonly required by several subsystems.
- (3) Exchanging data required uniquely by a few subsystems.

For the SMS this implies that any avionics bus connection must as a first priority serve a management and control function. Exchange of data must be relegated to a lower priority and, if required, be moved to another medium. If we are to achieve an Integrated Avionics system that has growth flexibility, top-level data bus resources cannot be totally consumed by data exchanges; a reserve must first be set-aside for additional management and control functions.

Examining current SMS systems, we can find several instances where the required signal interfaces do not lend themselves to data bus usage (e.g., weapon videos, radar/missile synchronization) due to bandwidth and latency considerations. Projecting requirements to future digitized stores we can foresee instances where very large blocks of data (e.g., digitized target images) must be transferred. In either of these cases we are looking at requirements generally on the third level of avionics bus priority: unique subsystem-to-subsystem information transfers. It is unlikely that the avionics bus of the future, regardless of technology, could satisfy these requirements.

Figure 1, SMS/Avionics System Scenario, illustrates the concepts put forward thus far:

- There are (will be) multipurpose subsystems providing top-level avionics management and control, both automated and via the crew.
- There are (will be) special subsystems for particular functions; however, their operation must be highly integrated.
- The avionics bus provides the first level medium for system integration and consequently its first priority must be management and control.
- The nature and size of certain SMS information interfaces with other subsystems suggest that use of the avionics bus is impractical and these requirements must be satisfied elsewhere.

In addition, the figure introduces the next topic for consideration: the impact of a multiplicity of AEIS interfaces.

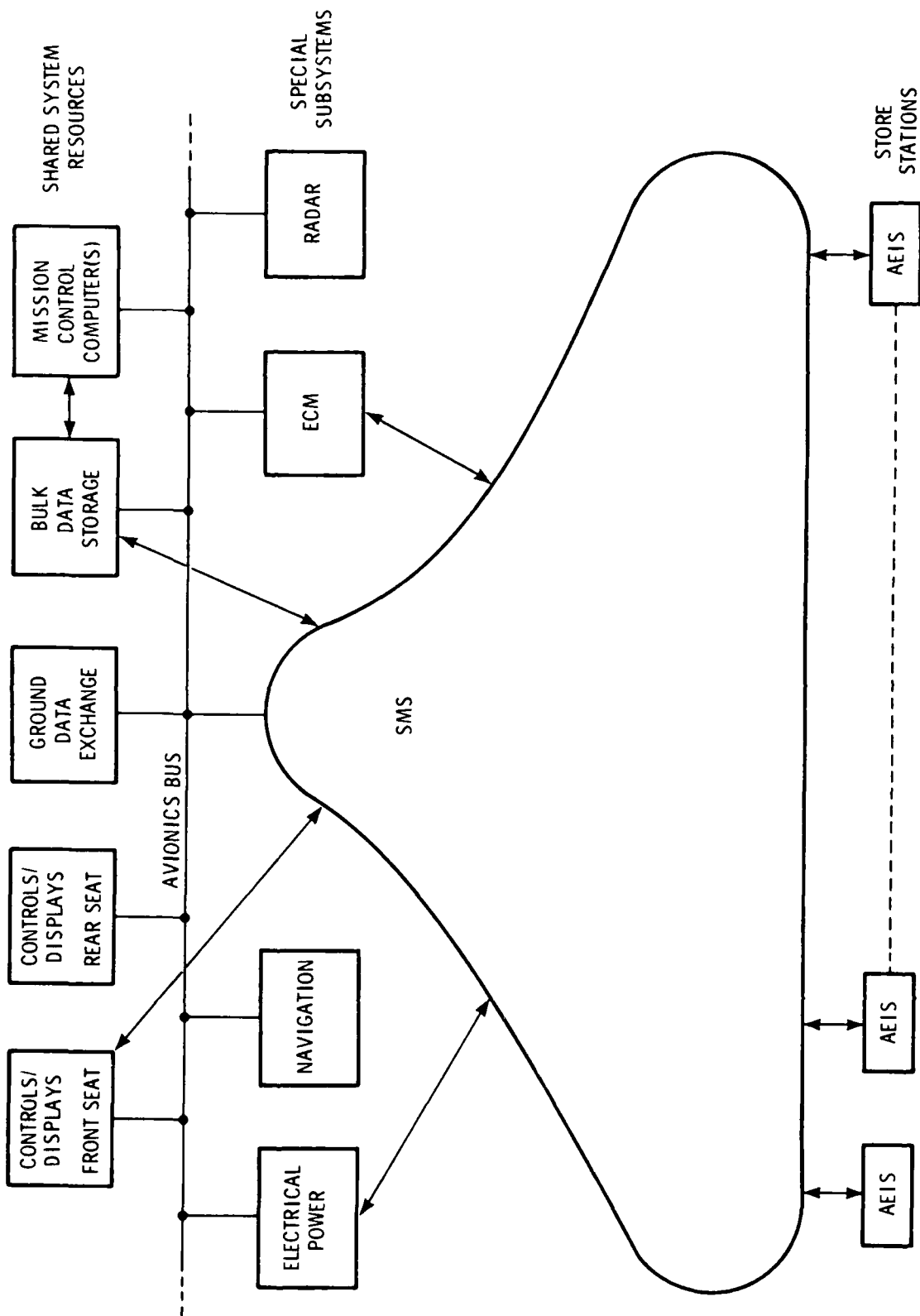


Figure 1. SME/Avionics System Scenario

THE STATION INTERFACES

The AEIS defines an interface(s), a multiplicity of which defines the bottom-up requirements for future Stores Management Systems. The objective of AEIS is to standardize this interface, providing aircraft/store interoperability and thereby ease of reconfigurability. The objective will only be partially achieved unless the system behind AEIS, the SMS, supports the standardization, flexibility, and reconfigurability of the interface. Implications for the SMS exist in both the types of stores carried and in the nature of the AEIS signals themselves.

Stores

For purposes of this discussion utilizing AEIS stores has been subdivided into three generic categories: Single weapons, multiple weapons (usually mounted via carriage equipment), and pods. Although not precise categories, these are sufficient to address most SMS requirements accurately.

Single weapons, connected via AEIS, require management and control, operating power, and varying amounts of information (data) on which to operate or to be monitored. Although not always the case in current aircraft, management and control should be established by the SMS; a further detailing and coordination of the management and control functions assigned to the SMS over the avionics bus. The data bus capability within AEIS is a natural vehicle for this, again extending the top-level system concepts. The information (data) exchange presents a situation similar to that discussed above for the SMS/Avionics system interface. Some of this information can be assigned to the data bus, since it is a natural outgrowth of information supplied to the SMS over the avionics bus or generated within the SMS management and control function. Other information is directly exchangeable with an associated subsystem (radar missile/radar subsystem) with no "value added" by the SMS. Information which is signal characteristic compatible (low bandwidth) and which does not threaten the higher level management and control requirements can be placed on the data bus. However, this suggests that a direct connection of the data bus to the associated subsystem is desirable. Finally, certain information is of such a high bandwidth that only a direct subsystem to weapon connection is possible with foreseeable technology. AEIS has provided for these with coaxial and audio (and probably fiber optic) capabilities.

Implications for the SMS from the single weapon category are:

- (1) The SMS exercises its management and control function over the AEIS data bus.
- (2) Certain information (data) is exchanged between the SMS and weapon over the data bus.

- (3) Other information is exchanged between the weapon and other aircraft subsystems via the data bus (participation by the SMS in other than a possible bus management role is not required).
- (4) High bandwidth connections between the weapon and other aircraft subsystems must be directly established (again participation by the SMS in other than a control of function is not required).
- (5) Since the AEIS is an interoperable/reconfigurable interface, the aircraft subsystem connected to may change as the type of weapon changes.

(The discussion above concentrates on the data bus and high bandwidth elements of AEIS; other aspects will be addressed later.)

The category of multiple weapons/carriage stores has many aspects similar to those discussed for single weapons. With multiple weapons (each AEIS compatible) interfaced to the aircraft via a single AEIS, at least two approaches seem feasible:

- (1) An intermediate control/distribution point is established (most likely in the carriage equipment).
- (2) AEIS signals are distributed in parallel to all weapon interfaces.

The first approach has advantages in a more straight forward AEIS interface implementation, particularly in the weapons and in the data bus addressing capability. However, it requires additional electronics in the carriage equipment, and for the data bus situations, introduces additional bus control/latency problems.

The second approach overcomes these disadvantages but requires that interface electronics in the weapons be at least on/off programmable - much like a tri-state devices on microprocessor peripherals. It also introduces complexity in the data bus addressing scheme.

Problems in either of the approaches seem surmountable and this is an area requiring further investigation. Consideration must at least be given to the fact that one station (AEIS) may in fact respond to several bus addresses.

The third category of stores, pods, introduces further complexity. These devices are typically singly mounted (one per AEIS), however, they are frequently an extension of another aircraft subsystem, a complete stand-alone subsystem, or extensions of several subsystems. Variations range from the Navy's Tactical

Air Reconnaissance Pod System (TARPS), to the Air Force's Low-Altitude Navigation Targeting Infrared for Night (LANTIRN). TARPS can be considered a self-contained subsystem requiring only high-level management, control and data, while LANTIRN has elements of sensors, flight controls, fire control, and stores management.

Implications for the SMS exist in the fact that the device on the other side of the AEIS may not be SMS related at all (excepting the consideration that overall management and control of all store stations should be an SMS function). Hierarchically the pod subsystem may need to be linked directly to the avionics bus for top-level management and control. Just as likely it may need to be linked to another aircraft subsystem of which it is a part. There may be no "added value" by passing this information through the SMS. An additional requirement is imposed (LANTIRN is one example) when two pods work together to perform a function or when pods contain control systems for particular weapons mounted on other aircraft stations. Consequently, an additional implication for the SMS is the ability to directly connect two or more AEIS interfaces.

Throughout the preceding and following discussions are references to the need for information transfer between stores and aircraft subsystems. The SMS is only one of these subsystems. Contrary to current approaches, one avionics bus linking all major substations to the SMS will not be a sufficient solution to tomorrow's problems. The flexibility/interoperability inherent in an AEIS approach strongly suggests that the interconnect system behind AEIS also be flexible.

The interconnection of stores-to-stores-to-other subsystems and the control of that interconnection is an SMS responsibility. Any other approach will result in a break-down of the hierarchical distribution of avionics requirements to subsystems and will endanger the interoperability objectives of the program. To achieve this interconnection flexibility the SMS needs a capability.

Figure 2 illustrates the Matrix Switch concept. In theory it is a generic capability to interconnect any of several AEIS interfaces with any of several subsystem interfaces. The capability of this device is bounded by AEIS signal definition, one of the knowns of the problem. Subsystem interfaces can be considered an unknown at this point, partly because a prime objective is to be able to adapt to new subsystems. However, a subsystem interface must be "in-scope" relative to AEIS; if not we are dealing with a non-standard store.

The complexity of the Matrix is further reduced by many "null set" interconnections. For example, electrical power is only a connect/disconnect situation, not switchable to other subsystems. Further simplification is likely and is addressed in paragraphs below.

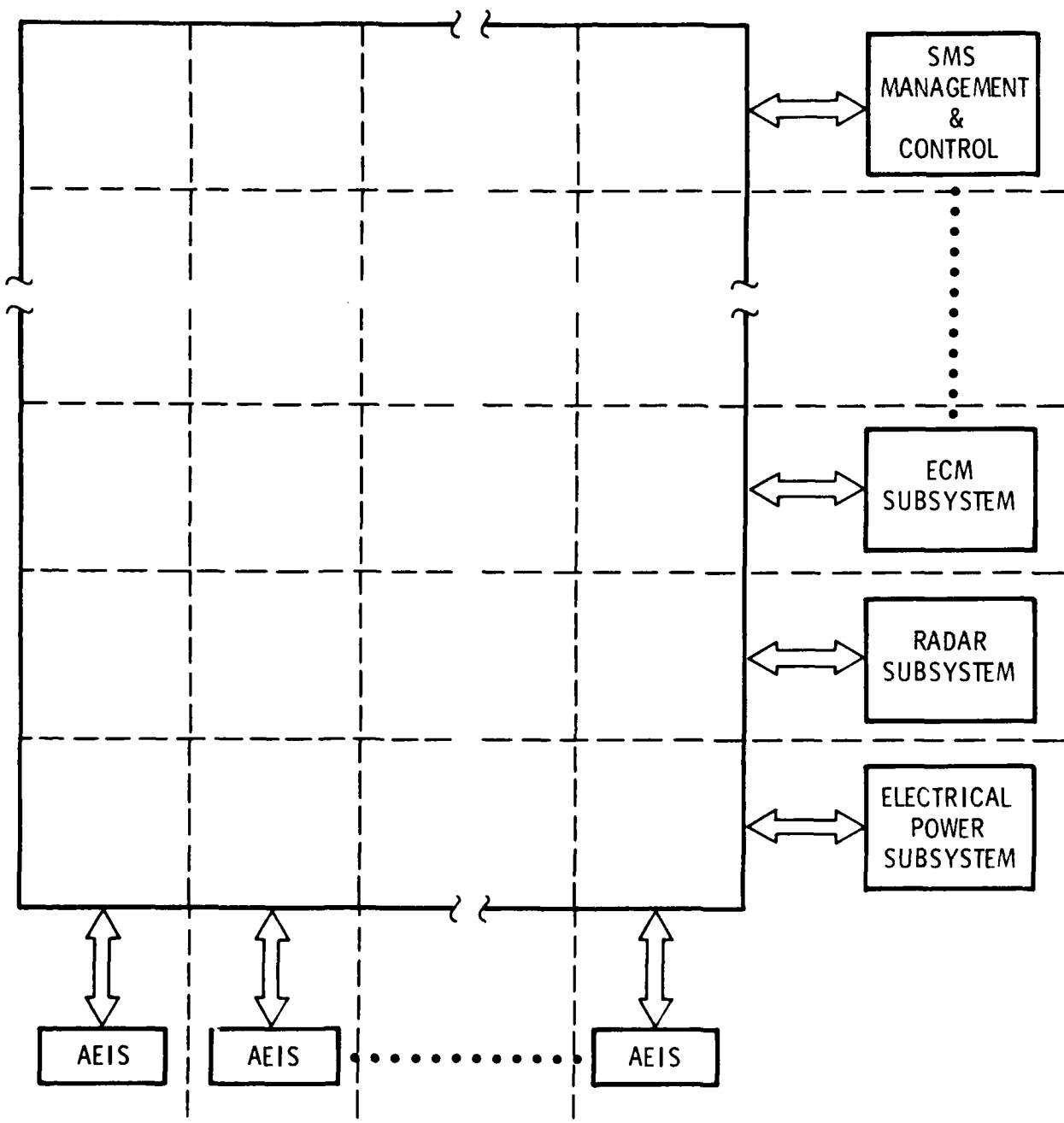


Figure 2. Matrix Switch Concept

It is noteworthy that SMS Management and Control is one of the subsystems interfaced through the Matrix. There may in fact be more than one SMS subsystem interface (multiple armament buses). The approach provides the capability to link-up one or more stores and one or more other subsystems under SMS management/control. This will allow configurations such as LANTIRN (where two pods and two Maverick launchers occupy four stations) to be linked-up into a mini-subsystem for that mission while other stations and subsystems can be simultaneously linked up for other functions such as self-defense. The computer data busing and matrix switch capability installed in any aircraft can be varied in accordance with mission(s) complexity.

Before proceeding to investigate SMS implications inherent in the nature of AEIS signals themselves (some considerations have already been suggested), a summary of requirements disclosed thus far is in order. Figure 3, Primary SMS Requirements, illustrates and integrates the following items:

- (1) The avionics bus provides the top-level linkage of aircraft subsystems - management and control must have priority over data exchange.
- (2) The SMS is one of the subsystems controlled via the Avionics Bus and also exchanging limited data thereby.
- (3) Primary SMS requirements fall into two categories: Subsystem Management/Control and Matrix Switching.
- (4) Due to signal characteristics or information volume non-avionics bus linkages must be set-up for/by the SMS.
- (5) Elements interfacing to the SMS Matrix Switch include the Store Stations, as represented by an AEIS, and various aircraft subsystems.
- (6) The reconfigurability desired by AEIS suggests that these linkages be reprogrammed as store loading suites are changed.

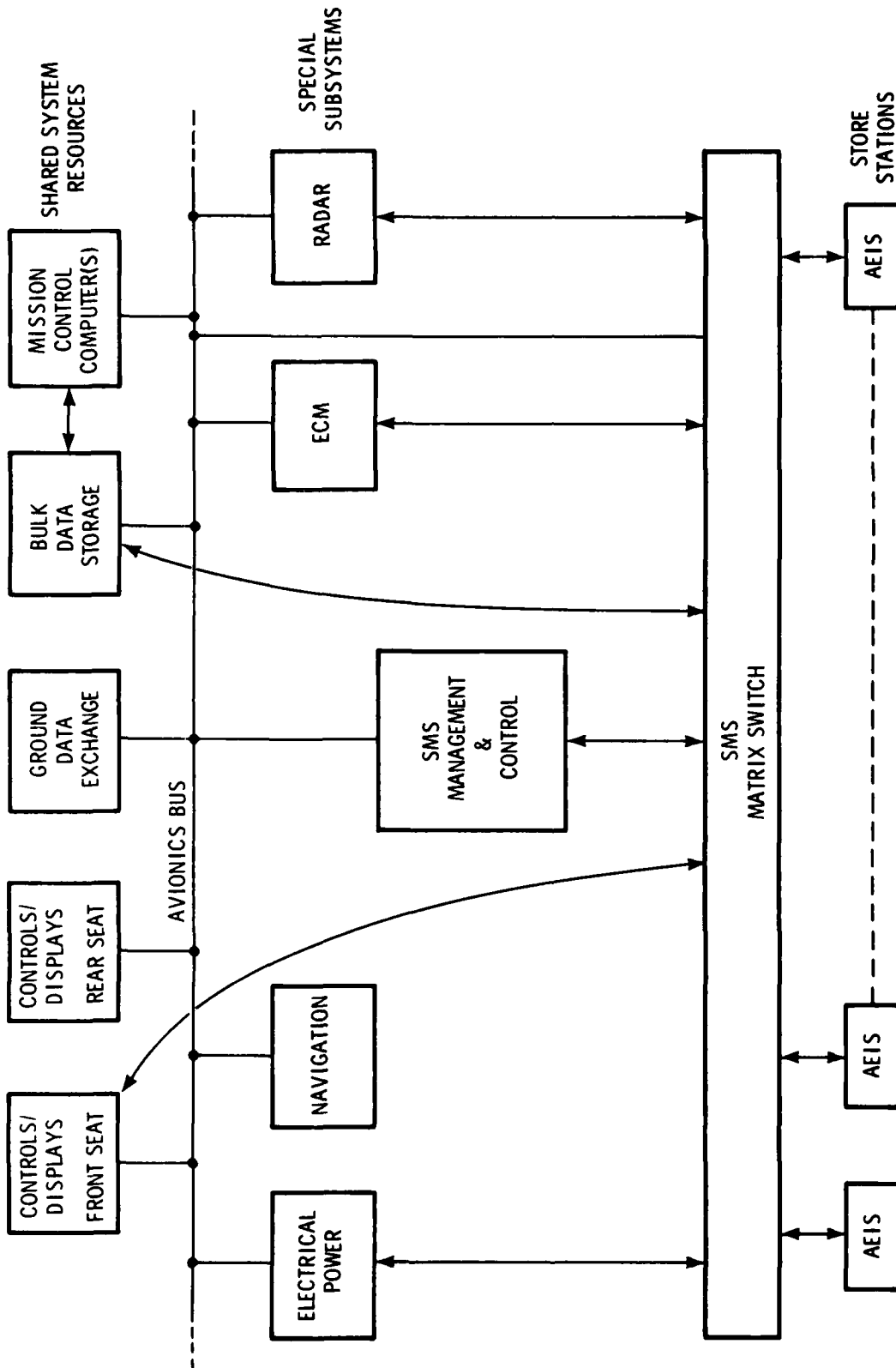


Figure 3. Primary SMS Requirements

- (7) AEIS to AEIS linkages must also be possible through the matrix.
- (8) The need for Avionics Bus and SMS Management/Control (Armament) Bus connections through the matrix is suggested.

AEIS Signal Analysis

Having reviewed the devices (stores) on the "other side" of AEIS for implications on the SMS, the nature of the electrical interface itself is worth consideration. This is especially worthwhile relative to the Matrix Switch aspect of SMS. Some of the AEIS signals lend themselves very easily to a Matrix Switch concept, for others it is more difficult, and for still others it seems unnecessary or inappropriate. Figure 4, summarizes AEIS Signal Classes versus Matrix Switch considerations, while the discussion following provides more detail discussion and rationale.

Power and Returns

Power distribution imposes only limited requirements on the Matrix Switch. Power "flows" in only one direction and has basically only the Aircraft Electrical Subsystem as a source. (Albeit the Electrical Subsystem may be redundant, have back-up systems, or have emergency only systems.) One of the key decisions in this area is the control point. If the application of power is controlled at each station (AEIS), then a power bus(es) can be run through the aircraft with taps at each AEIS. This will reduce aircraft harness requirements but increase the probability of single point failures causing major loss of function. On the other hand, a central switching point requires individual runs to each AEIS and, unless it is redundant, still provides single point failure possibilities, but with reduced probability. As a baseline approach let us follow inferences from MIL-STD-1760, with each of the four AEIS power types (28VDC-1, 28VDC-2, 3ØAC, 270VDC) independently switched at the station. Power can then be bused to the stations, providing the most harness-efficient configuration. A portion of the Matrix Switch is thus implemented at each store station.

2.1.2.2.2 Bus Address

In consideration of MIL-STD-1553 practices and aircraft safety, the bus address is determined by the aircraft harness at the station and is therefore not a matrix consideration. This decision limits possibilities in on-board bus reconfiguration.

MATRIX SWITCH IMPLICATIONS

AEIS SIGNAL CLASS

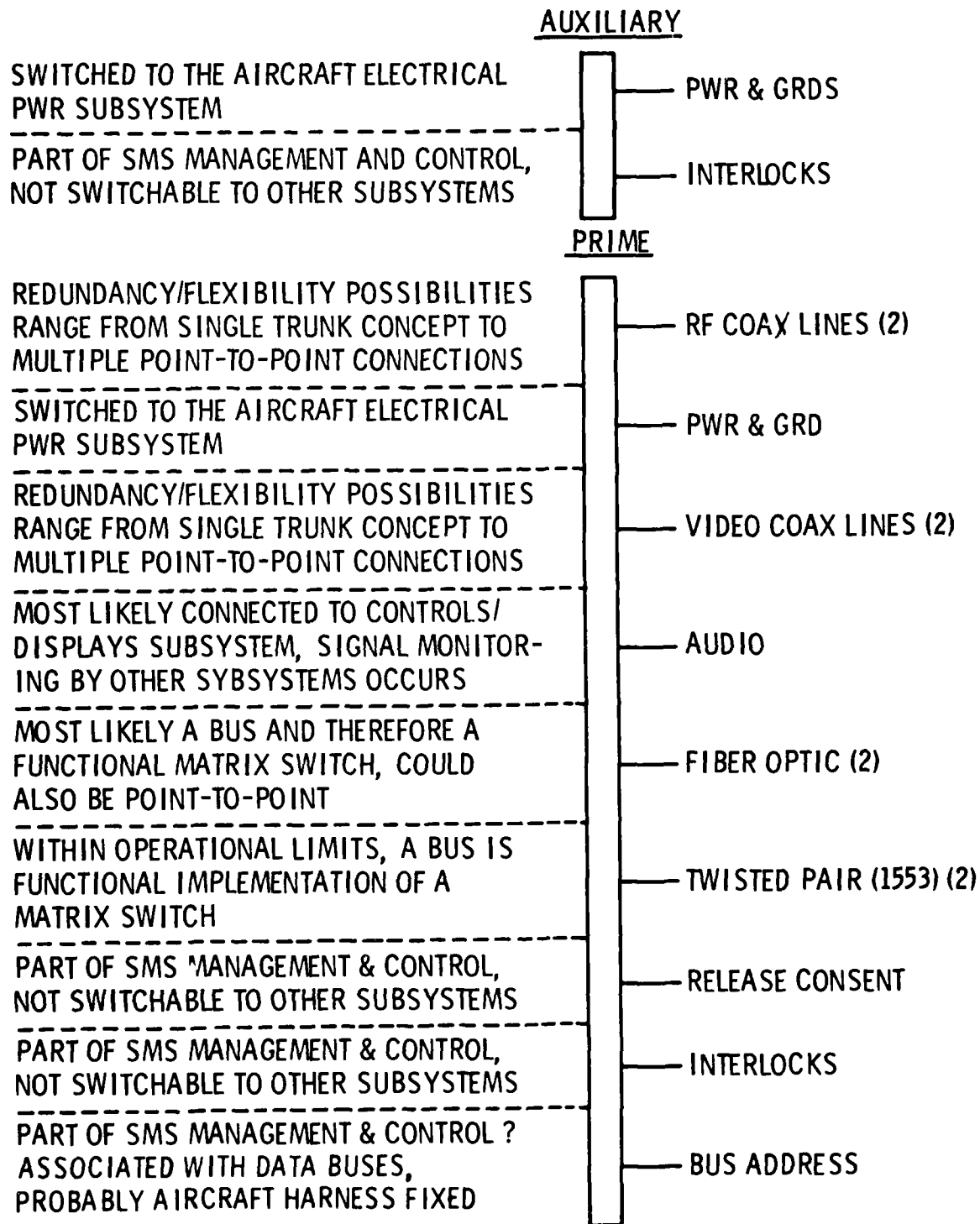


Figure 4. AEIS Signal Classes versus Matrix Switch Considerations

Twisted Pair and Fiber Optic Buses

A data bus system along the lines of MIL-STD-1553 is functionally a matrix switch. That is, information can be passed to/from any device or combination of devices on the bus. However, the bus is operationally limited and the desired information flow capability must be considered. A true matrix switch installation would allow only those stations and subsystems operating together to be connected together into a bus. The full information transfer capabilities of that bus are then available for that requirement. Although a matrix switched bus(es) seems an avant-garde approach, it should not be arbitrarily discarded. The interoperability objectives of AEIS require that for one flight a station (AEIS) may be carrying a radar missile and need an efficient connection to the radar subsystem. On the next flight the station may be carrying a LANTIRN pod and need an efficient connection to other stations and to the Inertial Navigation subsystem. The most straight forward approach is one bus that interconnects all anticipated points (terminals). However, this may not be the most flexible (due to information flow rate limitations) or effective technique. Additionally, the number of terminals on the bus is limited and if we want to keep open the possibility of extending the bus directly to multiple weapons at one station, the number of terminals becomes a limiting factor. For the moment let us reserve the option of putting the data bus through the SMS Matrix Switch and therefore the capability of reconfiguring armament buses to best fit store loading configurations. Other factors may help decide this issue.

It should be noted that conceptually the AEIS data bus carries SMS Management and Control information, limited data as a flow-down from the Avionics Bus, and quite probably large data transfers between a store and other stores or aircraft subsystems.

Discretes

The Interlocks and Release Consent are considered a part of the SMS Management and Control aspect of the interface and therefore not switchable to other subsystems. Therefore they are not put through the Matrix Switch per se. However, the information that controls them may in fact be passing through the matrix.

Coax Lines

The video and RF coax lines can place the maximum burden on the Matrix Switch concept. The optimum flexible/interoperable approach suggests routing each of these signals directly to a central switching point where any station signal can be connected to any other station or to other aircraft subsystems. The opposite extreme suggests that one trunk line be established for each of these four signals and that each station or subsystem be switchable on or off this trunk. Thus, only four different

point-to-point connections can be set-up at any one time. The latter is the most aircraft harness efficient approach but is quite capability limiting. The best solution probably lies somewhere between these two extremes.

Audio

The single trunk concept is probably acceptable for the audio line with connect/disconnect to the trunk performed at the station. This limits the crew to "listening" to only one store at a time, yet allows sampling of each of the AEIS audios sequentially. Since the audio line is frequently monitored automatically, the trunk should be routed through the SMS at a central point.

ADDITIONAL REQUIREMENTS

In the preceding paragraphs, SMS requirements have been viewed from the Avionics System downward and from the AEIS upward. It is now appropriate to consider other additional requirements. Most significant of these are two additional interface/control requirements at the stations. These are: management and control of the parent rack and provisions for non-standard (current) stores. These items are different in concept in that a parent rack interface is a recognized and accepted adjunct to an AEIS while a non-standard store interface is an unfortunate and hopefully temporary requirement. At least on a theoretical basis, two distinct requirements exist.

- (1) Standard Interface Equipment (SIE). This item addresses the long term needs of the system - providing the parent rack interface, local circuits for interfacing AEIS signals on to and off of the data bus, and implementing those elements of the Matrix Switch which are best located at the station. The intent of this device is to fully provide for station associated AEIS requirements.
- (2) Interim Store Adapter (ISA). This item provides for non-standard store interface, probably through adaptation of AEIS.

Figure 5 helps to clarify the concepts involved. The SIE, parent rack, and ISA are illustrated as discussed above, however, other concepts are also involved. The Matrix Switch has been added to the figure to illustrate its relationship and to highlight that some aspects of it may be located in the SIE.

The Interim Store Adapter is located below the ASI for conceptual purposes. In practice there are several possibilities for implementation as illustrated on Figure 6. Selection

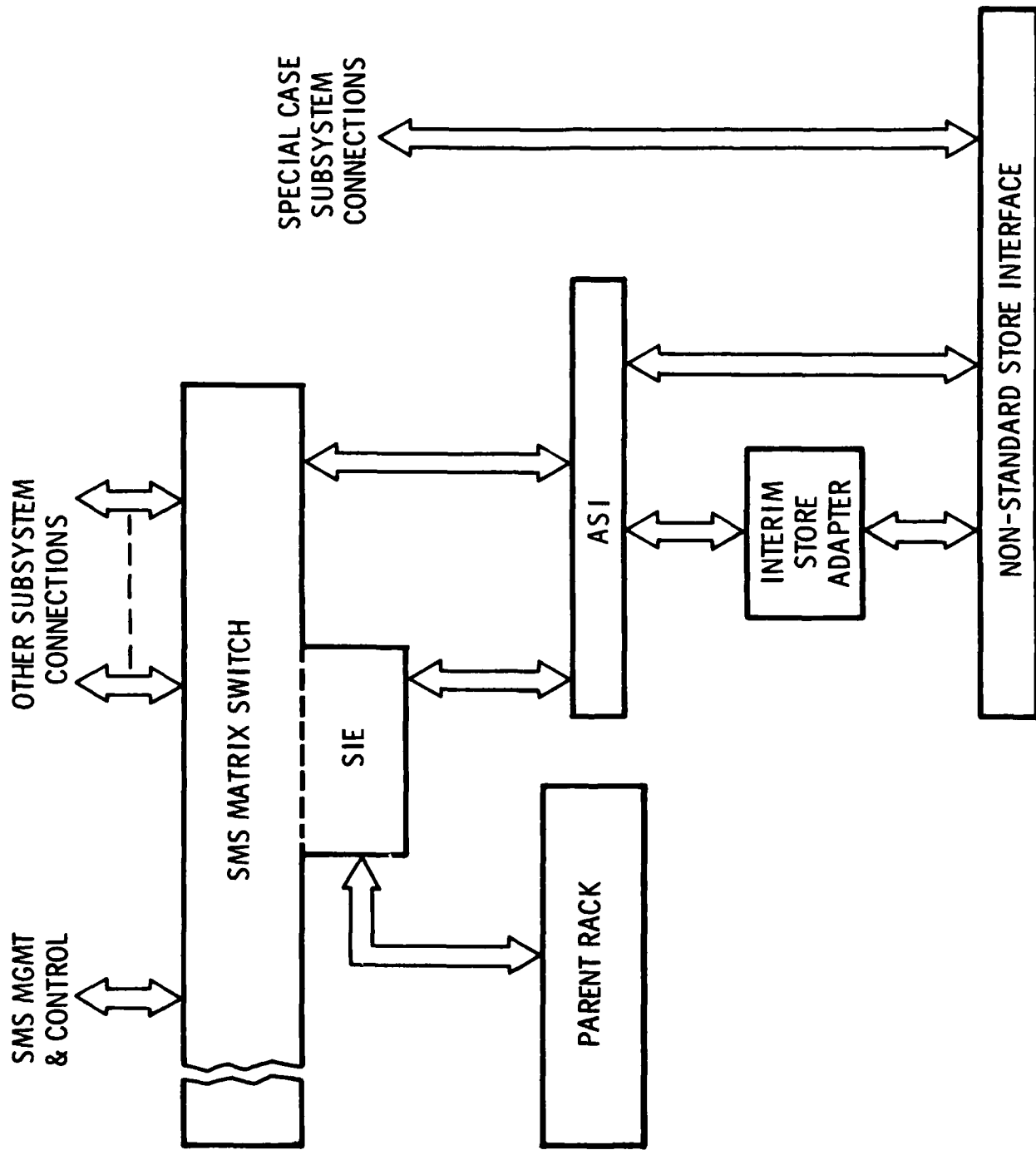


Figure 5. Establishing Non-Standard Store Interfaces

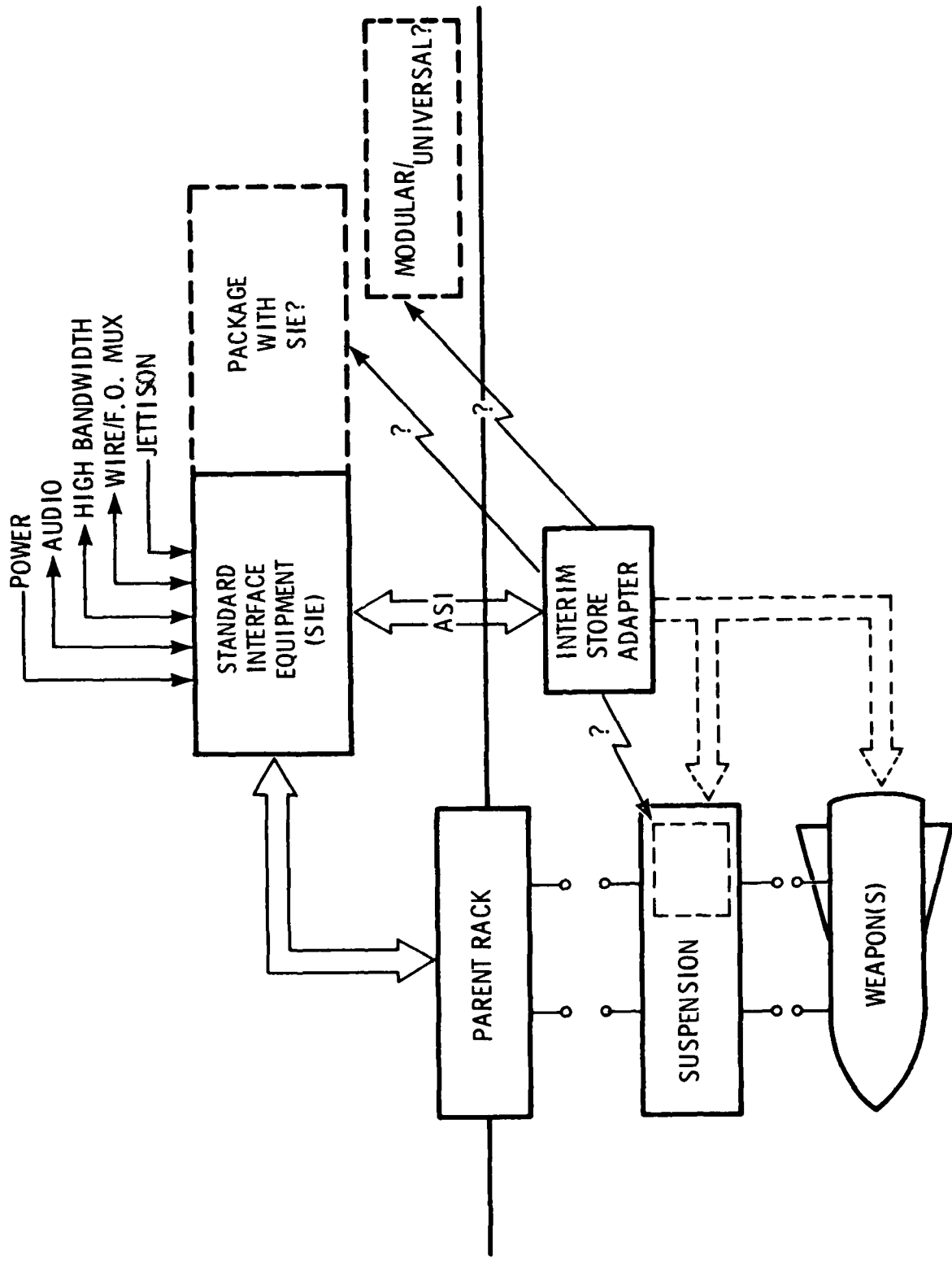


Figure 6. Store Station Equipment

of an approach probably requires evaluation of the particular aircraft and stores involved due to size, weight, power and cost considerations.

Adaptation for a non-standard store follows a three step priority.

- (1) Use any ASI signals which are compatible (e.g., power, coax, audio).
- (2) Adapt, via an ISA, for those signals which can be converted to ASI compatibility (e.g., discrettes, low bandwidth analogs, power).
- (3) Provide special aircraft harness connections for those signals which are totally incompatible with ASI.

Many non-standard stores can be adapted to the data bus of ASI, however, a store-by-store interface analysis is necessary to accurately scope the problem.

THE PROJECTED SYSTEM

Having analyzed major functional requirements for SMS implications, we can now postulate a likely system approach. Figure 3, Primary SMS Requirements, was discussed previously and illustrated two major SMS elements: Management and Control, and Matrix Switching. Subsequent discussions have pointed out that some of the Matrix Switching may be best accomplished in the Standard Interface Equipment (SIE) while other Switching is best accomplished in a central unit. We have seen that certain aspects of the Management and Control element need interface circuitry located in the SIE. We have also seen that the Management and Control element is a flow down of responsibility from the Avionics System, suggesting some (single) interface/control point at the highest SMS level.

Figure 7, the Projected System brings the concepts together into a single, high-level, block diagram. A multiplicity of SMS Management and Control Elements are programmably interconnected with a multiplicity of Aircraft Store Interfaces and with a multiplicity of subsystem interfaces. The Matrix Switch provides the programmable interconnection. An additional feature is apparent in the redundancy/fail-soft nature of the configuration. This starts with a redundant linkage to the higher level system, and requires that Management and Control Elements have some capability to perform each other's functions. A trade-off of reliability vs redundancy vs cost is beyond the level of this paper as it quickly leads into implementation issues. It must be sufficient at this point to simply say that the concept adapts reasonably well to those considerations if we accept the premise that loss of a single store station or single subsystem interface is acceptable, providing that other capabilities of the SMS are still available.

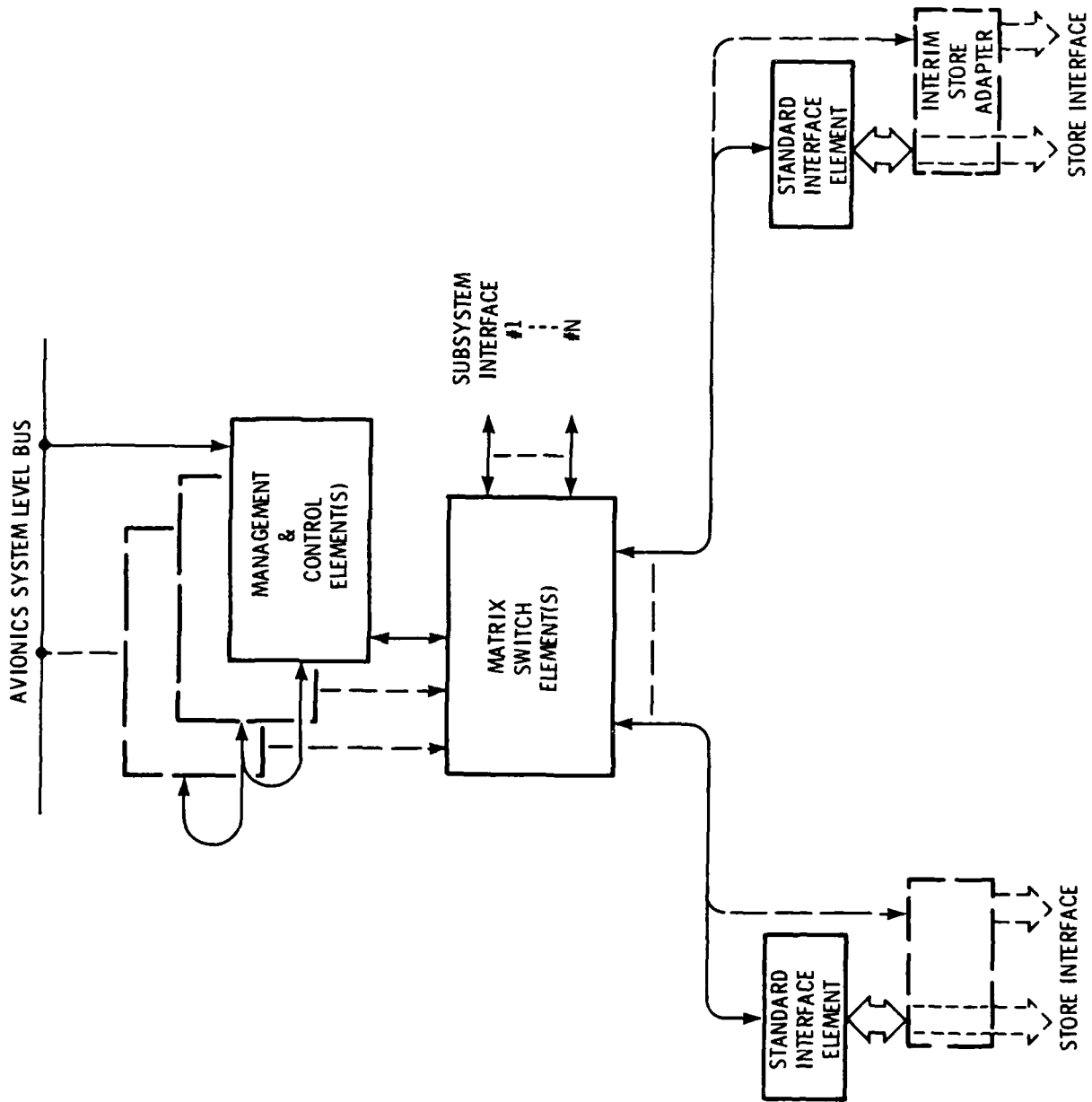


Figure 7. Projected System

As an aid to further understanding the system concepts put forward let us look at Figure 8, A Mission Configured Projected System. Here one of the Management and Control Elements has been assigned dual responsibilities of overall SMS Subsystem Management and direct control of one store (psuedo type "B") and its associated control pod. Two other Management and Control Elements provide control of store types "A" and "C" respectively. Each of these stores require a direct interconnection to one or more external subsystems (e.g., Radar, Mass Memory, Electronic Counter-Measures).

Nowhere does the Matrix Switch concept appear on the figure. That is because it is no longer a functional element of the SMS. It has performed its task in configuring the general capability inherent in Figure 7 into the specific mission architecture of Figure 8. The Matrix Switch will not come into play again until the system is reconfigured for a different mission (i.e., store suite).

CONCLUSIONS

Achieving the interoperability and flexibility objectives of MIL-STD-1760 will have a double impact on the Armament System designer. First, a standard electrical, mechanical and functional store interface will ease the designer's burden and provide a major step forward in interoperability. Second, it will challenge the designer to provide a system behind that interface that fully supports and exploits the adaptability desired in overall aircraft capability.

This paper has explored several problem areas and presented a system concept supportive of the standard. Even if the SMS system designer adopts this, many challenges remain. The biggest of these is balancing a flexible SMS implementation (with its inherent cost/size/weight penalties) against an efficient point design system which will quickly restrict the interoperability and flexibility goals of MIL-STD-1760.

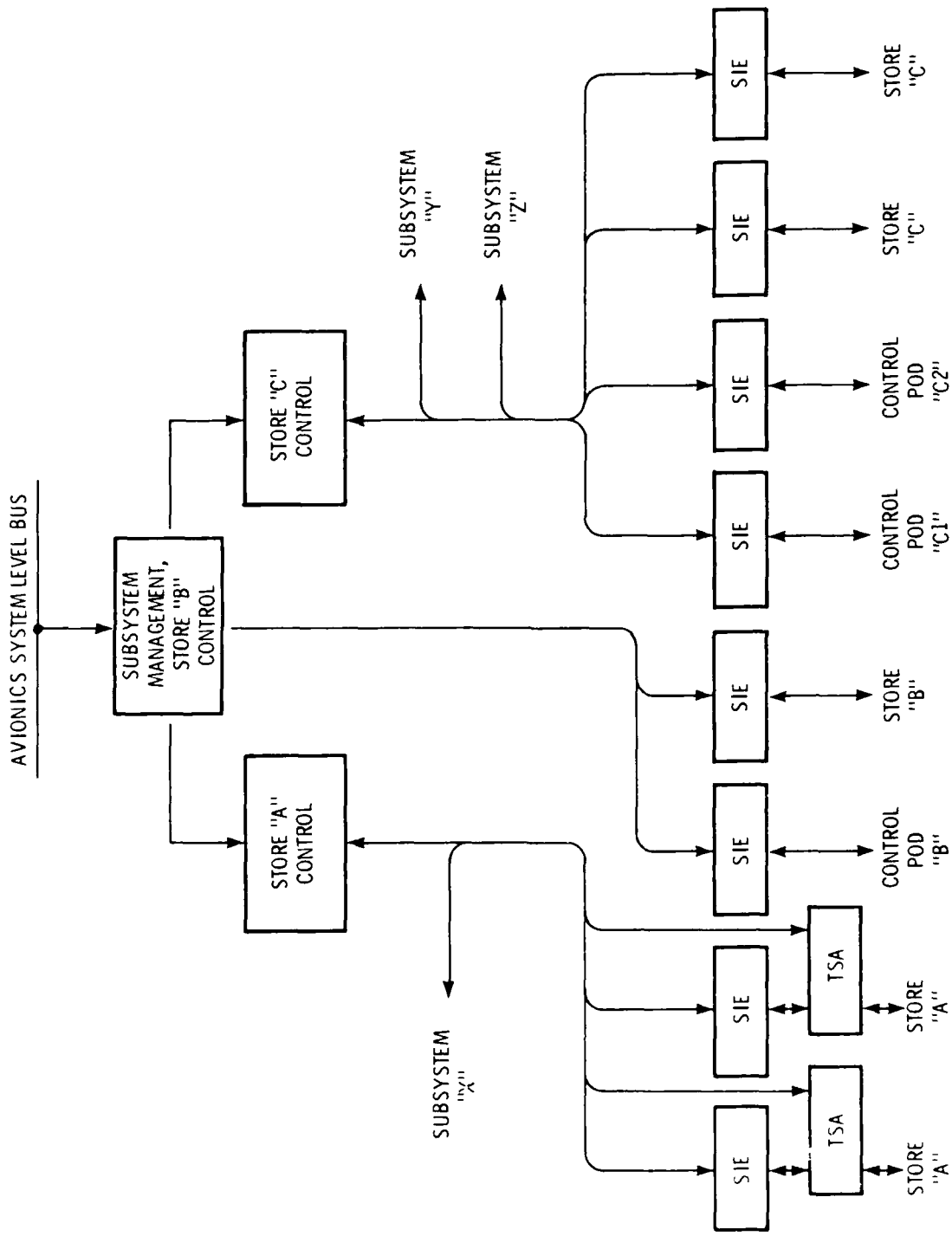


Figure 8. Mission Configured Projected System

JOHN SILL

Mr. Sill is currently Director of the Electronic Products Engineering Department at Fairchild Space and Electronics Co. in Germantown, Maryland and is responsible for the systems analysis, specification and technical approach selection for all Fairchild Avionics products. These include Data Processing, Multiplexing and Display, Stores Management, and Data Annotation. He has also been Program Manager for IR&D projects. These programs placed particular emphasis on SMS-related problems.

From 1973-1974, Mr. Sill was Project Engineer during development of the Armament Control System for the A-10 aircraft. His responsibility included electrical, mechanical, test and support engineering aspects and subcontract technical supervision. Prior involvements as a Senior Engineer included the F-111D Stores Management Set development; S3-A Avionics Definition study, and proposals for B-1 and F-15 avionics subsystems.

Mr. Sill has a BS in Physics from Canisius College and has completed graduate coursework at Georgetown University.

AD-P003 530

AIRCRAFT MULTIPLEXING SYSTEM
ARCHITECTURE AND STORES COMPATIBILITY

By

A. DeRuggiero
Grumman Aerospace Corporation
Bethpage, N.Y. 11714

and

B. Zempolich
Avionics Directorate
(Air 33) NAVAIR

The introduction of more capable and more complex vehicle avionics systems and the emphasis on the aircraft as a holistic system has led to a trend toward distribution of subsystem functions and a greater need to provide effective control over total system operation.

NOTE: The opinions expressed herein are not necessarily those of the United States Navy.

The subsystems impose severe loads which affect the control function. The stores subsystem, in particular, requires command, data and signal loads that are capable of overwhelming the system information flow:

- Power, data, command and signal protocols as specified by MIL-STD-1760 impose significant speed and control logic loads on vehicle information traffic
- The availability of massive amounts of data and the requirement to allow complex store and carriage store operations cause a risk of crew overload
- The protocols of MIL-STD-1553B allow high speed message transmission primarily in one direction. Those store subsystem components which require dialogue, impose special loads on message handling procedures
- The very existence of other subsystems, all competing for attention and direction; and some of which having data, messages or commands in common (as, for instance, the fire control and stores management systems) demands subsystem interface control that allows smooth communication flow and yet does not hinder the proper operation of any subsystem.

SYSTEM-WIDE CONSIDERATIONS

The designer of future systems, regardless of the architectural and functional levels of structure and organization, is faced with the need to incorporate system effectiveness into the system. The following examples are but a few of the overall system-wide considerations that must be factored into the proposed system at the initial definition stage of the program:

- Multi-function capability as, for example, the ability to change from bus controller to remote terminal and back again
- Useful redundancy, providing both backup and the ability to modify outputs,
- Component availability to the system in each of the operating modes
- Flexibility to change the operating mode as required.

With the proper incorporation of these features, the classic military systems problem of "capability versus numbers" will be resolved, or as a minimum, greatly reduced. With this approach, a minimally acceptable, adaptable systems capability can continue to exist throughout the mission, regardless of the degree of sophistication and dynamically changing characteristics of the threats. However, even with this increase in systems effectiveness, numbers of systems will still be important, as will system life-cycle costs. As a result, designers will be attempting to make the best use of available resources by providing compatibility with new components and subsystems as they are developed, while, at the same time, keeping maximum compatibility with existing operational resources already deployed. This approach will require even greater complexity in each new system, subsystem, or component if "satisfactory operational capability" is to be obtained.

A case in point is the current requirement for a vehicle-store interface that can handle new stores incorporated onto new vehicles. At the same time, the interfaces must also be backward compatible so as to be able to handle "older" stores that are on existing aircraft. The requirement for forward fit and backward compatibility initiated the definition of military standard MIL-STD-1760. This standard has been under requirements definition/development for several years. While the interface logic is not complete at this time, the physical and electrical characteristics (Fig. 1), which are completed, illustrate the complex problem of matching current systems with future interconnection requirements.

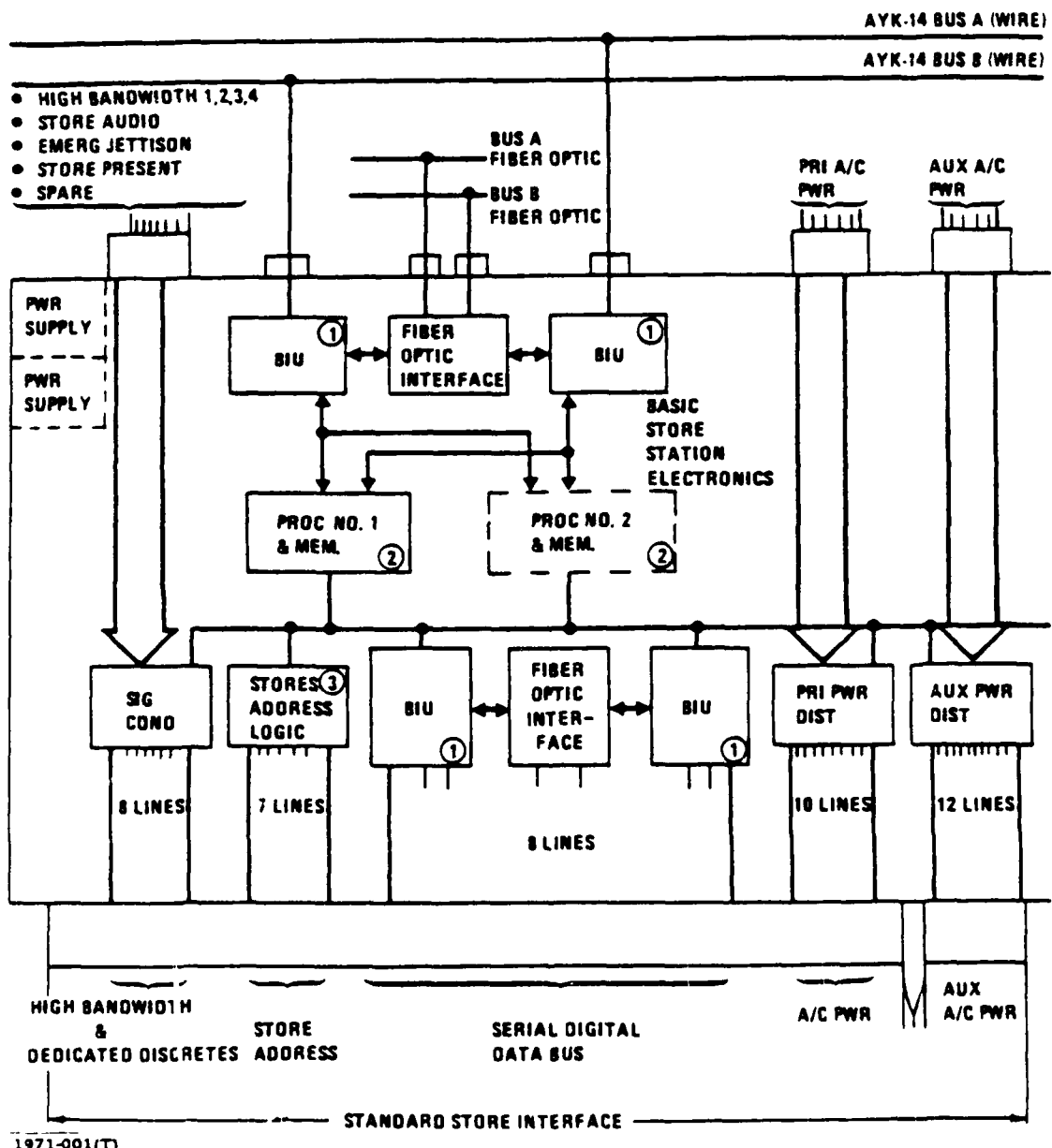
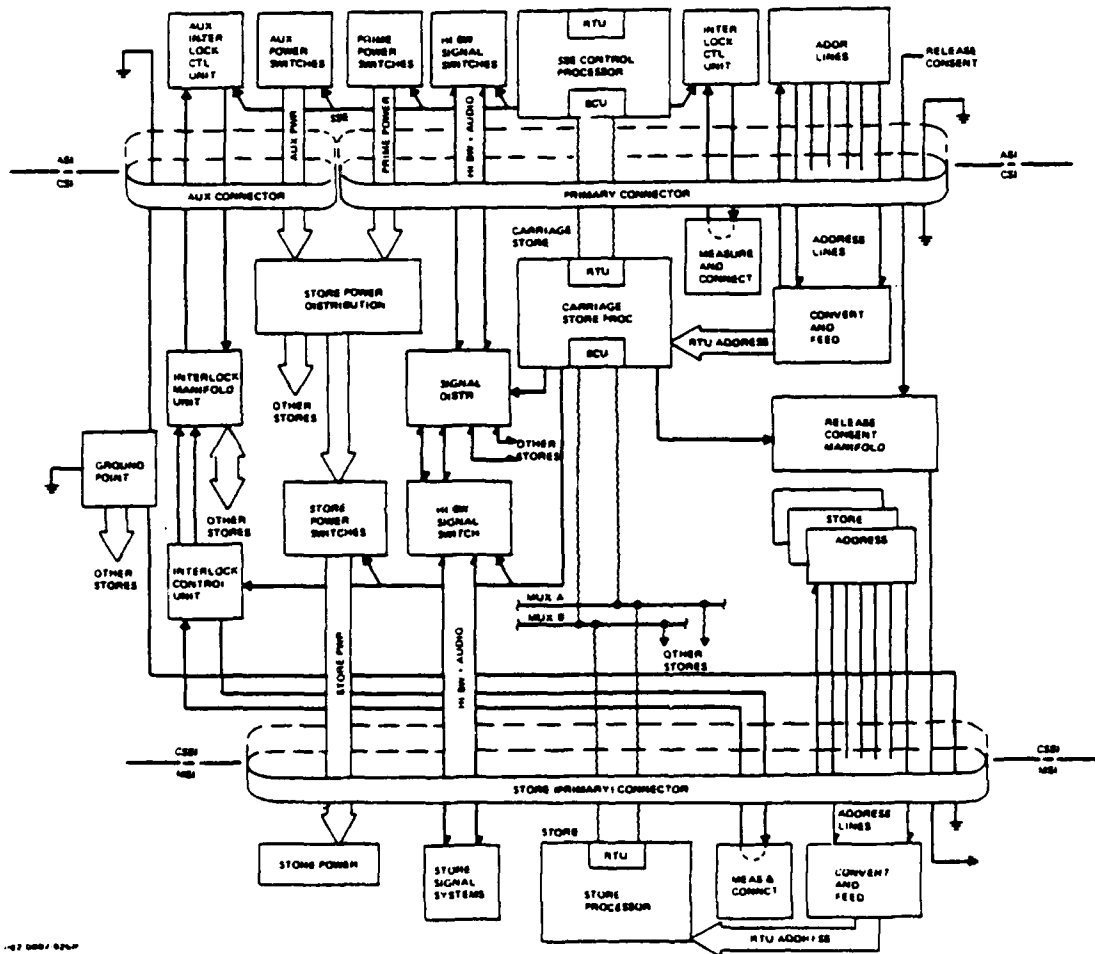


Fig. 1 MIL-STD-1760 Interface

INTERFACE THROUGH A CARRIAGE STORE

The MIL-STD-1760 interface appears possible, if EMC problems, made more poignant by the limited size of the physical connectors, can be eliminated. Figure 2 shows a schematic of a primary and auxiliary MIL-STD-1760 connection between vehicle and carriage store, and a primary connection from carriage store to one of several mission stores carried on the carriage store. It illustrates high and low voltage direct current and ac lines running along with multiplex bus lines, discrete and high bandwidth and audio signal lines. The complexity of these interconnections is called for by the standard itself, and is the price demanded to allow for using new, intelligent and highly capable stores.



1971-002(T)

Fig. 2 Interface Through a Carriage Store

The complexity illustrated in this figure is multiplied manifold when the overall aircraft system architecture is taken into consideration. The need for improved performance of future weapon systems has forced designers to look to more expensive avionic components with much more richly interconnected signal, data, and message infra-structures. The current design approaches have resulted in decreased operational availability and greatly increased costs, which, in turn, contribute a major impact to the "numbers affordable" situation that we are faced with today.

DRIVERS OF SYSTEM STRUCTURE

The reduced number of systems causes a demand for greater performance which serves as a forcing function to drive the cycle to iterate one more time. Eventually, we end up with the weapon systems equivalent of the Dirac delta function - infinite capability, but zero systems. Figure 3 illustrates this spiral. We have already cycled around the vortex several times.

If we are to break this negativistic management-technical cycle, we have to substantially reduce, not only stores management systems costs, but also the cost of interfacing the stores management system with the remainder of the total vehicle/avionic system - that is, the overall system architecture.

The major solutions proposed to reduce costs and increase availability and compatibility of stores management systems use standard interfaces as the basic integration elements. Standards are being used wherever possible to provide for lower cost system-wide components, limiting, though they may be, of "innovation" and "creativity". To use the old saying, "it goes without saying" that proliferation of equipments and systems have gone "hand and glove" with most programs defined, designed, and developed over the past twenty years. On the other hand, one recent innovation - the digital multiplex bus - has added value by permitting a standard interface to be defined, and at the same time, allowing a limited measure of system reconfigurability and improved system message traffic control.

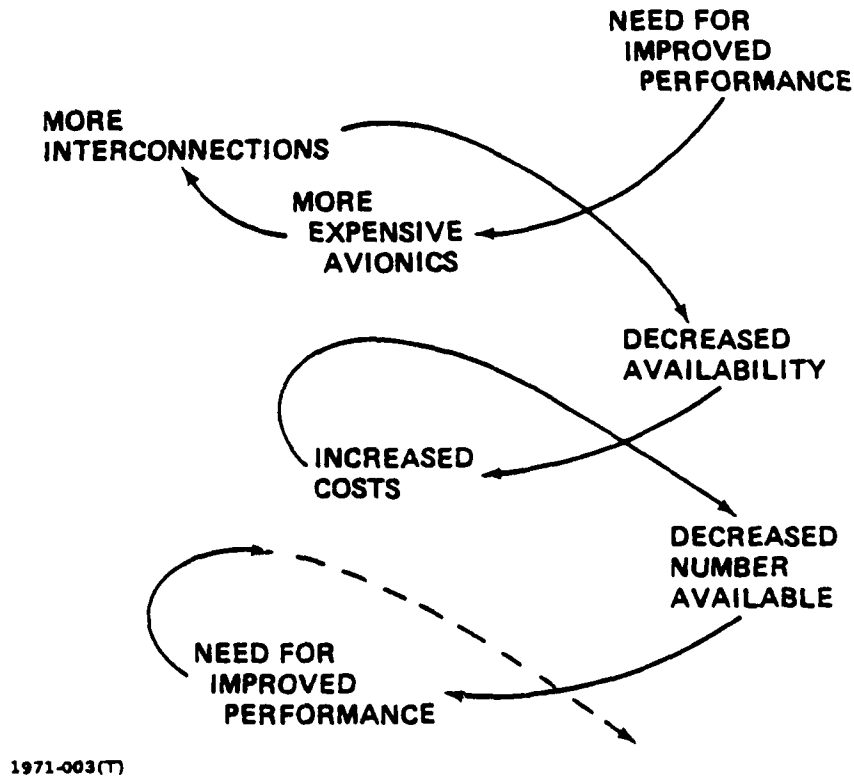


Fig. 3 Drivers of System Structure

SSI COMPATIBLE STORE

The combination of these new design approaches has, regardless of their inherent worth, caused new areas of concern to be uncovered. Figure 4, which shows one configuration of a top-level stores management flow, illustrates that at each control point in the system, the three following processors may be needed to be embedded into the system: one for the remote terminal to the superior MUX bus; one for the system control processing; and one for the bus controller. This multiple processor use is required because of the distributed, hierarchical structure which was selected for the illustrated system architecture. The system shown uses MIL-STD-1553B, the protocols of which require a considerable amount of processing with each interface. The least expensive implementations would appear to call for microcomputers to be incorporated into the terminals.

The system design approach is good; however, when we look at the need in the armament area, as well as all other vehicle subsystems, we see that:

- Only messages and data are being sent over MUX buses, and so signals of both low and high bandwidth, which are proliferating, are still uncontrolled

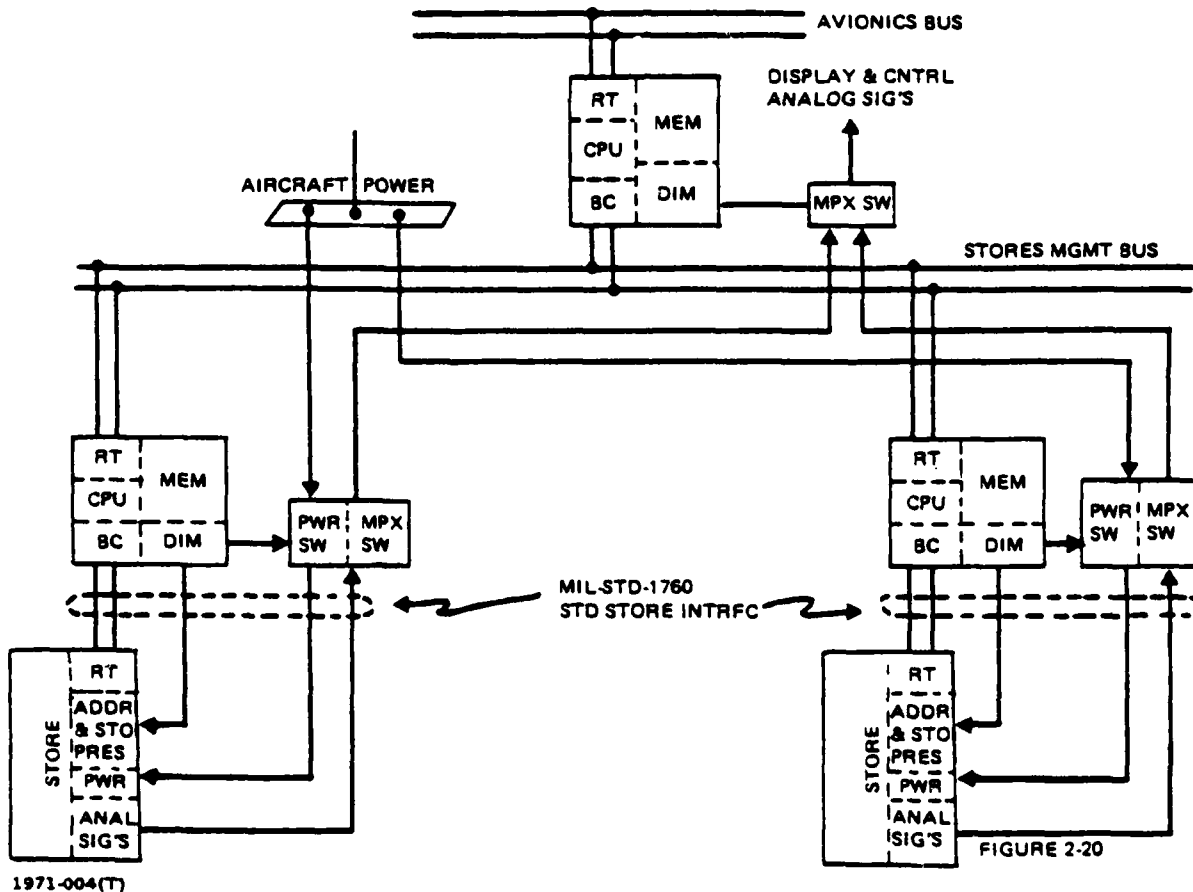
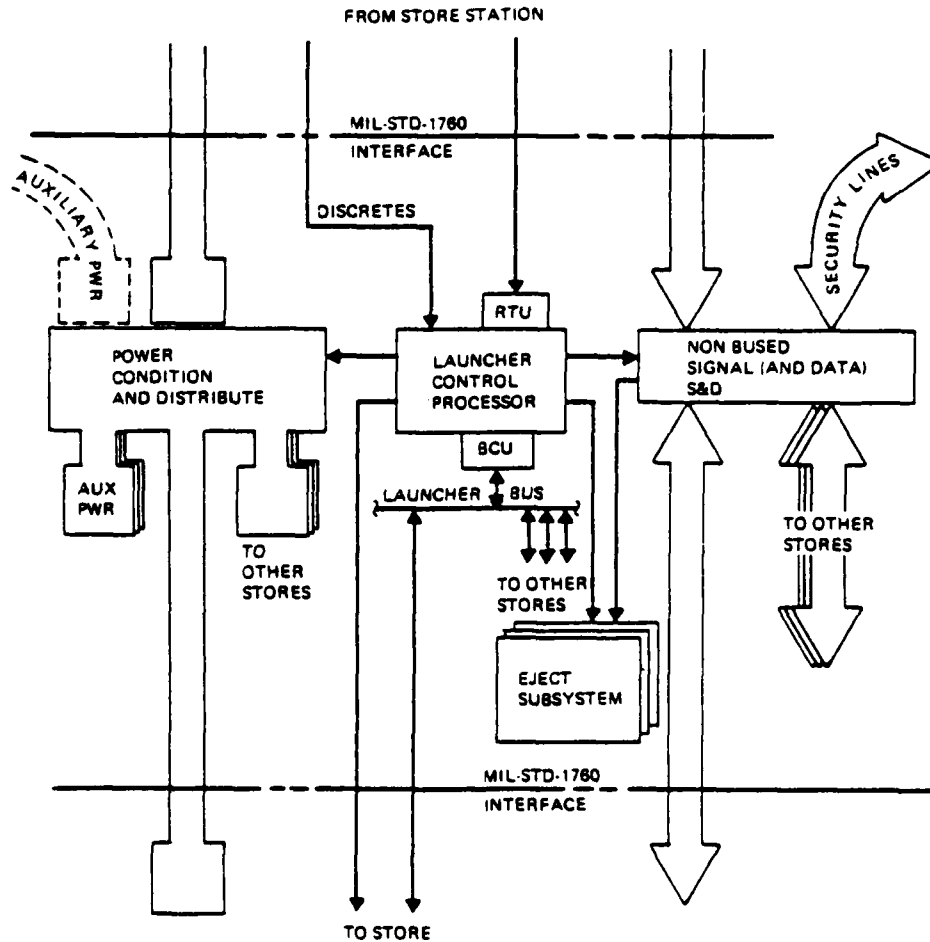


Fig. 4 Functional Hardware Blocks For SSI Compatible Store

- Redundancy is used to provide required minimum power, signal, message and command interconnections, not for increased availability or sub-functional reconfigurability
- Many parallel paths are needed in the system with the attendant increases in cost, performance risks, and protocol-caused time delays.

LAUNCHER CONFIGURATION

Figure 5 illustrates a possible launcher configuration for new "smart" store; the figure depicts MIL-STD-1760 interfaces, and shows that only a fraction of the power and information crossing the interface constitutes MIL-STD-1553 type data from the control processor. The figure also illustrates that the control processor is gating and controlling all of the non-MUX elements.



1971-005(7)

Fig. 5 One Launch Configuration, For Multiple "Smart" Stores

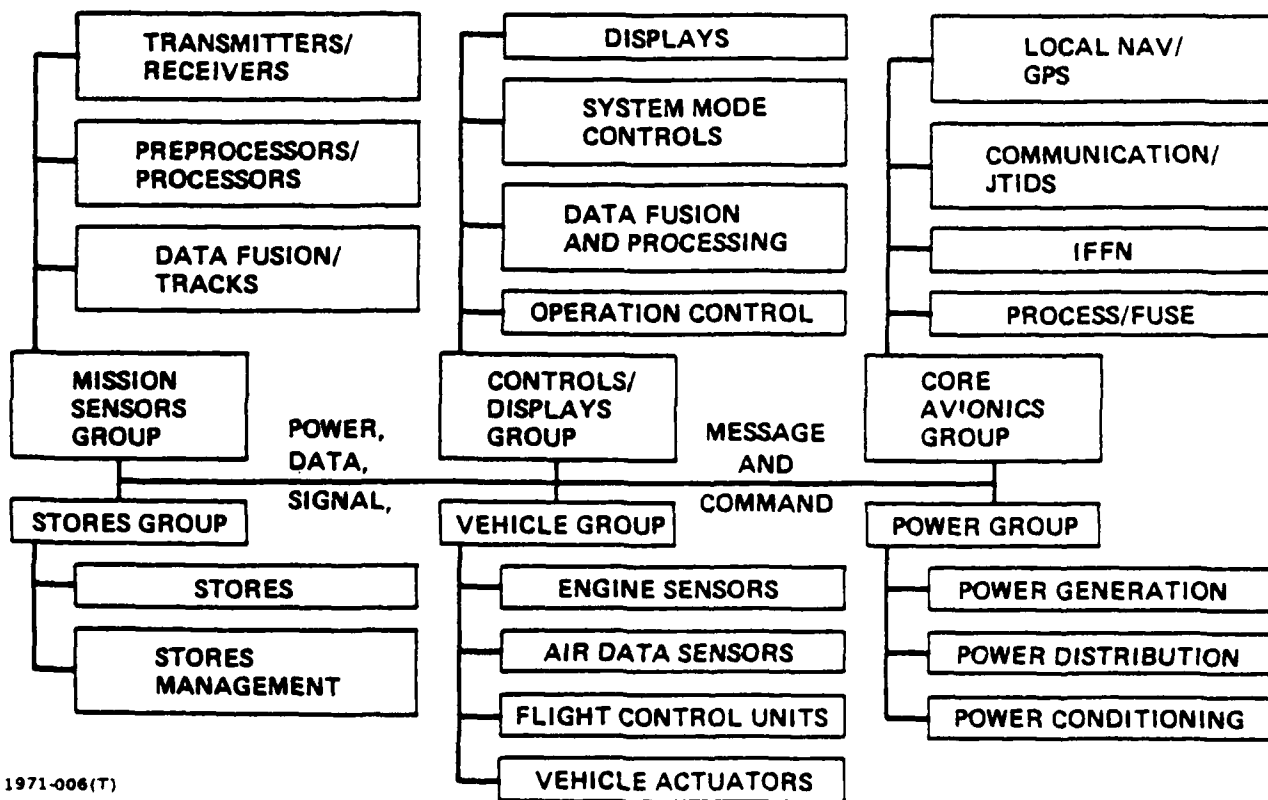
When MUX standards drive the overall system architecture, they have a major impact on system design; thus, we have systems that are more costly, more complex, less reliable and/or available, and perhaps have higher risks than currently deployed systems, with the only assured gain being that of inter-operability.

Successful multiplexing schemes follow from the definition of a sound system architecture. Successful architectures relate (tightly coupled) basic functions to be performed with the need to improve system characteristics in areas such as availability, flexibility, safety and cost, while concurrently maximizing capability and inter-operability.

VEHICLE SYSTEM FUNCTIONAL GROUPING

We find, over and over, as shown in Fig. 6, that certain functional groupings, as follows, are natural sub-divisions of aircraft systems:

- The Vehicle Group - contains kinesthetic systems, such as engines and flight control units (including fly-by-wire equipment)



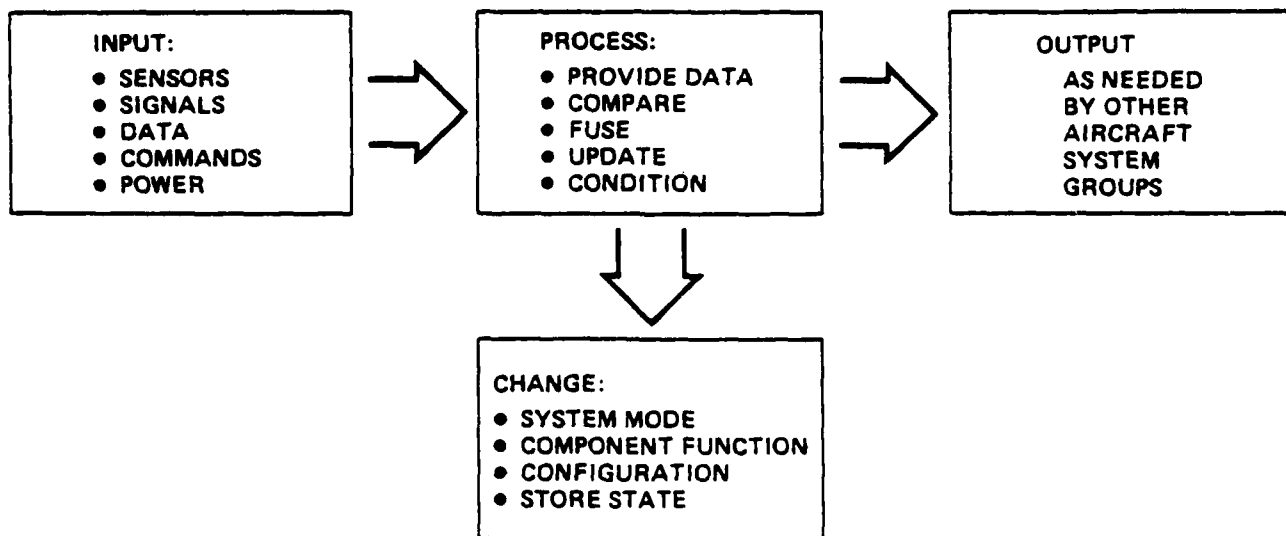
1971-006(T)

Fig. 6 Vehicle System Functional Grouping

- The Mission Sensors Group - consists of radars; IR or laser sensors; acoustic sensors (often partially deployed as sonobuoys), or perhaps on-board EW sensors
- The Core Avionics Group - contains communication and navigation equipment and general data processing, as well as other computational resources
- The Controls and Displays Group - where tactical, maneuver and maintenance decisions are made. More and more of the functions in this group are being automated to relieve the crew of very heavy data inputs and decision burdens, that is, more man-machine interfacing
- The Stores Management Group - the "effector" group of every aircraft system, the "muscle" so to speak. It normally consists of a number of stores such as the following: guns, jamming rods, sonobuoys launchers, and carriage stores of all types. It also includes the Stores Management System to supervise and control the use of these stores
- The System Power Group - including generators, power distribution units and controls for electrical power conditioning.

At the next lower level we see that each of these groups consists of similar, if not identical, functional flows. They each contain input ports; some from sensors, some from external control devices, and some from self-contained sensors. These inputs are processed to provide data, or in the case of the system power group, properly conditioned electrical power in a form useful for comparison with other inputs, and for utilization by other aircraft system groups.

As Fig. 7 illustrates, each of these groups produces its own functions to some extent and feeds necessary signals, data or power to other groups. The group that requires the most necessary signals, data or power, to other greatest number of self-contained functions, is the stores group. Yet, the basic structure of the stores group is the same as the other groups.



1971-007(T)

Fig. 7 Group Functions

Within the stores group the current "special problem" is the number and different variations within the group. However, these now appear to be able to be standardized into a small number of discrete signal types to handle all inter-group interfaces. This functional breakdown, from the top level down to the actual group functional elements, is the first requirement in the development of technically sound architecture and data multiplexing techniques.

AVAILABLE TECHNOLOGY

If we are to successfully implement the new systems, we must also understand the technology available in the time period considered. Systems operational at the turn of the millenium can expect to take advantage of the following:

- Previous software functions subsumed into firmware. As the cost of memory and processors decreases, almost all standard or even semi-stable, (that is, only infrequently modified software) will be represented by firmware modules. Even the most inexpensive firmware systems, however, will be expensive o change, implying greater emphasis on standardized software and on quality control
- In the long run, those functions which have been validated and which are universally useful, will have been subsumed into VHSIC elements, being no longer even considered functions at the level of today, but more as elementary system operations
- The hardware elements developed will become more standardized, becoming building blocks of elementary system functions
- Heavy emphasis on signal processing and the long awaited general recognition of the necessity for modular software are certainly no surprise, but they will impact the architecture of each avionics group, as well as the overall vehicle system
- The important areas of fault tolerance, reconfigurability and data extraction are also included in what we are aiming for in both architecture and data multiplexing.

We are not here concerned with the economic and political impact thest technological changes will have in the avionics industry, but only on how we can exploit them in providing better systems.

SYSTEM ARCHITECTURE

Another area of concern is that of system architecture itself. Our functional breakdown (vehicle system functional grouping) of Fig. 6, coupled with the modularity demands of new technology, forces us to look at architectures which allow for reconfigurations under the control of programmable microprocessors which are sensitive to the operating conditions of the basic architectural configurations. We need to look more closely at the implications of built-in-test (BIT) and its possible conflict with subsystem commands in situations heavily time constrained. This need is especially important for stores management systems, as the "mushrooming" of the number of logical and physical "sneak paths" forces a far more rigorous analysis of SMS system safety. For us, more

than for those involved in any other functional grouping of systems, even one short signal failure interval can be, in fact, a total system level failure. The specter of accidentally jettisoned, or even worse, launched stores, forces us to reduce the number of separate data channels in order to exercise greater control. This implies multiplexing as an inherent part of SMS architecture. Within the recent past, as noted in the public media, accidental "firing" or launching of missiles do indeed take place with their landing sites being unconventional.

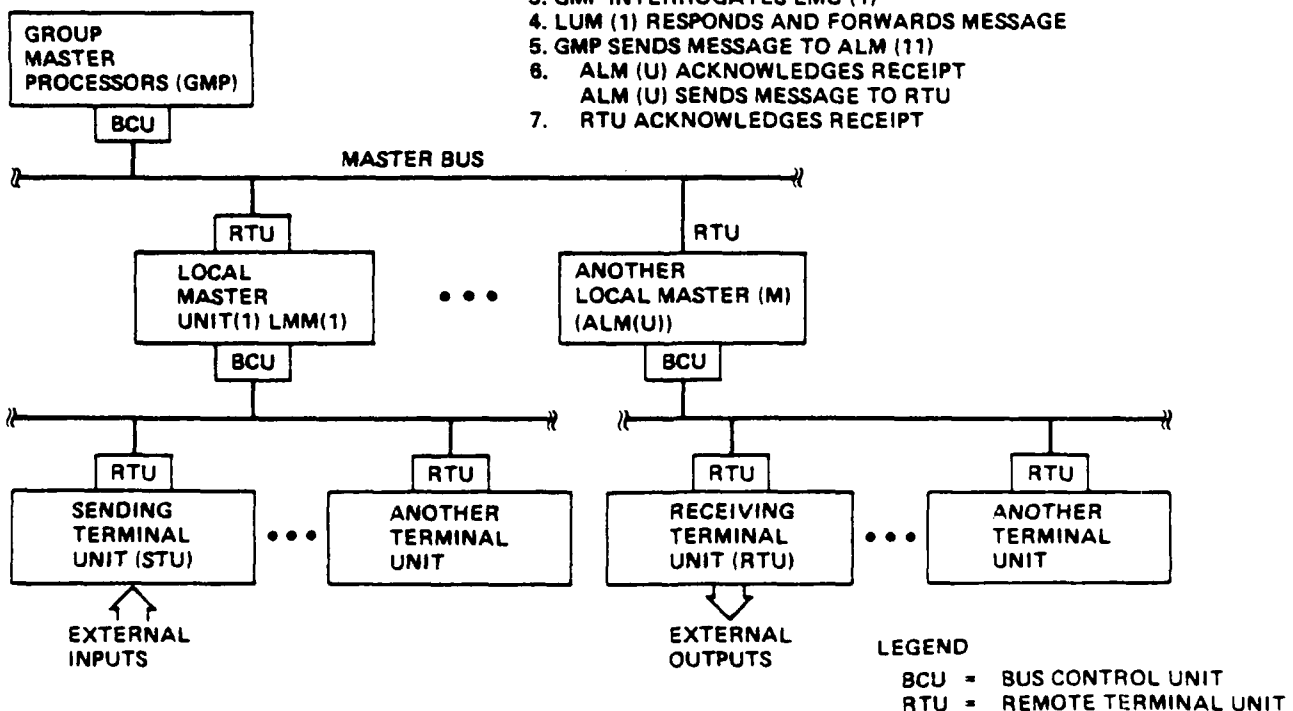
Timing constraints, more critical than in any other group, drive the architecture to as small a number of hierarchical levels as is possible, and impose requirements on bus protocols which make command response protocols less desirable if even feasible in large systems.

In Fig. 8, a two (2) level hierarchy is shown, where the timing is indicated for a message to be passed from one terminal to another within command-response constraints. We see that to pass one message, a minimum of six must be sent along these buses, and two acknowledgements returned.

If the depth of the system is reduced to but one level, as illustrated in Fig. 9, the system overhead is considerably reduced. In the figure, illustrating a single-level bus with the same terminals as shown in Fig. 8, and assuming the same message timing as in that figure, we can see that to pass the same message under command-response protocol,

STEPS:

1. LOCAL MASTER (LMU (1) INTERROGATES STU
2. STU RESPONDS AND SENDS MESSAGE
3. GMP INTERROGATES LMU (1)
4. LUM (1) RESPONDS AND FORWARDS MESSAGE
5. GMP SENDS MESSAGE TO ALM (11)
6. ALM (U) ACKNOWLEDGES RECEIPT
ALM (U) SENDS MESSAGE TO RTU
7. RTU ACKNOWLEDGES RECEIPT

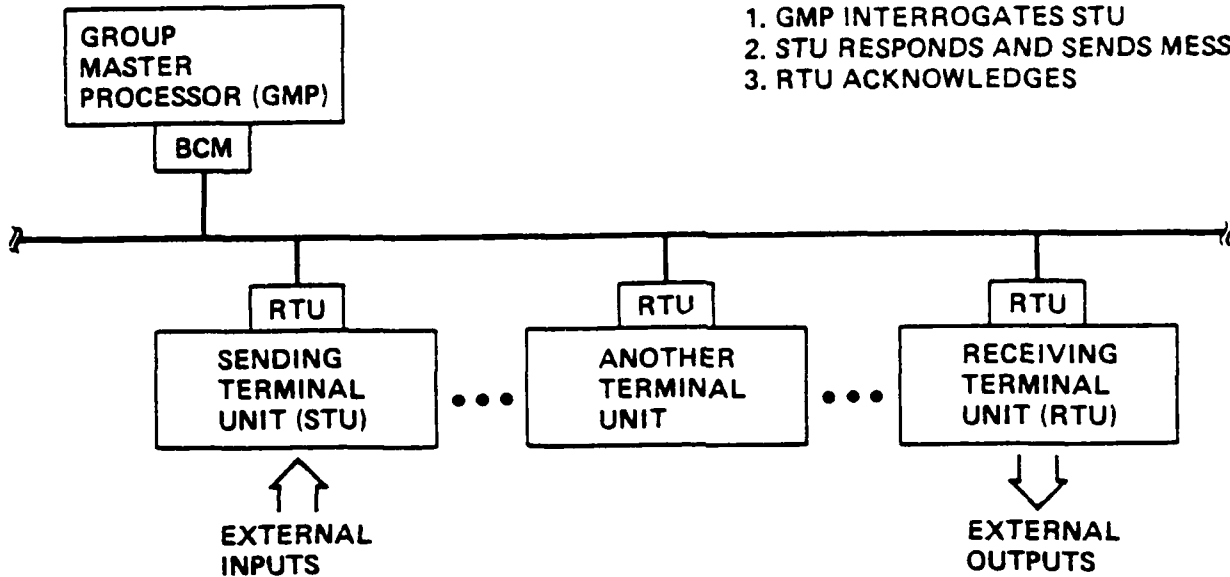


1971-008(T)

Fig. 8 2-Level Bus Hierarchy (MIL-STD-155313)

STEPS

1. GMP INTERROGATES STU
2. STU RESPONDS AND SENDS MESSAGE
3. RTU ACKNOWLEDGES



1971-009(T)

Fig. 9 Single Level Bus Structure (MIL-STD-1563B)

two messages must be sent and one acknowledgement elicited. This may take approximately the same time as the previous structure, depending on the terminal polling priorities, but still is an improvement because it does not clutter the buses with as much administrative data.

Both of these architectures contain inherent delays due to the multiplexing approach; both are serial MIL-STD-1553B and both operate in command-response protocol structure. These limitations appear necessary in the short term, as indicated in the MIL-STD-1760 interface, where the number of lines in one connector is already sufficient to cause worries about sneak paths and reliability.

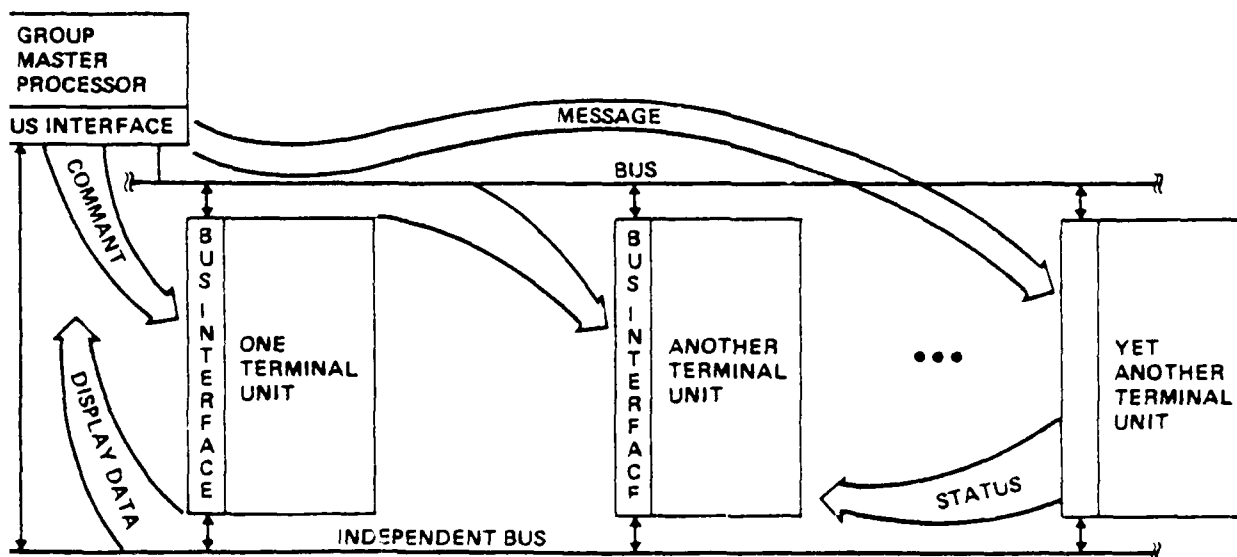
On the other hand, the standard itself contains elements that are dependent on the anticipated multiplexing procedures. Seven lines are used to provide hardware addressing of the store remote terminal. Two shielded twisted pairs are provided for wire multiplexing with provision for two fiber buses. A separate, shielded, twisted pair is provided for audio signals to and from the store, and four high bandwidth lines are also included. The primary connector carries 8 lines for power with provision for two additional high voltage lines. The auxiliary connector also contains the power lines of the primary connector along with interlock and ground. All these lines need be switched within any interface control unit under switching logic dependent on both the logic of the interface and the mutual requirements of store and vehicle.

One of the high bandwidth lines, the 2 GHz line, is capable of carrying, alone, all signal, message and control information passing through the interface, in a serial fashion. Because of the available line bandwidth, parallel message channels can be created by using frequency separation of word bits. Frequency separation of messages of message word structure can greatly speed up communication through these lines. Some of these lines would provide for simultaneous two-way communications with each other, at a saving of many possible failure sources. If EMC is not a problem, the entire connection can be made with an augmented auxiliary connector with two 2-GHz lines added, as suggested in fig. 10. Fixed purpose, low cost processors at each end of the connection, can reduce the interface to a more simple, cost effective scheme before the year 2000.

SIMPLIFYING THE SYSTEM

The approach shown in Fig. 10 can include the following desirable elements:

- Systems grouping - to define all the required interfaces
- Analysis of the interface requirements - to reduce the physical component size, complexity, and the potentially hazardous logical complexity of the interface
- Standardization of the interfaces - such that all designers know the characteristics of all their inputs and outputs, and so can concentrate on improving component function (A212). A212 was an initial attempt to accomplish this, and, in so doing, helped to generate MIL-STD-1760
- Definition of parallel bus paths - for different elements, and investigations of value of combining into one bus
- Investigation of the distribution of bus control - making full use of both time and frequency spreads for maximizing the utilization of the bus bandwidth.



1-010(T)

Fig. 10 Some Parallel Paths On Buses

42 778

PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND.. (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O..
C A PORUBCANSKY NOV 82

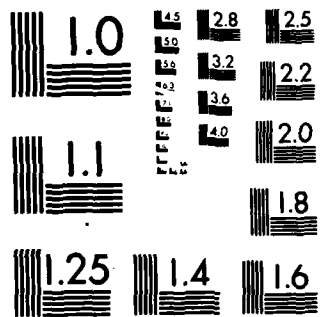
378

UNCLASSIFIED

F/8 9/3

NL

The table consists of a grid of approximately 13 columns and 15 rows. The top row is mostly blacked out, with only the first cell containing the text 'UNCLASSIFIED'. The remaining cells in the grid are also blacked out, obscuring any data or text that might have been present. The grid is contained within a white border.

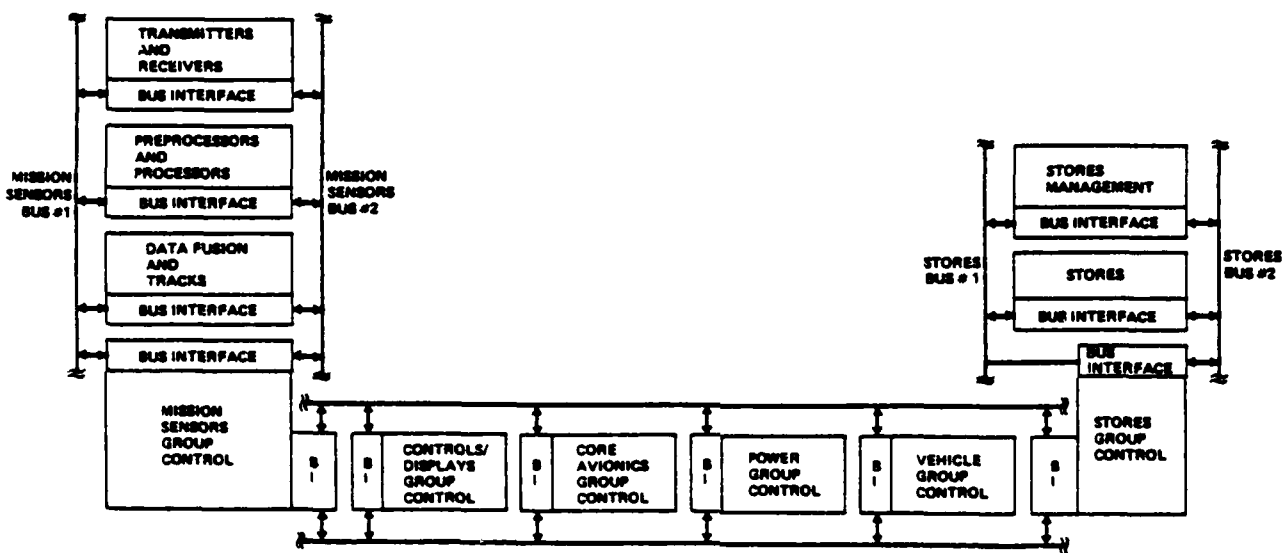


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

This approach will have the effect of simplifying the system while maintaining full capability, flexibility for reconfigurability and safety. It also has the advantage of providing for adding new and, as yet, undefined system elements, with greater bandwidth, as yet undetermined in magnitude, without increasing the system-level complexity.

CONCLUSION

Figure 11 shows a conceptual configuration that embodies these considerations for a future (as yet undefined) high performance aircraft. The figure is intended to suggest that all of the above desirable elements can be attained, and that, with the use of the very high bandwidth line suggested for buses, not only is distributed control possible, but multiple simultaneous control on even single buses can be expected to be the normal operating mode.



1971-011(T)

Fig. 11 Conceptual High Bandwidth Configuration

MIL-STD-1862

NEBULA 32 BIT INSTRUCTION SET ARCHITECTURE

SESSION CHAIRMAN: Captain Michael P. Vincent
HQ CECOM/DRSEL-TCS-MCF
Ft Monmouth NJ

MODERATOR: Dr. Edward Lieblein
HQ CECOM/DRSEL-TCS-MCF
Ft Monmouth NJ

THE NEBULA AND MCF STANDARDIZATION PROGRAMS

EDWARD LIEBLEIN
DIRECTOR, MILITARY COMPUTER FAMILY PROJECT
US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NJ

Military computers are currently employed in defense systems to perform a wide range of functions in the areas of weapon control, command and control, communications, intelligence analysis, navigation, surveillance, target acquisition, sensors, electronic warfare, and combat support services. Over the years, the number of types of computers used in such systems has increased. This proliferation has become a serious problem in that it increases the cost of software development and support, maintenance, training, and the supply pipeline. Further, it impedes competition for technology upgrades and reduces overall battlefield survivability.

Recognition by the Department of Defense and the military services of the need to curtail proliferation has resulted in efforts to standardize at various levels: high-order language (HOL) and its software environment, instruction-set architecture (ISA), and hardware. With regard to the first level, the services are working together to standardize the Ada language. At the second level, the Navy has standardized on the ISA's of the AN/UYK-43 and AN/UYK-44. The Air Force has standardized on the MIL-STD-1750 ISA and both the Army and the Air Force have standardized on the MIL-STD-1862 (Nebula) ISA. Computer hardware standardization, due to different needs, has not extended across service lines.

THE ORIGIN OF NEBULA

The Nebula effort began in 1979 as part of the Army's Military Computer Family (MCF) Program following a decision to use a single instruction set for the entire family; all members were to be software-compatible. Following this decision, the Army established requirements for the standard ISA: It would have to be (1) government owned, (2) a state-of-the-art instruction set that would be efficient for military real-time systems, (3) HOL oriented and optimized for Ada, (4) a 32-bit instruction set with a 32-bit virtual address space, (5) suitable for multi-level security, and (6) compatible with future advances in technology. Many existing ISA's, commercial and military, were evaluated against these requirements with the conclusion that none of these satisfied the requirements well enough to become a long term standard.

The requirement for government-ownership has been driven by the desire for neutrality in the ISA in order to insure real competition for all computer acquisitions while at the same time facilitating software transportability across hardware. If there were no standard ISA then the services would be unable to move battlefield software from computer to computer without expensive redevelopment of a large percentage of such software. This would hold in spite of the use of Ada due to ISA dependencies in the language and due to the use

of embedded code where necessary (e.g., I/O).

The design of Nebula was begun in 1979 by William Dietz and Leland Szeverenko at Carnegie Mellon University (CMU) under Army sponsorship. Comparative analyses were conducted to provide a basis for the design. Algorithms were established that represented the military computing environment, and then programs were developed in various candidate ISA's for these algorithms in order to obtain relative measures of efficiency in terms of ultimate hardware complexity.

There was extensive community involvement in the design effort. In December 1979 and January 1980, reviews of the preliminary design were conducted with all three services represented. An industry seminar held in March 1980 was attended by 190 industry, government and university representatives. Comments were solicited. A large number were received and many of the ideas were incorporated into the design. Ada experts made major contributions. A second industry review was solicited in the Spring of 1980 based on a draft specification prepared by CMU. This resulted in the publication of Nebula as MIL-STD-1862 in May 1980.

During the Summer of 1980, discussions were held between the Army and the Air Force on Nebula. While the Air Force was using MIL-STD-1750 in avionics systems it had a need for a 32-bit ISA for both ground-based and aerospace systems. Nebula became a joint Air Force and Army 32-bit standard under an agreement signed by the Deputy Commanders of HQ AFSC and HQ DARCOM in September 1980. The agreement established a Nebula Control Board (NCB) with equal Air Force and Army membership to manage the standard. Seven organizations from each service were placed on the board as voting members. Nebula continued to evolve through both public and government reviews.

The NCB formed a Tiger Team in November 1980 to solicit and consolidate recommendations for change. Many recommendations came from an in-depth review by the Electronic Industries Association that was completed in the Spring of 1981. Others came from ongoing reviews by Ada experts, in-house Air Force and Army personnel, MCF hardware and hardware-support contractors, and from several software contractors. These activities culminated in MIL-STD-1862B, the standard that will be employed in all Army and Air Force implementations. At the present time, both services are acquiring computers that will employ Nebula.

THE MILITARY COMPUTER FAMILY

The rapid growth in Army use of "go-to-war" computers over the last six years has resulted in an extensive proliferation of different and incompatible types. In 1979 there were 35 different types of computers employed in a total of 49 Army battlefield automated systems. By the end of 1981 these numbers increased to 50 and 65 respectively. Army management became seriously concerned about this situation for two reasons. First, the proliferation of types adversely affects system survivability. Second, it increases significantly the cost and complexity of hardware logistics support, maintenance, training and acquisition, as well as that for software development and support. It has been estimated that the cost to the Army of continued computer proliferation will reach \$360M per year by 1990 and go up to \$880M per year by the year 2000. The estimated cumulative cost to the Army of proliferation over the thirty year period from 1981 through 2010 is \$9B.

In order to make battlefield automation affordable, supportable, and survivable, the Army, in 1980, established the policy that future systems must use a standard software-compatible computer family. (The details of the policy, which are reviewed annually, are contained in AR 1000-1.) Development of this family commenced in 1981. Members of the family will include a super-minicomputer (AN/UYK-41), a microcomputer (AN/UYK-49), and a 6" by 9" single-board computer that is a component of the microcomputer. Smaller single-board computers for use in missiles, armaments, tanks, and helicopters may be added to the family prior to the start of full-scale engineering development (FSED). An Army Military Computer Family (MCF) Working Group, formed in the Summer of 1982, has been addressing these and other needs. Comprised of representatives of DCSRDA, DCSOPS, DCSLOG, DARCOM, TRADOC, CACDA, CSC, and each of the major DARCOM commands, including several PM's, the group was formed to make sure that the attributes of the family dovetail with the broad spectrum of Army needs and that that approach is cost effective. The group also has been identifying specific systems for which the use of MCF is planned in order to coordinate schedules and assure the early availability of models. In addition to the MCF family members described above, a set of very large scale integrated (VLSI) circuit chips, commonly called a "chip-set", will also become available as a byproduct of the basic development, for smaller computing requirements. All members of the family will employ standard interfaces and will be software-identical via MIL-STD-1862 (Nebula) and fully plug-compatible in order to facilitate replacement of faulty units, and to support mobility of software, distributed processing, and graceful degradation.

The Army recognizes that in standardizing it must prevent future lock-in to obsolescent technology and must also provide for competition on a long-term basis. These are particularly difficult to achieve with respect to the computer field where technology is advancing so rapidly and where the legal protection of software invariably leads to lock-in to individual vendors. To avoid technological obsolescence, multiple relatively short productions are planned (Fig 1). Advanced technology products will be introduced in successive generations while maintaining software and interface compatibility. Technology approaches will be competed for the production of each generation. Products resulting from successive generations are expected to have improved reliability, maintainability, power, size, weight, cost, speed and memory capacity. Future units will maintain instruction set (software) and interface compatibility with units produced in previous generations in order to provide the potential for upgrade/replacement of older units in the field. (Support for fielded units, however, is expected to continue for as long as it is practical to do so.)

The MCF Program will not, in itself, develop new technology but will attempt to extract the best technology available from the commercial/industrial base. Direct use of commercial technology and components will be encouraged. Commercial off-the-shelf computers will be employed for software development and post-deployment support in that they offer lower purchase costs, readily available and low cost maintenance, and available software. Also, extensive use is planned of commercial software for development and support.

The first phase of the program is underway. It will take five years to go from inception (1981) to production (1986). The MCF approach differs from the usual where the initial technology ends up as the final technology used in production units. During the first phase of MCF, two technology efforts are being pursued, one oriented toward the use of 1981 technology for the prototypes, and the other toward selection of technology for production. The latter requires analysis and assessment of potentially suitable technologies for use in late 1983 to early 1984. Thus, production units will embody the latest advanced computer technology. Contractors are free to choose their own approach to technology as well as to hardware system architecture.

Competition has been a hallmark of the MCF Program. The approach, which is aimed at reducing risk and achieving the best solutions, will provide for extensive competitive industry participation throughout the program. Each generation will start with an open solicitation for advanced development (AD) for which multiple competitive contracts will be awarded. To keep the competition focused, the most important evaluation factors and their relative priorities will be specified "up front". Evaluation factors and their priority order for the current generation are as follows: (1) reliability and maintainability, (2) life-cycle cost and power, (3) size and weight, and (4) speed and memory capacity. After the prototypes and approaches to production (technology, life cycle cost, etc.) have been evaluated, two contractors will be selected to continue into FSED. Competition will continue up to, and possibly through production. At the same time that production starts, new contracts will be awarded for advanced development for the next phase. The planned phasing and extensive competition are expected to prevent lock-in and, make available the best possible military computers at the lowest possible life-cycle costs.

When each new production commences, one of several alternative choices could be made with respect to MCF computers already in a fielded system: (1) continue to support such computers; (2) replace all such computers with those from the new production. (This will require re-testing of the system to qualify the use of the new product); or (3) qualify the use of the new computers as interchangeably equivalent to those currently in the system and use both types in the system.

The first phase of the program started in 1981 with awards made for advanced development to four of 12 bidders, GE/TRW, IBM, Raytheon, and RCA. In March 1982, IBM was eliminated from the competition. Another company will be eliminated in August 1983. Major goals for production units, established as the basis for this competition, are shown in Fig. 2. Significant deliverables during the AD phase include reliability and maintainability projections, life-cycle cost analyses, technology insertion (projection) plans, prototype models, prime item specifications, and producibility plans. Prototypes will be delivered at the end of January 1983.

There has been a strong emphasis on computer and system survivability in the MCF Program due to the degree of future dependence of our fighting forces on automation. The need for survivability has provided the impetus for fielding a family of standard, software-compatible computers. Should a high-priority system malfunction during a battle because of a computer failure, if parts or

replacement computers are not available due to a shortage or due to cut lines of supply, then it would be possible to restore operation quickly by taking parts or entire computers from lower priority systems. Further, standardization will facilitate the completion of repairs quickly due to the use of common parts and the availability of maintenance personnel. To minimize the probability of computer failure after the shooting starts, very high levels of reliability, maintainability, and ruggedness are being sought. Mean time between failures (MTBF) for the AN/UYK-41 is 10,000 hours (14 months). For the AN/UYK-49 and the single-board micro, MTBF goals are 33,000 hours (3.3 years) and 100,000 hours (11.6 years) respectively. To simplify maintenance it is required that 98% of all faults that would degrade performance be detected automatically by built-in-test (BIT) circuits. Of these, it is required that BIT automatically isolate the fault to the removable unit in 95% of the cases and to one of two units 98% of the time. The MIL-SPEC requirements, shown in Table 1, will insure that the MCF computers will operate over the full range of anticipated tactical environments. High survivability over the system life-cycle will be enhanced further through the ability to substitute superior (i.e., more reliable, more maintainable, etc) plug-compatible equivalents that result from successive generations.

In the event of damage or failure, provision has been made for graceful degradation (to keep systems functioning), for interchangeable hardware (to restore functionality without repairing) and to facilitate normal maintenance. The computer I/O interfaces (specifically MIL-STD-1553B and the high-speed parallel interface) as well as the instruction set provide explicit support for the design and operation of distributed processing configurations of the various members of the computer family. The AN/UYK-41, the AN/UYK-49, and the single board micro can be interconnected, combining types if desired, in either local or geographically dispersed processing networks. Since each computer will employ the same instruction set, it will be possible for the system to be designed to accommodate the transfer of software function (load-shifting) from faulty to operational computer units in the network. (This same capability will facilitate load-leveling during critical periods).

The Army's MCF Program, supported by the joint Air Force and Army Nebula effort, addresses the challenge to field the most cost-effective, survivable family of standard mil-spec computers. The unique acquisition strategy provides for time-phased introduction of advanced technology units through an intense amount of competition. The standardization approach is simple (one language, Ada, one instruction set, MIL-STD-1862, one software support environment, a common ILS system, and standard hardware units on the battlefield), yet the capabilities to be provided will be, at the same time, powerful, efficient, reliable, maintainable, affordable, and state-of-the-art.

BIOGRAPHICAL SKETCH
DR. EDWARD LIEBLEIN
US ARMY COMMUNICATIONS ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

Dr. Lieblein received the BEE and MEE from New York University in 1955 and 1963 respectively, and the Ph.D. in Computer Science from the University of Pennsylvania in 1968. He has been employed at Fort Monmouth, New Jersey since 1955 where he has worked on computer system developments, computer architecture, iterative machine arrays, languages, compilers, operating systems and tactical software systems. He was Chief of the Software Engineering Division in the Center for Tactical Computer Systems from 1975 to 1978 and has been instrumental in the development of the new DoD high order language, Ada. Since November 1978, he has been Director of the Military Computer Family Project at CORADCOM. Dr. Lieblein has been affiliate associate Professor at Stevens Institute of Technology and lecturer at Monmouth College. He is currently Adjunct Associate Professor of Computer Science at the University of Pennsylvania where he teaches graduate courses in compilers and operating systems.

FIGURE 1

MCF ACQUISITION STRATEGY

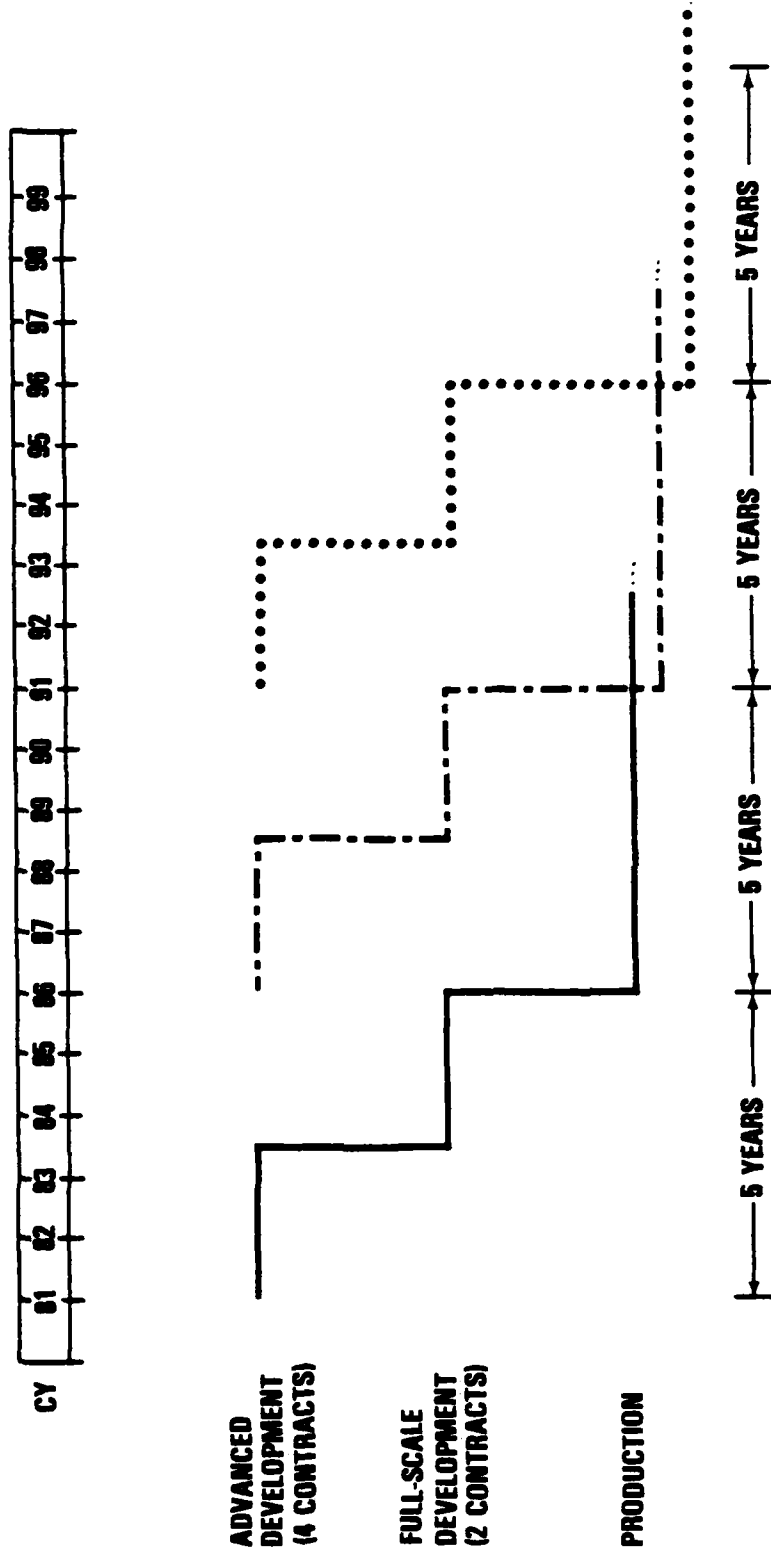
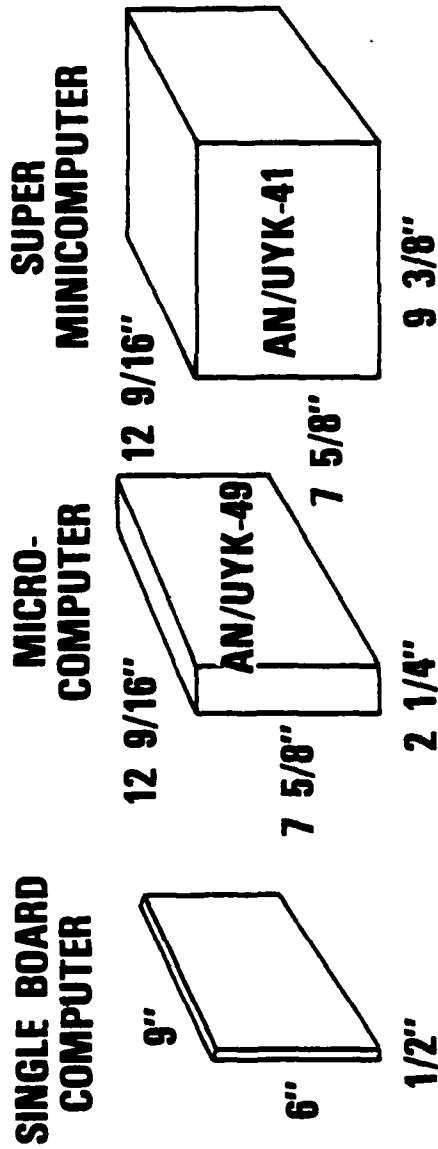


FIGURE 2

MILITARY COMPUTER FAMILY



SPEED	500 KIPS	500 KIPS	3 MIPS
MEMORY	128K BYTES	1M BYTE	21M BYTES
COST	\$5K	\$25K	\$75K
RELIABILITY	100K HRS MTBF	33K HRS MTBF	10K HRS MTBF
VOLUME	0.02 cu ft	0.12 cu ft	0.52 cu ft
POWER	5 WATTS	20 WATTS	100 WATTS
WEIGHT	12 OUNCES	10 POUNDS	40 POUNDS

132

TABLE 1



Environmental Requirements

TEMPERATURE (OPERATING)	-54°C TO 71°C
TEMPERATURE (NONOPERATING)	-62°C TO +85°C
ALTITUDE	0 TO 30,000 ft
TEMPERATURE SHOCK TEST	3 CYCLES
HUMIDITY (OPERATING)	95%
RAIN	2"/HOUR FOR 2 HOURS
SALT FOG	NO DETERIORATION
SAND AND DUST	BLOWING—2.2MPH—1 HOUR/SIDE
SOLAR RADIATION	104W/ft ² AT 49°C
FUNGUS	28 DAYS—5 SPORES
EXPLOSIVE ATMOSPHERE	13/1 FUEL/AIR RATIO
SHOCK	40Gs FOR 11 MS; BENCH HANDLING; BOUNCE (IN CASE)
VIBRATION	2G PEAK; 5 TO 2000 Hz
ACCELERATION	6G
INCLINATION	60°
EMI	
TEMPEST	
RADIATION HARDENING	

18.882-82





The Nebula Standard Computer Architecture

William B. Dietz

Tartan Laboratories Incorporated
477 Melwood Avenue
Pittsburgh, PA 15213
412-621-2210
ARPAnet: DIETZ@CMU-20C

Author's Biography

William Dietz has been involved with the Army's standardization effort since 1977, first in the evaluation of alternative computer architectures that were being considered as standards and, more recently, in the design of the Nebula computer architecture. His interests include computer architecture evaluation and specification as well as issues surrounding the generation of high quality code from compiler back ends. Dietz received his B.S. in Physics and Mathematics from Muskingum College, and his M.S. in Computer Engineering from Carnegie -Mellon University. Dietz is currently a member of the senior technical staff at Tartan Laboratories Incorporated.

Abstract

✓ Nebula is a standard instruction set architecture designed for embedded military applications. Several new ideas were incorporated in the standard that allow a family of implementations to effectively use advancing technology.

In addition to following design principles that allow Nebula to be a good target for high level languages, the designers also adopted a visibility approach in architecture design that provides more freedom for the hardware implementor while still maintaining software portability.

1 Introduction

There are two important properties that a standard instruction set architecture should have. The first property is generality. Computers built to an instruction architecture standard must be able to provide processing power to meet the needs of a wide variety of applications. The second property is implementability. Not only should it be possible to build computers that conform to the standard, it should also be possible to exploit advances in technology to provide new implementations with improved cost/performance characteristics.

The Nebula Architecture was designed to embody these properties [3]. Some important characteristics of the architecture include:

- 32-Bit general register architecture.
- Variable length instructions with variable number of operands.
- Explicit addressing of instruction operands.
- Multiple addressing modes for register, literal, memory and parameter access.
- Procedure based control structure with a local register set for each procedure.
- Support for several data types and representation lengths.
- Memory mapped I/O control.
- Vectored Interrupts and Exceptions.

The remaining material is divided into two main parts. The first part provides information about the Nebula architecture. The second part describes some of the considerations that contributed to the design of the Nebula architecture.

2 Nebula Architecture Overview

Nebula is a 32-bit general purpose architecture with byte-addressable memory. Instructions are represented as a sequence of bytes, with the first byte specifying the operation to be performed and succeeding bytes specifying the operands. All instruction operands are explicitly specified using the same addressing modes. The uniform generality of operand specifiers avoids the data positioning problems of more restricted formats.



Figure 1: Nebula Instruction Format

2.1 Operand Accessing

Operand specifiers are sequences of one or more bytes that specify the location and size of an operand. Some of the seven addressing mode classes are available in multiple forms of varying compactness. The modes provided are:

- Literal: an instruction stream constant
- Register: a local register
- Absolute: an absolute memory address
- Register Indexed: a register or program counter relative address
- Indexed by Value: an address indexed by another value
- Scaled Indexed: an address indexed by a value scaled for the data size
- Parameter: the parameter of a procedure

Additionally, each operand specifies a data size. Primitive data types, such as integer and real, are available in representations of various sizes. Integers are fully supported in 8, 16, or 32-bit sizes, and partly supported in a 64-bit size. Reals can be 32 or 64 bits. Operands of a given instruction can be of mixed sizes; no explicit size conversions are necessary.

2.2 Procedure Interface

At a higher level, the control structure of Nebula is based on the concept of procedures. An executing procedure has a local register set, an exception handler, and a list of parameters. The characteristics of a procedure are described by a 16-bit procedure descriptor located at the procedure's entry point. The descriptor indicates the number of parameters for the procedure (may be fixed or variable), the number of registers used by the procedure (up to 15), and the method used to handle errors.

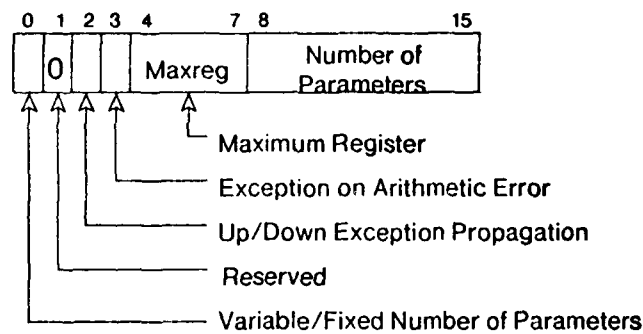


Figure 2: Procedure Descriptor

Procedures are invoked by various forms of calls. The call specifies an entry address and a parameter list, which is a sequence of operand specifiers identical to those used for instruction operands. The procedure invocation establishes a correspondence between the parameter list in the call and the parameter addressing modes of the called procedure; within the called procedure, a parameter is accessed by an index corresponding to its position in the parameter list.

The manner in which an entry address to a procedure is specified depends upon the type of call. The call instructions specify the address as an operand, supervisor calls and unimplemented instructions obtain the address by indexing into protected vector tables, and interrupts and traps are assigned fixed vectors. Once the entry address has been determined, all these forms of procedure

call are treated uniformly. The difference between an instruction implemented in the microcode and an unimplemented instruction code that invokes a software procedure performing the same operation is not detectable in the object code of the machine. This allows the implementor to determine the right mix of software-emulated instructions and microcoded instructions based on the implementation constraints. It also permits the instruction set to be extended and earlier models to be retrofitted in software.

The nesting of procedure calls produces a sequence of procedure contexts, each of which contains the registers, parameter list, program counter, and exception handler state for the associated procedure invocation. Storage for these contexts is allocated on a context stack associated with each task; the memory management facilities protect this area from instructions seeking access.

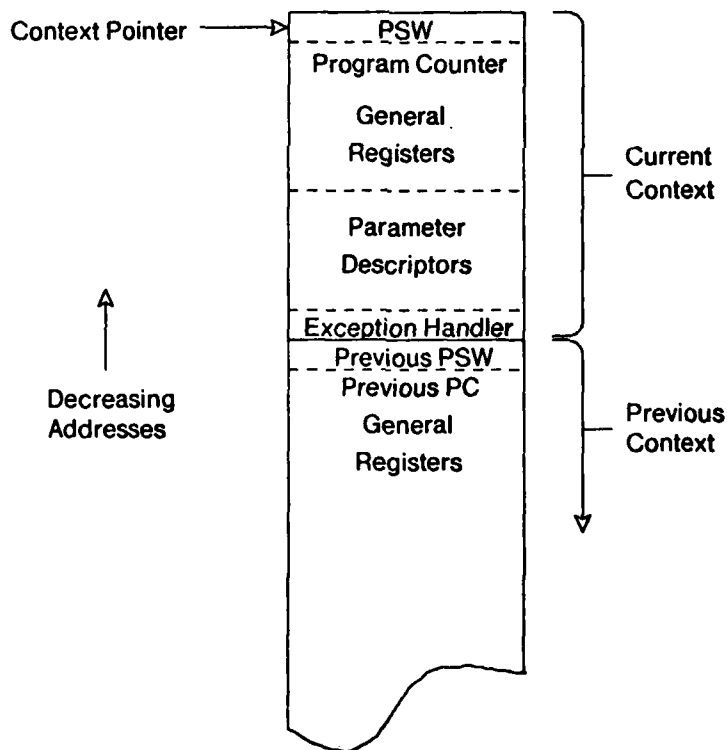


Figure 3: Stacked Procedure Contexts

2.3 Exception Handling

The NEBULA architecture handles program errors with an Ada¹-like exception facility. An exception handler is a piece of code to which control is to be transferred in the event of an exception. Such a handler may be defined for each procedure by use of the procedure entry or the EXCEPT instruction:

EXCEPT handler_address

The address of the exception handler for the current procedure may be altered by use of this instruction.

An exception may be raised by a hardware detected error or by execution of the RAISE instruction:

RAISE #{exception code}

The exception code is a 16-bit integer that is made available to the exception handler.

The action taken when an exception is raised may be selected from two options for each procedure. The options are:

- Branch to the exception handler specified by the procedure. If none is specified, the exception is propagated to the caller.
- Invoke the Supervisor Exception Handler.

If the supervisor exception handler is invoked, it may, after analyzing the situation, force the exception back to the procedure's handler by executing:

ERP #{exception code}

Transfer to user specified exception handlers is accomplished by a hardware forced jump. The exception handler may then determine the nature of the exception by executing:

ECODE Destination

that stores the exception code in Destination and resets the exception mechanism. The exception handler may deal with the exception and continue execution of the procedure in some altered manner, or it may pass the exception to the caller by:

ERET #{exception code}

that returns to the calling procedure and raises the specified exception.

¹Ada is a registered trademark of the US Department of Defense.

If an exception propagates completely up the execution stack, the entire execution context is removed. A new context is established by a call to the supervisor exception handler, and the exception Task_Failure is reported.

2.4 Modes of Operation

The Nebula architecture provides for independent selection of the following facilities:

- Kernel/Task Context. The kernel context is used for functions related to the overall system while a separate task context is dedicated to each task.
- Supervisor/User State. Programs executing in the supervisor state have access to the full address space as defined by the current maps. Programs in user state are restricted to the user map.
- Privilege. A privileged program may execute privileged instructions and access areas of memory designated as privileged by the memory maps.

This independent selection allows the processor to run in one of eight modes of operation. The architecture provides a controlled means of making transitions between procedures with various of these capabilities through vectoring and special instructions.

2.5 Task Control

A task is defined by its memory map (defines accessible code and data) and its context stack (defines its current state of execution). Nebula provides privileged instructions to manipulate tasks.

The currently active task is changed by the:

LTASK	- load a new task
STASK	- save the current task

instructions and is specified by a two word block containing the **Context Pointer** and the **Memory Map Pointer** for that task.

Execution control is accomplished with three instructions:

TINIT (PINIT)	- create an executing procedure
TSTART (PSTART)	- restart an existing procedure
TRAISE (PRAISE)	- force an exception into an existing procedure

These instructions allow tasks to be created and manipulated in an effective manner.

2.6 Timers

A trusted operating system must have the assurance that it will regain control of the processor at some time after it gives control to an untrusted task. This is usually accomplished through the use of a timed interrupt mechanism.

The Nebula architecture provides four timers. These are 32-bit down counters with

- Independent Control
- Software-assigned interrupt priority

Counting can occur in two modes:

- One microsecond time base
- Task instruction count

In task instruction count mode, each instruction execution in the currently active task causes a decrement. Kernel and interrupt functions do not affect the count. When a timer is decremented to zero, an interrupt is generated and counting continues. The timer mechanisms are protected from the effects of instructions with potentially long execution times by making such instructions interruptible.

2.7 Interrupts

NEBULA provides a 32-level priority interrupt structure. Hardware interrupts are assigned a priority by the requesting device. Software interrupts may be generated by setting the appropriate bit of the Software Interrupt Request Register. In the event of equivalent priority requests, the implicit hierarchy is:

- Processor
- Hardware Interrupts
- Software Interrupts

Interrupt handling is controlled by an interrupt vector selected by the interrupting device (fixed for software interrupts).

2.8 Memory Management System

NEBULA provides a conceptually simple memory mapping and protection facility utilizing variable size segments. Two processor registers contain pointers to the Supervisor and User maps in physical memory. The map used for a particular virtual address is determined by the most significant bit of the virtual address.

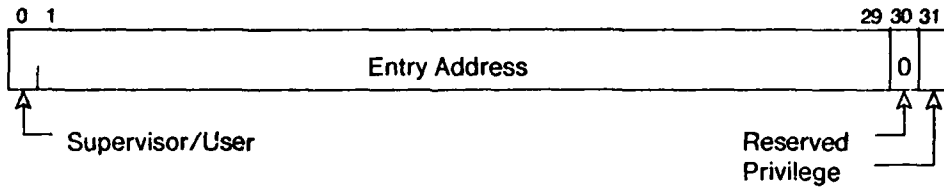


Figure 4: Interrupt vector format

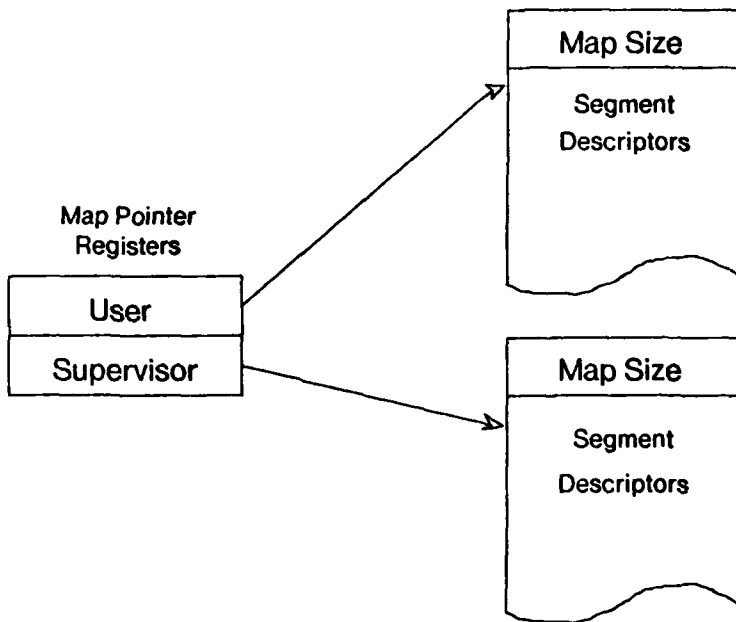


Figure 5: Two active memory maps

The map pointer registers contain a pointer to the first map entry of the respective map in memory. Bits 30:31 of this register allow relocation and protection to be enabled or disabled.

The map entries are double word descriptors containing the virtual address limit, relocation amount, and protection information for each segment.

The position and length of segments is truly variable. Each segment is specified by its upper address limit and that of the previous segment.

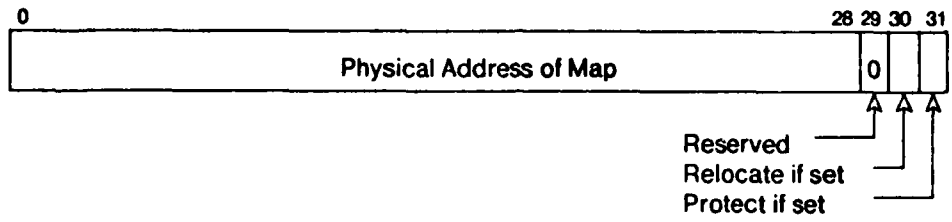


Figure 6: Layout of the map pointer register

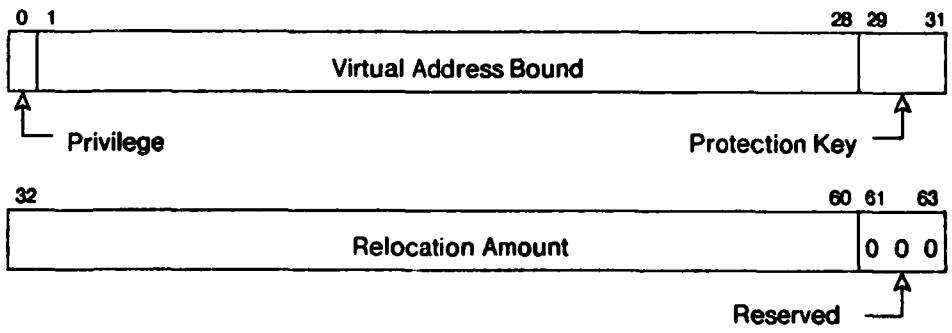


Figure 7: Memory map entry

Figure 9 shows the mechanism used to translate virtual addresses to physical addresses.

Each map entry specifies whether access to the corresponding segment is a privileged or unprivileged operation. Additionally, the type of access may be restricted to one of the following:

- No Access
- Instruction Access Only
- Data Read Only
- Instruction or Data Read Access
- Data Read/Write
- Context Access Only

Note in particular that pages allocated for context stack use can be accessed EXCLUSIVELY by that means.

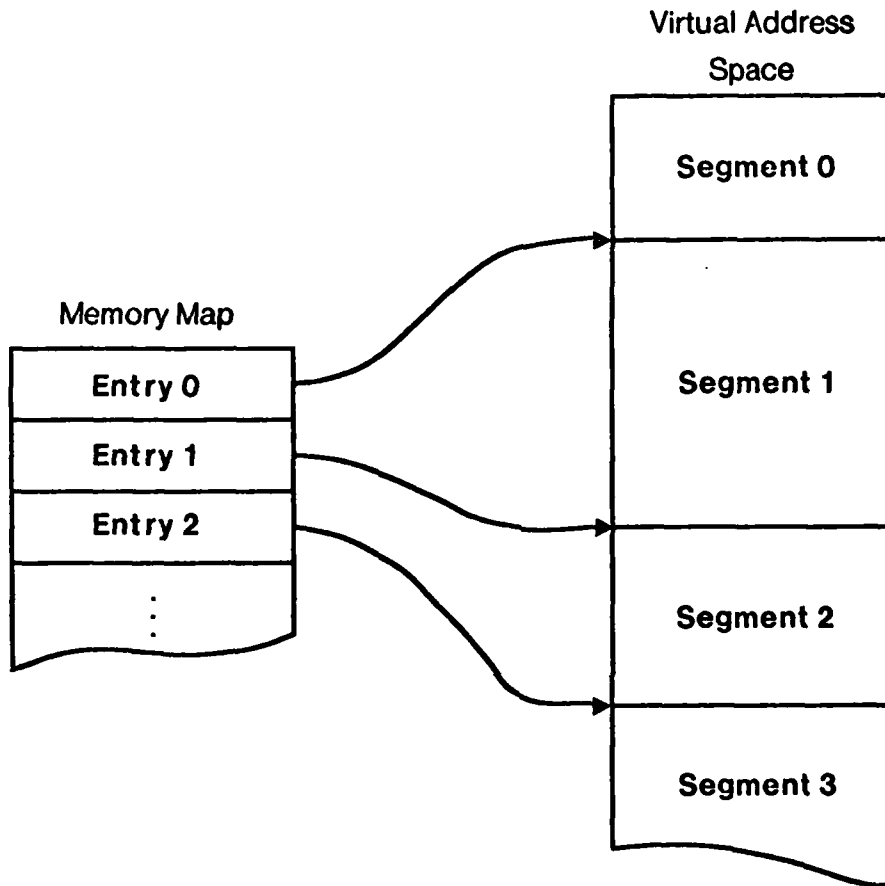


Figure 8: Map entries establish segment boundaries

2.9 I/O Control

The basic architecture allows control of I/O operations by access to I/O control registers. These device dependent registers are addressed as memory locations in the low 2^{20} bytes of physical address space. The generality of this scheme is highly desirable in dedicated interactive control operations.

Additionally, MIL-STD-1862A specifies a standard I/O controller structure for use in higher volume but less interactive situations. The IOC appears as a separate programmable processor with a limited instruction set. The IOC provides:

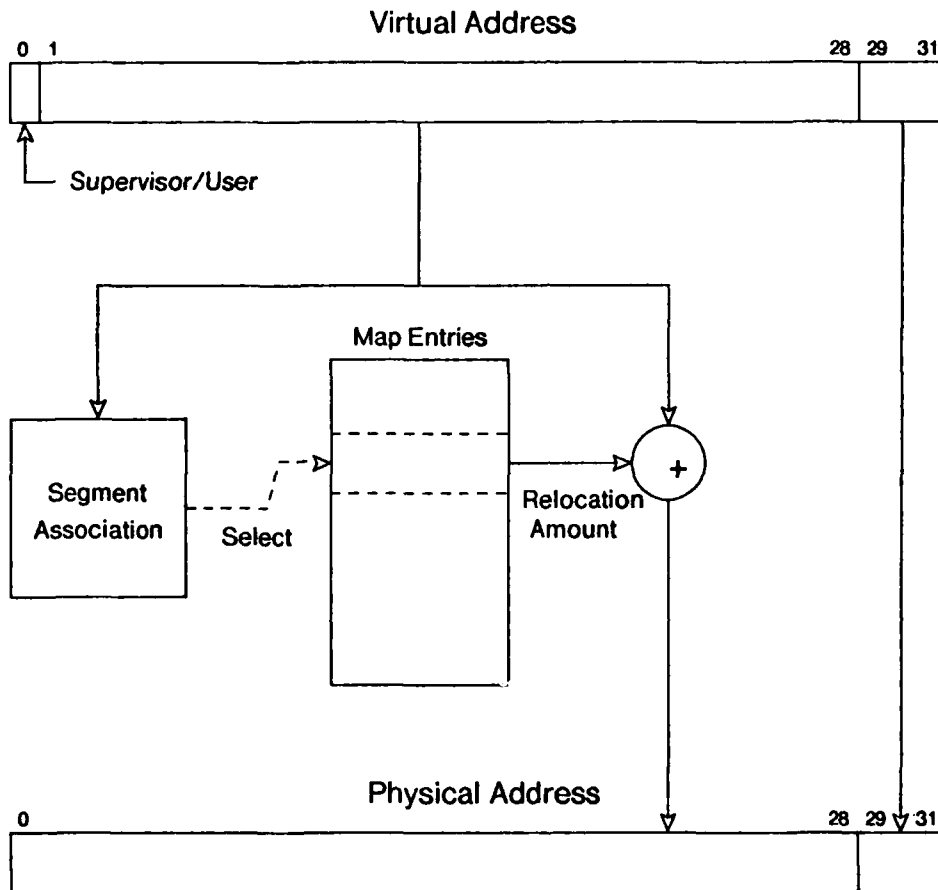


Figure 9: Address translation

- DMA transfers to connected devices
- Sequencing of complex I/O operations
- Queuing of I/O requests
- Programmable processor interruption
- Virtual Addressing (Secure User I/O)

A significant problem with programmable I/O controllers is that the IOC programs tend to diffuse into the system software. While self-modifying code is considered an atrocity when used in CPU programs, it is more or less the standard in IOC software. In fact, the situation is usually worse in that one program is dynamically re-writing another program that is potentially executing in parallel.

The Nebula IOC design attempts to avoid this situation by:

- Allowing transfer specific information to be separated from the channel program instructions.
- Prohibiting the modification of channel programs (memory protection).
- Allowing transfer parameters to be independently specified.

The IOC design attempts to build a "wall" between the CPU and IOC programs so that neither can see the other. Communication between the CPU and IOC is in the form of "signals" (initiations and interrupts) and messages.

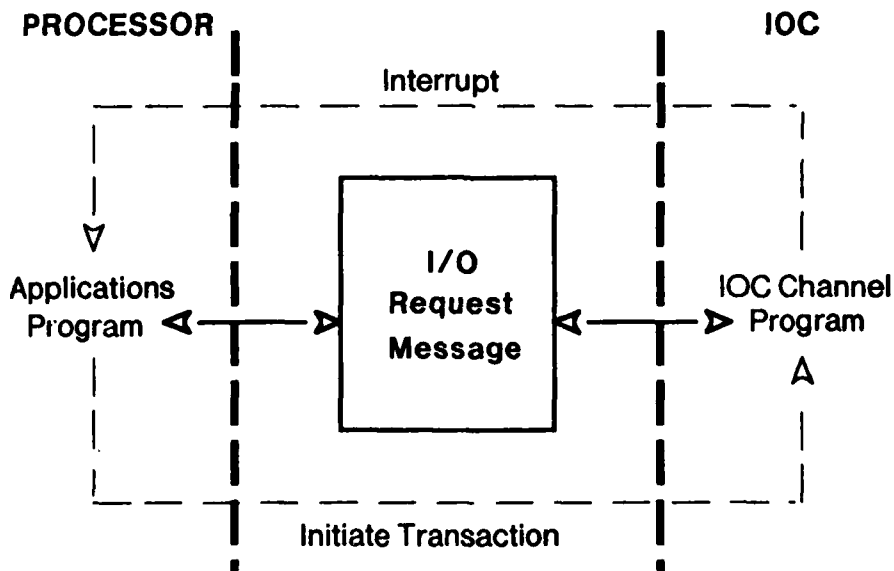


Figure 10: Processor - IOC communication

The IOC program functions as a "procedure" or "subroutine" executing in parallel with the caller. The "parameters" for a particular I/O request are contained in a message block.

A message block can contain:

- Commands

- Buffer Addresses (Virtual)
- Data Counts
- Status Masks or Return Areas
- Data

IOC operation is controlled by access to control registers within the I/O space. Each IOC has a set of "privileged" control registers that permit the specification of:

- Interface operational modes and timeouts
- Allowable interrupt priorities
- Accessible segments in memory (specified using the privileged SETSEG instruction).

Additionally, each IOC has an unprivileged control register set that contains:

- The Channel Program Counter (Virtual)
- The Message Address (Virtual)
- Device Control and Status Registers

Access to these registers may be granted to untrusted users.

The independent specification of segments accessible to the IOC allows IOC transfers to be initiated by a variety of means:

- Executive initiates transfers to exec or user buffers.
- User requests transfers but executive specifies channel program.
- User specifies both program and transfers.

Over this entire range of I/O protocols, the overhead of initiating an I/O operation is on the order of a procedure call.

3 Nebula Design Considerations

3.1 General Purpose Architecture

When the standardization of Computers for the military was first being considered by the Computer Family Architecture (CFA) committee in 1975, a question arose about the nature of a military computer architecture versus a commercial computer architecture. The implementation differences are obvious. The military computer must generally operate in a much harsher environment than its commercial counterpart. However, after careful comparison of the features of general purpose military and commercial architectures, it was determined that the data processing operations performed by military computers do not differ significantly from those typically performed by commercial computers [1]. This conclusion allowed us to draw upon knowledge gained about all

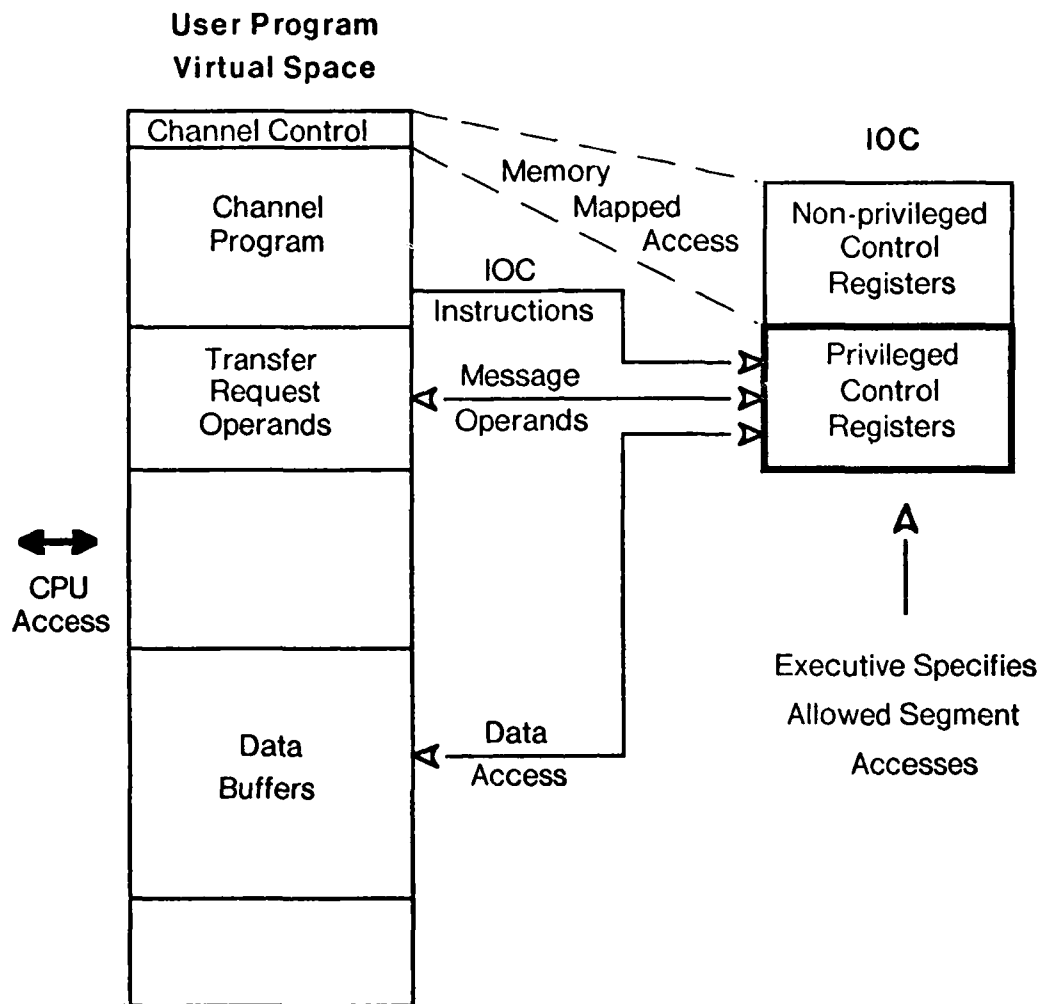


Figure 11: IOC Virtual Addressing

general purpose computers and apply it to the design of an instruction set architecture intended as a military standard.

A computer may be fully general in function but if it lacks the speed required to perform the functions demanded by most typical applications, that computer cannot be considered as general purpose. Even though the absolute speed of a computer system falls within the domain of the implementation rather than the architecture, the relative efficiency of the architecture can impact both the performance and cost of implementations. In developing the Nebula Standard,

consideration was given to the knowledge gained from four years of research in architecture efficiency that was conducted at Carnegie -Mellon University [2]. Some of that knowledge can be condensed into the following set of heuristics:

- Short literal operands should be provided.
- Variable-length instructions are efficient.
- A sufficient number of general registers should be provided.
- Maskable hardware priority vectored interrupts are important.
- General registers should have general function.
- Address computations should be efficient.

3.2 Compiled Code Efficiency

Since the use of high level languages is expected to increase in military systems, especially as AdaTM becomes more widely available, a standard computer architecture must be a good target for compiled languages in order to be efficient. In a recent article [5], Wulf described some of the principles that should be considered in computer architecture design for efficient compiled code.

These principles include:

- *Regularity.* If something is done one way in one place, it ought to be done the same way everywhere. This principle has also been called the "law of least astonishment" in the language of the design community.
- *Orthogonality.* It should be possible to divide the architecture definition into a set of separate concerns and define each in isolation from the others. For example, it ought to be possible to discuss data types, addressing, and instruction sets independently.
- *Composability.* If the principles of regularity and orthogonality have been followed, then it should also be possible to compose the orthogonal, regular notions in arbitrary ways. It ought to be possible, for example, to use every addressing mode with every operator and every data type.

These principles, in addition to others, were followed in the Nebula standard resulting in an architecture that allows optimizing compilers to make full use of the technology available without the complicated and extensive analysis required in optimizing compilers targeted to many existing architectures.

The Nebula architecture is regular.

- The same procedure call mechanism is invoked by Call instructions, Supervisor calls, the opcode exception mechanism, traps, interrupts, and the supervisor exception handler.
- Operands to procedures look like instruction operands.
- Independent source and destination forms of instruction operands are provided (sometimes optimizations are also provided).

- The order of instruction operands is regular.
- All simple branch instructions have two displacement forms.

The Nebula architecture provides orthogonality. Data types, operand addressing and instruction sets can each be dealt with independently.

There is composibility in the Nebula architecture. In general, any addressing mode can be used to define any instruction operand. Any addressing mode can be used to access any data type.

3.3 Implementability

In order to insure the successful implementation of the standard, Nebula had to be a low risk design. New ideas were carefully considered and "experimental" features were avoided. This led to what is considered a safe, yet innovative design. Part of this innovation has to do with the approach taken to allow the effective use of new and advancing technologies while still maintaining compatibility for a full family of proposed processors.

In order to increase the range of *implementability* of the Nebula architecture, we adopted the following *visibility goals* [4]:

- First, the architecture should increase the visibility of program operations to the hardware. This allows hardware enhancement and optimization of these functions.
- Second, the architecture should reduce the visibility of the hardware to the software in order to provide greater freedom of implementation for the hardware designer. This allows more freedom for the implementor in the use of different technologies and strategies of implementation while still maintaining software compatibility.

The computer architecture is the interface between the hardware and the software. As with any interface, the architecture allows the software to communicate with the hardware.

Older computers, and even some that are being built and used today, allowed complete software visibility by simply using the hardware itself to specify the architecture. This results in software that probes and explores the hardware to take advantage of every little "feature" that can be found and possibly used to some perceived advantage. The problem with this approach is not discovered until a new implementation is required, but the software base must be captured. Then either the existing implementation must be duplicated exactly (usually to the great disadvantage of the new technology) or the software base must be updated (at great expense) to accommodate any changes.

COMPUTER ARCHITECTURE (interface)

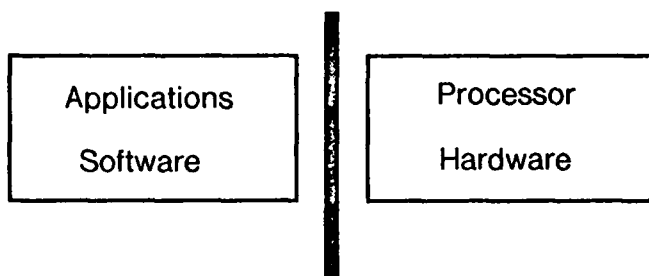


Figure 12: Interface between the software and the Hardware

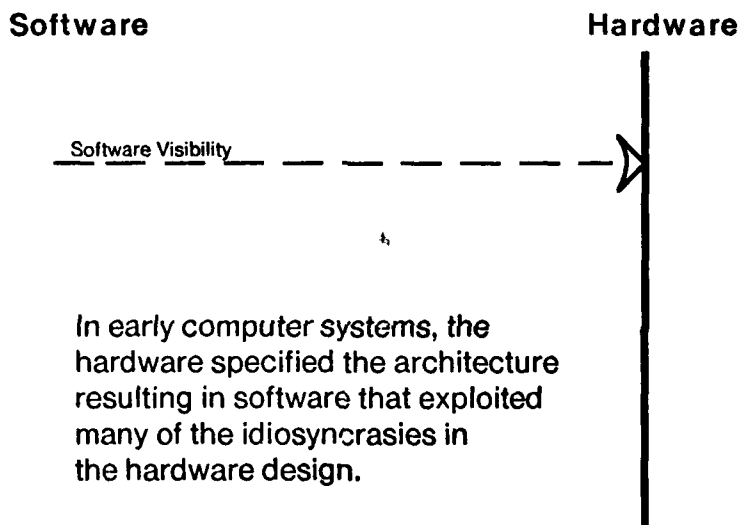


Figure 13: Complete visibility by the software

The other extreme is to make the software, or high level language definition, completely visible to the hardware. This approach results in a "direct execution" architecture. This architecture would only be able to use a single language effectively. In addition, any changes to the language definition will require changes to the hardware of the computer. This approach requires a enormous hardware development investment because all of the complexity normally associated with the compiler is pushed into the hardware.

Our approach with Nebula falls in between these two extremes. Increased hardware visibility of selected functions allows advancing hardware developments to absorb some of the burden that is

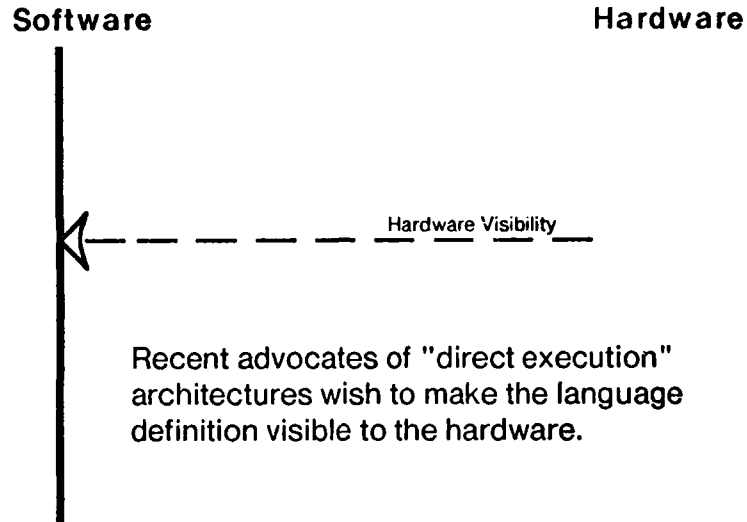


Figure 14: Complete visibility by the hardware

often relegated to software in older systems. These functions were selected based on their usefulness to a variety of high level language environments. The software's visibility of the exact implementation of these functions is restricted. Part of this restriction is enforced by the hardware and part is based on specification alone.

The best example of these visibility goals exhibited in the Nebula architecture is the procedure interface.

The term procedure interface refers to the state visible to a called routine, the manner in which the caller specifies this state, and the means by which parameters are passed. In a conventional register architecture, the caller and called routine see a common register set. This property of the *hardware* is then managed by various *software* conventions for preserving the caller's data, making registers available to the called routine, and passing back values. Thus, the procedure interface is largely software defined to make use of the hardware.

Nebula considers registers to be local to the procedure using them. Registers are allocated on the call, and deallocated by the return. Storage for the local registers is allocated in the context stack. This stack is protected and inaccessible to the software. The effect is that register usage of each procedure is *visible* to the *hardware* while the mechanism of allocating these local registers is *invisible* to the *software*.

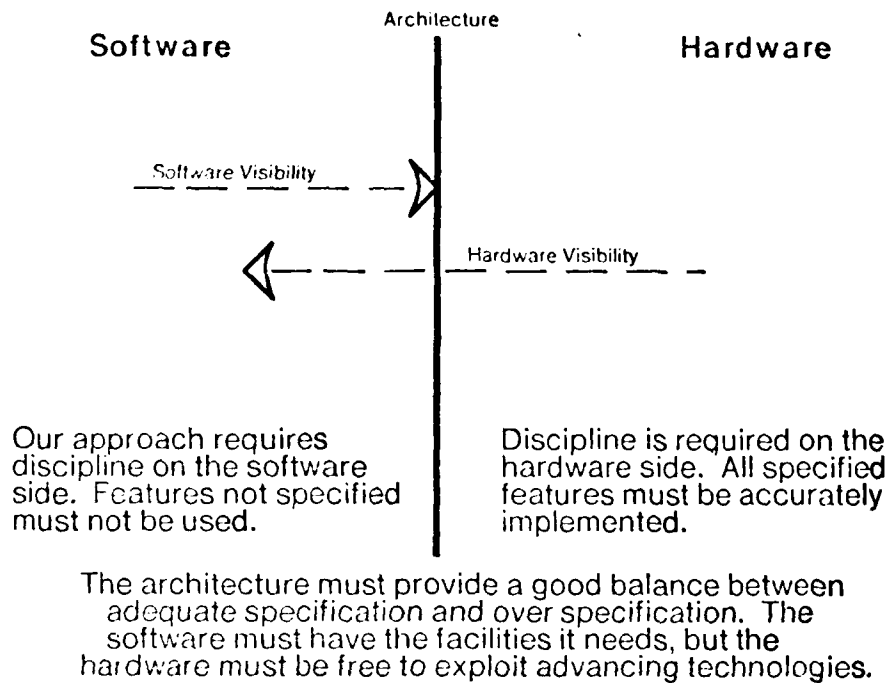



Figure 15: The Nebula approach to visibility

References

1. Burr, William E. and Robert Gordon. "Selecting a Military Computer Architecture." *Computer* 10, 10 (October 1977), 16-23.
2. Dietz, William B. and Leland Szewerenko. "Architectural Efficiency Measures: An Overview of Three Studies." *Computer* 12, 4 (April 1979), 26-33.
3. *Nebula Instruction Set Architecture, MIL-STD-1862A*. Naval Publication Center, Philadelphia, PA (2 November 1981).
4. Szewerenko, Leland, William B. Dietz and Frank E. Ward, Jr. "Nebula: A New Architecture and Its Relationship to Computer Hardware." *Computer* 14, 2 (February 1981), 35-41.
5. Wulf, William A. "Compilers and Computer Architecture." *Computer* 14, 7 (July 1981), 41-47.



Transportability of Nebula Software

Guy L. Steele Jr.

Computer Science Department
Carnegie-Mellon University
Schenley Park
Pittsburgh, Pennsylvania 15213
(412)578-2586
ARPANET: STEELE@CMU-20C

Author's Biography

Guy Steele has been involved with the Army's standardization effort since 1980. His interests include programming language design and implementation; programming language semantics, including denotational semantics; computer architecture design, specification, and validation; compilers for high-order languages; storage management techniques; hardware/software support systems for artificial intelligence; and the design of VLSI circuits. Steele received the A.B. degree in Applied Mathematics from Harvard College, and the S.M. and Ph.D. degrees in Computer Science and Artificial Intelligence from the Massachusetts Institute of Technology. Steele is presently Assistant Professor of Computer Science at Carnegie-Mellon University.

Abstract

~~We~~ discuss various aspects of the Nebula architecture definition that are purposely implementation-dependent. These might seem at first glance to present obstacles to software portability.

These implementation dependencies are intentionally included in the architecture definition to allow implementors the freedom to evolve new implementation strategies and to exploit advancing technology. Portability problems are alleviated by a combination of specification conventions, explicit protection mechanisms in the hardware, and special software modules that can be conventionally used to hide implementation details.

The result is an architecture definition that allows Nebula software to be compatible across a range of hardware implementations, while enabling more cost-effective hardware realizations by allowing implementors the freedom to explore a variety of implementation techniques.

1 Introduction

The Nebula architecture was designed to allow software to be compatible across a broad family of implementations. Other architecture designs, notably that of the IBM System/360 and System/370 series [2], have had this goal also. Normally such compatibility is achieved by carefully specifying every aspect of the hardware that could conceivably be visible to the programmer. However, such an approach usually overspecifies the hardware, in that restrictions that are not useful to software are placed on the hardware for the sake of having a complete specification.

These problems arise because the architecture specification interface (the description of what the implementation must provide and what the programmer may assume) is assumed to lie precisely at the boundary between hardware and software. Hardware is the domain of the implementor; the racks and circuits and registers and microcode must implement all the details of the architecture. Software is the domain of the programmer; any bit pattern that the programmer can construct is fair game, and may be given to the hardware purporting to be a program. The programmer expects to be told (by the architecture specification) the meaning of every bit in every register.¹

Nebula takes a different approach. While the architecture specification lies *approximately* at the hardware/software boundary, there are deviations designed to give the implementor maximal freedom to exploit new technology. In return for this freedom, implementation-specific software modules must be provided for each implementation; these modules support an implementation-independent functional interface to implementation-dependent aspects of the hardware. Similarly, the specification explicitly tells the programmer that certain information is none of his business; but in exchange for this lack of information the programmer is rewarded with more efficient program execution.

As an example, concerning the procedure call mechanism, the programmer is simply told: "In order to perform procedure calls, you must first reserve a region of virtual memory for use by the hardware. (There is a rule of thumb for calculating the necessary size of this region.) You must specify a certain mode of protection for the region to the memory management system. You must initialize a register to point to this memory region; and thereafter you must *keep your hands off that memory*." Only a little bit is said about what is put in that memory, and very little is said about the format in which that information is stored. The implementor is left free to choose data formats and implementation

¹Throughout this paper we will use "implementor" to mean a person or organization that constructs hardware instantiations of the Nebula architecture, and "programmer" to mean a person or organization that writes programs to be executed by the Nebula architecture.

strategies that will maximize the efficiency of the procedure call mechanism. This will be discussed in more detail below.

In some cases the programmer is prevented from meddling with data reserved to the implementor by means of explicit protection mechanisms similar to those used to prevent applications programmers from meddling with privileged instructions that are reserved to the operating system. In other cases the programmer is prevented from meddling merely by convention: the specification simply states that the results of certain actions by programs are undefined or unpredictable (but the unpredictable results of such meddling are of course always limited by the privileges of the meddling program).

In the remainder of this paper we will discuss four aspects of the hardware that are implementation-dependent but are carefully specified, protected, and software-supplemented so that they will be used in a completely portable fashion:

- Processor status word (PSW) bits reserved to the implementor.
- Implementation virtual address size.
- Procedure call context area.
- Memory map size.

2 Reserved PSW Bits

Nebula has a processor status word (PSW) that is used in the usual fashion to contain frequently-used information associated with the currently running task: condition codes, interrupt priority level, exception enable flags, whether the task is privileged, and so on. Two bits are explicitly reserved to the implementor. Now, in fact, these bits were provided for the purpose of indicating whether an instruction had been interrupted and not yet completed. For example, there is an instruction that will copy an arbitrarily long string from one region of memory to another. If Nebula is to provide adequate interrupt response (not to mention responding to memory traps), it must of course be possible to stop in the middle of the instruction, service the interrupt, and then resume the interrupted instruction. When the PSW is stored in response to the interrupt, the reserved bits may be used to indicate that the interrupt occurred during an instruction rather than between instructions.

To use a PSW bit for this purpose is not unusual. For example, DEC PDP-10 [1] had such a bit, to indicate for example that the "increment and load byte" instruction had performed the "increment" but not the "load byte" when an interrupt or trap occurred.

What is unusual is that the Nebula architecture specification nowhere says formally how these reserved bits are to be used or what non-zero values of these bits mean. Neither is any implementation required to make any actual use of them. The programmer is simply advised that the bits may be safely initialized to zero; that interrupts and traps may produce non-zero values in those bits when the PSW is stored; that if the field is loaded back into the PSW exactly as it was stored, then any interrupted instruction will be properly resumed; and that the effects of altering the field (other than to load a previously stored value) are unpredictable. This very abstract specification allows the implementor to experiment with various ways of using these bits. By contrast, the PDP-10 architecture specification explained in concrete detail all possible results of altering the bit, thereby allowing the programmer potentially to take advantage of this "feature" in unexpected ways, and thereby requiring all future PDP-10 implementations to support the precise same semantics for that bit.

It should be noted that general access to the PSW is restricted to privileged programs such as the operating system; applications programs normally cannot even examine the two reserved bits. This is a good example of a general rule of thumb used to decide in any instance whether simple specification is adequate prevention or whether an explicit protection mechanism is necessary. Privileged programs can be trusted to some extent, and are likely to be written by somewhat more expert Nebula programmers. Such programs can therefore be assumed to abide by simple specifications (such as to properly restore a PSW). Application (user mode) programs, however, are generally not trusted, and are as a rule walled off by explicit hardware mechanisms.

3 Implementation virtual address space

The Nebula architecture specifies a 32-bit word size. Addresses (that refer to 8-bit bytes) are also, at least potentially, 32 bits wide. This specification for addresses was made to allow for future growth: in the near term, 28 bits of address should be plenty, but in ten years 32-bit addresses may be necessary for certain applications.

However, there are many places in an implementation where an address plus a few associated bits must be stored together (for example, the address of an operand plus two bits encoding its size as byte, halfword, word, or doubleword). It would be very convenient and significantly more efficient if the address plus the extra bits could be fit together into 32 bits. This would be easy if addresses were only 28 bits long, but of course is impossible if addresses are a full 32 bits wide. It would be a pity to force implementations to be significantly less efficient for the next five years, even for the sake of future flexibility.

Nebula solves this tradeoff with a compromise. The *architecture* address space is defined to accommodate 32-bit addresses. This is the set of addresses that can be generated by instruction operand address calculations. However, there is also an *implementation* virtual address space that is permitted to accommodate only addresses of a somewhat smaller width. This is the set of addresses that can be accommodated by the virtual memory mapping hardware.

Certain loose constraints are placed on the implementation address width (it must be at least 24 bits, and must be at least 3 bits wider than the width of a physical memory address), so the programmer is assured that the virtual address space is no smaller than a certain minimum size. It is assumed that the operating system will know the implementation virtual address size and limit tasks to the corresponding amount of virtual memory. While execution of certain devious operations will reveal the precise implementation virtual address size to an application program, the limitation is extremely unlikely to have any effect on a normal program.

The implementor is therefore free to construct an implementation that provides only 28-bit or 24-bit virtual addresses but that, in exchange for the limitation, possibly has significantly higher performance (for a given level of technology) than an implementation that supports the full 32-bit virtual address space.

4 Procedure Call Context Area

The notion of a *procedure call* pervades the Nebula architecture. Procedure call instructions look like ordinary instructions, and arguments to procedures are specified in exactly the same way as instruction operands. Executing a procedure call suspends the current "context" and begins execution of the called procedure in a fresh context.

Conversely, unimplemented instructions simply perform procedure calls to specially vectored handlers; the handler receives the instruction operands as its arguments. (This allows the instruction set to be expanded in the future and retrofitted to earlier hardware merely by providing ordinary procedures, possibly written in a high-order language, to simulate the missing instructions.) Other traps simply abort the current instruction and instead perform a procedure call to a trap handler. Interrupts suspend the current context and forcibly execute a procedure call to the interrupt handler, which executes in a fresh context.

Whether a new context was generated by an explicit procedure call, a trap, or an interrupt, if the currently running procedure executes a return instruction, the context is discarded and the previous context is resumed transparently.

What is a context? It is the set of information and resources necessary to execute a single procedure invocation within a task. This includes a PSW, information about the procedure's parameters, the address of the exception handler (if any), and a set of registers. Every procedure call establishes a fresh set of registers, thereby eliminating the problem of register allocation in the presence of procedure calls.

This is about all the programmer needs to know about procedure calls, except for the details of which instructions to use and how to access parameters. Calling a procedure saves the current context, establishes a new context, and starts executing the new procedure; returning discards the current context and resumes the previous context. There are also special instructions that are used to switch tasks. These guarantee to preserve the current task context and to properly start up the new task context. The only problem is that procedure calls may be nested indefinitely, which means that *no a priori* limit can be placed on the amount of context information that must be saved. It is therefore impractical to expect context information to be held entirely in the CPU; at least some of it must, under some circumstances, be kept in main memory.

This is the purpose of the *context area* that was referred to above. The context area is the region of memory that the programmer must initialize and then keep away from. The region of memory must be reserved for the purpose; this is the *joint responsibility of the application programmer and the operating system*. The address of the region must be loaded into a special *context pointer* register in a manner prescribed by the architecture specification.

The most important point is that the context area must be protected by the memory map in a certain manner. The memory map, as is usual, not only describes how to translate virtual addresses into physical addresses, but also describes what kinds of access are permitted for various regions of memory. Typical access modes are *read-only*, *read/write*, *execute-only* (that is, instruction-fetch-only), and *no-access*. Nebula provides yet another access mode: *context-only*. The hardware will refuse to treat a region of memory as a context area unless its access mode is *context-only*; and if a region of memory is *context-only* then it may not be read or written in any manner other than in connection with hardware-supported context-area operations such as procedure call and return.

This specification provides memory to the hardware needed to support procedure calls. It prevents the application programmer from meddling with it, or even examining it, by means of explicit protection mechanisms. Nevertheless, the nature of the information stored in the context area is only loosely prescribed, and the format of the information, or even precisely when it is stored, is nowhere defined. Once again this vagueness in the architecture specification is intentional. An

implementation is free to keep all context information (even the current set of registers and the PSW!) in main memory; this may be appropriate for implementations where small size and low cost are more important than performance. On the other hand, an implementation may provide an elaborate cache for the context area, such that the context area is never stored into or loaded from except for cache overflow or task switching; this may be appropriate for an implementation with extremely high performance. An implementation with a 28-bit or 24-bit implementation virtual address space may be able to use a *much more compact format for the context area than one that provides 32-bit virtual addresses*, thereby reducing the number of memory cycles necessary for procedure calls and parameter access.

Now, it is of course impractical not *ever* to examine a context area. The operating system must at least be able to force the hardware to sweep any caches and store all context information back into main memory, if only so that tasks may be swapped out or checkpointed. Special instructions are provided for this purpose: the "store task" instruction guarantees to place all information related to the task in main memory in such a way that "load task" can resume the task properly. These instructions merely provide safe ways to load and store the context information; they do not provide for examining the information.

While it is best for a process not to meddle with its context area in the ordinary course of things, it is desirable for a debugging routine to be able to examine the context area in order to print a traceback of outstanding procedure calls, examine the register sets of various procedure contexts, and so on. The solution here is to divide the work between the implementor and the operating system. The implementor is responsible for delivering with the hardware the implementation-specific information necessary to *construct a set of software modules that provide an abstract interface to the context area*. The software is implementation-specific, in that it must contain details of the context area formats used by the particular hardware; but it performs services that are defined in an implementation-independent manner.

This implementation-specific software must be given privileged access to context areas, and so it must be trusted. It must therefore be incorporated into the operating system, the rest of which can be written in an implementation-independent manner (as far as context areas are concerned), because all implementation-specific operations are performed by the implementor-provided software modules. The operating system in turn provides debugging services to the application program, without ever actually letting the program directly access its own context area.

Notice again the general rule of thumb. The operating system can be trusted to abide by

convention, and access a context area only through a special software interface. An application program cannot be trusted, and is forcibly prevented by the memory management system from improperly accessing the context area.

It was suggested several times, by this author and others, in the course of development of the architecture specification, that perhaps special instructions should be defined for accessing the context area. The implementor could then insure that the access provided by these instructions was consistent with the context area format used throughout the implementation. This would increase the hardware costs, however, if only in the form of extra microcode. But the function can be provided in software at least as easily, and performance is not at all critical for debugging applications. Ideally, consistency could be insured simply by giving the responsibility for writing the software modules to the implementor rather than to the author of the operating system. Overall, the use of well-defined software modules to provide implementation-independent interfaces to implementation-dependent aspects of the hardware appears to be the most cost-effective solution.²

5 Memory Map Size

Nebula uses a rather unusual format for its memory map. Rather than dividing the virtual address space up into pages of fixed size and using a page table (which can be very large for such a huge address space!), Nebula divides the virtual address space into variable-length "segments". Each segment may be arbitrarily large, but must reside in a contiguous region of physical memory. This contiguity requirement imposes a constraint on the operating system. To make memory allocation more flexible, it is desirable for a task to be divided up into as many segments as possible. On the other hand, it is also desirable, for reasons of speed, to be able to store the entire memory map in an associative cache, because the map must be consulted on nearly every memory access! For a given level of technology, associative caches are impractical above a certain size, placing an upper limit of the number of segments supported by the memory map.

Nebula has attempted to resolve this with a compromise: an implementation must support *at least* 16 segments, but may support more for increased performance. This is possible because a memory

²It should be mentioned that Leland Szeverenko has written a fairly short (300-line) program that, when executed with suitable privileges on Nebula hardware, probes the context area and uses a number of heuristics to attempt to decipher the context area format used by the hardware. The heuristics may not succeed in every case, but they are amazingly effective for "typical" hardware implementations. If the heuristics succeed, then the program automatically generates a table of constants and offsets from which the software interface modules can be generated automatically. Because of this, we expect that very little programmer time need be expended in the future to generate these software modules. The use of modules is therefore not merely a cheap solution, henceforth it is very nearly a free solution!

map, like most data structures kept in memory by the hardware, has a variable-length format: the first word of the map contains the length of the rest of the map. (This allows any map cache to be loaded much more quickly if the map has only a few segments defined in it; it is not necessary to load a full 16 entries.)

This compromise was justly criticized as being useless. Programs (it was argued) will tend to use specific segments for specific purposes, such as instruction areas, data areas, and context areas. If a program uses more than 16 segments, then it will not be portable, because some implementations will support only the minimum number of segments; while if no program uses more than 16 segments, then it is fruitless for any implementation to support for than 16 segments.

It was suggested that if the memory map were large enough, one could get most of the benefit of a page-oriented map simply by making all segments the same size and calling that size a page. With this different pattern of usage it would make more sense to provide a variable-size map. It was therefore proposed that there be *no* limit on the size of a memory map: a map kept in main memory could specify any number of segments. Granted, it would not then be practical to keep the entire map in a cache, but perhaps some implementations would cache the first 16 entries, or the 16 most recently used entries, and access main memory to get a map entry if the entry was not in the cache. In this manner one would get the same performance for programs that used fewer than 16 entries, and yet make it possible for programs to use any number of segments, and even use a paged virtual-memory strategy, making it possible for Nebula to be used for time-sharing as well as the originally envisioned military applications.

The counter-argument is that caching part of a map is much more complicated than caching all of a map. It probably requires more hardware to first try the cache and then go to memory on a cache miss if the same level of performance must be maintained. It takes some clever algorithms to maintain cache entries with a least-recently-used replacement discipline, particularly if it is optimized for special cases such as never deleting the cache entry for the current context area or the current instruction.

The key phrase here is "clever algorithms". Such algorithms are almost always better put in software than hardware. It was observed that operating system software could take the place of hardware in maintaining a cache. If the operating system is organized properly, it can provide an application program with the illusion of having many segments even though the hardware only directly supports some small number such as 16. This combined hardware/software approach currently appears to be the most cost-effective solution to Nebula memory management.

This approach therefore suggests that the Nebula architecture specification should remain unchanged. Memory maps should be of variable size, and a given hardware implementation must support maps of up to 16 segments, but is permitted to support more than 16. There also must be a way for the operating system to know exactly how many segments are supported.

Let the map directly supported by the hardware be called the "actual map". The operating system must also maintain for each task a "logical map" that may contain arbitrarily many segments. When a task is to be run, some subset of the segments in its logical map must be selected (a "working set") and placed together to form an actual map. This actual map is then specified to the hardware when the task is resumed. As long as the application task stays within its working set, it may run at full speed with the (possibly cached) actual map. If the task tries to touch a logical segment outside its working set, a trap occurs, and the operating system must displace some segment in the working set with the new segment needed by the task.

This solution is more flexible and cost-effective than trying to support maps of arbitrary size in hardware. The cache replacement algorithm is entirely in software, and so can be much more clever and elaborate than any algorithm that could be practically embedded in hardware or microcode. More important, the cache replacement algorithm can be tuned for particular applications. Memory accesses can actually be a bit faster in the typical case because the virtual address mapping process is simpler. Most important of all, whether to use a hardware map cache, and if so, exactly how large it should be, are questions left to the implementor to decide. The operating system can be written to take full advantage of such a cache no matter what its size. As technology improves, allowing larger map caches, new Nebula implementations may take advantage of it for improved performance, in a manner completely transparent to application programs and compatible with appropriately written operating systems.

6 Conclusions

Software portability is achieved for the Nebula architecture, not by tying down the meaning of every last hardware register and bit in the architecture definition, but by defining an abstract interface visible to the programmer that is implemented by a combination of hardware and software. Parts of the architecture, such as the context area, are like "abstract data types"; the programmer is told specific ways in which they may be used, but is not told all the details of their implementation.

The Nebula architecture specification intentionally does not tie down details that the programmer need not know. This gives implementors of Nebula hardware the necessary freedom to provide more

cost-effective implementations over a wide performance spectrum. Certain aspects of the architecture are purposely left undefined or are reserved to the implementor so that new implementation technologies and strategies may be exploited in the future.

Coordinated, machine-specific software can provide a compatible, implementation-independent interface to the programmer, by mediating between an abstract, implementation-independent interface specification and the machine-specific hardware details. This combined hardware/software solution to certain portability problems is much more cost-effective than a solution expressed purely in hardware.

References

1. *DecSystem 10 Assembly Language Handbook (third edition)*. Digital Equipment Corporation (Maynard, Massachusetts, 1973).
2. *IBM System/370 Principles of Operation*. GA22-7000-5 edition. International Business Machines Corporation (Poughkeepsie, New York, 1976).

AD-P003 534

THE IMPACT OF NEBULA, MCF, AND ADA ON REAL-TIME
EMBEDDED COMPUTER SYSTEMS

F. E. Wuebker

RCA Government Systems Division
Missile and Surface Radar
(609) 778-3908

ABSTRACT

The 1862A Instruction Set architecture (ISA) is resulting in a Nebula - Military Computer Family Data Processing System. This combination will provide a significant improvement in software performances for both unit and distributed-processor real-time systems. The performance gain will be obtained from the short, compact code sequences resulting from the inherent power of the instruction set architecture (ISA), and from the inherent machine speed of the Military Computer Family. This gain obviates the need for any assembly-language programming for real-time applications. Productivity, reliability, and maintainability requirements demand that all future real-time applications be written in a higher order language. Experiments to date show that the Ada-Nebula combination results in the best source code to object code expansion values, hence best performance. Of the higher order languages most suited to real-time performance (Fortran, Jovial, CMS-2, and Ada), Ada with an Ada-based program design language provides the capability for the highest productivity. Traceability between design and code, using Ada and Ada PDL, and a machine processable documentation methodology will provide the reliability and maintainability desired by the software community.

DISCUSSION

In 1973-1974, a Department of Defense cost study¹ revealed that approximately 56% of the DoD software costs were for embedded systems. Furthermore, the same study showed, and our experience at RCA validates, that more than 50% of the cost for software is spent on life-cycle maintenance. That study further stated that because the annual DoD software bill would approximate 4.5 billion dollars per year, significant savings could be realized in software cost by reducing development costs while improving the reliability & maintainability of the product.

An instruction set architecture, Nebula, has been implemented in the latest-technology computers, such as the Military Computer Family (MCF), a group of embedded computer systems. The Nebula-MCF combination, used with the higher order language Ada*, results in the fusion of three current technologies to achieve improved reliability and maintainability for embedded computer systems.

Embedded computer systems primarily are used to control such larger systems as those involving radars, weapons, or satellite communications. The run-time efficiency of the computer programs, therefore, is of paramount importance because these systems must be able to function continually and accurately. Embedded system programs can be quite large, typically 300,000 to 1,000,000 lines of code. Also, the life cycle of embedded systems is long—typically 10 to 15 years. The Ballistic Missile Early Warning System, for instance, has already survived for 23 years.

Programs that have such long lives, however, are often maintained in later years by personnel who have no knowledge of the rationale and design factors that influenced the original design and implementation. Furthermore, embedded systems as a whole are subject to frequent modifications throughout their lives. From the Program Manager's perspective, the software for embedded systems represents a significant development cost and, typically, a significant schedule risk because software is a labor-intensive product. Additionally, the Program Manager wants the software to have 'robustness,' that is, the software must be capable of accepting changes and modifications without 'breakage' (having the software fail in such a manner that reprogramming of software segments is required). Historically, tightly written assembly-language programs have a large breakage factor.

The program manager must field reliable software. This reliability can only be achieved by building software without 'sneak paths of faulty logic' that result in errors that occur when a unique set of circumstances occur, or memory proceeds to become contaminated because of insufficient or inadequate garbage collection. Finally, the program manager also must build software for a machine that fits the system's requirements for space, power, reliability, and the environment. In the past, small space meant a limited capability machine. To cope with the machine capability and still have an efficient run-time software capability, the software had to be written in tightly coupled assembly-language, and was often the product of "creative super stars," with resultant poor or absent documentation.

The 32-bit SuperMini computers and the 32-bit micro-machines can provide computing power in sufficiently small physical packages to be useful to embedded computer systems. The Military Computer Family, which includes SuperMinis and micro-machines, will provide the computer power both in memory-addressing capability and in speed,

*Registered trademark of the US Government (AJPO)

permitting structured higher order language for the whole range of current military applications. This advantage can be exclusive of the instruction set implementation or the targeted higher order language.

The MCF, however, is being implemented using the Nebula Instruction Set Architecture (ISA), which provides additional benefits to the construction of embedded systems software. Nebula instructions are capable of many different contexts within each instruction, so optimum code generation can result for each higher order language statement. That is, the compiler often can build code that is as efficient as that produced by an assembly language programmer. A study² in December 1981 compared the amount of object code that would be generated by different architectures, including Nebula, for a selected group of Ada programs; the result of that study is shown in Fig. 1. Currently, a real-time programmer can expect to achieve at least a 2:1 efficiency using the NEBULA architecture, and a ratio of Ada source code to Nebula code that is approximately 1.4. The feature of a variety of contexts within each Nebula instruction enhances the use of pipelining and cache memory applications within the MCF, which results in additional processing speed.

Program	Nebula	IBM/370	PDP-11	1750A	UYK-43	VAX
VIP	20	64	52	56	60	
DIFF	20	52	56	44	44	18
COS	44	128	66	86	104	63
BIN_SH	49	118	92	80	112	63
INSRT	45	70				
HP_SRT	72	92				
TR_SH	65					
IN_ORD	25					
FFT	137	334	264			
XPO	55	126	82	108	136	
Subr.	$8+2n$	$36+4n$	$26+2n$	$26+4n$	$28+12n$	

Figure 1. Comparison of Nebula Object Code with Other Selected Instruction Set Architectures (in Bytes).

Finally, Nebula has great capability to support higher order languages, especially Ada. For instance, Nebula directly supports the FOR statement without the dedicated use of registers. The three-address arithmetic capability of Nebula incorporates, in a single instruction, the most common arithmetic forms of higher level language programs. As an illustration, the Ada construct "C: = AB" will result in the Nebula instruction "ADD A, B, C."

Besides being compatible with Nebula, Ada has several features that enable it to enhance efficiency and lower cost of realtime embedded systems software. For example, Ada will permit separate compilation³, as shown in Figure 2. Because specifications and bodies of procedures can be compiled separately, this capability supports top-down design. It also supports configuration management of Ada programs.

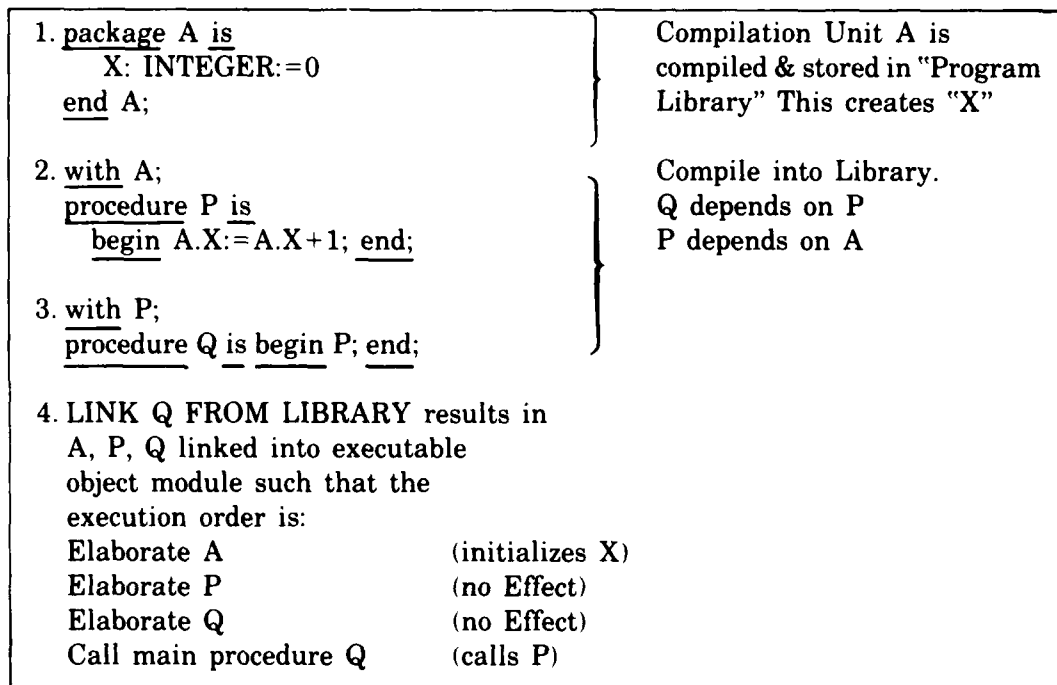


Figure 2. Compiling, Linking, and Running an Ada Program

Ada is a strongly typed language, that is, the type of data determines the type of operations permitted. A type is characterized by a set of values and a set of operations, resulting in more readable, hence more maintainable, programs and increased data security. For instance, if APPLES are defined as a type, i.e.:

```
type APPLES is range 0..10;
and ORANGES is also defined as type, then:
APPLES + ORANGES cannot be converted.
```

Instead, the programmer must make a deliberate type definition (such as: FRUIT (Apples) +FRUIT(Oranges) in order to achieve a conversion.

The Ada Package feature allows the programmer to logically group related items together, permitting the definition of new types and their operations without the details of implementation (data abstraction) being a concern of the programmer. This feature also permits variables, constants, and types needed by other program units (Name-Space Management) to be collected, resulting in programs that are easier to read and maintain.

The Ada also has low-level facilities support features, one of which is a predefined library package, called SYSTEM, that contains implementation-dependent specifications. Another feature, possibly the most important low-level facility, involves record representation, permitting the programmer to define data fields down to the bit level, if necessary. In control systems the requirement is often to have multiple control fields in a record. This usually ended up requiring some embedded assembly code simply for the packing, unpacking, etc. of those fields. With Ada directly supporting those functions, programs can be written completely in a higher order language.

SUMMARY

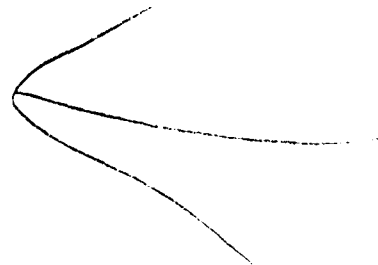
The Military Computer Family will provide computer power, in terms of speed and memory addressing, in packages satisfying almost all, if not all, embedded applications requirements. Further, the Nebula ISA will provide the real-time programmer with advantages in terms of execution speed and space. Finally, Ada, which has a high degree of compatibility with the Nebula ISA, permits top-down construction of modular programs that are highly readable and maintainable.

The result will be a new generation of computer programs for embedded real-time operations that will have greater capability and improved reliability and maintainability.

REFERENCES

1. Druffel, COL Larry E., et. al., Final Report of the Software Acquisition and Development Working Group, Prepared for the Assistant Secretary of Defense for Communications Command, Control, and Intelligence, July, 1980.
2. Anderson, Peter G., A Comparison of Nebula and Alternative Architectures via Selected Ada Programs, Rochester Institute of Technology, 1981.
3. Fisher, Gerald A. and Brosgol, M., Ada TEC Tutorial on Ada, presented at the Ada TEC Convention, October 4-5, 1982, at Arlington, Va.

Fred E. Wuebker is currently Manager, Military Computer Family, Software and Technical Director of MCF Operating System Project, at RCA, Moorestown. He is directing software development of the Military Computer Family, and was responsible for the radar control program development, interface, test, and integration system engineering for the AEGIS Combat System. He was a member of the RCA Definition Team for the Ada design language effort.



AD-P003 535

PAS/NEB: A PASCAL LANGUAGE TOOL FOR THE MCF

Elizabeth Souren
Guido Lonardo
Dr. Robert Couranz

Equipment Division
Raytheon Company
528 Boston Post Road
Sudbury, MA 01776
(617) 443-9521

ABSTRACT

> The ability to generate structured High Order Language (HOL) code for modern computers is an almost universally accepted requirement. Raytheon, one of the three Army Military Computer Family (MCF) contractors, has responded to this requirement by developing tools allowing the generation of code for MCF in PASCAL. The major features of PASCAL are available to the user including the intermixing of PASCAL and NEBULA modules.

This paper describes the general structure of the PASCAL tool and provides examples of how PASCAL has been effectively employed on the Computer Control Panel of the Raytheon MCF Advanced Development Model. Measures of MCF instruction set code generation are presented. Experience to date indicates that PAS/NEB provides an effective HOL tool for programmers while awaiting the availability of MCF ADA.

INTRODUCTION

One of the major objectives for the MCF is to provide a standard processor family that will improve major system survivability in the field. With standardized processor hardware, failed or battle damaged units in high priority systems can be replaced, if necessary, with operable units removed from lower priority systems. Such trading of components may extend to the circuit card assembly level.

The Computer Control Panel (CCP) is the component of the MCF providing field maintenance personnel with the tool set needed for rapid processor maintenance and validation. Because speed in repair is critical, a high probability of fault isolation to a single module is required. And because maintenance activity under battlefield conditions will invite errors in fault diagnosis, an intelligent, operator friendly interface is required. As a result, the MCF program includes a significant diagnostic and operator interface requirement for the CCP. Although ADA was not yet available, code generation in an HOL was desired.

Along with the obvious advantages a HOL offers, such as ease of coding and debugging and well-structured output, the addition of such features as recursion, records, and types would speed the software development effort. The use of a more efficient implementation tool would provide the opportunity to add some features such as an English-like command language for the user interface and produce a more sophisticated diagnostic executive.

In parallel with the design effort, evaluation of the Army-supplied simulator led us to decide to enhance this simulator's user interface in order to make it user friendly. Using the same software for both user interfaces (CCP and simulator) would allow us to spend more time developing a single sophisticated package instead of splitting the available resources over two similar efforts.

This led to a search for a HOL which could be interfaced with the existing simulator code and could also be easily compiled into NEBULA. As part of a continuing effort to improve our software and microcode development tools, techniques were being evaluated to transport software developed on one of Raytheon's Software Development Facilities (SDF) to Raytheon developed hardware. The SDF host processor in this case is the Digital Equipment Corporation's VAX* 11/780. Two important facts were uncovered:

- 1) The DEC* PASCAL compiler for the VAX will, as an option, output an assembly level listing.
- 2) Raytheon has a significant base of skilled PASCAL programmers.

We decided to design a "cross-assembler" which would translate the output of the PASCAL compiler into NEBULA. This would, within a reasonable time frame, provide a software tool giving a high payback on the actual MCF program.

Figure 1 depicts the general software development flow available as a result of utilizing PASCAL and a common user interface for assembly level software development. The programmer codes and checks out his program at the high order language level utilizing the host's (VAX) software development tools. The debugged PASCAL is then translated to assembly-level error-free NEBULA code which is assembled and linked with the GFE tools. The Army-supplied simulator, enhanced by Raytheon to provide a similar interface to the user as would be encountered on the CCP, is then used to check out the translated program. After simulator checkout is completed, the translated program is transported to the MCF equipment and verified on the real hardware. This last step is facilitated by the fact that the CCP user interface is a subset of the Raytheon simulator user interface.

DESIGNING PAS/NEB

If requested at compilation, the VAX PASCAL compiler will give the user an assembly listing of the machine code it generated. This list file is used as input by PAS/NEB to accomplish the translation. Translation is a one pass operation, each line being read and immediately translated. Figure 2 is a high level flow chart showing the basic structure of PAS/NEB. A line is

*DEC and VAX are trademarks of Digital Equipment Corporation

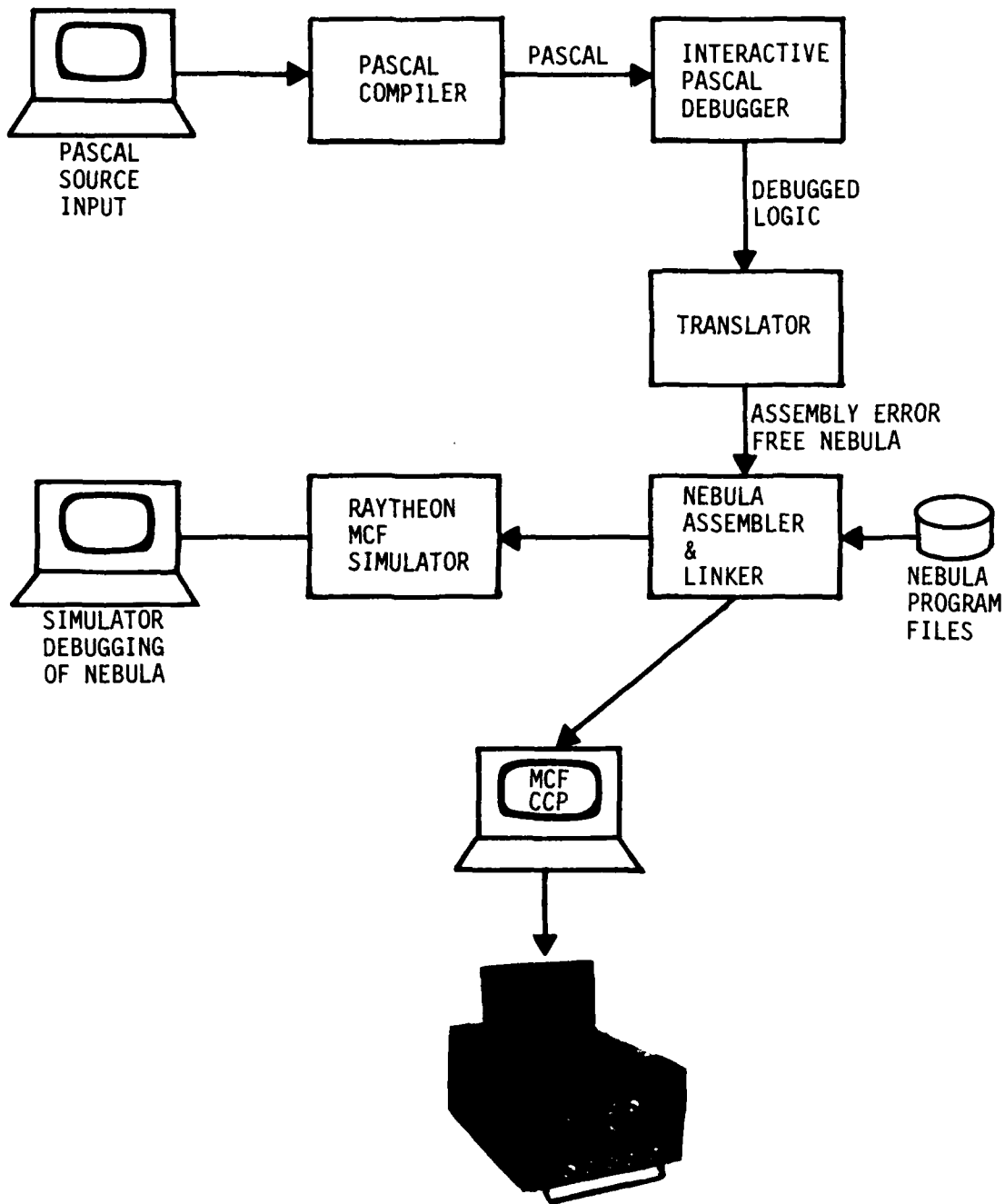


Figure 1 Software Development Flow with PAS/NEB

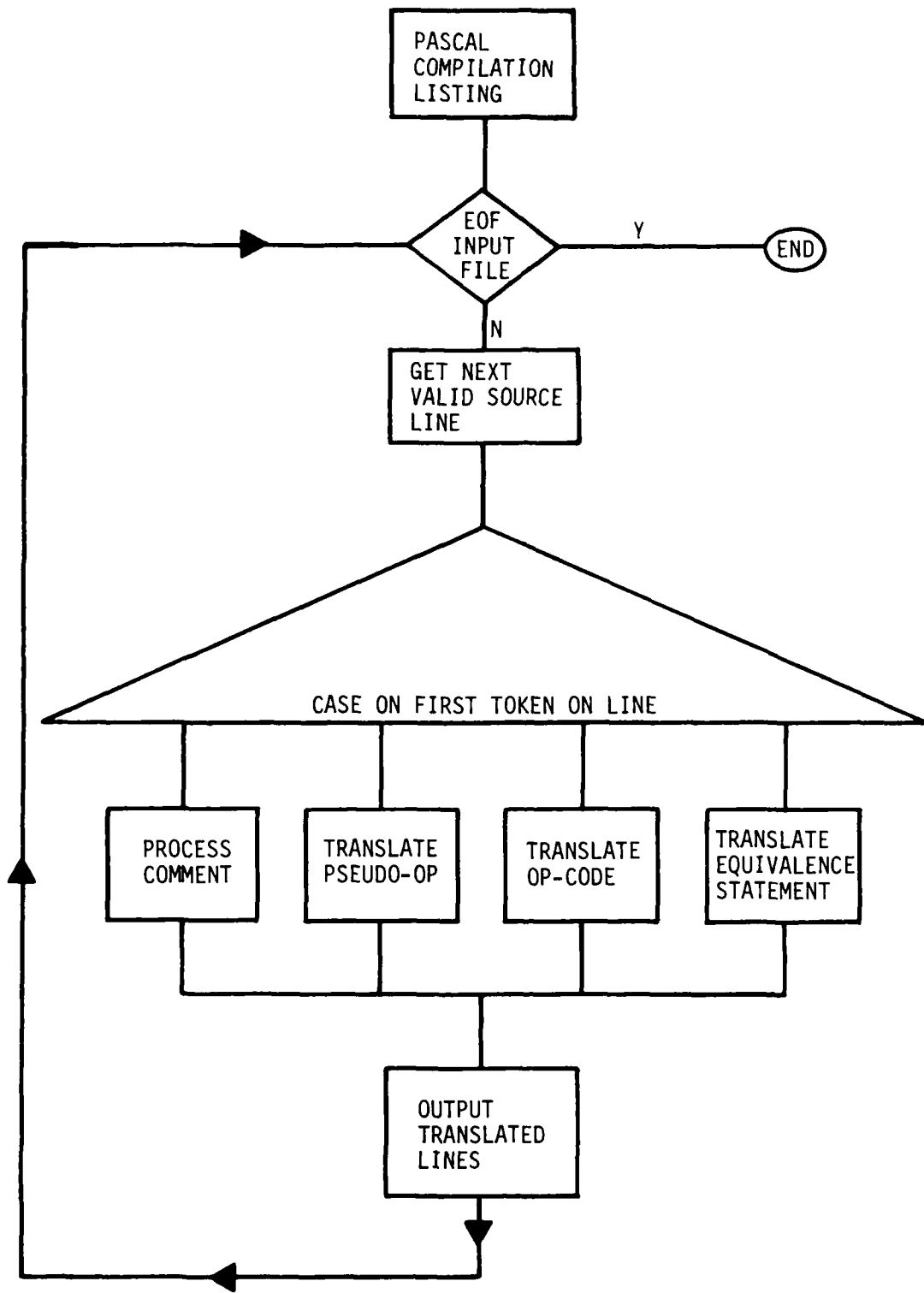


Figure 2 High Level Flow Chart of PAS/NEB

examined to determine if it contains VAX machine language. Page headings, summary lines, comments and blank lines are discarded. PASCAL source lines are saved in a temporary file to be retrieved if the user requests that PAS/NEB output contain PASCAL source lines interleaved with NEBULA as comments explaining the machine code.

PAS/NEB is table-driven. Opcodes are located in an internally-stored table which gives either the equivalent NEBULA opcode or a pointer. A series of unique subroutines in PAS/NEB handle those opcodes which do not translate in a 1-to-1 fashion. Approximately 70% of all opcodes in the table can be directly translated without any special handling. This assumes that every opcode in the table is encountered with equal frequency. Actually, a small set of opcodes, such as MOV and CMP, account for the majority of the code generated by the PASCAL compiler. Some of the opcodes, such as XOR or ACB (add, compare and branch), which PAS/NEB is capable of translating are never used by the PASCAL compiler. Others may be generated, but only if the programmer needs an infrequently-used feature of PASCAL, such as double-precision arithmetic.

A module from the MCF diagnostic software, about 1000 PASCAL lines, was run through PAS/NEB, resulting in 1782 VAX assembly language instructions. This translated to 2111 NEBULA instructions, or 84% of all opcodes being translated 1-to-1. It should be noted here that the 1782 to 2111 ratio does not indicate that NEBULA is a less efficient language. Each computer has some instructions which do not have an equivalent in the other and must be translated by generating a series of instructions. Analyses conducted by IBM¹ and EG&G², among others, have shown that the NEBULA instruction set provides highly efficient code, both in terms of static program size and memory accesses.

Writing PAS/NEB was rather like trying to translate from French to English by looking each word up in a dictionary. You can translate each word correctly, but the overall product will not necessarily make sense. Each language has idioms which cannot be translated literally and experience here shows that computers are the same. The problems encountered when designing PAS/NEB range from very simple and easy to solve to reasonable direct solutions requiring precise bookkeeping and determination. Finally, there is the class of problem that requires experience and technical judgement for a viable solution. Some problems in the former category include:

- 1) The program counter and stack pointer on the VAX are registers 15 and 14. They are registers 0 (zero) and 1 in NEBULA.
- 2) The VAX numbers its bits with 0 (zero) being the least significant bit in a word. 0 is the most significant bit in a NEBULA word. This causes complications when translating bit field instructions. The VAX compiler, for example, uses bit field instructions to move a byte from the middle of a word.
- 3) The VAX CMP instruction does both a signed and an unsigned compare, setting different condition codes for each. It has signed and unsigned branch instructions which look at the appropriate condition codes. NEBULA has separate signed and unsigned compare instructions, and only one set of branch instructions.

- 4) The VAX CALLS instruction automatically signals to the computer to save the values stored in all registers on the stack. The registers retain their values so that the procedure being called may make use of them, which the PASCAL compiler does. The NEBULA CALL instruction also saves the value of all registers, but does not necessarily retain the original contents of the registers while the called subroutine is being executed.

There were also a number of more complex problems caused by the fact that the machine language source generated by the PASCAL compiler is not exactly correct. One of the most difficult problems to be handled in the original design of PAS/NEB was that of variable names. The VAX PASCAL compiler handles all variables as offsets from registers. Each variable name is equivalenced to a constant representing that variable's offset from the proper register. All global variables are defined as offsets from register 11, all call-by-name parameters as offsets from register 3, etc. The problem occurs when a program contains a local and a global variable with the same name. Figure 3 shows a sample PASCAL program. The function returns the factorial of the number

```
PROGRAM EXAMPLE;  
  VAR  
    OPERND: INTEGER;  
  FUNCTION FACTORIAL ( OPERND: INTEGER): INTEGER;  
  BEGIN  
    IF OPERND > 1 THEN  
      FACTORIAL := OPERND * FACTORIAL(OPERND-1)  
    ELSE  
      FACTORIAL := 1  
    END;  
  BEGIN  
    OPERND := FACTORIAL(10)  
  END.
```

Figure 3 Sample PASCAL Program

passed. It computes the factorial recursively*. The variable OPERND is declared as a global variable in the main block, but is also the name of the parameter in the function FACTORIAL. The assembler listing output from the compiler is shown in Figure 4. The symbol OPERND is defined twice; once for the global variable and then for the parameter. The function will assemble correctly since the OPERND accessed is the parameter OPERND, which was defined in the last equivalence statement. However, the main block references the global variable OPERND. When assembled, the offset for the parameter OPERND will be used. To correct this problem, PAS/NEB builds and uses a symbol table, keeping track of all the symbols defined in each block currently in effect, along with the block name associated with each symbol. Every time a symbol is

*It should be noted that recursion is not the most efficient way of solving this problem. In fact, for large numbers, this function would overflow the stack. The recursive approach was used to demonstrate how the PASCAL compiler and, consequently, PAS/NEB handle a problem of this sort.

```

; SYMBOLS FOR EXAMPLE
OPERAND = 8
; SYMBOLS FOR FACTORIAL
OPERND = -24
FACTORIAL = -20
    .PSECT $CODE, PIC, REL, SHR, EXE, RD, NOWRT, 2
    .entry FACTORIAL, M<R9, R10, R11, IV>
MOVAB $GLBL, R11
MOVL $20, R10
PUSHL R1
PUSHL AP
SUBL2 R10, SP
MOVL @4(AP), OPERND(FP)
MOVL SP, -12(FP)
Cmpl OPERND(FP), $1
BLEQ 1$
SUBL3 $1, OPERND(FP), R10
MOVAB -28(FP), R9
MOVL R10, (R9)
PUSHL R9
CALLS $1, FACTORIAL
MOVL R0, R10
MULL2 OPERND(FP), R10
MOVL R10, FACTORIAL(FP)
BRB 2$
1$:
MOVL $1, R10
MOVL R10, FACTORIAL(FP)
2$:
MOVL FACTORIAL(FP), R0
RET
    .PSECT $CODE, PIC, REL, SHR, EXE, RD, NOWRT, 2
    .ENTRY EXAMPLE, †M<R10, R11, IV>
MOVAB $GLBL, R11
MOVL SP, -12(FP)
MOVZBL #10, 12(R11)
PUSHAB 12(R11)
CALLS #1, FACTORIAL
MOVL R0, R10
MOVL R10, OPERND(R11)
RET
; R11-BASED PROGRAM LEVEL STATIC STORAGE
.END

```

Figure 4 PASCAL ASSEMBLY LISTING


```

; SYMBOLS FOR EXAMPLE
; PROGRAM EXAMPLE;
; VAR
;   OPERND: INTEGER;
OPERND = 8
; SYMBOLS FOR FACTORIAL
; FUNCTION FACTORIAL ( OPERND: INTEGER): INTEGER;
FACTORIAL_OPERND = -24
FACTORIAL_FACTORIAL = -20
  .SECT $CODE, CODE
  .GLOBAL FACTORIAL
FACTORIAL:
  JSR   SAVE_REGS
  MOVA  $GLBL+B, R15+W
  MOV   #20+W, R14+W
  PUSH  R5+W
  PUSH  AP+W
  SUB   R14+W, SP+W
  MOV   @(0)(4(AP))+W, FACTORIAL_OPERND(FP)+W
  MOV   SP+W, -12(FP)+W
; BEGIN
;   IF OPERND > 1 THEN
  CMP   FACTORIAL_OPERND(FP)+W, #1+W
  BLEQ  FACTORIAL_1$+B
;     FACTORIAL := OPERND * FACTORIAL(OPERND-1)
  SUB   #1+W, FACTORIAL_OPERND(FP)+W, R14+W
  MOVA  -28(FP)+B, R13+W
  MOV   R14+W, @R13+W
  PUSH  R13+W
  PUSH  #1
;   ELSE
  JSR   FACTORIAL
  MOV   R4+W, R14+W
  MUL   FACTORIAL_OPERND(FP)+W, R14+W
  MOV   R14+W, FACTORIAL_FACTORIAL(FP)+W
  BR    FACTORIAL_2$+B
FACTORIAL_1$:
;   FACTORIAL := 1
; END;
  MOV   #1+W, R14+W
  MOV   R14+W, FACTORIAL_FACTORIAL(FP)+W
FACTORIAL_2$:
  MOV   FACTORIAL_FACTORIAL(FP)+W, R4+W
  JUMP  RESTORE_REGS
  .SECT $CODE, CODE
  .GLOBAL EXAMPLE
EXAMPLE:
  MOVA  $GLBL+B, R15+W
  MOV   SP+W, -12(FP)+W
; BEGIN
;   OPERND := FACTORIAL(10)
  MOVL  #10+B, 12(R15)+W
  MOVA  12(R15)+B, R4+W
  PUSH  R4+W
  PUSH  #1
; END.
  JSR   FACTORIAL
  MOV   R4+W, R14+W
  MOV   R14+W, OPERND(R15)+W
  RSR
  .END

```

Figure 5 Results After Running PAS/NEB

encountered, the symbol table is searched, and the associated block name is prefixed to the symbol name. Figure 5 shows the example program after being run through PAS/NEB. Here the parameter OPERND has been changed to be FACTORIAL OPERND, eliminating the ambiguity.

USING PAS/NEB

As previously noted in Figure 1, PAS/NEB is used in much the same way as the standard PASCAL compiler would be used. Programs and modules are written in PASCAL and tested. Depending on the nature of the software, a program can be tested fully on the VAX, making use of the VAX's debugger and any other programming aids desired, before it is translated to NEBULA. After all logical and PASCAL coding errors have been cleared, a final pass through the compiler is accomplished with the /MACHINE option invoked. The assembly level code output is then used as input to the PAS/NEB translator.

In the course of designing PAS/NEB, a number of features of PASCAL were discovered to be untranslatable. By untranslatable, it is meant that the use of these features generates assembler sequences that are either impractical or impossible to translate to NEBULA. For example, the use of the VALUE statement in PASCAL, which is used to assign an initial value to variables, does not have any effect on the assembly listing. If programmers were to use this statement, the variables involved would not be initialized in NEBULA. Other restrictions which had to be imposed include:

- 1) The WITH statement cannot be used. Fortunately, WITH is an infrequently used statement as demonstrated in Reference 3.
- 2) NEBULA instruction names cannot be used as global variable names. For example, TEST is a NEBULA mnemonic and so could not be used as a global variable name. It could, however, be used as a local name in a procedure.
- 3) Labels can be used, but all labels declared within one compileable module must be unique.
- 4) The /CHECK option on the compiler, which instructs the compiler to generate run-time checks on all array accesses to be sure that the subscript is within bounds, cannot be used during the /MACHINE option pass.

Perhaps one of the most amusing problem encountered so far with PAS/NEB is one we have dubbed "the two smart alecks." The PASCAL line reading

```
I := 255
```

is translated by the PASCAL compiler to the following sequence:

```
MOVZBL #-1, R10  
MOVL R10, I(R11)
```

The MOVZBL instruction means move zero-extended byte to longword. The PASCAL compiler designer has decided to be very tricky and has figured that, for a single byte, a negative 1 looks the same as integer 255. After being processed by PAS/NEB, the generated sequence is as follows:

```
MOVL    #-1 B,R14 W
MOV     R14 W,I(R15) W
```

For those familiar with NEBULA, this sequence would appear to be the same. However, the NEBULA assembler sees the first instruction and decides that, even though the user specified that the first operand was to be a byte, if the user wants a negative one, it'll give him a negative one. It therefore overrides the byte specification and moves an entire word into R14, changing I := 255 to I := -1. Various maneuvers were attempted to "trick" either the PASCAL compiler or the NEBULA assembler into doing what they were told to do. The only solution found was to require that users declare all constants and use the symbolic name instead of inserting constants directly into the code.

PAS/NEB is only as efficient as the PASCAL compiler itself. On the average, a long PASCAL program will translate to NEBULA with about a 1-to-2 ratio of PASCAL lines to NEBULA instructions. If program size or speed become a critical issue, there are some very simple optimizations on the NEBULA output from PAS/NEB that would significantly reduce the size of the generated code. Similar actions could be applied to VAX code generated by the PASCAL compiler. The PASCAL compiler is a one-pass compiler, which makes optimization rather difficult. In an effort to optimize the use of memory accesses as opposed to register accesses, it translates an assignment statement into a MOV to a register and then from the register into the desired memory location. If the memory location is accessed again before it is modified, the register may be used in place of another memory access. However, in many cases, the value in the register is never accessed again. A very simple optimization of the code generated would be for the programmer to scan the compiler generated code and remove the extra MOV statement in places, such as initialization routines, where the value in the register is not accessed again.

EXPERIENCE WITH PAS/NEB

PAS/NEB has been used very effectively on the Raytheon MCF program. It has been used both to reduced the time and the cost to develop NEBULA programs and to capture large amounts of existing software. Over 8000 lines of PASCAL are used in the various software packages that comprise the Raytheon Computer Control Panel. Figure 6 depicts the major functional elements of the CCP software package. The modules written in PASCAL are indicated by the bold outlining. Of these, 4000 lines were developed specifically for the CCP. The other 4000 were captured from the simulator enhancement effort, and other internal projects. The CCP specific software consists of three types:

- 1) A menu-driven user interface
- 2) Resource management functions such as display support routines, dynamic buffer handler, and file manager
- 3) Unique functions such as machine state logging.

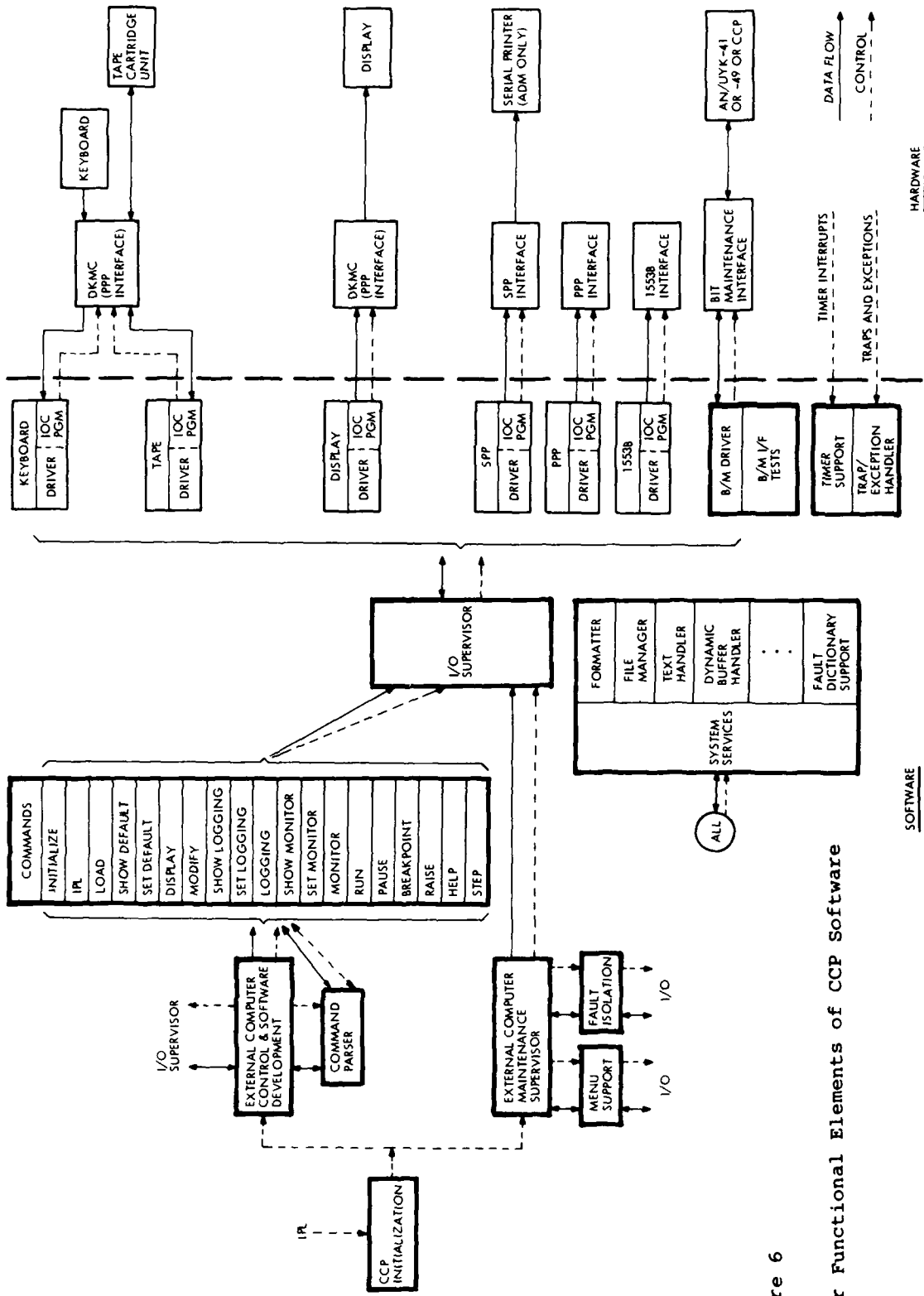


Figure 6

Major Functional Elements of CCP Software

The package that was captured from the simulator enhancement effort consisted of the entire software development mode user interface. This included the syntax parser, extensive format conversion routines, and all command executors; i.e., MODIFY, SET BREAKPOINT, IPL, LOAD, etc. The remaining captured software was the majority of a generalized diagnostic executive which is resident in the Computer Under Test.

Time was saved in developing these packages since they were designed, coded and verified on the VAX using all the resources of the VAX. Test programs were also written in PASCAL. Cost savings were realized from many aspects. Obviously, the shorter the development time the less the cost. But this was not the only savings. Additional savings resulted from the reduced use of the NEBULA assembler, since PAS/NEB generates assembly-error-free NEBULA source, and reduced simulation time since the bulk of the software was thoroughly checked in PASCAL.

As a specific example, the External Computer Maintenance Supervisor is a menu-driven program that allows the user to set up and run the MCF diagnostics. It generates menu displays, prompts the user for input, highlights selected fields in reverse video, and eventually, after setting up the diagnostic database, passes control to the computer under test. This program, totaling approximately 2900 PASCAL lines (not including menu text), was designed, coded and checked out on the VAX before I/O capability had been added to the simulator. This effort involved extensive review of menu format and content and operator ease-of-use, and took approximately 10 weeks. It was then run through PAS/NEB and, without any modifications except the addition of a NEBULA-specific module which handles input, ran successfully on the NEBULA simulator. Since neither the hardware nor the simulator could support I/O at the time the task was started, not only would the coding and checkout have taken longer (approximately 50 man-weeks for roughly 6500 lines of NEBULA, assuming an average throughput of 34 lines of NEBULA per day to design, code, test and document) but it would have been impossible to exercise the menu-handler in an interactive way until the hardware was available.

SUMMARY

PAS/NEB was developed to be an interim software tool for the Advanced, and eventually, the Full Scale Development phases of the MCF program. Our experience to date with code written for the CCP and other elements of the MCF indicates that the effort expended in producing PAS/NEB has been repaid many times in the ease of coding, checkout, and documentation. The few restrictions imposed on the PASCAL source have not had a noticeable effect on coding technique.

In certain functions such as real time interfaces, some manual optimization or linking to NEBULA code modules will be necessary. All functional elements necessary to accomplish are available as previously described.

The PASCAL debugging tools available on the commercial processor have been extremely valuable in developing the error-free code input to PAS/NEB. These tools also provide for easy modification of existing PASCAL code for MCF application.

It is anticipated that PAS/NEB will continue to be used for the development of MCF demonstration, test and evaluation programs.

BIBLIOGRAPHY

1. "The Army's MIL-STD-1862 and the Military Computer Family", Kogge, P. and Olsen, P., IBM Technical Directions, Vol 7, No. 2, Summer 1981, pp. 29-41.
2. "Comparative Evaluation of the NEBULA Architecture," TR-MCF-007, EG&G Washington Analytical Sources Center, 4 March 1981.
3. "A VLSI RISC," Patterson, David A. and Sequin, Carlo, Computer, September 1982.
4. MIL-STD-1862A Military Standard Nebula Instruction Set Architecture.
5. "VAX11 Architecture Handbook", Digital Equipment Corporation.

Elizabeth Souren received her BS degree in Mathematics from Carnegie-Mellon University. After joining Raytheon in 1979, Ms. Souren was involved in the development of CAD tools. As a member of the Raytheon MCF engineering staff, she has been the principal designer on PAS/NEB and developed user interface software for the MCF.

Guido Lonardo received his BS degree in Computer Science from the City College of New York in 1973 and an MAS degree in Computer Science from Boston University in 1978. Mr. Lonardo joined Raytheon in 1973 and since that time has specialized in diagnostic and systems software. He is presently Software Technical Director for the Raytheon MCF. Prior to his present assignment, Mr. Lonardo was software manager for the Fault-Tolerant Spaceborne Computer program.

Dr. Robert Couranz received his BS degree from Washington University in 1960, his MS in Electrical Engineering from the Air Force Institute of Technology in 1962, and his D.Sc. from Washington University in 1970. Dr. Couranz served as an officer in the United States Air Force from 1960 through 1966. Before joining Raytheon in 1970, he was with the Washington University Computer Systems Laboratory and the Foreign Technology Division, AFSC. Dr. Couranz has held both technical and managerial positions within Raytheon including Manager, Computer and Displays Laboratory. He recently returned to Raytheon as a Technical Advisor after two years as Vice President, Engineering, Pencept, Inc. Dr. Couranz is a member of IEEE, Tau Beta Pi, and Sigma Xi.

MIL-STD-1553

MULTIPLEX DATA BUS

SESSION CHAIRMAN: Donald H. Ellis
President, Aero Systems Associates, Inc.

MODERATOR: Donald H. Ellis
President, Aero Systems Associates, Inc.

A PROGRAMMABLE BUS CONTROL INTERFACE

Robert M. Salter

Sperry Univac
P.O. Box 3525
M.S. U2S25
St. Paul, MN 55165

ABSTRACT

Many aspects of MIL-STD-1553B design have become "routine", standard transceivers, encoder-decoders and nearly complete interfaces can be purchased. Bus controller and "smart" RT performance, however, is still evolving as capability beyond mere protocol handling is possible in an I/O module. The Air Force Programmable Interface for Multiplex Systems project created a 1553 embeddable processor on a 6"x9" module with sufficient power to support multiple dialects of MIL-STD-1553, operate as an independent processor executing an extensive general purpose instruction set. The conventions established for host-to-channel interface allow the channel to both efficiently execute bus lists and off load exception processing from the host.

I. INTRODUCTION

In the MIL-STD-1553 world recent emphasis has been on Remote Terminals (RT's), their size and cost reduction, and standard chip sets. In fact, MIL-STD-1553B is an RT standard, leaving a few fairly straightforward issues to be resolved by the designer.

Processors hosting 1553 channels can be built to an established standard, MIL-STD-1750A. However, the External I/O instructions in MIL-STD-1750A are not defined, leaving a major function of an avionics bus system without a standard for design (see Figure 1). This is the bus control function, whose design determines bus efficiency.

One reason for this is that bus controllers are still evolving from virtually hard-wired controllers, through simple microcoded designs, to potentially highly capable I/O processors on a module. A contribution to this evolution is PIMS.

The Programmable Interface for Multiplex Systems (PIMS) is an Air Force program with Sperry Univac and TRW which originally investigated the hardware requirements for a multi-dialect RT (compatible with various versions of MIL-STD-1553). It soon developed into a general channel definition task in which it was found that in about 50 square inches and 20 watts a respectable processor plus a 1553 front end, plus a host interface could be fit. Because of the processing potential, the channel could behave quite differently than older types. PIMS then explored what such a channel could do, and this paper gives a high level view of

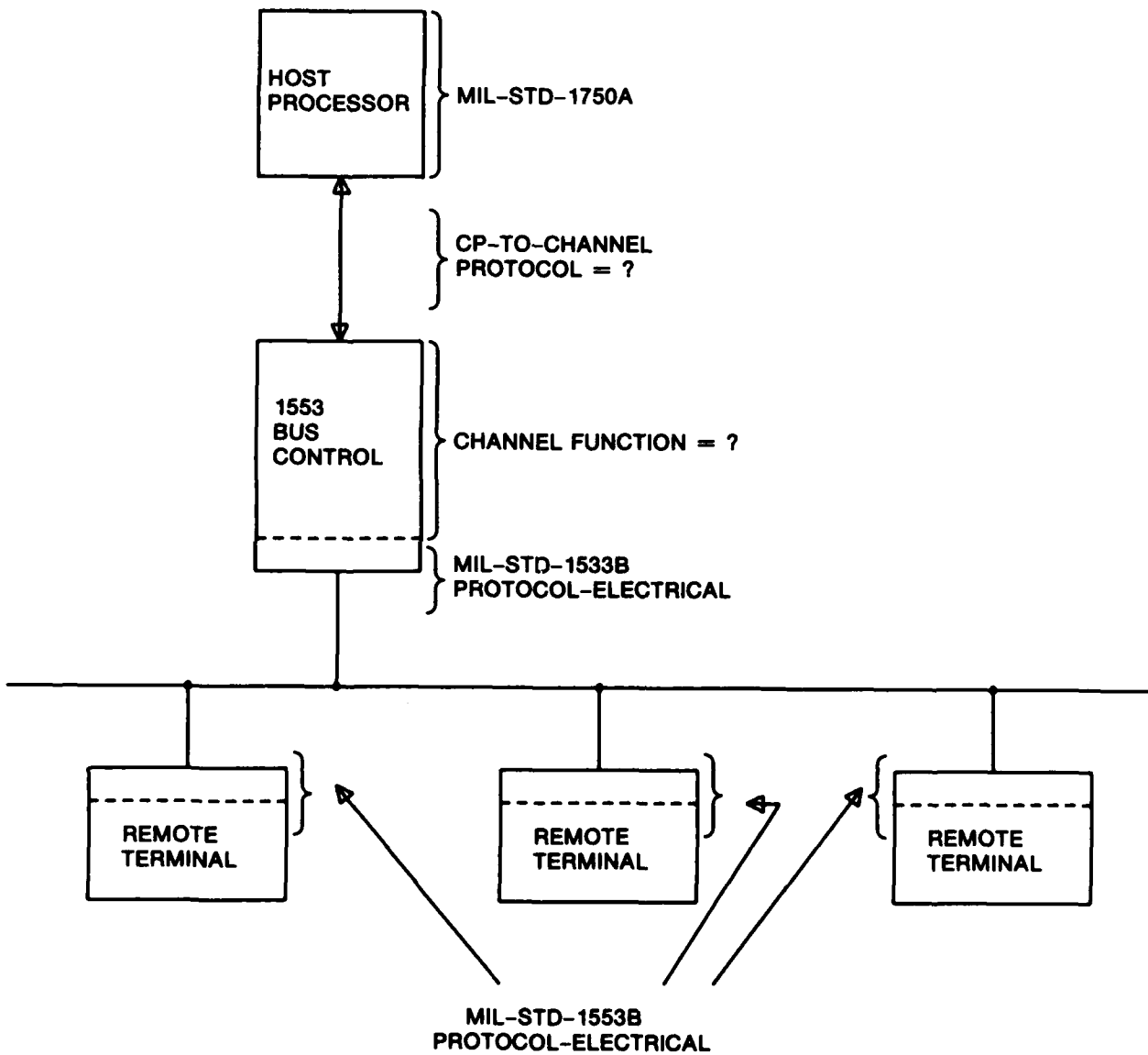


FIGURE 1. AN AVIONICS BUS SYSTEM

the results. First, here is a look at how controllers differ and how they have evolved.

HOW CONTROLLERS CAN DIFFER

All MIL-STD-1553 systems have one bus controller which directs all traffic and up to 32 remote terminals. Since all bus controllers must transmit and receive words on the bus according to the protocol in the standard, they are in that sense identical. Channels differ in where the processing intelligence resides and in related overhead on the bus.

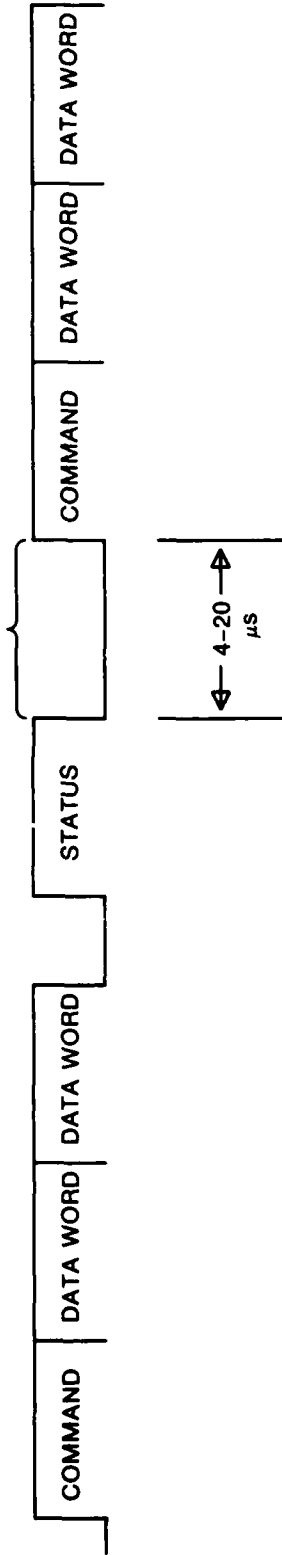
Figure 2 shows bus transactions in which the bus overhead associated with the controller, the intermessage gap, is identified. The "exception" referred to is either an error (parity, no response) or a flag set in the RT's Status Word. Note that with an exception-free transmission, intermessage gap is shown as less than one word time. For a message with an exception, however, the time is indeterminant because some intelligence must process the exception. This time is obviously a function of the handling algorithm, but it is also a function of the speed of the processing and of the efficiency of the initiation of the exception processing and the restart of the normal bus activity. Here controllers differ greatly. This is discussed below along with comment on the intrinsic processing power that controllers have always had.

CONTROLLERS OF THE PAST

Early bus controllers (e.g in an AYK-14) were multiple module units (a module arbitrarily defined as 50 square inches and 20 watts) and faced problems enough in packing encoding/decoding logic and required protocol control in the allowed space. Error handling and decision-making functions resided in the CP or I/O processor, leaving the 1553 channel with the "simple" task of mapping supplied command words into a series of transmissions and receptions. A simplified flow chart of non-mode bus control operation is shown in Figure 3. Thinking of the channel as a processor, it executed only one (but very complex) type of instruction, the bus instruction. The bus instruction encompasses the transmission of the command word, the transmission and reception of all words in the 1553 message called for in the instruction, receives and decodes the Status Word, initiates and completes any memory transactions required for the message, and tests error conditions. I.e., it may take 700 microseconds and over 50 non-trivial steps to execute a bus instruction. The demands of the protocol required that, for even the simplest bus controller, a complex algorithm in microcode or sequential logic had to be implemented on the channel. The task even today is beyond the capability of most microprocessors, such as a Z80.

Again thinking in terms of the bus controller channel as a processor, the logical advance was to add to the bus controller's capabilities. Controllers like that in the AP 101C fetched their own bus instructions out of main memory, and executed other simple instructions such as jumps, halts, and read/write of control memory. The ability to execute out of a second code stream ("priority chain") was included in some controllers. These channels still, on any exceptional condition such as a no response or Service Request from an RT, passed control back to the CP or I/O processor. The decision making and general purpose processing was still missing on the channel.

INTERMESSAGE GAP ASSOCIATED WITH EXCEPTION-FREE MESSAGE



INTERMESSAGE GAP ASSOCIATED WITH MESSAGE WITH EXCEPTION PROCESSING REQUIRED

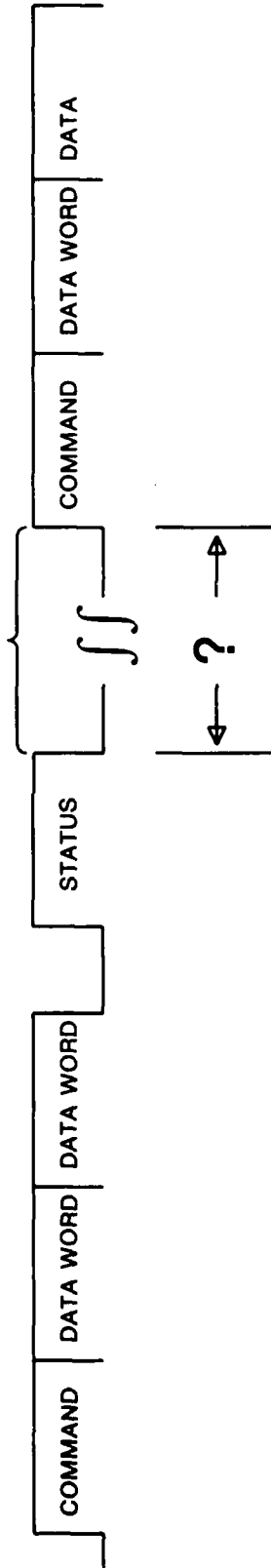
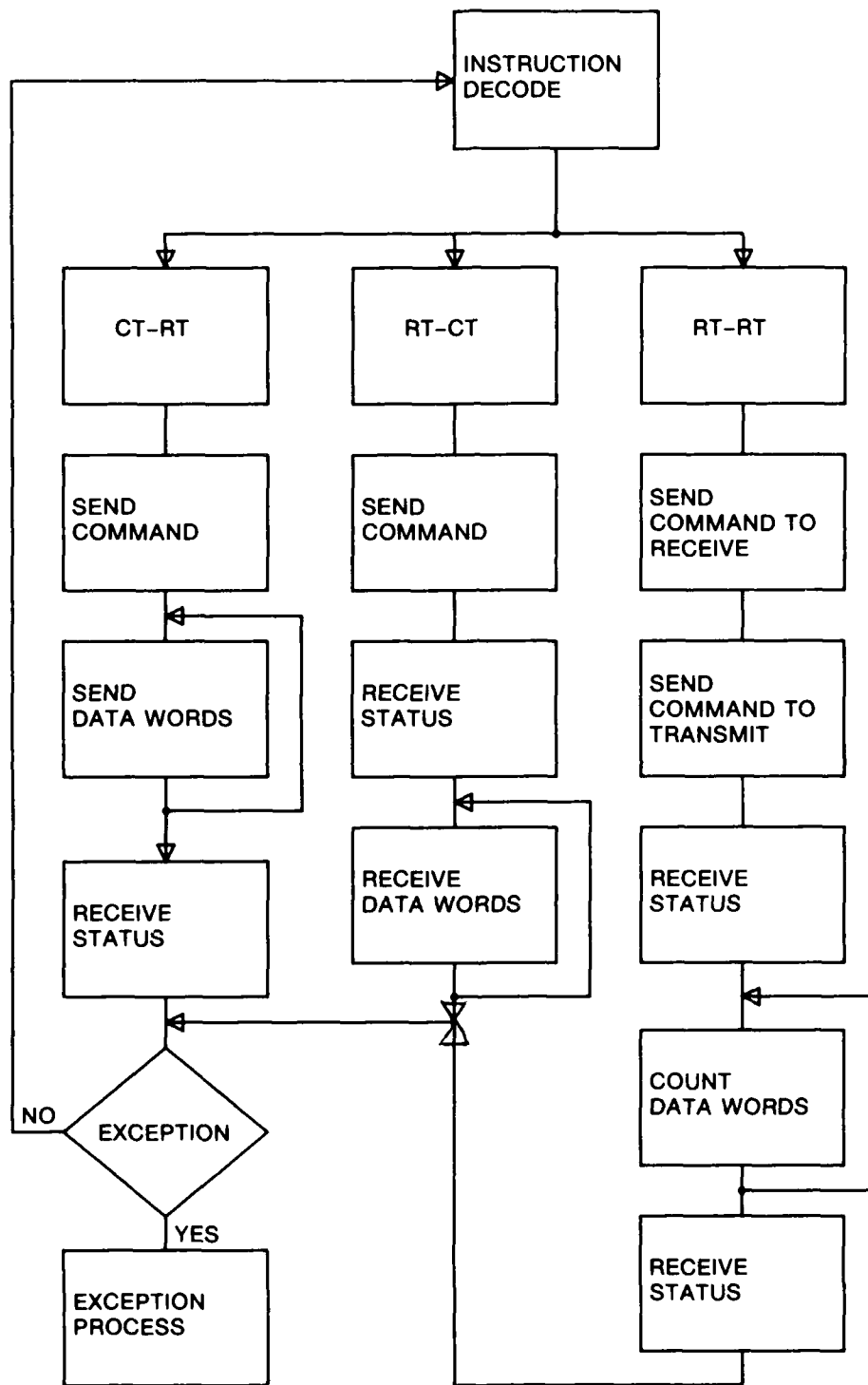


FIGURE 2. ILLUSTRATION OF BUS CONTROLLER OVERHEAD



CONTROLLERS OF THE PRESENT

A designer today can comfortably (?) put on 50 square inches and 20 watts the following:

- 256 x 16 Random Access Memory
- 16 Bit ALU
- 4K x n of Microcode and Vector PROM
- Microsequencer
- Host Interface (Processor and Memory)
- 1553 Interface

Notice the emphasis on processor capability and de-emphasis of the 1553 interface. A 1553 module today can be thought of as a processor with a port to the host computer and main memory, and a port to the 1553 bus.

Alternately it can be thought of, in light of past controllers, as a merger of the I/O processor function of, for example, an AYK-14 with the specific 1553 bus controller function, implemented on one micro-machine. Actual bus control operations become just another instruction in the repertory.

The important questions now arise: What should the channel do? What is its code flow like? What is its instruction set? How does it interact with the host? How is its power used to decrease intermessage gap? The answers the PIMS program produced are discussed in the upcoming sections.

II. HOW THE PIMS CHANNEL WORKS

The PIMS program developed a processor oriented design and it was implemented on one 6x9 inch PC card. This "processor plus host port plus 1553 port" was then microcoded with an instruction set and code structure tailored to its bus control function. This function can be broadly summarized as:

1. Provide most efficient bus use
2. Provide greatest ease of programming

PIMS worked toward these often contradictory goals in three basic ways:

1. It simplified the host computer's role in the 1553 operation
2. It set up the channel to execute autonomously out three specialized code streams
3. It implemented an instruction set with features which explicitly reduced intermessage gap.

HOST/CHANNEL INTERACTION

The general philosophy was to use the processor on the channel to do the 1553 processing that had been previously done by the CP or I/O Processor. Host Interrupts can be limited to synchronization and non-routine operations and to those calculations where the CP's faster speed is an advantage.

Except for the "Priority Chain" instruction stream discussed below, the channel allows the CP to avoid 1553 processing to whatever extent

the system designer desires.

Specifically, the two processors interact as follows:

- The host can read and write 16 channel control memory locations, plus two status words. The control memory includes configuration control, instruction counters, etc.
- The host can set and clear various channel flip flops, used to start/stop the channel operation, start Priority Chains (see below) and a few other functions.
- The channel can access the host (main) memory.
- The channel can interrupt the host either as a result of executing a macro-instruction calling for an interrupt or as the result of a channel detected transmission exception.

The minimum the host has to do is load one control memory location (an address pointer) and set a "start" flip flop on the channel. The channel begins executing its macro instructions using the provided address pointer and can complete setting itself up and proceeding with its task.

CHANNEL INSTRUCTION STREAMS (CHAINS)

Figure 4 shows the blocks of code associated with the channel, including the limited host setup code mentioned above and host interrupt handlers if any are required. The other three blocks are executed by the channel out of main memory and have three distinct roles, as described below.

NORMAL CHAIN

The pointer the host must provide the channel is the Normal Chain pointer, so that the first and primary stream of code is the Normal Chain. It tends then to have the structure shown in Figure 5, with a setup algorithm and the "bus list" or those instructions which generate the bus traffic. The subloops establish the "minor cycle" or periodicity of bus traffic.

Controllers in the past have had this bus list structure in which all the information needed to generate one message is packed in a two or three word instruction, and bus instructions are strung one after another with perhaps a jump instruction at the end to form a loop. Exceptions such as no response or Service Request caused the chain to stop and the host to be interrupted. This structure, if no exceptions occurred, had very little overhead and intermessage gap (between the reception of a Status Word and the transmission of the next Command Word) was minimal. Even in early controllers bus utilization could be quite high, as long as no exceptions occurred. It is desirable to retain this characteristic of older controllers, high efficiency with no exceptions, but add efficient exception handling to the channel's capability.

EXCEPTION CHAINS

The Exception Chain structure is somewhat analogous to the host's

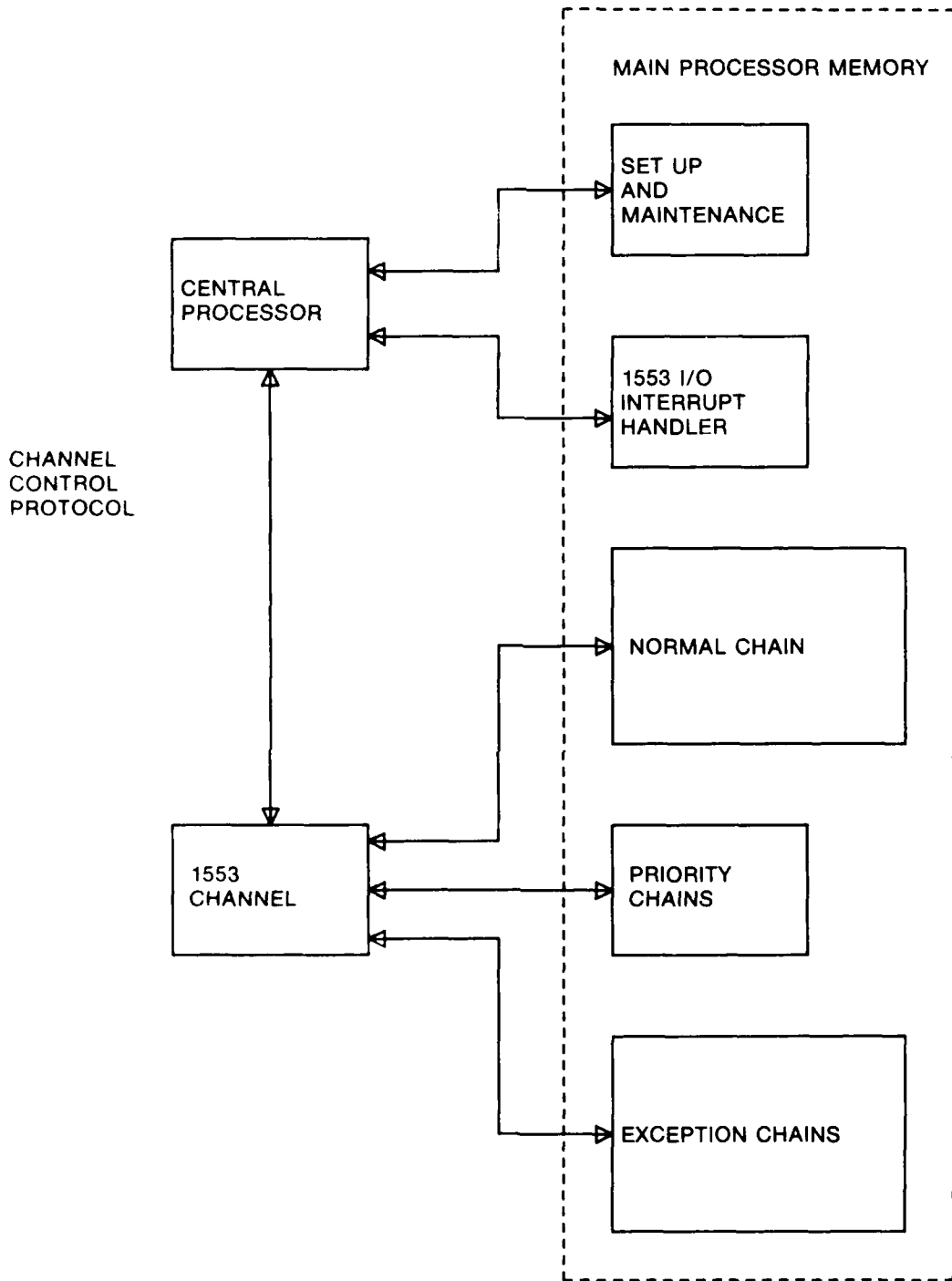
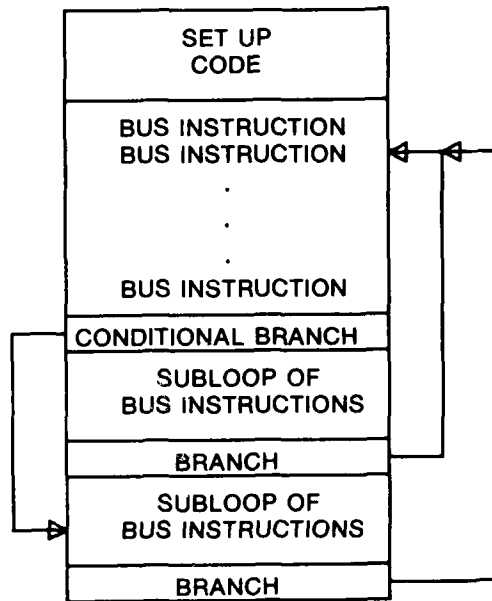
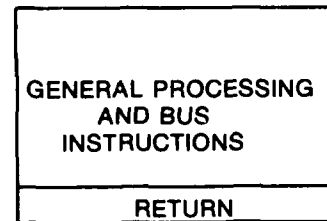
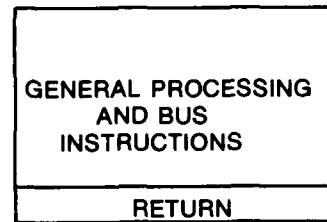


FIGURE 4. PIMS CODE STRUCTURES

GENERAL NORMAL CHAIN



GENERAL PRIORITY CHAINS



GENERAL EXCEPTION CHAIN

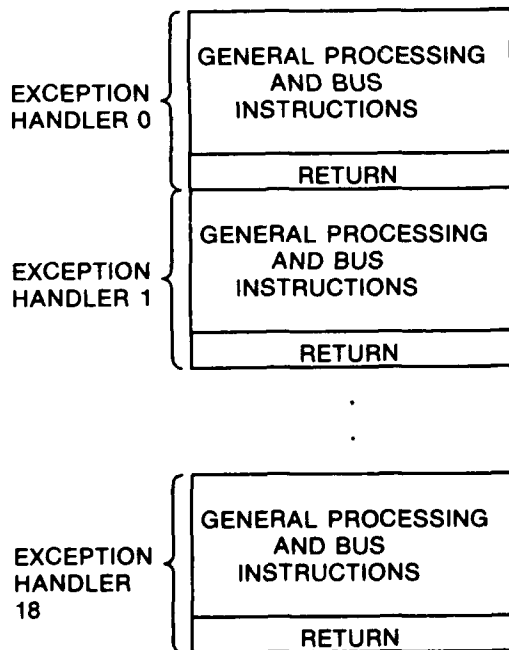


FIGURE 5. CHAIN STRUCTURES

interrupt structure. An exception such as a Service Request resulting from a bus instruction causes the channel to suspend the Normal Chain and start the Exception Chain associated with Service Request. PIMS in microcode maps exceptions into one of 18 vectors, fetches a starting address provided by the programmer and begins execution at that address. A special "Return" instruction will cause the Normal Chain to continue.

An Exception Chain can include any general instructions and any bus instructions required to handle the exception. In the handling of a Service Request the chain may fetch the RT's Vector Register via a bus instruction, index branch on the number in the vector to a specific service request handler, execute the task required and return. Or the Exception Chain may interrupt the CP to handle the Service Request.

The explicit vectoring and avoidance of interrupts allows the channel to handle exceptions efficiently and minimizes the long intermessage gaps associated with exception processing.

PRIORITY CHAINS

The Priority Chain gets the host back in the picture. If the host needs a transaction over the 1553 bus asynchronously (i.e. it is not part of the bus list loop) it can load channel control memory with a Priority Chain pointer and set a channel flip flop. The channel will complete the (Normal) Chain instruction being executed, then start executing the Priority Chain. Via a simple semaphore or an interrupt the host can tell if the Priority Chain has completed (executed a Return instruction). The Normal Chain then resumes. The Priority Chain can include any channel instruction and Exception Chains will be called as needed out of the Priority Chain.

PIMS Priority Chains are like priority chains in previous controllers in that they eliminate a tedious process of the host halting the channel, changing Normal Chain pointer, starting the channel, etc.

The broad goals of high bus utilization and ease of programming should be recalled: The Normal Chain allows a stream of exclusively bus instructions to be executed, effectively keeping intermessage overhead in the microcode domain. It also facilitates prefetch of instructions to further reduce intermessage gap. The Exception Chains are designed to simplify exception processing and as long as the exception algorithm is within some bound they are more efficient than an interrupted CP. I.e. intermessage gap on the bus due to exception handling is less. Priority Chains as suggested above are a very time efficient and code efficient way for a CP to initiate an asynchronous bus message.

CHANNEL INSTRUCTION SET

PIMS code structures have been devised for ease of programming and efficient bus usage. What of the instructions themselves? We have assumed a contemporary controller has a general purpose set plus bus instructions, at least. How many, what are they, and how fast do they run?

As an example of what can be done, PIMS implemented a full complement of fixed point arithmetic and jump instructions with several addressing modes (see Table 1).

Operations: Load
Store
Add
Subtract
XOR
AND
OR
Increment
Decrement
Test Bit
Test and Set Bit
Reset Bit
Compare
Shift Right Circular

Addressing Modes:
Immediate
Direct (Main) Memory Access
Indexed (Main) Memory Access
Channel Register

Table 1 Instruction Repertory

In addition to this general processing set, the channel executes bus instructions of two or three words for CT-RT, RT-CT, RT-RT mode and data transfers.

Other features of the instruction set explicitly simplify or speed up channel operation (see Table 2).

CHANNEL PROCESSOR PERFORMANCE

The channel processor will tend to be slower than the host for a number of reasons, including slower path to memory and less room per module for performance improving hardware. PIMS itself executes a register-to-register add in about 8 microseconds. A new design building on PIMS experience and using higher density integrated circuits could realistically achieve 5 microseconds or less for register-to-register types.

This relatively slow speed must be contrasted to the alternative: interruption of the CP. Generally, the more bus instructions embedded in exception processing the better the channel processor performs relative to an interrupted CP.

OTHER PIMS FUNCTIONS: RT MODE, MONITOR

Both a multi-dialect Remote Terminal function and a Bus Monitor feature were programmed on the PIMS module, making it a full function channel. The bus controller and instruction set microcode actually occupies only half the microcode PROM. The RT and Monitor functions are not discussed in detail here but it should be pointed out that the memory capacity and power of the channel micro-machine make their implementation straightforward.

It was also found that PIMS could execute chain instructions and support the Remote mode in a "busy" state simultaneously. I.e. a mode code such as Synchronize (Mode 1 in MIL-STD-1553B) typically caused the channel to respond busy to subsequent commands and created an interrupt to the host. PIMS will similarly begin to respond busy but instead of interrupting the host it will start an Exception chain. The channel can scan the 1553 bus for traffic addressed to it while it executes the Exception Chain. Should a command come in it suspends the chain, processes the message, and returns to the chain, which will eventually complete, causing the channel to return non-busy. This feature is a less important but revealing feature of the power of a channel processor.

III. WHY BOTHER?

It is not a foregone conclusion that because a channel processor like PIMS can be built it should be built. Why not keep the CP in control and the channel "simpler"? A few reasons follow.

MULTIPLE CHANNELS

Complexity of avionics systems is increasing and multiple channel processors will be required. To handle exception processing for multiple channels will take more CP time and require more complicated interrupt handlers. Many "exceptions" are actually routinely occurring events, such

Special Instruction Features

<u>Feature</u>	<u>Use</u>
- Implicit RT Address in Bus Instruction	Allows a message to be sent to the last referenced RT without wasting code to determine which it was. Facilitates exception processing.
- Configuration Control Table	Allows all instructions to a given RT to be NO-OP'ed by clearing one bit in a channel control register.
- Jump On Exception	Allows programmer to conveniently check for exceptions on bus instructions within an Exception Chain.
- "Skip" bit in Bus Instruction	Allows NO-OPing of a bus instruction by setting a single bit within it.
- Exception Masking	Allows certain exception conditions to be ignored via a bit mask. Simplifies exception handling.
- Exception Chain Override	Suppresses Exception Chaining on a per bus instruction basis.

Table 2 Special Channel Features

as Service Requests from RT's, so the CP time taken can be non-trivial.

If the channels are PIMS - like channel processors, only the load on main memory for instruction fetches and data buffer transfers increases with the number of channels.

FLEXIBILITY

While PIMS has the ability to act as a nearly autonomous processor, the system designer can use it any way he chooses. I.e. he can make the CP take as much of its traditional role as he wishes using the interrupt options in the channels instruction set. However, the more autonomy allowed the channel, the better the system throughput tends to be.

POTENTIAL FOR STANDARDIZATION

Full channel processing capability represents a plateau in that while processor speed, module power, instruction set power might change, the fundamental nature of the channel will not. Thus the channel is functionally mature enough to make a standard MIL-STD-1750A to MIL-STD-1553 channel protocol feasible.

SUMMARY

PIMS reflects ideas and realities which have existed since the first bus controller. There has always been a need for intelligent processing to support any non-trivial bus control function, and even the simplest bus controller requires processing power just to handle the protocol. PIMS has created a channel with power to support the protocol and the processing, all on a single module, and has further implemented code structures and instructions to exploit this power. PIMS can serve as a basis for further enhancements and the development of host to channel conventions.

BIOGRAPHY: Bob Salter has worked for Sperry Univac since graduating with an M.S.E.E. from the University of Minnesota in 1975. He has worked in I/O from discrete transceiver design to I/O processor design, primarily for serial interfaces. He is currently a member of the SAE A2K High Speed Bus Subcommittee.

AD-P003 537

SINGLE CHIP MIL-STD 1553B BUS INTERFACE UNIT
KEEPS PACE WITH CHIP SETS

Scott Schaire

Grumman Aerospace Corporation
Bethpage, N.Y. 11714
LR5-9418

ABSTRACT

The introduction of chip sets for multiplex terminal designs that provide savings in hardware and software may leave the designer confused as to which chip set to use for a particular application. The Grumman/Standard Microsystem Corp (SMC) Bus Interface Unit (BIU) is compared to other BIUs in terms of capabilities and amount of peripheral chips required. It is shown that the single-chip Grumman/SMC BIU performs similar functions of the other chip set BIUs at a lower cost.

INTRODUCTION

The MIL STD 1553B Large Scale Integration (LSI) chip sets replace a considerable amount of "older technology" circuitry reducing chip count, circuit complexity, power dissipation, and cost. What, in particular, is replaced; which functions are included in the LSI chip sets and what must be performed externally? What are the distinctive features of the new BIUs and for what applications are they best suited? These are some of the questions addressed by this paper.

First, the Grumman/SMC single-chip LSI is presented. Functional elements of the Grumman/SMC BIU are shown and their operation is discussed. The tradeoffs associated with placing MIL STD 1553B multiplex terminal functions on or off the chip are also discussed. Included on the chip is direct memory access (DMA) handshaking for the 16-bit parallel interface to the subsystem. All parallel transfers use this DMA handshaking; the latter is described in detail.

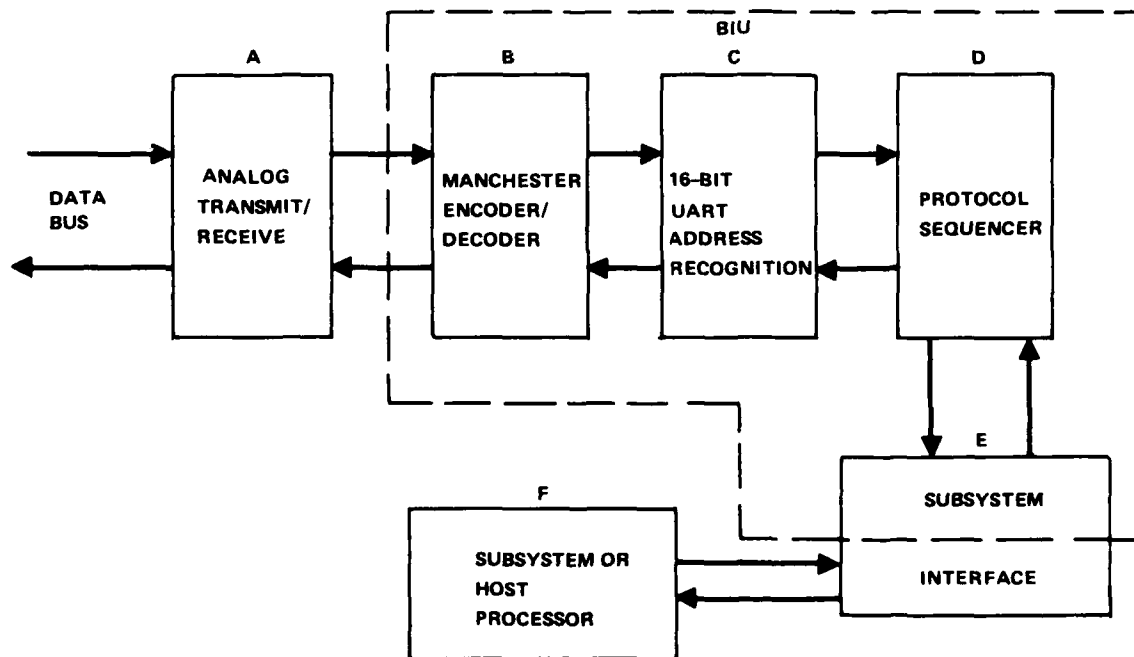
The Grumman/SMC BIU also includes unique capabilities, such as extensive error detection and diagnostics, which make it ideal for many multiplex terminal applications. Its universal DMA port enables the use of a minicomputer, 16- or 8-bit microprocessor, first-in/first-out (FIFO), or no processor as the terminal host processor. The BIU detects noncontiguous words and RT-RT transfers as a Remote Terminal (RT) or Bus Controller (BC) which avoids confusion for all types of RT-RT transfer errors. Switching from Remote Terminal to Bus Controller mode or vice versa simply involves a toggle of the RT/BC input. This feature is particularly desired for dynamic bus or back-up bus controller applications.

Besides the Grumman/SMC BIU, four other MIL STD 1553B LSI chip sets have been introduced. The Harris RT chip, Marconi chip set, Smith chip set, and Rockwell Collins chip are compared to the Grumman/SMC BIU. Major differences are highlighted and suggested applications presented. A comparison chart is included which covers areas such as cost, error-detection capability, and amount of peripheral circuitry required for specific applications.

Each chip set offers advantages for various multiplex terminal designs. For most applications, the Grumman/SMC BIU offers the MIL STD 1553B features desired at the lowest system cost.

WHAT IS REPLACED?

Figure 1 shows the functional elements required by a single bus MIL STD 1553B Remote Terminal, Monitor, or Bus Controller. The Analog Transceiver (A) includes signal limiting, threshold detection, bus driving, and 800 usec timeout circuitry. This element is not replaced by any of the BIUs because of the difficulty of combining analog and digital functions in a single monolithic device. The Encoder/Decoder (B) translates between Manchester and Non-Return-to-Zero (NRZ) serial formats, detects syncs and word errors, and encodes sync and parity. The UART (C) is a serial-to-parallel converter for words received by the BIU and parallel-to-serial converter for transmitted words. Command word decoding, wordcount recognition, mode code and broadcast detection, and status word and last command word manipulation are functions of the Protocol Sequencer (D).



1806-001P

Figure 1. Single Bus Terminal Functional Elements

The Grumman/SMC BIU includes elements B, C, D, and part of the Subsystem Interface (E). The latter is responsible for transferring 16-bit words between the BIU and the subsystem or host processor (F). This interface is partially replaced by the BIU. Flexibility for a variety of subsystems is provided by a smaller external interface (E) which tailors the BIU general-purpose handshake signals to specific subsystem host processors.

GRUMMAN/SMC BIU OPERATION

The Grumman/SMC MIL-STD 1553B BIU, shown in Figure 2, consists of a Manchester encoder decoder (see B, Figure 1), double-buffered transmit and receive registers (see C, Figure 1), five registers, response timer, word-counter, protocol sequencing and error logic (see D, Figure 1). Also included on the chip is the sequencing logic necessary to initiate and control DMA transfers to the subsystem (see E, Figure 1).

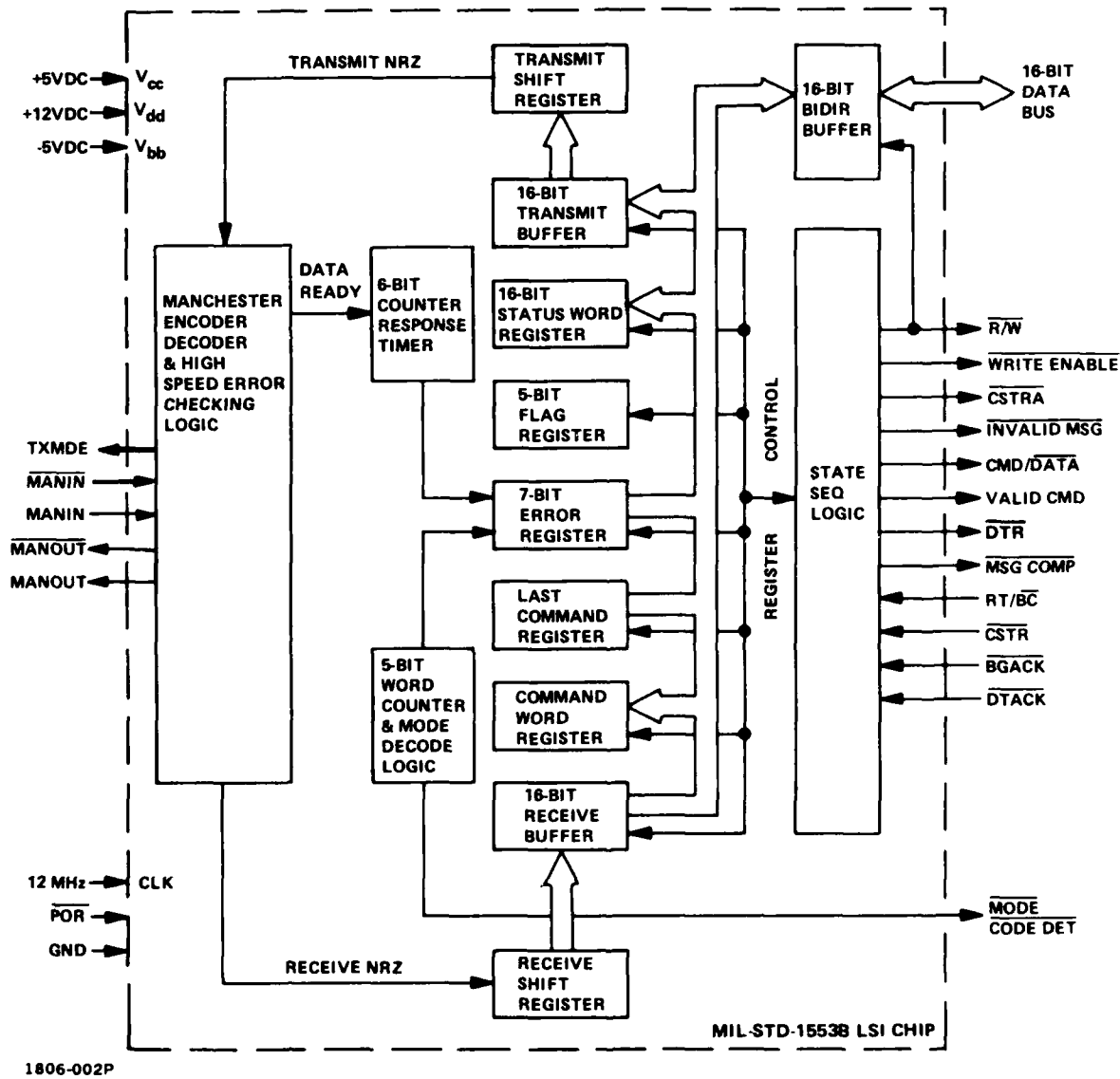


Figure 2. MIL STD 1553B LSI Chip

When operating as a Remote Terminal (RT), the RT/BC input is held high by the subsystem. The chip remains idle until the status word and remote terminal address are loaded. Message reception is then initiated when the chip receives a valid command word through the Manchester decoder, receive shift register, and receive buffer. The Last Command register is loaded with the contents of the Command Word register, and the receive buffer is transferred to the Command Word register.

Eleven bits of the Command Word register are decoded for mode codes (five-bit subaddress equals all ones or zeros), transmit or receive (T/R bit), and a word count. For non-mode code commands, the wordcounter is loaded with the five-bit wordcount field of the command word (00000 equals 32 words). For mode code commands, the wordcounter is loaded with the MSB of the wordcount field (<10000 loads zero, ≥10000 loads one).

For transmit commands, the status word is converted from parallel to serial through the transmit buffer and transmit shift registers and Manchester-encoded and transmitted. Then the command word and status word are consecutively written into memory by DMA transfers. The BIU reads subsequent data words from memory and transmits them over the MIL STD 1553B bus until the wordcount is zero.

For receive commands, the BIU performs the Manchester decoding and serial-to-parallel conversion, and writes the command word followed by the data words into memory. When the wordcount is zero, the status word is transmitted unless the command was a Broadcast or there was a Message Error.

The transmitted status word is a logic combination of a double-buffered status word register loaded by the host processor and those bits enabled and set by internal decode and error logic. After a status transmission, all bits are cleared except the dynamic bus accept bit until they are again set by the load RT address BIU command. However, transmit last command and transmit status word mode commands do not clear these bits.

Inclusion of the status word, last command word, and mode code logic on chip relieves the subsystem host processor of any high-speed command word decoding requirement. Transmit Last Status, Dynamic Bus Control Offer, Inhibit Terminal Flag, Override Inhibit Terminal Flag, and Transmit Last Command mode codes are automatically handled by the BIU. Other optional mode codes (such as Transmit Bit Word) may be properly serviced by relatively slow microprocessor host software.

When operating as a Bus Controller (BC), the BIU first reads a 3-bit word through DMA which specifies the operation (e.g., normal bus controller transfer, RT-RT, write register, etc). The next word it reads is placed in the command word register and the transmit buffer, encoded with a command sync, and transmitted. The BIU performs transmit, receive, decoding, and DMA functions in a manner identical to RT operation.

Universal DMA Sequencing

In order to achieve the complex timing and sequences required while minimizing chip area, a state sequencer architecture was chosen. The state

sequencer controls all registers, counters, logic blocks, and the DMA handshaking. It runs at 2MHz and is the equivalent of 15,000 bits of read-only-memory (ROM).

All data transfers required for operating the Grumman/SMC BIU are handled using DMA techniques. Although the mention of DMA may bring to mind critical timing requirements, the method incorporated in this design uses asynchronous handshaking to eliminate bus contention problems.

The DMA control is performed in a straightforward, simple manner. There are four types of DMA transfers (command write, data write, data read, and control read), and these are the only means of communication between the BIU and the subsystem. Therefore, the only external circuitry required for the subsystem is an interface for the four DMA transfers.

The format for writing a command word into memory by DMA is as follows:

- Grumman/SMC BIU activates Data Transfer Request (DTR) to inform the subsystem interface that it wants use of the 16-bit data bus. The Command/Data (C/D) line is asserted high and Read/Write (R/W) is asserted low to indicate a writing of a command word.
- When Bus Grant Acknowledge (BGACK) is recognized by the BIU, the command word is placed on the bidirectional data bus (DI5-DO) and DTR is negated. Write Enable (WE) is activated when the data is valid. The subsystem interface may optionally decode the command word subaddress field, set up an external address counter, and write the command word
- The BIU continues outputting the command word until Data Transfer Acknowledge (DTACK) is received. When the data bus is no longer in use by the BIU, WE and R/W are negated. This informs the subsystem interface that it may return use of the 16-bit bidirectional bus to the subsystem.

DMA handshaking for a data write is identical, except that C/D is negated for the transfer.

Processor-initiated DMA transfers are used for all BC operations, and loading and monitoring of internal registers as an RT. After three control bits and other required information (e.g., bus controller command files) are set up in memory, the subsystem interface may assert Command Strobe (CSTR). When the BIU recognizes CSTR (within 500 nsec, if not in the middle of a message), it will pulse Command Strobe Acknowledge (CSTRA) low for 500 nsec. This signal is utilized by the subsystem interface to negate CSTR and load an external address counter with the address of the three control bits.

The Grumman/SMC BIU will also concurrently initiate a DMA read cycle by providing the following handshake signals:

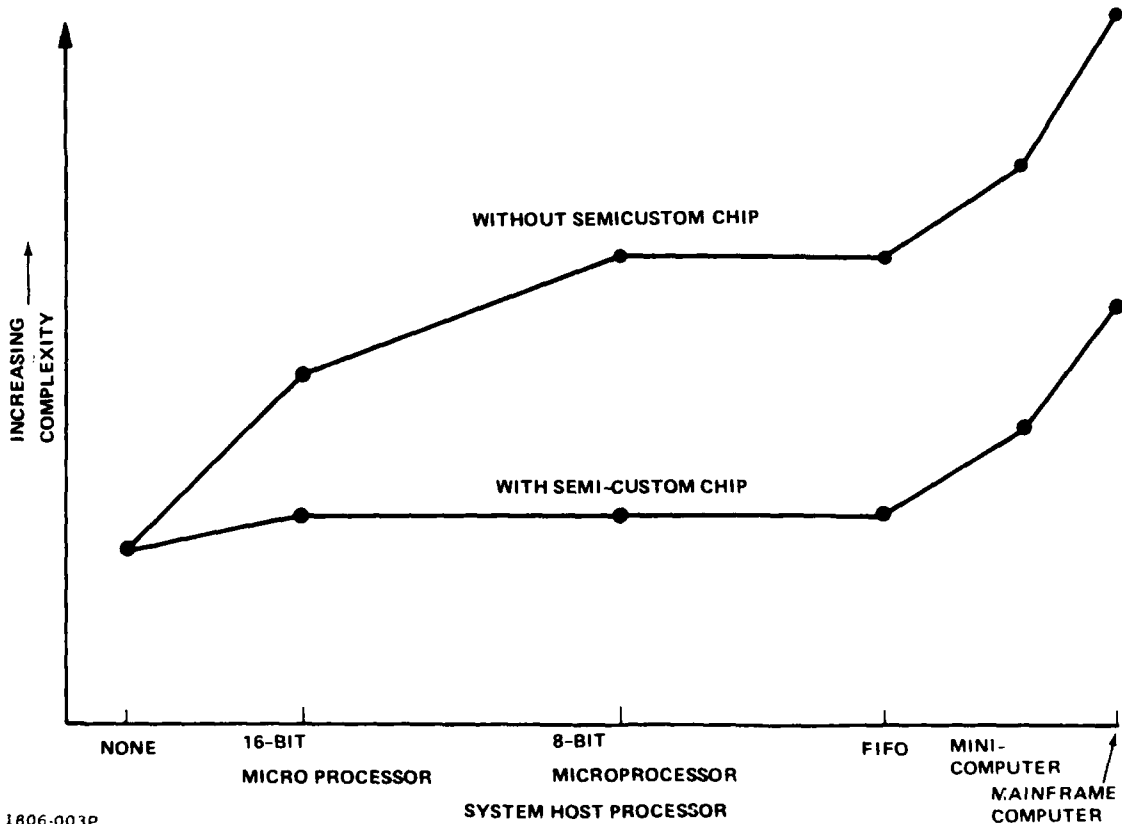
- BIU asserts DTR, R/W high, and C/D low for data
- When the subsystem interface asserts BGACK, indicating that the 16-bit bus is free, the BIU negates DTR

- The subsystem interface may then perform the read of the three control bits and assert DTACK when the data is valid
- The BIU will latch the data internally and assert C/D high, indicating completion of the DMA read
- The subsystem interface may then return control of the data bus to the subsystem.

Note that a data read is identical to the sequence described above, except that CSTR is not pulsed. The same handshaking applies to reading command words, RT address and status word, and data words as either an RT or a BC.

Since the subsystem interface requirement is for universal DMA handshaking, the BIU can be used in numerous applications with various subsystem host processors. Grumman is now ready to specify a semi-custom monolithic chip which will contain a 16-bit address counter and other registers to replace the multi-chip subsystem interface. Figure 3 shows the external circuitry required for applications using the Grumman/SMC BIU.

It should be obvious that the more sophisticated the terminal, the more intelligence is required in the subsystem DMA interface.



1806-003P

Figure 3. External Circuitry Required for a Subsystem Interface for the Grumman/SMC BIU

Error Detection

Another unique feature of the Grumman/SMC BIU is its error detection capability. The Grumman/SMC BIU recognizes the following errors:

- Improper sync
- Invalid Manchester II code
- Information field > 16 bits
- Not odd parity
- Improper word count - includes too few, too many, and noncontiguous words
- Response time - as a BC, the amount of time between the end of transmission of a message and the status word reply by a remote terminal is greater than 14 μ sec. For RT-RT transfers, the status words of both the transmitting and receiving remote replies are checked. As an RT instructed to receive an RT-RT transfer, the status word of the replying terminal does not occur within 14 μ sec from the receipt of the transmit command
- Address mismatch - as a BC, the address of the status word reply from a remote terminal does not match the remote terminal address field of the command. For RT-RT transfers, both status words are checked.

On-Line Diagnostics

The Error Register is written into memory automatically, along with the Last Command and Data Registers at the end of all RT messages. Any of these registers may be written into memory at the request of the subsystem host processor by issuance of a CSTR with the proper three control bits (i.e., 001 = error and remote terminal address, 000 = last command, 11 X⁽¹⁾ = data register). Writing of error and remote terminal address registers allows a remote terminal to update a Build-In-Test (BIT) word and check its own address. The Last Command Word register provides an automatic reverse-linked list of the commands received by the RT for additional software diagnostics. The Data Register can be viewed by the host processor to diagnose transmitter, receiver, or bus problems.

Transmission Continuity

There are areas of the 1553B standard that, when implemented on a BIU, require interpretation by the BIU designer. One such section is transmission continuity, which states that a terminal must verify that a received message is contiguous. However, the standard does not specify a tolerance on contiguity. That is, what interword gap is to be considered a message error?

(1) X indicates don't care.

Clearly, if the gap exceeds 4.0 μ sec, the minimum intermessage word may be considered part of a new message. Using 4.0 μ sec as a t will allow up to that gap without detecting a message error. But do your RT to respond when there's a 3.95 μ sec interword gap?

The Grumman/SMC BIU is designed with fail-safe features and ass the received message is in error when the interword gap exceeds 0.5 When the wordcount is not zero, the BIU will wait 0.5 μ sec for a new the word is not received, the Message Error (ME) bit is set and stat transmission is suppressed.

RT-RT

With a response timer and extensive error detection built into the Grumman/SMC device will never become confused by RT-RT transfer Place the BIU in RT mode as the receiving terminal and delay the tra terminal's response by more than 14 μ sec. The BIU will not hang up will timeout, setting the response time error bit in the error regis the ME bit in the status word. RT-RT smarts included in the Grumman make it ideal for these applications as a RT or BC.

RT-BC with the Flip of a Bit

The Grumman/SMC BIU is also best-suited for Dynamic Bus Allocat Backup Bus Controller applications because of both the subsystem int Bus Controller operation. The subsystem interface requires the serv four general purpose DMA transfers whether the BIU is in the RT or B Mode switching is accomplished by simply flipping the RT/BC input. no need to beef-up the subsystem interface to handle different paral transfers or timing with the Grumman/SMC BIU.

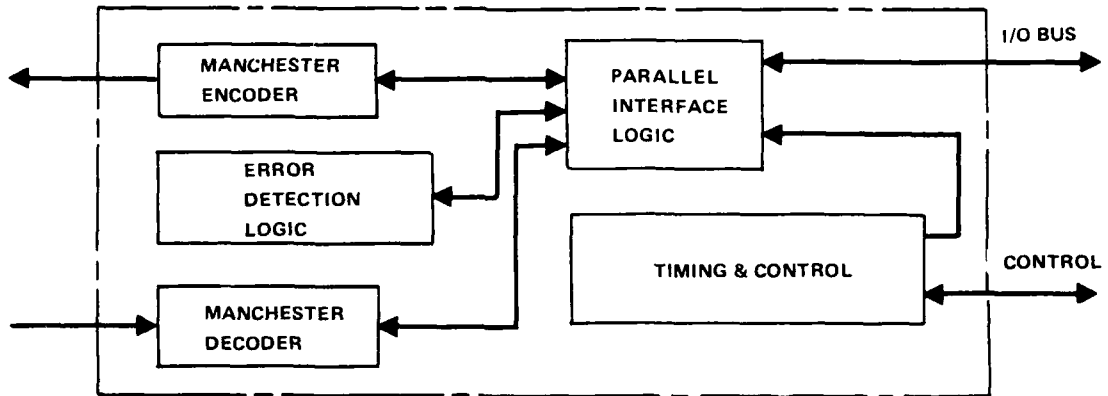
Not only is the external hardware simple for BC mode operation, the software. The host microprocessor must prepare a memory file wi control word, specifying normal or RT-RT transfer followed by the me There is no requirement for specification of transmit or receive, mo broadcast, or wordcount. The Grumman/SMC BIU automatically decodes fields from the command word that it reads from memory.

OTHER BIUs

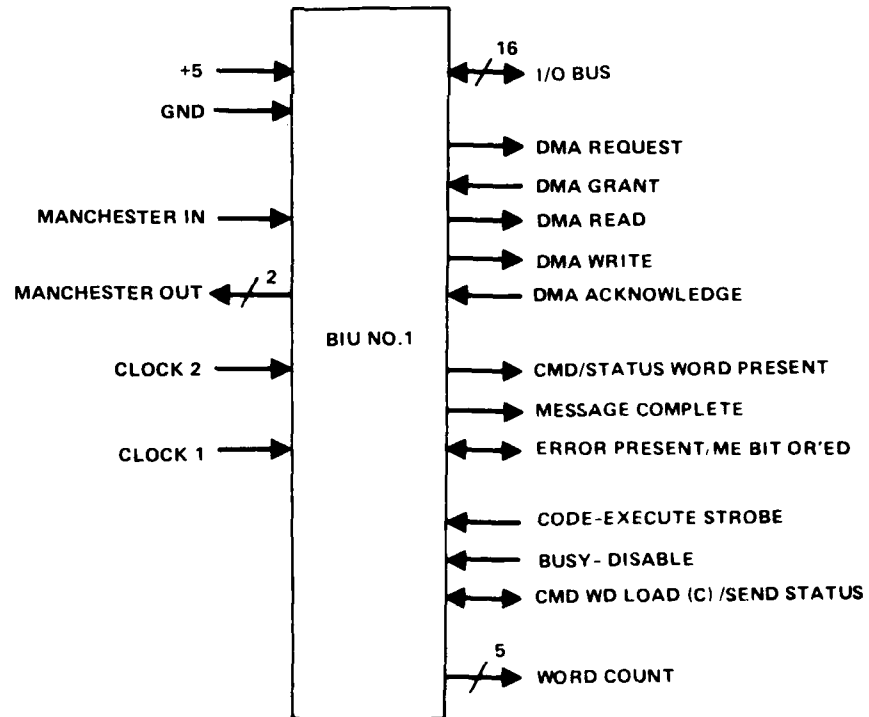
The other chip set MIL-STD 1553B BIUs being introduced are very to the Grumman/SMC BIU in terms of functional capability. Major dif are in the implementation of bus controller operation, error detecti diagnostic features, the subsystem interface, cost, and chip count. Harris BIU No. 1 chip, Marconi chip set, Smith chip set, and Rockwel chip are compared to the Grumman/SMC BIU. The outstanding advantage Disadvantages are discussed below. An overall comparison chart is p (Table 1) which provides a brief overview of chip set capabilities. Compared were chosen with many terminal designs in mind; however, ea tion is unique. It is recommended that readers perform their rison.

Harris BIU No. 1

Figure 4 shows a basic block diagram and functional pinout of the Harris BIU No. 1. Its major advantages are the ability to operate at up to 5 MHz data rate for non-MIL STD 1553B applications and its simplicity of use in MIL STD 1553B RT designs. However, designs of bus controller terminals using the HS-3273 (Harris BIU No. 1) require a larger number of components and host assistance unlike the Grumman/SMC BIU. This is due to the lack of protocol sequencing and subsystem interface logic. A second LSI chip, HS-3274, which



(BASIC BLOCK DIAGRAM)



(FUNCTIONAL PINOUT)

1806-004P

Figure 4. Harris BIU No. 1

will significantly reduce component and host assistance requirements is presently only in the conceptual design stage at Harris.

Smith BIU Chip Set

The Smith BIU chip set shown in Figure 5 consists of three devices, an encoder/decoder - UART (MT 32006); a protocol sequencer (MT 32004); and a 16-bit wide by 32-bit long first-in/first-out (FIFO) memory (MT 32003- not shown). Its unique features include extensive mode-code decoding and a FIFO for burst mode DMA applications. One major disadvantage of the Smith BIU chip set is its size and cost. Three chips cost more than \$1,000 as opposed to the single-chip Grumman/SMC BIU at approximately \$250.

Marconi BIU Chip Set

The Marconi BIU chip set consists of four devices: a receiver, transmitter, interface unit and internal highway control logic unit. Because of a separate receiver device, the Marconi BIU chip set is best-suited for triple and quad redundant applications. Each level of redundancy requires only an extra receiver to properly handle superceding valid commands. Similar to the Smith BIU, the Marconi BIU includes extensive mode-codedecoding. It also falls short of the Grumman/SMC BIU in terms of size and economics. Four chips costing more than \$1,000 comprise the Marconi BIU chip set. When component handling, board density, board handling, and wiring complexity are considered, the cost of the Marconi BIU chip set becomes even greater compared to the Grumman/SMC BIU.

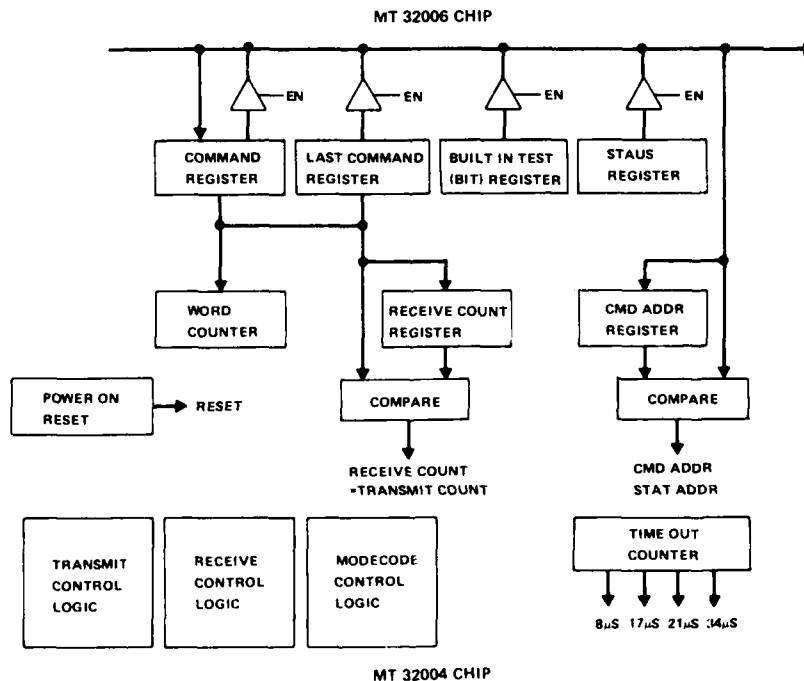
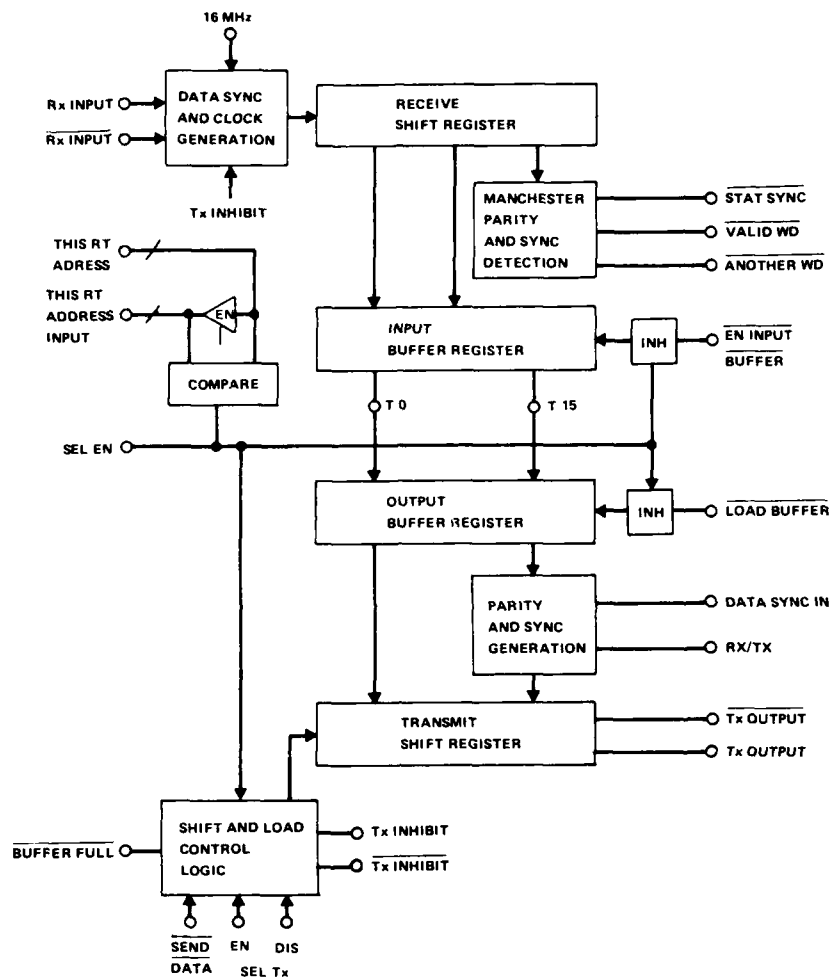
Rockwell/Collins BIU

The newest of the BIUs, the Rockwell/Collins, is shown in Figure 6. It represents the largest deviation from the other BIUs since two Manchester decoders are included on a single chip. It is obviously best-suited for dual-redundant terminal applications. It is capable of meeting the MIL STD 1553B, but only with quite extensive additional circuitry for protocol sequencing and a subsystem interface. Since the capabilities of this BIU depend largely on its external electronics, it is not fair to include the Rockwell/Collins in a comparison chart of "full function" BIUs.

CONCLUSION

Table 1 shows an overall comparison of four of the BIUs. The simple RT application listed is for an RT that transmits a bank of 16-switches in response to a transmit command. This simple RT also provides for the five mode codes handled by the Grumman/SMC BIU. The reason the chip set BIUs require less supporting circuitry is because their status word and remote terminal address are hardwired rather than soft loaded.

The BIUs discussed in this paper are all fully capable of implementing all MIL STD 1553B options. Some have less capabilities on chip than others while claiming increased flexibility for various subsystems and future applications. The Grumman/SMC single-chip BIU certainly keeps pace with the other BIUs in terms of capabilities on chip. Its subsystem interface provides the flexibility for future mode commands and status bits and a variety of host



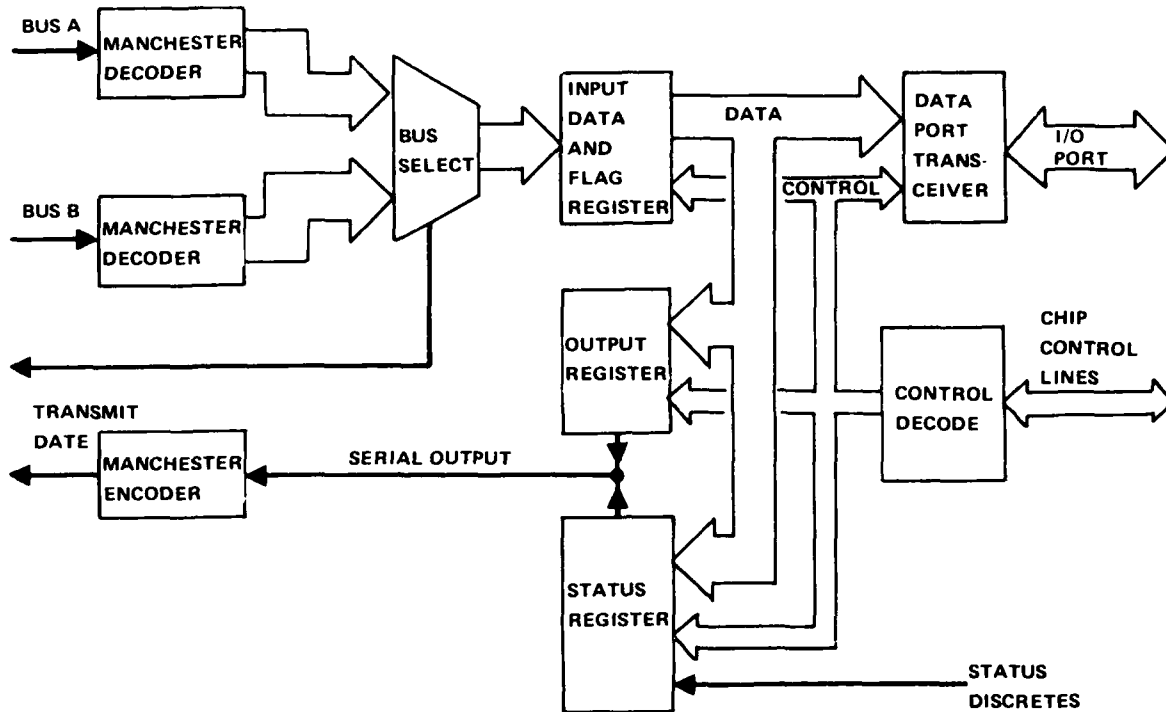
1806-005P

Figure 5. Smith BIU Chip Set Simplified Block Diagram

Table 1 Overall Comparison

GRUMMAN/SMC BIU CHIP	HARRIS BIU # 1 CHIP	SMITH BIU CHIP SET	MARCONI BIU CHIP SET
PROVIDES 16-BIT PARALLEL INTERFACE TO THE SUB-SYSTEM	8-BIT/16-BIT	16-BIT	8-BIT
ALLOWS FOR SETTING OF RESERVED STATUS BITS	NO	NO	NO
CONTAINS DOUBLE-BUFFERED I/O, 4 DOUBLE-BUFFERED STATUS BITS	SAME AS GRUMMAN/SMC	FIFO-BUFFERED I/O 4 DOUBLE-BUFFERED STATUS BITS	DOUBLE-BUFFERED I/O, 4 DOUBLE-BUFFERED STATUS BITS
PROVIDES DMA HANDSHAKE CONTROL ASYNCHRONOUS DATA TRANSFER	SAME AS GRUMMAN/SMC	SYNCHRONOUS TRANSFER CONTROL	DMA HANDSHAKE CONTROL SYNCHRONOUS DATA TRANSFER
HANDLES RT-RT COMMANDS	YES	YES	YES
HANDLES BROADCAST	YES	YES	YES
NO	PROVISIONS FOR OPERATING AT DIFFERENT BIT RATES	NO	NO
CAPABLE OF READING LAST COMMAND, ERROR, OR DATA REGISTERS	CAN ONLY READ ERROR - AUTOMATICALLY CHECKS OWN TRANSMISSION	CAN ONLY READ ERROR	CAN ONLY READ ERROR
ABOVE REGISTERS AUTOMATICALLY DMAED INTO MEMORY AT END OF RT MESSAGES	NO	NO	NO
40-PIN SINGLE CHIP	40-PIN SINGLE CHIP	48-PIN ENCODER/DECODER - UART 40-PIN FIFO 53-PIN PROTOCOL SEQUENCER	40-PIN RECEIVER 40-PIN TRANSMITTER 48-PIN INTERFACE UNIT 40-PIN INTERNAL HIGHWAY CONTROL LOGIC
FOUR REGISTER CHIPS THREE LOGIC CHIPS TRANSCIVER/TRANSFORMER AND OSCILLATOR IS SUPPORTING CIRCUITRY FOR SIMPLE RT APPLICATION	SIMILAR TO GRUMMAN/SMC	SLIGHTLY LESS THAN GRUMMAN/SMC	SLIGHTLY LESS THAN GRUMMAN/SMC
ADDITIONAL BIU NEEDED FOR REDUNDANCY	SAME AS GRUMMAN/SMC	ADDITION ENCODER/DECODER-UART	ADDITIONAL RECEIVER
HANDLES: DYNAMIC BUS CONTROL TRANSMIT STATUS WORD INHIBIT T/F BIT OVERRIDE INHIBIT T/F BIT TRANSMIT LAST COMMAND CODE CODES AUTOMATICALLY	SAME AS GRUMMAN/SMC	HANDLES ALL MODE CODES AUTOMATICALLY	HANDLES ALL MODE CODES AUTOMATICALLY
PROVIDES FOR READING OF SOFTWARE LOADED RT - ADDRESS	NO	HARDWIRED WITH PARITY BIT	HARDWIRED WITH PARITY BIT
NMOS	JUNCTION - ISOLATED CMOS	CMOS	METAL GATE CMOS
APPROX \$250	APPROX \$500	> \$1,000	> \$1,000

1806-007P



1806-006P

Figure 6. Rockwell/Collins BIU Chip Functional Block Diagram

processors. Considering the supporting hardware and software required, the Grumman/SMC BIU provides the MIL STD 1553B features required for most applications at the lowest system cost.

ACKNOWLEDGEMENTS

The author would like to thank Messrs. John Cavin and Steve Dzierzynski of Grumman Aerospace Corporation for their valuable systems support. The author would also like to thank Messrs. Kiran Shah, Doug Pastorello, and Richard Bozz of Standard Microsystems Corp. (SMC) and Messrs. Tony Haley, Page Kahler, and Harold Alber of the Systems Engineering Avionics Facility (SEAFAC) for assistance in testing and debugging of the Grumman/SMC BIU breadboard and chip.

REFERENCES

1. Schaire, S. and Susinno, C., "Advanced Smart Terminal Description," Grumman Aerospace Corporation, Engineering Development Center, Advanced Development Report, May, 1982.
2. Schaire, S. and Cavin, J., "Single Chip Bus Interface Unit Eases MIL-STD 1553B Remote Terminal/Bus Controller Designs," National Aerospace and Electronics Conference, May, 1982.

AUTHOR BIOGRAPHY

Scott Schaire graduated from Cornell University School of Electrical Engineering in May 1979 and has completed a Master of Computer Science Degree at Polytechnic Institute of New York.

At Grumman, Mr. Schaire has spent three years in various advanced development programs. As part of his responsibilities for the design of MIL STD 1553B Smart Terminal, he has developed sequencing logic for a custom LSI Bus Interface Chip (patent pending), designed a subsystem interface for the MC68000 Microprocessor, and developed the test software.

MIL-STD-1553B MARCONI LSI CHIP SET
IN A
REMOTE TERMINAL APPLICATION

Albert DiMarino

Circuit Technology Inc.
160 Smith Street
Farmingdale, New York 11735
Tel. (516) 293-8686

Mr. DiMarino has a BSEE from the University of Buffalo and is the National Marketing Manager, Standard Products at Circuit Technology Inc., a wholly-owned division of Marconi Electronic Devices Ltd., U.K. Prior to joining Circuit Technology Inc., he was employed by ILC Data Device Corp., Bohemia, N.Y. and Andersen Laboratories, Inc., Bloomfield, Conn. in marketing positions.

Marconi Avionics is utilizing the MIL-STD-1553B LSI Chip Set in the SCADC Air Data Computer application to perform all of the required remote terminal MIL-STD-1553B protocol functions. Basic components of the RIU are the dual redundant chip set, CT3231 Transceivers, 256 x 16 RAM and a Z8002 micro-processor.

Basic transfers are to/from the RAM on command of the bus controller or Z8002 processor. During transfers from the processor to the RAM, the chip set busy bit is set for a period not exceeding 250 microseconds. When the transfer is complete, the busy bit is released and transfers to the data bus occur on command.

The LSI Chip Set word count lines are used to locate each data word in the local memory and 4 mode codes are used in the application: reset remote terminal, transmit status word, transmitter shut-down, and override transmitter shutdown.

THE INTRODUCTION

Circuit Technology Inc. is marketing in the U.S. the MIL-STD-1553B LSI Chip Set developed in the U.K. by Marconi Electronic Devices Ltd. The LSI Chip Set is presently being evaluated in the U.S. by many companies for remote terminal and bus control applications. The basic chip set performs both functions and is converted from RT to bus controller on command. Marconi Avionics, Rochester, U.K. is using the MIL-STD-1553B LSI Chip Set to perform the remote terminal requirement for the SCADC Air Data Computer. A Z8002 microprocessor is used to control transfers to/from the LSI Chip Set.

REMOTE TERMINAL UNIT BASIC CONFIGURATION

The dual redundant remote terminal consists of isolation transformers,

suitable for direct and stub coupling, CT3231 transformers, the 5 chip MIL-STD-1553B set in leadless chip carriers, buffered and mounted on a 1.6" x 3.2" ceramic substrate, 256 x 16 RAM and additional logic to perform the necessary chip set/Z8002 transition hardware. See Fig. 1.

TRANSCIEVER DESCRIPTION

The CT3231 design transceiver used on the SCADC Program was developed by Circuit Technology Inc. and is used in substantial quantity on B-52, A-10, F-16, and other programs. The CT3231 features low offset, 4 ohm typical transmitter drive impedance, AC interstage coupling, and is compatible with MIL-STD-1553 A and B. The CT3231, CT3232, CT1487, and CT1589 transceivers all couple directly to the LSI Chip Set.

MIL-STD-1553B LSI CHIP SET

The Marconi LSI Chip Set meets all of the protocol requirements of MIL-STD-1553B for operation as a remote terminal, passive monitor, and bus controller. Four chips comprise the basic set for a single channel requirement. Up to triple redundancy can be accomplished with the addition of a single decoder chip for each redundant channel. See Fig. 2. The four chips and the functions performed of each are detailed below.

Decoder:

- Waveform reception from a suitable bus receiver element.
- Manchester encoding error check.
- Bit count check.
- Parity error check.
- Sync waveform check.
- Continuity check.
- RT address comparison.
- Mode command detection.
- Broadcast command detection.
- Terminal reply timeout check.
- End of transmission detection.

Encoder:

- Waveform encoding and transmission to a suitable bus driver.
- Command execution state sequencing.
- Transmitter timeout control.
- Bus shutdown control.
- Mode command execution.
- Self-test execution.

Subsystem Interface Unit:

- Command decoding.
- Current and last command recording.
- Illegal command detection.
- Word count comparison.
- Mode command execution.
- Status recording.
- BIT recording.

Internal Highway Control Logic:

- Bus selection.
- Strobe generation.
- RT control line generation.
- BC operation decoding and control line generation.
- Terminal - subsystem handshaking.

All protocol requirements of MIL-STD-1553B are performed by the LSI Chip Set including command word decoding, message validation, word count, status word return, etc. The chip set handles the complete message sequence and all mode codes are implemented as is broadcast capability. The chip set is available in leadless chip carriers, dual-in-line packages, as a hermetic sealed hybrid assembly, and as a leadless chip carrier assembly on a ceramic substrate. For detailed information on timing, performance parameters and packaging the reader is referred to in Circuit Technology Inc. Data Sheet CT1561. Standard units are screened to MIL-STD-883B quality level.

REMOTE TERMINAL MODULE DESCRIPTION

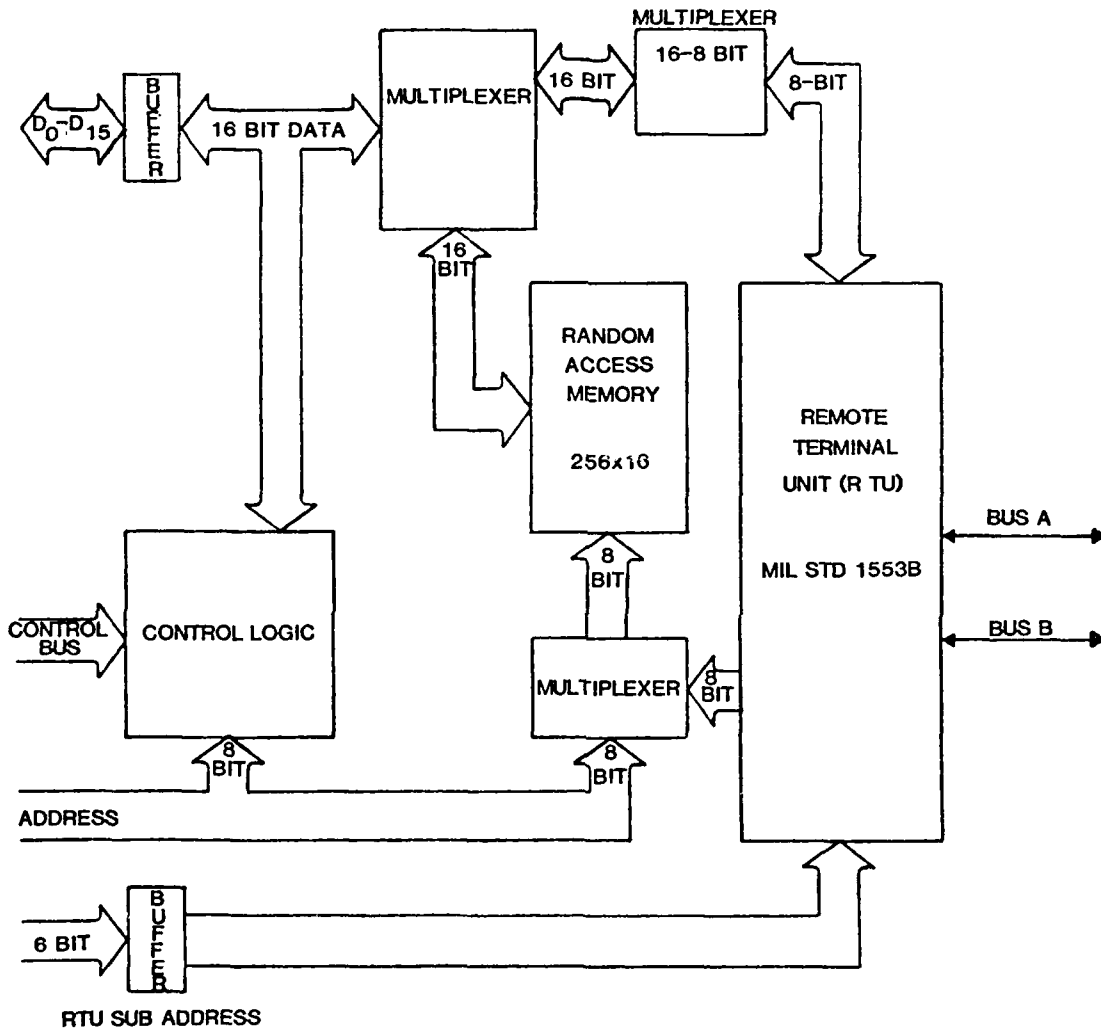
Marconi Avionics, Rochester, U.K. has designed and built one of the first operating remote terminal units using the MIL-STD-1553B LSI Chip Set. The remote terminal is now operating in an engineering model of the SCADC Air Data Computer. See photograph and the block diagram of Fig. 1.

The Remote Terminal Module (RIM) contains a 1553B chip set hybrid (RT hybrid) combined with two hybrid transceivers and two isolation transformers giving a dual redundant, transformer coupled, Remote Terminal (RT) configuration. A ram store, accessible by the Air Data Processor (ADP) or the RT, is provided for the temporary storage of data for transmission via the 1553B bus. This store of 256 words by 16 bits provides the RT with a choice of eight sub-addresses, or blocks, of 32 words.

After reception of a valid command word from the bus controller, data transfers with the ram will be controlled by 8 bits from the RT hybrid. The starting address in the ram is defined by a three bit subaddress which is latched with each new command word received. The other five ram address lines are the current word count lines (CWC) from the RT hybrid, which are incremented for each new data word, from 0 to 32. Reading or writing is defined by the Tx/Rx line from the RT hybrid. During the receive operation, an 8 bit addressable latch is provided for the RT hybrid to flag up a possible data message error (\overline{VBR}). As the addresses are defined, data is transferred between the RT hybrid and the ram in parallel 8 bit bytes. This operation is controlled by the RT hybrid which transfers the most significant byte followed by the least significant byte to or from the ram. The ram address remains the same until both bytes have been loaded.

Microprocessor Request (UP REQ) and Microprocessor Accept (UP ACC) handshake signals are used for the Air Data Processor (ADP) to gain control of the ram. Once control of the ram has been received by the ADP, a busy signal is flagged to the RT hybrid for a period not exceeding 250 usec during which period data transfers from the ADP to the ram occur. When the data transfer is complete, the busy signal is released and transfers from the ram to the 1553 Data Bus are controlled by the LSI Chip Set.

In addition to the basic message transfers, four of the mode codes implemented in the LSI Chip Set are utilized: reset remote terminal, transmit status word, transmitter shut-down, and override transmitter shut-down.



REMOTE TERMINAL MODULE (RTM)

FIG. 1

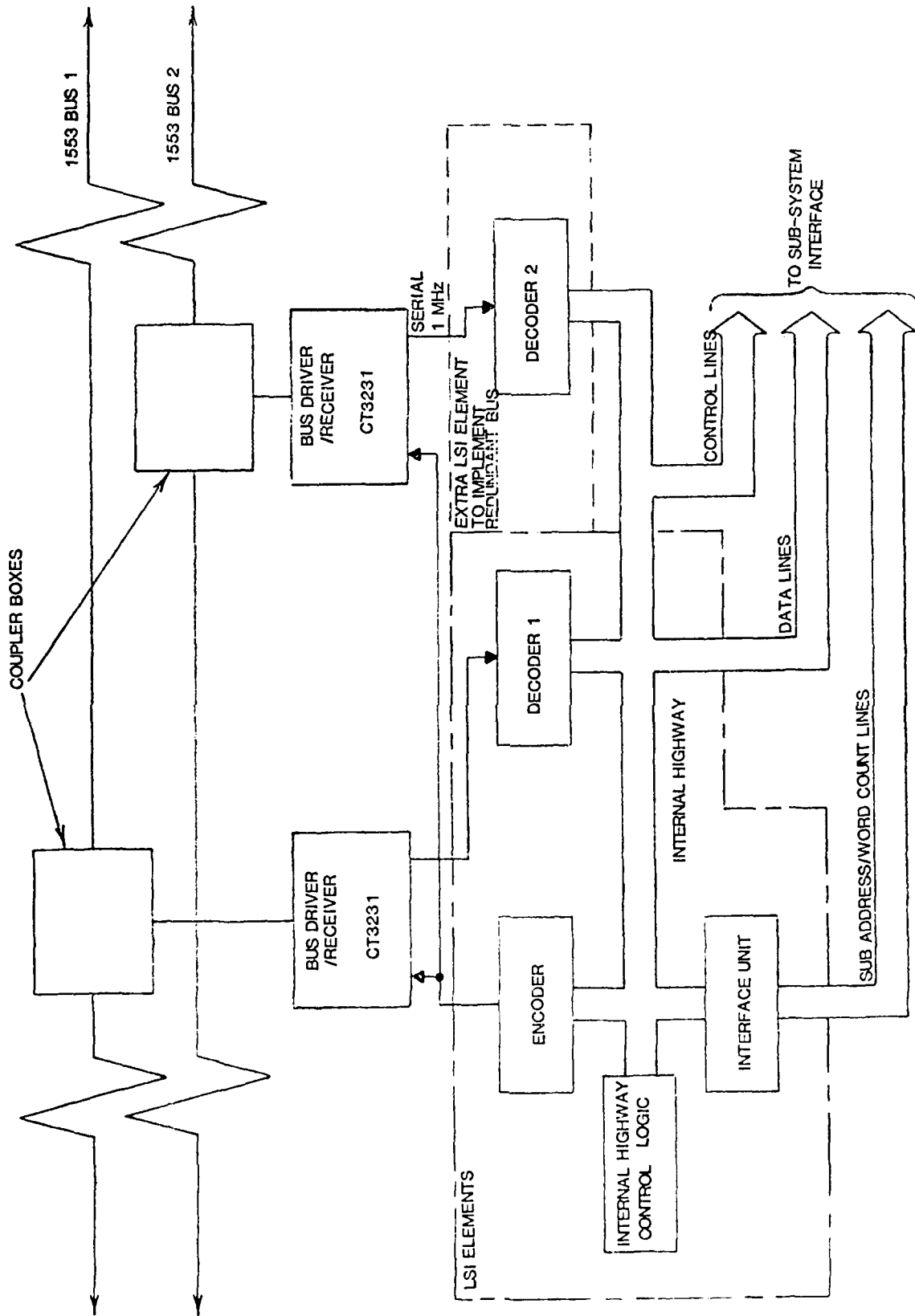


FIG.2 LSI ARCHITECTURE

SCADC
REMOTE TERMINAL MODULE
ENGINEERING PROTOTYPE
11 MAY 1982



WD-A142 776

PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND. (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O.
C A PORUBCANSKY NOV 82

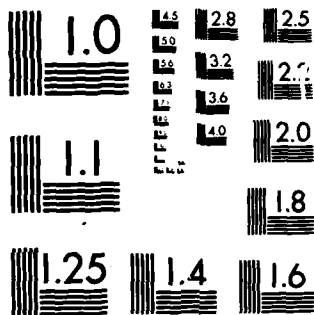
476

UNCLASSIFIED

F/G 9/3

NL

A microfiche card containing a grid of frames. The grid is approximately 13 frames wide and 12 frames high. The top row contains several white artifacts, likely from the original document or scanning process, located in the 5th, 6th, 7th, 8th, 9th, and 11th frames from the left.



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-P003 539

"MIL-STD-1553 INTERFACE
APPLICATION NOTES"

By Steve Friedman

ILC Data Device Corporation
(DDC)
105 Wilbur Place
Bohemia, New York 11716
516/567-5600

ABSTRACT

This paper describes Monolithic and Hybrid circuits used for interfacing to a MIL-STD-1553 Multiplex Data Bus. The focus is on how to implement smart and dumb remote terminals with existing devices. A menu table depicting product selection has also been provided.

The implementation of BUS CONTROLLER (BC), MONITOR and REMOTE TERMINAL UNIT (RTU) interfaces, will be fully described using standard off the shelf products. Custom Microprocessor Interface Hybrids with DMA or Double Buffered Memory Interface, will complete the picture for the implementation of MIL-STD-1553 A or B.

The MCE (Smith) LSI Chip Set for MIL-STD-1553B RTU and BC configurations will be described. Additional application topics covered will be the implementation of high speed data transfer circuitry, single, dual and quad redundant channel operation as well as implementing External status word controls.

A summary of product sizes and power requirements associated with BUS Controller, RTU and Monitor applications will be provided.

INTRODUCTION

The System designer no longer has the headache of tackling the detail design of a MIL-STD-1553 Interface. Whether the design engineer is considering Remote Terminal (RTU), BUS Controller or a Monitor MUX BUS Interface, implementation can be simplified with standard off the shelf products. These products are comprised of Monolithic LSI Chip Sets, hybrid packaged products or in some cases a combination of both.

One must be responsive to MIL-STD-1553A or programs such as F-16, B-520AS and YAH-64 that require a different Mode Code

response then that of MIL-STD-1553B. The McDonnell Douglas (McAir) (A-5690, A-3818, A-4905 and A-5232) sinusoidal waveform specifications must also be considered. To implement these variations, a full menu of off the shelf products are available (please refer to Table 1) to assist the Design Engineer.

The Menu of products offered consists of transformers, transceivers, Manchester II Converters, Protocol Logic Hybrids, I/O Hybrids and the MCE (Smith Industries) MIL-STD-1553B LSI Chip Set. The products can be characterized as follows in Table 1:

TABLE 1 - PRODUCT MENU

TRANSFORMERS

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-25679	Isolation - 1.4:1 & 2:1 Ratio's Plus Center Taps
BUS-27765	Isolation - 1:1 & 1.4:1 Ratio's Plus Center Taps
BUS-29192	Isolation - 2.3:1 & 3.2:1 Ratio's Plus Center Taps
BUS-29854	Isolation - 1.2:1 & 1.66:1 Ratio's Plus Center Taps

TRANSCEIVERS

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-63105	Monolithic 1553A/B Transceiver - Single and Dual Redundant 24 Pin DDIP and 36 Pin DDIP. OPTIONS: 5 = Standard ± 15 VDC PWR. (Uses BUS-25679) 6 = Thermal Shutdown. (Uses BUS-25679) 7 = ± 12 VDC PWR. (Uses BUS-29854 XFMR) 8 = ± 12 VDC PWR Plus Thermal Shutdown. (Uses BUS-29854 XFMR) OPTIONS: 0 = Single Channel - Interfaces with BUS-8937 and Harris HD-15530. 1 = Single Channel - Interfaces with MCE (Smith's) LSI Chip Set. 2 = Dual Redundant - 36 PIN DDIP - Interfaces with BUS-8937 and Harris HD-15530. 3 = Dual Redundant - 36 PIN DDIP - Interfaces with MCE (Smith's) LSI Chip Set.
BUS-63102	Universal McAir & 1553A/B Transceiver. Plug In. 24 PIN on 1.1" Centers. (Uses BUS-29192 XFMR)
BUS-8559	Variable Output (Driver) Transceiver. 24 PIN DDIP For ATE.

MANCHESTER II CONVERTERS

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-8937	HD-15530 + 19 MSI Chips With 16 BIT Parallel I/O 48 Pin Plug In
MT-32008	MCE (Smith's) Encoder/Decoder LSI Chip With 16 BIT Parallel I/O. 48 PIN DDIP
BUS-64100	HD-15530 & LSI Chip With 8/16 BIT Parallel or Serial I/O Terminal BIT Processor. 56 PIN Plug In.
BUS-1555	Decoder Module Which Flags Error Type. 52 PIN Plug In Module. For ATE.
BUS-1556*	Encoder Module Which Can Introduce Errors. 52 PIN Plug In Module. For ATE.

RTU INTERFACE (W/O PROTOCOL)

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-1553	Interface Module. 51 PIN Plug In
BUS-9253	2A Format B SEM Module. Key Coded "MAN". 100 PIN Edge Connector.
BUS-65101*	Single Channel 1553A/B Hybrid
BUS-65201	Single Channel McAir & 1553 A/B Hybrid. 78 PIN

PROTOCOL

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
MT32004	MCE (Smith's) LSI Protocol Sequencer. 64 PIN TDIP.
BUS-66101	Protocol 1 Hybrid. 36 PIN DDIP.
BUS-66102	Protocol 2 Hybrid. 68 PIN Plug In.

SMART RTU'S (W/PROTOCOL)

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-65400	Complete 1553B Dual Redundant Evaluation Board for MCE (Smith's) LSI Chip Set
BUS-65122	1553B Dual Redundant Super Hybrid (Uses BUS-63115 XCVR & MCE (Smith) LSI Chips). 78 PIN Plug In.
BUS-65112*	1553 A/B** Dual Redundant Hybrid. 78 PIN Plug In.
BUS-65212*	Universal Dual Redundant Hybrid. 78 PIN Plug In.

BUS CONTROLLER

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-65500	Double EURO Card Assembly. Dual Redundant BUS Controller with VME BUS Interface.

MEMORY OPTIONS

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
MT 32003	MCE (Smith's) 16 BIT X 32 Deep FIFO. 40 PIN DDIP.
BUS-66103	DMA with Double Buffered Memory Interface for the 68000 Microprocessor. 51 PIN Plug In.
BUS-66105	Dual Port Double Buffered Memory RAM Interface for the 8086 Microprocessor. 51 PIN Plug In.

MANUAL BUS EXERCISER

<u>PART NUMBER</u>	<u>DESCRIPTION</u>
BUS-68000*	MIL-STD-1553A/B Tester, IEEE Optional.

- * Products Soon To Be Available
- ** 5us Status Word Response

TRANSFORMERS

The isolation transformers listed in Table 1, "Product Menu", are manufactured by a subsidiary of DDC called BETA Transformer Corporation. They manufacture custom transformers plus the versatile pulse transformers that meet all of the electrical requirements of Manchester II, serial Bi-phase, data transmission. They are built in accordance with MIL-T-21038.

These transformers offer a high common mode rejection ratio (CMRR), good selection of turns ratios, frequency response and isolation necessary for accommodating all DDC MIL-STD-1553A/B and McAir Transceivers as well as competitive transceivers.

TRANSCEIVERS

The BUS-63105 MonobridTM is the first production single chip custom LSI (BI-Polar) transceiver. It features high reliability and low cost in a small 24 PIN DDIP (Please refer to Figure 1, BUS-63105) or dual channel transceiver in a 36 PIN DDIP (Please refer to Figure 2, BUS-63125) hybrid package. It can be used in any MIL-STD-1553A/B interface application.

MONOBRIDTM is a trademark of ILC Data Device Corporation.



FIGURE 1: BUS-63105 LSI TRANSCEIVER

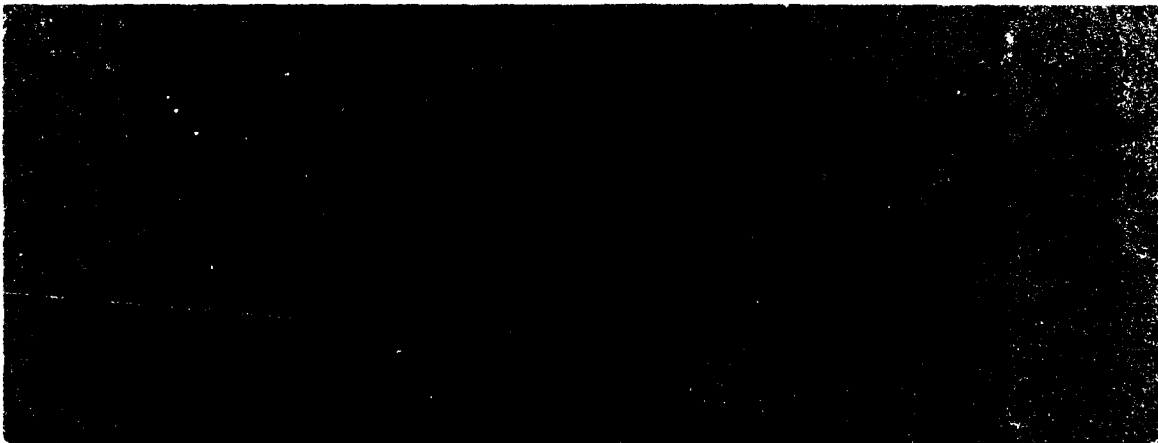


FIGURE 2: BUS-63125 DUAL REDUNDANT TRANSCEIVER

The BUS-63105 Series Transceiver features low power dissipation, improved receiver filtering, current limited driver output plus driver input time out protection. It is available in single and dual redundant configurations.

An optional +175 degree centigrade shutdown circuit with pin programmable override is available.

The BUS-63102 Universal Transceiver was designed to meet the McDonnell Douglas (McAir) A-5690, A-3818, A-4905, A-5232 sinusoidal waveforms and conform to MIL-STD-1553A/B trapezoidal waveforms. Please refer to Table 1 "Product Menu" and Figure 3, BUS-63102.

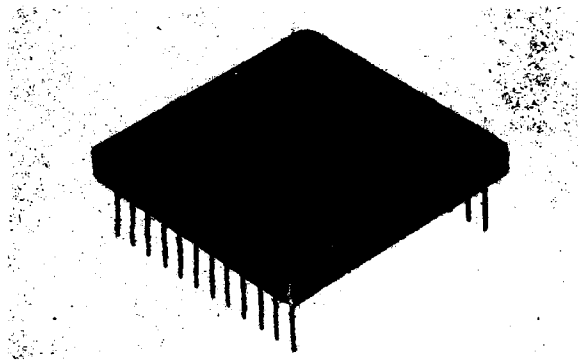


FIGURE 3: BUS-63102 UNIVERSAL BUS TRANSCEIVER

This transceiver is able to meet the waveforms and transmission group delays because of the special linear phase equiripple filter incorporated.

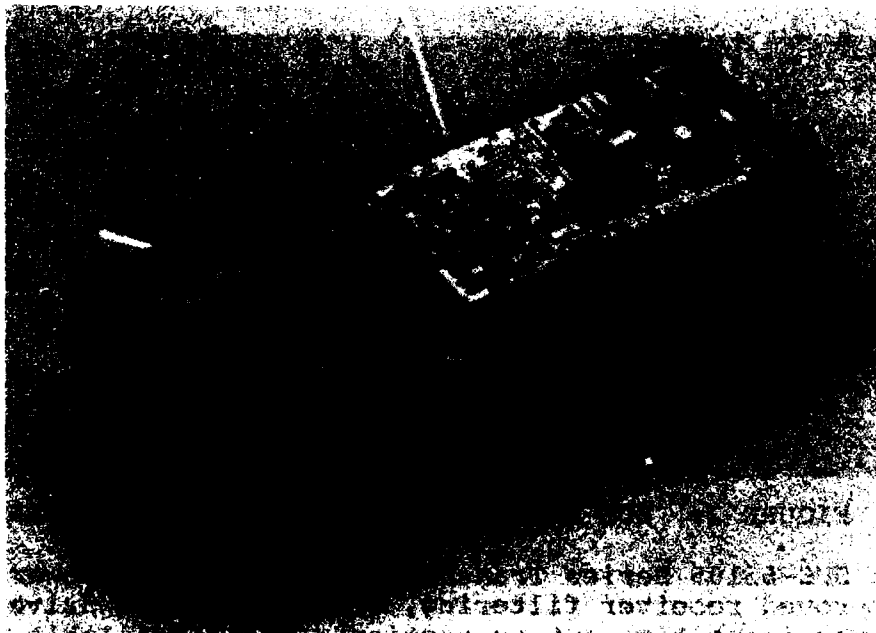


FIGURE 4: BUS-8559 & BUS-25679 Transceiver & Transformer

The BUS-8559 is a specially designed MIL-STD-1553A/B transceiver with a variable driver output. Please refer to Figure 4, BUS-8559 and BUS-25679 Isolation Transformer. This special feature is intended to be used in MUX BUS Simulators and test sets for the purpose of checking RTU receiver threshold operation.

MANCHESTER II CONVERTERS

Present Manchester II Converter products available are typically one of three types; they are based around the popular Harris HD-15530 Encoder/Decoder Chip, MCE (Smith's) LSI Encoder/Decoder Chip, and for special applications a separate module decoder and encoder with very special features.



FIGURE 5: BUS-8937 MANCHESTER II CONVERTER

The BUS-8937 Manchester II Converter (See Figure 5) and the BUS-64100 Terminal Bit Processor both use the HD-15530 CMOS Encoder/Decoder. They both feature three state, 16 Bit parallel interface, address recognition, encoder/decoder status and control lines. The BUS-64100 bit processor has additional features such as 16 or 8 bit Byte or serial I/O, self contained oscillator and clock driver, Broadcast Flag, Mode Code Flag, 800 microsecond time out, on/off line self-test plus low power due to its inherent LSI design.

The MT32008 MCE (Smith's) Encoder/Decoder LSI Chip can be used as a stand alone product. (Please refer to Figure 6, MT32008 Block Diagram). Its ISO-CMOS process gives it better radiation resistance than the HD-15530, lower power dissipation and with one more feature than the BUS-8937, namely Broadcast Recognition.

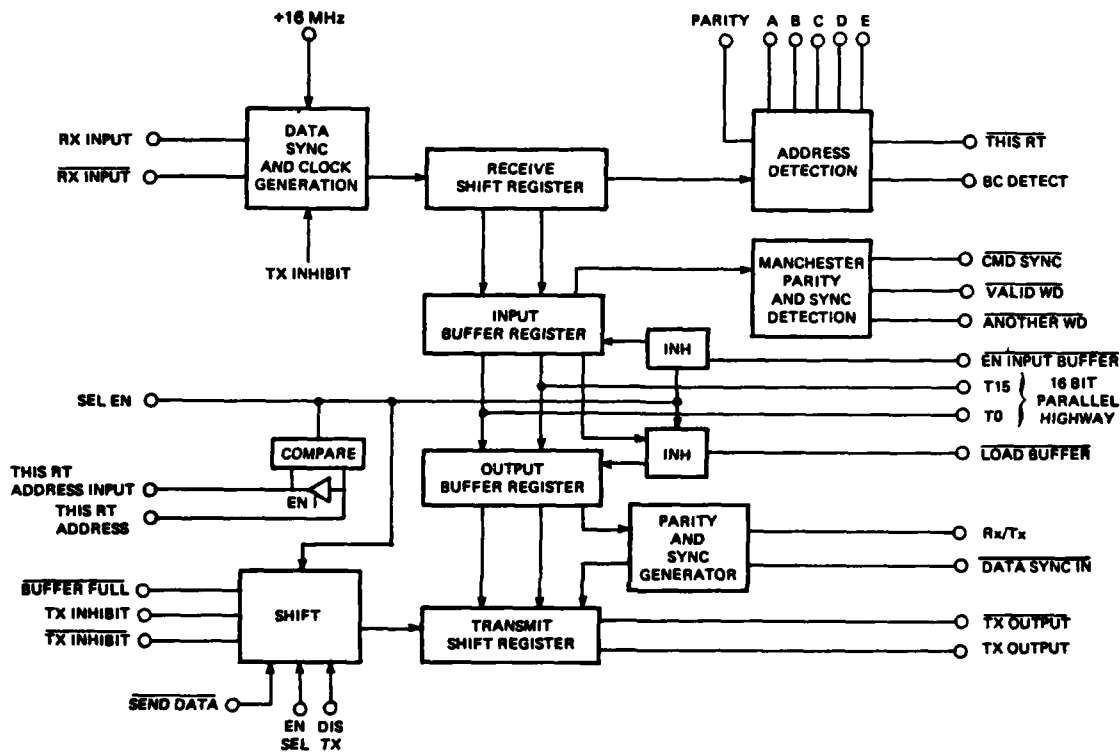


FIGURE 6: MT32008 BLOCK DIAGRAM

The BUS-1555 Decoder Module was designed for test equipment applications since it will flag sync errors, Manchester II errors, hi & lo bit counts and parity error. Its counter part is the BUS-1556 Encoder Module which will generate the various errors. These two products, plus the BUS-8559 Variable Output Transceiver, can be used in a dedicated simulator or heart of a MIL-STD-1553A/B tester such as the BUS-68000.

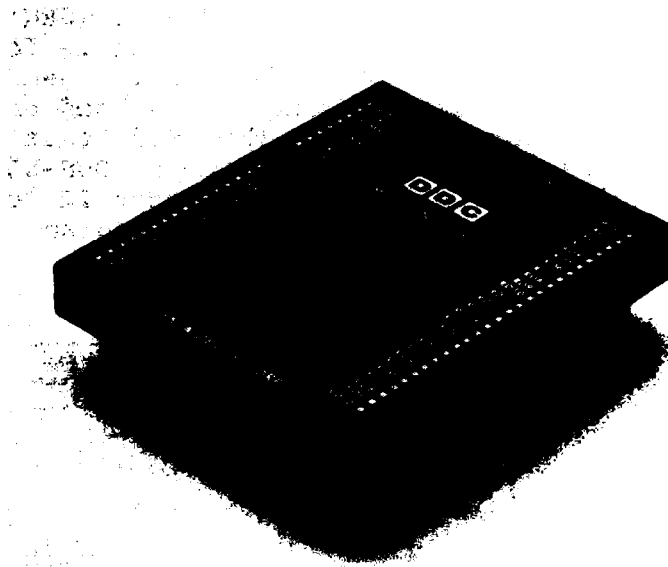


FIGURE 7: BUS-1553 INTERFACE MODULE

RTU INTERFACE (W/O PROTOCOL)

The BUS-1553 Interface Module (See Figure 7) is comprised of the isolation resistors, BUS-25679 Transformer, BUS-8553 Transceiver, HD-15530 Ceramic DIP, Hybrid 12MHZ Clock, control logic lines and shift registers necessary for one 16 bit parallel data input port and one 16 bit parallel data output port. It is ideal for breadboarding or applications with reduced temperature extremes.

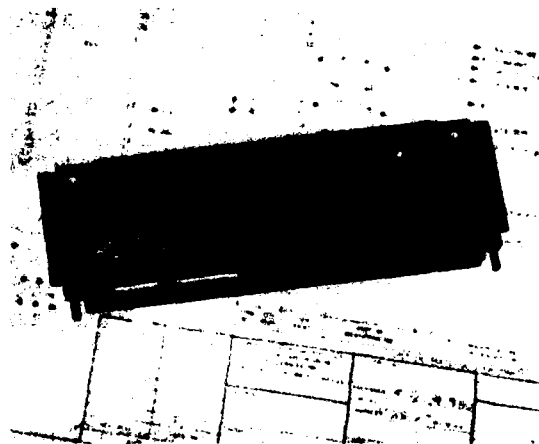


FIGURE 8: BUS-9253 SEM MODULE

The BUS-9253 Standard Electronic Module (SEM) is composed of standard hybrids, thereby making it suitable for full MIL environmental applications. This SEM Module (See Figure 8) is key coded "MAN", and is presently being used on the Rolling Air Missile Program for the Navy. This Module is composed of isolation resistors, BUS-25679 Transformer, BUS-8553 Transceiver, hybrid clock and BUS-8937 Manchester II Converter. Please refer to Figure 9, BUS-9253 Block Diagram.

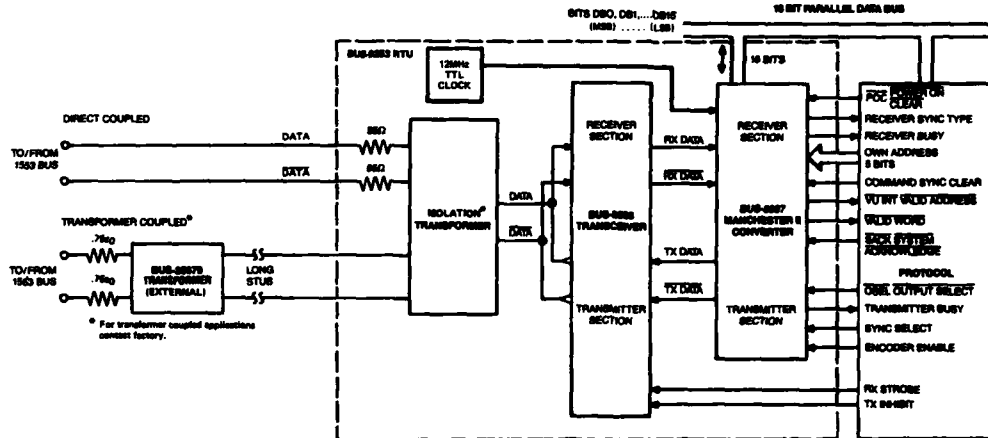


FIGURE 9: BUS-9253 BLOCK DIAGRAM

The BUS-9253 represents the basic building block of components useful for a single MIL-STD-1553A/B RTU Interface. When addressing a redundant interface, a duplication of these components would be necessary.

Should the application involve interface to the McAir specifications the BUS-63105 Transceiver would be changed to that of the Universal BUS-63102 Transceiver and BUS-29192 Transformer.

If space is limited, DDC has a single channel McAir and 1553A/B interface hybrid (BUS-65201) in a 1 X 2 inch 78 Pin Plug In package. This hybrid is based on the BUS-63102 transceiver and BUS-64100 LSI Terminal Bit Processor.

SMART RTU'S (W/PROTOCOL)

The system designer need not concern himself with the problems of McAir or MIL-STD-1553 implementation, as a full array of products are now available. These interfaces, unlike those w/o protocol previously discussed, are available with full response to Mode Codes, Broadcast Commands, RTU Address/Sub-

address, required status word response and specific micro-processor interface.



FIGURE 10: MIL-STD-1553B DUAL REDUNDANT MCE (SMITH'S)

The MCE (Smith's) MIL-STD-1553B LSI Chip Set is a fully functional Smart RTU. It consists of a MT32008 Manchester II Converter, MT32004 Protocol Sequencer and MT32003 FIFO Memory Chip. The three chips, plus the BUS-63115 Transceiver and BUS-25679 Transformer make up a complete single channel RTU. A dual redundant configuration is shown in Figure 10. Notice it uses two transformers, two transceivers, two MT32008's then one MT-32004 sequencer and one MT32003 FIFO. The MCE (Smith's) LSI Chip Set can be used to implement up to quad redundant interfaces.

DDC offers an MCE (Smith's) Dual Redundant 1553B Evaluation Board. It consists of the complete LSI Chip Set Transceivers, 16MHZ clock and two independent 800 microsecond time out circuits. All mounted on a 7 X 5 inch Printed Circuit Board.

The MCE (Smith's) LSI Chip Set features very low power dissipation for a MIL-STD-1553B application. The Chip Set is designed to implement all fifteen defined mode codes and can generate and respond with a status word in just under 11 microseconds. It makes use of a 16 Bit, 32 Word first-in-first-out (FIFO) Memory Chip which facilitates the transfer of decoded data to the Subsystem Interface Unit (SSIU) or for loading data in preparation for transmission on the MUX BUS. In the case of

transferring decoded 16 Bit data to the SSIU, a high speed transfer circuit (See Figure 11) must be implemented in order to fully comply with 1553B. This circuit can be omitted if one wishes to substitute the MT32003 with external memory capacity or simply by paralleling the FIFO with external memory.

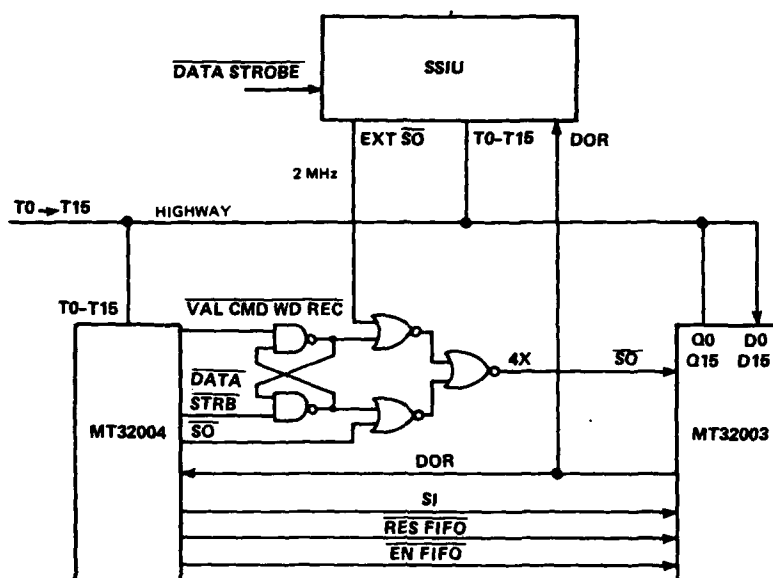


FIGURE 11: HIGH SPEED TRANSFER CIRCUIT

The LSI Chip Set offers a three state 16 Bit Parallel highway interface plus necessary control lines for subsystem interface. Additional circuitry would be required to implement DMA data transfers in either 8 or 16 Bit Bytes.

External status word control is possible with a minimum of external hardware. Any of the status word bits or an entire status word can be introduced externally simply by enabling bi-directional three state buffered latches with the Send Data-BAR signal. The latches will open the specific 16 bit parallel highway bit(s) from that of the MT32004 Sequencer, so that the subsystem can introduce the required logic state(s).

In an effort to reduce the price and size of a MIL-STD-1553B Dual Redundant RTU's, DDC has introduced the Superhybrid. The Superhybrid (BUS-65122) is a 78 PIN Plug In 2 X 2 inch hybrid package that has two transceivers plus 800 microsecond time outs, two MT32008 die, one MT32004 die, one MT32003 die, high speed transfer circuitry plus logic for "on" and "off" line wrap around testing. This can be considered the ultimate in

small size and low power dissipation for a dual redundant RTU.

The Superhybrid as well as the MCE (Smith) LSI Chips, can be used in BUS Monitor applications with a minimum of circuitry. It can be used in BUS Controller applications because of the MT32008 Manchester II Converter Chips accessible control lines. (Please refer back to Figure 6, MT32008 Block Diagram).

There are many existing programs and applications that differ from MIL-STD-1553B. These differences can be electrical such as McAir (waveforms) or Mode Code protocol implementation differences. It is for this reason BUS-66101 (Protocol 1) and BUS-66102 (Protocol 2) were developed. These two hybrids, plus the BUS-64101 (Modified BUS-8937), BUS-63105 or BUS-63102 Transceiver and Isolation Transformer form a fully operational MIL-STD-1553B or McAir (except A-3818) RTU.

Protocol 1 and Protocol 2 hybrids can be programmed by external ROM to implement the specific Mode Codes required for the application. This permits the system designer the flexibility for using the same hardware in different applications.

The Protocol Hybrids perform functions such as internal time outs, word counting, sequencing of data transfers-Mode Codes-Status response for a dual redundant RTU. The Status Word Response of this hybrid set is nominally 7 microseconds, which just falls outside of the MIL-STD-1553A specification. Please refer to Figure 12, Hybrid RTU Functional Block Diagram.

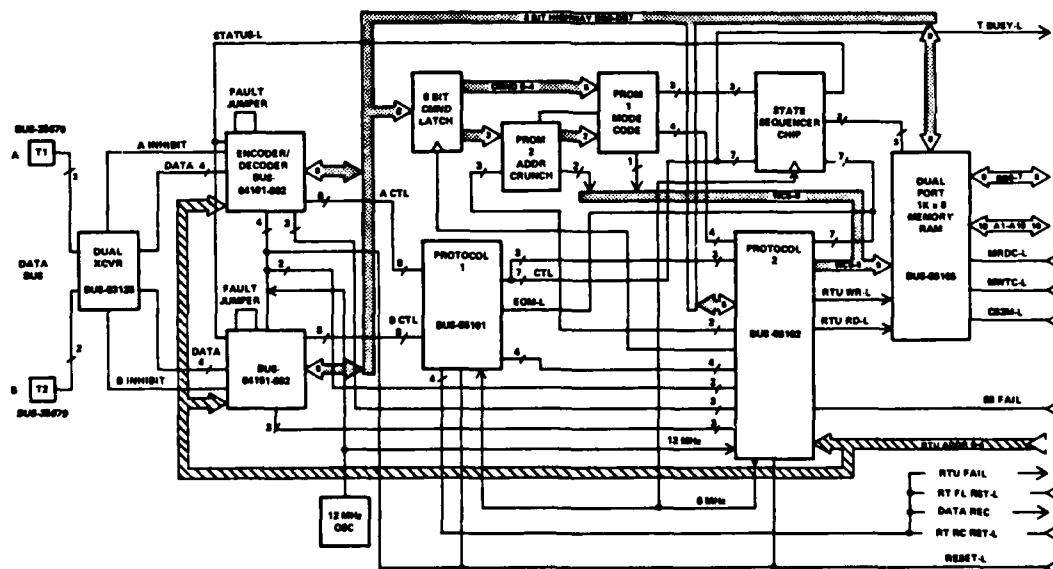


FIGURE 12: RTU BLOCK DIAGRAM

Two completely new dual redundant MIL-STD-1553A/B and respectively McAir Hybrids each in a 2 X 2 inch, 78 Pin, Plug In package, will be available soon.

The BUS-65112, is a dual redundant MIL-STD-1553A or B responsive hybrid RTU which features fully programmable mode code implementation and a status word response time of 5 microseconds.

The BUS-65212, is a duplicate of the aforementioned with the exception being that the front end consists of the Universal Transceiver. This facilitates meeting all of the McAir specifications, due to the 5 microsecond status word response.

These complete RTU/Monitor Functioning Hybrids will interface with the Memory Options presently available.

BUS-CONTROLLER

The BUS-65500, is a Double Eurocard (6.3 X 9.2 inches) assembly which meets the requirements for Bus Controller Interface Unit (BCIU) or Remote Terminal Unit (RTU) to a dual redundant MIL-STD-1553B Multiplex BUS and the 68000 VME BUS. This interface unit is designed to interface to a 68000 Microprocessor with a VME BUS using DMA transfers and programmed I/O transfers. Please refer to Figure 13, Hybrid BCIU Block Diagram.

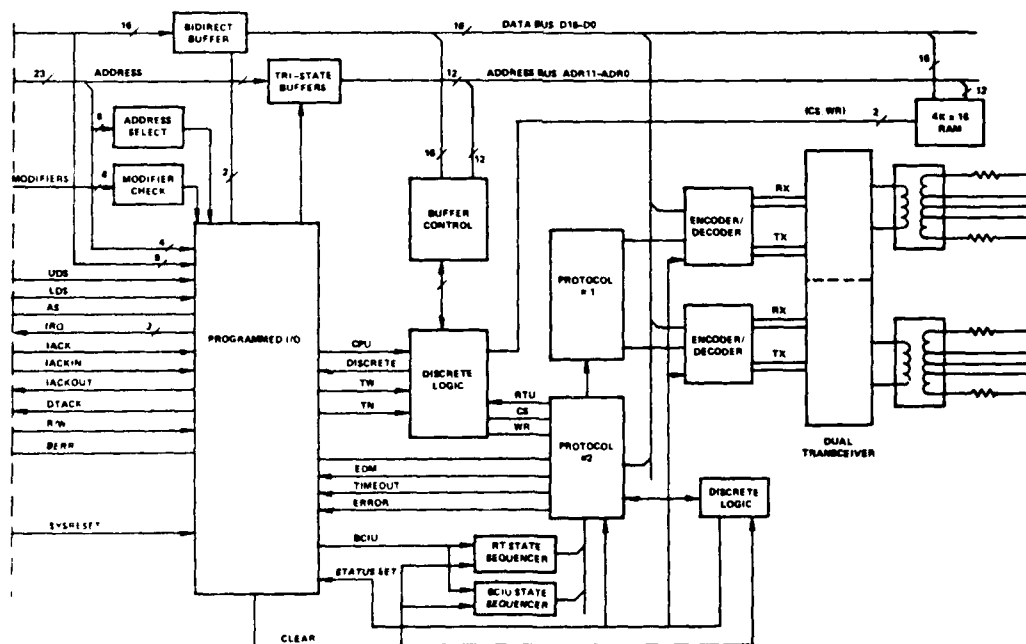


FIGURE 13: BCIU BLOCK DIAGRAM, BUS-65500

The BUS-65500 (BCIU) is composed of (2) BUS-25679 Transformers, (2) BUS-63105 Transceivers (2) BUS-64101 (Modified BUS-8937), BUS-66100 Series Special Protocol Hybrids, (1) BUS-66103 DMA with double buffered memory hybrid interface and a special 68000 Motorola VME Bus Interface Hybrid.

The Bus Controller function can be applied to McAir and to other standard interfaces by selecting the proper transceiver and presently available memory options. The special Protocol Hybrids can be programmed through external ROM's for specification differences.

MEMORY OPTIONS

The BCIU and RTU Product Menus described so far would only permit a 16 Bit or 8 Bit parallel Byte or serial data transfer to take place on demand. The exception being the MCE (Smith's) LSI Chip Set, which is supported by an integral FIFO Memory Chip as previously discussed.

The system designer may now select off-the-shelf hybrids for supporting an interface with the Motorola 68000 and Intel 8086 Microprocessors. These two products represent the most popular of I/O support hybrids available. (Consult factory for other interfaces.)

A Motorola 68000 DMA with Double Buffered Memory Interface is provided with the BUS-66103 hybrid. This Memory Option directly supports the BUS-66101 and BUS-66102 Protocol Hybrids and adds the following features:

- 1) High speed transfer using a DMA transfer for each data word. (Using 16 Bit Bytes or optional 8 Bit Bytes.)
- 2) Data blocks are organized on a subaddress basis.
- 3) Double buffering for each data block, and switching between the two buffers on an individual block basis.
- 4) Circular message area provides stacking up to 32 messages.
- 5) Buffer size for each system determined by the software.
- 6) Up to 64 different data blocks can be accommodated.

The BUS-66105 provides a Dual Port Double Buffered Memory RAM Interface for the 8086 Microprocessor. It too supports BUS-66101 and BUS-66102 Protocol Hybrids with features such as:

- 1) An internal 1K X 8 RAM for data buffering. (8 Bit Bytes used)

- 2) Provides six double buffered data blocks plus one wrap-around data block buffer, with switching between the two buffers on an individual block basis.
- 3) Message area reserved for stacking up to 6 different messages.

These off-the-shelf Memory Interface Hybrids support 8 and 16 Bit Microprocessor Systems. The Microprocessors need not be burdened with interrupts for storing and transferring data. Therefore, overall system speed is achieved.

MANUAL BUS EXERCISER

The BUS-68000 exerciser is intended to be ancillary to more powerful MUX BUS Analyzers, as an example Conic's SBA-100, and not their replacement. The BUS-68000 Exerciser can be used for testing and troubleshooting MIL-STD-1553B and McAir Systems manually, thereby off loading less critical tasks from the SBA-100. Its versatility makes it very attractive and its low cost makes it possible to have several available to off load the SBA.

The BUS-1555 Decoder, BUS-1556 Encoder and BUS 8559 Transceiver incorporated will allow the design engineer to test and troubleshoot both hardware and software. It has the ability to identify high or low bit counts, parity errors and Manchester II errors in the receive mode and to generate Manchester, parity, bit count, word count and sync errors.

The instrument will have keyboard entry and microprocessor control through a 8 bit (2 byte) three state I/O with all necessary hand shakes. IEEE 488 BUS I/O is optional while RS232 will be standard.

CONCLUSION

The design engineer who must interface his system to MIL-STD-1553 or McAir, can turn to a standard Monolithic or Hybrid product Menu. The RTU, Monitor and BUS Controller functions can easily be handled with combinations of a variety of different products. These products vary in power requirements, size and functional flexibility.

In order to highlight product implementation differences, Table 2, Dual Redundant BCIU/RTU MENU, was prepared with the most popular examples.

TABLE 2

"DUAL REDUNDANT BCIU/RTU MENU"

MCE (SMITH'S) SELECTIONS

<u>TRANSCEIVERS</u>	<u>ENC/DEC</u>	<u>PROTOCOL</u>	<u>I/O</u>	<u>SQ.IN.</u>	<u>WATTS</u>
A) (2)BUS-63115	(2)MT32008	(1)MT32004	(1)MT32003	6.6	2.1
B) (1)BUS-63135	(2)MT32008	(1)MT32004	(1)MT32003	5.8	2.1
C) (1)BUS-63135	(2)MT32008	-(1)MT32007-		4.6	2.1
D) BUS-65122	SUPER HYBRID			4.0	2.2

HYBRID RTU SETS

<u>TRANSCEIVERS</u>	<u>ENC/DEC</u>	<u>PROTOCOL</u>	<u>I/O</u>	<u>SQ.IN.</u>	<u>WATTS</u>
A) (2)BUS-63105	(2)BUS64101	(1)BUS66101	(1)BUS66103	14.5	8.0
		(1)BUS66102			
B) (1)BUS-63125	(2)BUS64101	(1)BUS66101	(1)BUS66103	13.7	8.0
		(1)BUS66102			
C) (2)BUS-63102 (McAir)	(2)BUS64101	(1)BUS66101	(1)BUS66103	15.3	8.2
		(1)BUS66102			
D) (1)BUS-63125	(2)BUS64101	(1)BUS66101	(1)BUS66105	16.2	8.0
		(1)BUS66102			

BCIU/RTU SET

<u>TRANSCEIVERS</u>	<u>ENC/DEC</u>	<u>PROTOCOL</u>	<u>I/O</u>	<u>SQ.IN.</u>	<u>WATTS</u>
A) BUS-65500	HYBRID SET COMPRISED OF:				
(1)BUS-63125	(2)BUS64101	(2)BUS66101	(1)BUS66103	20.0	10.0
		SERIES			

BIOGRAPHY

Steven N. Friedman, is presently a Senior Applications Engineer for Data Bus Products at ILC Data Device Corporation. Formerly, an Applications Engineer in the systems Marketing group.

Former positions include being a Senior RF Staff Engineer with Fairchild Camera, Space and Defense, in Syosset, New York.

As Chief Engineer at Modular Devices Inc., he directed Engineering programs, including a new Microelectronic Hybrid Department.

At Robins Industries, Steve held the position of Vice President of the Professional Products Division. This consisted of Robins Broadcast and Sound Equipment Corporation (formerly Fairchild Sound). He actively marketed and managed this division as well as directing the engineering programs for Robins Industries.

Steve received his Master of Science Degree in 1978, from C.W. Post College of Long Island University at Greenvale, New York, in Management Engineering. He also holds a 1970 Bachelor of Science degree from New York Institute of Technology in old Westbury, in Electrical Engineering.



APPLICATION OF 1553B TO MRASM-A SYSTEMS LOOK

John E. Leib
General Dynamics Convair Division
P.O. Box 80847
San Diego, California 92138
(714) 277-8900, x2101

ABSTRACT

(Medium Range Air to Surface Missile)

→ MRASM is the first missile to incorporate a MIL-STD-1553B data bus as the primary means of data transfer among the elements of the missile. The Standard, built around applications which could dedicate major computing power to manage the affairs of the data bus, posed a challenge to MRASM because this bus management function needed to be performed on the input/output card which fit in an existing computer design, while not utilizing its host computer on a continuing basis.

This paper reviews the process by which the data bus operation was defined, describes the protocol adopted for timely transfer of data, and argues the case for the system design decisions.

INTRODUCTION

The Medium Range Air to Surface Missile (MRASM) is an adaptation of Tomahawk for air carry and launch. For use by both the Air Force and the Navy, the MRASM must accommodate various configurations of payloads and avionics. A MIL-STD-1553B data bus was selected for data transfer among the many computers in the system, to aid in easy integration through use of a standard, widely used and understood technique.

Applying MIL-STD-1553B to this missile was a "first", so there was little history to guide the system designer as he groped for the proper definition to best serve the MRASM program. Severe space, weight and power restrictions made the luxury of committing large amounts of computing capability to support the operation of the data bus an unlikely solution; and some of the terminals were required to interface with existing computer designs which had been selected for use on MRASM.

Latency of some data could be critical, as MRASM is basically an unstable vehicle during some portions of flight, and requires a tight autopilot loop. And, as always early in the definition of a data bus, bus loading, or duty cycle was of concern.

The processes and decisions which resulted in defining the requirements for the MRASM internal MIL-STD-1553B data bus follows.

1.0 MRASM Hardware Configuration

The basic MRASM has five boxes (Figure 1) which communicate with each other over the data bus. These are:

- MCM - Mission Control Module
- GNC - Guidance and Navigation Computer
- ISA - Inertial Sensor Assembly
- DPU - DSMAC Processor Unit
- SMC - Stores Management Controller

In addition, on test flights there is a

- TIC - Test Instrumentation Controller

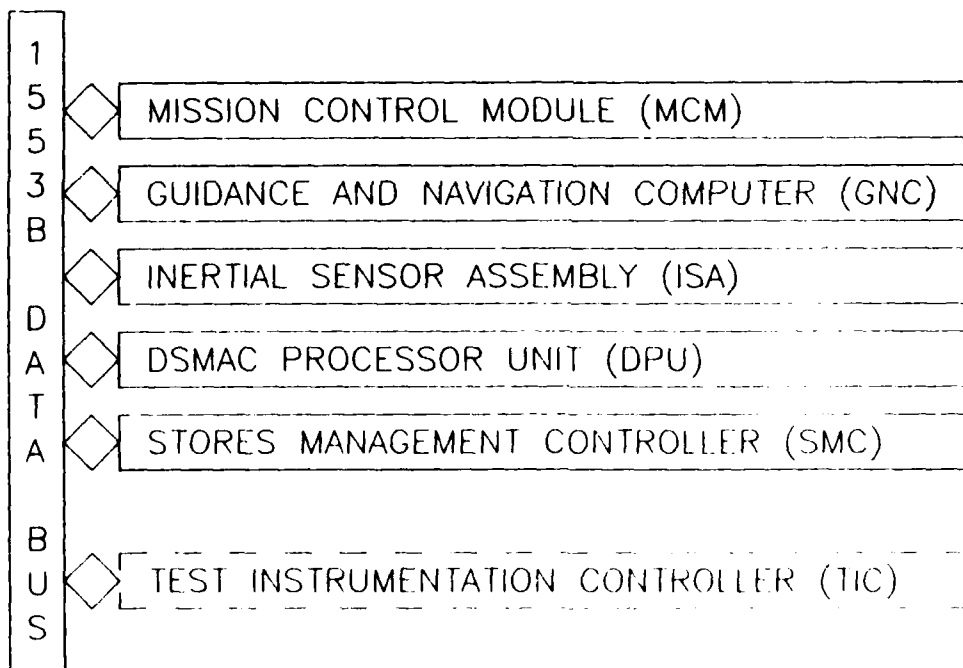


Figure 1. MRASM Data Bus Configuration

The MCM performs the sequencing and autopilot functions, and is a Digital Integrating Subsystem (DIS) computer. It receives guidance and throttle commands from the GNC and inertial data from the ISA, and provides air data to the GNC, DSMAC scene data to the DPU, and payload dispensing data to the SMC.

The GNC performs guidance and navigation computations, and is also a DIS computer. It receives inertial data from the ISA, air data from the MCM, and update data from the DPU; and provides guidance and throttle commands to the MCM.

The ISA contains gyros and accelerometers, and measures its inertial environment and makes some calculations to perform the "gimbal" functions. It sends autopilot data to the MCM, navigation data to the GNC, and attitude data to the DPU.

The DPU processes camera data and compares it with stored scenes to determine where it is. It receives the stored scenes from the MCM and sends its position match data to the GNC.

The SMC controls the dispensing of airfield attack payload submunitions. (For other payloads, it may not be aboard.) It receives payload dispensing commands from the MCM. It does not transmit any messages.

All of this interchange of data is carried out on a MIL-STD-1553B data bus. (Some other data moves on other mediums, but none of it is "computer to computer".) So the first question to be answered was, "By what specific protocol was this data to be made to move?". Constraints on the answer included the fact that software for the various computers was being developed by several independent entities and, while there is some rough synchronization among the computing activities, the precise time a computer would have a message ready for transmission was not known to any other computer. Also, the press of other activities made it unlikely that any of these computers could take on the additional task of truly managing the affairs of the bus in a real-time sense: the bus interfacing hardware must also perform this function.

On test flights, the TIC, also a DIS computer, collects and formats data for telemetry. Most of this is extracted from traffic on the data bus, the TIC acting as a monitor terminal. This aspect was not a factor in considering the bus protocol, but did contribute its share of requirements for the hardware implementation of the bus.

2.0 Protocol Options Evaluated

Within MIL-STD-1553B, three protocol options were identified as candidates for implementation in MRASM. These were called (1) Command/Response, (2) Passing Protocol, and (3) Poll for Transmission, and were considered to be the only technically viable possibilities within the Military Standard.

2.1 Command/Response

This protocol might be viewed as basic MIL-STD-1553B. The Bus Controller must know the schedule by which each message is prepared to be sent and appropriately command the transmission and reception of that message. (The word "message", as used in this paper, means a group of data words carrying functional information among the computers, and not a group of Command, Data and Status words, as defined in the Standard.)

2.2 Passing Protocol

Using this protocol, each terminal which may have a message to transmit is placed in the sequence of Bus Controllers, and control is passed in this sequence using the Dynamic Bus Control mode code command. Upon being designated "Bus Controller", a terminal transmits a message if one is ready, then passes control to the next terminal in the sequence. If no message is ready, control is passed immediately.

2.3 Poll for Transmission

This protocol requires a sequence of terminals to be known to the Bus Controller. (Any terminal may appear more than once in the sequence if higher frequency access is required.) The Bus Controller "polls" the terminals in this sequence for messages ready for transmission. If a terminal notifies the Bus Controller of a ready message, the Bus Controller provides the necessary command words to cause that message to be both transmitted and received. Of course, the Bus Controller must also be in this sequence and its messages are output during its turn.

2.4 Evaluation Arguments and Selection

MRASM being a tactical weapon which will be built by the thousands, total vehicle cost was a major consideration. Ease of integration among several contractors' equipments, and flexibility to add and delete equipment with minimum impact were also significant selection factors. While technical adequacy was mandatory, this did not appear as a serious threat to any of these candidates and was not a discriminator in the selection.

2.4.1 Evaluation of Command/Response

The advantage offered by the Command/Response protocol is that it is the most applied, and therefore the most familiar use of MIL-STD-1553B. The Bus Controller must know what is happening in all the computers on the bus in enough detail to know when a new message has been constructed and placed in an output buffer, how to address the output buffer, which terminal or terminals should receive this message, and how to direct the message to the proper input buffer or buffers. A significant, dedicated computational capability is required to support the Bus Controller in managing the data transfer on the bus, and tight synchronization among computers is required to avoid large latencies.

Because of this intimate, all-knowing involvement with every message on the bus, the Bus Controller must be altered in some way whenever there is an addition, deletion or change of any message.

MRASM is a system built from components from many contractors, and the integration problem for Command/Response protocol would be formidable, indeed. Not only would each message need to be agreed to by the sender and receiver, but much accurate information would also need to be incorporated into the operation of the Bus Controller.

2.4.2 Evaluation of Passing Protocol

The integration process becomes easier with Passing Protocol than with Command/Response. This is achieved because the messages and the protocol are decoupled to a great extent. No synchronization requirements are imposed simply to make the protocol work. Each computer determines when it has a message for transmission, then sends it when it is ready to send it. No other terminal needs to know when this will be. Additions, deletions and changes of messages are negotiated between the sender and receiver with no impact on the bus management function.

The bus management function at each terminal does need to know the next terminal in the sequence so that control can be passed properly. This imposes some attention to detail when terminals are either added to, or removed from the sequence, but this represents a major change in the vehicle configuration compared to changing or restructuring the messages on the bus, anyway.

A disadvantage of this protocol is that every terminal must be capable of becoming the Bus Controller, with its attendant added complexity.

2.4.3 Evaluation of Poll for Transmission

The Poll for Transmission protocol combines all of the benefits of Passing Protocol with the added benefit of allowing all but one terminal to be a Remote Terminal. The only bus-management-peculiar data required by the Bus Controller is the polling sequence.

The disadvantage of this protocol is that there is added protocol which increases bus loading. The significance of this, or lack of significance, is determined by the application. For MRASM, this was not considered to be very important.

2.4.4 Protocol Selection

For MRASM, the Poll for Transmission protocol was selected because it minimizes total program cost, integration complexity and attendant problems, and the impact of message additions, deletions and changes.

The specific protocol calls for the Bus Controller to "poll" each Remote Terminal in the selected sequence by addressing a Transmit Vector Word mode code command to that Remote Terminal. The Remote Terminal responds with a Status Word and a Vector Word, in accordance with MIL-STD-1553B. If that terminal has a message to be transmitted, the Service Request bit in the Status Word is set, with the data in the Vector Word supplying the information needed by the Bus Controller to cause the transmission and reception of the Remote Terminal's message. If the Service Request bit is not set, the Vector Word is ignored.

Figure 2 shows this process for an RT-to-RT message transfer. For RT-to-BC and RT-to-Broadcast transfer, the protocol following the Vector Word is adjusted in accordance with the Standard.

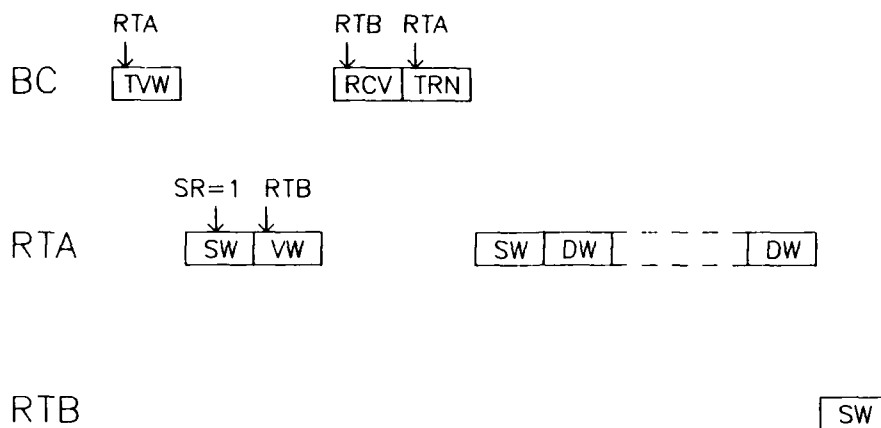


Figure 2. Poll for Transmission Sequence

When the Bus Controller has a message to send, it waits for its turn in the polling sequence, then issues a receive command followed by the message, also in accordance with the Standard.

3.0 Bus Implementation Detailed Requirements

Since two of the MRASM computers are from the DIS family (and a third computer on test flights), there was a requirement to supply one computer with a MIL-STD-1553B input/output channel which would act as a Bus Controller, and one computer with a Remote Terminal, each to fit into the standard DIS I/O slot. The third, flight test computer needed a Monitor Terminal. The detailed requirements included these facts, and the challenge for the I/O card designer was further elevated by the requirement that a single hardware/firmware design would act as all three, the specific type of terminal being selected by software in the host computer.

Several options are provided in the Standard, and from these options were selected the requirements for the specific design, as viewed from the bus. The other sets of requirements were dictated by the need for compatibility with the existing I/O card slots in the DIS computers, and by MRASM system considerations.

3.1 Bus Oriented Requirements

The requirements of MIL-STD-1553B were imposed on the card design. Options and alternative selections permitted by the Standard are described here.

3.1.1 Subaddress/Mode Field of Command Words

The Subaddress/Mode field is used in the subaddress application with the first bit (MSB) of the field set to one and the second bit set to zero. The third bit is used to indicate a "priority" message when it is set to one. In DIS computers, priority messages are handled entirely by the operating system. The fourth and fifth bits are set to zero for directed messages, and used to indicate the "broadcast group" to which a broadcast message belongs. Each DIS computer will receive one or more broadcast groups if it receives any broadcast messages.

For mode codes, only "1111" is used. Thus the first bit in this field will always be set to one, and may be used to distinguish command words from status words, which have a zero in this location (the "Instrumentation Bit").

3.1.2 Mode Codes

The only mode code required to be implemented is the Transmit Vector Word mode code. In response to this mode code command, a Remote Terminal will indicate the availability of a message by setting the Service Request bit in its Status Word to one, and providing a vector word which indicates, in Command Word format, which terminal or terminals the message is for, whether it is a priority message, and how many data words the message contains. Except for messages directed to the Bus Controller, the Bus Controller places the vector word on the bus with command word sync (it will be a "receive" command), followed by a "transmit" command directed to the "polled" terminal, with the last ten bits identical to the corresponding bits in the Vector Word.

All other mode codes are not used in the MRASM protocol and their implementation is optional.

3.1.3 Cable Stub Requirement

The requirements for direct coupled stubs, as described in 4.5.1.5.2 of the Standard, apply to the card design.

3.2 DIS Host Computer Oriented Requirements

In addition to interfacing with the data bus, the card must interface with the host computer. These are the major requirements imposed on the card by this interface.

3.2.1 Dimensions and Form Factor

To fit into a DIS computer, the card must be designed on one side of a DIS standard printed circuit board with a compatible, 70-pin connector. The major implication of this is that there are only about twenty square inches on which to fit the components.

3.2.2 Power

For the survival of the card and the DIS computer, the average power is limited to 10.5 watts, with peaks of no more than 13 watts.

3.3 MRASM System Oriented Requirements

MRASM system requirements were leveled on the card in great numbers. These requirements were those which would be necessary to make the system work. The major ones which drove the design are described here.

3.3.1 Card Reconfiguration

As the card is to perform as a Bus Controller, a Remote Terminal, or a Monitor Terminal, selectable at will by the host computer, a method for providing this information was developed and defined. Upon a signal from the host computer, the card extracts this information from the host computer memory, using its direct memory access channel. This information provides everything necessary for the card to know how to properly perform. Called the "reconfiguration message", it is composed of twenty 16-bit words.

3.3.2 Input Procedures

A problem sometimes encountered in a data bus implementation is overwriting data before it has been completely processed by the receiving computer. A similar problem occurs when two or more messages arrive before the first interrupt announcing their arrival can be honored by the host computer, and all but the last of the messages are lost.

For MRASM, messages are stored in sequentially-identified buffers in memory. This allows the host computer to handle all incoming messages at its own pace.

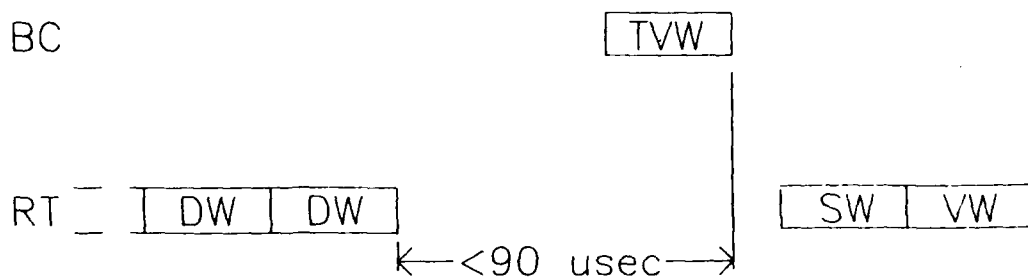
3.3.3 Message Retries

The requirement to assure the correct transmission and reception of every message falls to the Bus Controller. If the Bus Controller determines this has not occurred, it will initiate a "retry", and will continue to do so until it determines the message has been successfully transferred or it has initiated the number of retries called for in the reconfiguration message.

Remote Terminals must be prepared to support these retries, both as senders and receivers. A retry is commanded by issuing a repeat of the previous Transmit Vector Word mode code command within 90 microseconds after the final data word of a message has been placed on the bus (Figure 3).

3.3.4 Reliability

The MRASM requirement for card reliability at 75°C is 45000 hours MTBF.



A "Transmit Vector Word" mode code command received less than 90 u-sec after last data word is transmitted initiates a "Retry" of same message

Figure 3. Hardware Retries

4.0 Bus Performance

Concern for bus loading and data latency has resulted in close monitoring of estimates of these quantities as the bus traffic is being defined. An analysis program was developed to calculate loading and latency, with bus traffic the input. As these estimates have become more firm, the concern appears to be more weakly founded.

Present bus loading estimates for operational flights (without the TIC) are under 16% during the busiest phase of flight, and less than 21% with the TIC on test flights. Bus latency averages about one millisecond, with worst-case latency conservatively estimated at three milliseconds.

REFERENCES

- 1) MIL-STD-1553B, dated 12 February 1980.
- 2) General Dynamics Convair MRASM document No. 109-DRB-4005, Revision C, dated 19 August 1982.

JOHN E. LEIB

Mr. Leib has 29 years experience in technical and management assignments at General Dynamics Convair Division. These assignments have been in design and analysis of large systems, including the Atlas and Centaur space boosters and the Tomahawk cruise missile. Currently, he is an Engineering Staff Specialist, conducting and directing system definitions and analyses for the Medium Range Air to Surface Missile, and was one of the architects of the application of MIL-STD-1553B to the internal communications data bus of the MRASM.

Mr. Leib received the BSEE from the University of Michigan in 1951, and the MSEE from the California Institute of Technology in 1953.



MIL-STD-1553B IN MRASM -- THE DESIGNER'S CHALLENGE

Vicki L. Elmore

General Dynamics Convair Division
P.O. Box 80847
San Diego, California 92138
(714) 277-8900, x6108

ABSTRACT

Unique design challenges are created when incorporating MIL-STD-1553B into a low cost tactical missile system. The designer is challenged by the requirement to minimize host computer interaction with bus events, while utilizing "off-the-shelf" components on a single 5.8 inch x 4.8 inch printed circuit card. Further demands are imposed by the requirement to integrate the triple functions of bus controller, remote terminal, and monitor terminal into a single design.

This paper addresses one set of solutions to these and related problems. The paper will also review the process by which the card was designed, and the solutions to the general challenges faced in any 1553B implementation.

INTRODUCTION

The Medium Range Air to Surface Missile (MRASM) is currently under development at General Dynamics/Convair Division. An adaptation of the Tomahawk, MRASM was designed for use by both the Air Force and the Navy. One of the major departures from the Tomahawk design is the replacement of the single computer Tomahawk guidance set with a system of computers connected via MIL-STD-1553B. This computer system is highly flexible and will allow the MRASM vehicle to meet a wide variety of mission requirements.

Design of a MIL-STD-1553B interface for use in the MRASM system creates several unique challenges to the electronic designer. This paper outlines the major problems encountered while developing a MRASM interface and the decisions made to solve these problems. The specific hardware described is from the interface designed for Convair's Digital Integrating Subsystem (DIS) computers.

1.0 MRASM Requirements on 1553B

MRASM system engineers devised a set of requirements for the 1553B interface with goals to optimize the use of this standard in a tactical missile application. Their efforts resulted in the following main points broken down here into general requirements for any terminal on the bus and specific requirements for the DIS computers.

General

- Each terminal on the bus must meet all of the basic requirements of MIL-STD-1553B.

- Data transfer will be based upon a "Poll for Transmission" protocol.
- An automatic RETRY is initiated by the Bus Controller (BC) following transmissions detected as unsuccessful.

DIS Specific

- The design must be packagable on a 5.8 inch x 4.8 inch multilayer printed circuit card.
- The 1553B terminal may only interrupt the host processor at the completion of a successful data transfer.
- One card design must be capable of performing as Bus Controller (BC), Remote Terminal (RT) or Monitor Terminal (MT) following software configuration.
- Each terminal must then be capable of being reconfigured by software resident in the host processor at command.
- Built-in-Test (BIT) capability will be incorporated on card which verifies card operability.

The "Poll for Transmission" protocol is based on the BC polling each terminal on the bus with a "Transmit Vector Word" mode command. (The sequence in which the BC shall poll the terminals is downloaded at reconfiguration.) The RT or MT responds with a status word in accordance to MIL-STD-1553B. The BC then examines the status word to see if the terminal has a message to send (SERVICE REQUEST BIT = 1). If not, the vector word is ignored and the BC resumes participation in the polling sequence.

Reconfiguration of the card is accomplished through a 20 word message (see Figure 1). The card determines its functional type (BC, RT or MT) and all other information that it requires to perform as that specified type from the contents of this message.

The BC is required to insure a transaction is successfully completed or initiate RETRY. A RETRY begins with the retransmission by the BC of the "Transmit Vector Word" mode command which initiated the polling of the terminal. The number of retries that will be attempted is determined by the value downloaded during reconfiguration.

2.0 Architecture

The protocol and limited card space available required the development of a finely tailored architecture. Logistics concerns drove the requirement of a single card performing multifunctions. Cost concerns and schedule requirements drove the desire that any devices selected be available "off-the-shelf". The resulting architecture (see Figure 2) which will be covered in four sections, is a product of the protocol balanced against these concerns.

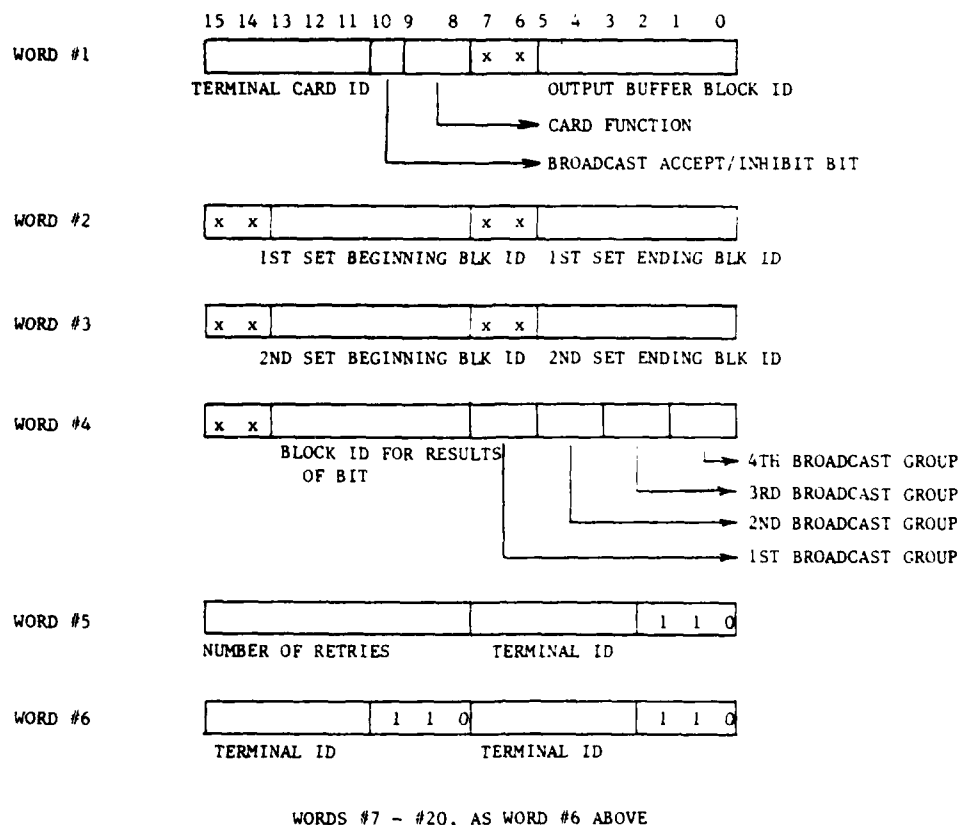


FIGURE 1. RECONFIGURATION MESSAGE

2.1 1553B Data Bus Interface

The 1553B Data Bus interface section is responsible for the receipt and transmission of data, and the Manchester encoding and decoding of that data. "Surface" level decoding of word type (command/status or data) was needed, with further decoding of message type (mode or broadcast command) being preferred.

The data bus was required through MRASM system requirements to be direct coupled stub. The bus transceiver circuit and transformer, were selected to be the 8553 from ILC Data Device Corporation and the X-1269-1 from Technitrol. They were selected mainly for power supply compatibility and size considerations.

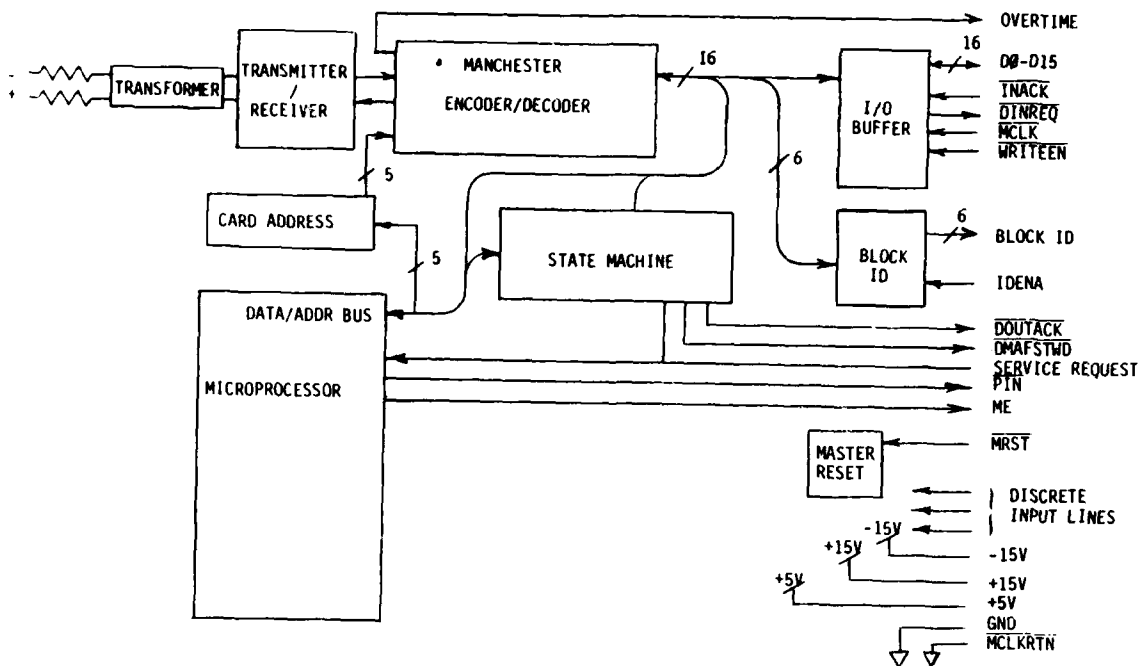


FIGURE 2. BLOCK DIAGRAM OF MRASM MIL-STD-1553B DATA BUS CARD

A large portion of responsibility of the data bus interface fell to a Manchester encoding/decoding device. We chose the 1555 from Circuit Technology Incorporated (CTI) primarily because of the degree of word and message decoding done by the device and the limited number of devices in production at the time of design. The CTI 1555 is a 56 pin hybrid which performs Manchester encoding/decoding. While encoding it performs a parallel to serial conversion and a conversion of this serial data to Manchester coding. It also places the appropriate synchronizing wave form (command or data) in front of each word and adds a parity bit to the end of each word. While decoding it detects the word type from the sync, converts the word from Manchester serial to parallel and checks parity. The 1555 signals whether a message is addressed to a particular terminal or whether it is a broadcast or a mode code command. This device also offers a "wrap-around" mode helpful for implementing Built-in-Test (BIT).

2.2 Microprocessor

The "heart and soul" of the design lies within the microprocessor. Implementing the requirement of a multifunction card, which is reconfigurable at command, was the factor which drove the decision to incorporate a microprocessor into the design.

The choice of microprocessor was determined from the amount of memory, both ROM and RAM, required and the I/O capability needed. The 8051 from INTEL was the final choice. This 8 bit microprocessor contains 4K bytes of ROM, 128 bytes of RAM, and four I/O ports, one of which is dedicated as the data and address bus for accessing external ports.

Card initial configuration, reconfiguration, built-in-test, BC, RT, and MT functions are all accomplished primarily by the 8051.

- RECONFIGURATION -- Reconfiguration is performed when commanded by software resident in the host processor. At initial configuration (after power-up) the 8051 will wait for the command to reconfigure. When receiving the command, the message will then be retrieved from a specific address in the host computer's memory. The card shall then perform as a BC, RT, or MT as directed. If the card is a BC, it will initiate the polling sequence.
- BUILT-IN-TEST (BIT) -- During BIT the microprocessor exercises approximately 80% of the card's circuitry which includes the following tests:
 - Various "wrap-around" tests in the CTI including card ID, broadcast and mode code recognition.
 - Host computer interface testing
 - 8051 ROM checksum test

After completion of BIT the microprocessor will transfer a word containing BIT results to the host computer and signal the completion with an interrupt.

The requirement for the design to have the ability to reconfigure and to perform Built-in-Test (BIT) at command came under the microprocessor's functional category. The main problem of implementing BIT or reconfiguration was deciding when to check commands from the host processor to perform the specific task. This was accomplished by sampling the signal to perform either task only after completing any 1553B transaction. This decision assures that any terminal, regardless of function, will complete a bus transaction without interruption by the host processor.

- BUS CONTROLLER -- As a BC the microprocessor functions include the following:
 - Polling of terminals
 - Generation of appropriate command sequence
 - Knowledge of the type of transaction
 - RETRY, if needed.

At all times, as a BC, the 8051 is aware of what actions are or should be occurring on card as well as on the 1553B Data Bus. Therefore as Bus Controller, the 8051 is always cognizant of which type of 1553B transaction is taking place. The microprocessor's primary responsibility as BC, is the polling sequence and assuring the transactions initiated by a polling are completed. The 8051 accomplishes this by monitoring the bus. Monitoring of the bus is accomplished by checking single input lines from the CTI, decoding for command/status, or data words. Therefore, by

checking the bus in this fashion and being aware of which transaction is taking place the microprocessor determines whether a RETRY should be attempted or not.

- REMOTE TERMINAL -- A RT will take action only when it receives a message directed* to it specifically or if it is configured to accept broadcast messages. The 8051, when acting in accordance with the Remote Terminal function, is aware only of commands, directed or broadcast, and the action it should take in response to those commands. The protocol required both Remote and Monitor terminals to support the Bus Controller's RETRY ability. This is accomplished through time-outs within the microprocessor after response to a transmit command is made. The Remote and Monitor terminals will wait 90 microseconds after transmission of the final data word. This is the time the BC requires to discover a RETRY must be attempted and then to actually transmit the command. Also within the 90 microseconds the RT or MT is required to decode the command.
- MONITOR TERMINAL -- The MT will operate in a manner similar to the RT by taking action when receiving a message directed to it specifically or a broadcast message while configured to receive them. However, the MT is also responsible for inputting any two or more contiguous valid data words which appear on the 1553B bus. Since the microprocessor has the capability to monitor the bus, it is aware of contiguous data words and inputs these words into host processor memory as they appear.

2.3 State Machine

As originally conceived the microprocessor was to have total control of data flow on card. However due to the complexity of commands to be decoded and the limited amount of response time available (see Table 1 and Figure 3), an independent State Machine was designed to perform on-card data flow control.

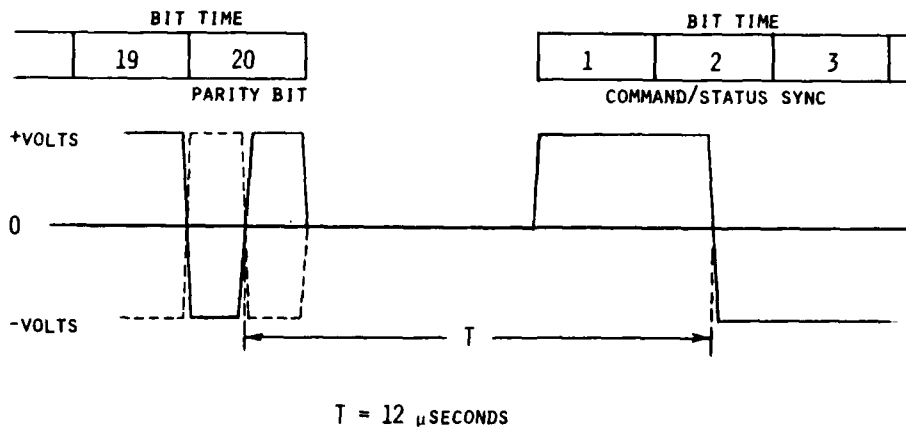


FIGURE 3. MICROPROCESSOR'S LIMITING FACTOR: 1553B RESPONSE TIME

*Directed here refers to a non-broadcast message in which the command word received contains the terminals 5-bit CARD ID.

Table 1. Commands per MRASM 1553B Protocol

ACCEPTABLE

Transmit Command
Receive Command
Transmit Vector Word Mode Command
Transmit Status Word Mode Command
"Selected"* Broadcast Commands (Protocol Specific)
High Priority Receive Commands (Protocol Specific)
High Priority "Selected"* Broadcast Commands
(Protocol Specific)

NONACCEPTABLE

Unimplemented Mode Commands
Commands Not Directed to Specific Terminal
Broadcast Commands Not "Selected"*

*Selection process is one of examining two bits in the Broadcast Receive Command, comparing them with acceptable, or "selected" patterns which are downloaded during reconfiguration.

The State Machine designed accepts commands from the 8051 for data movement. This technique still allows the 8051 control of card functions, with little processing time or memory in the microprocessor required to control specific data flow.

A 6-bit command from the 8051 is decoded by the State Machine's microcode. The command tells the State Machine to send data from a source (on card) to a destination (on card). The State Machine prepares the signals which will accept the data (destination), then enables the signals which output the data (source). The following is a list of commands the State Machine is able to decode to commence data movement on card:

<u>SOURCE</u>		<u>DESTINATION</u>
Status Word Register	to	1553B Data Bus Interface
Status Word Register	to	Microprocessor
Microprocessor	to	I/O Interface
I/O Interface	to	Microprocessor
Microprocessor	to	1553B Data Bus Interface
1553B Data Bus Interface	to	Microprocessor
I/O Interface	to	1553B Data Bus Interface
1553B Data Bus Interface	to	I/O Interface

Although the State Machine was added to allow more time for the 8051 to decode commands and prepare responses, the microprocessor is still responsible for the following functions on card:

- Address of input/output messages at host computer interface
- Performance of Built-in-Test and reconfiguration
- Command control of State Machine
- RETRY
 - attempt (BC)
 - support (RT, MT)

Due to the processing time required for the functions listed above, the microprocessor was unable to fully decode commands as a RT or MT. Therefore, to relieve the 8051 from total command decode, conditional microcode was implemented in the State Machine. This allowed the State Machine to pre-decode the command and send the status word to the CTI if required (transmit command or implemented mode command).

2.4 Host Computer

The host is a Z8000 based, 500 KOPs, 96K RAM, low power avionics computer developed specifically by Convair for use in tactical cruise missile systems. The basis of communication at the I/O interface is Direct Memory Access (DMA) protocol to the host processor. The location in host memory where data buffers reside is determined from a 6-bit field known as the BLOCK ID supplied by the 1553B card. Within the host computer's DMA controller this BLOCK ID is mapped into a 16-bit physical memory address which is the first word address of a DMA buffer.

BLOCK IDs are loaded into memory resident in the microprocessor on the 1553B card during reconfiguration. A single output buffer is used for message transmissions, while two sequential input buffer lists determine the location of the data words input from a receive message. The sequential input buffer lists allow the host processor to receive closely spaced messages without missing any.

Interrupts, both input and output to the host, are initiated by the 1553B card. A BC will interrupt its host processor following a transmission only after receiving a valid status word from the receiving terminal or immediately following a broadcast message. Generation of an output interrupt as a RT or MT begins if no "Transmit Vector Word" Mode command is received by the terminal in 90 microseconds. All card types (BC, RT or MT) receiving messages initiate input interrupts after determining the message is valid. When the MT monitoring the bus receives data words which are not part of an MT directed or broadcast message, an input interrupt will be initiated when a gap in data word reception is encountered.

The I/O interface also supplies four discrete input lines. These are implemented as signals from the host processor as follows:

- Built-in-Test Command
- Reconfiguration Command
- On-card Software Reset Command
- Service Request Status

REFERENCES

- 1) MIL-STD-1553B, dated 12 February 1980.
- 2) General Dynamics Convair MRASM document No. 109-DRB-4005, Revision C, dated 19 August 1982.

BIOGRAPHICAL DATA

NAME: Vicki L. Elmore

TITLE: Electronic Design Engineer

EDUCATIONAL BACKGROUND:

<u>Degree:</u>	<u>College:</u>	<u>Date:</u>
B.S. Electrical and Electronic Engineering Technology	Montana State University	1980

EXPERIENCE:

Ms. Elmore is an Electronic Design Engineer in the Processing Technology group at the Convair Division of General Dynamics. She has recently completed the development and design of the MIL-STD-1553B interface hardware used to link avionics computers in Medium Range Air to Surface Missile (MRASM).

THIRD GENERATION MIL STD 1553B LSI CHIP SET

Robin D Beasley

Flight Automation Research Laboratory
Marconi Avionics Ltd
Rochester
Kent
UK

1. INTRODUCTION

Marconi Avionics Limited's experience with Mil Std 1553 commenced with the implementation of an 'A standard' remote terminal in the Head Up Display for the General Dynamics F16 by our Airborne Displays Division. The Flight Automation Research Laboratory (FARL) have subsequently completed a circuit design for the digital section of a Mil Std 1553B terminal with the LSI implementation carried out by Marconi Electronic Devices Ltd. These LSI devices are currently available through Circuit Technology Inc of New York.

This paper will review the current LSI terminal activity undertaken by FARL. This activity has used experience gained during the previous five-element 1553B LSI development as the foundation for a third generation two-element 1553 LSI terminal design.

The subsequent semiconductor implementation is a collaborative exercise between the GEC Hirst Research Centre and FARL.

2. PRE DESIGN DEFINITION

This development originated from the requirements of a number of future military projects within the United Kingdom. These requirements can be summarised as follows:

- o Nuclear Hardness
- o Improved Architecture
- o Better Bus Control Facilities
- o Long Market Life.

The main features which required definition before the detailed logic design could commence were:

- o Modes of Operation
- o Chip Set Architecture
- o Subsystem Interface Philosophy.

2.1 Modes of Operation

The chip set can operate in three modes; as a remote terminal, a controller terminal and as a passive monitor terminal. As a remote terminal it is fully compatible with Mil Std 1553B. All options and mode commands specified by the Mil Std are implemented. Full and meaningful use is made of status word bits and a comprehensive BIT word is provided. A unique mechanism has been incorporated that permits the subsystem to declare an illegal command legal and vice versa before the chip set services the command. Use of this mechanism is optional, normal operation will ensue if this option is not taken.

The previous Marconi chip set, in bus controller mode, could initiate messages on a word by word basis under subsystem control. This mechanism has been greatly improved and this chip set can initiate complete transfers and error recovery under its own control.

As a passive monitor the two-element chip set will decode all messages on the bus, carry out error checking and pass all valid words to the subsystem.

2.2 Architecture

The basic parameters which governed the chip set architecture are the system requirements equated against the semiconductor technology. This chip set is to be implemented in Silicon on Sapphire (SOS) CMOS which is being developed at the GEC Hirst Research Centre for the UK MOD specifically as a nuclear hard military process. The existence of a five-element terminal chip set (dual, standby redundant configuration) meant that four architecture options existed. A five-to-four conversion offered insignificant architectural benefits. A five-to-three conversion introduced some undesirable aspects. The five-to-two and five-to-one options both gave sound architectures but the five to one conversion exceed the integration capability of the semiconductor technology available.

The details of the chosen two-element architecture is shown in Figure 1. The first element is a Tx/Rx function which can support up to a triple standby redundant bus system with no additional logic or LSI elements. The second element is the terminal control function. Both elements will be compatible with a 48 pin DIL package, the interconnections and system interface will consist of a sixteen bit highway and discrete control lines.

2.3 Subsystem Interface Philosophy

The subsystem interface philosophy adopted by Marconi Avionics for the five-element chip set has been maintained in the two-element, and so 1553 terminal logic defined by the subsystem rather than by the Mil Std, has not been included in either chip set. Again, effort has been concentrated on producing a set of interface signals that allow a user to integrate the chip set into a system efficiently.

Experience gained by users of the previous LSI chip set has allowed the selection of a more efficient set of subsystem lines. A full definition of all the RT subsystem signals is given later in the paper.

The Tx/Rx element carries out the following functions:

- o Waveform reception
- o Message validation checks
- o Broadcast command detection
- o Reply timeout and end of transmission detection
- o Bus selection and shutdown control
- o Terminal loop test and self test
- o Partial BIT word recording
- o Partial mode command execution
- o Waveform transmission.

The terminal controller carries out the following functions:

- o Command word recording
- o Command legality checks
- o Command execution state sequencing
- o Partial status recording
- o Partial BIT word recording
- o Partial mode command execution
- o Data word count
- o RT/BC subsystem interface control
- o RT subsystem handshake failure check
- o Instruction decoding
- o Message execution state sequencing
- o Report word generation and control
- o Subsystem interrupt control
- o Automatic retry control

3. FUNCTIONAL DEFINITION

This section reviews the main design features of the chip set. The chip set is fully compliant with Mil Std 1553B and the design has been arranged such that this compliance does not require a rigorous knowledge of the Mil Std by the user, neither will it permit an invalid bus response by incorrect use of the subsystem interface lines. The only functional aspects requiring subsystem intervention are those defined by 1553B as being subsystem dependent, such as the contents of the Vector word.

3.1 Basic Characteristics

The chip set is designed to operate over the temperature range of +125°C to -55°C and has a storage range of +150°C to -65°C. A power supply of 5 volts is required. Internal power up initialisation allows the first command to be fully serviced. The active user I/O lines to the chip set will be TTL compatible.

3.2 Data Transfers/Mode Codes

The chip set can handle all types of data transfers and mode codes. The mode codes have been fully implemented and protected against incorrect T/R bit and broadcast bit. The chip set will also check that the correct number of contiguous data words are present.

This chip set includes an illegal/legal message enable/disable facility which will allow a subsystem to selectively make any valid subaddress and/or word count illegal before the chip set starts to service the command.

A terminal loop test is also included, by which a receiver monitors the output of its associated transmitter. Loop test fail will cause a transmission abort and setting of the terminal flag.

3.3 Status Words

The bits in the status word have been meaningfully utilised. The instrumentation bit, busy bit, service request bit and the broadcast command received bit are utilised as per Mil Std 1553B.

The message error bit is set if:

- o The message is too long or too short.
- o The message or words are invalid.
- o Illegal use of broadcast is made.
- o A Tx command word with contiguous data is received.
- o The subsystem sets ILLEGAL COMMAND.

The terminal flag will be set if:

- o The loop test fails.
- o The RT address parity check fails.
- o Terminal self test failure occurs.
- o A transmitter overrun occurs.

The subsystem flag will be set if:

- o The subsystem makes incorrect use of the data transfer mechanism.
- o Set by the subsystem.

3.4 BIT Word

An internal BIT word is available via the bus by use of the relevant mode code. The bits of this word have been defined as follows:

- LSB
- o Tx timeout error
 - o Subsystem handshake failure
 - o Loop test failure
 - o Illegal T/R bit
 - o Illegal command
 - o Word count low
 - o Word count high
 - o Illegal broadcast
 - o Bus 0 shutdown
 - o Bus 1 shutdown
 - o Bus 2 shutdown
 - o Terminal flag inhibited
 - o Tx time out on Bus 0
 - o Tx time out on Bus 1
 - o Tx time out on Bus 2
 - o Reserved.

4. SUBSYSTEM INTERFACE

4.1 Signal Definition

Listed below are the main subsystem interface lines available to the user for remote terminal operation. Each signal definition has the signal name, with corresponding abbreviation, the number of lines and a short functional description.

Data Highway, B0 - B15, (16 off). This is a bidirectional highway used to transfer 16 bits of data to and from the subsystem.

Buffer Enable, $\overline{\text{BUFEN}}$, (1 off). This line goes low to enable the data highway buffer between the terminal and the subsystem.

Read/Write, $\overline{\text{R/W}}$, (1 off). This line indicates the direction of information transfer between the terminal and the subsystem.

Strobe, $\overline{\text{STROBE}}$, (1 off). This information transfer strobe will pulse indicating valid data present on the data highway.

Data Transfer Request, $\overline{\text{DTRQ}}$, (1 off). This line goes low to request permission to transfer a data word to or from the subsystem.

Data Transfer Acknowledge, $\overline{\text{DTAK}}$, (1 off). This line should be driven low to grant permission to perform the requested data word transfer.

Mode Data Transfer, $\overline{\text{MDT}}$, (1 off). This line goes low to indicate that the data word being transferred is associated with a mode command.

Receive Command, $\overline{\text{RXCMD}}$, (1 off). This line goes low to indicate that a valid command word for this RT is on the data highway and should be written into the subsystem command word latch.

Status Enable, $\overline{\text{STATEN}}$, (1 off). When low this line will enable the contents of subsystem status latch onto the data highway.

Address Enable, $\overline{\text{ADEN}}$, (1 off). During terminal initialisation this line will be used to enable the terminal address onto the data highway.

In Command, $\overline{\text{INCMD}}$, (1 off). When low this line indicates that the terminal is currently servicing a command word.

Good Block Received, $\overline{\text{GBR}}$, (1 off). When a fully validated block of data has been received this line will authorise its use by the subsystem.

Mode Data Received, $\overline{\text{MDR}}$, (1 off). This line will pulse low when valid mode data has been received.

Synchronise, $\overline{\text{SYNC}}$, (1 off). This line will pulse low if a valid synchronise without data mode code is received.

Busy Request, BUSYREQ, (1 off). A subsystem taking this line low will cause the chip set to set the busy bit in the status word and inhibit all data transfers to or from the subsystem.

Busy Acknowledge, BUSYACK, (1 off). This line will go low to indicate that the subsystem has free access to any shared store.

Reset, RESET, (1 off). This signal when low causes the internal circuitry to reset to the quiescent initialised state.

RT/BC, RT/BC, (1 off). This line when high will cause the terminal to function as a remote terminal, when low as a bus controller terminal.

4MHz clock, CK4, (1 off), 4MHz System Clock.

4.2 Remote Terminal Initialisation

When power is applied to the chip set a reset cycle will be automatically executed thus causing the internal circuitry of the chips to initialise.

Such a reset cycle will also take place if the open drain RESET input/output line is taken low. This can be achieved by sending a reset mode command to the RT via the data bus, in which case the chip set will pulse the RESET line low, or by the subsystem pulling the RESET line low for a minimum of 0.5 microseconds.

The reset cycle commences with the RESET line being taken low, this forces the ADEN line low which in turn enables the RT address, parity and broadcast enable information from the subsystem onto the data highway. The RESET line being low also forces the internal circuitry of the LSI devices to initialise. At the end of the reset cycle, that is on the low to high transition of RESET, the RT address, parity and broadcast enable information is latched into the terminal and internal circuitry is released for normal operation.

The RT address, parity, and broadcast enable information is derived from the subsystem and should be buffered onto the 16-bit data highway, B0-B15, by a tristate buffer.

The terminal makes use of an 8-bit external latch to record subsystem status information which is used by the terminal to control the execution of certain commands and to determine the terminal status word contents.

DBCACC is the dynamic bus acceptance line which is used by the subsystem to indicate whether or not it is willing to accept bus control if offered. If this line is low then the dynamic bus control acceptance bit of the terminal status word will be set in response to a legal mode command for dynamic bus control allocation.

SSERR, the subsystem error line is the means by which the subsystem can flag an internal fault condition such as a self test or BITE failure. This line being low will cause the subsystem flag of the terminal status word to be set.

By pulling SERVREQ service request line low the subsystem can cause the service request bit of the terminal status word to be set and thus initiate some predetermined asynchronous operation.

The ILLEGAL COMMAND line provides a means by which the subsystem can declare any command word to be illegal. If this line is low the terminal will inhibit data transfers to or from the subsystem and after message validation will respond with the message error bit of the terminal status word set.

The most obvious use of this facility is in a system which makes use of the instrumentation bit of the terminal status word. Within such a system any command word which has the most significant bit of the subaddress field set low must be illegal. Hence, by connecting the most significant bit of the subaddress field of the command word latch to the ILLEGAL COMMAND line such commands would not be actioned by the terminal.

The ALLOW CODE line provides the subsystem with the capability to declare any of the currently reserved mode codes as being legal and meaningful to the subsystem.

If a reserved mode command is received and this line is not taken low within the allocated time then the terminal will treat the command as being illegal and after message validation will respond with the terminal status word with the message error bit set.

The RES0, RES1, RES2 lines provide the subsystem with the capability of setting any or all of the currently reserved bits of the terminal status word, that is bits 5, 6 and 7 respectively.

4.3 Subsystem Interface Operation

Figure 2 shows the main subsystem interface waveforms that a user would encounter during a typical data transfer. This specific data transfer is a bus controller to remote terminal transfer of two data words into sub address one. The first word shown on line PDIN (the positive threshold logic line between the chip set and the bus driver/receiver) is the command word from the bus controller.

When the terminal receives a valid command word with the correct terminal address any current command execution will be aborted by the INCMD line being forced inactive. The terminal will then enable the command word onto the data highway B0-B15 and write this into the subsystem command word latch by means of the RXCMD and STROBE lines. There then follows a delay of approximately 1 us to allow the subsystem to compile its status information and specify whether or not it is busy, after which this information is latched by INCMD going active low. The subsystem status latch contents are then read into the terminal by means of the STATEN and STROBE lines.

Command execution will then take place as specified by the command word and subsystem status information.

When a valid, non-mode command word is received with a T/R bit of zero the terminal will initialise and the protocol state sequencer will enter the receive sequence.

The exact details of the receive sequence are dependent upon whether or not the subsystem has declared itself to be busy or has declared the command word to be illegal for that terminal via the ILLEGAL COMMAND bit of the subsystem status latch.

In order to declare itself busy the subsystem must pull the BUSYREQ line active low and wait for the BUSYACK line from the terminal to go low. So long as the BUSYACK line is low the busy bit of the terminal status word will be set. The BUSYACK cannot change state while the terminal is actively servicing a command, that is while INCMD is low. The terminal will not make any attempt to transfer data words to or from the subsystem while BUSYACK is low.

BUSYREQ and BUSYACK therefore constitute a true handshake mechanism between the terminal and the subsystem for access to the interface circuitry.

It is assumed the subsystem is not busy and the command has been allowed by the subsystem.

This condition is defined by BUSYACK and the ILLEGAL COMMAND bit of the subsystem status word being both high. In such a situation the normal receive data sequence will be entered.

The terminal will wait until the valid data word is received. When a valid data word is received the data transfer request line, DTRQ, to the subsystem will go active low. The subsystem must reply with a data transfer acknowledge, DTAK, within 1.5 us when the terminal will write the data word into the subsystem store by means of the DTRQ, BUFFEN, R/W and STROBE lines.

If the subsystem fails to reply with DTAK within the allowed time then a handshaking failure will be declared causing the Subsystem Flag of the terminal status word and the Subsystem Handshake Failure bit of the BIT word to be set. The GBR pulse to the subsystem will be suppressed.

At the end of the data word transfer the two data words received will be compared with that specified by the command word and the receive data sequence will continue until these numbers are equal.

When the correct number of data words have been received the terminal will check for an end of transmission, EOT, and if obtained will load the terminal status word into the transmitter, shown by PDOUT, for transmission on the bus. At the same time the good block received signal, GBR, to the subsystem will pulse low for 250ns to declare the received data valid.

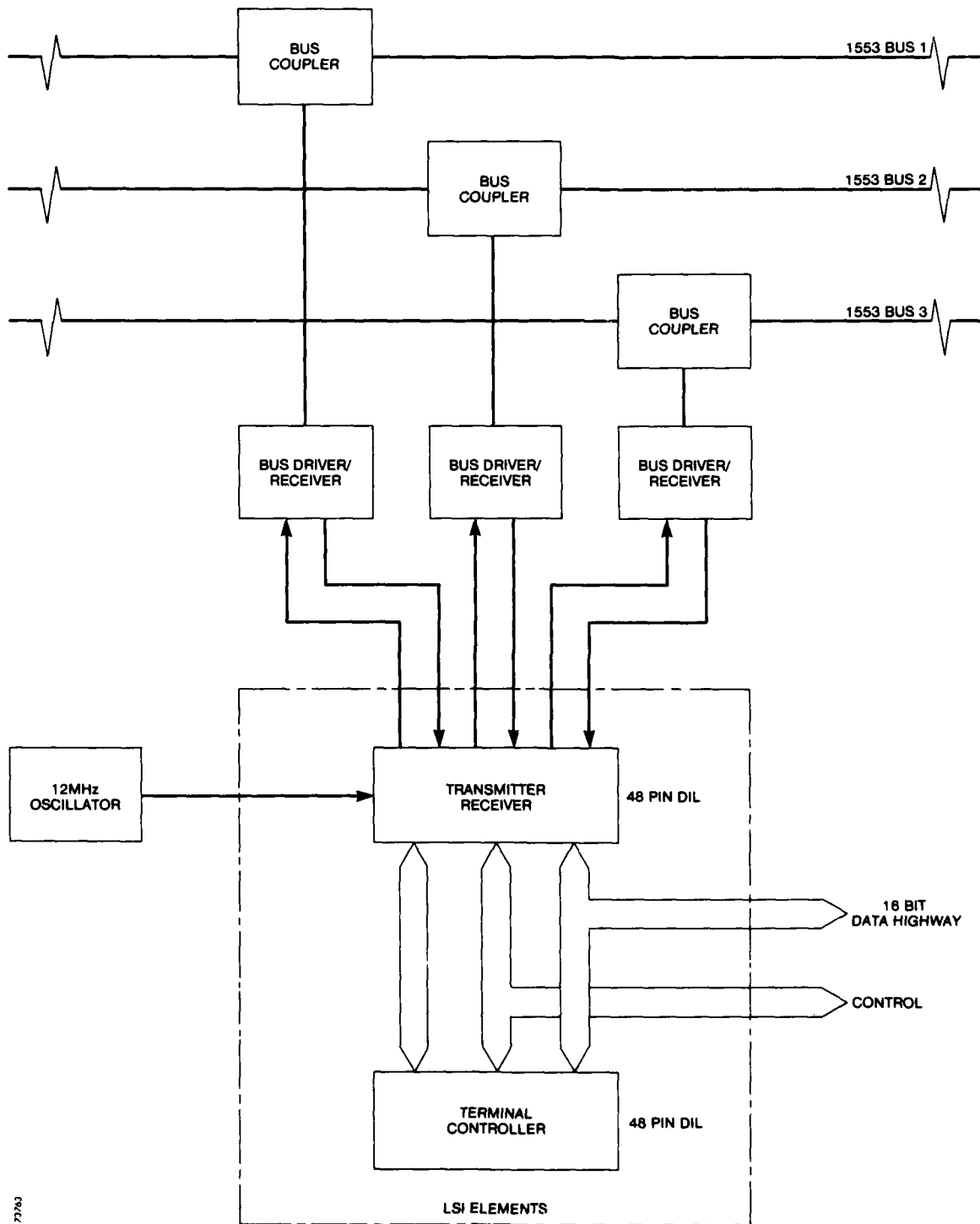
If fewer valid data words are received than specified by the command word the Message Error bit of the terminal status word and the Word Count Low bit of the BIT word will be set and status transmission and the GBR pulse suppressed.

If the received message does not terminate after the number of data words specified by the command word then the Message Error bit of the terminal status word and the Word Count High bit of the BIT word will be set. Status transmission and the GBR pulse will also be suppressed.

If the command word has the broadcast terminal address, execution will take place as above with the exception that transmission of the terminal status word will be suppressed.

ACKNOWLEDGEMENT

The author wishes to thank Marconi Avionics management for permission to publish this paper and MOD for their support of the technical development by the Data Management Team of FARL.



2783

Figure 1 LSI Architecture

71674

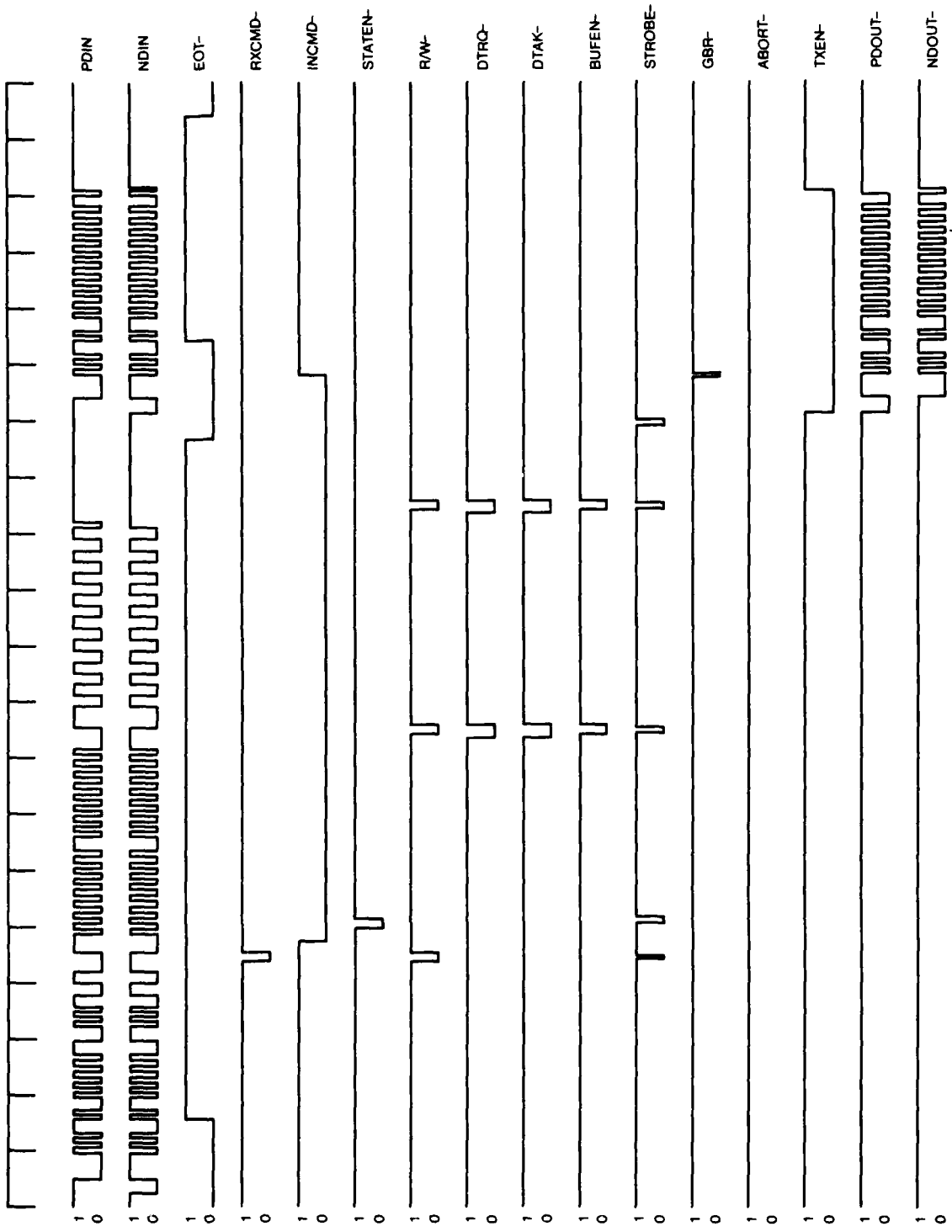



Figure 2 Subsystem Interface Signals



DATA WORD STANDARDIZATION



Francis E. Peter
Naval Avionics Center
6000 E. 21st Street
Indianapolis, IN. 46218
(317) 353-7871

AUTHOR:

Francis Peter is an electronics engineer in the Avionics Support Engineering Division of the Naval Avionics Center. Francis graduated from Ohio Northern University in 1970 with a BSEE degree. As a former member of the Electronic Warfare Branch, he has been designing real-time computer controlled systems since 1972. He has been a member of the SAE/A-2K Subcommittee since 1977 and has been a member of the MIL-STD-1553 Update Task Group and the Multiplex Applications Handbook Task Group. He is also a member of the Tri-Service Multiplex Committee. Now as a member of the Computer and Architecture Systems Engineering Branch, Francis is the Project Engineer for the NAC effort on the F-14 Avionics Improvement Program and provides support to the NAVAIR Avionics Architect and to the JVX and VTXTS aircraft development programs.

ABSTRACT:

Two defense contractors, Boeing and Sencor, have proposed standard data word formats for use with MIL-STD-1553 systems. These proposed formats differ in that the Boeing proposal is based on the historical usage of English units while the Sencor proposal is based on a conversion to metric units. At the request of the Tri-Service Multiplex Committee, the SAE/A-2K Subcommittee for Multiplexing has formed a Data Word Standardization Task Group to revise Chapter 11, Data Word and Message Format Guidelines for publication in the Multiplex Applications Handbook. The Chapter 11 will contain both English and metric units to allow industry an orderly conversion to metric units while providing an unofficial standard for English units.

The Task Group has revised the Boeing draft Chapter 11 with the following major changes: an improved ICD format, modified data formatting rules, a simplified procedure for developing nonstandard data words, and a detailed review of the data word formats. Chapter 11 will be submitted to the SAE/AE-9 (formerly A-2K) Subcommittee at the fall 1982 meeting. Development of an all metric MIL-STD for data words within the next two years will be recommended.

BACKGROUND:

The necessity for standardizing data word and message formats became evident as more and more subsystems used the MIL-STD-1553 interface as the basic input and output communication interface. MIL-STD-1553 establishes a

rigorous definition of the electrical parameters and the communications protocol for such an interface. However, no standard exists for the format of the data to be transferred across this interface. This has resulted in basic incompatibilities between similar subsystems developed for different aircraft. The solution to this problem in the past has been to utilize a processing element onboard the aircraft, typically the bus controller, to rescale and reformat data as required to make it compatible with different subsystems. This has resulted in increasingly high levels of processing overhead.

Another factor which must be considered at this time is NATO compatibility and metrication. Public Law 94-168, Metric Conversion Act of 1975, and Department of Defense (DOD) Directive 4120.18, Use of Metric System of Measurement, define a policy of conversion of military systems to the metric system. It is necessary to define a rational approach to the conversion of data used in avionics systems to metric units.

In response to these problems, in May 1980 the Naval Air Development Center (NAVAIRDEVCEN) contracted with SEMCOR to perform an analysis of avionic systems to determine whether data word format standards for aircraft armament systems were practical. SEMCOR's report 4092 TM-81-BASIC-006, AAAS Multiplex Armament Data Word Standardization Study, published in February 1981 developed standard data words for 20 data types.

In August 1980, the U.S. Army Avionic Research and Development Activity (AVRADA) contracted with SEMCOR to expand the scope of the NAVAIRDEVCEN investigation to data word format standards applicable to the full range of avionic systems, including communications, navigation, air data, flight control, environmental, self-protection, display and control, electrical, engine management, lighting, and stores management. SEMCOR's final report, STR-DD-81273-1, MIL-STD-1553 Data Word Standardization Technical Report, was published in September 1981. This report consolidated many of the data types previously recommended and added 12 data types for a total of 25 generic data types.

In December 1980, the U.S. Air Force Systems Command contracted with Boeing Military Airplane Company to perform an investigation of data word standardization based upon currently used data words in avionics systems. The final report, released in October 1981, was a draft Chapter 11, Data Word and Message Format Guidelines, for the Multiplex Applications Handbook. This study also included documentation guidelines and message formatting.

A parallel effort, Study 3914AVS, has also been initiated by NATO's Third AVS Working Party to investigate data word and message format standards in conjunction with STANAG 3838. The U.S. Army (AVRADA) is the project office for this effort.

TRI-SERVICE MULTIPLEX COMMITTEE:

In January and again in March 1982, members of the Tri-Service Multiplexing Committee met to review the data word standardization efforts to date. Since the primary difference between the SEMCOR and Boeing reports was the choice of metric or English units, resolution of this difference was necessary before any further progress on standardization could be made. After

much discussion of U.S., DOD, and individual service positions on metrication, the following position was adopted:

1. Metric units are desired for future standardization as soon as practical. To meet this goal, a metric MIL-STD should be developed.
2. Industry must pace the conversion to metric units. To allow this while meeting standardization needs, a dual units (metric and English) Chapter 11 for the Multiplex Applications Handbook would be developed.

This position would be submitted to the Society of Automotive Engineers (SAE) A-2K Subcommittee for Multiplexing for industry comments and approval. The SAE A-2K would also be asked to review in detail the draft Chapter 11 for the Multiplex Applications Handbook. Since the completion of Chapter 11 is preventing the publication of the Multiplex Applications Handbook as a MIL-HDBK, the A-2K would be asked to complete this review in a limited time.

TASK GROUP:

At the March 1982 meeting of the SAE A-2K Subcommittee for Multiplexing, the Tri-Service position on metrication was approved in principle. The A-2K also agreed to form a Task Group to review and submit for final approval, at the fall 1982 meeting of the A-2K, a revised Chapter 11. In addition, this Task Group would recommend whether the writing of a MIL-STD for data words was practical at this time or should wait until industry had some experience using the Chapter 11 word definitions. Francis Peter of the Naval Avionics Center was chosen as the Task Group Chairman.

The following members were chosen for the Task Group:

Military

Naval Avionics Center -- Chairman
Air Force Systems Com. (SEAFAC)
Army Avionic Research and Devel. Activity
Air Force Armaments Lab.

Airframe Integrators

Boeing Military Aircraft Company
Grumman Aerospace Corp.
LTV Vought Corp.

Equipment and Subsystem Manufacturers

Delco Electronics
Honeywell, Inc.
Rockwell, Collins
SCI Systems, Inc.
Singer Kearfott
Sperry Flight Systems
Teledyne Systems Company
Westinghouse Electric Corp.

Consultants
ARINC Research Corp.
SEMCOR, Inc.

Three meetings were scheduled and held to perform this review. The results of these meetings are summarized below.

DOCUMENTATION:

During the previously described studies, it became obvious that if standardization is to succeed, then a common documentation format must be developed. Such a format must allow easy computerization of the information as well as provide the necessary level of system documentation. Figures 1 and 2 show the presentation format selected by the Task Group. Note that these figures are not to scale and that the precise field locations are shown in Chapter 11. This presentation format provides several unique data fields including a descriptive name, a word unique identification (ID), original source(s) of data, final destination(s) of data, and scaling information as well as descriptive text fields. Similar sheets are defined for Command and Status words to enable the documentation of complete messages.

As a data word is defined, some of the fields are filled in including basic name, signal type, units, and scaling. When a specific subsystem uses a data word, this information is expanded including a more descriptive name, computation rate, refined scaling information, and bit names and descriptions. This process of adding and refining data is repeated through message definition and system documentation until a complete Interface Control Document (ICD) is developed for the system.

A unique capability of this presentation format is the existence of both a WORD ID and source and destination information. The WORD ID is a unique ID tag which identifies a specific transfer of the data word on the data bus. The source and destination are used to identify the originating point (source) for that data word and the ultimate user (destination) for the data word. For example, if the bus controller asks for attitude information from an inertial system and a vertical gyro and selects the best information and sends that information to a display and an autopilot, then the sources are the inertial system and the vertical gyro and the destinations are the bus controller, the display, and the autopilot. Also, if the bus controller converts the attitude data in direction cosines and sends this data to an antenna pedestal, the bus controller is the source of the direction cosines. Each of these transfers would have a different WORD ID associated with the transfer.

SIGNAL CODING:

Several guidelines were developed for use in documenting nonstandard data words. Foremost among these is the suggestion that all numeric data be represented as 2's complement even if the quantity is never negative. This is useful in that few computers are able to handle unsigned numeric quantities in an efficient manner. Other rules include the packing of discrete data in the most significant bit positions and the aligning of character and coded data on half and quarter word boundaries as appropriate. Another important guideline

DOC. NO.
DATE
SHEET OF

REV.

WORD NAME :

WORD ID :	MAX VALUE :
SOURCE(S) :	MIN VALUE :
DEST(S) :	RESOLUTION :
COMP RATE :	ACCURACY :
XMIT RATE :	MSB :
SIGNAL TYPE :	LSB :
UNITS :	FULLSCALE :

SIGNAL NAME	BIT NO.	SIGNAL DESCRIPTION
	-00-	
	-01-	
	-02-	
	-03-	
	-04-	
	-05-	
	-06-	
	-07-	
	-08-	
	-09-	
	-10-	
	-11-	
	-12-	
	-13-	
	-14-	
	-15-	

REMARKS:

(PAGE)

Figure 1. Presentation Format, Single Word

DOC. NO.
DATE
SHEET OF

REV.

WORD NAME :

WORD ID :	MAX VALUE :
SOURCE(S) :	MIN VALUE :
DEST(S) :	RESOLUTION :
COMP RATE :	ACCURACY :
XMIT RATE :	MSB :
SIGNAL TYPE :	LSB :
UNITS :	FULLSCALE :

SIGNAL NAME	BIT NO.	SIGNAL DESCRIPTION
	MSW -00-	
	-01-	
	-02-	
	-03-	
	-04-	
	-05-	
	-06-	
	-07-	
	-08-	
	-09-	
	-10-	
	-11-	
	-12-	
	-13-	
	-14-	
	-15-	
	LSW -00-	
	-01-	
	-02-	
	-03-	
	-04-	
	-05-	
	-06-	
	-07-	
	-08-	
	-09-	
	-10-	
	-11-	
	-12-	
	-13-	
	-14-	
	-15-	

REMARKS:

(PAGE)
Figure 2. Presentation Format, Double Word

is the concept of gathering validity bits into a separate data word as discrettes. This allows simultaneous testing of all validity bits associated with a given message.

SIGNAL CATEGORIES:

After a considerable review of the data words recommended in the previous studies, a set of 42 data parameters in 24 signal categories was selected. Table 1, Standard Signal Categories, lists the signal categories and the associated units and words used with each. All metric units used, except semicircles (defined as Pi radians), conform to National Aerospace Standard NAS10001, Preferred Metric Units for Aerospace. The signal set was selected to be all inclusive so that the broadest possible review of data types may be made at this time.

MESSAGES:

After a review of potential standard messages, the decision was made to not include any standard messages as a part of this standard. Instead, guidelines have been developed to standardize the structure of messages. The concensus of the Task Group is that message standardization must wait for industry to gain some experience in using the standard data words.

The following list of general rules for message construction should be followed:

1. Multiple messages from a subsystem containing the same data words should have those data words in the same order.
2. Shorter messages, which contain some of the data words found in a longer message, should be a subset of the longer message with the same data word positions.
3. A header word may be provided as the first word of the message.
4. A validity word may be provided to indicate the validity of individual data words within the message.
5. The word sequence within a message should be as follows:
 - (a) Header word (optional)
 - (b) Validity Word (optional)
 - (c) Time tag (optional)
 - (d) Data words.
6. Use standard data words whenever possible.
7. Do not assign 32 data words to a message, i.e. leave room for expansion.
8. Do not leave spare or reserved words in the middle of messages.

Table 1. Standard Signal Categories

CATEGORY	UNITS	WORDS
Acceleration	Meters/Sec/Sec	Double
	Feet/Sec/Sec	Single
Angular	Semicircles	Double
Angular Acceleration	Semicircles/Sec/Sec	Single
Angular Velocity	Semicircles/Sec	Double
ASCII Data	Unitless	Single
BCD Data	Unitless	Single
Convergence Factor	Unitless	Single
Cosine/Sine	Unitless	Double
Counts	Unitless (Signed)	Single
	Unitless (Unsigned)	Single
Data Validity	Unitless (Checksum)	Single
	Unitless (Error Protection)	Single
Deviation	DDM	Single
Distance	Metres	Double
	Feet	Double
	Kilometres	Double
	Nautical Miles (Low Range)	Single
	Nautical Miles (High Range)	Double
Flow	Kilograms/Hour	Single
	Kilograms/Min	Double
Frequency	Hertz	Four
	Kilohertz (ADF)	Single
	Megahertz (VHF/UHF)	Single
Mass	Kilograms (Low Range)	Single
	Kilograms (High Range)	Single
Percent	Unitless	Single
Pressure	Kilopascals	Double
	Inches of Mercury	Single
Ratio	Unitless	Single

Table 1. Standard Signal Categories (Cont.)

CATEGORY	UNITS	WORDS
Temperature	Celsius	Single
Time	Time of Day	Three
	Microseconds (Time Tag)	Single
	Seconds	Single
Torque	Newton-metres	Double
UTM	Unitless	Five
Velocity	Metres/Sec	Double
	Feet/Sec	Double
	Kilometres/Hour	Single
	Knots	Single
	Mach	Single
Voltage	Volts	Double

FUTURE PLANS:

At the fall 1982 meeting of the SAE/AE-9 Aerospace Avionics Equipment and Integration Committee (formerly the A-2K Subcommittee), the revised Chapter 11 will be presented for review by the committee membership. Pending formal approval of the revised document, any suggested changes will be incorporated into a final draft and the revised Chapter 11 will be submitted to the Tri-Service Multiplex Committee for final approval. This effort will complete the Multiplex Applications Handbook and the complete document will be submitted for publication as a MIL-HDBK.

Also, an all metric version of the data words has been submitted to NATO's Third AVS Working Party as an input to Study 3914AVS. The working party is currently reviewing the proposed data word standards.

ACKNOWLEDGMENTS:

The completion of an effort as large as this review in the short period available requires the cooperation of many people to be successful. The author wishes to extend his thanks to Don Ellis, AeroSystems Associates, Chairman of the AE-9A (formerly A-2K) and Al Crossgrove, Boeing Company, Chairman of the AE-9, for their support and patience during this review. Special thanks also go to John Respass, AVRADA, who co-chaired the data word section of the Task Group and to Doug D'Avino, Semcor, who under contract with AVRADA, supplied documentation services to the Task Group. Finally, sincerest thanks go to the members of the Task Group and their companies for their support in this effort.



MIL-STD-1553B VALIDATION TESTING

Duane J. Thorpe
ASD/ENASF (SEAFAC)
Wright Patterson AFB, OH 45433

Kumar V. Vakkalanka
ASD/ENASF (SEAFAC)
Wright Patterson AFB, OH 45433

ABSTRACT

One of the major responsibilities of the Systems Engineering Avionics facility (SEAFAC) is to act as the Air Force Office of Primary Responsibility (OPR) for MIL-STD-1553 applications. Since its inception in 1974, this organization played a vital role in promoting and enforcing the standard. In the area of testing, SEAFAC set the pace by developing the Test Plan and specialized test equipment. Our advanced validation Test Facility allows us to test a wide variety of MIL-STD-1553B interfaces, such as Remote Terminals, transceivers and, of particular interest, the Large Scale Integrated circuit chips, to ensure their conformity to MIL-STD-1553B. This paper outlines our present validation testing as well as planned automation of our test facility. A summary of our experiences is presented. The paper concludes with SEAFAC's role in future testing.

INTRODUCTION:

The efforts of Systems Engineering Avionics Facility (SEAFAC) are well known. In its capacity as the Air Force Office of Primary Responsibility (OPR) for MIL-STD-1553B, SEAFAC played an important role in promoting and enforcing the standard. Several papers were presented in the past few years to keep the industry up-to-date on the progress we made. We have seen innumerable papers on the advantages of Multiplexing. In 1980 Charles Gifford outlined the USAF goals for Multiplexing (Ref 1). He pointed out some of the engineering shortfalls, namely, lack of Standardized Test Plan, lack of Standardized Message Formats and lack of Specialized Test Equipment. In all these areas SEAFAC has made considerable progress. The MULTIPLEX HANDBOOK was developed which has become a valuable source of reference for all engineers (Ref 2). Several bus testers were developed by SEAFAC engineers (Ref 3). Of these the BUS TESTER IV developed by Loral Data Systems, under USAF contract is now being used by SEAFAC in its Validation Testing Facility. SEAFAC developed a TESTPLAN which forms the basis for all Remote Terminal testing. Standardized Word Formats were developed to meet the Air Force needs, under contract with Boeing Airplane Company. A Validation Test Facility was pioneered by SEAFAC which is being widely used for validation of Remote Terminals (RTs). This paper outlines the Validation Testing performed at SEAFAC.

PURPOSE OF SEAFAC'S MIL-STD-1553B VALIDATION TEST FACILITY

The following are the objectives of SEAFAC's MIL-STD-1553B Validation Test Facility:

1. To develop a basis for supporting MIL-STD-1553. SEAFAC's Validation Testing facility has been put to good use. Several leading Companies have used, and are continuing to use, this facility not only to confirm their own test results but also to gain an insight into to Air Force testing methodology. Companies lacking adequate Test Facilities of their own have ben coming to SEAFAC to check out their systems. Indeed, our Test Facility became a proving ground for all those contributing to the MIL-STD-1553B.

2. To Test Large Scale Integrated Circuit (LSI) Chips. SEAFAC is particularly interested in making the Validation Test Facility available to MIL-STD-1553B LSI Chip manufacturers. The LSI Chips implement all of the MIL-STD-1553B protocol features. Hence, Validation Testing of all Remote Terminals utilizing these chips would become easier once the LSI Chips are fully tested in our facility.

3. To provide direct support to our Systems Program Office. This, needless to say, is our primary function.

4. To provide a training base for the engineers. SEAFAC provides an excellent training base for all engineers to keep them current with technology. In addition to in-house training programs on the MIL-STD-1553B, our engineers get hands-on-experience in our Test Facility. Their association with industry personnel keeps them abreast of developments taking place outside. This ensures credibility in providing support to our Systems Program Office as well as to industry.

SEAFAC'S TEST PLAN

Tests performed by SEAFAC are broken down into three categories:

1. ELECTRICAL TESTS
2. PROTOCOL TESTS
3. NOISE INJECTION TEST

The Electrical and Protocol tests are performed under normal and abnormal conditions ."Abnormal conditions" refer to out of the ordinary stimuli the RT may be subjected to, such as errors- both electrical and protocol- which are rare but catastrophic if not handled properly by the RT. The Test Plan outlines acceptable responses to such stimuli by the RT. Testing under the under these conditions insures against RT failures.

An RT failure may be defined as any condition

1. which causes the RT to lock-up
2. which causes the RT to fail to the extent that power must be recycled in order to recover, or
3. which causes the RT to respond in a form other than that called out in MIL-STD-1553B".

ELECTRICAL TESTS:

These tests verify RT's compliance with the electrical specifications on Transmission method (paragraphs 4.3.3) and Terminal characteristics (paragraph 4.5.2) outlined in the MIL-STD-1553B. The Test Plan outlines the following nests under this category.

OUTPUT CHARACTERISTICS

1. Waveform Polarity.
2. Transmitted Word Size and Encoding.
3. Waveform Amplitude.
4. Waveform Risetime.
5. Terminal Response Time.
6. Frequency Stability.
7. Accuracy/Long Term Stability.
8. Short Term Stability.
9. Zero Crossing Stability.
10. Overshoot and Ringing.
11. Waveform Tailoff.
12. Output Noise.

INPUT CHARACTERISTICS

1. Zero Crossing Distortion.
2. Amplitude Variations.
3. Common Mode Rejection.
4. Input Impedance.

PROTOCOL TESTS:

These tests verify RT's compliance with the various protocol requirements set in the Standard.

REMOTE TERMINAL OPERATION:

1. RT Response To Command Words.

The purpose of this test is to verify that the RT under test responds properly, in form over the data bus, to all valid and illegal command words and that the RT does not respond to any address different from the one defined for the RT under test.

2. Intermessage Gap.

This test verifies RT's response to messages with minimum intermessage gap of 4 microseconds.

ERROR INJECTION TESTS:

These tests are intended to examine the RT's response when specific errors are forced into the message stream. When such an error is injected into the message stream The RT should not respond with a status word but should internally set the message error bit in its status word buffer. If the command word it decodes fails any of the validity tests it should ignore the command and wait for the next command and, in this case the Message Error bit should not be set in its status word buffer.

The following tests are performed under this category:

1. PARITY.
2. WORD LENGTH.
3. BIPHASE ENCODING.
4. SYNC ENCODING.
5. MESSAGE LENGTH.
6. DATA CONTIGUITY.

DUAL REDUNDANT OPERATION:

This section addresses the (optional) bus switching requirements of MIL-STD-1553B. These are:

1. Bus-Switching - RT Receiving.
2. Bus-Switching - RT Transmitting.

MODE COMMANDS

These tests verify that the Rt under test responds properly to mode commands in both form and function. (see Table 1, Mil- Std-1553B)

STATUS BITS:

These tests verify proper implementation of the optional status bits in the status word. The following status bits are covered.

1. Terminal Flag.
2. Subsystem Flag.
3. Busy.
4. Broadcast.

RT TO RT TIME OUT.

If the RT is capable of receiving data in an RT to RT transfer, it should time out at approximately 54 microseconds after the receive command.

ILLEGAL COMMANDS.

This test verifies the the optional illegal commands implemented in the RT under test

NOISE REJECTION.

According to MIL-STD-1553B, the terminal shall exhibit a maximum word error rate of one part in 100 million, on all words received by the terminal when operating in the presence of additive white Gaussian noise distributed over a bandwidth of 1.0 KHz to 4.0 MHz at an RMS amplitude of 140 mV. A word error shall include any fault which causes the message error bit to be set in the terminal's status word, or one which causes a terminal not to respond to a valid command. The word error rate is measured with a 2.1V peak-to-peak, line-to-line, input to the terminal. All data words used in this test contain random bit patterns. These bit patterns are unique for each data word in a message and change randomly from message to message. The noise rejection tests the RT for acceptance or rejection criteria as per Table II of MIL-STD-1553B.

SEAFAC'S EXPERIENCES IN VALIDATION TESTING

In the 7 or 8 years of SEAFAC's testing, we have observed that most terminals tested for the first time have at least one error under normal operating conditions. Most terminals also have an improper response to intentionally injected errors. This does not necessarily mean that the system will not communicate under normal circumstances, since most terminals have at least the basic command/response functions working well enough to support communication. However, in order to achieve standardization among terminals these sometimes apparently minor problems must be eliminated.

We have seen terminals that have peculiar functions which may or may not be allowed in the Standard, such as transmission rate sensitivity, partially updated data buffers, and over use of the busy bit. These functions may meet the requirements of the Standard or the specification to which the terminal is designed, but the user and system integrator should be aware of any peculiarities such as these for all the terminals in the system.

In the past two years we have seen a slow but steady improvement in the quality of terminals coming into SEAFAC for testing. This is probably due in part to the advances in LSI and hybrid devices for MIL-STD-1553B, the distribution of the SEAFAC TEST PLAN, and the general increased knowledge and understanding in the industrial community on the characteristics of MIL-STD-1553B. We still continue to find the same problem areas for those devices not using LSI parts or for a "first 1553 design" device. These problem areas are listed below:

1. Non-compliance with the Standard. Some people still feel that the Standard is only a guideline and should be optimized for their application. We can agree that in some cases there are changes to the Standard that could be made to increase the throughput, decrease the cost, and in general modify the subsystem to be technically better for that particular application. If this were allowed then we would no longer have a Standard.

2. Misinterpretations of the standard. MIL-STD-1553B is very clear and specific about most of the requirements. However, a misinterpretation occasionally still occurs. For instance, a question often arises in the area of bus switching, where a terminal under test is receiving a message on one bus. Some misinterpret the Standard to say that there are only certain times during this message transmission when they are required to receive a command on the opposite bus. This is incorrect. The terminal must be capable of receiving a valid, legal command on the opposite bus at any time during the original message transmission.

3. Peculiar Functions. There are many functions that are either left as options in the Standard or are beyond its scope. In these areas where complete guidance is not given, some very peculiar and often potentially disastrous conditions do sometimes occur. For example, a MIL-STD-1553B CRT display terminal that was designed for a certain transmission rate may blank out if the proper update rate is not maintained.

4. Incomplete Testing. Some people still believe that if a terminal will communicate in a system with no problems, then the terminal must be compliant with the requirements of the Standard. This is not true. In order to gain the full benefits from a standard such as MIL-STD-1553B, each terminal should be tested thoroughly to be fully compliant with the requirements of the Standard.

The solution to these problems takes much time and effort from both SEAFAC and all the implementors of MIL-STD-1553B. We are trying to attack these problems from several different angles using as many tools as we can, some of which are listed below:

1. We are recommending the use of the Multiplex Application Handbook which gives design examples, states lessons learned and explains all aspects of MIL-STD-1553.

2. We are encouraging the development and use of LSI and hybrid components. Once these components have been tested and their compliance with the Standard verified, then the problems of terminals that use these components should be greatly reduced.

3. We are developing a finalized Test Plan with the current preliminary SEAFAC Test Plan as a basis and inputs from the SAE-AE9 Test Plan Task Group. The Test Plan explicitly defines what the Air Force considers to be pass/fail criteria for compliance with the Standard.

FUTURE ROLE OF SEAFAC

AUTOMATION OF TESTING

It should come as no surprise that after 7 or 8 years of time consuming and laborous testing SEAFAC is attempting to automate its validation facility. The time and manpower requirements for the present manual testing are becoming a real burden on the small group of engineers at SEAFAC. Because of the increased popularity in MIL-STD-1553B, there has been a corresponding increase in requests for testing at our facility. We are also attempting to train engineers in all aspects of MIL-STD-1553B to provide support to different Air Force program offices. The complicated and tedious testing requirements tends to over extend the training period for these engineers.

An automated test facility that will speed up the time and effort required for validation testing is the goal for the automation effort. Once this is satisfactorily demonstrated as a useful tool, other companies can take advantage of our effort and build their own automated testing facility. By certifying a company's testing facility and having that company test all of their future terminals, we will multiply our own engineering capabilities many times. The key is to provide a cost effective, well documented automated system to perform the very complicated and detailed tests.

This automation task is not a trivial matter and can be broken down into at least 3 phases which are listed below:

Phase I - Hardware specification, design, implementation, and delivery. Consolidate all the required hardware items necessary for the tests into one test station.

Phase II - Some automated assistance in setting up the different tests especially in the protocol section. The interpretation of the terminal's response will still be done by the engineer performing the tests. A detailed test procedure will also be provided to accelerate the learning curve for new engineers performing the tests.

Phase III - Full Software support for a automated test station. The different hardware test equipment will be interconnected with the IEEE-488 instrumentation bus to a PDP-11/34 computer. Now even the electrical section of the test plan can be set up and run automatically. The PDP-11 will be capable of interpreting the terminals response and determining PASS or FAIL according to the Standard and the Test Plan.

The tasks for Phase I have all been completed. All of the hardware items have been integrated into one test station and all of the tests in the Test Plan can be performed using this test station. Table 1 lists the individual pieces of equipment and Figure 1 shows our physical layout of the equipment. We are presently on the final documentation stages of Phase II. Most of the protocol tests have been programmed into our LORAL SBA-100 Bus Tester. The bus tester can now at least set up and run these tests, but the interpretation of the response must be done by the operator.

The tasks in Phase III present many challenges since we have to automatically control not only the SBA-100 Bus Tester but also all the other pieces of equipment that are necessary to perform the electrical tests. The most difficult job is to program the PDP-11/34 computer to interpret the measurements and responses of the terminal and issue a Pass/Fail decision on these results. The Standard allows just enough flexibility and options to make this job difficult, but possible. Also, the trigger, capture, and stack capabilities of our MIL-STD-1553B bus tester do not allow flexible monitoring and interpretation by a host computer. Nevertheless, these problems can and will be overcome.

SERVICES OF SEAFAC

The testing services provided by SEAFAC in the areas of technical consulting and testing have been in heavy demand in recent years. This may be due to the fact that we do not charge for the testing, the use of our equipment, our test time, or the completed test report preparation. In 1982 a totally free service is almost unheard of. Whatever the reason, the demand for our testing support has almost exceeded the capabilities of our single test station and limited manpower. This has caused SEAFAC to set up a priority on scheduled tests which are listed below:

1. System Program Office (SPO) support
 - a. ASD programs
 - b. Other Air Force programs
 - c. Other DOD programs
2. LSI and Hybrid device for the OEM market
3. All other requests

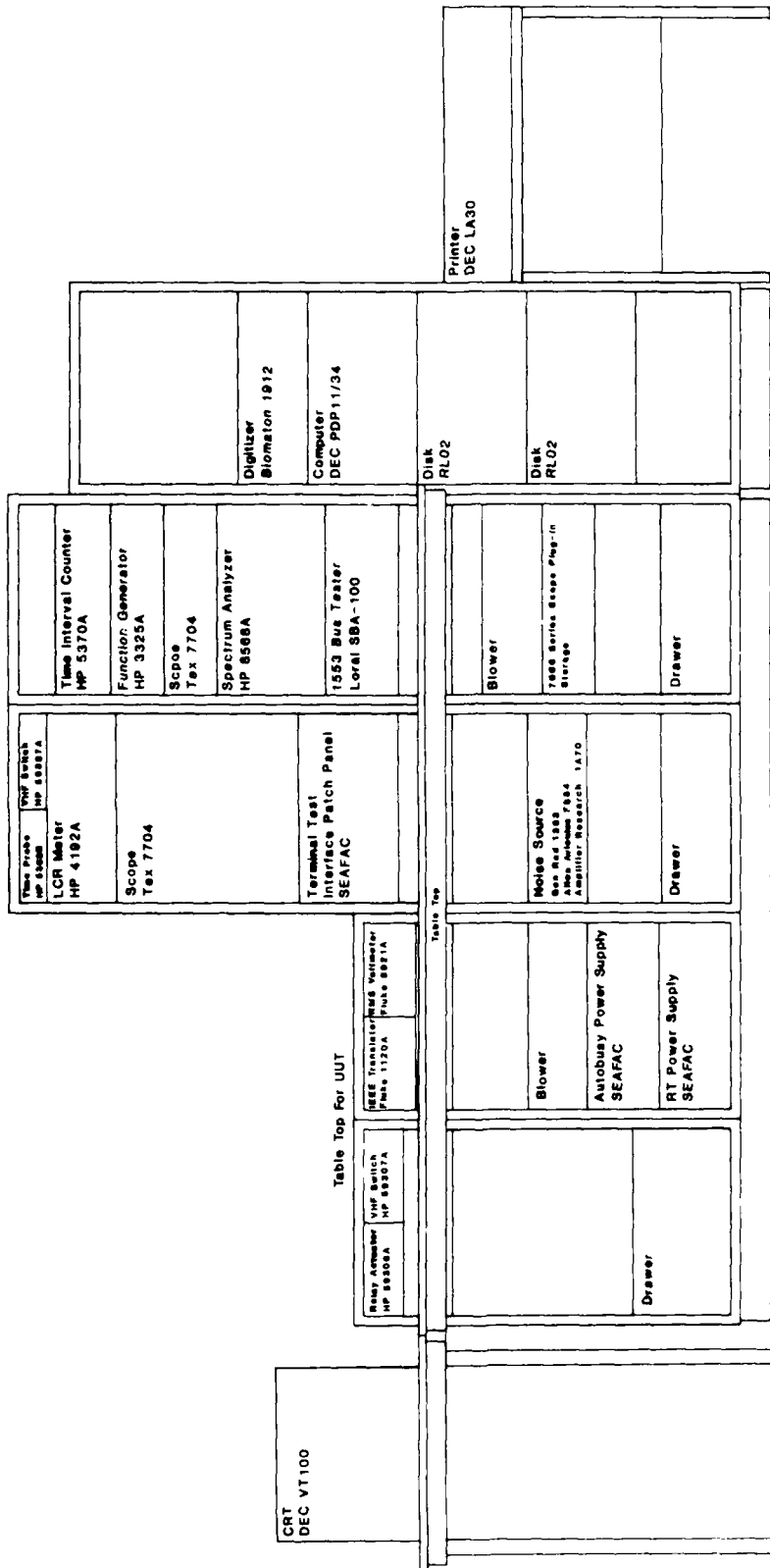
In addition to our continued testing support, we are developing a final version of the SEAFAC Test Plan with inputs from the SAE-AE9 Test Plan Task Group. This will eventually cover both Remote Terminal and Bus Controller test plans for 1st Article, Production, and Acceptance Tests. Our eventual goal is to use this test plan as a tool along with the experience of our automated test station to certify other such facilities in the industrial community.

TABLE 1

MAIN EQUIPMENT PARTS LIST FOR AUTOMATED TEST STATION

RELAY ACTUATOR, HP 59306A
VFH SWITCH, HP 59307A
RMS VOLTMETER, FLUKE 8921A
IEEE 488 TRANSLATOR, FLUKE 1120A
NOISE SOURCE, GEN RAD 1383
NOISE AMPLIFIER, AMPLIFIER RESEARCH 1A70
NOISE FILTER, ALLEN AVIONICS F884
AUTOBUSY POWER SUPPLY, SEAFAC
RT POWER SUPPLY, SEAFAC
TIME INTERVAL PROBE, HP 5363B
VFH SWITCH, HP 59307A
TIME INTERVAL COUNTER, HP 5370A
SCOPE, TEX 7704
IMPEDANCE ANALYZER (LCRMETER), HP 4192A
TERMINAL TEST INTERFACE PATCH PANEL, SEAFAC
SPECTRUM ANALYZER, HP 8568A
DIGITIZER, BIOMATON 1912
SCOPE, TEX 7704
3325A FUNCTION GENERATOR, HP 3325A
1553 BUS TESTER, CONIC SBA-100
7000 SERIES SCOPE PLUG-IN STORAGE
COMPUTER, DEC PDP 11/34
DISKS, RL02
CRT, DEC VT100
PRINTER, DEC LA30

Autobusy



Fold Out of Front View

FIGURE 1

REFERENCES

1. Charles Gifford, "MIL-STD-1553B - MULTIPLEX DATA BUS STATUS",
Proceedings of AFSC Standardization Conference, 1980
2. MIL-STD-1553 MULTIPLEX APPLICATIONS HANDBOOK, Final Technical
Report, Air Force Systems Command, Aeronautical Systems
Division, ENASD, Wright Patterson Air Force Base, 1980
3. Nancy L. Esken and Anthony L. Haley, "1553 BUS TESTERS",
Proceedings of the 2nd AFSC Multiplex Data Bus Conference, 1978

BIOGRAPHICAL SKETCH

DUANE J. THORPE

Duane J. Thorpe is an Electronics Engineer with Systems Engineering Avionics Facility (SEAFAC), Aeronautical Systems Division, Wright Patterson Air Force Base, Dayton, Ohio. He received a B.S. degree in Electrical Engineering from West Virginia University in 1977. Previously he was an Electronic Design engineer in the area of F16 Flight Simulators and real time 1553 data monitoring and storage at Simulation Technology Incorporated, Dayton, Ohio. He is presently on the SAE/AE9 Test Plan Task Group and is investigating methods for automating the 1553 Verification Testing performed at SEAFAC.

BIOGRAPHICAL SKETCH

KUMAR V. VAKKALANKA

Kumar V. Vakkalanka is a Systems Engineer with Systems Engineering Avionics Facility (SEAFAC), Aeronautical Systems Division, Wright Patterson Air Force Base, Ohio. He received an M.S. degree in Electrical Engineering from University of Pennsylvania, Philadelphia, Pa., in 1966. Previously he was an Assistant Professor at Sinclair Community College. He also worked as a Senior Field Service Engineer for Technology Scientific Services Inc, a Division of Technology Inc., and provided technical support to Flight Dynamics Laboratory, Wright Patterson Air Force Base, Dayton, Ohio.

MIL-STD-1553: TESTING AND TEST EQUIPMENT

Leroy Earhart

Test Systems, Inc.
217 W. Paltaire
Phoenix, Arizona
(602) 861-1010

Mr. Earhart is founder and President of Test Systems, Inc. His primary responsibility is marketing and applications of test equipment for MIL-STD-1553. He participates in an engineering course on multiplexing in which he discusses test requirements, test methods, test equipment for multiplex terminals and systems designed to MIL-STD-1553.

Formerly, Mr. Earhart was employed for Sperry Flight Systems as a design engineer. He was involved in the design of the Multiplex Remote Terminal Unit (MRTU) for the AAH, and the Multiplex-Demultiplex (MDM) units for the Space Shuttle. In both of these projects he designed test equipment to support his design work.

Mr. Earhart received both his BSEE and MSEE from the University of Arizona.

Abstract

In the last few years there has been significant increase in the application of MIL-STD-1553. This has generated the need for better understanding of all aspects of testing multiplex hardware and systems. This paper begins with a brief discussion of the philosophy of testing. It highlights the compromise involved in end-item testing. The various phases of testing are discussed. There is also a detailed discussion of the test requirements necessary to verify compliance with MIL-STD-1553B. The functional requirements of special purpose test equipment for multiplex hardware are described. The paper ends with an overview of commercially available MIL-STD-1553 test equipment.

I PHILOSOPHY OF TESTING

The purpose for testing is to determine the quality or the proper functioning of an item. In a classroom setting, a test is used to measure a student's knowledge, but it might also be a measure of the instructor's ability to convey information. When we test electronic equipment we can get it to work as designed but not necessarily as desired (the

designed unit does not always meet the requirements of the specification). Testing may be a measure of the performance of the designer.

When we test a piece of equipment, we want to provide a specified level of confidence that the unit is functioning properly. As the amount of testing is increased, we asymptotically approach the 100% level of confidence. To reach the 100% level of confidence, all permutations and combinations would have to be tested, which may require an infinite amount of testing. Since testing takes time and costs money, we generally have to compromise on the amount of testing. Frequently, the buyer wants a lot of testing and has to settle for less to keep the costs down. The seller, or manufacturer, wants to minimize testing to keep the costs down, but is forced to increase testing and increase his sales price to cover it. The manufacturer also needs to do enough testing to minimize equipment coming back for repair. If he does more testing than necessary, it is costing him extra money in testing. If he doesn't do enough testing, it will cost him extra in repairing returned equipment. Therefore, he must make a compromise, or reach a balance in the amount of testing to be done.

There are two aspects of testing equipment that communicate on the multiplex data bus. First, the 1553 interface to the data bus must be tested, and then the rest of the unit is tested through the 1553 interface. This paper focuses primarily on testing the 1553 interface. Since MIL-STD-1553 requires that words and messages be validated, it is necessary to test with injected errors in all phases of testing to adequately test the 1553 interface.

II PHASES OF TESTING

Some of the phases of testing that can be identified are development testing, design verification, system testing, production testing, and field testing. Each of these phases will be briefly discussed.

Development Testing

Development testing begins with testing breadboards of circuits or modules to provide the designer with a clearer understanding of their operation and capabilities. This helps to minimize costly redesigns due to misunderstandings. The development testing continues with the testing of the bus interface and then the subsystem. The subsystem may function on the data bus as a remote terminal, a bus controller, or a bus monitor. Development testing includes testing circuit operating margins and testing to the tolerance limits.

Design Verification

For design verification, generally a preproduction unit is thoroughly tested to verify that the unit satisfies the requirements of MIL-STD-1553 and the system specification. The unit would be tested over the specified environment range. The design verification testing is generally the most extensive phase of testing.

System Testing

Facilities used for system testing are referred to as System Hot Bench, System Integration Lab, or System Development Lab. Facilities normally provide for Bus Controller simulation and multiple Remote Terminal simulation. Bus Monitors which provide for data recording are used for collecting data for off-line data reduction. Primary concerns for system testing are hardware interface verification and software validation. All default conditions and system margins should be checked.

Production Testing

Production testing can include both in-process testing and end-item testing. Here it is assumed that the design satisfies the requirements of the standard, and the equipment specification and the tests are performed to verify that all of the circuitry is functioning properly. Even though there is in-process testing, it is still essential to do complete testing for end-item or acceptance testing. This complete testing requires testing with injected errors to verify that the error detection or message validation circuits are functioning properly.

Field Testing

Field testing at the system level involves fault isolation to a line replaceable unit. For trouble-shooting and repair it is necessary to have bus controller testers and/or remote terminal testers. Field testing philosophy should also consider complete periodic testing of systems to detect failures in the error detection or message validation circuits of the various subsystems (remote terminals, bus monitors, and bus controllers). After the 1553 bus interface is tested, then the rest of the subsystem is tested through the 1553 interface.

IN ALL PHASES OF TESTING IT IS NECESSARY TO TEST WITH INJECTED ERRORS TO VERIFY THE OPERATION OF THE MESSAGE VALIDATION AND ERROR DETECTION CIRCUITS.

III TEST REQUIREMENTS

Test requirements for terminals can be divided into the two main topics of electrical interface tests and terminal protocol tests. The electrical interface tests apply to all types of terminals, but the terminal protocol tests are a function of the type of terminal being tested. All of the electrical interface tests and the terminal protocol tests should be run on each of the buses when there are redundant buses.

Electrical Interface Tests

The electrical interface tests can be divided into three parts. These are input tests, output tests, and tests to determine the isolation between redundant buses. The specifications given in the standard define the requirements at the connector inputs of the terminal to be tested. These points are illustrated in Figure 1 as point A for the direct coupled terminal (short stub) and point B for the transformer coupled terminal (long stub). It is important to note that the requirements for a terminal are for that terminal by itself and are not tested or measured in the system where they would be dependent on other system elements. Seven input tests for the electrical interface will be briefly discussed.

The input polarity is tested by determining that the terminal responds appropriately to a word with a positive sync.

All terminals are to respond to Manchester signals with peak-to-peak amplitudes in the range of 0.86 volts to 14.0 volts for transformer coupled stubs or 1.20 volts to 20.0 volts for direct coupled stubs. Terminals are not to respond to signals with peak-to-peak amplitudes less than 0.20 volts for the transformer coupled stubs or 0.28 volts for the direct coupled stubs. The response to signals with peak-to-peak amplitudes in the range of 0.20 volts to 0.86 volts for transformer coupled stubs or 0.28 volts to 1.20 volts for direct coupled stubs is indeterminant. It is sometimes desirable to determine the actual threshold voltage for a terminal by measuring the peak-to-peak amplitude of the signal where the response of the terminal is intermittent.

The input impedance of a terminal is specified to be a minimum of 1000 ohms for transformer coupled stubs and 2000 ohms for the direct coupled stubs. This input impedance is specified over the frequency range of 75.0 kHz to 1.0 MHz. The best way to measure the input impedance is with an impedance meter.

The common mode rejection specification for a terminal requires that the performance of a terminal should not be

degraded when a common mode voltage is imposed on the input signal. The common mode voltage range is from direct current to 2.0 MHz with amplitudes equal to or less than ± 10.0 volts, line-to-ground. This test can be run for a remote terminal with a configuration as shown in Figure 2. The bus tester would repeatedly send receive messages to the remote terminal under test and verify that the terminal received the data by monitoring the status response. During the transmissions, the generator output would be varied over the frequency and amplitude range. If the remote terminal fails to respond properly at any time due to the common mode voltage, the test is not passed.

The zero crossing deviation specification requires that a terminal be capable of receiving and operating with input signal waveforms with a maximum deviation in the zero crossing from the ideal of ± 150 ns.

The terminal is to be capable of operating with input signals with transmission rates from .999 to 1.001 Megabit per second (long term stability 1.0 Mb/s $\pm 0.1\%$). It is best to test the terminal with the longest message it is capable of receiving.

The noise rejection specification requires that a terminal exhibit a maximum word error rate of one part in 10^7 when operating in the presence of additive white Gaussian noise distributed over a band width of 1.0 KHz to 4.0 MHz. The band limited RMS amplitude of the noise is 140 mV for transformer coupled stubs and 200 mV for direct coupled stubs. The peak-to-peak signal amplitude is 2.1 V for transformer coupled stubs and 3.0 V for direct coupled stubs. All data words used in the test are to contain random bit patterns. The bit patterns are to be unique for each word in a message and are to change randomly from message to message. This noise test is run continuously until the number of words received by the terminal exceeds the required number for acceptance or is less than the required number for rejection for a particular number of errors. The acceptance/rejection criteria is specified in Table II in the standard. In this test, message errors are normally detected and it is assumed that only one word in the message is in error, independent of the number of words in the message (up to 33 words). To minimize the probability of the words to the bus tester being "garbled", the test can be run with gated noise where the noise source is turned on while the terminal under test is transmitting. A typical test set-up for the noise rejection test is illustrated in Figure 3.

Ten output tests for the electrical interface will be briefly discussed. Measurements for the output tests are made at the connector of the terminal under test in the configuration illustrated in Figure 4. The load resistor, R_L , is

specified as 70 ohms for a transformer coupled stub, and 35 ohms for a direct coupled stub.

The output polarity is tested by verifying that the terminal transmits a word with a positive sync when it is caused to transmit a command word or a status word.

The amplitude of the transmitted signal from a terminal is to be within the range of 18.0 volts to 27.0 volts for the transformer coupled stubs and 6.0 volts to 9.0 volts for the direct coupled stubs. The amplitude is specified as peak-to-peak, line-to-line.

The rise and fall times of the transmitted waveform is specified to be from 100 ns to 300 ns when measured from the levels of 10% to 90% of the full waveform peak-to-peak, line-to-line voltage.

The zero crossing deviations of the transmitted signal are to be less than or equal to ± 25 ns from the ideal crossing points when measured with respect to the previous zero crossing.

The overshoot and ringing on the transmitted signal is not to exceed ± 900 mV for the transformer coupled stubs or ± 300 mV peak for the direct coupled stubs.

The output noise from a transmitter when a terminal is not transmitting is to be less than 14.0 mV RMS for the transformer coupled stubs or 5.0 mV RMS for the direct coupled stubs. It is most convenient to measure the output noise with a differential true RMS voltmeter.

The output symmetry or tailoff, is tested with the terminal under test transmitting the maximum number of words it is designed to transmit (up to 33). The maximum peak-to-peak voltage measured after 2.5 μ s following the mid-bit transition of the parity bit of the last word is required to be less than or equal to ± 250.0 mV for the transformer coupled stubs or ± 90.0 mV for the direct coupled stubs. This test is to be run six times with each word in a contiguous block of words having the same bit pattern. The six word contents that are to be used are 8000_{16} , $7FFF_{16}$, 0000_{16} , 5555_{16} , and $AAAA_{16}$. When running this test for either bus controllers or remote terminals it may not be practical to force the first word (command or status) to have the required bit pattern.

The transmitted signal must be checked to verify that both polarities of word sync and both logic one and logic zero bit encoding conforms to the requirements in the standard.

The word length should be checked to verify that the words are 20 bit-times long. This can be checked by measuring

the time from the mid-sync transition to the mid-bit transition to the mid-bit transition of the parity bit which should be 18 us.

The transmission bit rate or clock stability is most easily tested by measuring the clock. If the clock is not accessible, the transmission bit rate can be checked with the terminal transmitting the maximum number of words it designed to transmit, and measuring the time from the mid-sync transition of the first word to the mid-bit transition of the parity bit of the last word. This time should be $[(20 \times \text{Number of Words}) - 2]$ us. The accuracy and long-term stability is required to be within ± 0.1 percent, and the short term stability is to be within ± 0.01 percent.

The isolation between redundant buses is to be a minimum of 45 dB. The isolation requirement is given as the ratio in dB between the output voltage on the active bus and the output voltage on the inactive bus. The output voltage for the data buses can be measured with a RMS voltmeter.

Terminal Protocol Tests

The terminal protocol tests will be discussed in two parts: remote terminal tests and bus controller tests. Bus monitor tests could be considered a subset of the remote terminal tests and will not be discussed here. When testing the terminal protocol, it is important to know what the terminal will do for all conditions. Some tests are run to characterize the terminal rather than to verify compliance with the standard or system specification.

The first step in testing the remote terminal protocol is to verify that the remote terminal responds properly for all of the legal (valid) information transfer formats. The three basic information transfer formats that are required are BC to RT, RT to BC, and RT to RT. These should be tested with all subaddresses and data word counts that have been implemented in the remote terminal. All mode operations that have been implemented should be tested. This may include Mode Commands without data, Transmit Mode Commands with a data word, and Receive Mode Commands with a data word. If broadcast has been implemented in the remote terminal, the broadcast information transfers should be tested.

The unique terminal address for a remote terminal is checked by sending commands with the RT Address bits set to all 31 combinations (0 to 30 decimal) and verifying that the remote terminal responds to only its own assigned address. If the assigned address is programmable, the test should be run for all possible combinations of assigned addresses. The response to broadcast commands (RT Address 31) also needs to be checked.

The remote terminal is required to respond to a command within 4.0 to 12.0 us. This response time is measured from the mid-bit transition of the parity bit of the last word (command or data) to the mid-sync transition in the status response. This time corresponds to a dead time on the bus of 2.0 to 10.0 us.

The remote terminal should be tested for its ability to respond to superseding commands. The intermessage gap time of the second command on the same bus is to be a minimum of 4.0 us and the command is not to occur during the response time of the remote terminal or while the remote terminal is transmitting. A second command on the alternate bus may occur at any time. A second valid command to a remote terminal is to take precedence over the previous command.

The remote terminal response to illegal commands must be specified in the equipment specification. The standard has left it optional as to whether or not the remote terminal is to monitor for illegal commands. An illegal command is a valid command that specifies an operation that has not been implemented in the remote terminal.

A remote terminal is not to respond to invalid commands. An invalid command is a command with any of the following errors:

- Sync Field Error - either inverted sync or a shift in the mid-sync transition.
- Bit Encoding Error - also referred to as Manchester error or biphasic error.
- Bit Count Error - too many bits or too few bits.
- Parity Error - even parity.

For a valid command with invalid data, the remote terminal is to set the Message Error bit in its status word and suppress the transmission of the status response and not use any of the data received. If any of the data words are invalid, the entire message is invalid. In addition to the four types of errors given above, invalid data may be caused by data words being discontinuous (gaps between words) or by a data word count error (too many or too few words).

When testing the bus controller protocol, it is important to realize that part of the protocol is done in hardware and part is done in software. The first step in testing the bus controller protocol is to verify that the bus controller can issue the desired valid commands and data. The bus controller is never to issue invalid words. The proper processing of normal valid remote terminal responses must be tested.

The bus controller must be tested for its processing of abnormal or invalid remote terminal responses. These abnormal or invalid responses may include the following:

- No Response to Command
- Improper Status Bits
- Word Errors (Sync error, Bit Encoding error, Bit Count error, or Parity error)
- Discontiguous Data Words
- Word Count Errors

The intermessage gap time is the time from the end of one message to the command from the bus controller for the next message. The minimum intermessage gap is 4.0 us. This time is measured from the mid-bit transition of the parity bit in the last word of one message to the mid-sync transition of the command starting the next message.

The minimum no-response time-out for a bus controller is 14.0 us. This is the minimum time the bus controller must wait before it assumes there will not be a response, and it issues another command.

The bus controller should also be tested to verify that it transmits on only one data bus at a time.

IV TEST EQUIPMENT FUNCTIONAL REQUIREMENTS

Special purpose test equipment to support MIL-STD-1553 testing can be described in three categories. There is a need for a remote terminal tester, a bus controller tester, and a bus monitor (non-flight). Some of the requirements or considerations for each of these testers are common. These common requirements will first be discussed, followed by a brief discussion of the requirements for each of the three testers.

Common Requirements

The bus interface is common to all types of testers. The things to consider in the bus interface is whether or not it provides for dual standby redundant operation, and, if so, consider how the two buses are controlled. There should be provision for controlling the amplitude of the transmitted signal to allow for testing the response and no-response voltage levels. The bus interface should also provide for operating on both direct coupled stubs and transformer coupled stubs.

For manual testing it is important to consider the man-machine interface. Ease of operator understanding and use can result in increased productivity.

For automated testing or dynamic simulations, it is important to consider the computer interface. Full parallel, IEEE 488, and RS 232 are common interfaces. The parallel interface is desirable if high data rates are required, or for dynamic simulations.

Remote Terminal Tester

The primary function of the remote terminal tester is to format commands and data for transmission to the remote terminal under test. The tester also needs to receive, validate, and store the responses from the remote terminal under test. To provide the capability for testing the word and message error detection circuitry, the tester needs to be able to generate controlled errors. The types of errors that can be generated may include the following:

- Sync Error (inverted or shifted mid transition)
- Bit Encoding Error
- Parity Error
- Bit Count Error (too few or too many)
- Discontiguous Data (gap)
- Word Count Error (too few or too many)
- Mixed Bus Transmission in Message
- Incomplete Message with Subsequent Command

For testing operating margins in the remote terminal, the tester may provide for controlled variations in transmission bit rate, zero crossing deviation, and intermessage gap time. The tester needs to validate the responses from the remote terminal under test to verify that they conform to the requirements of the standard. The validation checks that the tester makes may include the following:

- Response Time
- Bit Encoding
- Bit Count
- Off Parity

- Transmission Continuity
- Word Count
- Simultaneous Bus Activity

Bus Controller Tester

The primary function of a bus controller tester is to process received commands and data and format status and data responses for transmission. For testing the error detection circuitry the tester needs to be able to inject controlled errors. The controlled errors generated by a bus controller tester may include the following:

- Sync Error (inverted or shifted mid transition)
- Bit Encoding Error
- Parity Error
- Bit Count Error (too many or too few)
- Discontiguous Data (gap)
- Word Count Error (too many or too few)
- No Response (failed bus)
- Wrong Terminal Address in Status
- Error Flags in Status
- Response on Wrong Bus

For testing operating margins in the bus controller, the tester may provide for controlled variations in the transmission bit rate, zero crossing deviation, and response time. The tester needs to validate the commands and data issued by the bus controller to verify that they conform to the requirements of the standard. The validation checks that the bus controller tester makes may include the following:

- Intermessage Gap Time
- Bit Encoding
- Bit Count
- Odd Parity

- Transmission Continuity
- Word Count
- Simultaneous Bus Activity

Bus Monitor

Bus monitors are used during system integration and system testing. The primary function of the bus monitor is to receive data bus traffic and extract selected information. There are two basic types of bus monitors. One basic type of bus monitor does the function of Statistical Analysis, and the other type of bus monitor functions as a Logic Analyzer.

The Statistical Analysis type of bus monitor stores and tabulates information from data bus traffic for statistical purposes. Information may be tabulated in the following ways:

- Active Terminals by Address and Data Bus
- Unique Commands
- Error Conditions

The Logic Analyzer type of bus monitor triggers on a specified event in the data bus traffic and stores a snap-shot of the data bus traffic. The snap-shot of traffic stored may be preceding, centered about, or following the trigger event. This type of monitor may also provide a trigger output for synchronizing other instrumentation.

V OVERVIEW OF AVAILABLE TEST EQUIPMENT

Today there are several companies which make test equipment to support work for MIL-STD-1553. The equipment available ranges from simple manual testers, to strictly computer controlled testers, to testers which provide both manual control and computer control. Testers range in capability from single function testers (ie. remote terminal testers) to multi-function testers. The following Table gives manufacturers and their products for testing MIL-STD-1553 hardware and systems.

MIL-STD-1553 TEST EQUIPMENT

MANUFACTURER	PRODUCTS
SCI SYSTEMS, INC.	BUS CONTROLLER SIMULATOR (BCS 101 A) BCS/REMOTE TERMINAL (BCS/RT 202) BUS SYSTEM TESTER (BST 1100) BUS ACTIVITY SIMULATOR
TEST SYSTEMS, INC.	BUS CONTROLLER/MONITOR (BCM) MULTI-TERMINAL SIMULATOR (MTS)
FAIRCHILD SPACE & ELECTRONICS	DATA BUS MONITOR/CONTROLLER (DBMC TM)
DIGITAL TECHNOLOGY, INC.	MICROPROGRAMMABLE BUS TERMINAL (MBT)
LORAL DATA SYSTEMS	SERIAL BUS ANALYZER (SBA 100)
COMPUTER DATA SYSTEMS	BUS SIMULATOR CARD (53A-553)
SPECTRAL SYSTEMS	MODEL 20 DATA BUS TESTER

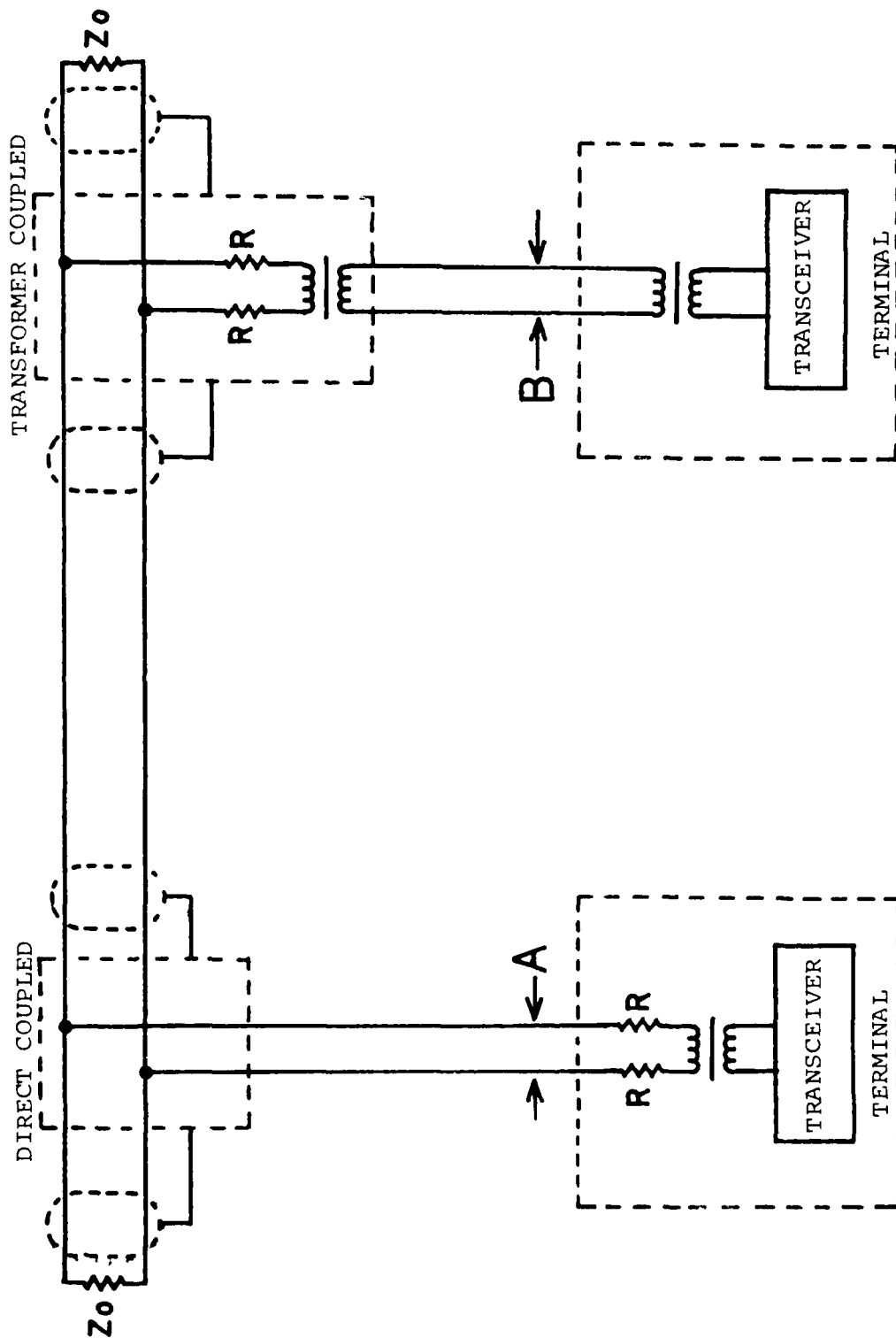


FIGURE 1
CONNECTIONS TO THE DATA BUS

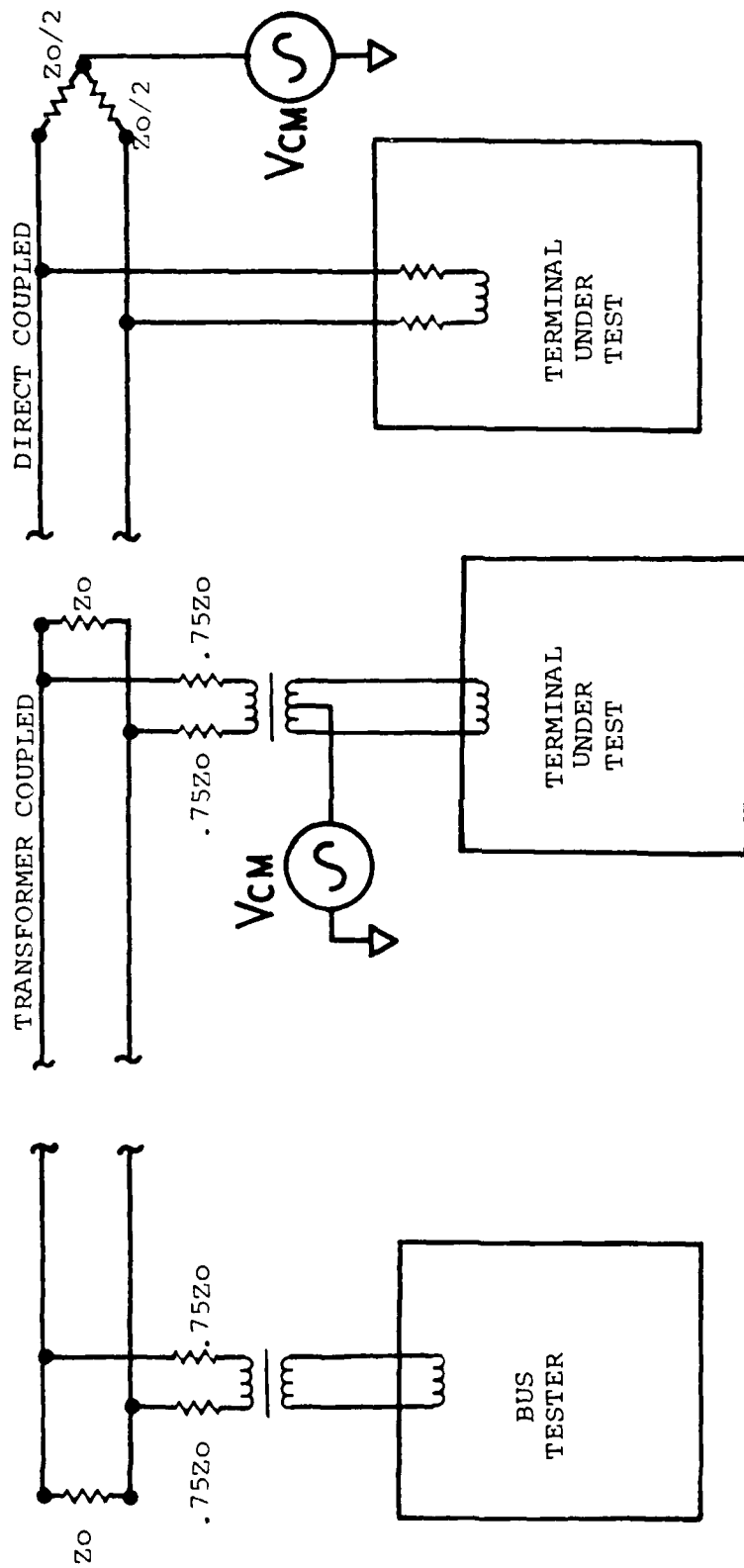


FIGURE 2
CONFIGURATION FOR TESTING COMMON MODE REJECTION

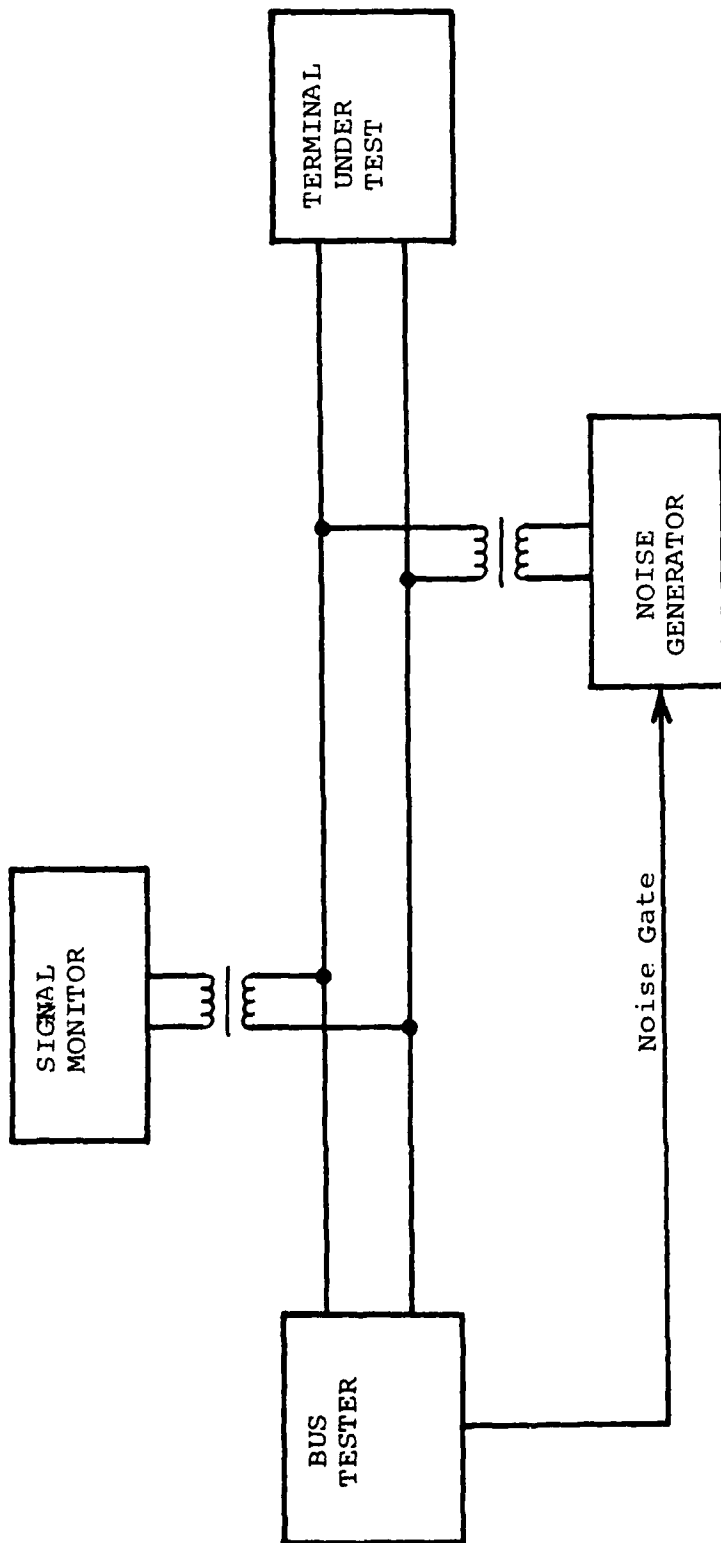
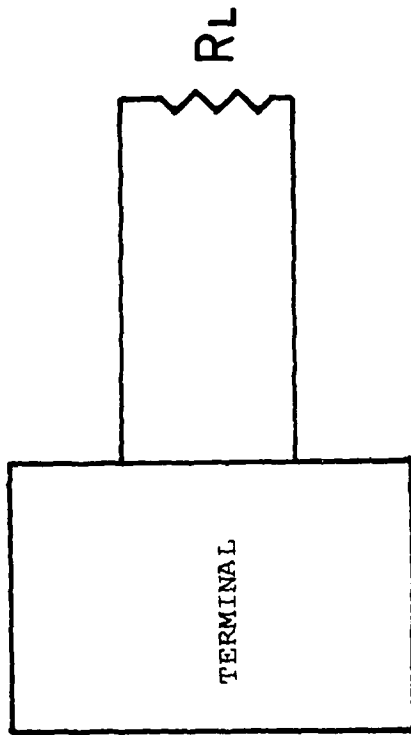


FIGURE 3

TYPICAL SET-UP FOR NOISE REJECTION TEST



TRANSFORMER COUPLED: $R_L = 70$ OHMS

DIRECT COUPLED: $R_L = 35$ OHMS

FIGURE 4

CONFIGURATION FOR TERMINAL OUTPUT TESTS



A COMMON 1553B I/O CHANNEL FOR THE F-16

Stephen Alford

General Dynamics - Fort Worth Division
P.O. Box 748 Fort Worth, Texas 76101
(817) 732-4811 x3195

AUTHOR

Stephen Alford is a 1975 electrical engineering graduate of Rice University. He is now a Software Design Specialist for General Dynamics Fort Worth and is the co-designer and chief programmer of the 1553 bus control software for the current F-16 aircraft. Mr. Alford was lead programmer for F-16 Fire Control Computer software from 1977 to 1980. Since that time he has been coordinating the development of the executive and bus interface software for the four F-16 MSIP subsystems being programmed by General Dynamics. Mr. Alford is an active participant in the MIL-STD-1750A Users Group.

ABSTRACT

The 1980's will see increased standardization in military avionics. MIL-STD-1553 has proven to be an effective means of assuring communications among independently developed avionic subsystems. Future applications of the Air Force standard computer architecture, MIL-STD-1750A, and standard programming language, MIL-STD-1589B, will further decrease the life cycle costs of many systems currently under development.

While MIL-STD-1750A defines a specific CPU architecture, and MIL-STD-1553B defines the method of communication among subsystems, no current Air Force standard defines the I/O channel that links the 1750 processor to the 1553 bus. In order to reduce both the cost of software development and maintenance, General Dynamics has developed a 1553 channel architecture to be applied to all the subsystems being programmed in-house for the F-16 Multi-National Staged Improvement Program (MSIP).

Copyright © 1982 by General Dynamics Corporation
All Rights Reserved

BASIC REQUIREMENTS

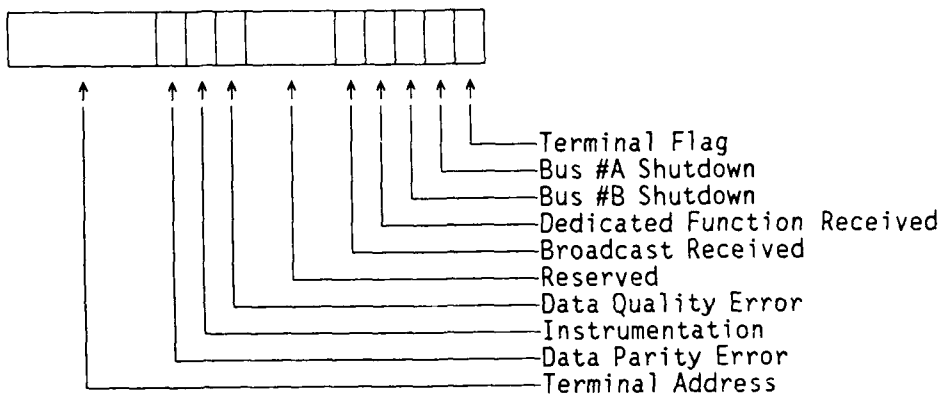
The first and foremost requirement for the MSIP 1553B channel is that it support MIL-STD-1553B. The F-16 interface requires the implementation of the majority of the 1553B standard. (Exceptions are certain mode commands and features such as broadcast commands and dynamic bus control.) An additional requirement is to support communications to equipment already on the current F-16, and scheduled to be included, without modification, on the MSIP system. These systems communicate via 1553-F16, a subset of the original 1974 MIL-STD-1553. Communications among old and new terminals is possible as long as the bus controller interprets the status words from each subsystem according to the proper version of the standard. Figure 1 shows how the status words of the two standards differ.

The channel must support the transfer of time consistent blocks of input and output data. This is easily accomplished by the 1553 controller since the CPU is aware of when bus transactions may be expected to occur. For the asynchronous terminal, this task is much more difficult to accomplish. Added to this is the requirement that normal channel operations require a minimum of intervention from its host computer's CPU. In order to reduce the complexity of the avionic system, bus controllers on the F-16 are incorporated into existing LRU's, rather than into dedicated units. A rudimentary intelligence, incorporated into the channel, can assure the task of 1553 bus controller (or terminal) does not adversely affect the throughput of it's host computer. The power of the host CPU is always present to handle unusual situations, and to assure the flexibility for future system growth. Provisions must therefore be made for interrupting the CPU when required. However, the generation of interrupts should be selectable for each transmission and situation, so that the CPU is not burdened with needless overhead.

A requirement more specific to the F-16 is the support of transaction time-tagging[1]. Each multiplex terminal and controller on the F-16 maintains a 16 bit counter clocked every 64 microseconds. The counter may be loaded or reset by the 1553B controller, and may be read by the host CPU for internal use. Multiplex data time-tagging allows all F-16 avionic subsystems to operate asynchronously while still maintaining a common time referencing system for age sensitive data.

Finally, some additional considerations went into the design of the F-16 channel architecture. The architecture had to be entirely consistent with MIL-STD-1750A and MIL-STD-1589B (the Jovial programming language). In addition, the good and bad characteristics of other 1553 channels were evaluated and lessons learned were applied to the design where possible.

1553/F-16 Terminal Status Word



1553B Terminal Status Word

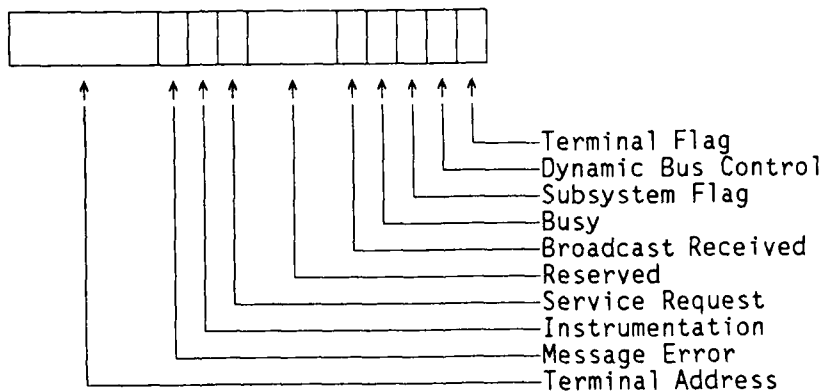


Figure 1

CONTROLLER ARCHITECTURE

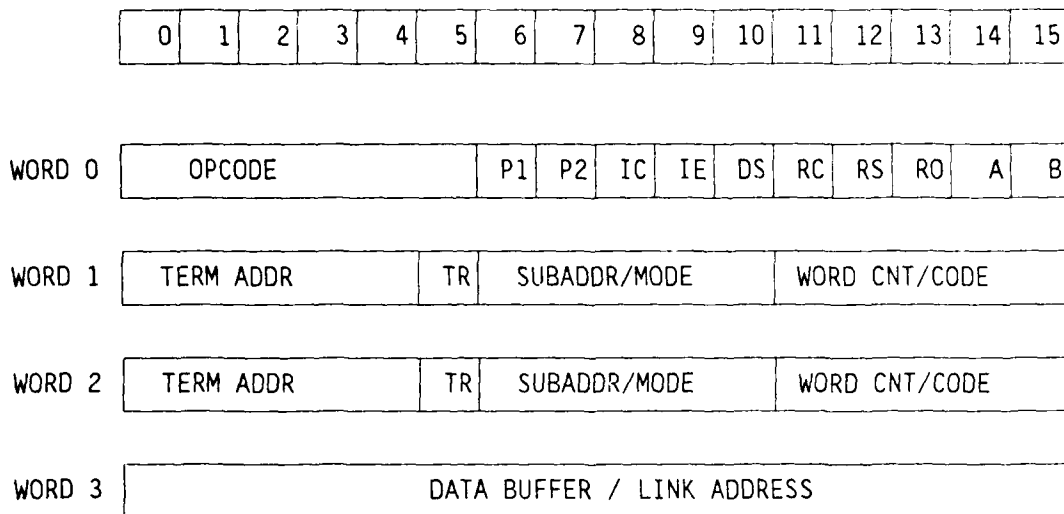
The F-16 controller executes a channel program stored in the main memory of the host computer. Once provided the start address of the program and the controller enable command from the CPU, the I/O channel executes independently from the CPU. All input and output data needed by the channel is obtained from the host computer memory via direct memory access. Each instruction in the I/O channel program is four words long and encompasses a complete 1553 operation. Figure 2 illustrates the basic format of each channel instruction.

Word 0 of each instruction includes an operation code and other required control information. Allocation of the first word entirely to the specification of opcode plus options allowed space for many desirable features. Two bits specify the protocol (1553 or 1553B) of the transmitter and receiver for the operation. Two other bits specify a request for a CPU interrupt on successful completion and a similar request if the transmission fails due to an external error. The channel always interrupts the CPU in the case of an internal error. Additional bits specify that upon an external error, an automatic re-try be attempted on the same or on the alternate bus wire. Automatic transmission re-tries, when combined with external error interrupt control, serve to greatly reduce the overhead placed by the channel on the host computer CPU. For many data blocks, there is no need to involve the CPU in transmission failures. If the re-try of the transmission should also fail, the channel simply proceeds to the next command. Experience on the F-16 has shown that a single re-try resolves almost all 1553 recoverable transmission failures[2].

Word 1 is the first 1553 command word to be transmitted over the bus for the transaction. Word 2 contains the second 1553 command word, required only for terminal to terminal operations. These words contain the transmitter and receiver terminal address, subaddress, and word count. The last word of the command contains the host memory address for data retrieval or storage, or the new channel program address in the case of the branch operation code. A fourth word would not be required if the controller relied on his own terminal address and subaddress, and a subaddress vector table, in order to locate data storage and retrieval areas in main memory. However, the hardware complexity of the I/O channel is reduced by allocating the instruction size as a power of two, and requiring boundary alignment. For this reason, along with the flexibility of each command having an individual data storage address, the fourth word was included in each command.

When the I/O channel does interrupt the host processor, the processor may determine the cause of the interrupt by reading key channel registers via programmed I/O (the MIL-STD-1750A XI0 instruction). The channel is always halted following a CPU interrupt request. The command table address word provides the CPU with the location of the failed command. The interrupt status word (shown in Figure 3) provides the CPU with the cause of the error. Two additional words provide the status words received by the controller from the transmitting and receiving terminals, should they be required by the CPU for its analysis. Table 3 summarizes

CONTROLLER CHANNEL INSTRUCTION FORMAT



- OPCODE - SEE TABLE 1
- P1 - PROTOCOL OF TERMINAL #1
- P2 - PROTOCOL OF TERMINAL #2
- IC - INTERRUPT ON COMPLETE
- IE - INTERRUPT ON ERROR
- DS - CONTROLLER DATA STORAGE DURING TERMINAL
TO TERMINAL TRANSMISSION
- RC - RESET CONTROLLER TIME TAG
- RS - RETRY ON SAME BUS
- RO - RETRY ON OPPOSITE BUS
- A - SELECT BUS "A"
- B - SELECT BUS "B"

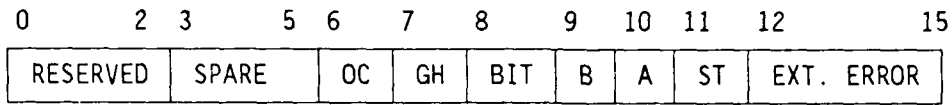
Figure 2

CONTROLLER TRANSACTION OPCODES

<u>OPCODE</u>	<u>DESCRIPTION</u>
000000	Perform a controller to terminal data transaction
000100	Perform a terminal to controller data transaction
001000	Perform a terminal to terminal data transaction
001100	Perform a mode command (with no data word)
010000	Perform a mode command (with controller generated data word)
010100	Perform a mode command (with terminal generated data word)
000001	STOP (do not update command table address)
000010	LINK (set new command table address)
000011	Store time-tag word
xxxxxx	Perform channel built-in-test (optional) (OPCODE selected by vendor)

Table 1

CONTROLLER INTERRUPT STATUS WORD FORMAT



<u>BIT</u>	<u>DESIGNATION</u>	<u>DESCRIPTION</u>
0-2	RESERVED	Set to logic zero
3-5	SPARE	Set to logic zero
6	OC	Operation complete
7	GH	Graceful halt complete
8	BIT	Built-in-test-complete
9	B	Bus B inoperative
10	A	Bus A inoperative
11	ST	Self-test/Built-in-test fail
12-15	EXT. ERROR	External error code (see Table 2)

Figure 3

CONTROLLER EXTERNAL ERROR CODES

<u>CODE</u>	<u>INDICATION</u>	<u>DESCRIPTION</u>
0000	No error	Cause of last interrupt was not an external error
0001	Invalid bus activity (optional)	Uncommanded activity present on bus. Sets interrupt status bits 9 and/or 10
0010	External status unavailable	Expected status word not received or could not be decoded with proper parity
0011	Reserved	Reserved
0100	Status word flag	a received status word had an error indicator set
0101	Terminal replied Busy	A received status word had the busy bit set
0110	Terminal replied Service Request	A received status word had the Service Request bit set
0111-1011	Spare	Spare
1100-1111	Reserved	Reserved

NOTE: Code 0001 has the highest priority.
Code 1111 has the lowest priority.

Table 2

CONTROLLER XIO COMMANDS

<u>OPCODE</u>	<u>VALID STATES</u>	<u>DESCRIPTION</u>
XX00	BC/OFF	Read/Write Channel Control Word
XX01	OFF	Read Interrupt Status Word
XX02	OFF	Read/Write Command Table Word
XX03	OFF	Read Received Status Word #1
XX04	OFF	Read Received Status Word #2
XX05	BC/OFF	Graceful halt Request
XX06	BC/OFF	Read/Write Time Tag Word

NOTES:

XX = Eight bit code specified for each channel by the hardware development specification. Most significant bit shall be a logic one for a Read command and a logic zero for a Write command.

BC/OFF = May be read by CPU in either Bus controller or OFF states.

Table 3

the set of XIO commands used by the CPU to control the 1553B channel in the controller mode.

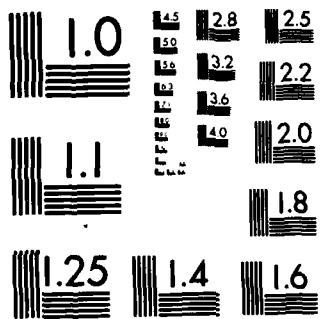
TERMINAL ARCHITECTURE

The 1553 terminal mode is the most unique aspect of the F-16 I/O channel. The primary consideration in designing the 1553 terminal is to provide the CPU with consistent sets of input (and output) data while remaining constantly online to the bus. This must be achieved while the CPU executes its program totally asynchronous to the input and output transmissions of the multiplex controller. This has been a weak point of many previous 1553 terminal architectures. Asynchronous data consistency was previously not supported, or was maintained by interrupting the CPU at the end of each input transmission and allowing the CPU to capture the data in the input buffer. To allow this process sufficient time to take place, the terminal was required to go off-line (accomplished by setting the BUSY bit of the 1553B terminal status word). The CPU reestablished normal terminal mode after the data capture was complete. However, the controller could not send additional transmissions to the terminal during this time interval.

The F-16 approach to this problem is to provide the I/O channel with the addresses of two independent receiving buffers for each subaddress. The channel automatically alternates filling the two buffers with input data. In addition, to support interrupting the CPU while remaining on-line, a sixteen word interrupt queue is provided in computer main memory. The channel accumulates information concerning interrupts in the queue area, so that information from new interrupts does not destroy information from interrupts yet to be processed by the CPU interrupt software.

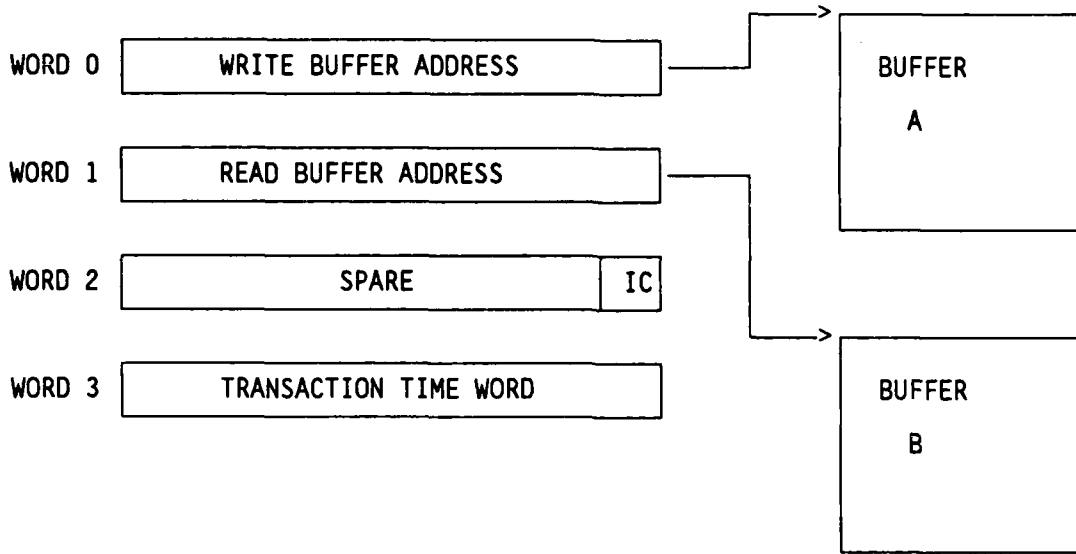
Two major data structures, located in main computer memory, support the terminal mode of the I/O channel. The Subaddress Vector Table is a typical feature of 1553 terminals. MIL-STD-1553B supports 30 input and 30 output messages, with the subaddress and transmit/receive fields of each 1553 command from the controller uniquely identifying each message. Using these fields as a numerical index into the subaddress vector table, the terminal can determine the proper response for each 1553 message. The F-16 terminal vector table contains 64 entries that are each four words long. (Four entries correspond to subaddresses that identify 1553 'mode' transmissions and hence do not identify data transmissions.) Figure 4 illustrates the format of each entry. While 256 words is more storage than has been required by previous 1553 terminals, the added capabilities provided by this structure more than compensate for the added storage requirements.

The first two words of each entry contain pointers to buffers in main computer memory that are to contain input or output data. For 1553 input, the channel stores the incoming data words in the buffer identified by the Write Pointer. At the end of the transaction, if all data words have been received correctly, the terminal swaps the contents of the two pointer



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TERMINAL VECTOR TABLE ENTRY FORMAT



IC - INTERRUPT ON COMPLETE

Figure 4

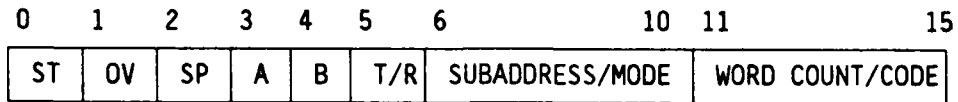
words so that the Read Pointer now identifies the newly received data. The CPU may use the Read Pointer to access the most recently received data totally asynchronous to further channel activity. For output subaddresses, the CPU uses the Write Pointer to fill a buffer with new output data, and then swaps the two pointers. The channel uses the Read Pointer to access the data when the controller requests the transmission of that subaddress.

The third word of each vector table entry contains a CPU interrupt flag. When this flag is set for an input subaddress, the channel will generate a CPU interrupt request after successfully receiving the selected subaddress. Unlike the 1553B controller, this does not cause the terminal to halt operations. The interrupt request is generated following the vector table pointer swap. In order to identify to the CPU which subaddress has been received, the channel stores an entry into the Terminal Interrupt Queue (Figure 5). This 16 word data structure is a first-in, first-out queue managed by both the channel and the CPU. Two pointers in main memory identify the current position in the queue for the channel, and the current position in the queue for CPU processing. Each time the channel adds an entry to the queue, it increments its queue pointer word. After 16 entries, the pointer is set to point again to the first queue word. The channel then checks the CPU's processing pointer in order to halt operations should a queue overflow occur. As each interrupt is processed, the CPU increments its own processing pointer. Using this interrupt queue structure, the F-16 1553B terminal may successfully receive up to 16 (interrupting) input subaddresses in a burst from the controller, while always remaining on-line to the bus. It should be noted that an interrupt is only required for those input subaddresses that require the immediate attention of the terminal's host CPU. The Vector Table pointer swap allows the CPU access to time consistent input blocks without the need for a transaction complete interrupt.

As in the case of the terminal mode, the complexity of the channel is reduced if the size of each vector table entry is a power of two words. To fill the entry to four words, the last word in each vector table entry is used to receive a copy of the terminal's time-tag at the time of the successful reception of the corresponding subaddress. The terminal's host CPU may use this word to determine the relative freshness of its input data.

Table 4 summarizes the set of XIO commands used by the CPU to control the 1553B channel in the terminal mode. Once the channel has been started in the terminal mode, it needs no further XIO communication with the CPU, unless it halts due to an internal (self-test) error.

TERMINAL INTERRUPT QUEUE ENTRY FORMAT



BIT	DESIGNATION	DESCRIPTION
0	ST	Self-test/built-in-test failure
1	OV	Queue overflow
2	SPARE	Set to logic zero
3	A	Transaction performed on bus A
4	B	Transaction performed on bus B
5	T/R	Value of T/R bit used for vector table entry index
6-10	SUBADDRESS	Value of subaddress/mode used for vector table entry index
11-15	WORD COUNT	Value of word count field used for vector table entry index

Figure 5

TERMINAL XIO COMMANDS

<u>OPCODE</u>	<u>VALID STATES</u>	<u>DESCRIPTION</u>
XX00	RT/OFF	Read/Write Channel Control Word
XX01	RT/OFF	Write Channel Status Word
XX02	OFF	Write Vector Table Address Word
XX03	OFF	Write Interrupt Queue Pointer Address
XX04	OFF	Read Bit Word
XX05	OFF	No operation
XX06	RT/OFF	Read/Write Time Tag Word

NOTES:

XX = Eight bit code specified for each channel by the hardware development specification. Most significant bit shall be a logic one for a Read command and a logic zero for a Write command.

RT/OFF = May be read by CPU in either remote terminal or OFF states.

Table 4

EXPERIENCE TO DATE

An engineering prototype of the F-16 1553B I/O channel is now operational in the laboratory under a MIL-STD-1750A host computer. Algorithms to operate the channel in the controller mode have been written and tested, successfully establishing 1553 communications with other F-16 equipment. All of these programs, including the main body of the interrupt handler, are written in Jovial J73. Programs to exercise the terminal mode are still under development. A possible improvement to the channel was noted during the development of its operational software. For many transmissions, the controller external error interrupt is enabled solely so that the CPU may mark the contents of the input data buffer as invalid. This is a task that could easily be performed by the I/O channel itself.

SUMMARY

The F-16 was one of the first major applications of MIL-STD-1553. While numerous 1553 I/O channels have been developed since then, it is the F-16 common I/O channel for MSIP that has solved many of the common problems found in earlier designs. Like MIL-STD-1750A, the F-16 channel is the definition of an architecture, not an implementation. Its implementation is not bound by any level of technology. Perhaps other users of both MIL-STD-1553B and MIL-STD-1750A may find some benefit to an implementation of the F-16 channel for their own applications.

REFERENCES

- 1) B.D.Brumback, "Time-Reference of Data in an Asynchronous Environment", AIAA 4th Digital Avionics Conference, November, 1981.
- 2) D.E.Sundstrom, W.B.Anderson,Jr.,S.A.Alford, "F-16 Multiplex: A System Prospective", AFSC 2nd Multiplex Data Bus Conference, October, 1978.
- 3) MIL-STD-1553B, "Aircraft Internal Time Division Command Response Multiplex Data Bus", 21 September 1978.
- 4) MIL-STD-1750A, "Sixteen-Bit Computer Instruction Set Architecture", 2 July 1980.
- 5) MIL-STD-1589B, "Jovial (J73)", 6 June 1980.
- 6) General Dynamics Requirement 16PP378A, "Serial Digital Interface Command and Control Requirements", 15 June 1981.



MIL-STD-1750

16 BIT INSTRUCTION SET ARCHITECTURE

SESSION CHAIRMAN: Marlin Wagner
Sperry Univac Corp.

MODERATOR: Dr. V. V. Griffith
Department Manager Digital Computer & Software
McDonnell Aircraft Co.

AD-P003 547

MIL-STD-1750A USERS GROUP

C. Ray Turner
The Boeing Company
P.O. Box 3707
Seattle, WA 98124
(206)655-5158

and

Marlin L. Wagner
Sperry Univac Defense Systems
Univac Park
P.O. Box 3525
St. Paul, MN 55165
(612)456-4580

Abstract

The MIL-STD-1750 Users Group was established in August 1979 as a voluntary organization of industry representative to exchange information and status of MIL-STD-1750, and to recommend changes to the standard. This paper is a brief description of the Group, its committees, accomplishments, and future direction. The purpose of the standard is reviewed.

Benefits of MIL-STD-1750

MIL-STD-1750 is a standard for and instruction set architecture ISA. It does not define specific implementation details of a computer.

The standard states, in paragraph 1.4:

" The expected benefits of this standard ISA are the use and re-use of available support software such as compilers and instruction level simulators. Other benefits may also be achieved such as: (a) reduction in total support software gained by the use of the standard ISA for two or more computers in a weapon system, and (b) software development independent of hardware development."(1)

Organization and Membership

The MIL-STD-1750 Users Group was established in August 1979 as a voluntary organization of industry representatives to exchange information and status of MIL-STD-1750, and to recommend changes to the standard. (Appendix A contains the by-laws of the group (2)). The group is organized into committees, and the group generally does not vote on a change to the standard until one of the committees has itself approved it for consideration of the entire group. Any member of the group may present a problem or recommendation to either the group as a whole or to any committee chairman for discussion and possibly eventual vote by the entire group.

The membership of the Users Group, excluding government personnel, is approximately 350 persons. Industry representatives from England regularly attend, and other NATO country representatives also occasionally attend. Figures 1 and 2 depict the growing and changing attendance (2).

At the time the group was formed, the Air Force had released the standard, and was seeking industry comments and recommendations for changes to it. The Air Force recognizes the group as the sole industry body to recommend changes and improvements to the standard. Although the Air Force and other government representatives participate in the committee and group discussion, they do not vote. The Air Force uses a "Control Board" to accept changes or refer them back to the users group. The control board and the users group is part of the control structure which the Air Force has established for MIL-STD-1750. (See Figure 3.) A similar structure exists for MIL-STD-1589, JOVIAL.

The committees are the backbone of the group. The following is a summary of the function of the committees.

Standards - To interpret and clarify definitions and descriptions appearing in MIL-STD-1750; to assess the scope and applicability of the standard.

Architecture - To assess the value and impact of proposed architecture modifications or extensions to the standard.

Verification - To address issues related to verifying and certifying MIL-STD-1750 hardware implementation.

Software Tools - To act as an information exchange to MIL-STD-1750 related software tools, and to assess the need for MIL-STD-1750 support tools.

Liaison - To retain communication and coordination with other related standardization groups.

The group has had meetings three or four times a year, each for about two days. The committees elect their own committee officers and make committee reports to the full Users Group at each meeting.

The MIL-STD-1750 Users Group is governed by its adopted set of by-laws and conducts its business meetings generally by Roberts Rules. Voting procedures are set forth by the by-laws. It has already been noted that government personnel do not participate in the voting of User Group issues. No company, including all of its divisions, may be represented by more than five voting members at any meeting.

Accomplishments

The principal accomplishment of the group during its first two years was the revisions to the standard documented as MIL-STD-1750A and as Notice 1. At each of the meetings in those first two years, the status of changes was well known to all of industry. This is also an accomplishment since any company wishing to implement the standard could always know which changes were being studied by committees, or were already accepted by the control board.

The work load of the group may be judged by the number of substantive changes to the original MIL-STD-1750. The following is the best available record of the changes accepted by both the Users Group and the Control Board/Control Agent for changes from MIL-STD-1750 to MIL-STD-1750A.

Architecture	-	8
Standards	-	38
Verification	-	1
Editorial	-	25

MIL-STD-1750A was completed in March 1980 and published on 2 July 1980. In addition to these, several more changes were made to MIL-STD-1750A which were incorporated in Notice 1. The work was completed in November 1981 and published 21 May 1982.

Future Direction

The Users Group will be considering what changes to MIL-STD-1750A are justified in light of technology advancements and lessons learned. The Air Force has established a policy that the standard will not be changed until 1985 to provide needed stability. New applications are being considered that will require revision to the standard, for example, multiple processors.

References

- (1) "SIXTEEN-BIT COMPUTER INSTRUCTION SET ARCHITECTURE", MIL-STD-1750A (USAF) NOTICE 1, 21 May 1982.
- (2) MIL-STD-1750A USERS GROUP MEETING MINUTES, Meeting Number 1 through Number 11.

Author's Biographical Sketches

C. Ray Turner, The Boeing Company

PROFESSIONAL EXPERIENCE - Mr. Turner has over 25 years experience at Boeing in aircraft and missile programs. Currently he is Technology Manager in the Boeing Commercial Aircraft Company's 707/727/737 Division. He is responsible for Flight Management Sensors, Performance Data Computers, and Flight Management Computers. Previously, he held engineering management positions in the C-14/C-X cargo aircraft programs as manager of Flight Management System Design and Integration, and Technology Manager. Prior to C-14/C-X program assignments, Mr. Turner held management positions in Software Research, Systems Analysis, Test Engineering, and Human Factors Engineering. Mr. Turner is currently serving as chairman of the MIL-STD-1750A Users Group.

EDUCATION - B.A., Psychology 1950, Seattle Pacific College
M.S., Experimental Psychology, 1964, University of Washington
Many technical and management seminars

Marlin L. Wagner, Sperry Univac

PROFESSIONAL EXPERIENCE - Mr. Wagner has over 22 years of experience in digital processing. He joined Sperry Univac in 1960 as a digital test engineer on the Naval Tactical Data Systems and has been involved in the development and application of digital computers in various command and control, air traffic control and avionic weapon and navigation systems. In addition, he has direct programming experience on 12 Sperry Univac digital processors. Since 1978 Mr. Wagner has been involved in the development and implementation of MIL-STD-1750 for the AN/AYK-15A and MIL-STD-1750A for Sperry Univac 1625/1630 processor and other applications. In August 1980, Mr. Wagner was elected Vice Chairman of the MIL-STD-1750A Instruction Set Architecture Users Group and is currently serving his second two-year term in that position.

EDUCATION - B.S. (Electrical Engineer), South Dakota State College, 1960.

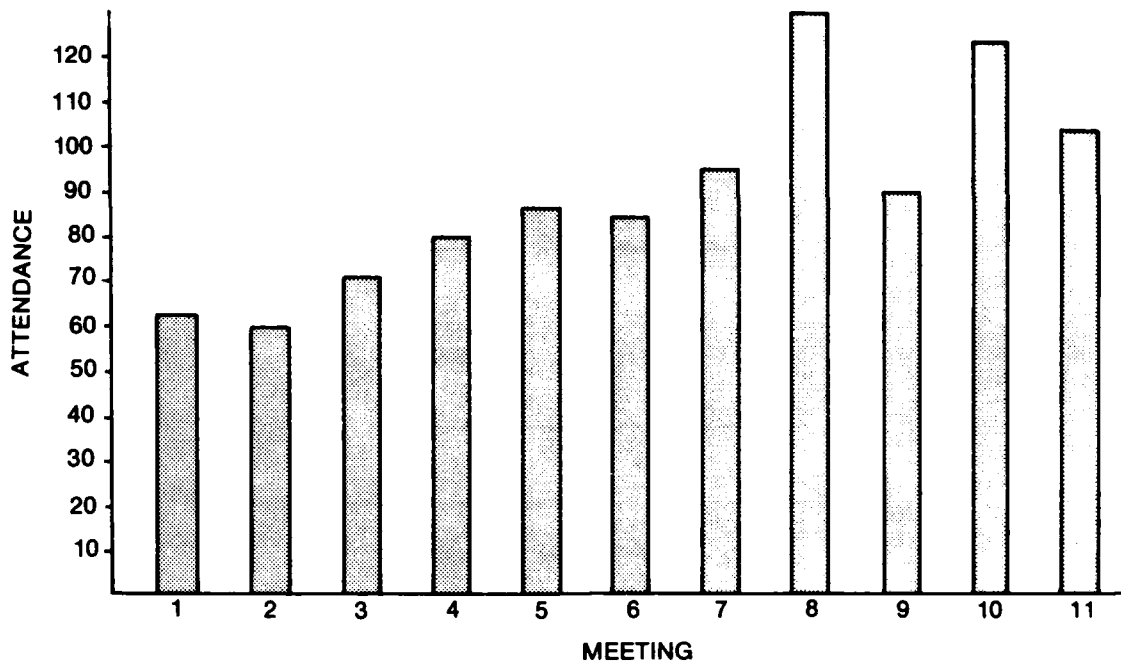


FIGURE 1. INDUSTRY ATTENDANCE

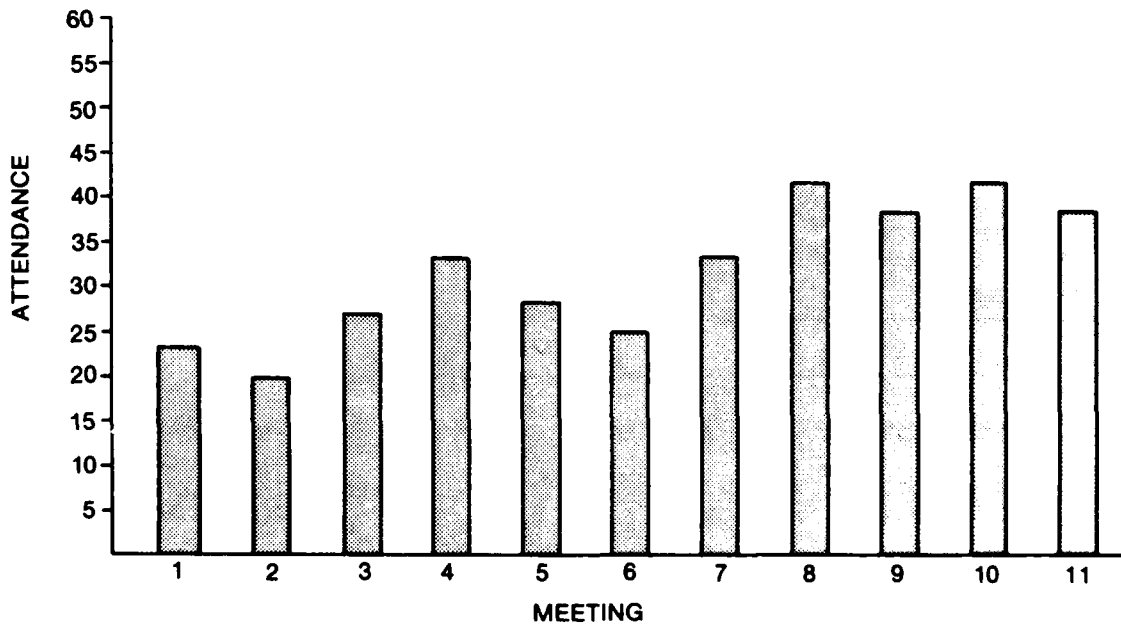


FIGURE 2. NUMBER COMPANIES DURING ATTENDANCE

AIR FORCE CONTROL STRUCTURE

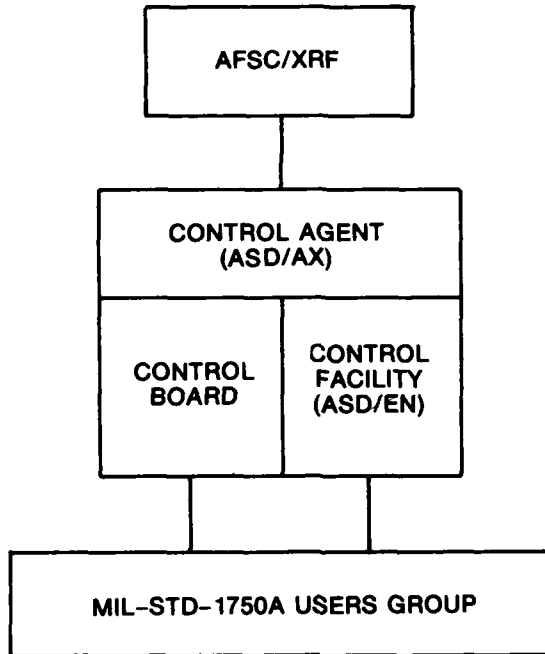


FIGURE 3. THE MIL-STD-1750A CONTROL STRUCTURE

MIL-STD-1750A USERS GROUP

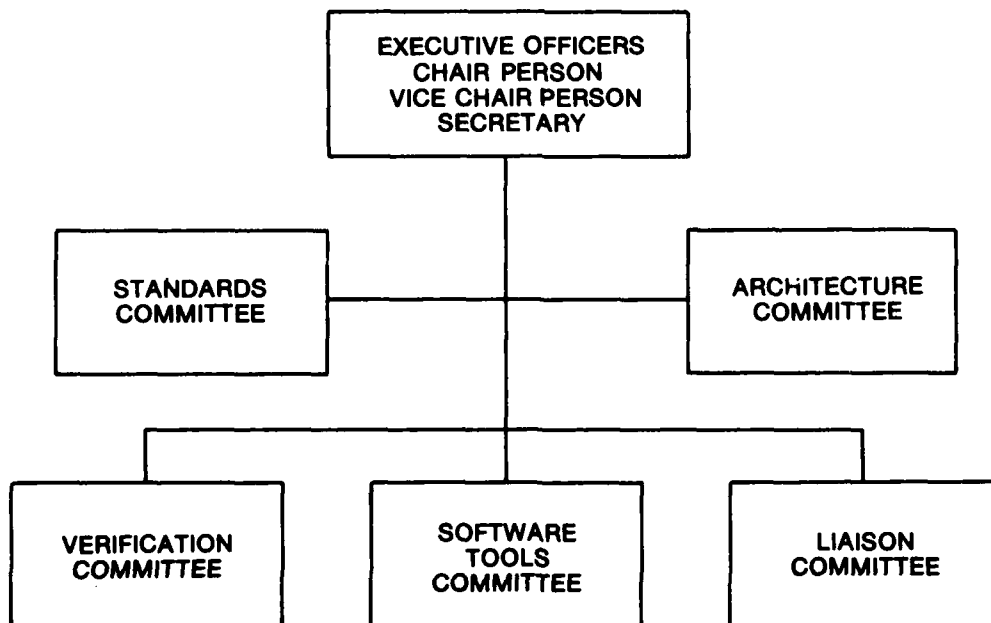


FIGURE 4. THE MIL-STD-1750A USERS GROUP ORGANIZATION

APPENDIX

BY-LAWS OF THE MIL-STD-1750 INSTRUCTION SET
ARCHITECTURE USERS GROUP

ARTICLE I - NAME

1. This organization shall be called the MIL-STD-1750 Instruction Set Architecture Users Group and referred to as 1750 ISAUG.

ARTICLE II - PURPOSE

1. The group is organized and will be operated exclusively for the following purposes.
 - a. To provide a means of communication between persons involved with the use of MIL-STD-1750 Standard Instruction Set.
 - b. To serve as forum for making users recommendations to the Air Force Instruction Set Architecture Control Board.

ARTICLE III - MEMBERSHIP

1. Open to all government contractors, government employees, and others interested in the use of the MIL-STD-1750 Standard Instruction Set.

ARTICLE IV - OFFICERS

1. The Officers of this organization shall be a Chairperson, Vice-Chairperson, and Secretary-Treasurer.
2. The Officers shall be elected from the general membership.
3. The Officers shall serve a term of two years.
4. No member shall serve as chairperson for consecutive terms.

ARTICLE V - DUTIES OF OFFICERS

1. The Chairperson shall preside at all meetings of this group and of its Executive Council. The

Chairperson shall also appoint all committees of the group.

2. The Vice-Chairperson shall assume the duties of the Chairperson in the event of the Chairperson's absence. The Vice-Chairperson shall assume those duties of the Chairperson that are delegated to him by the Chairperson.
3. The Secretary-Treasurer shall keep minutes of all Group and Executive Council Meetings. The Secretary-Treasurer is responsible for preparation and distribution of all meeting minutes and all correspondence with the Air Force Instruction Set Architecture Control Board. The Secretary-Treasurer shall maintain financial and membership records.

ARTICLE VI - EXECUTIVE COUNCIL

1. The Executive Council shall consist of the present officers, the chairpersons of the standing committees, and the most recent past chairperson. The new Executive Council shall take office immediately following the election of the group's officers.
2. Minutes of all Executive Council meetings shall be available for inspection by any member of the group and shall be filed with the group records.
3. The Executive Council shall meet before a general group meeting to plan the general group meeting.
4. The Executive Council shall fill vacancies which occur between election of officers.

ARTICLE VII - STANDING COMMITTEES

1. Standing Committees shall be:
 - a. Standards Committee
 - b. Architecture Committee
 - c. Verification/Certification Committee
 - d. Software/Hardware Development Tools Committee
 - e. Liaison Committee

ARTICLE VIII - TEMPORARY COMMITTEES

1. The Chairperson, with the approval of the Executive Council, may appoint temporary

committees as is deemed appropriate.

ARTICLE IX - MEETINGS

1. Meetings shall be held as planned by the Executive Council. The group shall hold meetings uniformly around the country.
2. Written notices shall be sent to all members at least thirty (30) days prior to any meeting.
3. There shall be at least one (1) scheduled meeting each year.
4. The standing committees shall meet in concert with the Group Meeting. Additional working meetings of the committees may be held at the discretion of the committee chairperson with the concurrence of the committee membership. Minutes of the committee meetings shall be recorded and distributed.

ARTICLE X - CONFERENCE/REGISTRATION FEES

1. Fees shall only be assessed to cover the meeting and administrative costs of the group.
2. Fees shall be fixed and adjusted by the Executive Council and be subject to the rules for amendments.
3. The conference/registration fees must be published in a Meeting Announcement prior to becoming effective.

ARTICLE XI - AMENDMENTS AND PROCEDURE

1. Amendments to these By-Laws shall be made in the following manner.
 - a. The intent to change the By-Laws will be published in a Meeting Announcement, discussed with the membership and voted upon. Two-thirds of the membership present must vote for the change in order for it to be carried.
 - b. The result of the change must be published in the next Meeting Announcement.
2. By-Laws must be republished and distributed to the membership at appropriate intervals as directed by the Executive Council.

3. Rulings on any point of procedure not included in these By-Laws shall be made by the Chairperson. The 1750 ISAUG shall use Robert's Rules of Order.

ARTICLE XII - VOTING

1. Apportionment in matters requiring group consensus voting shall be as follows:
 - A. General Membership - one vote per person present but not more than five (5) votes per institutional representation.
 - B. Mail Ballots will only be permitted for emergency issues called for by the Executive Council. Only attendees at the previous General Membership meeting are eligible to vote. At least half of eligible voters must vote for ballot to be valid.
2. Consensus
 - A. General or Procedural Issues

General or Procedural Issues voted upon within the group shall be considered a consensus (positively or negatively) by a majority vote of the voting general membership.
 - B. Technical Issues

Technical Issues voted upon within the group shall be considered a consensus by a two-thirds majority vote of the voting general membership and two-thirds majority for rejection shall imply rejection, otherwise, deferral shall be implied.

ARTICLE XIII - REFERRAL TO AIR FORCE ISA CONTROL BOARD

1. Approved technical issues shall be documented and forwarded to the Air Force Control Board by the Secretary of the 1750 ISAUG.
2. Any member or group of members has the privilege of preparing a minority opinion and forwarding this to the Air Force Control Board thru the Secretary of the 1750 ISAUG.

AD-P003 548

MIL-STD - 1750A MICROPROCESSOR CHIP SET DEVELOPMENT

Dr. Thomas A. Longo - Vice President, Schlumberger LTD.,
and Chief Technical Officer - Fairchild

Dan Wilnai - Manager, VLSI Research and
Development for Real-Time Processing

FAIRCHILD CAMERA & INSTRUMENT CORPORATION
Advanced Research and Development Laboratories
4001 Miranda Avenue
Palo Alto, California 94304
(415) 493-7250

Dr. Longo has been a technical executive at Fairchild since 1970 and presently heads the Advanced Research and Development Laboratory (ARDL). Prior to joining Fairchild, he held positions as Vice President of Transi-tron and Director of Research at Sylvania Semiconductor.

His IEEE Fellow award in 1974 was for Development of TTL Integrated circuits and other contributions to semiconductor technology. Dr. Longo has numerous patents and publications in the semiconductor field and is a member of American Physical Society, Sigma Pi Sigma, and Sigma Xi.

Education: * BS - Semiconductor Physics - Purdue University
 * MS - Semiconductor Physics - Purdue University
 * PhD - Semiconductor Physics - Purdue University

Mr. Wilnai is presently responsible for digital signal processing and real-time processor development in the Computer Sciences Department. As a member of the advanced-logic group, he was the chief architect for the F9440-microprocessor design and led the design effort of the F9445. Prior to joining Fairchild, he worked for Elta Electronic Industries in Israel and in the Israeli Air Force. Mr. Wilnai holds one patent with one pending and has written numerous papers.

Education: * BSEE - The Technion Israeli Institute of Technology,
 Haifa, Israel
 * Graduate study in Computer Science - The Technion IIT,
 Haifa, Israel
 * Graduate Study in Computer Science - Weizman Institute,
 Rehovot, Israel

ABSTRACT

Fairchild ARDL, under contract with General Dynamics/Fort Worth, is developing a high performance MIL-STD-1750A microprocessor chip set for use in embedded computer applications in U.S. Air Force avionic systems. This paper describes the development program, the design methodology and the process technology.

Fairchild MIL-STD-1750A Microprocessor chip set is centered around the F9450A a 20-MHz 64 pin microprocessor with 200ns cycle time that implements all 1750A instructions including floating point. Two support devices under development are the F9451, a 1750A Memory Management Unit, and the F9452, a 1750A Block Protect RAM device. In addition to the microprocessor chip set, Fairchild is developing an Engineering Test and Evaluation Equipment based on the FS-1 Microprocessor Development System.

The paper discusses current trends in processing technologies and their effects on system performance.

INTRODUCTION

Fairchild Advanced Research and Development Laboratories, under contract with General Dynamics/Fort Worth, is developing the F9450 - a single chip high performance MIL-STD-1750A microprocessor for use in embedded computer applications in the U.S. Air Force avionic systems. This paper describes the development program which includes in addition to the F9450 two support circuits (F9451 and F9452) and Engineering Test and Evaluation Equipment. It discusses the design methodology and current trends in processing technologies and their effects on system performance.

PROGRAM OVERVIEW

In May 1981 Fairchild submitted to GD/FW a proposal for the MIL-STD-1750A Microprocessor development program. In October 1981, after competitive bidding, we were awarded the development contract. The program entails development of a single chip microprocessor that fully implements MIL-STD-1750A (Notice 1) ISA including floating point (F9450); it also includes development of the F9451 - an LSI circuit that implements the MIL-STD-1750A Memory Management Unit (MMU) functions, and the F9452 - an LSI circuit that implements the MIL-STD-1750A Block Protect RAM (BPR) functions. Engineering Test and Evaluation Equipment (ET/EE) is also developed under the program.

Fairchild started the logic design of the F9450 in June 1981 (see Figure 1). PDR and CDR were held in November 1981 and February 1982 respectively. The F9450 design and circuit layout was completed in April 82. At that time while the original design of the F9450 was fabricated as a test device (Figure 2), a circuit optimization activity was started to improve the manufacturability of the F9450 microprocessor. Four additional design iterations are planned, the first of which comes out of fab in mid-November 1982 and the last one by the end of May 1983.

The two LSI support circuits, developed using a standard I³L gate array, were designed in a very short time - 6 1/2 months for the F9451 and 7 1/2 months for the F9452. Only 3 design iterations are planned for these parts which are much simpler than the F9450.

There are two delivery milestones in the program - 15 prototype chip sets and 5 units of the ET/EE will be delivered by March 1, 1983; 100 preproduction chip-sets will be delivered after qualification by September 1, 1983.

F9450 - MIL-STD-1750A CPU

The F9450 microprocessor (Figure 3), in a single chip, completely implements MIL-STD 1750A (Notice 1) Instruction Set Architecture.

Implementation in the I³L-II Bipolar VLSI technology affords static operation with 200 ns bus cycle times, low-power Schottky input/output, inherent radiation tolerance (1×10^5 rads), and operation at 20 MHz over the full military temperature range.

The F9450 provides 16 user-accessible general-purpose registers, and operates on the following data types: bit, byte, single precision integer (16 bit), double precision integer (32 bit) floating point (32 bit), and extended precision floating point (48 bit).

Real-time processing capabilities include (on chip) 2 programmable timers, a complete 16-level interrupt processor and a comprehensive fault handler. Instruction Abort mechanism is provided and it may be utilized for demand paging virtual memory organizations.

The microprocessor features high throughput: 0.2 μ sec integer add, 1.85 μ sec integer multiply, and 5.6 μ sec floating point multiply at 20 MHz clock rate and 150 nsec internal cycle time (Table 1). Performance is achieved by means of advanced technology as well as advanced architecture techniques incorporating high level of pipelining and parallelism and utilizing fast algorithms such as the Modified Booth for multiply operations.

DESIGN METHODOLOGY

To realize the very aggressive development tasks in an extremely tight schedule Fairchild had put together a multidisciplinary design team including Computer Architects, Logic, Circuit and Mask Designers as well as support personnel. Working very closely with GD/FW, Fairchild's team was enhanced with additional design resources from Honeywell Avionics, Minneapolis, Minnesota, GTRI, Atlanta, Georgia and SPCI, Salt Lake City, Utah.

The design process was managed with heavy emphasis on close interaction and cooperation between the team members and had high visibility from Fairchild upper management. An Advisory Board was put in place at the beginning of the program and is monitoring its development. Members of the Advisory Board include Dr. Jim Early - head of the VLSI Development Lab in ARDL; Dr. Gil Amelio - General Manager of the Microprocessor Division and Dr. Peter Verhofstadt - Engineering Manager of the Microprocessor Division. Dr. Tom Longo - Vice President of Schlumberger and Chief Technical Officer of Fairchild

while personally supervising the 1750 microprocessor development program is heading this Advisory Board.

The design process was heavily oriented towards usage of Computer Aided Design (CAD) techniques from System Level Simulation to Automatic Layout and Routing, including Logic and Circuit Simulation and automated Design Rule Checking. In order to combine high performance, maintaining low power and implementing this logic in a small number of components, powerful logic design techniques and innovative circuits were developed. Each CPU logic block was carefully weighed for delay, power and cell area. A functional block (cell based) layout was used on this chip. The functional block structure permitted increased functionality by minimizing the random logic and interconnect constraints.

A special layout technique was used resulting in reduced silicon area and design time: In the strip layout concept, bit zero of the ALU and bit zero of all the registers, etc are laid out to line up in a strip. Bits zero through fifteen are laid out in strips identical to bit zero and placed vertically next to each other separated by power buses. Each strip is designed with identical bussing where data flows between Register File and the ALU on first layer metal. Control signals flow across the data path in second layer metal. This layout technique resulted in a significant layout efficiency and high packing density and reduced bus capacitances. The repetitive nature of the layout also had dramatic impact on design layout time.

TECHNOLOGY

Since the first days of the semiconductor era, Fairchild has been the leader in bipolar semiconductor technology. As early as 1974, we started work on bipolar VLSI technology. Our Isoplanar Integrated Injection Logic (I³L), a technology that combines VLSI packing density with features that are inherent to bipolar technologies like full military temperature range operation and radiation hardness, has been evolving since then.

Table 2 describes the evolution of the I³L technology, showing its usage in various products with increased complexity and improved performance.

The first I³L products were designed in a 5-6 μ technology in 1974-76. the second generation 16-bit bipolar microprocessor, the F9445, was designed in 1979-81 in a 4.5 μ technology. The F9450 is designed in I³L-II with 3 μ feature sizes and better than 5ns gate delay leading to instruction execution times of 700,000 instructions per second (700 KIPS) for the U.S. Air Force DAIS mix with 16% floating point operations.

Processing technology is evolving rapidly and in 1984 we will be introducing into production the next generation bipolar VLSI technology - I³L-III. With feature sizes of better than 2 μ , this technology will improve by more than a factor of two the basic logic propagation delay to less than 2 ns; at the same time reducing chip size and power dissipation.

An enhanced F9450 with innovative architecture and implemented with I³L-III Technology can be developed for 1985-6 production with 2-4 MIPS DAIS mix performance.

SUMMARY

Implementation of MIL-STD 1750A ISA in a single-chip microprocessor is yielding a very powerful tool for avionics system designers. Combining low power dissipation with small physical size and inherent high reliability, the F9450 offers at the same time relatively high performance (700 KIPS) over the full military temperature range. Current trends in processing technology will allow us to upgrade the F9450 to a 2-4 MIPS machine for production in 1985-86.

References

1. MIL-STD-1750A "Sixteen Bit Computer Instruction Set Architecture" - (Notice-1) 1982 (USAF)
2. Critical Item Development Specification 16ZE181 for MIL-STD-1750A Microprocessors - 1982 (GD/FW)
3. Microprocessor Products Data Book - 1982 (Fairchild Camera and Instrument)
4. F9450 (MIL-STD-1750A) 16 Bit Bipolar Microprocessor Preliminary Data Sheet - November 1982 (Fairchild Camera and Instrument)
5. LSI Elements Using I³L Technology - Dan Wilnai, Compcon 1977

F9450 Instruction Execution Times (uS) - 50 ns CPU Clock Period

	Single Precision Integer	Double Precision Integer	Floating Point	Extended Floating Point
Register Add/Sub	0.2	0.8	4.5	5.75
Register Multiply	1.85	5.75	5.6	12.4
Register Divide	4.7	12.0	9.8	21.15
Load Direct	0.6	1.25	1.25	1.3
Branch	Taken = 0.75 μ s		Not Taken = 0.2 μ s	

Table 1

TABLE 2 - I³L TECHNOLOGY EVOLUTION

	<u>1974-78</u>	<u>1979-81</u>	<u>1982-84</u>	<u>1985-86</u>
Technology	I ³ L-I	I ³ L-Is	I ³ L-II	I ³ L-III
Process Features	5-6u Projection Print Mixed Wet-Dry Etch	4.5u Projection Print Mixed Wet-Dry Etch	3u Projection Print Dry Etch Full Implant	2u Direct Step Dry Processing Enhanced Interconnect
Gate Delay	10 ns	8 ns	5 ns	< 2 ns
Major Products	9440 9408 9414	9445 9447/8/9 9480	9450 9443	Enhanced 9450
Complexity (Active Components)	10,000	20,000	150,000	200,000 *

* Although the Enhanced F9450 will require about 200,000 active components, the I³L-III technology is capable of implementing VLSI devices with up to 300,000 active components.

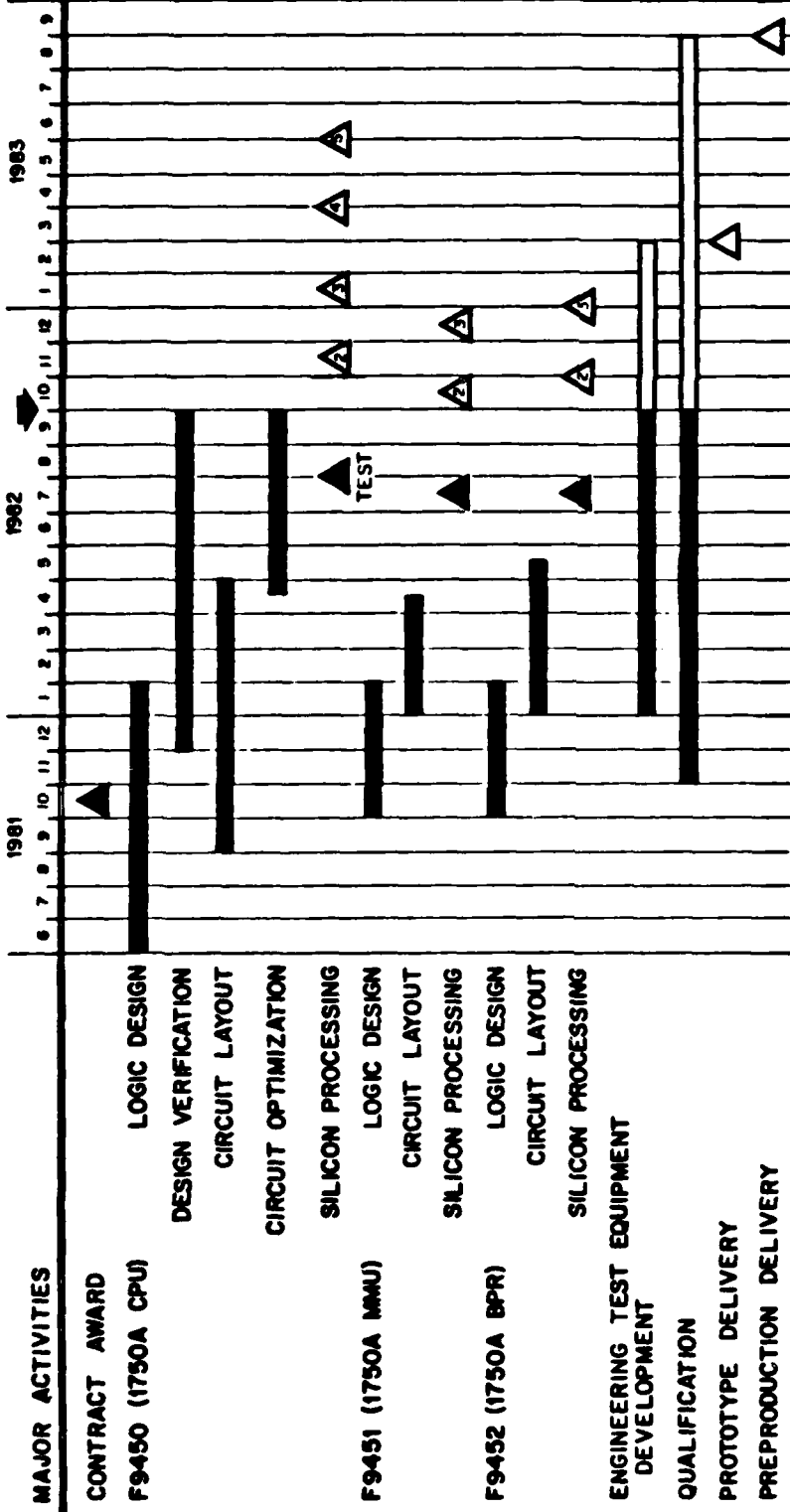


Figure - 1

MIL-STD 1750A MICROPROCESSOR DEVELOPMENT PROGRAM

MAJOR ACTIVITIES

Status as of September 30, 1982

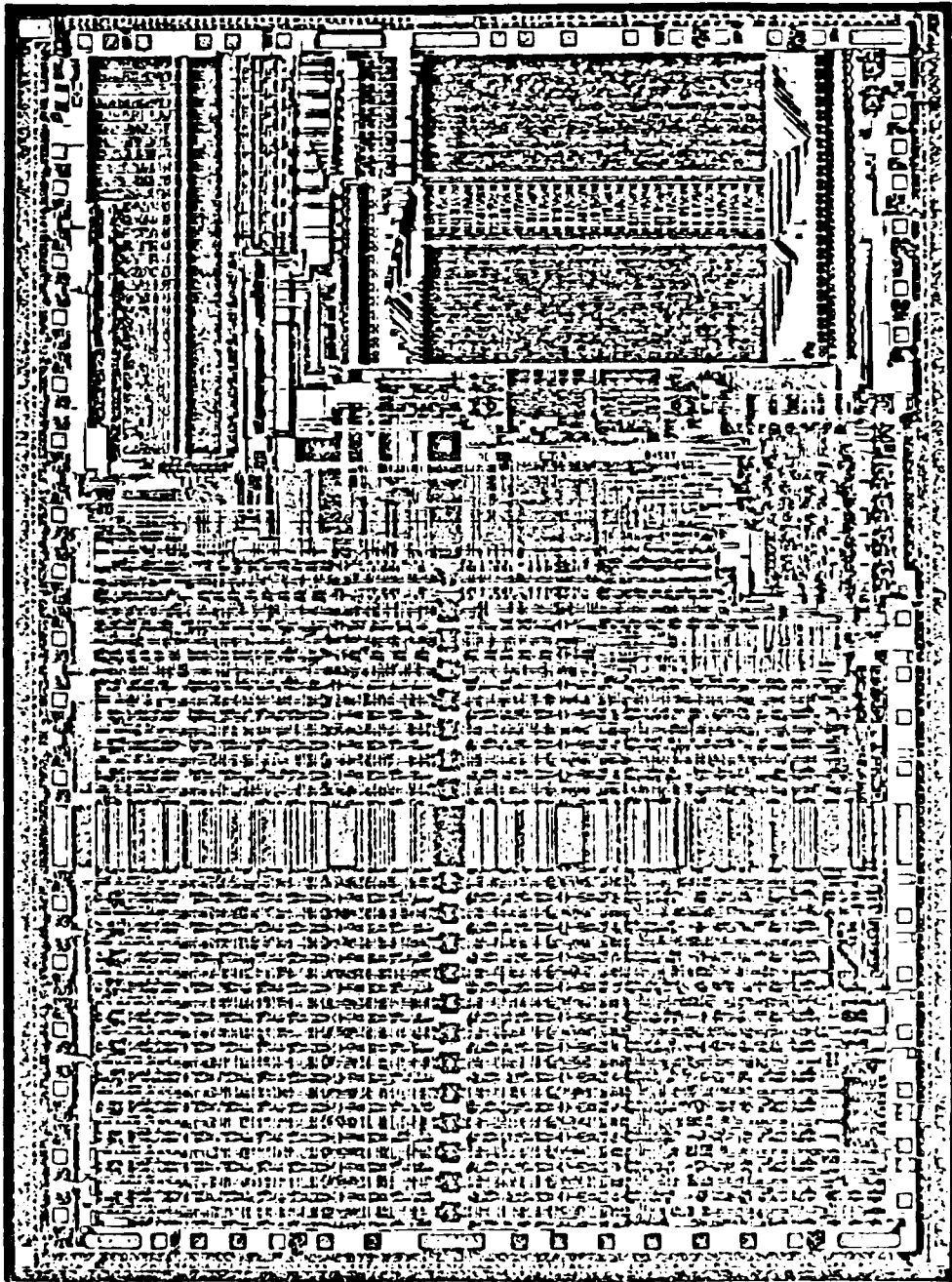


FIGURE 2 - F9450 PHOTOMICROGRAPH

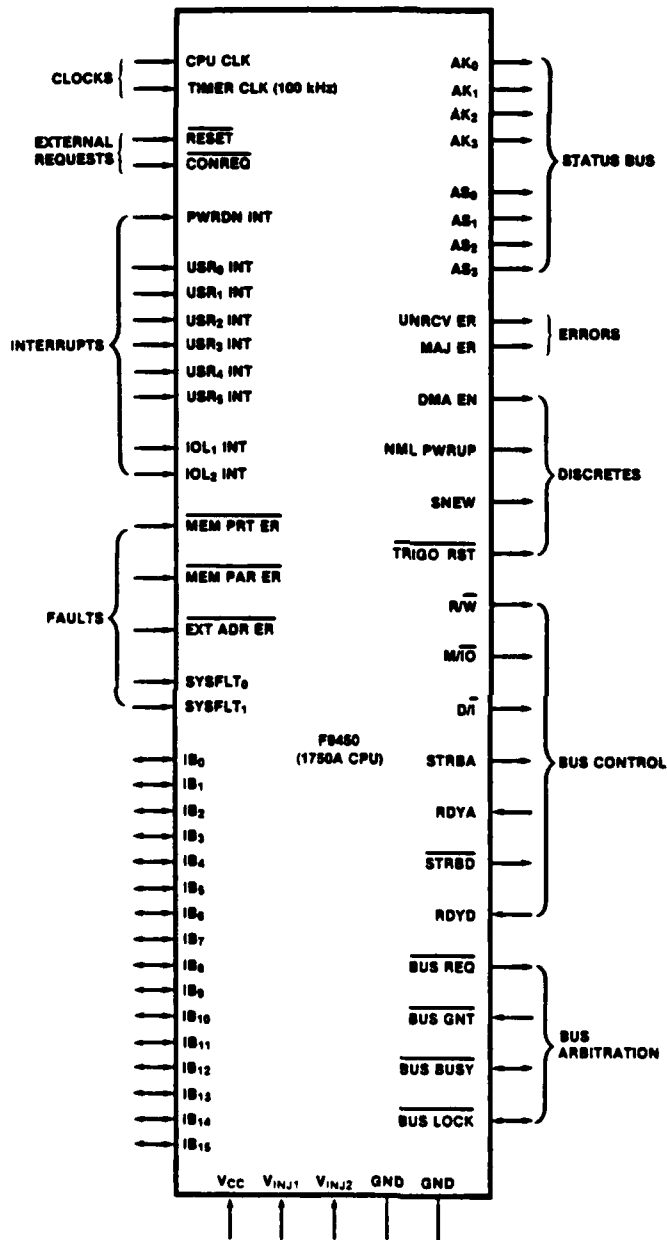


FIGURE 3 - F9450 PIN FUNCTIONS



A HIGH PERFORMANCE MIL-STD-1750A
MICROPROCESSOR

T. L. Rasset and J. H. Lane

McDonnell Douglas Astronautics Company
5301 Bolsa Avenue
Huntington Beach, California 92647
(714) 896-3311

BIOGRAPHIES

Mr. Lane is a Unit Chief responsible for advanced processor architecture and design in the Data Control and Processing Department. He obtained his BS in physics from New Mexico Institute of Mining and Technology in Socorro, New Mexico, and his MS in electrical engineering from the University of New Mexico in Albuquerque, New Mexico. Mr. Lane has worked on computer architecture and logic design since 1961 when he participated in the design of a serial drum computer for missile applications. In 1973, he designed and programmed a computer used as the autopilot for a remotely piloted vehicle. He supervised the development of a GP and a signal processing computer used in the ALWT Underseas Development Program. For the last 3 years, he has supervised the development of a series of MIL-STD-1750 computers culminating in the development of a high-performance CMOS/SOS chip set.

Mr. Rasset is a Design Engineer responsible for processor microcode design and development in the Processing Subsystems Group. He joined MDAC in 1978 after graduating with a BA and a BS in computer science and mathematics from Western Washington University. He has been involved in several of MDAC's past processor development efforts as well as work on real-time operating systems and software development tools for embedded processors. He is currently a graduate student in computer science at the University of California, Irvine where he is doing research on data flow architecture.

ABSTRACT

As part of a continuing research and development program in embedded computer systems, McDonnell Douglas Corporation (MDC) is developing a high-speed MIL-STD-1750A microprocessor. The central processing unit consists of three integrated circuits, which together form a 1750A processor capable of performing the DAIS instruction mix at greater than 900 KIPS.

The technology used in the 1750A microprocessor is a mature complementary metal oxide semiconductor/silicon on sapphire (CMOS/SOS) process in use at MDC. This is a radiation-tolerant, low-power, high-speed technology that makes it possible to achieve the processor's high throughput while consuming less than 2 watts of power.

This paper describes the architecture of the processor and the user-oriented approach taken in the development of the processor and its support tools.

INTRODUCTION

Since the early 1960's, McDonnell Douglas Astronautics Company has been involved in developing aerospace computers for use in company-developed systems. During the last few years, these have been built primarily from available transistor-transistor logic (TTL) bit slice parts. However, the physical size and power consumption of this technology have limited the applicability of these processors. Commercial microprocessors offer an alternative for some lower performance systems, but their performance is generally inadequate and their instruction sets are oriented to a different class of problems. Fortunately, by the end of the 1970's, IC design and processing technology had matured to the point where it was realistic to consider developing custom, high performance large scale integration/very large scale integration (LSI/VLSI) processors.

During the 1970's, McDonnell Douglas Electronics Company worked with several integrated circuit technologies, among which was CMOS/SOS. By the end of the decade, a very mature 5-micron process was in regular use and work was under way on advanced processes. CMOS/SOS is an ideal technology to meet the power, performance, and environmental requirements that are imposed on an aerospace computer. It consumes very little power, while being capable of operating at speeds equal to Schottky TTL. Additionally, it is capable, through careful design and process rules, of achieving space qualification levels for radiation tolerance. Given the need for a low-power, high-speed LSI processor, and a technology capable of achieving these goals, all that remained was the selection of a suitable instruction set architecture.

From the beginning, McDonnell Douglas has been active in the evolution of MIL-STD-1750A from MIL-STD-1750. Under a contract to the Air Force, a TTL bit slice 1750A processor was developed to gain experience with the 1750A standard as it was evolving. In addition to the computer, a set of software tools, including a simulator, assembler, and linker, was developed for the Air Force. As a result, a group of people at MDAC became knowledgeable about the 1750A instruction set architecture and had an appreciation for its capabilities. It was their judgment that 1750A would be a good choice as the instruction set architecture (ISA) for a custom LSI microprocessor. It offered a realistic ISA for implementation in a small chip set, with all the necessary features (such as floating point, numerous levels of prioritized interrupts, etc.) plus a standard set of software development tools supported by the Air Force.

By 1981, all of the pieces were in place and a joint independent research and development effort between the McDonnell Douglas Astronautics and Electronics Companies was launched. The goal of this effort is to design and build a MIL-STD-1750A chip set that is small in physical size, low in power consumption, space qualified for radiation tolerance, and has a throughput of close to 1 MIP with the DAIS instruction mix.

OVERVIEW

The result of this development effort is the Model 281 microprocessor, a microprogrammed CMOS/SOS processor that implements the 1750A ISA to notice 1. In addition to full compliance with the mandatory items in the 1750A standard, the 281 CPU contains timers A and B, the trigger go timer, DMA control, and startup ROM control. For the extended address and memory protect options, an additional MMU/MPR IC was designed for use with the 281 CPU. The MMU/MPR chip may be configured to be a memory management unit controller for a full complement of page registers or a memory protect RAM controller for anywhere from 65,536 or 64K to 1 million words of memory.

The 281 CPU is partitioned into three functional units, each of which is a CMOS/SOS integrated circuit. The execution unit is the heart of the CPU, containing all of the arithmetic/logic functions of the machine. The control unit is the microengine that controls the processor, containing the microsequencer and the microcode ROM. Through an external pin connection, the control unit may be configured as a ROM only, with the microsequencer functions disabled. This ROM may then be connected to a control unit as an additional microcode ROM. In this way, the microstore can be expanded up to 4096 or 4K words, of which 1500 are used for the 1750A instruction set. The third chip contains the interrupt system and fault register, plus several 1750A options, including timers A and B, the trigger go timer, DMA controls, and startup ROM controls.

For a non-radiation-hard version of the 281 processor only three ICs are required, one of each type. However, for a space-qualified, radiation-tolerant processor four ICs are needed, one of each type plus an extra microcode ROM. This is due to the redundant elements that are required in the hardened ROM which reduces its capacity to one half of that in the nonhardened ROM. In all other respect the two versions are identical, with radiation-tolerant design rules followed throughout. With a 20-MHz clock, the non-radiation-hard 281 processor is capable of performing the DAIS instruction mix at 940 KIPS while consuming less than 2 watts of power. Because of the second microcode ROM, the radiation-tolerant version runs somewhat slower, performing the DAIS mix at 780 KIPS. These performance times are for a processor configured with a maximum of 64K words of memory with a cycle time of 200 nsec. If a memory management unit or memory protect RAM is used, the throughput slows down by about 25% to 705 KIPS for the nonhardened version and 585 KIPS for the hardened version. If faster memories are available, a portion or all of the performance may be regained.

Current plans are to package all of the ICs in 64-pin leadless chip carriers and to mount them on a thick-film assembly (TFA). Two TFA versions are planned, a minimum version that will contain a 1750A CPU capable of addressing up to 64K words of memory and a maximum version that will support all of the expanded memory and memory protect options with fully buffered TTL outputs. The TFAs will be multilayer ceramic substrates, with the circuits between ceramic dielectric layers. The minimal version will be the size of a standard 64-pin DIP. The maximal version will measure 2.7 by 5 inches and will have 100 pins. Both TFAs are designed for easy mounting on a printed circuit board.

DETAILED DESCRIPTION

A block diagram of the 281 processor is shown in Figure 1. Each of the three integrated circuits is shown with the primary chip interconnections. The M bus is the microinstruction bus, over which the control unit provides microinstructions to the other two ICs. The address/data bus is used for communications between the execution unit and the interrupt unit, as well as between the execution unit and the memory and I/O systems. The 281 uses a multiplexed address and data bus with the control signals necessary to allow the memory or I/O system to operate asynchronously. Time-outs are used to detect illegal I/O operations or references to nonexistent memory locations.

The execution unit is based upon a three-bus architecture, two operand source buses and one destination bus, each of which is 16 bits wide. There are four major structures within the execution unit: the register file, the barrel shifter, the ALU, and the multiplier. Each of these structures is highly regular, allowing the IC layout to take place through the replication of a small set of cell types. This greatly simplifies the IC design and layout, which reduces the risk of an error.

The register file contains 24 16-bit registers in a dual-port RAM structure. Sixteen of the registers serve as the 16 general registers called for in the 1750A standard and the remaining 8 are temporary registers for use by the microcode.

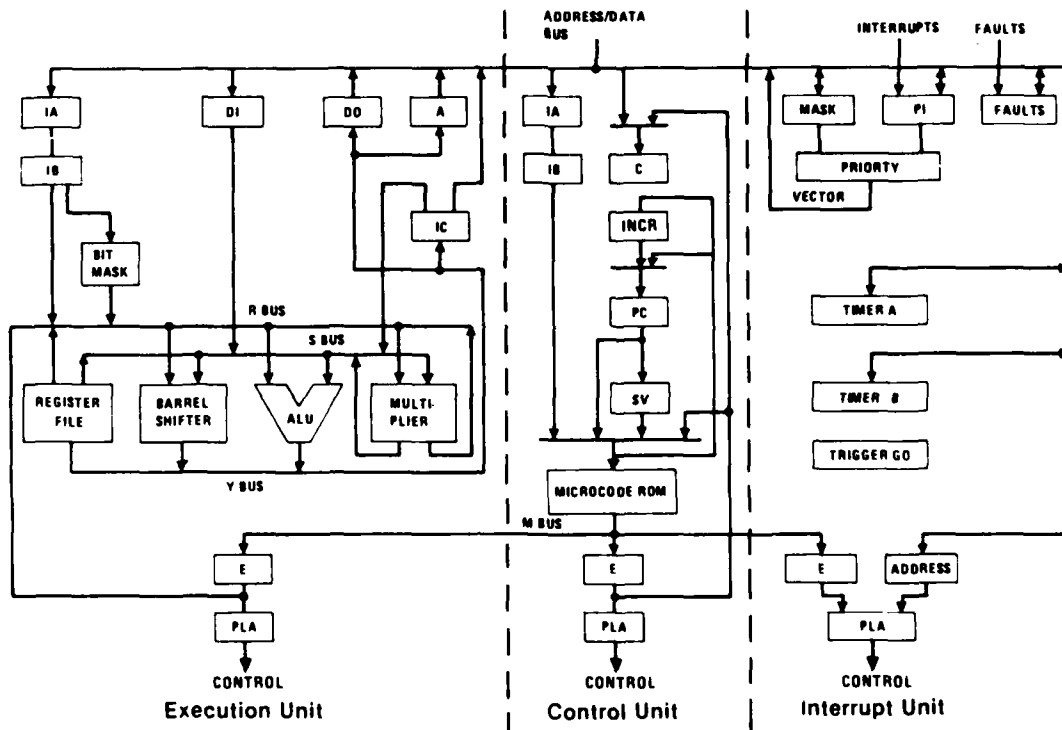


Figure 1. Block Diagram of The 281 Processor

The barrel shifter is a 32-bit input, 16-bit output right-shift network. A double-precision operand can be shifted right arithmetically, logically, or cyclically 0 to 31 bit positions in one machine cycle. The shift count can be directly programmed via the microinstruction or can first be loaded into a shift count register. The shift count register can be loaded from the destination bus or from a normalize test circuit that provides the shift count necessary to normalize a given operand.

The ALU is a 16-bit arithmetic logic unit capable of performing arithmetic or logic operations in either 16- or 8-bit mode. In addition to selecting ALU source operands from among the other elements within the execution unit, a 16-bit constant may be loaded directly from a microinstruction. Not shown in Figure 1 is a 1-bit left shifter and a shift extension register that are used for divide operations within the ALU.

The multiplier performs a 4 by 24 bit multiply plus accumulation in a single machine cycle. Contained within the multiplier is a 48-bit product accumulation register, the lower 24 bits of which serve as a source operand register. On each cycle, the lower 4 bits of the accumulator are multiplied by 24 bits from the R and S buses. The lower 24 bits of this 28-bit product are then added to the upper 24 bits of the accumulator and the whole accumulator is shifted right 4 bits, making room for the upper 4 bits of the product. The 4 bits shifted out are then used in the next multiply iteration. By iterating the appropriate number of times, the different precision multiply operations required by 1750A may be performed.

Instruction look ahead is employed by the 281, with the IB register holding the current 1750A instruction and IA the next instruction. This allows the 281 to always have the next instruction read in and ready for execution when the current instruction is completed. For 32-bit instructions the data input (DI) register will be loaded with the second 16 bits of the instruction during the instruction decode cycle so that this value will be available at the start of the microinstruction. In addition to a data input register, there is a data output (DO) register, an operand register, A, and the 1750A instruction counter and status word within the execution unit.

The control unit microcode ROM contains 2K or 2048 40-bit words in the non-hardened version and 1024 words in the hardened version. In addition, there are 8-bit extensions to 512 ROM locations that serve as the lower 8 bits of the next microinstruction address when the instructions at these 512 locations are executed. These 8 bits are added to a fixed upper 4 bits to form a full 12-bit microinstruction address. This allows nonsequential microinstruction execution without having to use bits of the microinstruction for a branch address.

The remaining area of the control unit is taken up by the microsequencer, which consists of a microprogram counter (PC) with increment logic, micro-PC save register, an iteration counter, and 1750A instruction decode logic. The microinstruction address is 12 bits, as are all associated registers. The microinstruction address is available on pinouts from the chip for use in configurations that require more than one microcode ROM.

The interrupt unit is largely a collection of miscellaneous functions. It contains the interrupt system with its associated registers, masking, and prioritization logic. Nine interrupt levels are brought off chip on pinouts, the remaining levels are set from on chip elements or by the microcode via the pending interrupt register. Eight pins on the interrupt unit IC are inputs for the fault register. The fault register is a 16-bit register, as defined in the 1750A standard, that is used to generate a machine fault interrupt. The rest of the chip is dedicated to 1750A options that are desirable for most systems. These include the two timers defined by the 1750A standard, timers A and B. In addition, there is a trigger go timer that counts down from a fixed value and generates a machine fault if it reaches its terminal value. This functions as a watchdog timer to ensure that the processor is alive and functioning at all times. For external monitoring, there is an output pin on the IC that indicates when the trigger go has counted out. There is also control logic for a startup ROM, for DMA, and for a power-up normal discrete.

The MMU/MPR chip, which is not shown, is used for systems that require expanded addressing or memory protect RAM. Through external pin connections this chip may be configured as either a memory management controller for the fully expanded memory option or as a memory protect RAM for anywhere from 64K words to 1 million words of memory. When used as a memory management unit, latches are needed to store the extended address bits and RAMs are needed to contain the actual page registers. When configured as a memory protect RAM for more than 64K words of memory, additional memory chips are needed to contain the protect RAM. However, the chip does contain sufficient memory to provide memory protect RAM for up to 64K words of memory.

TESTABILITY

Special elements, such as shift registers, are built into the 281 to aid in testing. Each of the buses has a 16-bit shift register associated with it and all of the shift registers are linked together so that they can be shifted out over a single pin. The timing of the 281 was designed so that it is possible to freeze the processor at midcycle and all the buses will contain the source and result data values. These values can then be shifted out over the shift registers to verify that all of the buses have the expected values.

The control unit circuits that bring the microinstruction address off the control unit chip are bidirectional so that microinstruction addresses can be fed into the control unit. This allows external selection of microinstruction addresses for verification of the ROM contents or for single stepping at the microinstruction level. Similarly, test microinstructions can be fed into the execution unit and the interrupt unit on the M bus pins to force execution of special test instructions on a single chip.

A built-in test (BIT) is included with the microcode. The BIT is executed whenever the processor is reset or can be entered via a built-in function (BIF) instruction without resetting the processor. The same circuit simulations that are being used to verify the IC design are being used to test the effectiveness of the BIT to ensure that a majority of the circuit elements will be exercised.

Control signals are provided so that the 281 can be single stepped by 1750A instructions. This will be used by the monitor system to allow single stepping for both hardware and software checkout.

USER AIDS


A monitor system has been developed to provide a user interface to the 281 processor for hardware and software checkout. The monitor consists of a single board microcomputer that connects to the 281 and a Fortran program that runs on a central computer system. A standard RS-232 communication link between the central computer and the microcomputer is all that is needed to allow the user full control of the 281. The user simply logs into the central computer and calls up the monitor program to establish a link to the 281. Through the monitor, the user has all of the resources of the central computer at his disposal for storage of 1750A load modules and processor check point files.

The goal of the monitor design is to offer the user as many of the features of an interpretive computer simulator as is possible. A user should be able to develop software using software tools, including a simulator, on a timeshare computer system and then without changing terminals, access the actual hardware and load the program for execution. The 281 monitor offers this capability. The monitor control program is written in standard Fortran and uses the same command mnemonics as the 1750A simulator developed by McDonnell Douglas for the Air Force.

The 281 monitor system allows the user to control the 281 for run, halt, and single step. Any of the processor's registers may be read or modified, as can any memory location. The processor's memories and registers can be loaded from a disk file or read and saved on a disk file. Memory locations may be addressed symbolically using program labels by connecting the symbol table from the 1750A linker to the monitor program. The monitor hardware contains a trace buffer for tracing all activity on the address/data bus, including I/O operations. The trace buffer contents are tagged to identify each value as a memory references are identified as read or write instruction or operand fetch. Instructions fetched may be disassembled into 1750A mnemonics when the buffer contents are displayed.

SUMMARY

The 281 processor and support tools represents a significant improvement over processors currently available for low-power embedded computers. With its high throughput and small size, plus low power consumption, high noise immunity, and radiation tolerance, the model 281 is an excellent processor for embedded computer systems. A 281 processor mounted on a thick-film assembly offers the performance of a large computer on a single 64-pin DIP. Additionally, the monitor system provides a powerful and convenient user interface for hardware and software checkout. The three-chip partitioning of the 281 processor has the advantage of placing the microcode on a separate IC, which makes reprogramming straightforward and allows for multichip microcode expansion.



A MIL-STD 1750A COMPUTER
EMPLOYING THE MDAC CMOS-SOS CHIPSET

Charles Frank/Larry Speelmar.

ROLM Corporation - Mil-Spec Computers Division
4900 Old Ironsides Drive
Santa Clara, California 95050
(408) 988-2900

BIOGRAPHIES

Charles F. Frank Jr. - Member of the Technical Staff, ROLM Mil-Spec Computers

During his four year tenure at ROLM, Mr. Frank has been a key engineer in development of custom products, Nuclear Survivability/Vulnerability, low level interfaces, and single card processors for computers in major programs such as Cruise Missile Common Support Weapon System and complex Electronics Warfare systems. Over 10 years of experience at E.S.L. and Dalmo Victor in systems and hardware design engineering and management complete his well rounded background.

Education - BSEE California State Polytechnic University

Larry W. Speelman - Marketing Manager for Air Force Programs, ROLM Mil-Spec Computers.

Currently responsible for Air Force Programs Marketing at ROLM, Mr. Speelman has been involved with avionics system design, development, and flight test for over twenty years. Previous assignments at Kaiser Electronics, Teledyne Systems Company and the U.S. Air Force involved development, program management, and marketing of computer navigation, and display elements of modern weapon systems.

Education - BSEE California State University of Northridge
MSBA California Western University

ABSTRACT

The need for higher performance, low power, radiation tolerant, MIL-STD-1750A computers for single card up to full ATR configurations is satisfied by ROLM Corporation through application of the McDonnell Douglas Astronautics Corporation (MDAC), CMOS-SOS Model 281 Chipset. Combining this chipset with the standard ROLM bus on an AN/UYK-19 type card permits 1750A insertion into a wide variety of computing applications for new or retrofit programs. Government and Prime contractors are provided access to standard catalog products where projects must employ 1750A processors with associated memory, I/O, and Mux Bus components without incurring large engineering expenses for these elements. This allows 1750A processor use from the initial stages of any 16 BIT computer based system. Design trade off factors of emulator, remicrocoded mini, and chipset approaches are presented along with typical card set and system application.

INTRODUCTION

ROLM Mil-Spec Computer Division designs and produces a broad spectrum of rugged and military application 16 and 32 Bit computers, peripherals and software with the unique combination of high performance, off the shelf availability, low cost, and high reliability. This business philosophy allows our customers to access state-of-the-art hardware without incurring R&D costs or penalties for low volume, quick reaction programs.

ROLM internally funds its development efforts and uses these assets to incorporate advancing technology and pragmatic operational improvements into its evolutionary product line. ROLM is also very active in addressing the specific requirements of our customers in such areas as nuclear hardening, EMP, special interfaces and memories, software, and ILS tasks. This results in ever increasing families of high performance equipments such as the AN/UYK-19, the 4050/4150 35 MB Winchester Disk, the powerful 32 Bit MSE/800 computing system and the first available production quality Ada (1) compiler and development environment hosted on a 32 BIT software development system. Figure 1 shows some salient features of ROLM processors and a few of the major programs currently in production.

With Air Force imposition of the MIL-STD 1750A ISA, many programs and customers were potentially not able to access the large ROLM resource base. Therefore ROLM defined and is developing a high performance full 1750A Notice 1 processor which employs the MDAC 281 Chipset (2). This TFA mounted CPU is interfaced to the family of ROLM I/O and memory products through circuitry located on the CPU module. Also included in the module is up to 64KW of CMOS or NMOS memory to provide a high performance, low power processor module.

- (1) Ada is the registered trademark of the Department of Defense (Ada Joint Program Office)
- (2) A high performance MIL-STD-1750A Microprocessor, T.L. Rasset and J.H. Lane, McDonnell Douglas Astronautics Corp., 2nd AFSC Standardization Conference, November 1982.

ROLM PROCESSOR CHARACTERISTICS

- HIGH PERFORMANCE
- NUCLEAR HARDENED
- HIGH RELIABILITY
- EASE OF MAINTENANCE
- VALUE ENGINEERING
- STRONG SOFTWARE ASSETS
- FULL INTEGRATED LOGISTICS SUPPORT SYSTEM
- CUSTOMER SUPPORT AND TRAINING SERVICES

SAMPLE MAJOR PROGRAMS

- U.S. AIR FORCE DATA BASE MANAGEMENT
PROCESSING FOR AIRBORNE COMMAND POST
- U.S. AIR FORCE AFSATCOM TERMINAL PROGRAM
- U.S. NAVY AN/USQ-81 COMMAND CONTROL AND TARGETING
SYSTEM
- U.S. NAVY AN/SLQ-32 SNEWS
- JOINT CRUISE MISSILE COMMON WEAPON COMPUTER SYSTEM
(CWCS)
- U.S. ARMY AN/TSQ-114 TRAILBLAZER
- U.S. ARMY AN/ALQ-151 QUICK FIX
- U.S. ARMY AN/ARN-132 TACAN INERTIAL NAVIGATION SYSTEM

FIGURE 1. ROLM OVERVIEW

PROCESSOR SELECTION

In 1981 a ROLM task team prepared a complete business plan for developing a 1750 processor that would be compatible with and take advantage of the current and forecasted product line. This analysis resulted in three options:

1. Remicrocode an existing processor.
2. Design a new computer.
3. Apply a custom chip CPU.

As shown in Figure 2, these options each exhibited desirable and undesirable characteristics. Combinations of the paths were compared for suitability to the range of speed, cost, and flexibility typically required by ROLM's customers and their wide variety of applications. Each approach was rejected in turn for the following reasons.

- 1). Remicrocode an existing processor - This would result in a four card CPU higher in performance but otherwise very similar to those already funded by the government and representing the first generation of 1750A machines.
- 2). New Machine Design - A completely new design would involve an expensive, lengthy development schedule which would yield a multi card full ATR processor. This would have been physically too large for many Air Force applications.
- 3). Custom Chip CPU - This promising approach offered a single card CPU but uncertainties in performance, schedule, and government sponsor interfaces made this approach appear somewhat risky. Also lacking was flexibility in the microcode and Nuclear Hardness potential for many anticipated applications.

Upon reviewing the options, the potential advantages of a chip set approach were recognized. ROLM then decided to more fully explore the custom chips currently under development to determine if one could be suitable for ROLM's needs. Microprocessor emulators were reviewed but throughput would be too slow for complex processing and data management. Again referring to Figure 2, ROLM pursued discussions with McDonnell Douglas Astronautics Corporation (MDAC) to apply their IR&D project Model 281 CPU product to the ROLM family and an agreement formalized. Major advantages of the Model 281 project that influenced the decision were:

- o Flexibility in Microcode and Configuration
- o High Performance (throughput)
- o Low Power Consumption
- o Nuclear Hardening potential
- o Maturity of the design (fourth generation)
- o MDAC experience in 1750 software and development tools

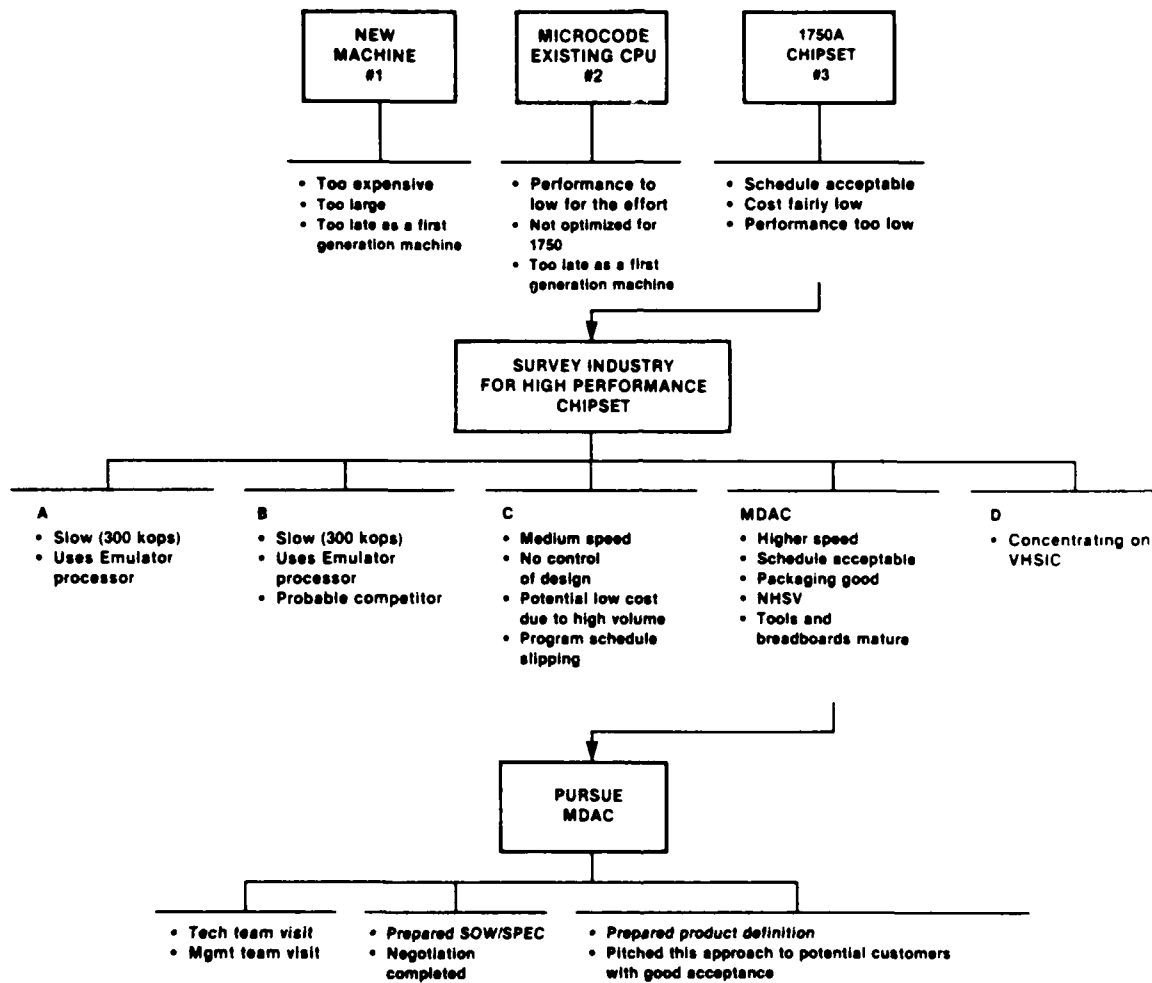
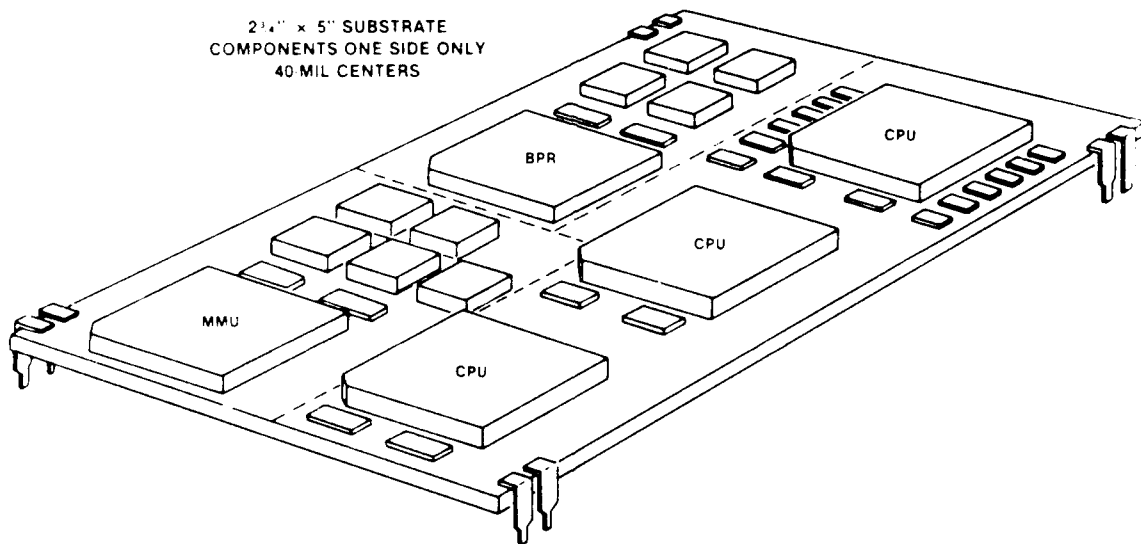


FIGURE 2. 1750A PROCESSOR APPROACH TRADE OFF ANALYSIS SUMMARY

1750A PROCESSOR

The ROM 1750A Processor Module is based around the MDAC Model 281 CMOS-SOS Chipset as mounted on the Thick Film Assembly (TFA) ceramic substrate shown in Figure 3. The TFA is designed for normal production mounting on a Printed Circuit card and is positioned as shown in Figure 4. The NOVA I/O and support circuitry are mounted on the remaining surface of CPU A board. CPU B board contains up to 128 KBytes (64K Words) of CMOS or NMOS memory and the associated memory addressing and timing circuitry.

The 1750A module is a standard 6"x9" shape that fits into standard ATR type chassis (Figure 4). A simplified block diagram of the CPU A/B functional layout is depicted in Figure 5.



- CHIPS MOUNTED ON LEADLESS CHIP CARRIERS (LCC)
- LCCs MOUNTED TO TFA
- CHIP SET TECHNOLOGY IS CMOS-SOS

FIGURE 3. MDAC SUPPLIED THICK FILM ASSEMBLY (TFA)

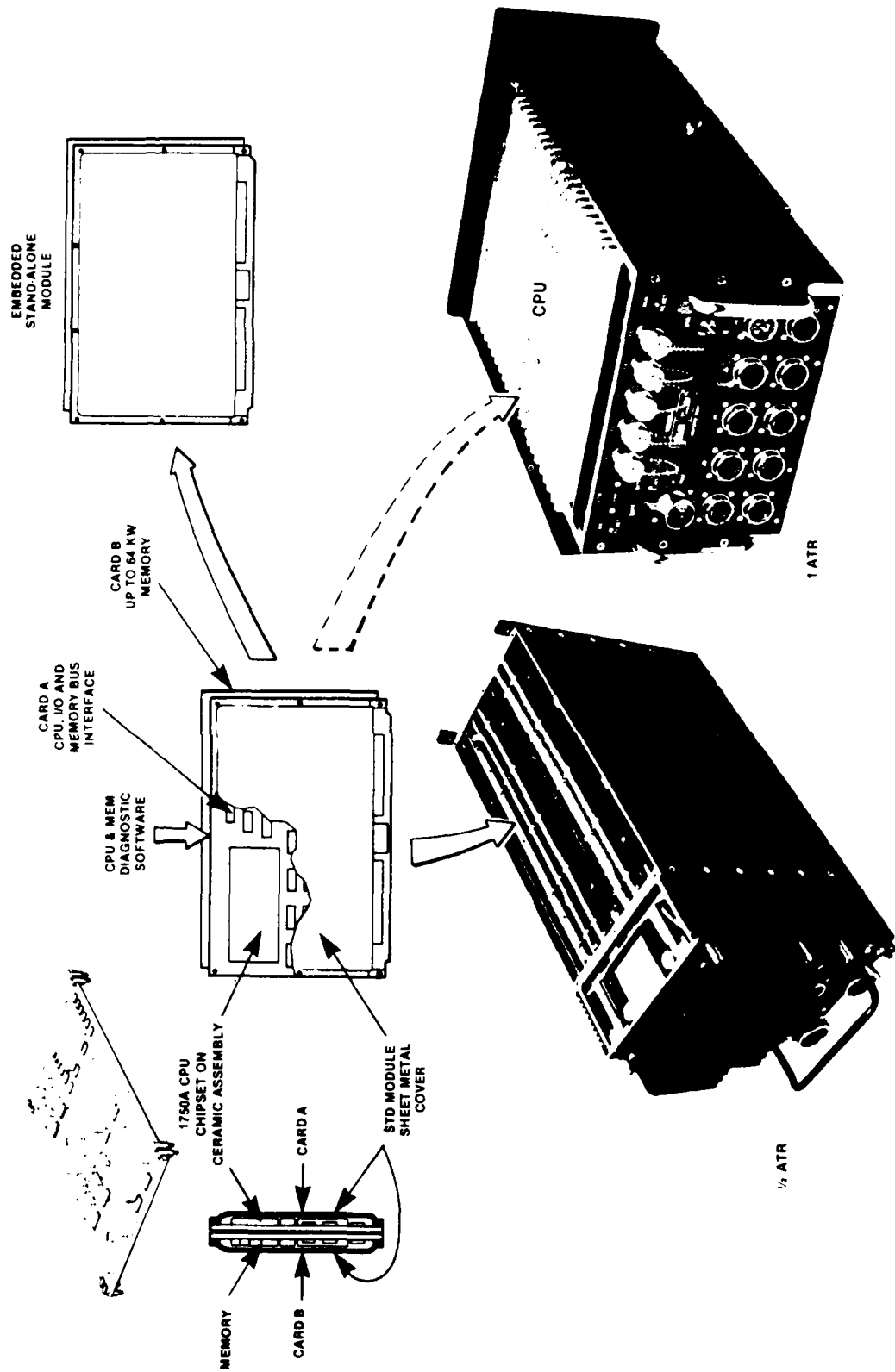


FIGURE 4. 1750A PRODUCT DEVELOPMENT SUMMARY

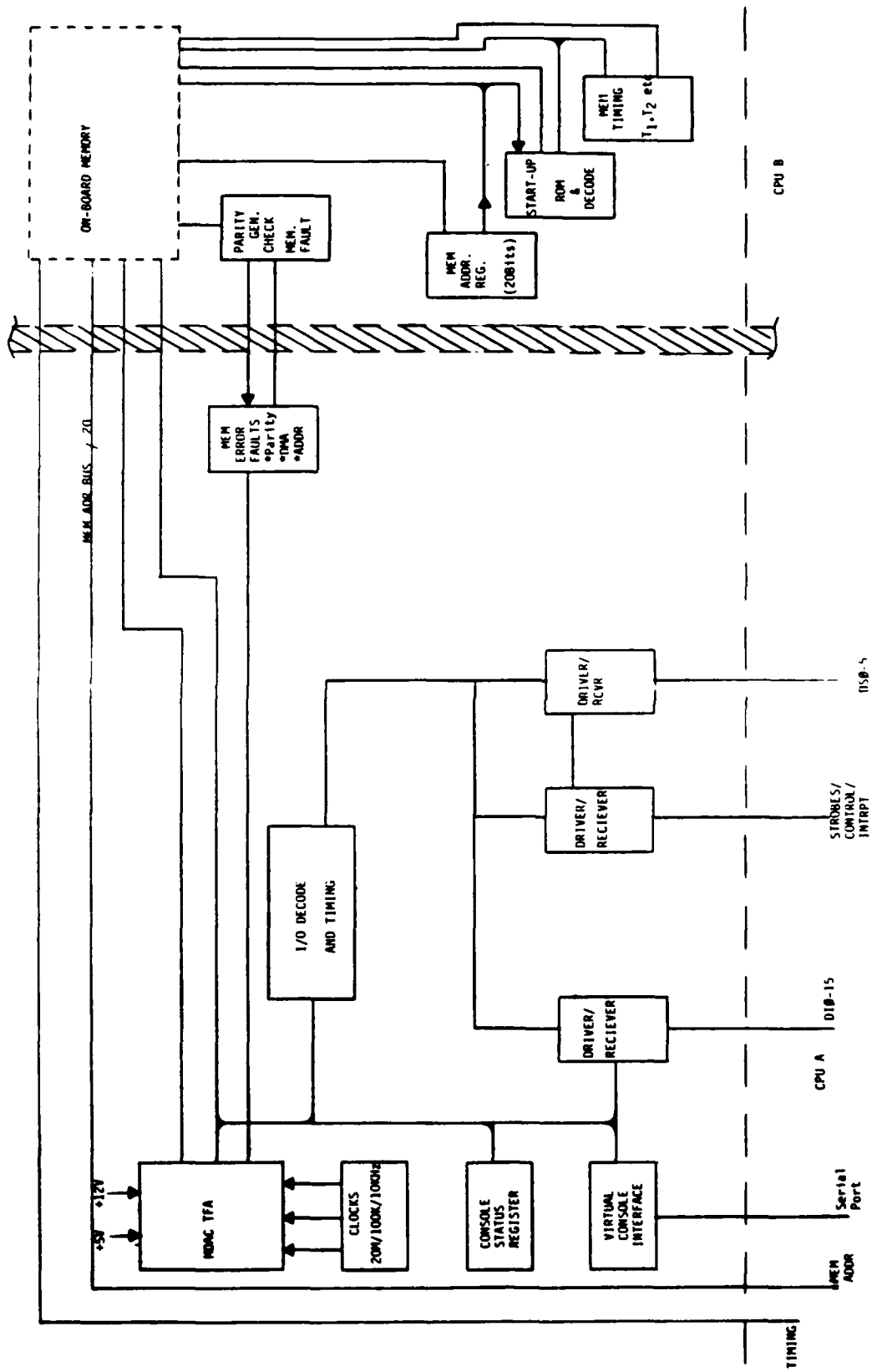


FIGURE 5. SIMPLIFIED 1750A CPU MODULE BLOCK DIAGRAM

The impact of this high performance single module CPU in packaging medium and larger processing systems is illustrated in Figure 6. The ATR unit on the left is a standard ROLM chassis with a full module load. The single module CPU along with a new 512K Byte RAM memory module means that a full 1 million word 1750A processing system can be packaged into the 1 ATR chassis and still have room for up to 12 dedicated I/O slots. Similarly, the 1/2 ATR chassis may be configured to accommodate 5 I/O slots and 384K Bytes of DRAM for smaller applications. The CPU Module may also be employed as a stand alone processor with 64K Words of memory and the ability for growth to a full 1 million words through additional memory. Combining the 1750A module with the ROLM product line provides customers with system configuration flexibilities only available with a broad range of complimentary products.

- o 1750A CPU module is pinout compatible with other ROLM processors, hereby directly utilizing the same 1/2 ATR chassis, power supply, I/O and memory hardware.
- o A single card MIL-STD-1553B module (3761) may be configured as terminal, monitor, broadcast, or complete mux bus controller.
- o The ROLM developed Nuclear Event Detector circumvention system permits system configurations where stringent Nuclear Hardness requirements are present. Many ROLM products have been tested and approved for these environments.
- o The basic 1750A CPU operates with NMOS and CMOS memory. Other existing types of memory used in the ROLM product family could be adapted to this processor such as dynamic ram, EPROM, and core.
- o A wide variety of systems design interfaces including Multi-processor, communications, synchronous and asynchronous communications, and digital/analog I/O are available through the standard ROLM catalog and could be interfaced to this processor. Diagnostics have to be rewritten to operate under the 1750A ISA.
- o A complete Integrated Logistic Support capability exists at ROLM to support the ROLM product line with analysis, training, spares, and technical publications throughout the life of a program.

SUMMARY

The ROLM 1750A Module employs the MDAC Model 281 CMOS-SOS high performance CPU and is packaged in a standard 6" x 9" ATR module size. Main features of the product are its high performance, low power, flexibility of application, and nuclear hardness potential. This product provides ROLM customers access to the extensive ROLM family of memory, I/O and peripherals for a wide variety of programs.

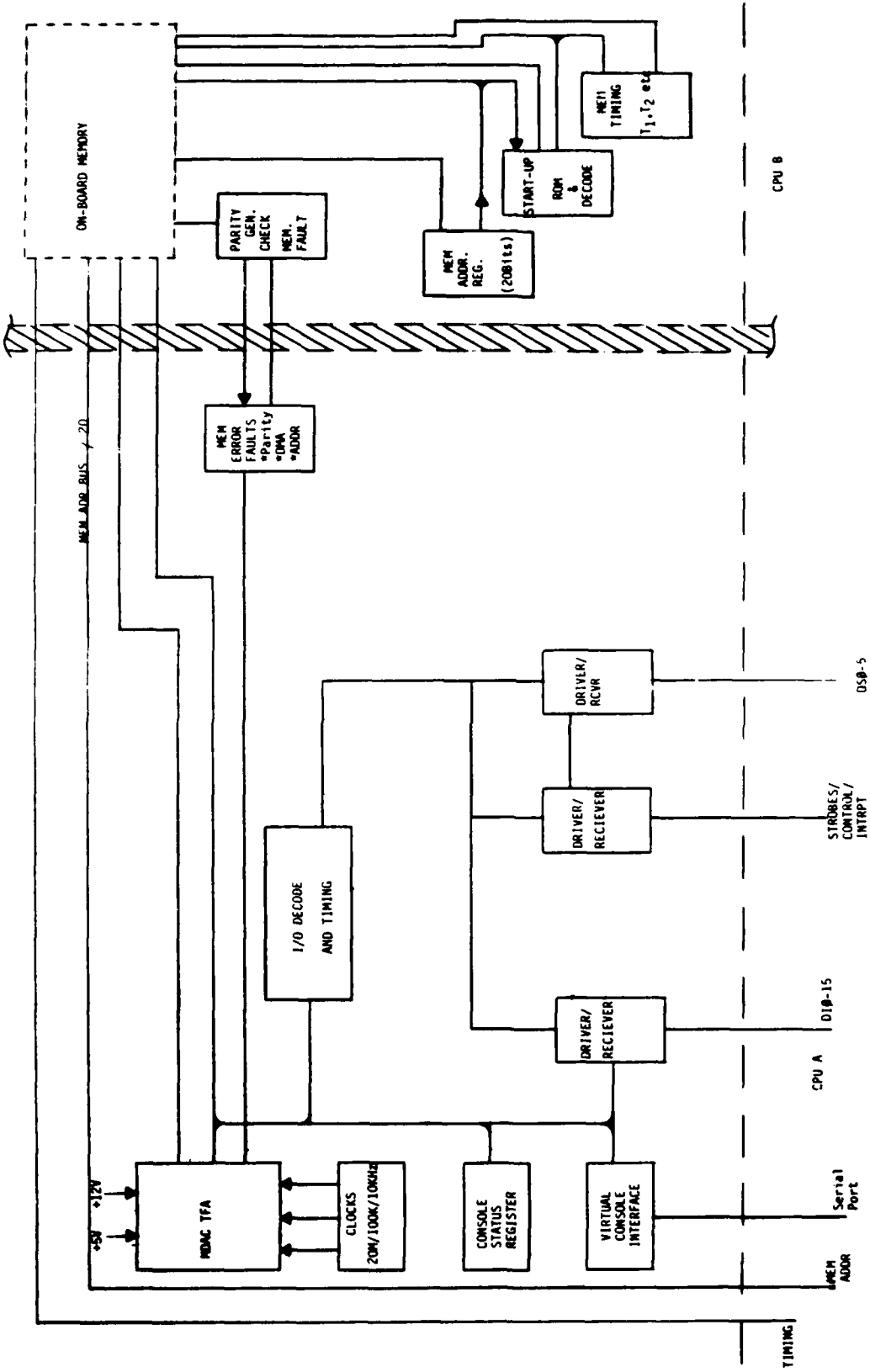


FIGURE 5. SIMPLIFIED 1750A CPU MODULE BLOCK DIAGRAM

ROLM PROCESSOR CHARACTERISTICS

- HIGH PERFORMANCE
- NUCLEAR HARDENED
- HIGH RELIABILITY
- EASE OF MAINTENANCE
- VALUE ENGINEERING
- STRONG SOFTWARE ASSETS
- FULL INTEGRATED LOGISTICS SUPPORT SYSTEM
- CUSTOMER SUPPORT AND TRAINING SERVICES

SAMPLE MAJOR PROGRAMS

- U.S. AIR FORCE DATA BASE MANAGEMENT
PROCESSING FOR AIRBORNE COMMAND POST
- U.S. AIR FORCE AFSATCOM TERMINAL PROGRAM
- U.S. NAVY AN/USQ-81 COMMAND CONTROL AND TARGETING
SYSTEM
- U.S. NAVY AN/SLQ-32 SNEWS
- JOINT CRUISE MISSILE COMMON WEAPON COMPUTER SYSTEM
(CWCS)
- U.S. ARMY AN/TSQ-114 TRAILBLAZER
- U.S. ARMY AN/ALQ-151 QUICK FIX
- U.S. ARMY AN/ARN-132 TACAN INERTIAL NAVIGATION SYSTEM

FIGURE 1. ROLM OVERVIEW

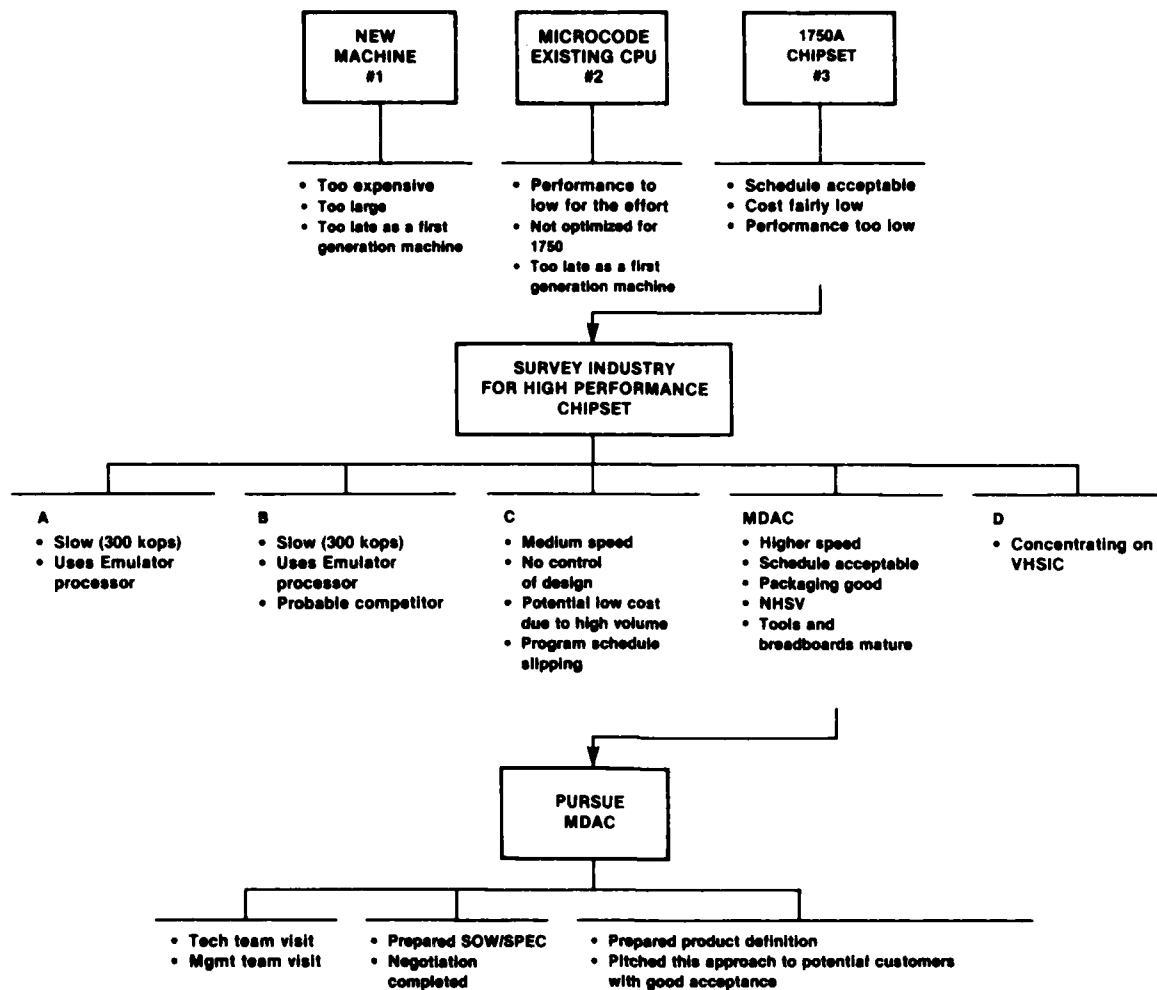
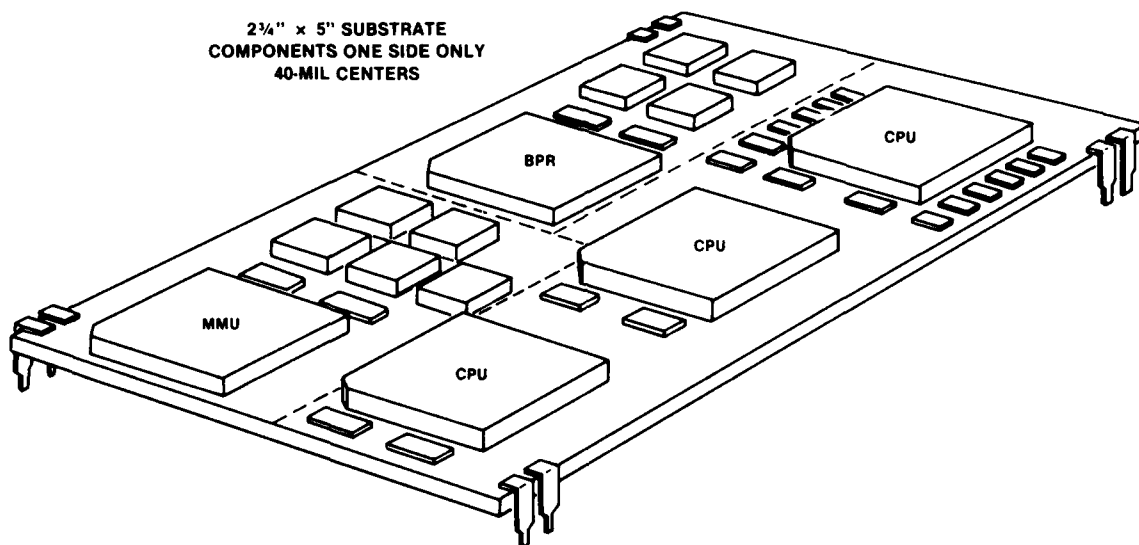


FIGURE 2. 1750A PROCESSOR APPROACH TRADE OFF ANALYSIS SUMMARY

1750A PROCESSOR

The ROLM 1750A Processor Module is based around the MDAC Model 281 CMOS-SOS Chipset as mounted on the Thick Film Assembly (TFA) ceramic substrate shown in Figure 3. The TFA is designed for normal production mounting on a Printed Circuit card and is positioned as shown in Figure 4. The NOVA I/O and support circuitry are mounted on the remaining surface of CPU A board. CPU B board contains up to 128 KBytes (64K Words) of CMOS or NMOS memory and the associated memory addressing and timing circuitry.

The 1750A module is a standard 6"x9" shape that fits into standard ATR type chassis (Figure 4). A simplified block diagram of the CPU A/B functional layout is depicted in Figure 5.



- CHIPS MOUNTED ON LEADLESS CHIP CARRIERS (LCC)
- LCCs MOUNTED TO TFA
- CHIP SET TECHNOLOGY IS CMOS-SOS

FIGURE 3. MDAC SUPPLIED THICK FILM ASSEMBLY (TFA)

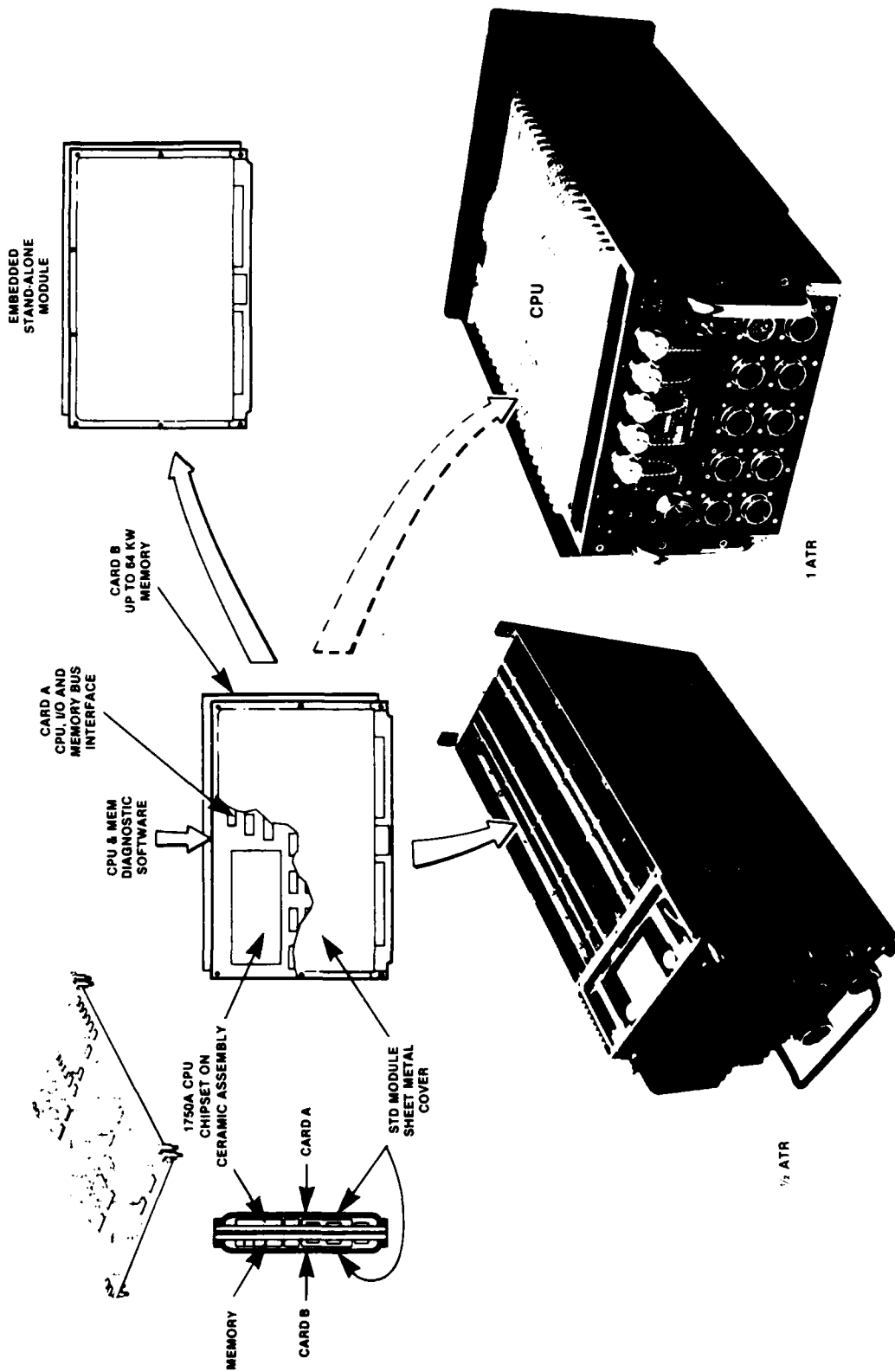
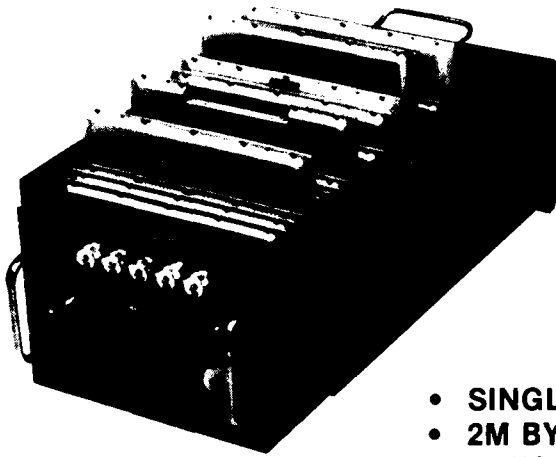
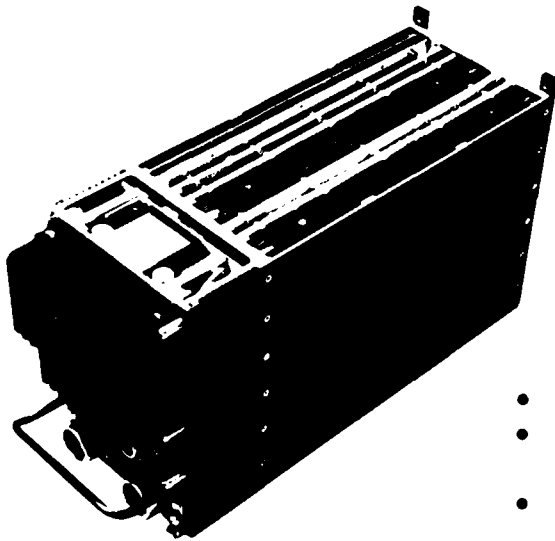


FIGURE 4. 1750A PRODUCT DEVELOPMENT SUMMARY



FULL ATR CHASSIS

- SINGLE MODULE CPU
- 2M BYTES DRAM MEMORY
- 12 I/O SLOTS
 - 1553B (TERMINAL OR CONTROLLER)
 - DETECT EVENT MODULE
 - A/D & D/A
 - ASYNCH MUX (RS-232/188-114)
 - DIGITAL I/O
 - PERIPHERAL INTERFACES (DISCS, MT, LP, ETC.)
 - CPU TO CPU COMMUNICATIONS
 - NUMEROUS OTHER TYPES



1/2 ATR CHASSIS

- SINGLE MODULE CPU
- 384K BYTES SEMICONDUCTOR MEMORY (STATIC NMOS OR CMOS)
- 5 I/O SLOTS

**FIGURE 6. COST/PERFORMANCE IMPROVEMENTS
IN LRU PACKAGING**

A NEW SILICON-ON-SAPPHIRE MIL-STD-1750A MICROPROCESSOR

Joseph R. Burns, Henry Silcock,
Gregory Portanova, Steven Nicholas

Mikros Systems Corporation
Mercerville, New Jersey

ABSTRACT

This paper describes the MKS1750, a new implementation of the United States Air Force MIL-STD-1750A architecture fabricated entirely in SOS/CMOS technology. The MKS1750 is a chip set based on a proprietary microprogrammable 16-bit microprocessor (the MKS16) supported by additional logic implemented as semi-custom gate arrays and ROMs for control store. A total of eleven chips is required. Power consumption of the MKS1750 is less than 1W and throughput is greater than 200 KIPS, measured using the USAF DAIS mix.

A complete description of the chip set is given, including the architecture of the MKS16, gate array partitioning and the functional specification of each array. The paper also discusses the structure of the MKS1750 microprogram, available software support and packaging considerations.

1. INTRODUCTION

The advantages of SOS/CMOS technology in computer applications are well known: high speed, low power consumption, high packaging density, and radiation resistance. While the cost of the sapphire substrate precludes its use in high-volume commercial applications, the aforementioned attributes of SOS/CMOS are overwhelmingly important (and in some cases, imperative) in military computers (1,2).

The Mikros strategy has been to use SOS/CMOS for the design and development of a full 16-bit CPU chip. Additionally, the basic CPU is microprogrammable using external control store to allow emulation of a variety of instruction set architectures.

This paper describes the emulation of the USAF MIL-STD-1750A architecture using the Mikros MKS16 processor chip supported by

SOS/CMOS gate arrays and control store. The chip set executes the full 1750A instruction set (including floating-point) at a throughput of 265 KIPS (DAIS instruction mix) and dissipates less than 1 Watt.

2. MKS1750 ARCHITECTURE

2.1 OVERVIEW

The MKS1750 is a microprogrammed processor which executes the USAF MIL-STD-1750A instruction set (3). It has the following features:

- o MULTIBUS™-compatible interface (4)
- o 16-bit microprogrammable processor
- o Eleven chip LSI implementation

The architecture of the MKS1750 is based on a 16-bit internal bus (the I-bus, see Fig. 1). Sequential execution of 1750A instructions is controlled by the MKS16 processor, which is responsible for instruction fetch and decode, effective address calculations, operand fetch and store, and arithmetic operations. External logic is provided to support 1750A emulation in the following areas:

- o additional microsequencing features
- o MULTIBUS™ interface
- o 1750A interrupts and faults
- o additional register file and addressing logic

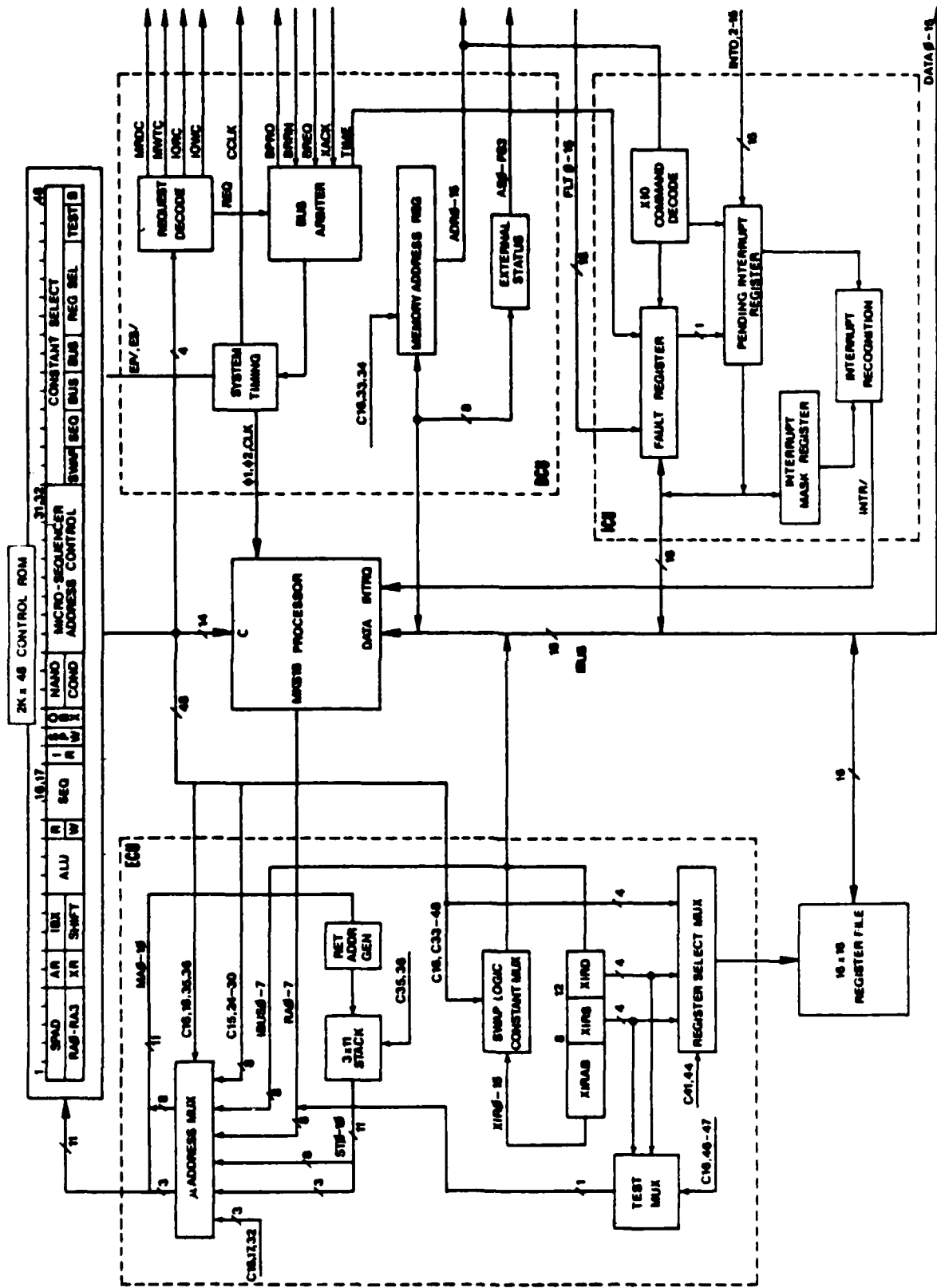
The control ROM (2Kx48) provides synchronous control signals to both the external logic and the MKS16.

2.2 THE MKS16 MICROPROGRAMMABLE PROCESSOR

The MKS16 has a register file/ALU/sequencer architecture with internal status, instruction and shift registers (see Fig. 2) (5). The processor communicates with a bidirectional data bus through two multiplexers (IBX, OBX). Internal storage is provided by the scratchpad (SPAD), a 16x16 register file. The A and X registers (AR, XR) are used as accumulators and for shift operations. The status register (SR) contains condition codes and status flags. The 16-bit parallel ALU provides two's-complement arithmetic and boolean operations. The architecture supports a limited degree of parallelism at the micro-operation level.

The MKS16 is driven by a 27-bit microinstruction word. The microinstruction is time-multiplexed into PRIMARY and SECONDARY words which are fetched sequentially from control store during one microcycle. The format of the standard microinstruction word is shown in Fig. 3. In this case, bits C15-17, C31, C32 are not used by the MKS16 and may be used as control bits for external logic.

Fig. 1 MKS1750 Functional Diagram



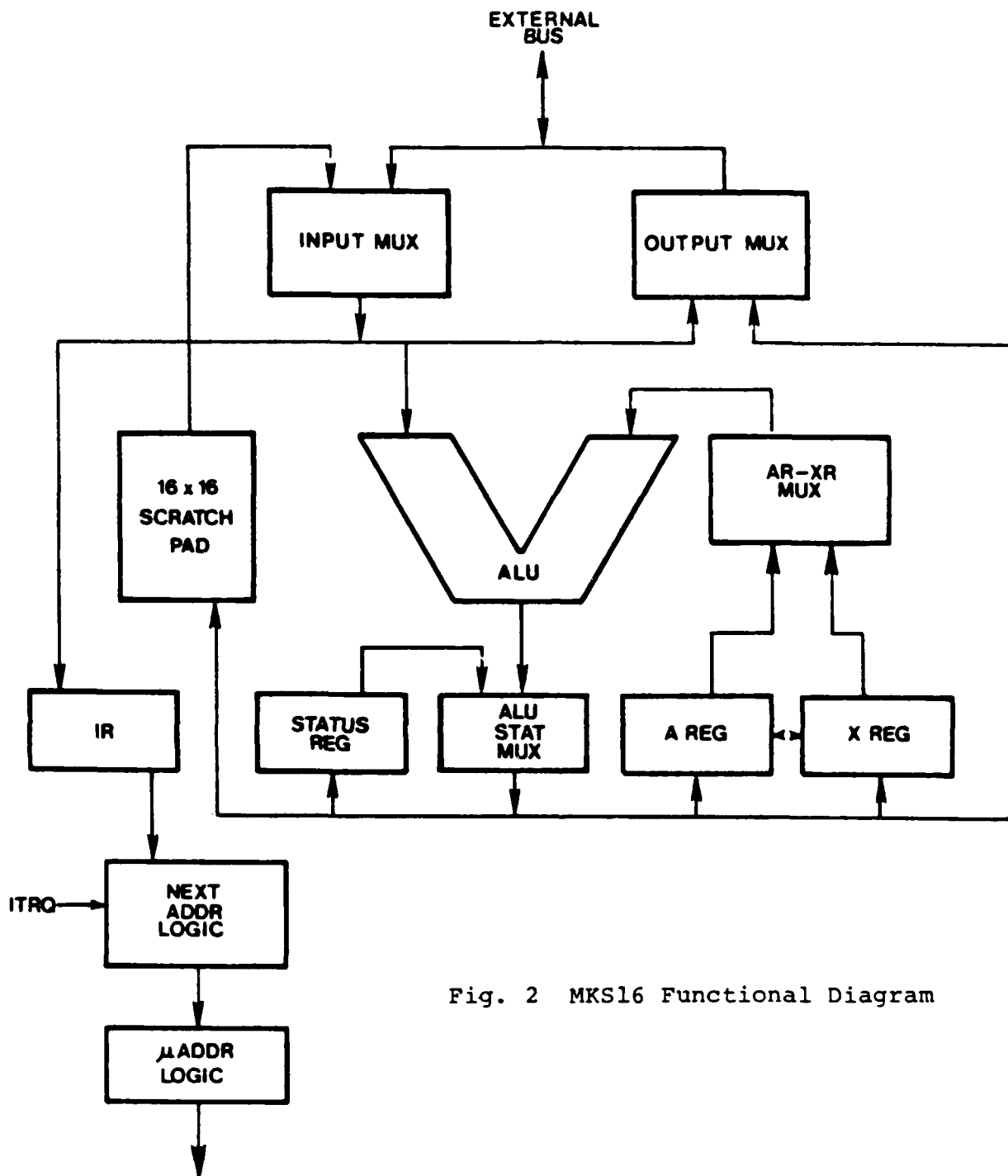


Fig. 2 MKS16 Functional Diagram

The MKS16 microsequencer does not use a default flow-of-control convention; each microinstruction specifies the next microaddress, explicitly. The low-order four bits of the address (the WORD ADDRESS) are given by C27-C30. The high-order four bits (the PAGE ADDRESS) are determined by a LINK MODE specifying the source of the page address. The sequencer allows conditional branching based on the condition codes.

The MKS16 requires three clock signals, and can run at either TTL or CMOS levels. The maximum clock rate is 5.5 MHz, giving a microcycle time of 180 nsec.

2.3 REGISTER STRUCTURE

The MKS1750 uses an external 16x16 register file to provide the 1750A registers. The MKS1750 contains dedicated logic to support register concatenation and register addressing using macroinstruction fields. This logic uses the external instruction register (XIR), which is loaded with a 1750A instruction during the fetch cycle. The XIR S and D fields are used by most 1750A instructions as register addresses, and may be used directly to access the specified register. Alternatively, a literal-4-bit address may be specified directly by the microinstruction word. The S and D fields are implemented as four-bit up/down counters, to provide access to adjacent registers as required by 1750A multiple-precision instructions. The S and D fields may also be concatenated to form an eight-bit up/down counter. Zero-detect logic is provided for all three counters. This logic is used extensively in implementing indexed addressing and shift instructions.

2.4 THE MKS1750 MICROSEQUENCER

The MKS16 internal sequencer is oriented towards instruction decoding. Typically, four bits of the instruction register (IR) are used as the page address, providing a sixteen-way branch in control store. The MKS1750 microsequencer provides logic for expanded microaddress space, microsubroutining, instruction decoding and testing of external conditions.

Three extra microaddress bits are taken directly from the microinstruction word, and are appended to the MKS16-generated address as the high-order bits. The resulting 11-bit address is known as the NORMAL ADDRESS (N-address) and is one of four inputs to the NEXT ADDRESS MULTIPLEXER in the microsequencer (see Fig. 4). Another input is the LONG ADDRESS (L-address) which is a literal 11-bit address specified in the microinstruction. The next address may also be obtained from the return address stack (3x11). The final address mux input is obtained by concatenating the three high-order extension bits with an 8-bit value from the I-bus.

N-addressing is required for a microsubroutine call operation. In this case, the return address is not specified explicitly by the microinstruction, but is determined by incrementing the high-order three bits of the current micro-address. N-addressing is also required by microinstructions which test internal or external conditions.

The MKS16 internal link modes may also be combined with the external modes to provide extra sequencing features. The microsequencer also allows the execution of an instruction to be aborted if certain machine faults occur during execution.

2.5 THE BUS INTERFACE

The MKS1750 is logically compatible with the MULTIBUS™ standard. Address, data, command, and status signals are supported.

The 20 address lines are TTL-compatible signals which provide 1 Mword of physical address space. The 16 bidirectional data lines are TTL-compatible signals which provide a 16-bit data bus. To communicate with slave devices, several control signals are provided: memory read/write (MRDC, MWTC) and I/O read/write (IORC, IOWC). Slave devices acknowledge transfers using the XACK control line.

A multi-master environment is supported using serial priority arbitration. BREQ and BUSY are polled by each requesting device to determine whether a bus access may occur.

There are 8 user-defined interrupt signals which are processed by the interrupt controller. A bus transfer example is shown in Fig. 8.

2.6 INTERRUPTS AND FAULTS

The MKS1750 contains three registers dedicated to the 1750A pending interrupt (PI), interrupt mask (MK), and fault (FT) registers. The hardware determines from the values of PI and MK if there is a valid interrupt which requires service by the MKS16. If so, this logic asserts the MKS16 interrupt request line, and this in turn is detected by the MKS16 sequencer. The MKS1750 normally tests for the presence of a valid interrupt at the end of each instruction. If one is detected, the processor determines its priority by reading PI and MK, calculates the address of the new context, and performs the context-switch operation. Certain 1750A interrupts (executive call and arithmetic exceptions) are not asynchronous events as they occur only as the result of instruction execution. In these cases the microprogram must set the correct PI bit.

Machine faults set the appropriate FT bit to cause a machine error interrupt (provided it is not masked). Certain machine faults cause the current instruction to be aborted (see 3.7).

2.7 SYSTEM TIMING

The MKS1750 uses a four-phase clock. $\phi 1$ is used to latch the primary microinstruction word into the processor and gate arrays. $\phi 2$ is used to latch and output the contents of the MKS16 scratchpad during a read cycle. $\phi 3$ (WAIT) is used to synchronize processor-bus transactions. The processor waits during this phase for a bus access and for the accessed slave device to indicate transfer completion. $\phi 4$ (CLK) is used to latch the secondary microinstruction, the registers, the next address and the condition codes. A system timing example is shown in Fig. 8.

3. PRINCIPLES OF EMULATION

3.1 THE CONTROL WORD

The MKS1750 uses a 48-bit microinstruction word, shown in Fig. 5. Bits C1 - C14 and C18 - C30 are used to control the MKS16 processor.

Several microinstruction fields are used to determine the next microaddress. The high-order three bits (the BLOCK ADDRESS) are given by C17, C31, C32. C15, C24 - 30 are used to form the L-address. C35 - 36 are used to control the next address multiplexer and return address stack.

The interpretation of bits C33 - 48 depends on the value of C16. If C16 = 1 then C33 - 48 is a 16-bit constant which is read by the MKS16 from the I-bus. This provides arbitrary constants without the use of a separate constant ROM. If C16 = 0 then C33 - 48 are external logic control signals. C33 - 34 control XIR reformatting. C37 - 38, C48 control the MULTIBUS™ interface. C39 - 40 control latches for the XIR, the memory address register (MAR) and the register file. Register file addressing and the XIR counters are controlled by C41 - 45. C41, C46 - 47 control the external condition multiplexer for testing the values of the XIR counters.

3.2 CONTROL STORE ORGANIZATION

Control ROM is organized as 2Kx48. The physical micro-address format is shown in Fig. 6. The distinction between page and word address is valid only for N-addressing. The return address for a subroutine call is obtained by incrementing the block address of the calling microinstruction. The microprogram is functionally partitioned so that floating-point and diagnostic microcode resides in blocks 4 - 7, while the rest of the microcode occupies blocks 0 - 3, and for this reason the return address generator increments the block address modulo 4 rather than modulo 8.

Fig. 3-MKS16 Microinstruction word

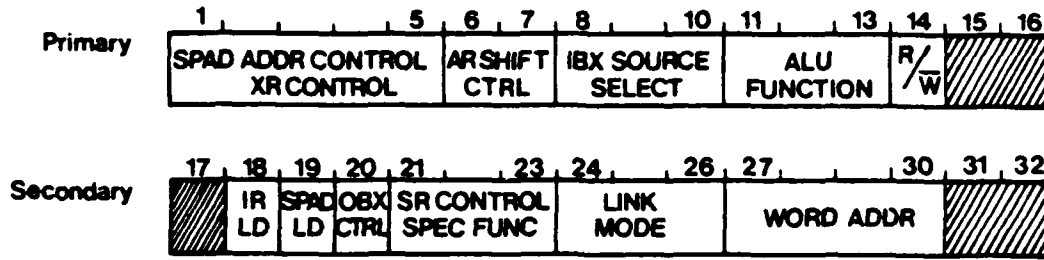


Fig. 4 - MKS1750 Microaddressing

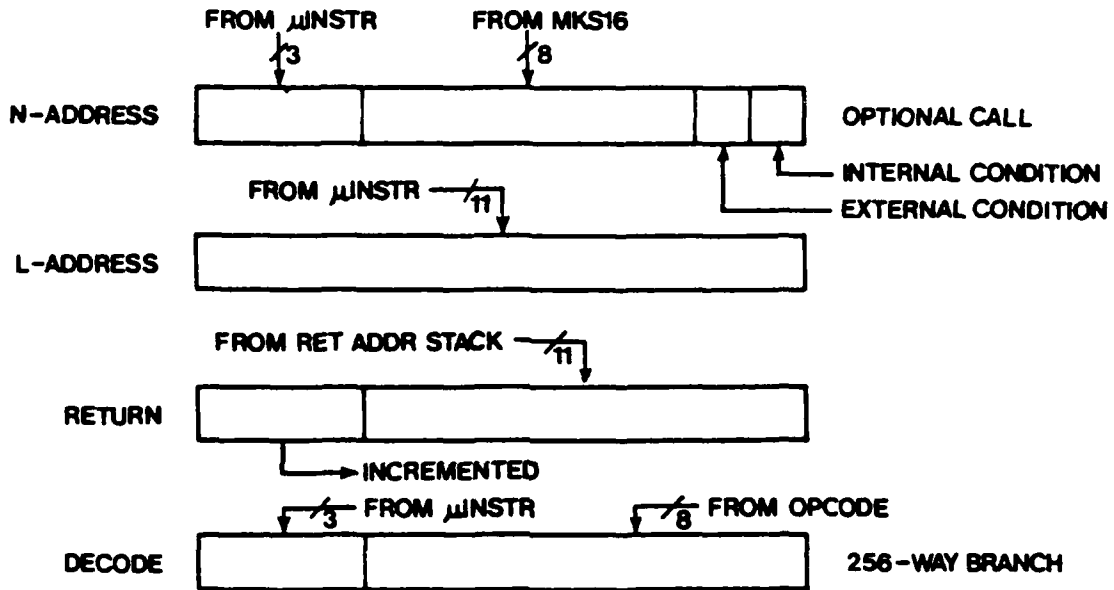


Fig. 5 - MKS1750 Microinstruction Word

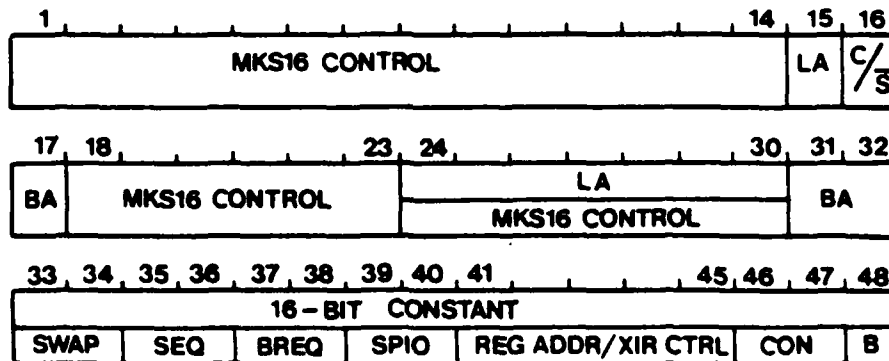
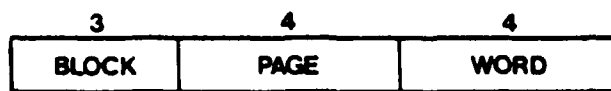


Fig. 6 - Physical Microaddress Format



3.3 MICROPROGRAM STRUCTURE

The MKS1750 microprogram makes extensive use of micro-subroutines; three levels are supported. The functions performed by the subroutines are determined by analyzing the 1750A instruction set in the following areas:

- o instruction format
- o address mode
- o length of operand(s)
- o precision of operand(s)
- o effect on condition codes

The execution of a 1750A instruction is divided into several phases, each of which requires only a subset of the above information. For example, effective address calculation only requires knowledge of the instruction's address mode. All effective address calculation routines return with the address loaded to the MAR and all execution routines leave their results in prespecified internal MKS16 registers. As a result, there are three "operand store" routines, for 16, 32 and 48-bit operands. Each arithmetic operation has one routine for each applicable data type; there are four ADD routines, for single/double precision fixed, floating and extended floating-point numbers. Effective address calculation is combined with the operand fetch phase, so that each 1750A address mode has one routine for each applicable operand length. Each of these routines is responsible for both calculating the effective address and fetching the correct number of operands.

3.4 REGISTER ALLOCATION

The MKS16 internal 16x16 register file (scratchpad) is used for working registers and to maintain the status of the 1750A machine. It is possible to perform a read-modify-write operation on an internal register during one microcycle. In addition, the MKS16 instruction register (IR) is used to contain the 1750A instruction word and the MKS16 status register (SR) is used to maintain the 1750A condition codes. (See Fig. 7)

Registers 0 - 9 are available as temporary registers for any instruction. Registers B - F are used to contain the 1750A status. Register F is dedicated to the instruction counter; register C contains a copy of the instruction word. Register B is used by the BEX instruction. Registers D and E are used in conjunction with the MKS16 SR to maintain the 1750A status word.

3.5 INSTRUCTION EXECUTION

Instruction execution starts with the fetch/decode phase. During this phase the IC is incremented, the next instruction is fetched, and the MKS16 tests for a valid pending interrupt. If there is a valid interrupt the fetch sequence is aborted.

The next microaddress is determined by the high byte of the instruction fetched, providing a 256-way branch. The effective address is then calculated and the operand(s) are fetched and loaded to internal working register(s). For 1750A immediate short address modes, one operand is extracted from the instruction word itself; for IC-relative mode (branch instructions) the jump address is calculated and loaded to a working register.

Final instruction decoding is performed at this point for base-relative indexed and immediate long modes, and the operation specified by the instruction is performed. The result is left in internal register(s). Conditional branch instructions update the IC if necessary, and arithmetic and logical instructions update the condition codes as required. The operand store phase transfers the results to the appropriate destination, usually the register file. The number of words transferred is determined by the precision of the instruction.

3.6 I/O

The MKS1750 implements I/O instructions as follows:

- o the command word is loaded to the MAR, and the processor issues an I/O bus request.
- o the data transfer takes place during the next cycle.

However, certain XIO commands require the microprogram to manipulate the status word or interrupt enable flag explicitly. The MKS1750 implements all mandatory 1750A XIO commands.

3.7 ABORT FAULTS

The 1750A Standard defines an "Instruction Set Architecture" to be the programmer's view of the machine. The standard specifies the state of the machine "between instructions" and does not address what happens during the execution of an instruction. As a result, there are certain "gray areas" in the standard which must be resolved in an implementation-dependent manner.

In particular, certain machine faults may occur as the result of a memory access during the execution of an instruction. If the execution of the instruction is allowed to proceed, the results are unpredictable (e.g. executing an instruction when the fetch caused a parity error). In the MKS1750, this situation is resolved by terminating the instruction, if any of the following "abort faults" occur: CPU memory protect fault, memory parity fault, or illegal address fault. When such a fault occurs, the microprogram jumps immediately to an abort fault handler. This microroutine uses an instruction-length flag to ensure that the IC saved during the subsequent machine error interrupt has a consistent value.

3.8 FLOATING-POINT ARITHMETIC

1750A floating-point instructions are implemented in firmware. The MKS16 contains shift control logic which permits detection and automatic correction of partial product overflow during multiplication, and of mantissa overflow during floating-point addition. The microprograms also use some MKS1750 features in non-standard ways. For example, the MKS1750 may perform multiple operations and test multiple conditions simultaneously. This is used in floating-point normalization, shown below. AR and XR contain the un-normalized mantissa, and XIR contains the exponent plus one, internally represented in excess 128 form:

```
708:  SLLX DECSD TESTSD TOV      :7,CI,8
709:  RRCX TOV                   :(exit)
70A:  XIR = IRCOPY;              :(underflow)
70B:  RRCX TOV                   :(exit)
```

The instruction at 708 is the single-instruction normalize loop. It shifts the entire mantissa left, decrements the exponent, and tests two conditions: overflow of the mantissa and zero of the exponent. If both are false, instruction 708 repeats. If the exponent reaches zero before the mantissa overflows, the next instruction logic selects 70A, which is the first instruction of the exponent underflow handler. Mantissa overflow indicates that normalization is complete, and the next instruction logic selects either 709 or 70B, depending upon whether the exponent has reached zero. Either case is acceptable, and so both instructions perform a right shift to correct for the mantissa having been shifted one bit too far.

3.9 ARCHITECTURAL INSTRUCTIONS

Several 1750A instructions require special attention. For example:

- o TSB - the microprogram inhibits external memory accesses during a TSB instruction by issuing a bus request every cycle, which causes the arbitration logic to lock the bus.
- o MOV - the microprogram checks for the presence of a valid interrupt between each single-word transfer. In addition, for the MOV instruction the microprogram does not increment the value of the instruction counter until all transfers are completed. Thus, if an interrupt occurs and is serviced, on return the execution of the MOV instruction will start over again. This scheme produces the desired result since the registers used by the MOV instructions always contain the correct values.

Fig. 7 MKS16 Register Allocation

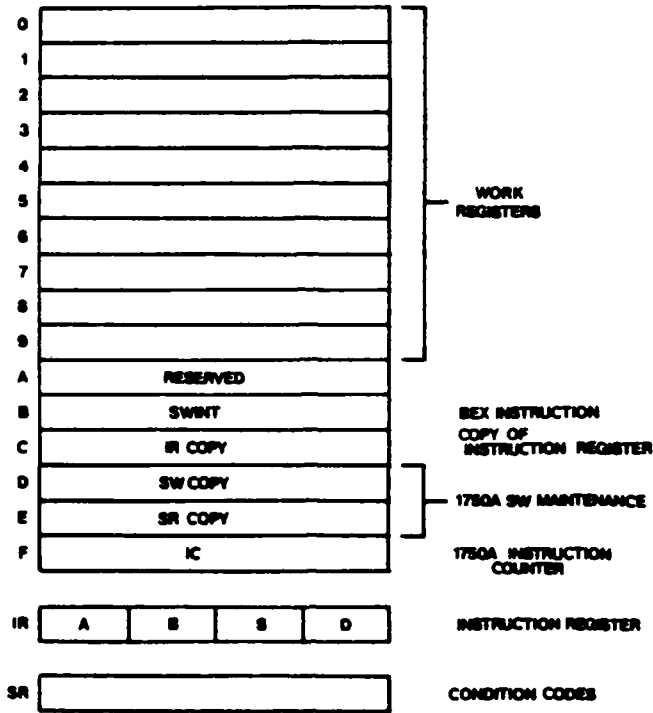
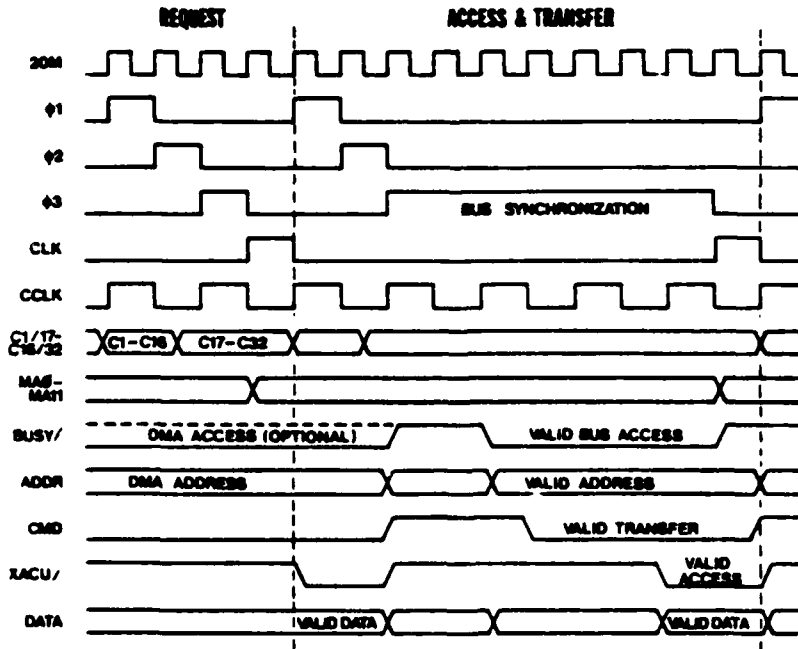


Fig. 8 MKS1750 System Timing



4. MKS1750 IMPLEMENTATION

4.1 FUNCTIONAL PARTITIONING

The partitioning is based on a 1400-gate 102-pin SOS/CMOS universal array manufactured by RCA. The MKS1750 logic is partitioned into three arrays. The boundaries of each array are shown as broken lines in Fig. 1. The emulating controller unit (ECU) contains the logic required for microsequencing, XIR manipulation, and external register control. The bus controller unit (BCU) handles bus transfers, system timing, logical addressing, and bus arbitration. The interrupt controller unit (ICU) processes interrupt requests and executes mandatory XIO instructions for PI, MK and FT. The external 16x16 register file is implemented using a second MKS16 chip. Six 2Kx8 ROMs are used for control store.

4.2 EMULATING CONTROLLER UNIT (ECU)

The ECU contains the logic required for instruction decode, register control, and microsequencing. The XIR provides instruction fields for register addressing. A 3x11 microaddress stack allows microsubrouting.

The XIR may be read into the MKS16 from the I-bus. The value may be modified to facilitate instruction decoding, as follows:

<u>C16</u>	<u>C33</u>	<u>C34</u>	<u>OPERATION</u>
0	0	0	NOP: no operation
0	0	1	SWAB: swap bytes
0	1	0	SWSD: swap S and D fields
0	1	1	SEXT: extend sign of low order byte
1	X	X	read 16-bit constant

The register file may be addressed by the XIR S or D fields, or by a four-bit literal address. The counters are controlled as follows:

<u>C42</u>	<u>C44</u>	<u>C45</u>	<u>OPERATION</u>
0	0	0	NOP
0	0	1	DEC(SD): 8-bit decrement
0	1	0	NOP
0	1	1	INC(SD): 8-bit increment
1	0	0	DEC(S): 4-bit
1	0	1	INC(S): 4-bit
1	1	0	DEC(D): 4-bit
1	1	1	INC(D): 4-bit

Bits C16, C35 - 36, C46 - 47 are used to control the microsequencer.

4.3 INTERRUPT CONTROLLER UNIT (ICU)

The ICU contains the logic required for the MIL-STD-1750A PI, MK and FT. In addition, a valid interrupt request to the processor is generated according to the following equation:

$$\text{valid interrupt} = PI_0 + PI_5 + (PI_n \cdot MK_n) \cdot ENB + PI_1 \cdot MK_1$$

where $n = 2, 3, 4, 6-15$ and ENB is the interrupt enable flag.

The Pending Interrupt Register is a set of sixteen flip-flops which are set and reset using the RPI, SPI and RPIR XIO commands. The Interrupt Mask Register is a set of sixteen flip-flops which are set and reset using the SMK and RMK XIO commands. MK is also saved and restored as part of the processor context by the interrupt microprogram.

When an XIO instruction is executed, the 16-bit command field is loaded to the MAR, and an I/O request is issued. For mandatory XIO commands, no system bus access is required. If the XIO command is not local (such as a programmed I/O channel), then bus arbitration is required. A timeout will occur if the I/O device is not present, and the appropriate FT bit is set.

The Fault Register is a set of sixteen RS flip-flops used for indicating machine faults. Setting any FT bit causes bit 1 of PI to be set. The FT is controlled by the RCFR XIO command.

4.4 BUS CONTROLLER UNIT (BCU)

The BCU provides logic to address memory and I/O devices, synchronize bus transfer cycles, supply processor clocks, and interface to the MULTIBUS™. A 16-bit logical address is provided by loading the MAR prior to a bus transfer. For memory management, the AS and PS fields of the SW are loaded to the 8-bit External Status Register (XSR).

During the request cycle, the address of the slave device (I/O or memory) is latched into the MAR if the device is not local, a bus request is issued, and the transfer type is latched. During the transfer cycle, the processor can access the bus if no higher-priority device is requesting it. The processor delays the CLK phase until XACK is received. CLK is then used to latch the slave data into a register, XIR, or MAR.

4.5 PACKAGING CONSIDERATIONS

The MKS1750 microprocessor is packaged as a set of eleven hermetically-sealed leadless chip carriers, which are standard JEDEC types. This includes:

- o three 132-pin type A carriers, for SOS gate arrays
- o two 48-pin type C carriers, for the MKS16 and register file

- o six 32-pin type C carriers, for 2Kx8 control ROMs

Leadless carriers provide greater packaging density as they are smaller than conventional DIPs and the shorter lead lengths improve the switching characteristics of the processor. The chip set may be mounted on a small ceramic card to accommodate the particular application form factor required (6).

4.6 SOFTWARE SUPPORT

Software support for MKS1750 development is based on the MIL-STD-1750A Support Software Package which was developed by McDonnell-Douglas under USAF contract. This package includes a macropreprocessor, assembler, linker, 1750A simulator and utilities for formatting and library maintenance.

This package, installed on a local IBM 3033 VM/CMS time-sharing system, was used to develop a resident monitor/debugger for the MKS1750. This monitor is compatible with the support software package, and allows users to develop and debug 1750A software during system prototyping. The monitor supports "quick look and change" commands for memory and 1750A registers, down-line loading of 1750A programs developed using the support package and program execution under breakpoint control. The monitor also includes I/O utilities to support two serial RS232C lines (for a terminal and downline loading) and optional dual floppy disk drives.

4.7 MKS1750 SUMMARY SPECIFICATIONS

ARCHITECTURE	full implementation of MIL-STD-1750A (Notice 1), based on 16-bit microprogrammable processor
TECHNOLOGY	SOS/CMOS - eleven chips
POWER DISSIPATION	less than 1W at 5.5 MHz, 10V
TEMPERATURE	-55°C to +125°C
SIZE	4 x 4 x 0.5 in. (typical)
RADIATION RESISTANCE	10^{10} rad (Si) transient, 10^4 rad total dose
MICROCYCLE TIME	180 nsec
PERFORMANCE	265 KIPS (100 nsec memory) 240 KIPS (250 nsec memory) using DAIS mix (16% floating-point) 450 - 540 KIPS (fixed point only)

ADDRESS SPACE 64K (optional memory management)

EXTERNAL
INTERFACE MULTIBUS™-compatible

5. SUMMARY

The MKS1750 is a new hardware/firmware realization of the standard 1750A architecture. The eleven-chip set is an all SOS/CMOS microprocessor which is high performance, low power, small in size and radiation resistant. These features make the MKS1750 attractive for a variety of Air Force and other embedded computer applications.

REFERENCES

1. J. Rogers, 'Nuclear Survivability Concepts for Airborne Computers', Trans. AIAA Conf. on Aerospace Computers, Oct. 1977.
2. E.M. Reiss, J.P. McCarthy and A. Bishop, 'A Versatile CMOS/SOS LSI Chip Set for Military Communications Applications', RCA Solid State Division, Somerville, NJ.
3. U.S. Dept. of Defense, "Military Standard Sixteen-Bit Computer Instruction Set Architecture", MIL-STD-1750A (USAF), July 1980 (Notice 1, Nov. 1981).
4. Intel Corporation, "Intel Multibus Specification", Santa Clara, California, 1979.
5. Henry Silcock, 'A User-Microprogrammable Sixteen-Bit Microprocessor for Military/Aerospace Applications', Proc. IEEE National Aerospace and Electronics Conference (NAECON), May 1981.
6. Darrell Barker, '16-Bit Radiation-Hard Emulating Computer', Proc. IEEE NAECON, May 1981.

MULTIBUS™ is a trademark of Intel Corporation.

This paper copyright Mikros Systems Corp. 1982.

DELCO ELECTRONICS STANDARD ARCHITECTURE
MILITARY COMPUTERS

PRESENTED BY CLIVE D. LEEDHAM

DELCO ELECTRONICS DIVISION
GENERAL MOTORS CORPORATION
Santa Barbara Operations
6767 Hollister Avenue
Goleta, California 93117

ABSTRACT

Delco Electronics accepted and participated with early Department of Defense efforts to define military standard computer architectures. This early acceptance enabled Delco Electronics to compete for and capture the first Department of Defense (Air Force) programs requiring a standard computer. Delco Electronics will continue to compete for Department of Defense programs requiring these standards. Specifically, Delco Electronics will offer to the Air Force and in the Army current and future Delco Electronics computer products incorporating the MIL-STD-1750A and MIL-STD-1867 architecture.

BIOGRAPHICAL SKETCH

OF

DR. CLIVE D. LEEDHAM

Dr. Leedham is currently Business Development Manager, Digital Systems, with the Delco Electronics Division, General Motors Corporation, located in Santa Barbara, California. Prior to this assignment he had undertaken various assignments within General Motors including Staff Specialist for the Defense Research Laboratories and Business Development Manager, Ocean Systems, with the AC Electronics Division.

Prior to joining General Motors he was on the faculty at the University of California, Purdue University, Massachusetts Institute of Technology and London University.

His military service was six years (1952-1958) with the Royal Air Force.

His B.S. is from London University, M.S. from the Massachusetts Institute of Technology, and Ph.D. from Purdue University, all in Electrical Engineering.

THE USE OF COMPUTER ISA AND SOFTWARE STANDARDS AT
WESTINGHOUSE DEFENSE ELECTRONICS CENTER

Dr. Manvel A. Geyer, John G. Gregory, and Harvey R. Moran, Jr.

Westinghouse Defense Electronics Center,
Baltimore, Maryland 21203
Telephone: (301) 765-7743, 4235, 2108

Abstract:

> Mil-Std-1750A is the fourth generation of computer hardware standards that have been embraced at Westinghouse since the early 1960's. The use of standards has had many advantages: rapidly maturing support software and hardware; a large pool of software engineers that can readily move from one program to another; lower recurring and non-recurring costs; less lead time on programs; similar or identical software benches for different programs. Mil-Std-1750A is presently utilized as an embedded computer on three major programs and has been proposed on several others.

Paper

Westinghouse has long espoused the use of standard ISA's and support software, not for love for standards, but for lower cost reasons. The use of standards makes our job easier, more efficient, and less costly, thus raising both our white and blue collar productivity. Fewer unique module designs require less engineering effort. Fewer designs also provide the means to use the same factory equipment, and personnel trained to use it, on multiple programs. This in turn lowers the cost of our products to the Government. Lower life cycle costs also result from a smaller number of unique modules which must be supported at depot level maintenance. Since designs are reused, system implementation time is also frequently decreased as a result of the ability to order parts early in the program cycle. And savings accrue due to volume production production.

In the late 1960's, Westinghouse utilized an internal standard ISA and standard hardware modules to supply digital computers to five different programs in less than one year. Without the standard ISA and standard hardware modules, these schedules and costs could not have been met -- the required number of hardware designers and software engineers would have exceeded our capabilities. Mil-Std-1750A provides us with all the benefits we had with our own internal standard and it is known to be acceptable to a wide selection of programs.

In the mid 1970's, Westinghouse built the AN/AYK-15 for AFWAL utilizing an ISA which was the basis for Mil-Std-1750. This ISA was specifically designed to ease the task of generating a Jovial compiler -- providing multiple registers, symmetrical addressing modes and hardware supported floating point arithmetic. During that time, Westinghouse was under contract with AFWAL to design and build the Electronically Agile Radar (EAR). With Air Force approval, Westinghouse decided to use the AN/AYK-15 design for the control computer of EAR. Thus a proven design with hardware and support software in place was used to improve white collar productivity. The AN/AYK-15 has since been used in several other programs.

Concurrent with the issuance of Mil-STD-1750A, Westinghouse was under contract to provide new modes and features for the AN/APG-66 (F-16) radar. Since these new modes and features required an enhancement to the existing radar computer, Westinghouse chose, with Air Force approval, to convert to the Mil-Std-1750A ISA. Since then, two other Westinghouse programs, BlB ORS and F-16 AFTI E-O have also embraced the standard. In fact, much of the same hardware is used in all three programs. The control CPUs for this programmable signal processor also utilize the Mil-Std-1750A ISA. The Westinghouse VHSIC Program is also using Mil-Std-1750A as the embedded control computer.

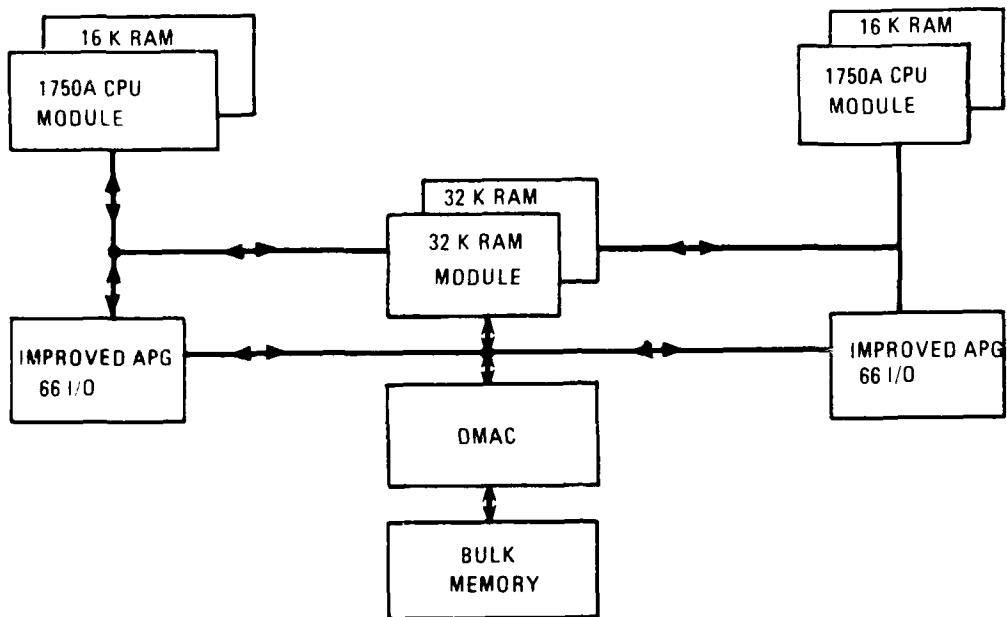
The general architecture of the processors within the systems is shown in Figure 1. The structure shown provides for up to four computers in a multi-processor configuration. Each processor design has modules which contains a Mil-Std-1750A computer. Figure 2 shows a picture of the 'computer' module which holds an Mil-Std-1750A CPU with interrupts, timers, and 16K words of dual ported, parity checked, RAM including write protection circuitry. Figure 3 summarizes its characteristics. Figure 4 is a block diagram of the computer.

The Westinghouse 1750A hardware consists of building blocks (plug-in-modules) which are normally embedded within other electronic equipment to save volume and weight. For required applications, the building blocks are assembled in a chassis which has its own plug-in power supply.

The building blocks consist of: the above mentioned 1750A computer module; a 64K word RAM module configured as 4 separate dual ported, parity checked, write protected 16K banks; a companion non-volatile 'bulk' memory available either as EPROM or in an electrically alterable technology. The system contains an I/O processor termed a Direct Memory Access Controller (DMAC) which boot loads RAM from bulk. The DMAC then accepts I/O transfer commands from either or both of two CPU's. The Bl-B configuration also contains a Mil-Std-1553B multiplex bus module which can act as both bus controller and remote terminal. This module is designed to require minimal attention from a CPU once initialized. Figure 5 shows a block diagram of the 1553 interface.

Computer and memory modules are identical in all applications. This has many advantages in non-recurring cost and schedule, with even more in the recurring cost. With respect to most avionics buys, the total quantity of

IMPROVED APG-66



B1 RADAR

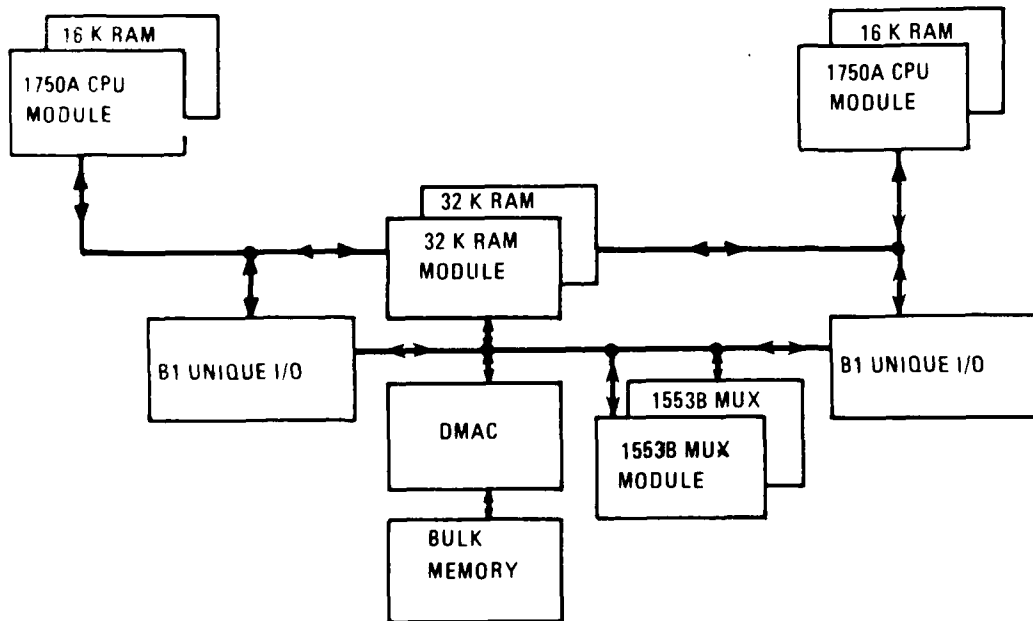


Figure 1. General Radar Processor Architecture

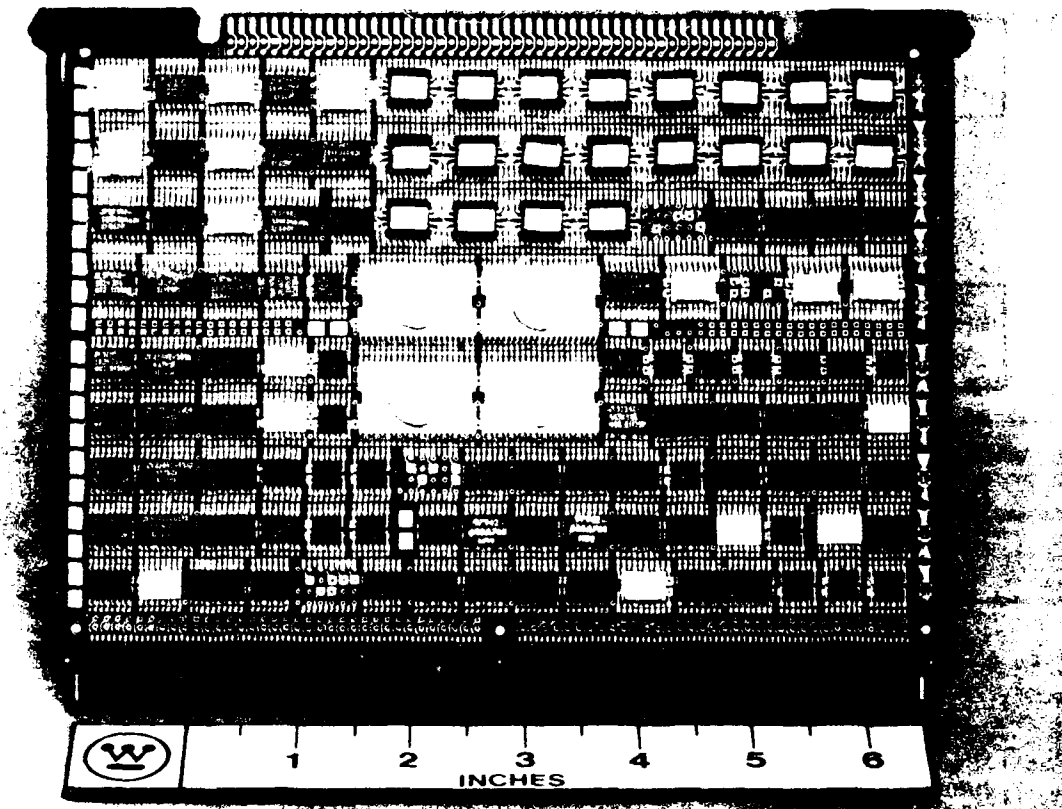
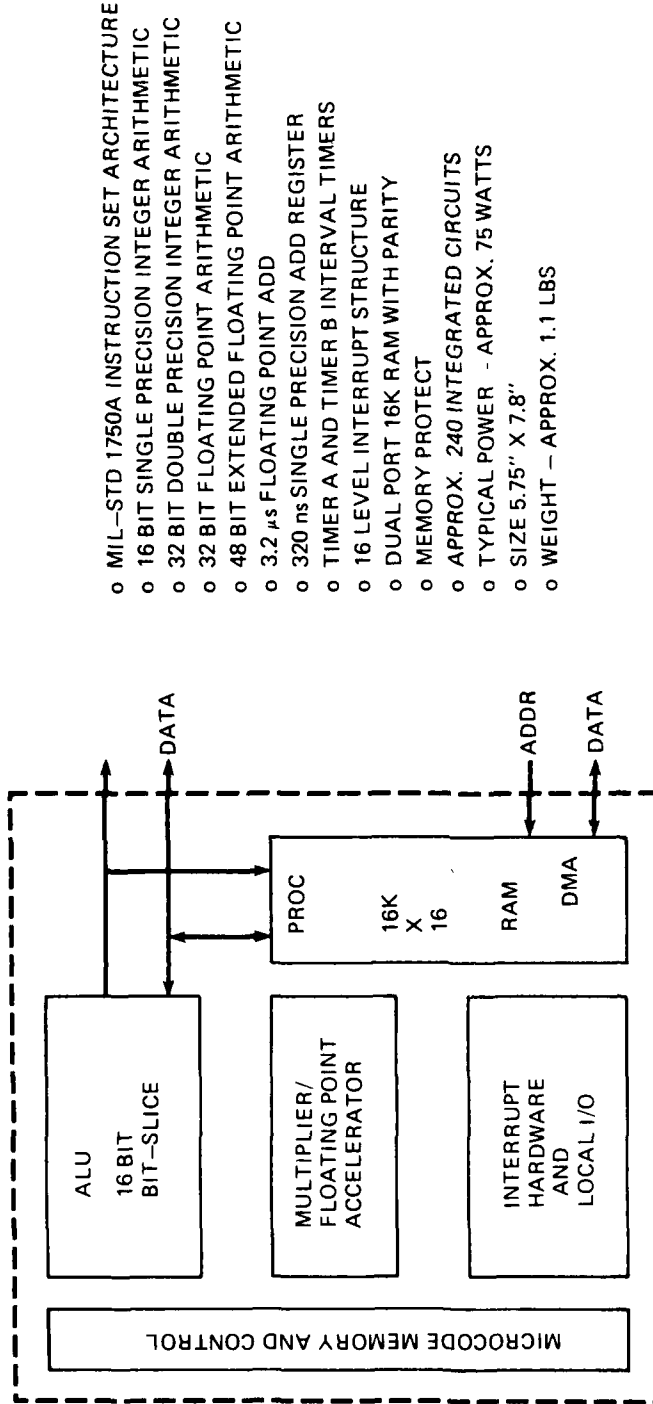


Figure 2. Picture of Embedded Mil-Std-1750A Computer



- o MIL-STD 1750A INSTRUCTION SET ARCHITECTURE
- o 16 BIT SINGLE PRECISION INTEGER ARITHMETIC
- o 32 BIT DOUBLE PRECISION INTEGER ARITHMETIC
- o 32 BIT FLOATING POINT ARITHMETIC
- o 48 BIT EXTENDED FLOATING POINT ARITHMETIC
- o 3.2 μ s FLOATING POINT ADD
- o 320 ns SINGLE PRECISION ADD REGISTER
- o TIMER A AND TIMER B INTERVAL TIMERS
- o 16 LEVEL INTERRUPT STRUCTURE
- o DUAL PORT 16K RAM WITH PARITY
- o MEMORY PROTECT
- o APPROX. 240 INTEGRATED CIRCUITS
- o TYPICAL POWER - APPROX. 75 WATTS
- o SIZE 5.75" X 7.8"
- o WEIGHT - APPROX. 1.1 LBS

Figure 3. Westinghouse Mil-Std-1750A CPU/MEM Module

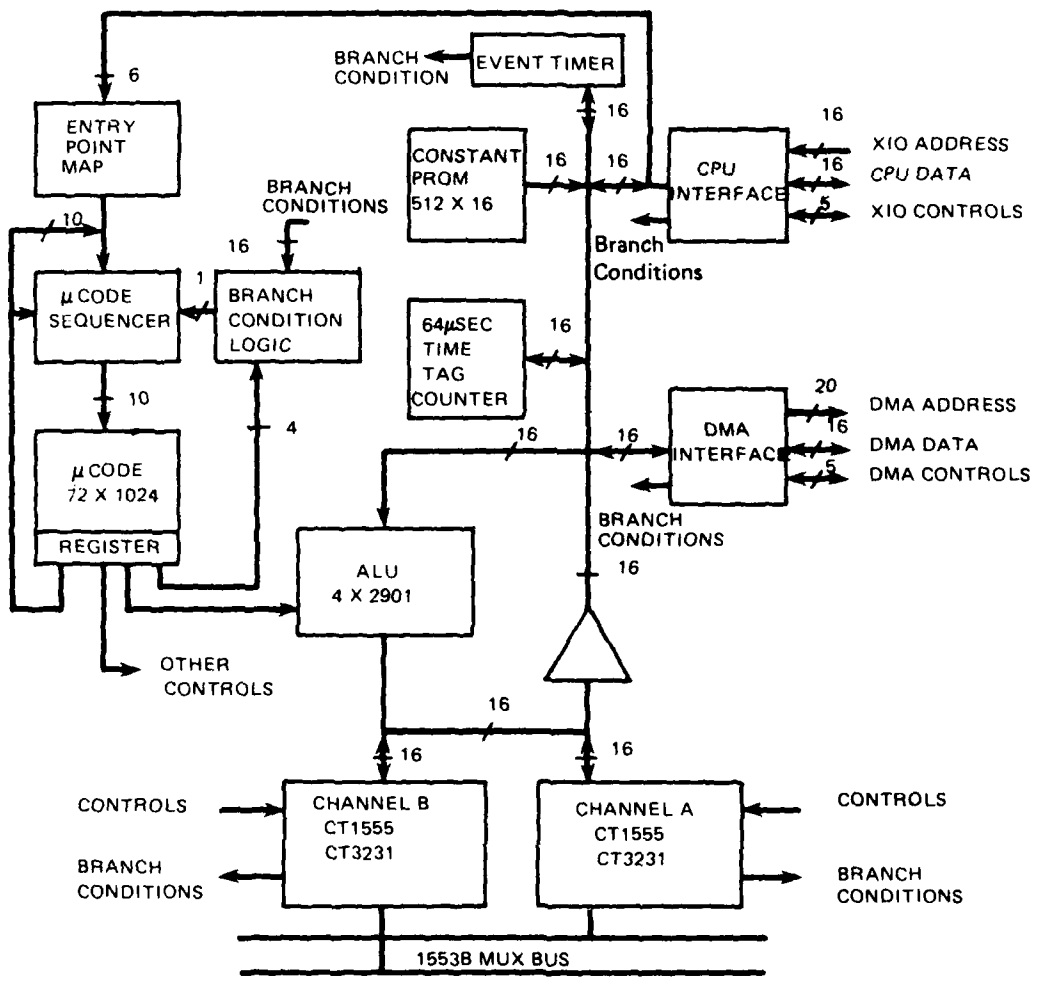


Figure 4. CPU/MEM SRU Block Diagram

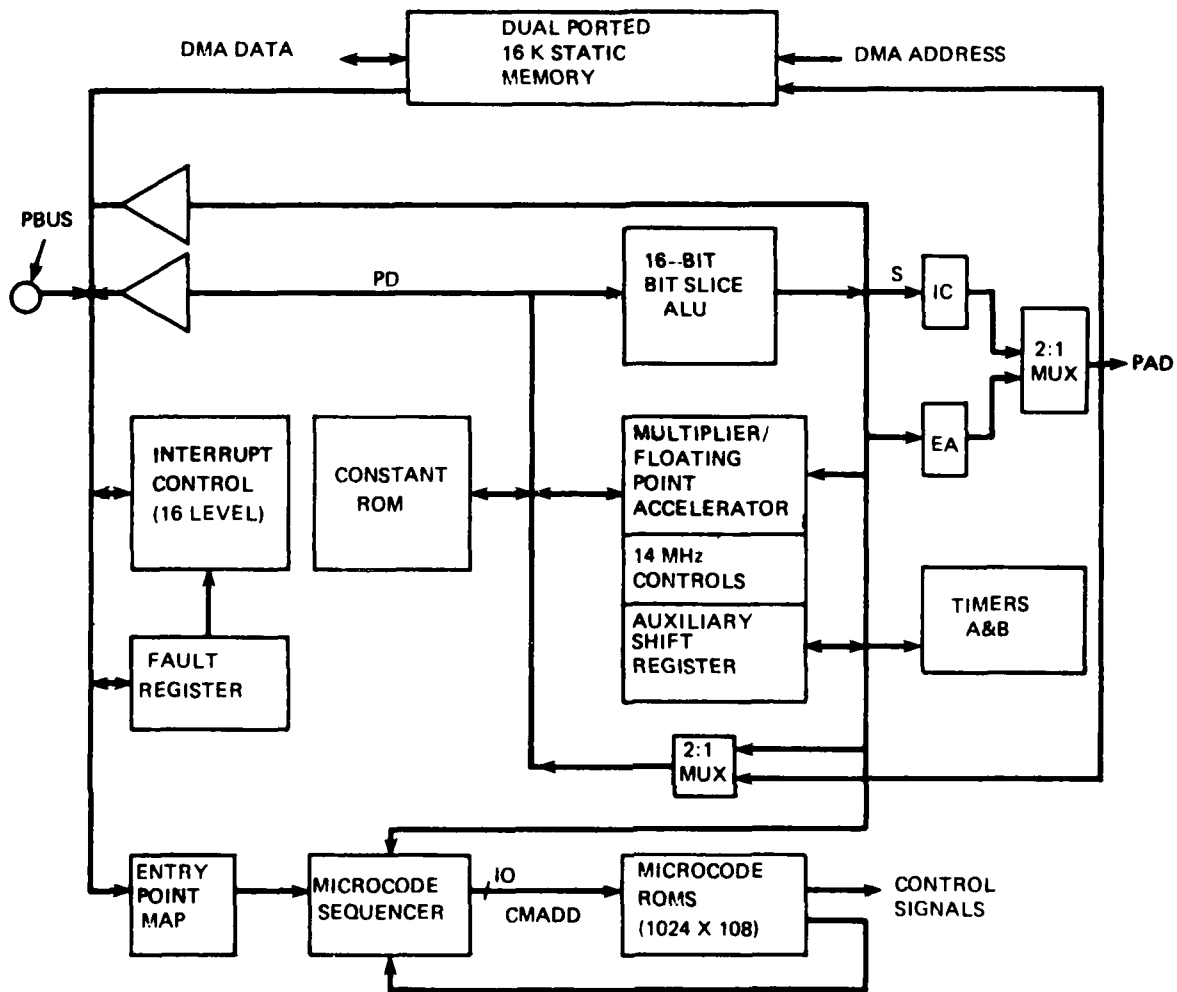


Figure 5. 1553 Bus Controller Block Diagram

each part type (over 30 computers produced per month) becomes very large. The disadvantage is in configuration control. To offset this, a configuration control board staffed by appropriate personnel from each program, has been established to cover all common modules.

Common support hardware and software have proved to be great advantages. Over ten similar or identical software benches have been constructed for three major programs. Consoles are identical or upward compatible, allowing new features, which succeeding programs add, to be retrofittable into previous consoles.

Westinghouse has installed, and is using, the Air Force 1750A Jovial J73 compiler and support software package. The symbol table entries which are available have been integrated into the debug environment of our consoles. The present environment is to use an IBM host computer for compilations associated with all three programs and to downlink through a number of VAX host computers to the 1750A software and system benches.

In order to achieve additional code optimizations, and advance symbolic information into our debug environment, Westinghouse is now in the process of containing the complete host computer and supporting hardware in a localized environment. This will also provide for classified software development. The complete software support package, including compilation, is being hosted on VAX computers which are installed near the 1750A software and system benches. A performance monitor utilizing a PDP-11 is also available.

Presently, there has been approximately 40,000 lines of code generated with the J73 Mil-Std-1589B compiler. By early 1983 approximately 250,000 lines of code will have been generated. Support software added to the package by Westinghouse includes "automatic" documentation and configuration control as well as a J73I to J73 translator which can directly translate over 90% of the code. The support software package is rapidly becoming mature due to the number of users and the of problems being solved.

In conjunction with another program, we are presently developing an Ada, Mil-Std-1815 compiler for a VAX host and a Mil-Std-1750A target using Mil-Std-483 as a guideline for documentation.

Advantages of Software Standards and ISA Standards

- o Support software matures rapidly due to the multiplicity of users and combined problem reports.
- o A large pool of software engineers who can move readily from one program to another with a minimum of training is created.
- o Software benches can be identical and shared across programs.
- o User consoles features become sophisticated due to each program adding capabilities to a proven base.

- o Software development can begin at the outset of the contract.
- o Identical problems can have identical application software packages which are transportable between programs.
- o Non-recurring costs are lower due to fewer designs.
- o Maintenance inventory costs are lower due to fewer designs.
- o Recurring costs are lower due to the economies of volume production both in delivered and support equipment.
- o Lead times are reduced as a result of early ordering of parts for existing designs.
- o Technology innovations which do not affect architecture can be readily retro- or forward-fitted.

Disadvantages of Software Standards and ISA Standards

- o The delivered hardware must be slightly more powerful than in the past to account for compiler inefficiencies and software design techniques made feasible by the abstraction level of HOL programming.
 - o The delivered hardware cannot be readily tailored (minimized) for a program with reduced requirements; e.g., one with no use for floating point arithmetic.
 - o Technology innovations which have architectural impact can not be used to full advantage.
 - o Hardware and support software configuration control must be well managed across multiple programs.
-

Manvel A. Geyer - PhD, 1969 from the Ohio State University. Previously Program Manager for the AN/AYK-15 and AN/AYK-15A computers. Served on the SAE A2-K Committee since 1970. Served on the task group which generated Mil-Std-1553 and Mil-Std-1553B. Joined Westinghouse in 1955.

John G. Gregory after graduating from Johns Hopkins University started his career in computers at the Army Aberdeen Proving Ground working on the first Electronic Calculator ENIAC and the first real digital stored Program Computer EDVAC. He designed and put into operation the last large tube vacuum computer which was the fastest of its day. In 1962 he joined Westinghouse where he contributed to parallel processing computers. From there he worked on early digital integration concepts. Presently, he is Manager of Data and Signal Processing Programs.

Harvey R. Moran, Jr. is a Senior Engineer in Computer Design at the Westinghouse Electric Corp. Defense Electronic Center in Baltimore, Md. He participated in the design of Radar Computers for the F-16, DIVAD, E3A, improved F-16, and B1 programs. Current interests lie in architectural support for HOL and operating system constructs. He received a BSEE in 1969 from the University of Maryland. He is a member of the IEEE Computer Society and the ACM.

AD-P003 553

"TRANSPORTABLE GPU CHIP SET TECHNOLOGY FOR
STANDARD COMPUTER ARCHITECTURES"

Robert E. Fosdick

And

Harvey C. Denison

Tracor Aerospace

6500 Tracor Lane

Austin, Texas 78721

(512) 926-2800

> The USAF-developed GPU Chip Set has been utilized by Tracor to implement both USAF and Navy Standard 16-Bit Airborne Computer Architectures. Both configurations are currently being delivered into DOD full-scale development programs.

Leadless Hermetic Chip Carrier (HCC) packaging has facilitated implementation of both architectures on single 4½" x 5" substrates. The CMOS and CMOS/SOS implementations of the GPU Chip Set have allowed both CPU implementations to use less than 3 watts of power each.

Recent efforts by Tracor for USAF have included the definition of a next-generation GPU Chip Set that will retain the application-proven architecture of the current chip set while offering the added cost advantages of transportability across ISO-CMOS and CMOS/SOS processes and across numerous semiconductor manufacturers using a newly-defined set of common design rules. The Enhanced GPU Chip Set will increase speed by an approximate factor of 3 while significantly reducing chip counts and costs of standard CPU implementations.

INTRODUCTION

Tracor's efforts in computer implementation began in June of 1973 when under a contract with

SAMSO, Tracor became involved with the architecture, logic design, simulation, circuit design, and layout organization of a General Processor Unit (GPU) using LSI technology. This effort led to the development of high-performance circuits fabricated with the CMOS/SOS process. In February, 1977, Tracor received a contract from the Air Force Avionics Laboratory to review the control requirements of the AN/UYK-20, AN/AYK-14, PDP-11, and SKC-2000 processors and define the CMOS/SOS LSI controller requirements to maximize utilization and minimize custom designs. The Emulating Microcontroller Device (MCD) that resulted, along with the GPU and a family of Gate Universal Arrays (GUA's) fabricated in the same CMOS/SOS process, became the basis for the USAF GPU Chip Set.

Tracor's leadership position in LSI technology for implementing high-performance computer architectures led in 1978 to a contract with the Naval Ocean Systems Center (NOSC) to implement the basic instruction set of the AN/UYK-20 computer utilizing the Air Force CMOS/SOS circuit family.

With the need arising for improving packaging densities for highly complex electronic circuitry used in LSI implementation, Tracor began investigating the Hermetic Chip Carrier (HCC) technology and its use in packaging CMOS/SOS circuits for military applications. Tracor's confidence in the future growth of HCC technology resulted in the commitment of corporate IR&D funds and the investment in capital equipment to develop a facility for assembling and testing circuit card assemblies populated with HCCs.

Tracor's efforts in developing the HCC manufacturing technology led to a contract in October, 1980, from the Air Force Materials Laboratory to establish HCC assembly to packaging and interconnect (P/I) structures as a viable manufacturing technology.

Currently, Tracor has one of the few complete production HCC facilities in the country. The line features automatic placement equipment with provisions for automatic testing of individual devices. Supplementing the HCC assembly facility is a thick-film processing facility used to manufacture ceramic Printed Wiring Boards (PWB).

Tracor's efforts as a contractor/contributor in high-performance computer technology development for the Navy and Air Force resulted in a contract with the Magnavox Government and Industrial Electronics Company to develop the CPU for the Receiver Processor Unit of the user segment of the GPS/NAVSTAR System using the Air Force CMOS/SOS LSI circuit family with a modified AN/UYK-20 emulation. The GPS/CPU design includes memory management and uses HCC packaging technology throughout. The HCCs are mounted on a 4.5" x 5" substrate which is assembled to the Magnavox motherboard in a piggy-back configuration.

Tracor is also under contract with the Air Force Avionics Laboratory to develop a 16-Bit Radiation Hard Emulating Computer (RHEC) that will implement two target architectures, one of which will be MIL-STD-1750A, using the Air Force CMOS/SOS LSI circuit family. Two new GUA circuit designs were developed during this program to facilitate MIL-STD-1750A implementation. The other target architecture is the Navy UYK-20/AYK-14.

In October 1981, Tracor received a contract from General Electric to develop a MIL-STD-1750A computer to fit on a three-quarter ATR circuit card to be used in the F5G radar system. This MIL-STD-1750A CPU is the baseline product that will be used as a yardstick for enhancements to the USAF GPU Chip Set.

BASIC CPU ARCHITECTURE

Figure 1 is a block diagram of the basic architecture. Each block in the figure represents

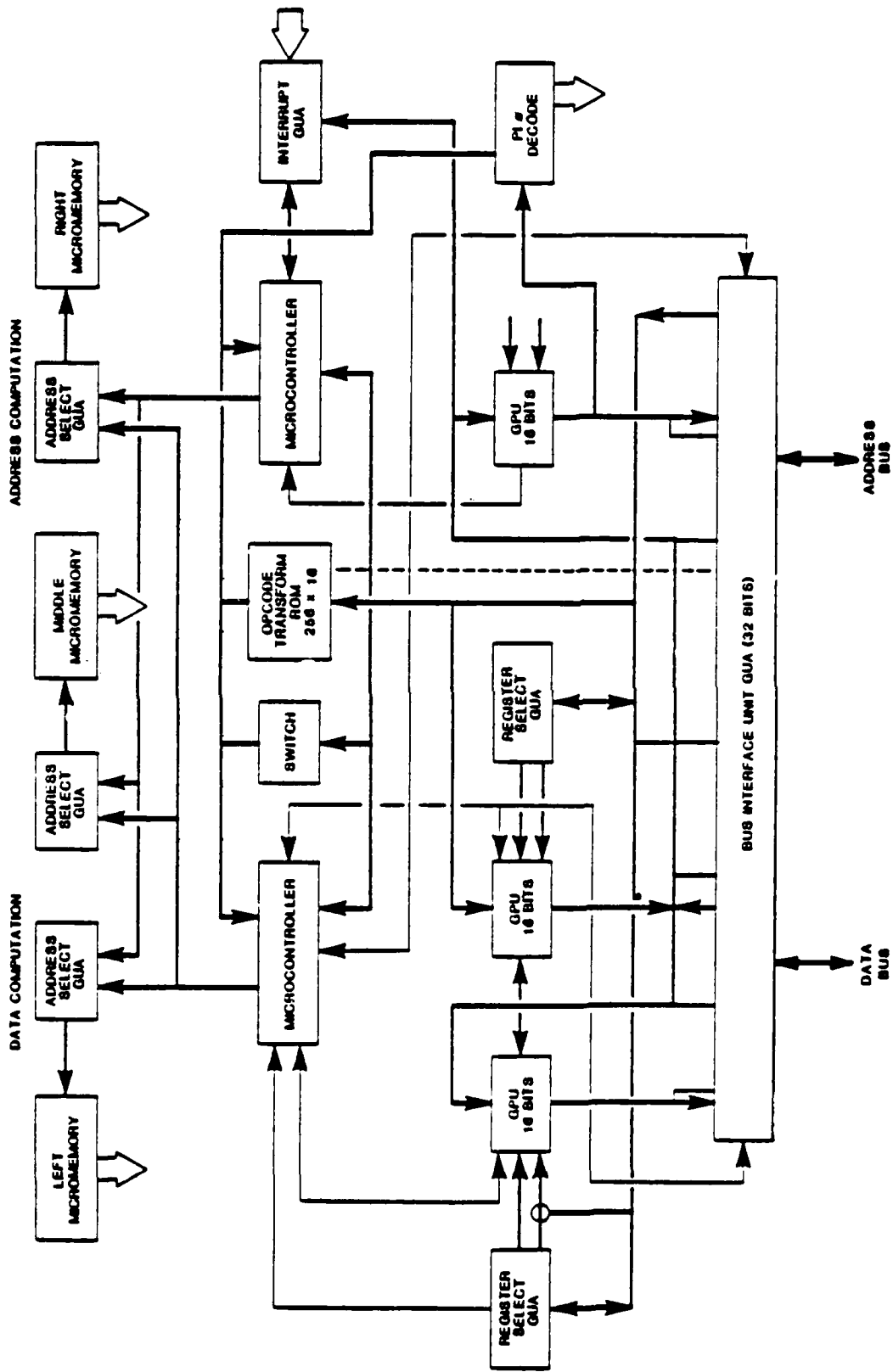


Figure 1. Basic Architecture

a functional circuit used one or more times. The entire implementation of the CPU is accomplished with seven generic LSI circuit types.

GENERAL REGISTER ORGANIZATION

The computation section of the CPU contains two pairs of General Processor Units (GPUs). Each GPU is an 8-Bit slice. The GPU pairs are referred to as an "upper ALU" (Arithmetic Logic Unit) and a "lower ALU" deriving their designations from the respective most-and least-significant halves of 32-Bit arithmetic computations. The dual 16-Bit ALU also provides a 32-Bit parallel arithmetic capability.

Each GPU pair contains sixteen 16-Bit registers. The internal registers are assigned to be the 16 general registers addressed by the 4-Bit register fields of the respective macroinstructions. The 16 general register assignments are identical in the two GPU pairs. In other words, at the end of each macroinstruction the contents of the registers of the upper ALU are the same as the contents of the respectively addressed registers of the lower ALU. The dual 16-Bit ALU provides the following advantages:

- a. 32-Bit parallel configuration simplifies double-precision arithmetic, and floating-point arithmetic macroinstructions.
- b. Requires less microprogram memory.
- c. Enhances execution speed for floating point and double precision macroinstructions.
- d. Avoids multiplexing of GPU status signals.
- e. Provides efficient built-in-test capabilities.

The ALU is further extended to 48-Bits by concatenating the Address Computations GPU pair

for extended Precision Floating Point. Therefore, the CPU has the ability to configure a 16-, 32-, or 48-Bit ALU as required by the microprogrammer.

CONTROL

The architecture uses two independent micro-controllers. The approach incorporates several concepts designed for more efficient utilization of micromemory that results in reduced part count and minimization of the number of cycles for Register-Register operations.

The dual-controller approach is based upon the observation that performance of most macro-instructions can be accomplished with two sequences of microcode, both of which can be independently common to other macroinstructions. One sequence is classification. Macroinstructions are classified by whether they use a second general register, a general register pair, a macroinstruction literal field, a bitposition mask, a memory word, or a memory double word as an operand source. In most cases, this is called the operand derivation. The other sequence is mathematical function. Among these functions are load, add, subtract, one's complement, negate (two's complement), compare, and logical AND and OR. This sequence is called execution.

The micromemory is divided into three sections. The ASU provides the means by which each section can be allocated to either controller in a well-defined manner. One section is primarily addressed by the classification controller. This section contains most of the control fields to the circuits that are used mainly for fetch, operand derivation, and interrupt control. Another section is primarily addressed by the execution controller. This section contains most of the common controls necessary to perform function execution of the macroinstruction. The third section is in the

middle, and control of it is passed from the classification controller when operand derivation is complete to the execution control to finish functioning processing. The middle section provides the missing control signals to indicate the operand location and controls the Bus Interface Unit (BIU).

This architecture is designed to give optimum performance with no duplication of microcode. The same microcode that performs a single precision add is used for all corresponding addressing and data formats. This approach also allows for better packaging of microcode because, with sectioned micromemory, much of the respective microcode can occupy the same address space. At times the program being executed in each controller may require the entire processor to perform a task.

MICROMEMORY

The micromemory is implemented with 4K CMOS/SOS ROM circuits. The ROM circuits contain a transparent latch to hold the microword for the current operation simultaneous to accessing the next microword.

OPCODE TRANSFORM

The remapping or ordering of the instruction opcodes requires a small memory (512 X 16-Bits). The memory is implemented with 4K ROM circuits.

STATUS REGISTER

The CPU contains a distributed status word (SW). In other words, depending upon the function, the fields or individual bits are stored where used and recombined when the SW is output.

INTERRUPT

The Interrupt Control Unit (ICU) circuit is

an 8-BIT slice containing all of the circuitry required to perform the functions defined by the Interrupt System Flowchart (Figure 2 of MIL-STD-1750A). The circuit is concatenatable and requires two circuits to implement the 16 interrupts.

PIO DECODE

The 256 X 16-Bit (4K) ROM circuit is used to decode the XIO operand into individual control signals. Four outputs are used to set up entry points to the MCU for XIO operations requiring several microwords (i.e., READ TIMER, LOAD TIMER, READ FAULT REGISTER, etc.).

INPUT/OUTPUT

The CPU has an interface for a multi-Channel, High-Speed Memory Controller. The CPU places no constraints on the DMA process. The CPU can also interface with any type of random access memory. The CPU Control Panel Interface is an EIA standard RS-232C Serial I/O Channel which operates independently of the CPU Microcode Control Logic. The CPU transfers data to and from the RS-232C Interface using the programmed I/O instructions.

TIMERS A AND B

Each ICU device includes an 8-Bit concatenable clock circuit that is loaded, read, and enabled under microcontrol. The two circuits will implement the two timers by using the 100K cycle and 10K cycle for Timer clock inputs generated by the 10MHZ Oscillator and divide input timing circuit.

CPU DEVICES

The circuits selected for the CPU realization utilize the Air Force GPU chip family and other compatible CMOS devices wherever practicable. These devices consist of two custom arrays, ROM and five customized Gate Universal Arrays (GUA) as listed in Table 1.

TABLE 1
GPU CHIP SET DEVICES IN TRACOR MIL-STD-1750A CPU

<u>Part Number</u>	<u>Designation</u>	<u>Function</u>	<u>Design Concept</u>	<u>Usage (Per CPU)</u>
ST67454	GPU	8-Bit ALU Slice	Custom	6
ST67453	MCU	Microcontroller	Custom APAR	2
ST67451	ROM	Micromemory	Custom	26
ST67496-0001	RSU	Register Select	GUA	1
ST67456-0002	BIU	Bus Interface	GUA	4
ST67455-0001	ASU	Address Select	GUA	3
ST67456-0003	ICU	Interrupt/Fault Reg	GUA	4
ST67510-0001	ILT	Interrupt Latch	GUA	3

INTERFACE CIRCUITS

The CPU interface power and signal conditioning circuits will consist of compatible standard devices including an oscillator, three 54LS162 counter/divider, 5407 buffer, a CD4024B counter, five CD40116 buffers, a resistor array, and the startup ROM. The oscillators and the counters will provide the external clock functions.

BASELINE CPU PHYSICAL DESCRIPTION

The devices listed in Table 1 plus four each CD4000 series support circuits (53 devices) are mounted on a 4.6-inch X 5.1-inch single-sided thick-film substrate which is then mounted to a conventional 3/4 ATR PWB as shown in Figure 2. The substrate assembly includes chip resistors and capacitors and two circuit protection devices. The complete CPU design includes the power and signal interface devices, and connectors.

The CPU operates with 10 volts in the CMOS/SOS LSI implementation and uses 5 volts to implement the external clock functions. Tracor can provide the CD40116 level convertors as shown in Figure 2 mounted on the host PWB to interface with 5VDC user memory and Schottky TTL signal interfaces. The two bus networks may be isolated to allow simultaneous operation.

Tracor's CPU weighs less than 0.75 pounds, including the host PWB, and occupies approximately 16 cubic inches.

CURRENT PRODUCT STATUS

Under the GPS User Equipment Contract with Magnavox Advanced Products Division, Torrance, California, TRACOR commenced deliveries of UYK-20/AYK-14 Architecture Single-Card CPU's in March 1982. Through 1983, approximately 100 preproduction units are scheduled for delivery. Volume production for GPS will commence in 1984.

Under the MIL-STD-1750A CPU contract with G.E., TRACOR commenced deliveries of preproduction single-card units in September 1982.

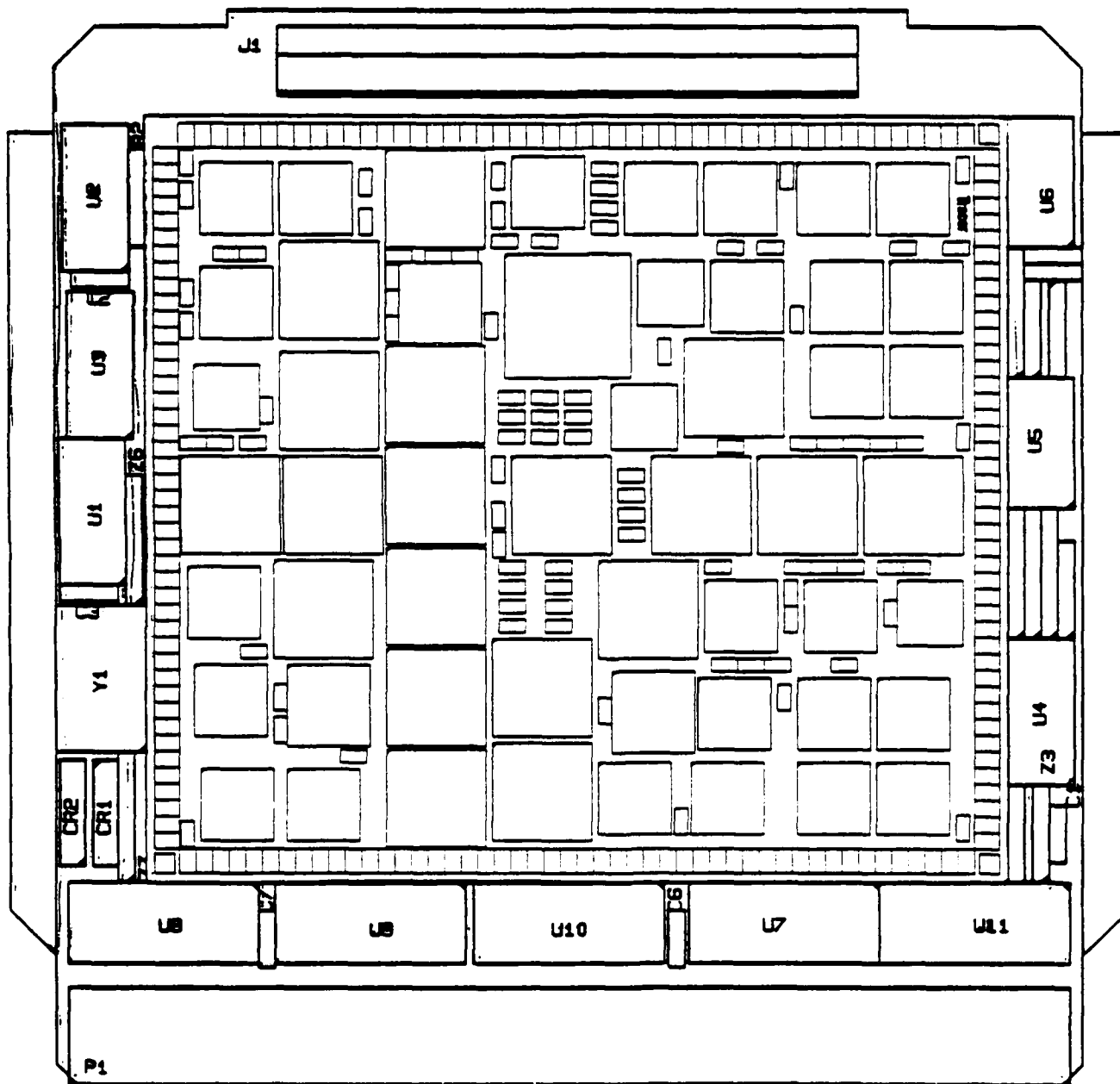


FIGURE 2 CURRENT MIL-STD-1750A CPU ON 3/4 ATR HOST PWB

Production for this program will commence in 1983.

ENHANCED GPU CHIP SET

Development efforts are currently commencing on the next generation GPU Chip Set. The overall objective of this evolutionary upgrade is to incorporate available new circuit technology to the maximum practicable extent while minimizing impacts to the architecture of the chip set. The reason for the latter is to take maximum advantage of the existing chip set architecture, which has evolved through the implementation of two major standard military instruction set architectures, the AN/UYK-20 and MIL-STD-1750A.

A special set of objectives has been set for the enhanced chip set based on potential user requirements and high confidence projections of what can be done with current circuit technology.

Performance-wise, we have an interim goal to provide a throughput capability of 500 KOPS, based on the full DAIS Mix, in the 1984 time frame. This is a "half-generation" update and requires little or no architecture change, primarily new, higher density implementations of the existing devices. The "full-generation" update calls for a throughput of 1 MOPS based on the full DAIS mix and requires some architectural enhancement. This "full-generation" update is projected to be available in 1985.

Figure 3 is a generalized illustration of the distributed network architecture targeted for MIL-STD-1750A application. In addition to the processor elements, the system would include data buffers, system input/output modules and central memory modules. Data and variables would flow through the system on separate buses.

DISTRIBUTED PROCESSING SYSTEM

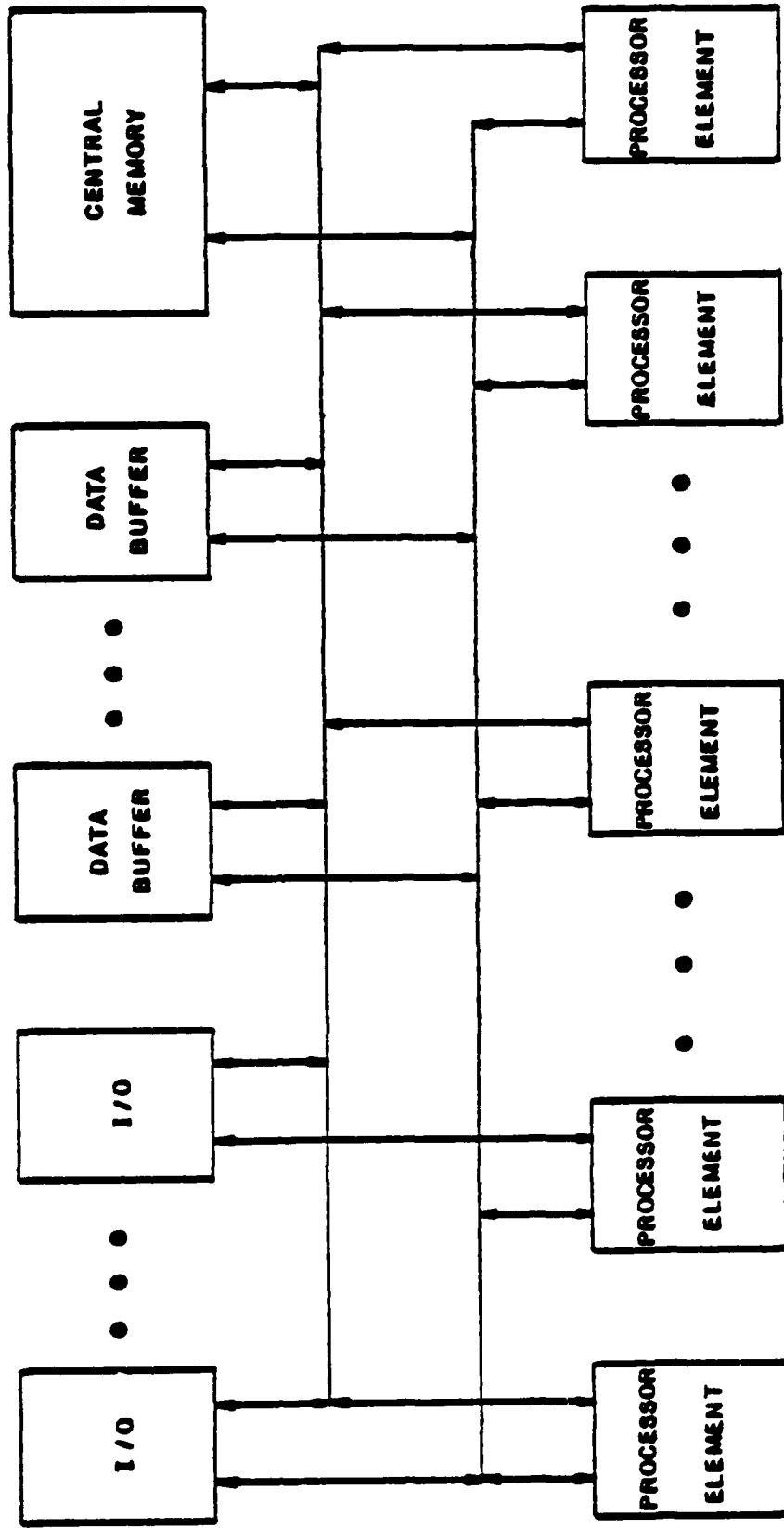


FIGURE 3

The data buffer will be a random access memory module used for buffering blocks of input/output data, providing residence for files of data shared by two or more of the distributed processor elements, and serving as a system level cache memory and a postman for messages within the system.

System input/output will consist of allocatable assets that can be used to set up autonomous channels in response to commands originating from the system executives residing in individual processor elements. Global data records would be buffered via the system data bus. Multiple word inputs could be directed to specific processor elements via the system variables bus.

Central memory would consist of large blocks (1 megaword or larger) of bulk storage, e.g. bubble memory or equivalent.

System buses would segregate global data from global variables. The variables bus would transmit commands to set up transfers, pass variables, request status and other high priority communications. This bus could be MIL-STD-1553B compatible if desired. The data bus would transmit large blocks of data within the system under the control of commands transmitted on the variables bus.

Figure 4 is an illustration of the distributed processor element architecture appropriate for this Hierarchical Network Application. It depicts the architecture changes required which, although transparent to MIL-STD-1750A, will have some effect on GPU Chip Set Architecture, particularly the Bus Interface Unit and the Projected Memory Management Unit.

The Memory Management Unit (MMU) device design will be architecturally keyed to the targeted distributed system requirements. It will be designed to be the functional interface between the processor element (local) and system (global) addressing

PROCESSOR ELEMENT

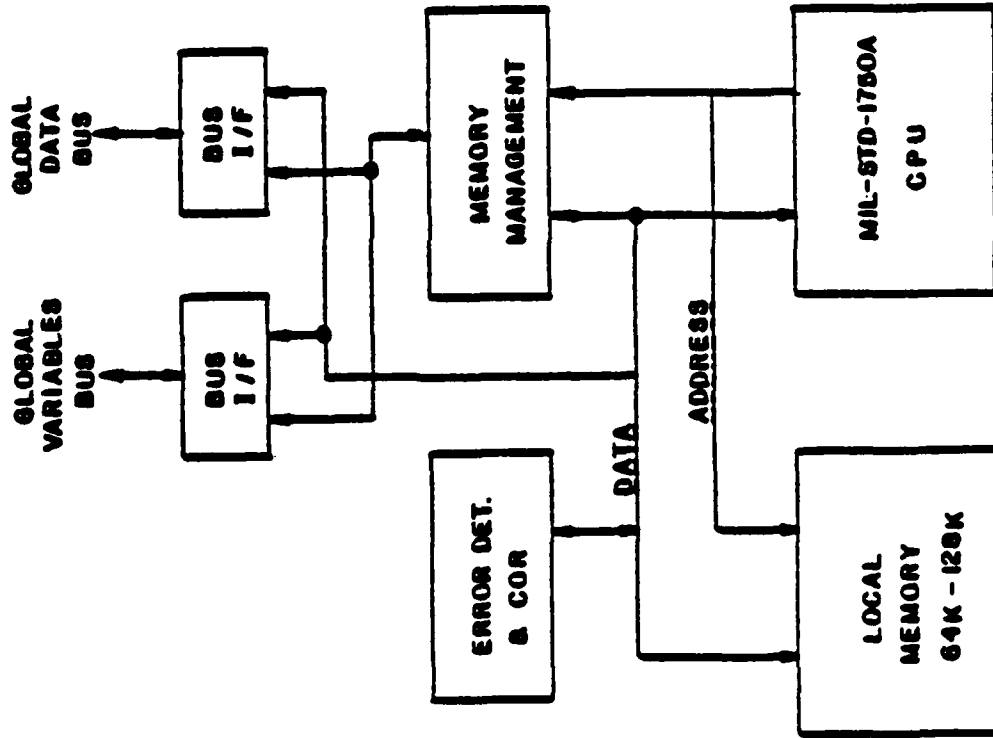


FIGURE 4

systems. It will contain 4 sets of 64-page registers to provide a 1 million word address space at the system level. There will be provisions made for adding additional sets of page registers. The MMU will also have access comparison and memory protect comparison functions per MIL-STD-1750A. The CPU Bus interface will potentially change to segregate data and address functions into local variables and local data. This would segregate local memory also and expand it's capacity from 64K to 128K words.

Architecture changes in the MIL-STD-1750A CPU, other than the Bus interface, will be primarily aimed at throughput enhancement to achieve 1 MOPS on the DAIS mix. The operand derivation (right) side functions and the execution (left) side functions of the CPU will be further separated and pipelined to a higher degree. One or two levels of instruction prefetch and partial decode will be added in parallel. "Referenced" register status will be maintained to aid in pipelining of the operand derivation (right) side of the machine. Double word fetch in a single access will also be considered. Bidirectional CPU Bus interfaces will reduce time devoted to inter-bus switching. Look-up tables housed in ROM devices will be considered for implementation of high-speed function conversions. A wider shift path will be added and an addition of a high-speed parallel multiplier device (existing design) will be traded against multiply on the ALU chip for speed. Although architectural considerations, these GPU changes will have some impact on the enhanced GPU Chip Set.

CHIP SET ENHANCEMENTS

Specific changes to the GPU Chip Set will include moving to 2 or 3 Micron feature-size devices. Smaller feature size will allow repartitioning of devices into larger functions, allowing smaller

chip counts for a specific implementation. This will not only reduce the size of a MIL-STD-1750A CPU to about 3.5 inches on a side, but will also significantly reduce cost and increase reliability. A bonus of the smaller feature size is 5VDC operation at speed. This will obviate extra circuits and the associated delays for level conversion to 5V buses and thus provide both size and speed advantages.

Another specific change for the GPU Chip Set will be in the method of sourcing. It is desirable to have the option of implementing these devices in either CMOS/SOS or Bulk CMOS Rad-Hard Technologies, as well as Low-Cost Bulk CMOS, depending on the application. It is also desirable to have multiple sources available for the same set of Government-owned device designs with the ability to procure them competitively. A set of common design rules has been defined as a "common denominator" of the design rules of 10 suppliers of CMOS/SOS and Bulk CMOS. The unique features (alignment marks, etc.) of each supplier's design rules is filed separately and used in making specific program generation tapes of designs to which that contractor can directly bid for fabrication. We expect to have 5 to 10 sources available for the Enhanced GPU Chip Set in both CMOS/SOS and Bulk CMOS technologies. Major advantages will include lower costs through competition and greatly increased circuit availability, as well as protection from source failure.

Examples of the most significant changes in the Enhanced GPU Chip Set are the double GPU (ALU circuit) and the 1000-gate government-owned Gate Universal Array (GUA).

The double GPU will be a 16-Bit wide ALU slice compared to the current 8-Bit GPU. Like the current GPU, it will have the ability to concatenate directly with other GPU's without

support circuitry. The new GPU will feature BYTE recognition and have ability to operate in either upper or lower BYTE mode. It will have an 8-bit-wide shifter compared to the current 2-bit-shifter. This will greatly enhance multiply and floating point operation. Two bidirectional data ports will aid execution side pipelining and limit critical pin-out requirements on the Bus Interface Unit (BIU). A temporary register will be moved from the BIU to the GPU in order to minimize off-chip execution functions. Decode functions will aid prealignment for normalize or operations using a matrix multiplier. A position location register will be considered to keep track of boundary conditions for concatenation and a configuration register will limit extra pinouts by internally keeping track of word, upper BYTE and lower BYTE modes in the ALU. The 16-Bit GPU with the above features incorporated can be implemented and packaged in a 64-Pin HCC.

The new 1000-gate GUA is intended to address current lack of ability to procure random logic devices in GUA form from multiple sources. Current GUA's are proprietary company-developed circuit devices. Most are not organized for high gate utilization efficiency. The current software aids available to support these devices are also proprietary and, for the most part, inadequate. They range from none to verification checks. The more sophisticated the software is, the less adaptable it is to various GUA types.

The new GUA will be a government-owned design. It will be designed using the new common design rules and thus be multi-sourceable in both CMOS/SOS and Bulk CMOS. It will be sized and organized to achieve both a reasonable degree of complexity and a high percentage utilization of the available gates. It will be designed for compatibility with an automatic software router.

New circuit types included in GPU Chip Set Enhancement plans include a Memory Management Unit (MMU) device for distributed network system application, capable of global/local address transition. It will have protect checks built in. It will have 4 sets of 64-page registers and be control-concatenable to add additional page register sets. Most importantly, it will be MIL-STD-1750A compatible.

A new high-density output-latched ROM device for micromemory will be developed. This memory will be organized as 1K X 32. It will reduce the current MIL-STD-1750A micromemory device count from 23 to 4 devices and increase micromemory capacity from 768 to 1024 words.

A new Bus Interface Circuit (BIC) will replace the current BIU in the 1 MOPS distributed system applications. Some functions will be moved and some pipelining functions added in order to further separate CPU execution and operand derivation. The new circuit will contain memory protocol, a program counter, instruction queuing, partial instruction decode, prefetch, and register preference for operand address.

An Error Detection and Correction(EDAC) device will be implemented using a proven, existing EDAC algorithm. Depending on chip space and pin-out, this function may be a part of the new BIC device discussed above.

In order to connect elements of the distributed network system to its global buses, a new Bus User Connect (BUC) device will be implemented. This device will format, send, receive and execute parallel-to-serial and serial-to-parallel buffer functions.

SUMMARY

The complete program of GPU Chip Set Enhancement planned will produce parts for a 1 MOPS MIL-STD-1750A CPU substrate.

The effect of the enhancements on the Chip Set device complement for the MIL-STD-1750A is summarized in Table 2. Coupled with multiple sourcing of the Chip Set, this will produce dramatic cost improvements and greatly increased availability. The preplanned distributed system architecture enhancement will also expand the application of MIL-STD-1750A significantly.

TABLE 2
MIL-STD-1750A CPU CHIP SET COMPLEMENT

<u>PRESENT CHIP SET</u>		<u>ENHANCED CHIP SET</u>	
<u>TYPE</u>	<u>QTY</u>	<u>TYPE</u>	<u>QTY</u>
8-Bit GPU	6	16-Bit GPU	3
MCU	2	MCU	2
RSU (1000)	1	RSU (1000)	1
ASU (300)	3	ASU (1000)	1
ICU (452)	4	ICU (1000)	2
ILT (300)	3		
BIU (452)	4	BIU (1000)	2
ROM (4096)	26	ROM (32K)	5
	<hr/>		<hr/>
	49		16

AWO-A142 776

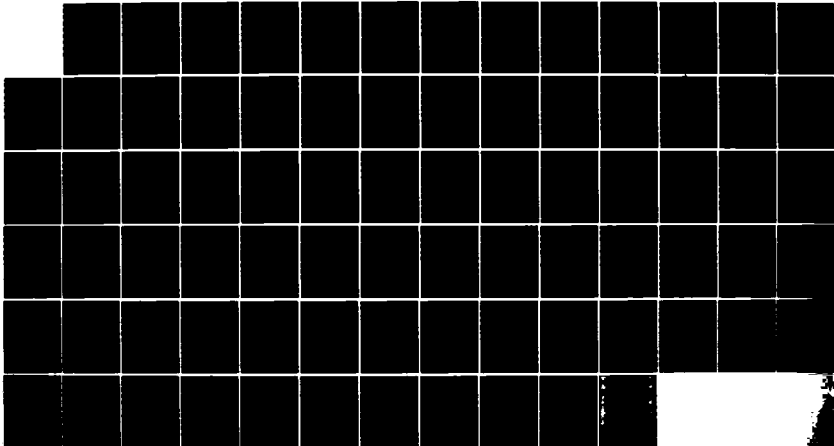
PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND. (U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O.
C A PORUBCANSKY NOV 82

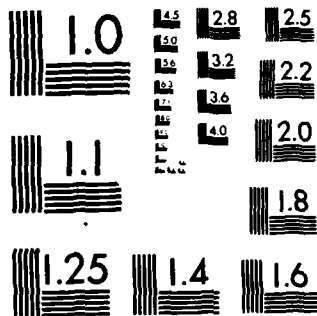
6/6

UNCLASSIFIED

F/G 9/3

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Mr. Denison is the Director of Business Development for Digital Systems, Military Avionics and Telecommunications Systems at TRACOR Aerospace in Austin, Texas. In 1978, Mr. Denison joined TRACOR as Director of Marketing for a new line of intelligent teletypewriter terminals. Since then he has been developing new product markets for TRACOR. Prior to joining TRACOR, Mr. Denison was involved in computers and computer-based systems for military aerospace applications with Sperry Univac Defense Systems Division for 12 years. For Sperry Univac, he served in various capacities, including Senior Systems Design Engineer, Northeast Regional Marketing, Avionics Sales Manager, and Director of Avionics and Special Programs Marketing. Mr. Denison also held various engineering positions with Autonetics Division of North American Aviation and Westinghouse Baltimore Divisions. Mr. Denison holds a B.S.E.E. degree from the University of Oklahoma.

Mr. Fosdick is Principal Scientist for the Computer Products Activity within TRACOR Aerospace in Austin, Texas. He is the Principal Architect of the GPU device, the Emulating Microcontroller (MCU) device and the microarchitecture that has been utilized to implement the products discussed in this paper. Mr. Fosdick has been with TRACOR since 1972. Prior to joining TRACOR, Mr. Fosdick served as a Design Project Leader, Program Manager and a Staff Engineer with Texas Instruments Integrated Circuit and Computer Development Operations in Dallas, Houston and Austin, Texas. He also has held Key Engineering positions with National Cash Register in Dayton, Ohio and the Martin Company, Space Systems Division in Baltimore, Maryland. Mr. Fosdick holds a B.S.E.E. Degree from the University of Illinois, where he worked on the ILLIAC and IBM 650 Systems. He also has an M.S.E.E. in Computer Sciences from Southern Methodist University.

MIL-STD-1750A AS A SPACEBORNE INSTRUCTION SET ARCHITECTURE

by

Robert N. Constant,
Richard C. Fleming,
Edwin M. Garcia,
Marvin Lubofsky
and
Donald R. O'Bell

THE AEROSPACE CORPORATION
P.O. Box 92957, L.A., CA 90009
(213) 648-5756

AD-P003 554

ABSTRACT

A study was undertaken to examine the issues involved in establishing instruction set architecture (ISA) standards for computers used in embedded spaceborne applications on Air Force Space Division projects. The specific areas addressed were (a) Space Division requirements and ISA tradeoffs, (b) comparisons of military standard and commercial spaceborne ISAs, (c) an LSI implementation study and (d) an LSI cost study. The bottom line of the study is that the MIL-STD-1750A ISA addresses the onboard processing needs of all present and near term Space Division projects as well as many of those on far term projects. In addition, the development of space qualified 1750A computers should be a low risk project with relatively modest costs once the efforts currently sponsored by other agencies are successfully completed.

INTRODUCTION

During FY81 a study was conducted under Project Element 64740F in order to provide Space Division (SD) with the information required to establish the degree of instruction set architecture (ISA) standardization desirable and possible within SD for onboard processing. This paper is an overview of portions of the final report of that study.

The project was divided into four major interrelated activities, the first of which was the task of establishing requirements and benchmarks. After examining a number of existing and near term Space Division satellite projects, onboard processing was divided into the three broad functional areas of housekeeping, station keeping and mission data processing. Each of these areas in turn was divided into numerous subfunctions and several SD projects were examined in detail to produce data on timing, sizing, loading and instruction mixes for these subfunctions. Based on comparisons of the findings with the properties of the MIL-STD-1750A ISA and existing implementations, it was determined that there are no inherent limitations which would preclude the use of 1750A. The details of the results of this portion of the study are not essential to an understanding of the ISA comparisons and costing study and are therefore not covered in depth in the present paper.

The second major task, which forms the bulk of this paper, was to compare the ISA's of a cross section of existing spaceborne computers with the ISA of MIL-STD-1750A. A set of five current commercial space qualified computers ranging from the smallest to the largest and most powerful computers in use was selected for the comparisons. The basic result of the comparisons is that the functionality of the MIL-STD-1750A ISA exceeds that of all of the commercial computers studied, with minor exceptions. The only significant areas where some of the commercial ISA's surpassed the 1750A was the inclusion of specialized application dependent instructions. As will be seen, the flexibility and extendability of the 1750A ISA will allow it to overcome these handicaps.

The third major portion of the study was an investigation into the feasibility of building low power, low weight, hardened LSI implementations of MIL-STD-1750A computers. This task involved considerable discussions and interactions with other government agencies and private companies conducting such work. In addition, detailed algorithms for the implementation of several MIL-STD-1750A instructions were developed. These experiments revealed that it is indeed feasible to implement a suitable VLSI version of 1750A. Historically, spacecraft computers have generally been developed as derivatives of avionics computers. The data gathered in the course of the LSI study indicates that the level of developments for 1750A is such that the normal evolution from avionics to spacecraft computers is indeed possible. Many of the efforts surveyed are the subject of other presentations in the present conference and thus are not discussed here.

The fourth task in the project was a cost study. The cost study was divided into two major efforts. The first was to develop a comprehensive cost model for the implementation of ISA standards. The resulting model includes 1750A R&D costs, investment costs and O&M costs. The second part of the cost study was to use the model to estimate the relative costs of the various options for the imposition of ISA standards. Not surprisingly, it was found that for the near term, it would be less expensive to continue using the existing commercial ISA's. Next, it was established that modifying existing spaceborne computers to conform to 1750A would cost less initially than building new 1750A computers. The cost differential largely depended on the ISA of the existing machine as well as the degree of performance

improvement to be achieved during the upgrade. Most importantly, it was established that building new 1750A computers should cost no more initially than building to some nonstandard ISA whether off-the-shelf components or new VLSI devices were used. In addition, the use of a standardized ISA would result in a lower cost per hardware unit and eliminate support software costs and thus would lower overall life cycle costs.

ISA COMPARISONS

The ISA comparison task was to compare an existing military standard instruction set architecture with those of a cross section of existing space computers. The computers chosen for the study were the Litton LC-4516 which has been used in a number of satellite projects, the Delco Magic 432S used in an upper stage, the GE Alpha-16 which serves as the flight computer on a communications satellite, the RCA SCP-234 used on a weather satellite and the CDC 469, a bare-bones spaceborne machine with a minimal instruction set used as a spacecraft controller. A summary of the characteristics of these five computers is given in Figure 1.

Before performing the comparisons, it was first necessary to choose from the two current military standard ISA's. The first is the 16-bit instruction set architecture embodied in MIL-STD-1750A. The 1750A ISA was originally developed by the Air Force as an ISA for avionics applications. Since most presently existing space qualified computers have evolved from avionics computers, the 1750A ISA is a natural candidate to consider for a possible Spaceborne ISA. The second available military ISA is the "Nebula" ISA which is defined by MIL-STD-1862A. Nebula is a very advanced 32-bit ISA which was designed to support Ada programming in embedded applications. Unlike 1750A which is strictly an Air Force effort, Nebula originated as an Army project (it is the ISA of the Army's Military Computer Family) and is now a joint Army-Air Force ISA.

The ISA of MIL-STD-1750A was deemed to be closer to those presently implemented in space qualified computers. In particular, the 1750A is a 16-bit machine, as are most of the commercial machines presently in use, while Nebula is a 32-bit machine. Thus, it was decided to compare the ISA's of the selected commercial computers to the 1750A ISA, while saving comparisons with the Nebula ISA for later.

Before giving the results of the comparisons, several notes are in order. The first is that the five selected commercial computers are actual machines which exist and are being used in current programs. However, rather than being interested in the actual hardware implementation, the current section is concerned with the ISA only. In contrast, the 1705A ISA, for the purpose of this study, is just that, an instruction set architecture, not a physical implementation of a machine. Although MIL-STD-1750A is being implemented in a variety of machines, none of them are presently space qualified. Thus, it would make no sense to compare the execution rates for execution rates for a particular instruction. On the other hand, the presence or absence of a particular instruction or class of instructions in an ISA can be indicative of the efficiency of the ISA. In addition, the

FLIGHT COMPUTER CHARACTERISTICS SUMMARY

Identification	Delco Magic 362 S	RCA 234	GE ACE (Alpha 16)	Litton 4516 E	CDC 409
Weight (pounds)	52	11	10	13	10
POWER (watts)	200	10	25	80	20
Word Length	16/32	16/32	16	16/32	16/32
Memory size	64K (32K x 39 ECC)	64K (16K used)	1K CMOS RAM 8K PPOM	64K	16K
Memory Technology	CMOS	CMOS	1K CMOS 8K PPOM	CMOS	Plated Wire or CMOS
CPU Technology	Bipolar	CMOS	Bipolar	Bipolar	PMOS
Executing Speed (sec)					
Add	1	4.8	5.5	2.5	4
Multiply	2.8	60	x20.5	21	10.4
Floating Point	Yes	No	No	Yes	No
Time Frame	1975	1970	1975	1978	1975

Figure 1

efficiency can be impacted by the number and types of addressing modes included in an ISA. Figure 2 summarizes the areas where major differences were encountered in the various ISAs.

The first ISA compared with the 1750A ISA was that of the Litton LC-4516, a space qualified computer which has been used successfully in three space projects. There are six instructions in the LC-4516 repertoire which are not included in the 1750A ISA. All of the missing instructions can be performed by short sequences (one, two or three) of 1750A ISA instructions. Examples are the LC-4516 "Compare Register (and Jump)" instruction which can be implemented on a 1750A ISA machine as a "compare" instruction followed by a "jump" instruction or, more trivially, the LC-4516 "increment register" instruction which is equivalent to adding a code stream literal with value "one" to a register on a 1750A machine. In general, other than examples falling into the above class, there are no instructions in the LC-4516 ISA which do not have equivalents in the MIL-STD-1750A ISA. Furthermore, the LC-4516 instructions not found in MIL-STD-1750A tend to be very low frequency instructions thus minimizing the impact of requiring a sequence of 1750A instructions to perform their operations.

On the other hand, there are at least two dozen instructions in the MIL-STD-1750A ISA which are not found in the LC-4516 ISA. Among the missing instructions are an extensive set of bit and byte manipulation instructions as well as implied register stack operations. In addition, the 1750A ISA contains several sophisticated "move" instructions not included in the LC-4516. Finally, the 1750A has vectored I/O instructions, a feature not found in the LC-4516D. The bottom line is that the MIL-STD-1750A ISA can not only replace the LC-4516D ISA in spaceborne applications, but also includes an extensive body of additional powerful instructions which could easily find use in onboard applications.

The Magic 362 computer produced by the Delco Electronics Division of General Motors is used as the flight computer in the upper stage of the space shuttle. While the ISA of the Magic 362 has several specialized and/or exotic instructions lacking in the 1750A instruction set, the study revealed that, in general, the everyday, more common instructions forming the heart of an ISA were more powerful and had more useful addressing modes in the MIL-STD-1750A ISA.

Among the specialized Magic 362 instructions not included in MIL-STD-1750A are a set of macro-instructions designed to facilitate the coding of flight equations. These include the floating point instructions "square root," "sum of squares," "polynomial expansion," normal and odd powers and a limiting function. It should be noted however, that if such instructions are, in fact, required to achieve performance goals, MIL-STD-1750A does include provisions for built-in functions. In effect, the 1750A built-in function mechanism allows the implementor to define one or more of the presently undefined opcodes. This mechanism was included in the standard in order to allow the freedom to implement application dependent instructions without burdening the entire user community with them.

Results of ISA comparisons

Deficiency ISA	Registers			Addressing			Arithmetic					Other Operators					BIF
	Small # of general purpose registers	Restricted register usage	No multi-register operations	Lacking in addressing modes	Limited address space	No stack operations	No floating point	No extended/double precision	Lacking some specialized F.P. instructions	No 64-bit addition & subtraction	No double precision fixed point multiply	Missing 2-step operations	Minimal bit/byte operations	Minimal data transfer operations	Limited compare/branch operations	No vectored I/O	
Litton LC4516			X		X	X	X	X	X	X	X	X	X	X	X	X	
Delco Magic 362S		X	X	X	X							X	X	X	X	X	
GE Alpha 16	X	X			X	X	X	X	X	X	X	X	X	X	X	X	
CDC 469		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
RCA SCP 234	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	
MIL-STD-1750A								X	X	X	X					X	

Figure 2

With the exception of the above mentioned macro-instructions, the 1750A ISA was found to be superior to that of the Magic 362 in several areas. Foremost among these was the addressing modes. The 1750A ISA allows extensive use of code stream literals and more powerful branching capabilities than the Magic 362. Although both ISA's have sixteen general registers, the Magic 362 requires them to be addressed in specific pairs and triples, while the 1750A may address the registers arbitrarily. These features result in more compact code which presumably will execute faster. In addition, the 1750A has numerous instructions which have no equivalent in the Magic 362. These include a wide selection of load/store, logical, comparison, branch, arithmetic and floating point instructions. In addition, 1750A has a full set of 48-bit extended precision floating point operators while the Magic 362 has 64-bit extended precision with only addition and subtraction implemented. In summary, while the Magic 362 has several powerful instructions not found in MIL-STD-1750A, 1750A has more powerful addressing modes and many additional instructions. To date, no flight functions have been coded in both Magic and 1750A assembly languages so a quantitative comparison cannot be made. However, it is quite clear that, with the possible inclusion of some built-in functions, the MIL-STD-1750A ISA could replace the Magic 362 ISA in spaceborne applications.

The Alpha-16 is a space qualified computer produced by GE and serves as the flight computer on a communications satellite. The Alpha-16 was designed to emulate the DEC LSI-11 instruction set architecture. An advantage of this ISA over that of MIL-STD-1750A is the inclusion of numerous extra addressing modes. Of these, the most commonly used ones are register, absolute and immediate, all of which are in 1750A, thus minimizing the advantages. In addition, the 1750A ISA has sixteen general registers rather than eight and, additionally, does not have even-odd usage restrictions. The LSI-11 ISA is rather unconventional, making an instruction-by-instruction comparison difficult. However, it was clear that 1750A has numerous instructions in all categories that are not included in the LSI-11 ISA. Since these two ISA's were so difficult to compare, it was decided to recode portions of a real flight program in 1750A assembler language. The result was that the 1750A version not only contained fewer instructions but also required 20% less storage. The conclusion was that the MIL-STD-1750A ISA could easily replace that of the LSI-11 in the DSCS-III flight computer.

The ISA of the RCA SCP-234 computer, which is used in a weather satellite, was compared with the MIL-STD-1750A ISA. The results were very similar to those obtained when the LC-4516 ISA was compared with the 1750A (mentioned previously). In particular, there are six instructions in the SCP-234 ISA which are not in MIL-STD-1750A and 58 instructions MIL-STD-1750A which are not implemented on the SCP-234. As was the case with the Litton LC-4516, most of the SCP-234 instructions not found in 1750A tend to be compound instructions such as "Multiply Cumulative" or "Add to Memory." Again, they can all be easily performed by a 1750A machine as a short sequence of 1750A instructions without any great loss of performance. The 58 MIL-STD-1750A instructions not found on the SCP-234 consisted mainly of the same types of bit, byte and stack manipulation and floating point instructions which were found to be missing from the LC-4516. Again, it was

found that the MIL-STD-1750A ISA could replace that of the SCP-234 in the DMSP project.

The final ISA examined during the study was that of the CDC 469, a general purpose, 16-bit, single address machine used in several applications as a spacecraft controller. The CDC 469 has a very limited number of addressing modes. In addition, although it has sixteen registers, their usage is, in general, severely restricted and depends in part on the interrupt level. The CDC 469 has a very brief set of instructions, most of which correspond directly to MIL-STD-1750A instructions. It is quite clear that the MIL-STD-1750A ISA, or even a small subset of it, could easily replace that of the CDC 469 in spaceborne applications.

The primary conclusion reached from the ISA comparisons is that the MIL-STD-1750A ISA could replace that of existing spaceborne computers now in use for near term missions. In fact, the richness of the 1750A exceeds that of most present spaceborne ISA's, particularly in the area of floating point which is not presently implemented on many spaceborne computers. On the other hand, 1750A is lacking some of the more powerful and specialized macro-instructions found on some of the larger spaceborne computers. However, as mentioned earlier, this does not pose a serious problem since MIL-STD-1750A does allow for the inclusion of application dependent built-in functions. Whether or not such functions should be included in a proposed standardized spaceborne ISA or left as options is not clear without further study. Actual utilization, of course, depends on having a 1750A implementation meeting Space Division's throughput, power, weight, size and radiation hardness requirements.

The overall conclusion is that there is sufficient commonality between the ISA's of existing near term spaceborne processors currently performing housekeeping and station keeping functions to warrant the use of a standardized instruction set. This conclusion is independent of the question of whether or not MIL-STD-1750A or some subset, superset or modification of it will serve as the standard. On the other hand, for mission data processing requirements of the future, the current 16-bit architectures could prove a severe limitation. There are indications, however, that the VHSIC versions of 1750A offer the potential to satisfy a number of mission data processing applications. When a more powerful conventional ISA is required, the Nebula ISA of MIL-STD-1862A should be examined in this regard. Since it is a thirty-two bit ISA there are few of the limitations inherent in sixteen-bit machines. In addition, Nebula was specially designed to run programs written in high level languages, particularly Ada, and includes hardware features for multi-level secure operating systems. On the other hand, Nebula is unlikely to be implemented in a suitable configuration for several years.

Thus, it appears that MIL-STD-1750A, perhaps with suitable alterations, could serve as a spaceborne ISA standard for near term projects as well as for the housekeeping and station keeping processing for far term projects. At the same time, the Nebula ISA, again with possible alterations, could serve as a standardized spaceborne ISA for those projects where a larger, more powerful ISA is required.

COST ESTIMATES

The cost study task was to explore the cost of implementing an ISA for space applications. As was the case in the part of the study concerned with ISA comparisons, the MIL-STD-1750A instruction set architecture was used as the baseline. As part of this study, information was obtained from vendors experienced either in building spaceborne computers or in implementing MIL-STD-1750A computers. Foremost among this group of vendors were Litton and Delco.

The initial part of the cost study was to develop a comprehensive cost model and cost estimating relationships related to the 1750A's research and development costs, investment costs and operations and maintenance costs. Since the immediate interest in the present study lies with the R&D costs of a 1750A computer, exploration of these costs was given priority. Basic cost models of the two other major cost elements were explored but not evaluated in great detail.

The second part of the cost study was to obtain estimates of the costs associated with pursuing the various ISA options. Since earlier parts of the study had indicated that the MIL-STD-1750A ISA was a viable, and in fact the leading, candidate for use as a spaceborne ISA standard, particular attention was paid to costing those options which would result in a space qualified 1750A computer meeting the special SD requirements. The options ranged from relatively modest upgrades of existing computers to full-scale development of a new 1750A computer using new VLSI developments. To provide a baseline, or set of constraints, the costs of obtaining an existing space qualified computer and of developing a new non-1750A ISA computer were also explored. This task involved considerable interaction with vendors to obtain costing information as well as with Air Force program offices (particularly F-16) to substantiate and validate the findings. In addition, extensive use was made of the costing models discussed above. The results of the cost study, summarized below, are highlighted in Figure 3.

In order to provide a frame of reference, prices were obtained for several of the existing computers discussed in the section on ISA comparisons. Typical of these is the Delco Magic 362S. The M362S is a non-VLSI machine of high weight (11 pounds) and power consumption (65 watts) which does not conform to MIL-STD-1750A. A space qualified version of the machine can be obtained for \$150K. This figure is a purchase price for an off-the-shelf machine and does not include any development costs. To provide a contrast, an avionics version of the same machine costs \$80K (in large quantities) and is heavier and consumes more power while having higher throughput and a larger memory. On the other hand, it has a lower MTBF (300 vs. 36,400 hours) and a lower degree of radiation hardness.

It was determined that the least expensive method of developing a space qualified MIL-STD-1750A ISA computer would be to modify an existing machine. The ISA comparison task revealed that the ISAs of the Litton LC4516E and the Delco Magic 362S were closer to that of 1750A than the others. It was found that the cost of converting either of these two machines to implement 1750A would fall in the range of \$1 - 5 million. (In

Cost Estimates for Developing a Space
Qualified MIL-STD-17501A ISA Computer

- 1) By upgrading existing computers to conform to MIL-STD-1750A
(Contractor supplied/derived estimates)

Litton LC-4516E	\$1 - 5M
Delco M362S	\$1 - 5M
Delco M372 (LANTIRN/F -16)	\$3M

- 2) By developing a new 1750A computer

Using off-the-shelf components	\$7.4
With new VLSI development	\$12.4

- 3) Cost of developing a new non-1750A computer

Using off-the-shelf components	\$10M
With new VLSI development	\$15M

NOTE: These costs are estimates obtained from the costing model or from informal discussions with vendors. No contractual obligation is implied.

Figure 3

contrast to the purchase price of the M362S mentioned above, this is the development cost.) The low end of this range corresponds to modifying the machine so as to implement MIL-STD-1750A instead of the native ISA. Such an implementation would result essentially only in those performance enhancements inherent in converting to the 1750A ISA. To substantially increase the throughput of either machine would require a much more extensive redesign and would thus fall into the higher end of the range. Similarly, implementations reducing the size, weight and power requirements of these non-VLSI computers would result in development costs near \$5M.

A similar, but possibly less expensive alternative is the redesign of the Delco M372. The Delco M372 is an avionics computer implementation of MIL-STD-1750A presently in use on the LANTIRN and F-16 projects. The M372 can be modified to conform to Space Division's needs for an estimated \$3 million. The resulting computer would be comparable to that obtained via the more expensive modification of the LC-4516E or M362S mentioned above.

In contrast, it was determined that a new space qualified, militarized MIL-STD-1750A computer conforming to SD's requirements can be developed for \$7.4 million using off-the-shelf components. To build such a new 1750A machine using new VLSI developments (seven devices would be required) would cost an additional \$5 million. The total of \$12.4 million would result in a machine with higher throughput than those in the \$5 million range. In addition, the new machine would have substantially lower power, size and weight requirements than the lower priced alternatives.

The final cost estimates obtained were for developing an implementation of a non-1750A ISA. It was determined that the hardware development costs would be the same as for MIL-STD-1750A: \$7.4 million for an implementation using off-the-shelf components and \$12.4 million with new VLSI developments. However, there is an immediate extra cost associated with this option: The cost of support software. There are presently several sets of MIL-STD-1750A support software available or under development. These can be obtained from the government and do not add to development costs. For a nonstandard ISA such software would have to be developed. It is conservatively estimated that a compiler and assembler alone would cost at least \$2 million with additional costs for acceptance test, debug and interpretive simulator software. Thus, in effect, the real cost of developing a suitable non-1750A computer is significantly greater than developing a 1750A implementation.

As seen above, the use of the MIL-STD-1750A ISA would result in a significant upfront savings by avoiding the cost of compilers, assemblers and other software development tools. However, this is only a one time savings. Of far greater potential impact is the savings inherent in the ability to reuse software modules during upgrades and to share compatible software modules between projects. In addition, the use of a standard ISA and a standard set of software development tools on a wide spectrum of projects would help to alleviate the problems with personnel resources since programmers would be portable from project to project with little or no retraining.

If an existing onboard computer is upgraded into a 1750A machine, then clearly the software will have to be rewritten. On the other hand, even if the same ISA is retained, but the computer is upgraded to today's hardware technology, a considerable portion, if not all, of the existing software will also have to be rewritten. This is due to the fact that a different implementation will result in different timing dependencies and the real time software will have to be changed to accommodate it. In addition, it is nearly unheard of to have a hardware upgrade without concurrent modifications of the mission or other parts of the satellite hardware. Such changes would clearly result in the modification of existing software requirements or the inclusion of new ones. Finally, if new instructions are added to the ISA of the machine, then extensive modifications of the code must be made to take advantage of them. Thus, it is not clear that retaining or slightly extending an existing ISA during a hardware upgrade will always result in lower software recoding costs than would converting to MIL-STD-1750A. Thus in an upgrade where it appears that an existing machine can be salvaged, it is recommended that a study be made to compare the costs with those of converting both the machine and the software to 1750A.

In conclusion, the numbers speak for themselves. If an existing spaceborne computer meets the requirements of a project then it should be used, but only after its costs have been compared with those of using MIL-STD-1750A computers. On the other hand, in those projects where an off-the-shelf computer does not meet the requirements, the development and use of MIL-STD-1750A computers can and will result in significantly lower costs.

CONCLUSION

The overall conclusion resulting from the present study is that it is not only possible but is, in fact, advantageous to establish standardized instruction set architectures for use in onboard housekeeping and station keeping processing. Additionally, it has been determined that there is sufficient commonality among the ISA requirements of the various SD projects that it is possible to formulate a standard ISA encompassing all of their features. Extensive comparisons of standard ISA's and existing spaceborne ISA's have shown that the ISA of MIL-STD-1750A is a viable candidate for a Space Division standard. The 1750A ISA will not only satisfy present and near term SD needs but will address most long term requirements for housekeeping and station keeping processing. As more and more mission data processing is done onboard the spacecraft VHSIC versions of 1750A with signal processing capabilities could prove useful. It is recommended that the Nebula ISA of MIL-STD-1862A be considered when a larger, more powerful ISA becomes necessary. There are two additional reasons to consider these existing standard ISA's rather than formulating an entirely new SD ISA. The first is the obvious benefit derived from reduced development costs, both in hardware and in support software. The second is that the development of a mature ISA (just the ISA, not the hardware) is a long lead time task. For example, it took over two years to define and refine the Nebula ISA. In addition, using or even modifying 1750A and 1862A would not only reduce the

risk involved but would provide a previously existing network of user's groups, control boards, etc. to aid in maintenance and growth of the standard.

After it was found that it was possible to define a standardized ISA and MIL-STD-1750A was found to be a candidate, questions of implementation and costs still remained. The study has documented that it is feasible to build a space qualified MIL-STD-1750A computer and that such an undertaking would not be a high risk project. Finally, it was established that the cost of building such standardized ISA space computers is reasonable. In fact, it was determined that, by capitalizing on the progress made by other agencies on 1750A implementations and support software, it would cost considerably less to develop a new VLSI 1750A spacecraft computer than it would to build one using a nonstandard ISA. In addition, it has been shown that non-VLSI space qualified 1750A computers can be obtained for even lower costs by upgrading existing spacecraft computers to conform to MIL-STD-1750A.

MIL-STD-1750A Verification Testing.

Luis E. Velez, ASD/ENASF (SEAFAC)
Wright-Patterson Air Force Base, Ohio.

ABSTRACT

This paper describes reasons for and importance of verifying that a computer correctly implements MIL-STD-1750A architectural specifications. It includes a brief summary of verification history. It describes present methodology for verification testing including test procedures, results analysis, and test reporting. Finally, a future approach for verification testing is discussed.

INTRODUCTION

The intention of this paper is to give a brief overview of what the U. S. Air Force is doing to ensure architectural compliance to MIL-STD-1750 Instruction Set Architecture (ISA) of different computers from various contractors.

BACKGROUND

MIL-STD-1750A defines the Air Force standard 16-bit computer ISA for airborne computers from a machine language programmer's point of view. It does not define specific implementations of a computer such as speed or size.

Use of this standard will allow the Air Force to use and re-use available support software such as compilers, linkers, loaders, assemblers, etc., allowing reduction in software costs. Reduction in logistic management, number of support software packages, and software development cycle will also be a factor in cost reduction.

Knowing in advance which ISA is to be used for a computer contract, allows reduction in development time. This means we can begin software coding very early in the system life

cycle.

Competition would be encouraged by requiring the use of the standard. Programs like LANTIRN HUD, F-16 FCC, F-16 microprocessor, F-111, C-X, F-5G, HH-60D, MATE, and B-1B show this competition when contracts awards were made to five different firms.

Developments in technology allow users to have smaller, faster, lower power consumption, and higher reliability computers. An example of this is the development of chip sets by manufactures such as Delco, Mikros, McDonnell Douglas, Honeywell, Tracor and Air Force sponsored development of a single chip by Fairchild. Higher performance chips are under development by Westinghouse and Texas Instruments using VHSIC technology. This clearly shows potential growth of technology along with the architecture.

When the Air Force decided to adopt ISA standardization, a method of verifying that a vendor's computer conforms to the standard architectural specification was needed. The Systems Engineering Avionics Facilities Branch (SEAFAC), ASD/ENASF, Wright-Patterson Air Force Base, Ohio has received the responsibility for compliance certification of MIL-STD-1750 computers.

VERIFICATION HISTORY

The Architectural Test Program (ATP) was originally developed by TRW (under contract to the Avionics Laboratories) as an acceptance test for the AN/AYK-15A (MIL-STD-1750) processors developed by Sperry-Univac and Westinghouse. The ATP was subsequently chosen by SEAFAC for an interim verification tool, and was updated by TRW to MIL-STD-1750A. Since delivery by TRW in March 1981, SEAFAC has added a number of features to the ATP, including a stand-alone interrupt test and changes which reflect MIL-STD-1750A Notice 1. The ATP has been distributed to 28 contractors, four Air Force offices, one university, and three foreign agencies in either 1750A or 1750A Notice 1 configuration and has been successfully used in verification testing of 10 processors. It is a relatively mature tool, and will continue to be used until the Verification Software (VSW) comes on line next summer.

CURRENT APPROACH

Currently, testing of MIL-STD-1750A processors is done using the SEAFAC 1750A ATP. A period of one to two weeks is allowed for test completion. Testing can be done either at SEAFAC or contractor facilities depending on the contractor needs.

Various requirements must be met to make verification testing possible. On the contractor side, a request for the ATP must be received by ASD/AXT. AXT will send a Terms and Conditions Agreement Contract to be signed by the contractor agreeing not to sell the ATP back to the Air Force. Upon receipt of the signed contract by AXT, SEAFAC will send a copy of the most current version of the ATP along with its documentation. The contractor will then use the ATP to test the processor at their facilities prior to final verification testing by SEAFAC. A minimum of 32K continuous random access memory from address 0 is required, although 64K is preferred. A single RS-232C I/O channel is required for both downloading the ATP to the processor or unit under test (UUT) and interactive execution of the ATP. This interface must be controllable by XIO commands, use a standard 25-pin RS-232C connector, and have selectable baud rate (9600, 4800, 2400, 1200, 600, 300). Data format will be 1 start bit, 2 stop bits, and no parity. The contractor will also supply a maintenance (user) console with a mass storage device (tape, disk), I/O connectors and cables. The contractor must supply, prior to verification testing, two routines called GETCHR and PUTCHR used in the ATP to perform input and output through the RS-232C interface. They are called by SJS R15, GETCHR and SJS R15, PUTCHR for GETCHR and PUTCHR respectively. Upon return from GETCHR, one ASCII character is expected in the least significant eight bits of general register RO. If no input is available, GETCHR must wait until input is available before returning. In PUTCHR, one ASCII character is supplied in the least significant eight bits of RO. If output is not immediately possible, PUTCHR must wait and perform the output before returning. These routines must be linked in with any ATP load or stand-alone module that is to be executed. For on-site testing, a hard copy terminal with a keyboard is required for recording test results. Also, a TV monitor or TV set for use with an APPLE computer is needed.

SEAFAC will supply all ATP load modules which include executive-controlled and stand-alone tests, and an ATP loader which interacts with a VAX or APPLE driver to allow load or dump of UUT memory. SEAFAC personnel will brief the contractors on test

procedures, recording and analysis of results, and test reporting. Communication is established between UUT and the VAX or APPLE using the loader program. Then each ATP load module is downloaded, executed, and results are recorded.

Recorded results are analysed and a test report is written and forwarded within two weeks of test completion to the contractor tested. The test report contains date of test, ATP version used, processor ID, a list of testable options present in the processor such as expanded memory addressing, memory block protect, etc., results of all executive controlled loads and stand-alone tests, and SEAFAC interpretations of the results.

IBM AND SPERRY UNIVAC STUDIES

An IBM study investigated eight different verification methods. These verification approaches were grouped into four generic types: Functional, Random, Lockstep, and Analytical.

The Functional types consist of programs which verify the architecture by executing a number of test cases which test the architecture at a functional level.

In the Random Instruction verification approach, a sequence of randomly generated instructions is executed and verified that the proper results are generated by comparing them to results from a simulator.

The Analytical approach consists of manual generation of an architectural specification in a special high order language called Language for Symbolic Simulation (LSS). A manually generated description of each machine in the LSS format is also required. A set of defined relationships is automatically generated through a computer program. The program then compares the specification of the architecture to the design implementation using proof tree analysis.

In the Lockstep test approach, a functional type test program is execute in parallel on two computers. One is a certified MIL-STD-1750A computer and the other is the UUT. Data describing the state of the UUT is saved and compared to the state of the certified computer.

The recommendation of this study to the Air Force was a two phase approach to certification. Phase I consists of a Functional verification approach based on modification to the AFAL (DAIS) ATP to be run on the MIL-STD-1750A computer being tested. In Phase II, a Random verification approach would be used to generate large numbers of test cases. A test case contains a sequence of 32 randomly generated instructions which is first run on a MIL-STD-1750A simulator and then run on the MIL-STD-1750 computer being tested. The results from the computer being tested are compared with the simulator results.

The Sperry Univac study investigated four different verification methods. These methods were evaluated relative to the design goals which consisted of complete testing of each functional entity, testing of interrupts, I/O, and processing, ease of system use, test data easy to expand, validated certification process, certification process adaptable to specific options and/or changes in MIL-STD-1750, and a certification facility able to provide a means of generating, testing, and archiving test procedures.

Method I suggested use of the existing AFAL ATP and/or the Sperry Univac ATP. Method II required the development and use of a simulator to validate either or both ATP's to be used. Method III called for the design of a new set of test programs that would utilize bootstrap and/or control console functions for entering programs and test data and for reporting results. Method IV also consisted of design of a new set of test programs, but to be controlled from the VAX-11/780 computer via data links. Detailed evaluation of these methods showed that Method IV was technically superior to the other methods and that Method I was of lowest cost.

The recommended approach was to design a test program that maximizes the number of instruction codes and instruction code sequences to be executed by the UUT. It would contain test modules defined so that the testing effort could be concentrated in a particular architectural entity. A simulator would be used to obtain the expected results. Test programs, data, and expected results would be transferred between the UUT and the VAX-11/780 test control computer via an RS-232C or MIL-STD-1533B interface. The VAX test control program would produce summary reports which include test modules that were executed and provide information about test failures. The test control program would invoke options that were or were not tested.

VERIFICATION SOFTWARE (Intermetrics)

The future approach for verification testing will be the Verification Software (VSW). It is divided into two main parts; the VAX-11/780 resident VSW Control Programs and the 1750 resident UUT executive and test programs. The VSW Control Programs are divided into two functional segments, the User Interface and the Test Execution, Analysis and Reporting Segment. The 1750 resident software is divided into two functional segments, the Test Control Executive and the Deterministic Test Programs.

VSW will have the capability to set up configuration parameters of the UUT such as MIL-STD-1750 options, spares, and characteristics of the UUT. The VSW will also set up test control parameters that describe communications and testing modes, which include test composition, and reporting. Two methodologies will be used, deterministic and random testing. In deterministic testing, defined MIL-STD-1750 features will be verified using deterministically chosen test cases. In the random testing, streams of randomly generated legal instructions are executed in both the UUT and a simulator hosted on the VAX computer. The Air Force supplies, maintains, and provides configuration management of this simulator. The analysis portion of the test compares test case results to expected results obtained from the simulator to detect errors.

FUTURE APPROACH

Once the VSW is operational, the ATP will no longer be used as the test for compliance to MIL-STD-1750A. Test requirements will be as mentioned before except that the testing will be done at SEAFAC exclusively and either a single RS-232C I/O channel or MIL-STD-1553 data bus will be required.

SUMMARY

The proliferation problem of multiple ISA's results in multiple hardware and software systems, leading to high software costs. Using MIL-STD-1750A ISA, we can reduce software costs, software development time and encourage use of new technology by industry. Use of the ATP as the verification testing tool to

ensure architectural compliance to MIL-STD-1750A makes these benefits possible. The verification process will be enhanced by the use of the VSW when it comes available in the third quarter of 1983.

BIBLIOGRAPHY

Kushner, M. L., Reisiger, D. C., Tracz, W. J., White, L. A., "MIL-STD-1750 CERTIFICATION STUDY". Final report, IBM No. 6176175A, February 29, 1980.

Sperry Univac, "MIL-STD-1750 CERTIFICATION STUDY". Final report, Document No. PX 13243.

Reisiger, D. C., Kushner, M. L., Stapp, D. O., "CERTIFICATION OF MIL-STD-1750 COMPUTERS".

Intermetrics Inc., "FINAL COMPUTER PROGRAM DEVELOPMENT SPECIFICATION OF THE 1750-RESIDENT SOFTWARE FOR 1750 VERIFICATION SOFTWARE". F33657-81-C-0448, IR-OH-00602, 9 September 1982.

Intermetrics Inc., "PRELIMINARY COMPUTER PROGRAM PRODUCT SPECIFICATION OF THE 1750-RESIDENT SOFTWARE FOR 1750 VERIFICATION SOFTWARE". F33657-81-C-0448, IR-OH-021, 9 September 1982.

Intermetrics Inc., "PRELIMINARY INTERFACE DOCUMENT FOR 1750 VERIFICATION SOFTWARE". IR-OH-019, 15 October 1982.

USAF, "MILITARY STANDARD SIXTEEN-BIT COMPUTER INSTRUCTION SET ARCHITECTURE". 2 JULY 1982.

BIOGRAPHICAL SKETCH: LUIS E. VELEZ

Mr. Velez was born in Mayaguez, Puerto Rico in 1957. He received the B.S. degree in electrical engineering from the University of Puerto Rico, Mayaguez Campus in 1979. Between January 1980 and March 1981, Mr. Velez worked in the Electronic Warfare, Expendables and Optics Branch of Aeronautical Systems Division, Wright-Patterson AFB. Since April 1981 he has been working with the MIL-STD-1750A Group within the Systems Engineering Avionics Facility at Wright-Patterson AFB, and is currently responsible for all Air Force architectural testing of Mil-Std-1750A computers.

MIL-STD-1815

ADA HIGH ORDER LANGUAGE

SESSION CHAIRMAN: Paul Wood
Sperry Univac Corp.

MODERATOR: Clyde E. Allen
Vice President, Product Engineering

ABSTRACT

Update of the State of Ada Language Standardization
and Other Ada Related Standards

Larry E. Druffel

Ada Joint Program Office (OUSDRE R&AT)

The Ada Program is a joint service activity to adopt a common language and support system for embedded computer applications. It serves as the basis for a shared environment through which the state of practice for defense software systems may be improved. This presentation will provide an update of the state of Ada language standardization and other Ada related standards.

BIOGRAPHIC SKETCH

NAME: Larry Druffel, Lt. Colonel, USAF

TITLE: Director, Ada Joint Program Office

RESPONSIBILITY: Manages the DoD Ada Program.
Coordinates all DoD Ada activities and provides principle DoD contact with non-defense organizations.

Ada Experience:

- o DARPA member of the High Order Language Working Group
- o Managed development of STONEMAN
- o Managed development of Ada Compiler Validation Capability

Education:

B.S. Electrical Engineering	- University of Illinois
M.S. Computer Science	- University of London
PhD Computer Science	- Vanderbilt University

Other Experience: Nineteen years experience in computers and communications, including four years at the Defense Advanced Research Projects Agency and five years as an Associate Professor of Computer Science at the U.S. Air Force Academy. Author of over 25 professional papers on a variety of computer related topics. Senior member of the IEEE. Member ACM.

Abstract

NAVY TRANSITION TO ADA - POTENTIAL FOR A FRESH START

There are many problems associated with Navy software development and post deployment support that must be solved in order to provide affordable systems that enhance fleet readiness. These problems will intensify in the coming decade with the increased use of embedded computer resources in Navy weapon systems. The Navy's computer software development and support workload is clearly outstripping Navy and industry resources to adequately execute. The situation only promises to worsen as national competition for the diminishing supply of trained weapon system engineering personnel intensifies. One workable solution appears to be through application of the productivity improving concepts of automated support embodied in compatible, large scale computer-based support systems, such as the Ada Programming Support Environment. This paper discusses the Navy's current software engineering initiatives, short and long range Ada implementation plans, and Ada implementation policy. A short discussion of current Navy language standardization policy is also included.

Biography

Owen L. McOmber
Tactical Embedded Computer Program Office
Headquarters, Naval Material Command
Washington, D. C. 20360

Experience

Association with the Navy spans 35 years, including electronics, computer hardware, and computer software. Joined the Naval Tactical Data Systems Service Test Team in 1960. Upon retirement from active Navy duty in 1967 accepted position of Head, Systems Programming at the Fleet Combat Direction Systems Support Activity, Dam Neck, Virginia. Accepted present assignment with the Tactical Embedded Computer Program Office in 1979.

Current Assignment:

Software Acquisition and Policy Manager in the Tactical Embedded Computer Program Office. Navy Program Director for research, development, and life cycle support of Navy standard support software, including Ada; NAVMAT software acquisition management and development policy; MIL-STD-1679; Chairs Navy language configuration control boards; Navy focal point for software related efforts in DoD/Joint arena.

INTRODUCING ADA INTO THE USAF

MAJOR DAVID A. HAMMOND

Major Hammond is Deputy Director for Architecture and Language Standardization, Directorate of Computer Resources, Headquarters Air Force Systems Command. In that capacity, he is heavily involved in planning for the introduction of Ada into Air Force systems. Prior to moving to the headquarters, he was Chief of Information Management in the Space Shuttle program office. Before that, he was Computer System Manager on the Advanced Location Strike System program. His previous assignments include the Pave Onyx defense suppression, Igloo White surveillance, and 407L Tactical Air Control System programs.

CAPTAIN MICHAEL C. VINYARD

Captain Vinyard is a Computer Systems Standardization and Planning Officer with the Directorate of Computer Resources, Headquarters Air Force Systems Command. He is the editor of the Air Force Systems Command Computer Resource Newsletter and is involved in all Air Force weapons systems hardware and software standardization efforts. He was previously assigned to the Air Force Weapons Laboratory, Albuquerque, New Mexico. While there, he was a member of the three man team that designed and built the data acquisition and processing systems for the TRESTLE Electromagnetic Pulse Simulator. He later served as system manager for that system.

ABSTRACT

Air Force Systems Command has developed a four-phased plan for introducing Ada into Air Force systems, and has established definite criteria for advancing to each successive phase. The objective is to start using Ada as soon as possible, but to do it prudently. The four phases are: laboratory development and exploration; product division parallel operational system development; use of Ada on selected programs; and mandatory use of Ada on nearly all programs. At each phase there are readiness criteria related to compiler maturity, completeness of tool sets, availability of documentation, maintenance, and training, and adequacy of the experience base. The criteria can prevent errors of haste that have plagued past Air Force language introductions; they can also provide near-term goals for projects in the early phases. The next step in the planning process is to design projects for each phase that will accomplish the goals.

AD-P003 556

Look before, or you'll find yourself behind
Poor Richards Almanack, 1732

ADA AS A PROGRAM DESIGN LANGUAGE
A Rational Approach to Transitioning Industry to
the World of Ada Through a Program Design Language Criteria

Robert M. Blasewitz

RCA Government Systems Division
Missile and Surface Radar
(609) 778-3955

ABSTRACT

The Department of Defense requirements to use the higher order language Ada* by the mid-1980s will create challenges to developers of military software that encompass two major concerns: (1) developing a core of Ada software personnel, and (2) achieving productivity and software quality gains that have been targeted as Ada life cycle objectives. Because of recent government direction to use Ada-based PDLs, many organizations are developing prototype Ada-based design methods. The IEEE working group on "Ada as a PDL" is working on guidelines for the use of Ada-based design languages. The guidelines will include recommendations reflecting the current state of the art as well as alternative approaches in order to preserve good practice. The extent of industry's involvement with Ada PDLs, and also the status of the IEEE guidelines, may substantially impact both the acceptance of the Ada language and the efficiency of its use.

INTRODUCTION — PROGRAM DESIGN LANGUAGES AND ADA

During the past several years, industry has seen an explosion in the cost of generating and maintaining software products, coupled with a decline in the quality and reliability of the software product. A need for a radically different approach to the development of software is readily apparent.

One of the first tools for documenting software — the flowchart — was developed from the belief that a program should be documented after it was written. Today, the view is that program design and documentation, at the very least, must precede coding.

*Registered trademark of the US Government (AJPO)

A current tool for software design is the program design language (PDL), or pseudo-code approach. PDLs are based on a common theme of software engineering that complex technical developments require an iterative approach. The human mind works more efficiently at successive refinements of an idea — seldom is perfection achieved on the first attempt. However, iterative coding can be burdensome because of the very exacting syntax of the programming language. Pseudocoding allows the developer to experiment with many successive versions of a program with a minimum investment in time.

What is pseudocoding? It is a mixture of language-oriented control key words and English-like statements used to concretely describe an abstract design. It should support the following identifiable software goals:

- (1) Focus attention on appropriate levels of design detail without becoming overwhelmed with minor issues.
- (2) Provide a process that is amenable to the creation of well-structured programs
- (3) Replace flowcharts and other difficult software tools with an efficient approach to software production.
- (4) Provide a natural transition from high levels of logic abstraction into detailed code.
- (5) Facilitate program logic documentation and maintenance.

Besides their use in coding, many computer scientists believe that Ada-compatible design methods and languages can also be applied throughout software life-cycle phases. Such uses include requirements definition, specification, and analysis; design; testing; documentation; maintenance; and project management and control. Many organizations have already derived their own unique Ada-based pseudocode or design language in order to (1) allow immediate training in the Ada language (2) realize Ada's software potential as a design tool; and (3) respond to present DoD interest in program design languages as specification tools via RFPs.

This paper will outline the importance of creating a guideline for an Ada Program Design Language (PDL) and will present the activities of the IEEE working group on "Ada as a PDL." The value of a guideline (IEEE goal) or recommended practice will aid the widespread acceptance and use of Ada. If a guideline is produced, it will illustrate software practices centered around an Ada PDL that will support and enhance military documentation.

The selection of either a guideline or recommended practice is accomplished by means of a consensus of technical opinion within the IEEE working group. It is hoped that this group's efforts will not have damaging effects on present corporate investments in PDL design. However, the IEEE guideline will be directly influenced by the lessons learned from these developers and will hopefully bring together the wide scope of work in the PDL area. The availability of a PDL tool in the APSE will also foster development of DoD software throughout the life-cycle of the software, if such an end product can be realized in a timely fashion. (Some of the technical problems associated with the IEEE effort will be detailed throughout this report.)

DoD initiated the Ada program to save taxpayer money, with savings coming from the portability and reuse of operational software, more effective use of support software (including PDLs), improved programmer productivity, and reduced software maintenance. There is little question that the entire software industry is in need of a modern, efficient, and highly portable system implementation language and tool set. In summary, technical arguments about which language to use at best miss the point, for only the Ada language will benefit from DoD's investment in standards enforcement for compilers and supporting tools, and in particular a program design language tool.

PROGRAM DESIGN LANGUAGES — WHY A COMMON ADA PDL?

Computer programming is unlike other engineering disciplines in that it is exclusively a design process. The software life cycle is initiated with a functional requirement; the design of the system interfaces with its external environment; continues with the design phase (architectural, module, internal interfaces, data bases); and is followed up with coding, a stepwise refinement of previous designs. Testing, integration, and maintenance of these phases are repeated throughout the development process.

With this point of view, software tools can be considered design tools. One of the most widely used tools to construct software is the programming language, which is really a software design language although a machine that executes algorithms also embodies a design language. Algol, Pascal and Ada are prime examples of design languages that are also used as programming languages.

Higher order programming languages typically have been successfully used as tools at the coding phase of software development cycle. Tools that supported software designs earlier in the development cycle were informal ones, such as Von Neumann Flow Charts, Hierarchical Structure Charts, and Data Flow Diagrams. These graphical methods worked at different levels of detail. For instance, detailed flow charts may tend to correspond one-to-one with a machine-language program, a logical flowchart could have English text, which might correspond to any amount of machine code in its defined representation (usually boxes, diamonds, etc.).

One machine-processable design language tool that is widely used is "P.D.L.", a product of Caine, Gordon and Farber, Inc., which provides a programming language-like structure for expressing in English or "structured English" what a to-be-written computer program will do without having to describe exactly how it will do it. Informal expressions of program design follow the P.D.L. idea quite closely. From Wirth, the programming language is Pascal, which is used as a design language by enclosing non-Pascal descriptions in English, set off by quotation marks, as in:

```
begin "move an item from j to k" end
```

Dijkstra uses a mythical programming language designed to express formally provable algorithms expressed with non-deterministic flow of control.

In considering Ada as a program design language, we can use the definition from IEEE Standard 729, "Standard Glossary of Software Engineering Terminology," (Approved by IEEE Standards Board, 23 September 1982):

"Design Language — A language with special constructs, and sometimes VERIFICATION protocols are used to develop, analyze, and DOCUMENT a DESIGN."

An efficient PDL should not only be useful in the stage between the requirements phase and the implementation phase of a project, it should also (1) aid the designer in expressing the design, (2) aid the design reviewers in evaluating it, (3) aid the implementation and validation of the design into a product, and (4) aid the maintenance team in correcting or changing the design.

What other features should a PDL possess? Consider the following desirable features or characteristics. It should:

- (1) Be engineered as an aid to the programmer and not be an extra nuisance, as documentation is often perceived by programmers.
- (2) Be a repository for design decisions.
- (3) Have selective focus to help programmers abstract patterns from complex software.
- (4) Allow a system to be viewed top-down so that system level considerations are not overlooked respective to details of specific programs.
- (5) Support early prototyping — the PDL should aid a programmer in designing complex software systems at the beginning of a project.

A PDL should also have the capability to describe sections of a design that are 1) tentative or sketchy or 2) frozen and dependable. The PDL should resemble the actual language used in the project or at least should be machine-translatable into the project language. Programmers should find the PDL easy to learn and use within their projects.

CAN ADA BE USED AS A COMMON PDL?

Ada is new and relatively untested as a PDL, but nevertheless meets several or most of the requirements of a design language. While Ada programs are not particularly easy to write, the complexity of the language exists largely to enforce good programming practices. Ada was designed to be readable, which meets one PDL goal — of being an aid to programmers.

Like most modern higher order languages, Ada has a subroutine mechanism (procedures and functions) so that top-down programming can be practiced and partially specified programs can be put to paper. Other design language features of Ada include:

- (1) A generic subprogram — Programmers can specify programs independent of the data types to be used; i.e., code can be written without specifying all details.

- (2) Packages, which allow external interfaces of a software module to be stated while concealing the main sections.
- (3) Pragmas that allow special requirements to be communicated to the compiler, which would in turn allow the processor to know that a program text under consideration is being used for a design language or for final program description.

An Ada PDL will surely support early design activities as well as a compiler, but what of other advantages? The basic advantages to using an Ada PDL include:

- (1) It is a "human-engineered" design tool that is available early in the software life cycle.
- (2) A programming staff can be trained to full Ada programming, which reduces costs.
- (3) Enhanced communications can occur among managers, customers, and maintenance personnel.
- (4) Detailed design can be eliminated until required by the project.
- (5) It can be used for Ada software design until a validated Ada compiler becomes available.

ISSUES ASSOCIATED WITH ADA AS A PDL

There is currently time to explore and resolve alternative usable strategies of Ada as a PDL. Some basic issues derived from the IEEE working group meetings include:

- (1) What features separate an Ada PDL from a non-Ada PDL?
- (2) Should the Ada PDL be a subset or superset of the Ada language?
- (3) Should an Ada PDL try to encompass software engineering methodology? Is one methodology more suitable than another?
- (4) Will the Ada PDL be machine processable or compilable by an Ada compiler?
- (5) What tools will be needed to support the Ada PDL?
- (6) Should the PDL be based upon the syntax and semantics of Ada?
- (7) How does the PDL relate to required documentation?
- (8) How will the PDL be used to support all the phases of the software life-cycle?
- (9) Should the PDL aid and support formal verification, testing, and V&V techniques?

Obviously Ada, by itself, provides much of what we need in terms of a PDL. However, the answers to the above issues are required before a common Ada PDL can be designed and developed. The answers to these questions may or may not lead to a single Ada PDL and methodology (under the assumption that Ada does not represent a

methodology). Because Ada has such strong features, which enable it to support software design activities, many organizations* have already developed prototype Ada-based PDLs. If a commonly accepted PDL is developed and enforced by the Department of Defense, will these developers be satisfied with the return on their investment?

The IEEE Working Group on Ada as a PDL has forecast that a guideline for Ada-based design will be completed by early 1984. This guideline will include recommendations reflecting the current state of the art and alternative approaches to good practice. Whether or not the group will recommend a single Ada-based PDL (or a single methodology) remains to be seen.

EDUCATION AND TRAINING ISSUES

The introduction of Ada as a PDL will involve more than just learning a new language or a pseudo-language. It offers a unique opportunity to provide training in modern programming methods that are appropriate to Ada but which are inappropriate to lower level languages. Realizing that Ada permits a coordinated view of modern programming practice, complete with a language and support environment, does it not make sense that a common Ada-based PDL will do the same?

Ada was designed to incorporate the best software engineering research of the 1970s. An Ada-based PDL will certainly also encapsulate the same advances in its design. It will reduce personnel training requirements, increase proficiency in software-related job skills, and give managers a "larger" labor force from which to draw. For DoD to derive these benefits at an early stage in the software life cycle, the Ada-PDL must be widely accepted and used. The rewards could include:

- (1) Conservation of manpower, time, and money
- (2) Enhancement of portability, reliability, and maintainability
- (3) Optimal use of processes and practices relating to software development
- (4) An early transition of personnel to Ada-based design and Ada usage.

Not only is the software procurement community interested in Ada and software development methodology, but so are a number of organizations that are required by the DoD to make a transition to the new language. Transition is the key — can there be a simpler, more esoteric means of achieving this transition than by means of a common PDL?

*A full compilation of industry activity in the area of PDL development is also compiled by Ada TEC (the SIGPLAN Technical Committee on Ada by the Design Methodologies group).

THE AIM OF THE IEEE IN PRODUCING A GUIDELINE

The main objective of the IEEE is to advance the theory and practice of engineering or related arts and sciences. In a 1968 membership attitude survey, a large percentage of IEEE members indicated that they felt that issuing of technical guidelines was an important means by which IEEE could carry out its objectives, providing a common ground among those using the guidelines. Guidelines also could provide criteria for the acceptable performance of equipment or materials pertinent to the electrical engineering field. IEEE guidelines are published to record a consensus of the engineers who are substantially concerned with the scopes of these guidelines, reflecting the best thinking of the experts of the institute and ensuring that proper procedures have been carried out. The production of an Ada-based PDL guideline is certainly in line with IEEE objectives.

SUMMARY

The advent of the Ada programming language provides a means for bridging the gap in software development methodology. Ada, by means of introducing formalized constructs such as packages, generics, concurrent tasks, exceptions, and separate program unit specifications, provides design representation. The use of Ada as a PDL is not only a realizable goal, but one that has been achieved by a number of organizations at this time.

The IEEE Ada as a PDL Working Group has been chartered to generate, at the very least, a guideline document for Ada-based PDL(s). The derivation of a guideline by the IEEE working group will add to the momentum of the Ada effort and should help ensure both the acceptance of the Ada language and its efficient use as a learning mechanism through its use as a PDL throughout industry.

BIBLIOGRAPHY

"Ada Process Description Language Guide," Software Methodology Development Group, Harris Corporation, Government Communications Systems Division (17 March 1982).

Anderson, P.G., "A Design Language Based on Ada," Rochester Institute of Technology (May 17, 1982).

Bogdon, W. R., "Ada Program Design Language Issues," NADC, Warminster, PA (1982).

Buckley, F., "IEEE Software Engineering Standards — An Administrative Overview," presented to the Ada as a PDL Working Group (May 1982).

Chase, A. I. and Gerhardt, M. S., "The Case for Full Ada as a Design Language," Raytheon (1982).

Carlson, W. E., Fisher, D. A., "First Complete Ada Compiler Runs on a Micro," Western Digital Corp., Mini-Micro Systems (Sept. 1982).

Dijkstra, E., "GO TO Statements Considered Harmful," Communications of the ACM, March 1968, pp. 147-148.

Freeman, P. and Wasserman, A., "Tutorial on Software Design Techniques," IEEE Computer Society (1977).

Hart, H., "Ada for Design: An Approach for Transitioning Industry Software Developers," TRW, (14 Oct. 1981).

"IEEE Standards Manual and Style for IEEE Standards," The Institute of Electrical and Electronics Engineers, Inc. (March 1978).

Ledgard, H., Ada, an Introduction, Springer-Verlag, NY (1981).

Jackson, M., "The Jackson Methodology," IEEE Computer Society (1977).

Minutes from IEEE Ada as a PDL Working Group (10 June 1982).

Oberndorf, P.A., "Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report," NOSC Technical Document 509 (TD 509) (1 April 1982).

Privitera, J. P., "Ada Design Language for the Structured Design Methodology," Ford Aerospace and Communications Corporation (1982).

"Rationale for the Design of the Green Programming Language," Honeywell, Inc. and CII Honeywell Bull (March 15, 1979).

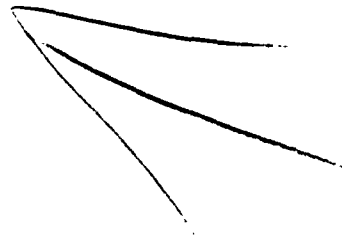
Saib, S. H., "An Ada Program Design Environment," General Research Corp. (Sept. 1982).

Sammet, Waugh, and Reiter, "PDL/Ada — A Design Language Based on Ada," Proceedings of the ACM Annual Conference (Nov. 1981).

Wirth, N. "Program Development by Stepwise Refinement." Communications of the ACM, March 1968, pp. 147-148.

Yourdon and Constantine, Structured Design, Fundamentals of a Discipline of Computer Programs and Systems Design, Yourdon Inc., NY (1978).

Robert M. Blasewitz, Unit Manager, Software System Design, has been responsible for systems engineering and programming for a wide range of scientific and real-time applications, including costing, budgeting, contract negotiation, and systems consultation. He has extensive software and hardware computer architecture experience, and has published a textbook on Microcomputer Systems. He has been Program Manager for the AEGIS EDM-1 Software System at Computer Sciences Corporation and lead evaluator of the Ada Programming Support Environment (APSE) for RCA. He is IEEE subcommittee chairperson for the "Ada as a PDL" Standards Working Group.



THE KAPSE INTERFACE TEAM

Patricia A. Oberndorf

Naval Ocean Systems Center
Code 8322
San Diego, California 92152
(714) 225-6682

ABSTRACT

The Kernel Ada* Programming Support Environment (KAPSE) Interface Team (KIT) was established by the Ada Joint Program Office to address the issues involved in sharing Ada support tools among APSEs. The goal is to establish interface standards which will assure the transportability of tools and data bases between APSEs. The KIT and its auxiliary team, the KAPSE Interface Team from Industry and Academia (KITIA), have been meeting since early 1982. The paper discusses their progress to date, the plans for the future and the major issues confronting them in achieving this major goal of the Ada program.

1. BACKGROUND

From its inception, one of the major objectives of the Ada program was to provide the three military services with a common programming language which would allow them to make use of a common set of programming support tools. In the past, each service had its own standard language and a requirement to invest time and money in the development and maintenance of the unique tools required to write systems in that language. Thus each service developed compilers, linkers, loaders and other support tools which none of the other services could use. With Ada it was anticipated that the three services could now develop tools which were of interest to the other services and which could be shared.

The first step in making this sharing a reality came with the publication of STONEMAN in February 1980. STONEMAN presents the requirements for an Ada Programming Support Environment (APSE). It discusses some general characteristics of an APSE and the minimal set of tools which an APSE must contain (thereby called a Minimal APSE or MAPSE). In order to address the requirements for sharing, STONEMAN put forward an architecture based on a machine-transportable kernel of software (the Kernel APSE or KAPSE). A KAPSE provides the interfaces the tools need to function in a way that is not dependent on a particular operating system or machine. Thus to move the tools from one machine to another, it is not necessary to change any of the tools; it is only necessary to provide a KAPSE on the new machine which will

* Ada is a trademark of the Department of Defense (Ada Joint Program Office)

support the tools with the same interfaces as the KAPSE on the old machine. The STONEMAN picture of this is shown in Figure 1.

In late 1979, while the finishing touches were being put on STONEMAN, both the Army and the Air Force announced plans to develop MAPSEs. The Army contract for the Ada Language System (ALS) was initiated with SofTech Inc. in June 1980. The first ALS version that generates code for a target computer is expected in late 1983. The Air Force contract for the Ada Integrated Environment (AIE) began with a six-month competitive design phase, which was won by Intermetrics, Inc. with Massachusetts Computer Associates. The second phase (to develop the AIE) was initiated in May 1982. The first delivery of the AIE is expected in 1984.

The existence of two different contracts for the development of two different MAPSEs presented the Ada Joint Program Office (AJPO) with a problem. Each of these MAPSEs includes a KAPSE, a unique KAPSE that would most likely be unable to support the tools being developed by the other service. This would put the DoD in the same position again: Army tools for Army systems, Air Force tools for Air Force systems, etc., and no way to share them. It was clear to the AJPO that, in addition to agreeing on a KAPSE-oriented architecture, it would be necessary to standardize on the set of interfaces which the KAPSEs provide in support of the tools.

In order to achieve this standardization of KAPSE interfaces, the AJPO obtained a Memorandum of Agreement (MOA) between all three services. This MOA established the KAPSE Interface Team (KIT), a Navy-led DoD team, as the agent for formulating appropriate KAPSE interface standards. The MOA also contains agreements by the Army and Air Force to evolve the ALS and AIE, respectively, towards these standards when they are established.

Subsequent to the organization of the KIT, it was decided to also organize an auxiliary team of experts from industry and the universities. This team, the KAPSE Interface Team from Industry and Academia (KITIA), is intended to provide the KIT with a broad range of expertise in interface problems and a source of ideas and feedback.

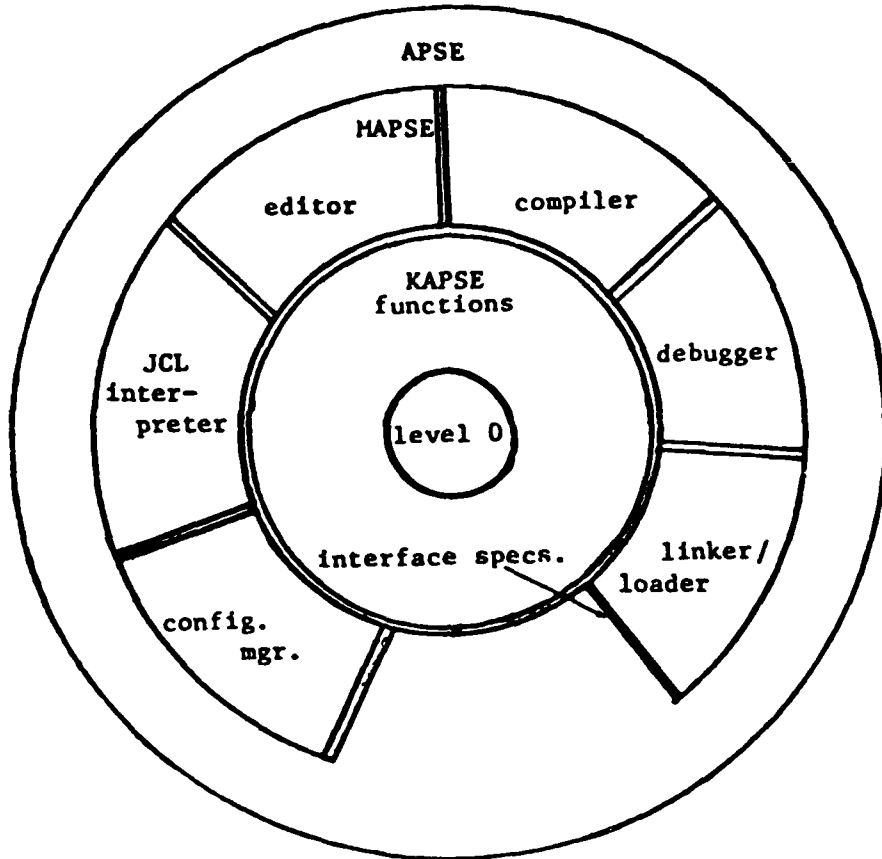
2. KIT STATUS

The KIT held its first meeting in January 1982; the KITIA's first meeting was the following month. Since then each team has had three other meetings, including their first joint meeting in October 1982. The work to date has been in five major areas: basic definitions, KAPSE interface categories, AIE/ALS analysis, requirements and tool development.

2.1 BASIC DEFINITIONS

Definitions for terms that are central to this work have been established. Chief among these are:

INTEROPERABILITY - Interoperability is the ability of APSEs to exchange data base objects and their relationships in forms usable by tools and user programs without conversion.



level 0: Hardware and host software as appropriate

level 1: Kernel Ada Program Support Environment (KAPSE), which provides database, communication and run-time support functions to enable the execution of an Ada program (including a MAPSE tool) and which presents a machine-independent portability interface.

level 2: Minimal Ada Program Support Environment (MAPSE) which provides a minimal set of tools, written in Ada and supported by the KAPSE, which are both necessary and sufficient for the development and continuing support of Ada programs.

level 3: Ada Program Support Environments (APSEs) which are constructed by extensions of the MAPSE to provide fuller support of particular applications or methodologies.

FIGURE 1

TRANSPORTABILITY - Transportability of an APSE tool is the ability of the tool to be installed on a different KAPSE; the tool must perform with the same functionality in both APSEs.

Additional terms are being defined as their use becomes important to the effort.

2.2 KAPSE INTERFACE CATEGORIES

As a starting place for organizing everyone's thoughts, a set of interface categories was created. These are areas which the collective wisdom and experience of the KIT and KITIA members show to be important to interoperability and transportability (I&T). The current interface categories are:

- Program Invocation and Control
- Logon/Logoff Services
- Device Interactions
- The Subset MAPSE
- Basic I/O Interfaces
- Database Management and Control
- Inter-tool Data Interfaces
- Ada Program Run-Time System (RTS)
- Bindings and Their Effect on Tools
- Performance Measurement
- Recovery Mechanisms
- Distributed APSE
- Security
- Support for Targets
- Extensibility

Each of these is described in a KAPSE Interface Worksheet. The descriptions give an explanation of the interface category, its relevance to I&T, standardization problems it presents, its priority, other categories that are related to it, proposed approaches (e.g., AIE, ALS), the risk associated with not standardizing in this category and recommended actions to be taken. These descriptions are not yet complete and will evolve as the work progresses.

2.3 AIE/ALS ANALYSIS

One of the most important KIT/KITIA jobs in this first year is to understand the similarities and differences between the ALS and the AIE. There are several reasons why this is of great interest to the KIT/KITIA:

(1) as the first KAPSEs, these two systems will provide the KIT/KITIA with a great deal of insight into the basic properties and interfaces that are required of a KAPSE;

(2) as the first DoD MAPSEs, these two systems will be required by the MOA to evolve towards the standards established by the KIT/KITIA; therefore, the KIT/KITIA do not want to choose interface standards which are arbitrarily different from the interfaces found in the AIE/ALS; this is not to say that

the standards will not differ from the ALS/AIE, but that an attempt will be made to be consistent with their interface choices when there is no reason to differ;

(3) the KIT/KITIA is also charged by the MOA with making recommendations for how the AIE/ALS can be changed to meet the standard; this task requires intimate knowledge of how and why the two systems differ, both philosophically and technically, and a firm understanding of what can be changed without destroying the very differences that make the dual development worthwhile.

2.4 REQUIREMENTS AND CRITERIA

Any undertaking of this size and importance must have some ground rules and requirements which guide the decision-making process; this effort is no exception. The requirements and criteria define the guidelines we plan to use in choosing which interfaces to standardize and the form those standard interfaces should take. Many of the requirements and criteria have been formulated using three Operating System Command and Response Language (OSCRL) requirements documents as models: the OSCRL User Requirements, Functional Requirements and Design Criteria. In addition, as other issues arise and are resolved, those resolutions are added to the requirements and criteria document in the appropriate section.

2.5 TOOL DEVELOPMENT

The MOA which created the KIT also called for the KIT to provide for the development of three or more APSE tools. The purpose of these tools is to provide the KIT with real experience with the AIE and ALS interfaces. They are each to be developed to run on both systems. While each tool is to be functionally useful to the process of developing and/or maintaining Ada programs, their main purpose is to expose interface differences and transportability problems which arise as a result of trying to develop them to run on both the ALS and AIE.

Two of the tools to be so developed have been chosen from the designs that were not selected for the AIE. One will be a Configuration Management System by Computer Sciences Corporation and the other will be an APSE Interactive Monitor by Texas Instruments. One or more other tools will be selected through competitive procurement.

Representatives from the tool-builder companies participate on the KIT. All of their experiences and recommendations will be documented for the KIT and will form another part of the basis for the interface standards.

3. KIT PLANS

All of the work discussed in the last section is still in progress. Each will evolve during the remaining three years of the KIT/KITIA work. The definitions are relatively stable, and the descriptions of the KAPSE interface categories have reached a plateau and will now be completed as new information becomes available. The AIE/ALS analysis and the requirements and criteria are both in their early stages, and the tool developments have a number of steps left to take. In addition, three other major areas of work

have not yet been initiated: interface standards, interface implementation, and guidelines, conventions and standards documents.

3.1 AIE/ALS ANALYSIS

A basic model for comparing the AIE and ALS has emerged. Now this model must be used to display the capabilities of each system. In this way it can be seen where they have features in common and where they differ. The common features will help provide a first draft of the interface standards. The differences may be of two kinds: some differences may lie in areas which are covered by one system and not by the other, while other differences may lie in incompatible treatments of the same area. In either case, these differences will be important to further KIT/KITIA decision-making.

3.2 REQUIREMENTS AND CRITERIA

These will continue to evolve from the October rough draft. As new issues are resolved, the results will be incorporated. Several areas will be refined and made more consistent. In addition the KIT/KITIA are looking at deficiencies in STONEMAN and making recommendations for revisions to it.

3.3 TOOL DEVELOPMENT

The first two tool developments are underway and should have initial implementations ready in the next three months in one case and about eighteen months in the other. Following initial development, each of these tools must in turn be integrated into the ALS and the AIE as those systems become available.

The competitively procured tools will follow the same pattern. It is expected that these tools will be selected and underway during the first half of 1983.

3.4 INTERFACE STANDARDS

The first rough incomplete version of the interface standards will consist largely of those interfaces which the AIE and ALS are found to have in common. This initial version will evolve as ALS/AIE differences are examined and other decisions are made.

To the greatest extent possible, these interface standards will be expressed as Ada package specifications. These will be augmented by narrative or other more formal specification techniques for those aspects which package specifications do not cover (e.g., semantics).

3.5 INTERFACE IMPLEMENTATION

Although it is not the objective of this work to produce a standard KAPSE implementation, it also is not realistic to expect that 50 people from across the U. S. and Europe will be able to develop a complete, consistent, workable set of interface standards without the ability to "try them out." The ability of the tool developments to address this need is limited. Thus a means will be found to experiment with the emerging interface set in order to

assure its practicality and suitability. A full implementation may not be necessary; some means of simulation/emulation may be more useful.

3.6 GUIDELINES, CONVENTIONS AND STANDARDS

The KIT/KITIA will document to the greatest extent possible the ideas and lessons learned in this effort, in addition to the standards themselves. We expect to publish guidelines for the use of the standards and for attention to other topics related to I&T which are not addressed by interface standardization (e.g., designing tools for reusability). These guidelines will be directed to both the tool builder and the KAPSE builder. They will also contain the KIT/KITIA recommendations for a future agency whose charter is to verify conformance of KAPSEs to the interface standards.

The conventions document will be used as a holding place for potential standards. It will represent practices that are encouraged but not yet standardized. For areas in which standardization is considered to be premature, it will contain various possibilities for a standard in the area. This document will evolve as the standards evolve.

The standards document will be in a form appropriate for issue as a military standard. As such it will contain a number of required sections in addition to the Ada package specifications which document the standard interfaces.

4. MAJOR ISSUES

Since the KIT and KITIA are still in their infancy, the majority of the issues still lie ahead. The following is a sampling of the issues that have been raised to date.

4.1 DOD POLICY

There is a major concern, particularly among the industry representatives, that the KIT/KITIA effort may be too late. To much of the world outside the DoD, it appears that once again each of the services is pursuing its own direction, thus defeating the tool-sharing objective. While the Navy plans to use the Army system to the greatest extent possible, it still appears that two DoD-sponsored APSEs is one too many. The DoD must formulate a clear policy for dealing with the existence of two APSEs in the near future and for how it plans to evolve to one in the long run.

4.2 RESOLUTION OF ALS/AIE DIFFERENCES

Although this issue is related to the first one, it expresses very real technical concerns. There appear to be some major philosophical differences between the approaches taken to the ALS and AIE. While these differences are part of the reason for having two DoD efforts, their eventual resolution is not obvious. Chief among these concerns is the apparent disparity between the approaches taken to the central data base; for example, one is hierarchical while the other is relational. Such differences need technical solutions as well as policy ones.

4.3 SECURITY

Multi-level secure support systems for use in the development of military systems are currently receiving a great deal of attention. Since a system such as an APSE cannot be more secure than the system which hosts it, the APSE cannot guarantee security. But it is desirable that the KAPSE be built in such a way that it does not compromise security provisions that do exist in an underlying system. The proper approach to this relatively new aspect is very much at issue.

4.4 SEMANTICS

While Ada package specifications are an excellent form for the expression of the interfaces themselves, it is not as clear how to describe the semantics intended by the standard. The semantic systems developed for programming languages are generally quite complex, but English narrative is known to allow too much ambiguity for a rigorous specification. Some satisfactory middle ground must be found which suits the needs of rigor while being appropriate for use in a military standard document.

4.5 VALIDATION

Although it is not in the KIT/KITIA purview, the question of how one will validate the conformance of a KAPSE to the interface standard is an important one. It is clear that what the KIT/KITIA produce could have a very great influence on how validation can be approached. While no answers are available today, this issue is being investigated, and it is expected that the results will contribute to decisions concerning the interface standards.

5. CONCLUSIONS

The KIT/KITIA work is very important to the future of Ada and its ability to fulfill its objectives. A large number of very difficult issues and decisions lie ahead. Widespread participation and comment are invited, as they have been with all other aspects of the Ada program.

A Standard Run-Time Executive for Compiled Ada.

Ben Hyde, Intermetrics

A vast range of applications will be addressed with Ada. Can a single runtime executive satisfy the the needs of both a multiprocessor operation system and a toaster oven controller? It seems unlikely. The programming environment that supports the Ada programmer must allow him to progressively refine his runtime system. There are important advantages to a standard, and they do not need to preclude the tuning of a system. A catalog of standard runtime support components can go a long way toward providing the advantages of a standard. Such a catalog should have at least these features: a modular design allowing new components to address special needs, a default set of modules to support all Ada semantics, and a set of specialized modules to allow fine tuning. The support enviroment tools (MAPSE tool set) must be designed with care not to preclude such tuning. The Air Force Ada Integrated Programming Environment provides some examples of how such goals may be achieved.

Mr. Hyde is a member of the Air Force Ada Integrated Enviroment (AIE) project at Intermetrics Inc. in Cambridge Mass. with special intrest in the Ada compiler, a component of that system. Mr. Hyde graduated from Carnegie Mellon University where he also taught in the Computer Science Department. Both at CMU and since, Mr. Hyde has worked extensively in the fields of multiprocessor operating system design and data communication systems. In recent years Mr. Hyde has aided in the development of a number of real time control systems in both industry and the high energy physics community.

1. The scope of the problem.

Ada will be used for a vast range of applications. Take as one example the AIE's KAPSE, an operating system's kernel [AIE]. This system provides a file system, multitasking, memory management, message systems and I/O support. On top of the KAPSE will be built a programming environment, the MAPSE. Over its life span of, say, 20 years, this system may run on many different kinds of machines. Some of those machines will be large, expensive mainframes costing hundreds of thousands of dollars.

How are the runtime system needs of that system comparable to the needs of a toaster oven controller? The toaster oven's product life span is likely to be only a few years, the software will be frozen after, say, six months of development, and it will run on only the machine costing a few dollars. It is likely that there will be more copies of the toaster control program than the AIE's KAPSE.

There are more than enough differences between those systems to make any parallels one might attempt to draw quite suspect. In spite of that there is something compelling about one image of a multitasking toaster oven garbage collecting the burnt toast while in rendezvous with an english muffin.

2. The value of standards.

Before I climb too far out on a limb, let me say there are significant advantages to standards. Just because the problem is hard does not mean we can ignore it. There are many standards that address hard problems. Data communication standards, documentation standards, and even Ada her self, are all good examples. A standard that addresses a hard problem is always a tradeoff; one hopes to gain more than one precludes.

Ada's language designers tackled a number of hard problems. The language's mechanisms for addressing these problems do not make the problems disappear. Like all standards which address hard problems, Ada's contribution is to provide a common notation.

It is a constructive contribution to any exercise to recall one's goals. Standards can deliver substantial return on almost any investment. All of the costs of any particular task, e.g., training, staffing, testing, equipping, startup, etc. are reduced by any standard. The standard allows costs to be shared by the community; why reinvent the wheel? The leverage of a standard is so great that quality can seem irrelevant.

Historically, standards developed after a consensus had developed in a community of users. Today the standards making process is often the mechanism by which consensus is reached. Anyone who has participated in this process will attest to its unpleasantness. There is substantial payoff in a standard, so substantial that the pain of reaching a consensus can be well rewarded.

3. A proposal: A catalog of runtime systems.

We at Intermetrics are addressing the problem of implementing Ada. We want to make it a powerful tool for a wide range of problems. As a part of that problem we have been addressing the problem of run time support.

There, as in the rest of the language we have attempted to achieve two not always compatible goals. First, it is clear, we must support the full semantics of the language. Second we want the user to be able to craft the system he is building exactly as he needs.

These are only incompatible goals in so far as the user may not need the full power of the language. The designers of the language have taken great care not to charge applications for language features they do not use. The pragma suppress, which allows the programmer to indicate that he accepts responsibility for bounds checking, is a typical example. The language implementor, Intermetrics to take an example, must continue that care.

One can not serve two masters. Our primary goal is to implement the full semantic power of the language. To enable both goals to be addressed, we use what the author calls the "safety net" approach. A sequential view of this approach might run as follows: first one implements the general, and then one implements the special cases. Of course, clever people may do both at the same time.

The set of cases form a "catalog"; the system builder selects from that catalog as he tunes his system. Free space management is a good example. The language leaves open the exact management technique used to control the various heaps in the users program. A default solution might implement garbage collection, or mark release. The catalog will, as time passes, grow to contain a large number of methods [Space].

One aspect of the safety net approach is that it allows the user, as well as the implementor, the advantages of successive refinement. The user can start with a rough model. Tuning can

then be done, as it should, as the system matures. This approach doesn't relieve the user of the necessity of good design, but it does reduce the risks as the system develops along unexpected paths.

I want to emphasize the importance of tuning. The language designers took great care to leave tuning to the system builder. The minimal specification of free space management is one way they did this. The powerful semantics of tasking in Ada are another.

4. An examples: tasking

I want to say a little about tasking. Ada provides very powerful tasking semantics, so powerful that many people's initial reaction to them is that they are not practical. System designers have over the last few years identified quite a catalog [Sync] of practical multitasking constructs: monitors, mail boxes, critical regions, path expressions, etc. Researchers have done extensive work on both issues of practicality and provability for all of these methods. It is a strength of Ada tasking that these methods can be used by relatively straightforward application of Ada tasking constructs. It is up to implementors to ensure efficiency.

These methods are a valuable resource. The language implementor must take care not to preclude their application within the language. We hope to do even better; we hope to make it possible for the system builder to integrate those methods with the language's constructs. A single example will help to illustrate how we hope to reach that goal.

A common example of an Ada task is a queue.

```
task queue is
    entry insert(e:in element);
    entry remove(e:out element);
end queue;
```

The task supports two kinds of access, inserts and removes. The body of the task is a simple loop whose body either accepts an insertion or a removal. In the body of the task, conditionals are used to refuse removals when the buffer is full and refuse insertions when the buffer is empty.

```
loop
    select when not empty =>
        accept remove do
            ... code to return an element ...
```

```

        or when not full =>
            accept insert do
                ... code to insert an element ...
            end select
        end loop

```

To some, this device shouldn't be a task at all; they would say its really just a data structure and two procedures. Of course, those two procedures must do some scheduling activities to avoid asynchronous access to the queue. Key to their argument is the idea that this "task" never runs except when some other task is rendezvousing with it.

This use of tasks to implement protected data structures is expected to be very common in Ada. A simple minded implementation might allocate a stack for each instance of this queue. Our intention is to allow the user, via a pragma we call `monitor`, to indicate the desirability of implementing this without the overhead of a stack.

Of course, the safety net solution would provide a stack, even if its never used. The compiler might notice the legality of a special case, for now though we leave that to the user. The catalog of possible special case implementations of the tasking primitives can be very large. For example, if a task is forced to wait for the queue, it might do a busy wait or pass control to some other task.

The details of the example are unimportant; it is the architecture used that is important. The user must be able to tune his system.

5. Some final words.

What would any talk about standards be without mentioning modularity. The successful construction of a catalog-like runtime system requires the careful specification of the interfaces between the parts. As anybody who has ever tried knows, this is no easy task.

Consider the interface that would enable three different kinds of free space management schemes to be plug compatible, say garbage collection, mark release, and explicit allocate and deallocate. There seem to be a lot of activies on this interface; what to do on an allocate, what to do to intialize a collection, what to do to intialize an access type, what to do as a background task and when to do it and on and on. Some of those interfaces lie deep within the compiler.

These are hard problems. The standardization process can make a significant contribution in providing a forum for reaching consensus about where such interfaces should be required. Hard problems don't go away just by being ignored. Making Ada an even more valuable tool requires that we work for consensus on some of these issues. I believe that the catalog approach reduces some of the risks in the process of selecting such a standard in reaching a consensus.

6. Bibliography

[Catalog]

Modularization and Hierarchy in a Family of Operating Systems.
A. N. Habermann, Lawrence Flon and Lee Cooperider
Carnegie-Mellon University,
May 1976, Vol 1^a, #5, pp266-272

This paper provides an overview of a catalog like approach to the construction of whole operating systems, a problem that shares many features with the problem of building a standard run-time executive.

[Sync]

Synchronization Primitives and the Verification of Concurrent Programs.
Sten Andler
Proceedings of the Second International Symposium on Operating Systems.
IRIA, Le Chesnay, France, October 1978.

This paper includes a pleasant annotated bibliography of papers on various methods of synchronizing a multitasking program.

[AIE]

Developing an Ada programming support environment.
Michael Ryer
Mini-Micro Systems, September 1982, p223-226

This paper provides a casual overview of the UNIX-like design of the AIE programming environment that Intermetrics is building.

[Space]

Knuth, D.E., Fundamental Algorithms, Addison Wesley, 1968

In this classic work is reviewed a number of different ways of managing a heap.

AD-P003 558

THE ADA RUN-TIME ENVIRONMENT

Dr. Joseph K. Cross

Sperry Univac, Defense Systems Division
P.O. Box 3525
St. Paul, MN 55164-0525
(612) 456-4929

ABSTRACT

The requirements on an Ada run-time environment are surprisingly few and straightforward. The free choices left up to the implementors of a run-time environment are many and significant. These requirements and freedoms are enumerated and discussed, and the importance of these issues to the success of an Ada software system is described.

DEFINITION OF "RUN-TIME ENVIRONMENT"

An Ada run-time environment is, roughly, the set of target-machine facilities that an Ada compiler can use to carry out the run-time operations required by Ada programs. Those facilities consist of the instruction set provided by the physical target machine, possibly with additions and deletions. Additions to the facilities provided by the physical target machine's instruction set are generally provided by some predefined software, such as an executive, that, in the compiler's eyes, might as well be implemented in hardware. Other additions to the physical target machine's facilities can be provided by additional hardware, such as an array processor, and by user microcode. Deletions from the physical target machine's facilities generally result from a conscious decision not to use some capability, generally in the interest of safety or simplicity. For example, after it had been decided to use a certain executive in the target machine, it might be determined that all code emitted by the Ada compiler will run only in task state; then the privileged instructions in the hardware's instruction set would not be usable by the Ada compiler, and would therefore not be part of the run-time environment.

The Virtual Target Machine

After the target hardware has been chosen, after any predefined software (exec, I/O handlers, math routines, etc.) have been specified, and after all restrictions and conventions have been imposed, we have what the compiler sees as a new target machine - the virtual target machine for which code is actually to be emitted. To the compiler, this virtual target

machine is as different from the original physical target as if it were a different box: for example, the virtual machine may have a SIN instruction while the physical machine did not, and the physical machine might let any register be used as a stack pointer, while the virtual machine reserves register 15 for that purpose.

The Role of the Executive

A common question about Ada is "What executives does it run under?" The common answer is "Ada programs don't need an executive." This leaves both parties staring blankly at each other.

In fact, Ada compilers can generate code for bare machines -- that is, a physical target machine with no support software in it. Phrased otherwise, a run-time environment need not provide the facilities generally provided by executives (such as task scheduling and I/O initiation). In such a case, the compiler emits code to perform these operations. Now, in all but the most highly optimizing compilers, this exec-function-performing code will be boilerplate -- the same for every program the compiler compiles. Hence that boilerplate code constitutes an executive, even though it was written by the compiler writers, and may change whimsically from one release of the compiler to the next.

On the other hand, a run-time environment may be designed to contain a fixed executive. Provided that the executive in question provides the services Ada needs, this works fine (see Standardization of the Run-Time Environment, below). Unfortunately, all existing executives that were not specifically designed for Ada do not provide a comfortable fit with Ada's requirements, especially its tasking requirements. In some applications, such as software development facilities, this poor fit is tolerable; for example, an entire Ada program may have to be regarded by the executive as a single task. In other applications, notably some embedded applications, the poor fit is not tolerable: the Ada program's interrupt-handling tasks may not be able to wait for two levels of scheduling bureaucracy to decide on whom to dispatch.

The Present Viewpoint

The viewpoint taken in this paper is that a certain computer has been picked to run some Ada software, but that the corresponding run-time environment has not been chosen. That is, the physical machine is specified but the virtual machine is not. The rest of this paper is a discussion of the choices left open by Ada to the specifiers of the run-time environment, with the purpose of appalling the reader with the amount of freedom Ada gives the specifiers of the run-time environment, and with the consequences of an inappropriate specification.

WHAT ADA REQUIRES

This section describes the requirements imposed by Ada on its run-time environments.

Operations

Ada requires that its run-time environments provide the usual kinds of operations (addition, comparison, assignment, indexing) on the usual kinds of values (integer, floating point, boolean, record, array). The usual sorts of branches (GOTO, IF, CASE, LOOP, subprogram call and return) are also required. Considerable care was taken in the design of Ada to specify these operations in such a way that as many as possible could be supported directly by the physical machine's instruction set (e.g., integer addition) and the rest could be implemented easily in software (e.g., CASE branch).

Ada also requires tasking operations, including the dynamic creation and destruction of tasks, and rendezvous between tasks (i.e., simultaneous synchronization and data interchange). Ada tasking operations are not like those of any other implementation language, and as a result, the question arose of whether these operations could be implemented with reasonable effort on real target machines. The result of considerable study is that there is a straightforward implementation of the Ada tasking operations on every general purpose computer.

Input/Output

Ada requires the ability to do sequential, direct-access, text, and low-level I/O. While the specification of Ada I/O is voluminous (e.g., what happens if when outputting a real value to a text file the specified width of the exponent field is not sufficient to hold the exponent's value and its sign?), and its implementation nauseating, no technical challenge is presented.

Exception Processing

Ada requires that certain errors be detected at run-time, such as an attempt to assign the value 11 to a variable that has been declared to hold only values between 1 and 10. Such a run-time error is called an exception, and the result of an exception is to transfer control to a user-specified exception handler.

It is not true that Ada requires its run-time environments to support exception processing. An Ada compiler can easily emit all the code that's necessary. Some run-time environments do support exception processing, in the hope of getting a speed or size improvement. How much difference is made by run-time environment support is hard to say -- it depends on the merit of the compiler (which can optimize away most exception checks) and on the style of the source code (sloppy code requires more exception checking than clean code).

Memory

The run-time environment must provide the ability to store and retrieve values. This requirement, although breathtakingly obvious, is sometimes the hardest to satisfy in practice. The difficulty is that on one hand, space or time inefficiencies in accessing data have a heavy impact on system efficiency (unlike, say, inefficiencies in exception handling), and on the other hand, the Ada memory model is hard to map onto some physical addressing structures.

This is not the place for details, but here's a brief discussion of the Ada memory model. Ada would like to store its data - mostly variables - in a tree structure, with the main program's data at the root, each task activation causing a new branch, and each procedure call extending an existing branch. Ada would like all this memory to be visible (i.e., addressable without too much overhead) at all times, but Ada will settle for having all the memory from the root to the current leaf visible at once. It is hard to map this kind of tree structure onto some hardware addressing structures, such as certain base register schemes.

That discussion concerned memory for data. Memory for code has similar problems, but they are less bothersome.

WHAT ADA PERMITS

This section discusses the facilities that a run-time environment may provide, over and above the requirements described in the previous section.

Interrupt Handling

In Ada, an interrupt is treated like an entry call from an invisible task of very high priority. Hence a run-time environment can satisfy the letter of the law by treating interrupts just like any other tasking operation. In some cases (the reactor overtemperature interrupt, say), the user community might appreciate some form of expedited dispatching of the interrupt handling task. That is, interrupt handling tasks may be given control directly upon receipt of the interrupt, and without an intervening enqueueing, scheduling decision, and dequeuing of the request; also, advantage may be taken of hardware register-saving capabilities.

Fancy Memory Management

Many software system designers regard the issues of memory management -- overlays, non-resident data, and garbage collection -- as major questions needing to be settled early in a system design. But the specification of

the Ada language is altogether silent on the topic¹ (with one exception: the programmer may explicitly decline garbage collection). That means that an implementation is free to provide any fancy memory management facilities it chooses, including none.

It is important to note that the provision of such facilities does not buy any indulgence from the rules of Ada. For example, an implementation that provides non-resident data, rolling in and out off mass storage, may not mumble in its specification anything like "It is the user's responsibility to insure that all referenced data is in main memory". On the other hand, an implementation may provide predefined procedures such as "Load_Task", provided that their use affects only timing and not legality.

Fancy I/O

Nothing prevents an implementation from providing I/O facilities in addition to those (rather elementary) capabilities required by Ada. For example, some applications might want formatted I/O facilities², asynchronous I/O, or DMS-style access functions. As long as the new functions are expressed as Ada packages (no statement labels as parameters), no difficulties arise.

Distributed Processing and Multiprocessing

The virtual machine on which an Ada program runs may have more than one processor. Physically, these processors may be anything from two CPUs sharing a memory, one CPU and several I/O controllers, up to the ARPANET.

As in the other cases, the Ada language definition leaves these issues up to the implementation. If distributed processing or multiprocessing facilities are crucial for success of the application, it is up to the specifier of the run-time environment to see that they are provided.

STANDARDIZATION OF THE RUN-TIME ENVIRONMENT

All readers of this paper are aware of the benefits of standardizing: reduced costs in procurements, shakedown, maintenance and training. The difficulty in standardizing on a single Ada run-time environment (i.e., a

¹ The facts that Ada does not deal with memory management and that system designers like to talk about memory management provide a chewy tidbit for those who would use Ada as a system design language.

² A member of the business community recently announced his determination that Ada cannot be used for business applications because of its lack of the floating dollar sign.

set of facilities) is also probably obvious: not every application wants the same facilities. Some want overlays and some don't; some want garbage collection and some don't; etc.

There is an analogy with interfaces between physical devices. Not every pair of devices can communicate over any single standard physical interface, but nevertheless, the RS232 standard has been a great blessing.

One run-time environment whose standardization would have tremendous significance is the KAPSE. See "The Kernal Ada Programming Support Environment (KAPSE) Interface Team (KIT)", in these proceedings.

CONCLUSION

The standardization of the Ada language has greatly reduced the dependence of application code on the executing system. But Valhalla has not quite been achieved. It is still true that the success of an application can depend on the optional facilities provided by the run-time environment in which the application will execute. It is therefore still the responsibility of the specifiers of the run-time environment to ensure that adequate functionality is available; else failure is guaranteed before the first pencil hits the coding pad.

Dr. Cross graduated from the University of Michigan in 1971 with a doctorate in mathematics. He taught mathematics and computer science for four years, and then spent one year as a visitor at New York University, working with the SETL project.

Since then, he has worked at Sperry Univac, Defense System Division. He is project leader of the Ada Generation project, which has produced a compiler for a subset of MIL-STD 1815. That project is now building a code-generator generator, which is a tool that facilitates the retargeting of Ada compilers by outputting Ada code generators.

THE ADA WORK CENTER
ITS FEATURES, CAPABILITIES AND DEVELOPMENTS

David Babcock
ROLM

BIOGRAPHY

David Babcock is presently Manager of Software Development at ROLM Corporation's Mil-Spec Computer Division. Prior to his 4 years with ROLM, he was the system's analyst and later Supervisor of Academic Applications at the computer center of California State University Northridge.

A biographic in Who's Who in the West, he has authored several articles in the computer field, co-authored a computer textbook and was, for 8½ years, the associate editor of Popular Computing magazine. He has also been an active member of ACM's technical group on Ada (Ada TEC) for the past 3 years.

ABSTRACT

Ada, the DoD sponsored high-order language, is having a major impact on software development. As important as Ada is, its goal of significantly improving programmer productivity can only be met by having a complete programming environment within which to develop software. Such an environment must address and support all aspects of a software project from definition, through design, coding and testing to final product. The "tools" of the environment must be complete, consistent, easy to use and yet powerful. They must aid, not hinder, the programmer in what he wants to accomplish while at the same time protecting him from making serious errors.

This paper will explore just such an all-inclusive environment - the ROLM Ada Work Center - its features, capabilities and development.

A CODE OF PRACTICE TO CONSTRAIN ADA

Dr. T. G. Swann

Airborne Software Division
Marconi Avionics Limited
Elstree Way
Borehamwood
Hertfordshire
England
01-953-2030

Tim Swann obtained his degree in Cybernetics in 1966, followed by a doctorate in Control Engineering. He has worked in electronics, microprogram and system software, and is now with Marconi Avionics as Assistant Chief Engineer of the Airborne Software Division.

SCOPE is more than just a code of practice for software development, it is the way we work in Marconi Avionics. Whatever project, whatever language, SCOPE enables us to maintain a "house-style" and a level of visibility and transportability across all our software.

But the scale and complexity of Ada introduce a new dimension. It will be difficult to ensure a similarity of style within a project team, let alone across the breadth of the company. And while Ada encourages good practices, it cannot prevent vices.

We must extend our code of practice. But to keep it simple we propose to integrate SCOPE with the Ada manuals to show not only what can be done, but what should be done, from choosing a language construct to setting up a project database.

For the price of a manual we achieve a standard.

BACKGROUND

In 35 years of computer programming the greatest single step forward has been the introduction of high level languages. Before, programming was an obscure art akin to crosswords and jigsaws. The tools of the trade were store locations and instruction codes. Binary and octal were essential knowledge. After, programming became a discipline. The tools became procedures, vectors and loops. Obscurity became a mark of failure rather than a source of pride.

At least that is the theory. In practice the step has been long and painful and is still far from complete.

It is certainly true that programmers can now take a higher level view of their task, and produce solutions far more elegant than machine code or assembler permit. This is an overwhelming benefit.

But as the problems of endless detail have been swept aside, new problems have taken their place.

With machine code it was possible to make elementary mistakes, such as multiplying code by data. With modern languages this is no longer possible. We are protected by data typing, data structures and so on. But this protection implies some degree of abstraction from the numerical details, indeed each statement is equivalent to several primitive operations. This leads to a "credibility gap" between the source language and its implementation.

When we write a high level statement we must consider:-

- i) What we think should happen.
- ii) What the language designers intended should happen
- iii) What the compiler manual says should happen
- iv) What the compiler writer thought should happen
- v) What the compiler actually does.

It is not unknown for all these to be different. The worst offender is mixed mode arithmetic. A typical compiler manual requires 12 pages to describe how scaling algorithms are implemented - and still leaves ambiguities.

For some programs we may be able to ignore these considerations but in the avionics business we cannot afford to. We require a high level of confidence in the detailed operation of the software beneath the facade of high level statements. No language system to date has given us this confidence.

The difficulties have been amplified by an increasing optimism that has led to systems of ever increasing size. Thus the declaration of a variable may no longer be "at the top of the page" but embedded in a 10 page block, perhaps written by another programmer. Indeed most of our recent projects have been sufficiently large to break the available tools.

Large systems have large teams many programs with hundreds of modules and thousands of variables. There are interfaces between modules, between programs and between hardware and software. Such systems bring whole new problem areas in complexity management.

It could be said that the software crisis of the 1960's was caused by encountering the problems of size without first having solved the problems of detail. Programmers could not, and cannot, deal with objects that are both detailed and large.

The answer to the crisis has been a systematic approach, a disciplined use of appropriate software tools. The tools are mainly high level languages and operating systems. The disciplines are project planning, role allocation, configuration control, test methods, structured programming and so on.

Alongside the new languages we have a whole new art of software management, embodied in Codes of Practice and reflected in company management structures. The success of a large project depends as much on this organisation and control as it does on the skills of the individual programmers.

INTRODUCTION OF ADA

We are about to embark on large high integrity systems using a new language, Ada.

Ada is in many ways a consensus language. It brings together a wide range of language features that have been tried and found useful. Examples are: records, arrays, loop control, CASE statement, separate compilation. It also introduces new features that until now have only been available in specialised forms. Examples are: tasking, generics, packages, strong typing. The result is a powerful language with features that assist all levels of software design, from bit level to system level.

But such power does not come cheaply. The wealth of features makes the language extremely complex. And this means a whole host of new problems.

One of the design goals of Ada was to produce a language that could be read as well as written. And to a large extent this has been achieved. If we look at the fragments of Ada text in the guide books we can see what the code is doing far more than would be possible in many other languages, especially those that are facetiously termed "write-only". But in the real world "readability" means much more than this.

A real system may have 100,000 lines of code. An individual writing this code can always stick to just a few well known structures, and fellow programmers soon get to know the style. But that is not the problem - it never has been. The problem comes:-

- When a system engineer needs to read the code: which of the 100,000 lines should be read first?
- When a specification change forces an update - years after the original team have left.
- When, dare I say it, software managers want to check things with their own eyes.

The "readability" of the local text is still an asset. But just as important is the visibility of the whole software structure - or rather the hardware and software structure, for we are talking here of embedded systems. And this visibility can only come from a systematic approach.

The source text of the programs is just one component of the whole development structure - an intermediate form between the customer's need and the delivered avionics system. And in this context the power of Ada is not something we can leave to the whims of individual programmers.

As an example consider the deceptively simple statement: $D=V*T$. In most current languages we would check that D, V and T were all of the same type, and deduce the effect of * from the context.. (This alone is more subtle than most programmers believe). But the flexibility of Ada allows the definition of new types of variable and new operations: to the extent of redefining the standard operators. This is termed "overloading". It is a powerful feature as it effectively allows us to extend the language.

The drawback is that to understand and check this Ada statement we need to know the definitions of not just D,V and T but also *. A clerical error could even redefine * by accident.

So the subtlety of having operators defined implicitly has been replaced by the complexity of defining them in the program. This in turn brings new possibilities for confusion.

For another example we can look at "typing". Strong typing is a great asset of Ada, but it too can lead to confusion. Consider the program of Fig.1. We can suppose that the package, of types, was written in 1985. In 1986, the type SMALL was used in procedure DOUBLE - using this standard package. But then in 1987, a post-delivery team had to modify the code. They introduced a new type which, by accident, was also called SMALL. The new type hid the old and procedure DOUBLE promptly failed. Luckily they thought to test it.

What was wrong? Everyone had done their best, with the best of intentions. But a name chosen by the second team clashed with another name chosen two years before.

This was not a program bug, it was a management error. The management had allowed programmers to pile up packages as if they were children's bricks. And they had failed to apply configuration control to the identifier names. The remedy is not clever programmers but better management. And this brings us to the real value of Ada.

Until now, nearly all our software has been written in CORAL 66, the U.K. Standard. The way we document our systems is described in a volume called AvP70, another U.K. Standard. The interpretation of AvP70 and the way we write CORAL, is described in our own Marconi Avionics Software Code of Practice:SCOPE. All the problems of variable names, data encapsulation, tasking, and so on, are discussed within SCOPE which recommends ways to tackle them - in CORAL.

But more than any other language, Ada impacts on the way programs are constructed. The jargon phrase is "programming in the large", and the key constructs are packages, abstract data types and tasking.

We could just extend our Code of Practice to include the Ada constructs: but there is a far better way. We propose to revise SCORE to change it from a CORAL base to an Ada base. And we believe that the power of Ada will give us a better manual than we had before.

With Ada we need no longer just make recommendations. We can actually demonstrate solutions, even to the extent of providing standard packages, and tools to manipulate them. Programmers so often find they are re-inventing the wheel. The high level features of Ada let us describe not just the details (which are always custom built), but the general shape - which may be the same from project to project. So instead of discussing parallelism in general, we can show standard solutions using Ada Tasking. Instead of recommending data encapsulation, we can write example code. The problems foretold above will be avoided, not by warning against them, but by preventing the circumstances in which they arise.

Of course there will be constraints and even forbidden constructs. Ada is a large consensus language and contains many things that are inappropriate to the avionics environment. And sadly there may even be restrictions due to run-time efficiency. There are plenty of PASCAL programs around the world which avoid even the use of procedures because of the overheads involved. And sadder still, there will be language features which simply do not work (or cannot be trusted to work, which is the same thing).

But the key to our approach is that no programmer works in isolation. The high level features of Ada are not "clever tricks" to be either encouraged or forbidden. They are software design aids to be used in appropriate ways by teams of software engineers; just as hardware engineers might use wire-wrap joints or rivets - each in the appropriate place. The trick is to know when and how to use them. And that is what SCOPE is all about.

CONCLUSION

When we first came to study Ada we were sceptical. Here was an immense new language breaking new ground in complexity for compiler writers and programmers alike.

But when we take a positive line we see that Ada is more than just a new language, it impacts on our whole approach to software development. We may still be sceptical - there are many problems to be solved - but we have the opportunity to integrate the best of software engineering practice with a language designed for the job. Now, we have two books a Code of Practice and a Language Manual. In future we will have just one: SCOPE: a Software Code of Practice, and more than that, a useable standard.

Package P is

...

type SMALL is range 0..100;

end P;

...

1985

type SMALL is range 0..50;

...

1987

procedure DOUBLE is

begin

use P;

X : SMALL := 30;

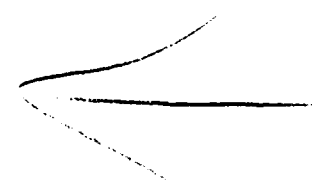
X := 2 * X

end;

call DOUBLE;

1986

Figure 1 A "Typing" problem



Use of Ada in System Design: A Case Study

Michael B. Patrick and Hal C. Ferguson
General Dynamics Data Systems Division Central Center

Since Ada has been adopted by the Department of Defense as the standard programming language for embedded computer systems, it is vitally important that government and industry personnel understand the consequence of using Ada in system development. A case study was recently completed in which Ada was used throughout the development of a large digital message switch. Prior to the start of system design, personnel were trained in the use of Ada and a methodology incorporating Ada compatible requirements and design techniques was developed. With judicious application of the methodology, a system design was produced. One major component of the system was programmed in Ada. In this paper, the case study effort is described. Examples of system design structures and Ada code are presented. Lessons learned and conclusions regarding the use of Ada are discussed.

Michael B. Patrick, an employee of General Dynamics Data Systems Division, has 20 years of experience in computer programming and software design, specializing in real time embedded computer systems. He served as project manager for the recently completed Ada Capability Study. In that capacity he was instrumental in selecting the team members, securing expert consultants, and coordinating contract activities.

Hal C. Ferguson, an employee of General Dynamics Data Systems Division, has extensive experience in both hardware and software design and the development of real time process control systems. He served as chief systems engineer on the Army Ada Capability Study contract. In that contract he lead the design team effort which produced a well-documented redesign of an AN/TYC-39 message switching system, using Ada throughout the development process.

ADA* TRAINING CONSIDERATIONS

Christine L. Braun

SoftTech, Inc.
460 Totten Pond Road
Waltham, MA 02154
617/890-6900

ABSTRACT

The government has instituted the Ada program with the objective of reducing its rapidly-increasing software development costs. Ada will do this by providing programmers with modern capabilities that have been demonstrated to promote more cost-effective software development. Clearly, the government's objective can be met only if programmers actually learn to use these capabilities effectively. This requires significant change from the way they are used to working, and poses a massive retraining requirement. SoftTech has been working with the U.S. Army to assess the training needs of various segments of the industry and government work forces, to identify training issues and effective techniques for addressing them, and to recommend a training approach. This effort has resulted in development of a complete recommended Ada curriculum. The curriculum provides training in the Ada language, the environment, and modern development methodologies. It consists of a set of modular building blocks that can be configured to meet varying individual or organizational needs, adapted to different organizations' practices, and packaged to meet scheduling needs. This approach answers many of the difficult questions that have been asked about Ada training, and presents a realistic roadmap to widespread industry competence in Ada.

THE DESIGN METHODS PROGRAM

The Army established the Ada Software Design Methods Formulation** program as the first step in developing a comprehensive Ada education program for industry and government. The overall effort involved three participating contractors. Two design contractors (General Dynamics and Control Data Corporation) redesigned and reprogrammed portions of two existing embedded computer systems, an air defense system and a message switch. The objectives of these contracts were:

*Ada is a trademark of the Department of Defense (Ada Joint Program Office).

**The work described in this paper was supported by the U.S. Army CECOM under Contract No. DAAK80-81-C-0187.

- to identify issues (including potential benefits and potential problems) with the use of Ada for design of embedded software
- to define and work with a set of methodologies supporting Ada design and development, recording results of experience with these methodologies
- to identify difficulties encountered in learning and using Ada.

SofTech's role as the Design Methods contractor was threefold, as illustrated in Figure 1. One major activity was aimed at the development of Ada case studies. SofTech met with each design contractor, and with Army representatives, in monthly Technical Interchange Meetings. These meetings involved detailed review and discussion of the contractors' designs, and of their observations concerning the Ada design process, at each project phase. Observations at these meetings were the starting point for development of the Ada case studies. Each case study addresses a major issue that arose during the effort, expanding on the initial observation to provide a more general discussion of the underlying issue, which might be a particularly effective use of Ada, a recurring Ada design problem, or a methodological issue. The case studies provide valuable material for inclusion in a training program; their primary intent is pedagogical. These materials are particularly important because they provide significant examples from real-time applications systems, a noted deficiency of most existing courses and texts. Some example case studies are:

- Task Structure for a Target Tracking System
- Use of Types to Describe Hardware Interface Requirements
- Stubbing and Readability
- Memory-mapped I/O in Ada
- Decoupling partly independent activities.

The case studies effort also identified a number of areas for future research in the effective use of Ada.

In parallel with the case studies work, SofTech conducted two surveys designed to provide additional input to the training program recommendations. The major survey was the work force survey, which involved respondents from six major DoD prime contractors and from the Army. This survey asked respondents about their backgrounds and current knowledge levels, and about their job duties, responsibilities, and outputs. This data was used to define generic job categories representative of the overall DoD Ada user community. These job categories were then used as a basis for identifying Ada training requirements, as described in subsequent sections.

The second survey was the industry training survey. This survey was designed to elicit information about training practices currently in use by industry. Respondents were asked questions dealing with issues such as:

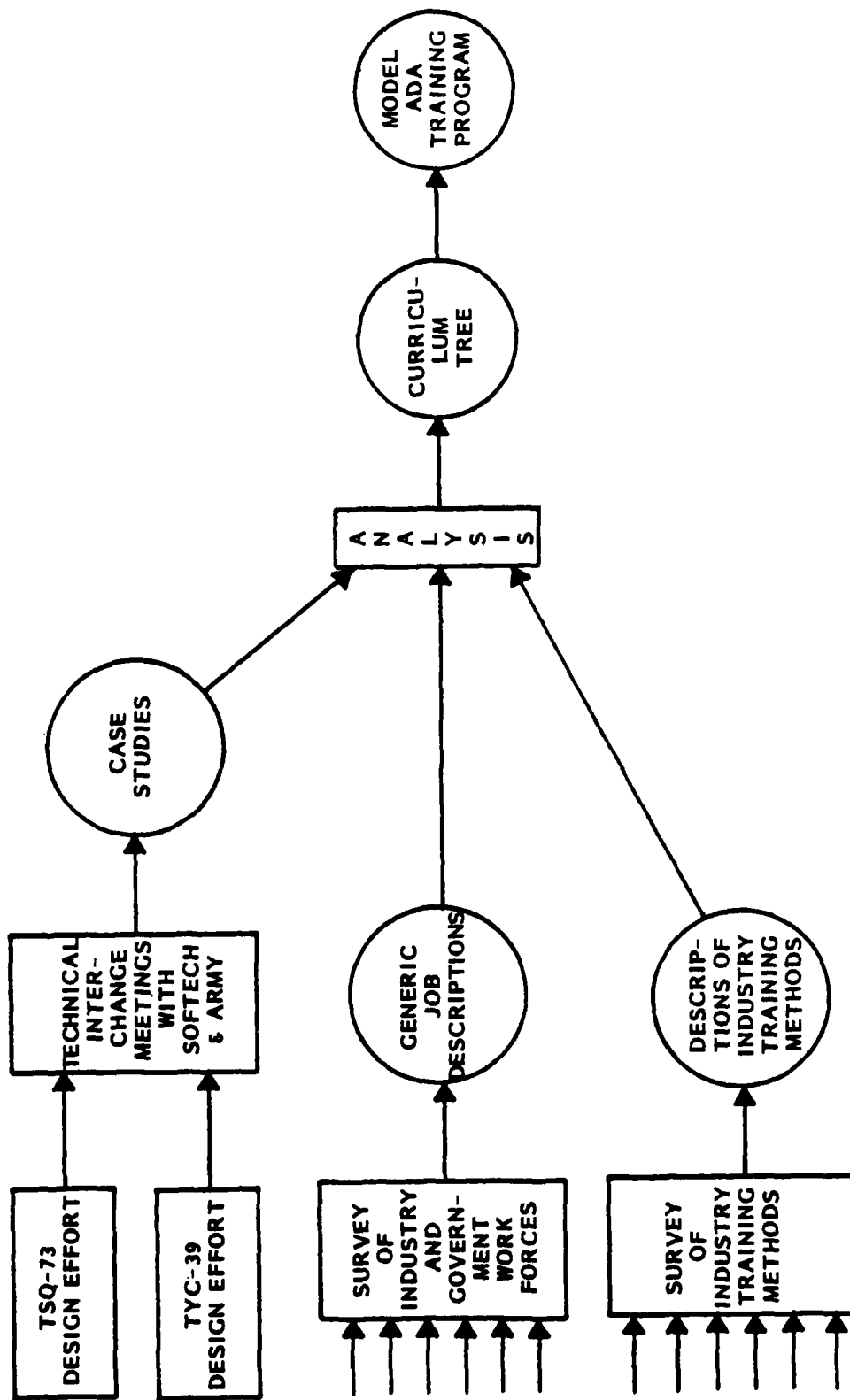


Figure 1. The Design Methods Program

- amount of time devoted to training employees
- ways of integrating training with on-the-job practice
- experience with and receptivity to various training media, e.g., videotape, computer-aided instruction
- current plans for Ada training.

This information was considered in the design of the recommended training program.

Results of these three parallel activities (the case studies and the two surveys) were used to define a curriculum tree, which identifies the background and Ada knowledge requirements of each generic job category. This information was then the basis for the specification of a model Ada training program that effectively meets identified industry requirements. These are described later in the paper.

THE GENERIC JOB CATEGORIES

As indicated above, a major emphasis of the work force survey was to determine what job the respondent performs. This is not simply his job title; it is what he actually does and what he actually produces. Extensive sets of questions asked respondents to classify potential activities and job outputs according to frequency and importance in their job. These results were subjected to computer analysis using MIT's Consistent System.¹ An Interpretive Nominal Clustering Technique² was used to cluster respondents according to similarity of job function. The first level of clustering resulted in five clusters (Figure 2). Representative titles were then assigned to each cluster. (It is important to note that the generic titles used for the clusters were assigned by SofTech based on the activity and output information for the cluster. They do not necessarily correspond to actual titles of any individual respondent.)



Figure 2. Initial Job Classification Clusters

Two categories in the initial clustering, the Development Engineering and System Integration Engineering groups, included large numbers of respondents. These groups were analyzed individually, resulting in five subclusters for Development Engineering and three for System Integration Engineering (Figure 3).

In analyzing the Ada requirements of the eleven categories that resulted from the clustering process, it appeared that further subdivision was desirable in two areas (Figure 4). The first was simply a breakdown of the Configuration Manager/Quality Assurance Engineering into those who perform in-depth technical review and analysis and those who function at a more general overview level. The second subdivision was in the Software Developer area. This is a significant subdivision that reflects one of the major choices underlying the training recommendations.

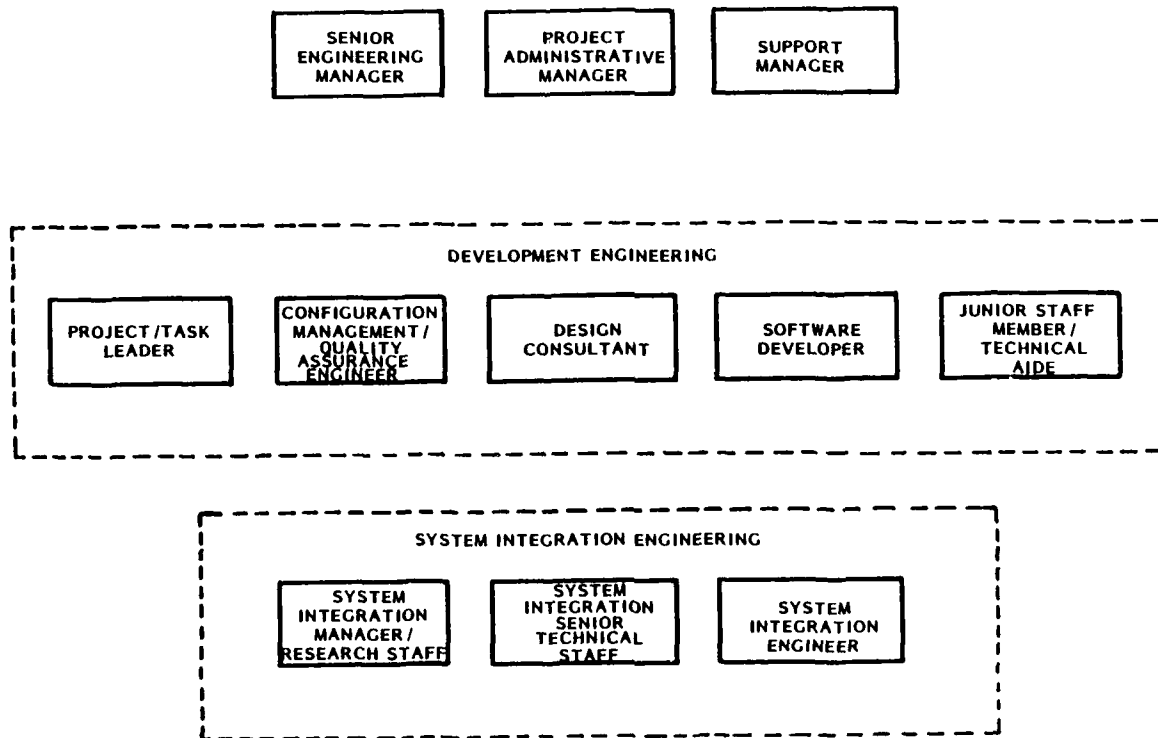


Figure 3. Secondary Breakdown of Development Engineering and System Integration Clusters

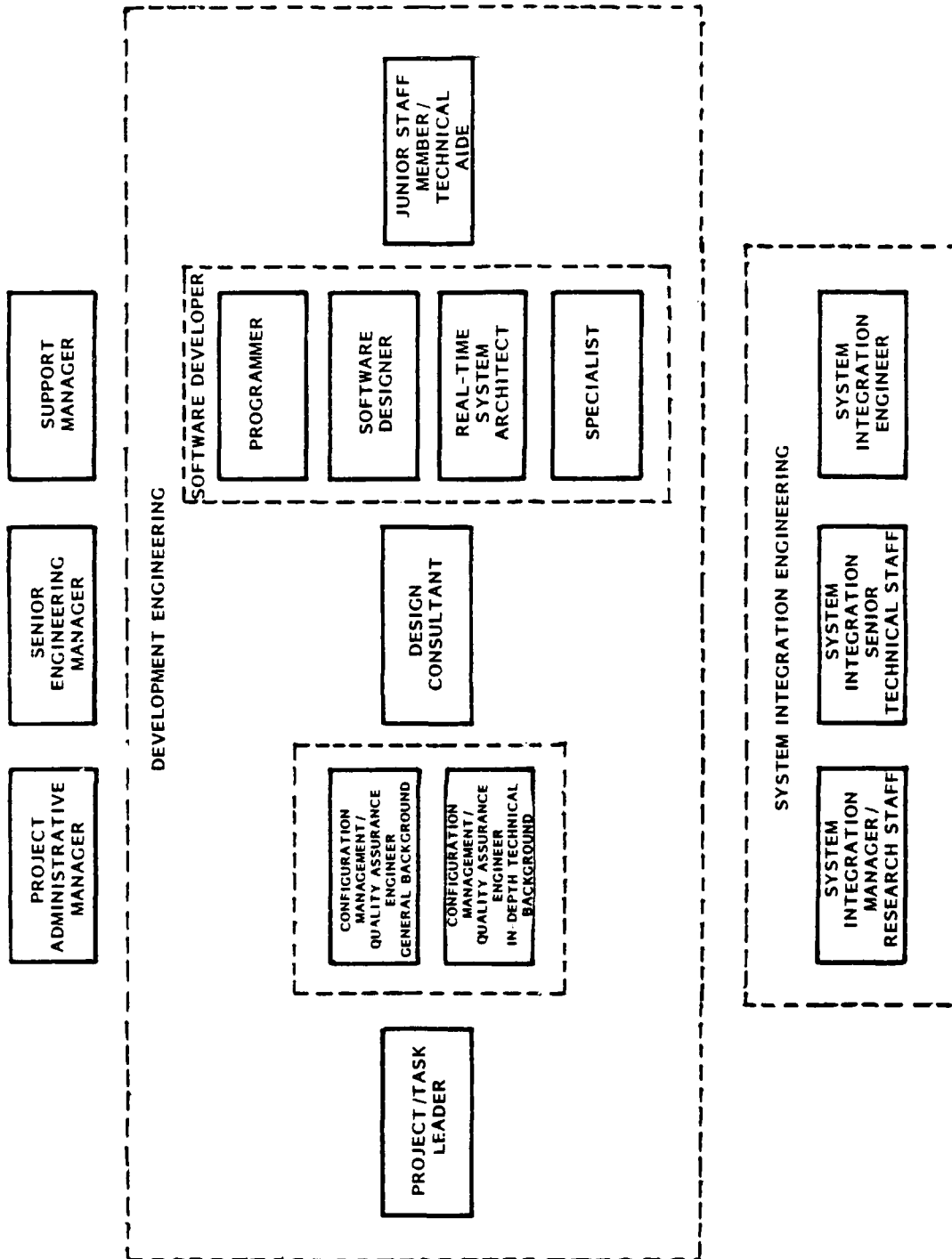


Figure 4. Final Job Classifications Including Breakdown of Software Developer Cluster

One program objective stated by the Army was to identify any new job categories that, while not apparent in the survey of the current work force, might be required in an Ada work force. This projection is the basis for the subdivision of the Software Developer category. In the survey, responses in this cluster did not support any further breakdown according to phase of development in which respondents participated or complexity of tasks performed. However, based on observations during technical interchange activities with the design contractors, it became apparent that a different view would be required for Ada. The complexity of certain aspects of the language is such that it is neither realistic nor practical to train all developers in full use of the language. Furthermore, such complete training is not necessary. The most complex tasks need be performed by only a relatively small subset of the development staff. Training individuals to only the necessary level of competence permits them to be productive more quickly and is much more cost-effective for industry and government.

The specific subcategories identified for Software Developers are as follows:

- Programmer - These individuals will implement program units according to specifications prepared by more senior designers. They will require training in roughly the "Pascal subset" of Ada.
- Software Designer - These individuals will have a good overall working knowledge of Ada, adequate to design sizable modules, but will not perform high-level real-time and concurrent systems design.
- Real-Time Architect - Real-time architects will design the high-level system architecture. They must be thoroughly versed in Ada's real-time features and in how to actually use them effectively in system design. They must also be capable of viewing the entire software/hardware system as a whole.
- Specialist - This group includes individuals who will specialize in a selected aspect of Ada. An example is numerical programming; only these specialists will require in-depth understanding of Ada's fixed and floating point features.

The clustering process ultimately resulted in 15 job categories, which formed the basis for the curriculum tree.

THE CURRICULUM TREE

The curriculum tree presents background and Ada viewpoint for each identified generic job category. Background is derived from the results of the work force survey. It includes summaries of years of experience, education, and present knowledge of language and software engineering concepts for the individuals in the category. The Ada viewpoint is the Ada knowledge considered necessary to perform the job functions identified for the category. (These job functions were inferred directly

from the responses to the work force survey; it should be remembered that the functions defined the clusters, not vice versa.) These recommendations are based on SofTech's observations during the technical interchange and case study activities. The Ada knowledge requirements are grouped in three categories:

- Environment - The Ada Programming Support Environment (APSE); i.e., the tools, file system, etc., provided in the Ada environment that is to be used for development.
- Language - The Ada language itself.
- Methodology - The design and development methodologies to be used in conjunction with Ada.

All three of the categories are essential to effective Ada training. The language was designed to be used in conjunction with modern software engineering methodologies; students must learn to use them when designing and developing Ada software. The APSE is necessary in order to actually develop Ada programs; it is most effectively taught in conjunction with programming courses. Thus, all three areas must be considered when planning the transition of today's work force to Ada.

The Ada knowledge requirements identified in the curriculum tree were then used to design a model Ada training curriculum.

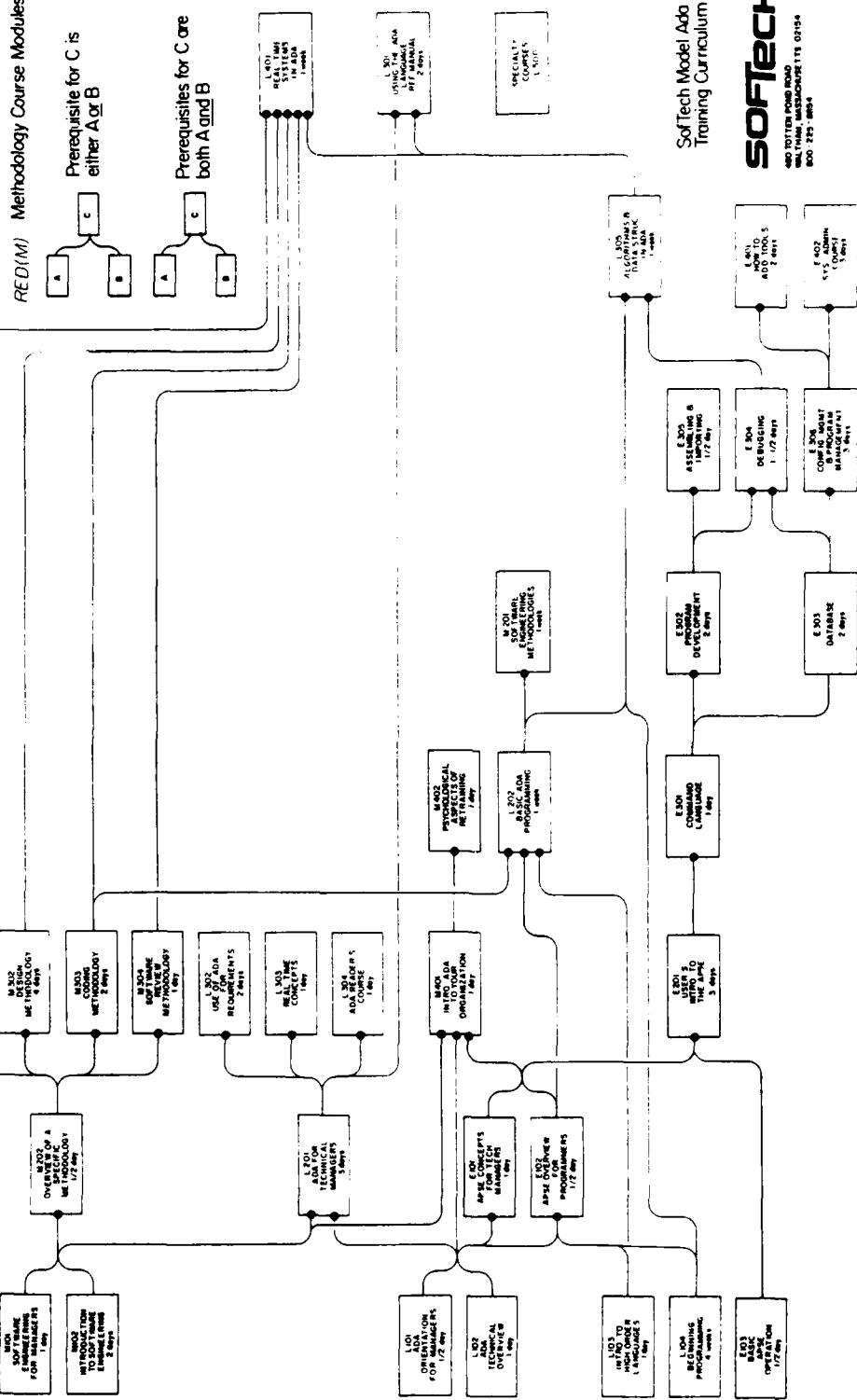
THE MODULAR CURRICULUM

The model Ada training curriculum was designed as a set of modular "building block" courses that can be configured in a variety of ways to meet the needs of any individual or organization. Figure 5 illustrates the course modules and their prerequisite relationships; Tables 1-3 give brief descriptions of the modules.

The modules are grouped into Environment, Language, and Methodology categories, corresponding to the Ada knowledge requirements discussed in the preceding section. The Environment and Methodology courses represent "replaceable components;" a specific organization's training plan would include modules that reflect organization-specific environment and methodology requirements. For example, an organization might use a particular MIL-STD that would be taught in the methodology modules, or might require use of particular design techniques such as Structured Design, HIPO Charts, etc., or of a particular Program Design Language. In the environment area, an organization would select a particular environment such as the Army's Ada Language System³ or the Air Force's Ada Integrated Environment.⁴ They might then customize the environment by developing new tools, command procedures, etc. The best training plan for the organization would be one that included specific training in

SofTech Model Ada Training Curriculum

Ada Programming Support Environment Course Modules
 Ada Language Course Modules
 Methodology Course Modules



SofTech Model Ada Training Curriculum
SOFTECH
 400 TOTTER FORD ROAD
 WALTHAM, MASSACHUSETTS 02154
 603 252-8864

Figure 5. Modular Curriculum

TABLE 1. ADA PROGRAMMING SUPPORT ENVIRONMENT CURRICULUM MODULES

NO.	TITLE	DESCRIPTION	DURATION
E101	APSE Concepts for Technical Managers	broad overview of APSE emphasizing how it supports s/w life cycle	1 day
E102	APSE Overview for Programmers	broad overview of APSE for software developers	1/2 day
E103	Basic APSE Operation	introduction to APSE concepts, basic editing, etc., for people who will not be real users	1/2 day
E201	User's Introduction to the APSE	basic use of the APSE database, file system, command language; tool overview	3 days
E301	Command Language	command language, substitutors, I/O redirections	1 day
E302	Program Development	Compiler, linker, exporter, loader	2 days
E303	Database	files, directories, attributes, associations, access control, node sharing, program libraries, etc.	2 days
E304	Debugging	debugger, timing analyzer, frequency analyzer	1 1/2 days
E305	Assembling and Importing	assembly language, importer	1/2 day
E306	Configuration Management and Program Management	tools to support CM and PM, example tools one might build	3 days
E401	How to Add Tools	programming with the command language, KAPSE tool interfaces, examples of useful tools	2 days
E402	System Administrator's Course	user authorization and protection, installation, backup, system support	3 days

TABLE 2. ADA LANGUAGE CURRICULUM MODULES

NO.	TITLE	DESCRIPTION	DURATION
L101	Ada Orientation for Managers	overview of development and features of Ada	1/2 day
L102	Ada Technical Overview	overview of language-introduction to language features in more depth than above	1 day
L103	Introduction to High Order Languages	key HOL concepts for assembly language programmers	1 day
L104	Beginning Programming	introduction to computer programming in an Ada context	4 weeks
L201	Ada for Technical Managers	use of Ada for good systems design; packages, types, generics, portability features, etc.	3 days
L202	Basic Ada Programming	essentially the Pascal subset	1 week
L301	Using the Ada Language Reference Manual	how to use the manual effectively as a reference	2 days
L302	Use of Ada for Requirements	Ada as a requirements definition language	2 days
L303	Real Time Concepts	real time design concepts for technical managers	1 day
L304	Ada Reader's Course	reading an Ada design or program for its key points and overall structure	1 day
L305	Algorithms and Data Structures in Ada	packages, access types, private types, discriminated records, generics, basic tasking, basic algorithms	1 week
L401	Real Time Systems in Ada	everything about tasking, external interfaces, low-level features	1 week
L500	Specialty Courses	numerical analysis, hardware diagnostics, man/machine interface database management, etc.	varying

TABLE 3. ADA LANGUAGE CURRICULUM MODULES

NO.	TITLE	DESCRIPTION	DURATION
M101	Software Engineering for Managers	software life-cycle, top-down concepts, documentation, testing	1 day
M102	Introduction to Software Engineering	life-cycle, top-down concepts, overview of various methodologies	2 days
M201	Software Engineering Methodologies	thorough coverage of major methodologies	1 week
M202	Overview of a Specific Methodology	overview of an organization's selected life-cycle methodology	1/2 day
M301	Requirements Methodology	requirements definition techniques and methodology	1 week
M302	Design Methodology	how to do design, with required methodology	4 days
M303	Coding Methodology	structured programming, coding standards, programming style, etc.	2 days
M304	Software Review Methodology	Walkthroughs, code reading	1 day
M401	Introducing Ada to Your Organization	how to use the recommended curriculum to meet specific needs	1 day
M402	Psychological Aspects of Retraining	techniques for overcoming resistance to change	1 day

these areas. The curriculum can also be customized by developing exercises and examples that are representative of the kinds of applications actually performed by the organization, e.g., signal processing, communications, etc.

The modular curriculum is thus not a single, fixed curriculum. Rather, it is a framework that can be used to design a training plan to meet any stated requirement. Its flexibility is its major asset. It provides a basis for planning an individual or corporate approach to the Ada training problem.

APPROACHING THE TRAINING PROBLEM

The modular curriculum is a starting point for an Ada training program, but only a starting point. How can an organization develop an Ada training program designed to meet its specific requirements? First, the organization must analyze its requirements. It must assess:

- number of people who need training
- skill level required - how many at each level
- required time frame
- sources of people - are they there now, or must they be hired
- current skill levels
- tailoring required
 - specific methodology
 - specific environment
 - application-specific examples
 - specialist courses
- scheduling/cost constraints
- additional training aids desired (e.g., video, texts)

The organization must then arrange to develop/adapt the training materials as required, either on its own or via outside contractor support. Instructor resource requirements must be identified and courses must be scheduled.

Courses can most effectively be scheduled in a top-down manner, training managers before staff and designers before implementors. This has two advantages. First, prior training of managers ensures that they will understand and support the transition to Ada before leading their staffs in that transition. Management commitment to the concepts behind Ada and understanding of potential problems is essential to its positive reception by employees. The second advantage is that this approach leads to a "bootstrap" process of introducing Ada. A core of well-motivated top-level designers is trained, and begins the design process. While lower-level developers are in training, they have the support and direction of the initial group, and can work with them to gain hands-on, on-the-job experience to augment their training. The initial group provides an Ada nucleus on which to build.

Finally, once courses are given much of the cycle repeats itself. Needs continue to evolve, new employees arrive to be trained or existing employees advance to new responsibilities, and materials require improvement. Ada training must be an ongoing process.

SUMMARY

The Ada Software Design Methods Formulation program was a comprehensive program designed to gain an in-depth understanding of Ada training requirements and specify a curriculum that meets those requirements. Based on Ada case studies, an industry training survey, and a work force survey, a model Ada curriculum was developed.

The curriculum is modular, consisting of 35 modules comprising 89 class-days of training. This modular approach provides maximum flexibility to individual and organizational needs. It is also cost-effective; it minimizes investment in employee time, as people can be productive quickly. The curriculum has been designed to support customization to specific organization methodologies and training environments.

This curriculum, and the supporting study that explains its derivation, provides a strong foundation for training program design. Its modularity and adaptability offer a solution to many of the problems confronting today's Ada course designers. These designers, trying to develop courses that will be all things to all people, have been frustrated by the many unanswered questions that still exist about Ada (e.g., what is the best PDL?) and by the difficulty of teaching the entire language to all students. This framework lets curriculum developers produce useful materials without answering all of these questions. It provides a basis for gaining maximum industry-wide benefit from the work of all individuals involved in development of Ada training materials, thus supporting a cost-effective transition to widespread industry competence in Ada.

ACKNOWLEDGEMENT

I would like to thank the other members of the SofTech project team -- Christine Ausnit, John Doyle, Sterling Eanes, John Goodenough, Nico Lomuto, Karen Sather, and Putnam Texel. All of these people made important contributions to the work described in this paper.

REFERENCES

1. J.C. Klensin, editor, The Consistent System: Handbook of Programs and Data, Library of Architecture and Planning, MIT, February 1982.
2. R. Muller, "Data Organization: The Integration of Database Management, Data Analysis, and Software Technology Applied to the National Crime Survey," Ph.D. Dissertation, Massachusetts Institute of Technology, 1982.
3. Ada Language System Specification, U.S. Army CECOM, Ft. Monmouth, NJ, Contract No. DAAK80-80-C-0507, August 1982.
4. System Specification for Ada Integration Environment, Type A, Rome Air Development Center, Rome, NY, Contract No. F30602-80-C-0291.

THE AUTHOR

Christine L. Braun is Manager of Quality Assurance at SofTech, Inc. Her department is responsible for developing and applying procedures and methodologies leading to the production of quality software and for independent quality review of SofTech projects and of all proposals and deliverable items. She is also responsible for internal training programs and seminars and for developing training programs and course products for SofTech clients. She participates in SofTech's Ada program as leader of the Senior Technical Review Board, a group of senior managers responsible for technical review of the project.

Braun is a specialist in software engineering with many years of experience in the design, development, and program management of software systems. She has published several technical papers and has taught college level courses in programming languages, compiler construction, and system programming. She is an active member of IEEE Computer Society's Software Engineering Standards subcommittee and of the Air Force standards organizations.

She received an AB in mathematics in 1970 from Brown University and a MS in computer science in 1971 from the University of Toronto.

FINED

PAID

8