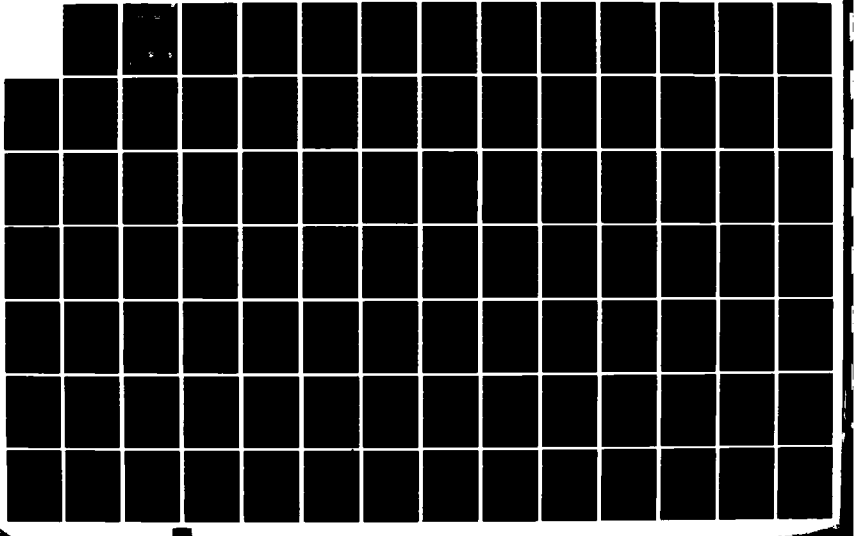
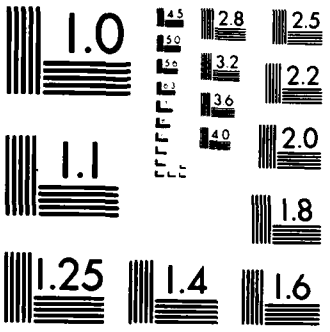
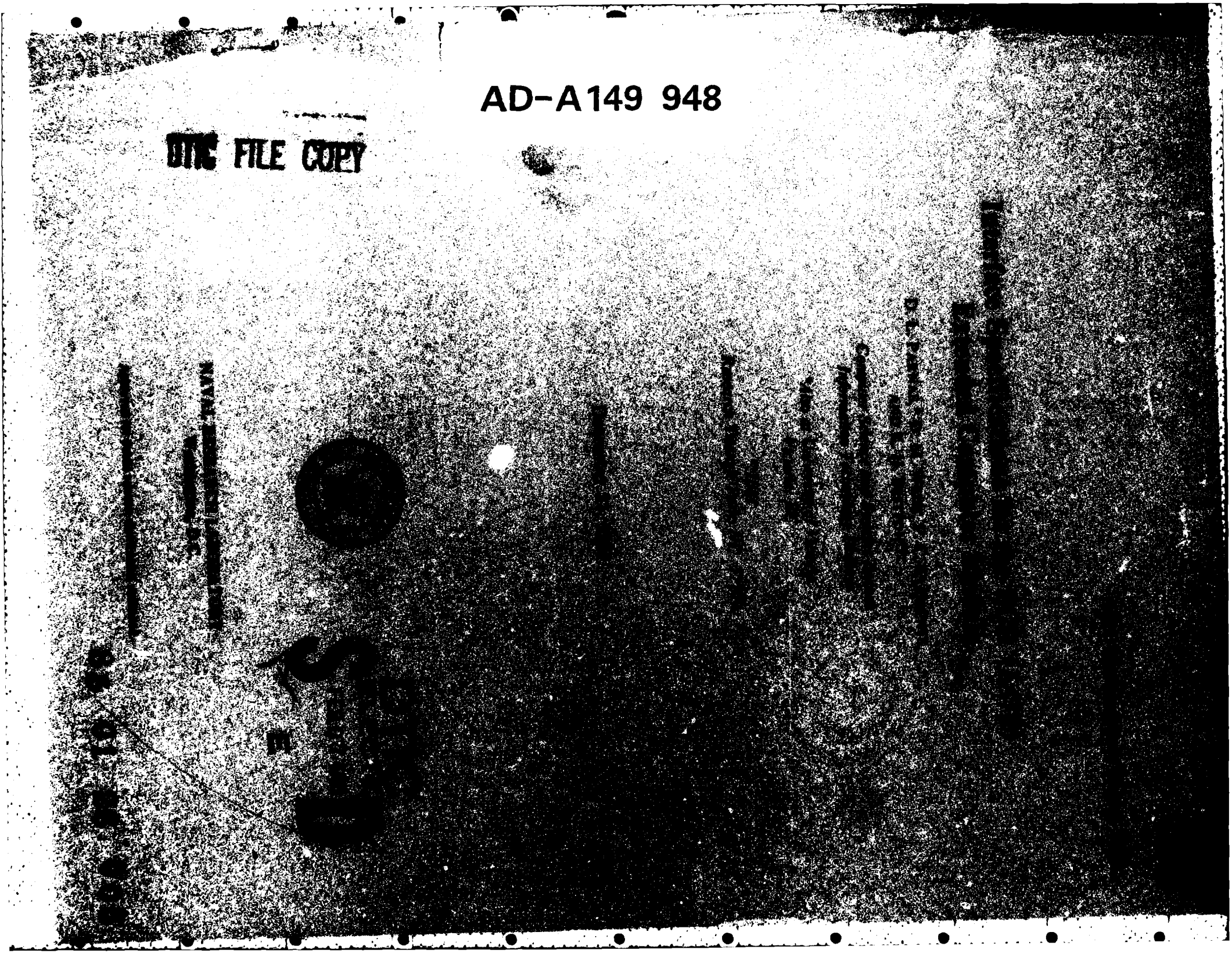
AD-A149 948 INTERFACE SPECIFICATIONS FOR SCR (SOFTWARE COST REDUCTION) (A-7E) EXTENDED COMPUTER MODULE REVISED(U) NAVAL RESEARCH LAB WASHINGTON DC D L PARNAS ET AL. 1/2

UNCLASSIFIED 31 DEC 84 NRL-MR-5502-REV F/G 9/2 NL







SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 5502	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b OFFICE SYMBOL (If applicable) Code 7590	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION Naval Electronic Systems Command 8b OFFICE SYMBOL (If applicable) Code 613	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Washington, DC 20360	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62712N; PROJECT NO.: ; TASK NO.: SF212-43601; WORK UNIT ACCESSION NO: 75-0106-0-4

11 TITLE (Include Security Classification)
Interface Specifications for the SCR (A-7E) Extended Computer Module

12 PERSONAL AUTHOR(S)
Parnas, D.L.,* Weiss, D.M., Clements, P.C., and Britton, K.H.**

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM _____ TO _____	1984 December 31	129

16 SUPPLEMENTARY NOTATION
*Also at University of Victoria, Victoria, BC
**IBM, Research Triangle Park, NC 27709 | This is an updated version of NRL Memorandum Report 4843.

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Abstract interfaces; Information hiding; Avionics software; Modular decomposition
(Continues)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This document describes the programmer interface to a computing machine partially implemented in software. The Extended Computer is part of NRL's Software Cost Reduction (SCR) project, to demonstrate the feasibility of applying advanced software engineering techniques to complex real-time systems in order to simplify maintenance. The Extended Computer allows code portability among avionics computers by providing extensible addressing, uniform i/o and data access, representation-independent data types, uniform event signalling, a standard subprogram invocation mechanism, and parallel process capability. The purpose of the Extended Computer is to allow the remainder of the software to remain unchanged when the host computer is changed or replaced.

This report describes the modular structure of the Extended Computer, and contains the abstract interface specifications for all the facilities provided to users. It serves as development and maintenance documentation for the SCR software design, and is also intended as a model for other people interested in applying the abstract interface approach on other software projects.

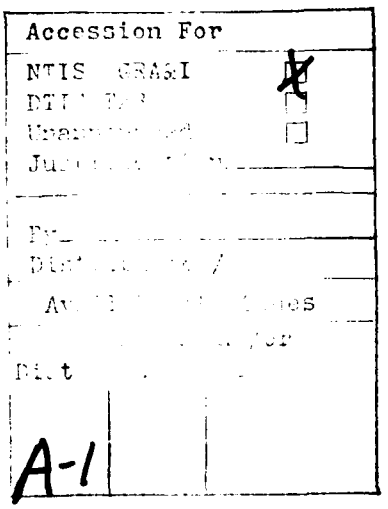
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Paul C. Clements	22b TELEPHONE (Include Area Code) (202) 767-3477	22c OFFICE SYMBOL Code 7595

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

18. **SUBJECT TERMS (Continued)**

Modules
Real-time systems
Software engineering
Software maintenance
Software specifications



CONTENTS

INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) EXTENDED COMPUTER MODULE

INTRODUCTION

The Extended Computer (EC) is a computing machine partially implemented in software. It was designed as part of the Software Cost Reduction (SCR) project at the Naval Research Laboratory. The design goals are 1) code portability, 2) abstraction from computer hardware idiosyncracies, 3) more easily understood code, and 4) sharing of solutions to common machine dependent coding problems. The Extended Computer is designed to be efficiently implemented on avionics computers such as the IBM 4PI TC-2. The instruction set allows straightforward, efficient code generation using a macroprocessor.

The Extended Computer has the following features:

1) Extensible addressing: There is no syntactic limit to the amount of memory that can be addressed. The actual memory size is a parameter that is set at system-generation time.

2) Uniform data access: Hardware addressing techniques, such as use of base and link registers, are hidden from programmers.

3) Uniform subprogram access: All subroutines are invoked in a uniform manner; linkage mechanisms are hidden from users.

4) Uniform input/output: Variations in I/O operations are hidden. All input (output) data items are read (written) using the same statements.

5) Uniform event signalling: The difference between hardware interrupts and software-detected events is hidden. All interrupt handling is hidden.

6) Data types: Data types representing reals, bitstrings, and time intervals are provided together with the necessary conversion functions. Data representations are hidden. Hardware arithmetic and bitstring operations are hidden.

7) Parallel processes: Programs can be written as a set of cooperating sequential processes. The number of hardware processors and their scheduling are hidden.

8) State control: Computer state transitions among various states (including off, operating, and failed) are signalled to the user programs. The mechanics of state transitions are hidden.

9) Built-in test: Diagnostic programs to test the integrity of memory and the correct operation of the hardware are built-in. The tests and evaluation criteria are hidden.

Manuscript approved October 23, 1984.

10) Exception handling: Both a development version, with extensive checks for programming errors, and a production version are available. Programs that cause no undesired events [WUER76] on the development version will compute the same values on both versions. The version can be selected at system-generation time.

The Extended Computer has been designed to hide the interface characteristics of a computer with capabilities similar to those of the IBM 4PI/TC-2. Were the present A-7 computer to be replaced by one with different capabilities, we would shift some responsibilities to/from other parts of the software. For example, if the new computer used an external device for timing, the implementation of the timeint data type would become a part of the device interface modules. Or, if the new computer included a capability for angle implementation, the machine-independent implementation of an angle data type would be replaced by a machine-dependent module that was part of the EC, but with the same interface as the present angle data type. Of course, under such unlikely circumstances, the appropriate documentation (such as [REQ], [MG], and [AT], as well as this document) would be changed to remain consistent with the new hardware. If the EC design were to be used in an application that did not require all of its capabilities, a compatible subset could be used.

We recommend that this procedure be followed by anyone maintaining this system, and by those who are designing other systems using a similar approach.

This document specifies the user interface to the Extended Computer. The contents, form, and notation are in accordance with the guidelines given in [SO], with the following addition.

Events signalled by incrementing a semaphore: The EC signals all events by incrementing semaphores. The semaphores and the events they represent are listed in this section. The semaphores are built-in (users need not declare them), and are given an initial value of zero at system generation time.

EC.DATA.1 INTRODUCTION

EC.DATA.1.1 ENTITIES

The Extended Computer provides literals, constants, and variables. We refer to these as entities. Literals are values appearing in programs. Constants have names and values; run-time programs can read the values but not change them. Variables have names and values; the values can be read or written by run-time programs. A register is a variable with a faster access time than other variables. There is one register for each process (see EC.PAR). All constants and variables other than registers may be accessed from any process of the program. It is possible to declare arrays of variables or constants. An element of an array may be used as an individually declared entity of the same type. Users are given the facility for providing information to the Extended Computer about the relative speeds with which declared entities should be accessed.

EC.DATA.1.2 TYPES

Types are classes of entities. The Extended Computer provides a hierarchy of types; an entity is either numeric, bitstring or pointer. Numeric types are characterized by range and resolution. Bitstring types are characterized by length. The value of a pointer is another entity. Pointer types are characterized by the type of entity to which members of the pointer type may refer. The value of a characteristic for an entity is called an attribute.

For a particular numeric type, every numeric value between the upper bound and lower bound (inclusive) has a representative in its type. Any representative will differ from its nearest neighbors by no more than the resolution of the type, and no numeric value will differ in value from its representative by more than half the resolution.

For numeric types, users may require that the representatives include exact multiples of the resolution between the lower and upper bounds, inclusively.

A type class is a type that contains entities with different behavior. A specific type (also called spectype) is a subclass of a type class in which all variables have identical behavior; i.e., they can take on the same set of values and one may perform the same operations on them with the same results. The behavior of the program will not change if two variables of the same specific type are interchanged throughout the program.

For each type class, there are any number of specific types. These either have fixed attributes, or attributes that may vary at run-time.

Figure 1 provides an overview of the EC data types by showing the Extended Computer's type classes and specific types. Lines connect a type with its sub-types. The terminal nodes represent specific types, of which entities may be declared. Not all of the types in the table are currently implemented; to see which ones, refer to Appendix 4.

The Extended Computer provides two numeric type classes illustrated in Figure 1, but not described in this chapter. They are semaphores and timers, whose operations are described in EC.SMPH and EC.TIMER, respectively. The Extended Computer also provides the program type class. This is described in EC.PGM.2.

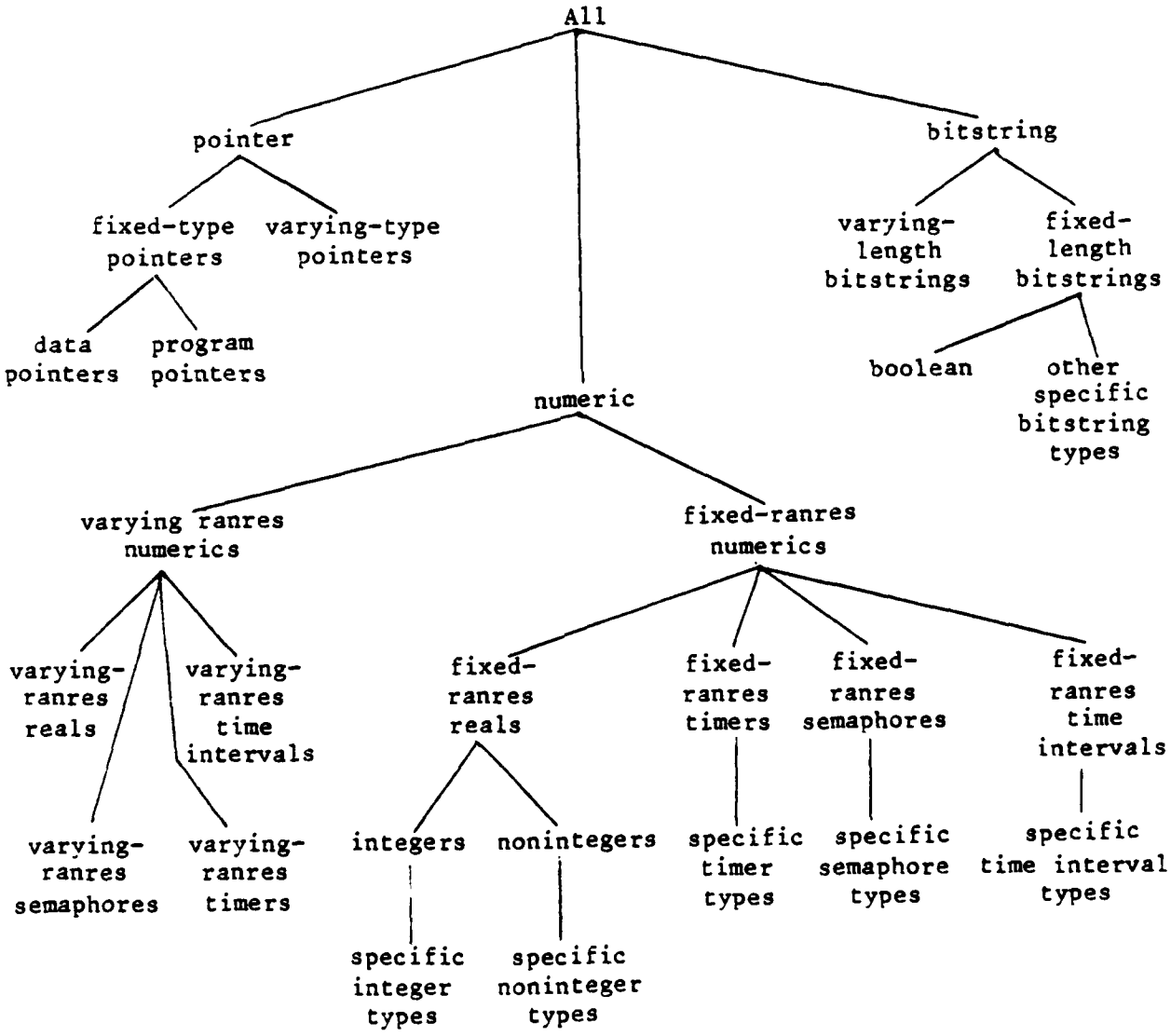


Figure 1

EC.DATA.1.3 SCALAR LITERALS

A scalar literal belongs to exactly one of the type classes bitstring, real, or timeint, and may belong to more than one specific type of its type class. Formats for writing literals of these type classes are specified in the type definitions for these type classes in EC.DATA.3.

Numeric literals will be represented with at least the precision implied by their written representation.

EC.DATA.1.4 REGISTERS

A single register is provided for each process. Run-time operations using the register are likely to be faster than operations on other variables. (A system-generation-time program cannot use the register.) A register is a variable with varying type class and varying attributes; each operation that uses a register must include information sufficient to determine a type class and the appropriate attributes.

A process cannot access the register of another process.

The contents of a register may be changed by a) using the register as a !!destination!!, or b) performing an operation without specifying that the register contents be preserved. Each run-time access program defined in this chapter may appear with or without a suffix "-SAVE" (e.g. +MINUS+ or +MINUS-SAVE+). Use of the suffix specifies that the contents of the register will be preserved by the operation. Omission of the suffix specifies that the contents of the register shall be the same after execution of the program as immediately before.

Table EC.DATA.b shows how the value of a register is affected by an operation. Value undefined indicates that the value contained in the register is unspecified. If a program reads a register when its value is undefined, the results will be unpredictable.

Table EC.DATA.b: Effects of Operations on Register Contents

Register use	Suffix	Effect of the operation on Register
read only	none	value undefined
read only	-SAVE	value not changed
written or read and written	none	new value produced by operation
written or read and written	-SAVE	undesired event
not referenced	none	value undefined
not referenced	-SAVE	value not changed

EC.DATA.2 INTERFACE OVERVIEW

EC.DATA.2.0 DECLARATION AND RANKING OF DATA SETS

The EC requires users to assign entities to data sets. The user is then allowed to specify a partial ordering on the data sets to determine speed of access to the sets' members. The rankings apply to sections of code. Significant performance improvements are possible if the entities used in a section of code belong to a data set that is highly ranked.

Program name	Parm type	Parm info	Undesired events
++DCL_DATA_SET++	p1:name;I	name of data set	%%name in use%%
++RANK_DATA_SET++	p1:data-set-reln;I		None.

Effects

++DCL_DATA_SET++ Declares p1 to be the name of a data set, and allows that name to be used as p5 of ++DCL_ENTITY++ and/or p6 of ++DCL ARRAY++.

++RANK_DATA_SET++ Defines a partial ordering on all data sets; if (A,B) is in the relation given by p1, then data set A has a higher rank than data set B. Data sets not named in p1 have an arbitrary rank lower than any set named in p1.

The ranking applies until the next textual occurrence of ++RANK_DATA_SET++.

Access to entities and arrays in a data set will be made not slower than access to members of a lower ranked data set.

EC.DATA.2.1 DECLARATION OF SPECIFIC TYPES

All specific types must be declared and given a name. Numeric types are characterized by !!range!! and !!resolution!!, bitstring types by length. Pointer types are characterized by the name of a previously-declared specific type. The type declaration must indicate whether or not these attributes can vary at run-time. The EC allows users to choose among different versions of the implementation for each type; each version is especially efficient for performing certain operations. The versions, and the advantages and disadvantages of each, are specified in Appendix 6.

Program name	Parm type	Parm info	Undesired events
++DCL_TYPE++	p1:name;I	name of new type	%%name in use%%
	p2:typeclass;I	containing type class	%%inappropriate attributes%%
	p3:attribute;I	attributes of type	%%length too great%%
	p4:binding;I	Can attributes vary at run-time?	%%range too great%%
	p5:version;I	implementation version	%%ranres too great%%
			%%res too fine%%
			%%unknown operand in attribute%%
			%%undeclared spectype%%

Program Effects

A specific type that is a member of type class p2 and has binding p4 and implementation version p5 is declared to have identifier p1. If p4=FIX, then all entities and arrays of this specific type will have the attributes given by p3. If p4=VARY, then p3 gives the !!hardest attributes!! that any entity or array of this spectype will ever assume. If p5 is not a version associated with the given type, as specified in Appendix 6, then the EC implementation will use an appropriate version of its own choosing. The identifier can be used as the spectype (p2) parameter in calls on ++DCL_ENTITY++ and ++DCL_ARRAY++ in programs that follow the declaration.

CRF 152 154 168 181 205 209 221 262

EC.DATA.2.2 DATA DECLARATIONS

EC.DATA.2.2.1 DECLARATION OF VARIABLES AND CONSTANTS

Variables and constants must be declared before they are used. The declaration must specify the name of the new entity, a previously declared specific type (one of the terminal nodes on the tree of figure 1), whether the entity is a constant or a variable, and an initial value.

Program name	Parm type	Parm info	Undesired events
++DCL_ENTITY++	p1:name;I	entity name	%%name in use%%
	p2:spectype;I	entity's specific type	%%undeclared spectype%%
	p3:convar;I	when writeable?	
	p4:constant or literal whose value is in domain of type named by p2;I	initial value	%%unknown initial value%% %%varying constant%% %%wrong init value type%% %%loadcon too big%% %%literal or ascon too big%%
	p5:data_set;I	data set name	

Program Effects

An entity with identifier p1, spectype p2, and initial value p4 is declared. If p3=VAR, the entity may be used as a !!destination!! in a subsequent operation. The entities that have been declared may be used as operands in the programs that follow. The entity is assigned to data set p5.

CRF 154 209 266

EC.DATA.2.2.2 DECLARATION OF ARRAYS

Program name	Parm type	Parm info	Undesired events
++DCL_ARRAY++	p1:name;I	array name	as for ++DCL_ENTITY++
	p2:spectype;I	element type	plus:
	p3:convar;I	when writeable?	
	p4:array-init;I	initial value	%%wrong init value size%%
	p5:indexset;I	array indices	
	p6:data_set;I	data set name	%%illegal index set%%

Program Effects

A one-dimensional array with identifier p1, spectype p2, initial value p4, and index set p5 is declared. If p3=VAR, the elements of the array may be used as !!destination!!s in subsequent operations. The array is declared to belong to data set p5. Elements of the array can be used wherever an entity of the same specific type could be used. An array may also be a parameter to a user-defined program.

CRF 154 186 266

EC.DATA.2.3 ACCESS SPEED RANKING OF DATA

The Extended Computer can implement a "not-slower-than" relation between any two variables, constants, or arrays.

Program name	Parm type	Parm info	Undesired events
++RANK_DATA++	pl:rank-data-relation;I		%%undeclared operand%% %%inconsistent data ranking%%

Program effects

Let A and B be previously declared arrays, constants, or variables. If the rank-data-relation for this invocation of the program includes (A,B), then using A (or an element of A if A is an array) as an operand in an EC program will take no longer than using B (or an element of B if B is an array) in the same program, if both are in the same data set. The rank relation for all EC data objects is composed of the rank-data-relations given in each invocation of this program. The relation is transitive, antireflexive, and antisymmetric.

EC.DATA.2.4 OPERAND DESCRIPTIONS

EC.DATA.2.4.1 INDIVIDUAL PARAMETERS

Table EC.DATA.c summarizes the description of operands for EC run-time access programs and user-defined programs. Brackets shown are required.

Table EC.DATA.c INDIVIDUAL PARAMETER SPECIFICATION

Nature of Parameter	Form of Parameter
Literal	Literal value
Scalar constant or variable without qualifier(s)	Name of entity
Qualified parameter	⟨ Name of entity or array, qualifier-list⟩ where qualifier-list ::= qualifier OR qualifier-list, qualifier
Entity referred to by a pointer	⟨ DEREF , entity⟩

The latter two forms are discussed below.

EC.DATA.2.4.1.1 QUALIFIED PARAMETERS

Variable with varying attributes: The attributes must be specified. A qualifier is given that is the name of a previously-declared specific type with fixed attributes. If the attributes thus specified when the variable is used as a !!source!! are not the same as when that variable was most recently used as a !!destination!!, the results are undefined.

Register: If the parameter is the register, the name of the entity is REG. The typeclass and attributes must be specified by giving a qualifier that is the name of a previously-declared specific type with fixed attributes. If the attributes thus specified when REG is used as a !!source!! are not the same as when it was most recently used as a !!destination!!, the results are undefined.

Array elements: If the parameter is an element of an array, the index is given as a qualifier. It may be any integer entity (including an element of an integer array). The element specified is chosen before the operand in which the parameter appears is performed. An array element may be used anywhere that an entity with the same attributes may be used.

CRF 109 126 163 245 261

Rounding numeric results: The qualifier ROUND may be given with any numeric !!destination!! variable with the EXACT_REP attribute. This has the effect of storing into the variable the integer multiple of the variable's current !!resolution!! that is closest to the actual result of the operation in which it appears.

Truncating numeric results: The qualifier TRUNC may be given with any numeric !!destination!! variable that has the EXACT_REP attribute. This has the effect of storing a value into the variable such that (a) y is one of the two integer multiples of the variable's current !!resolution!! closest to the value resulting from the operation; and (b) absv(y) lt absv(computed result).

Specifying subrange information for variables: The user may supply range information when a numeric variable is used as a parameter. The qualifier is of the form

lb : ub

where "lb" and "ub" are the lower and upper bound, respectively, of the variable, given as literals or ascons within the range of the variable. Specifying a subrange for a !!destination!! of an operation asserts that the result of the operation will be within the subrange given. Specifying a subrange for a !!source!! of an operation asserts that, at the time of the call, the operand will have a value within that subrange. A subrange specification for an IO parameter is interpreted as the conjunction of both !!source!! and !!destination!! assertions. The implementation may be based on the assumption that the assertion is true, and results will be unpredictable if the assertion is violated.

Meaningful combinations of qualifiers: The following combinations of qualifiers, and of entities and qualifiers, are not allowed:

- ROUND and TRUNC both occurring in the same parameter specification;
- more than one occurrence of the same kind of qualifier;
- attribute specification for a fixed-attribute variable or a constant;
- ROUND or TRUNC with an entity not having the EXACT_REP attribute, or with an entity that is not a !!destination!!;
- an array index with an entity that is not an array;
- a subrange specification for a non-numeric entity.

EC.DATA.2.4.1.2 USING POINTERS

Anywhere that an entity may be used, the reference may be replaced by

⟨DEREF,pointer⟩

where "pointer" is the name of a previously-declared entity of the PTR typeclass. This has the same effect as using the entity that is the current value of the pointer.

CRF 109 111 121 163 196 198 261

EC.DATA.2.4.2 LISTS OF DESTINATIONS

Any !!actual parameter!! given for an O (output) parameter in an EC access program may be given as a !!list!! of operands (possibly including the register). Each element of the !!list!! must be suitable for use as a destination of the operation. All of the parameters will receive the same value (subject to any ROUND or TRUNC effects); the assignments may be made in an arbitrary order or simultaneously.

EC.DATA.2.4.3 NOSTORE

The special identifier (not an entity) NOSTORE may be used in place of an !!actual parameter!! for an O (output) parameter in an EC access program. This has the effect of not storing the computed result into any entity, although the computed value can be used to determine the exit of the program (see EC.PGM.1).

EC.DATA.2.4.4 UNDESIRED EVENTS

The following undesired events can occur when parameters are specified using the forms described in this section:

```
%assertion violation%
%%attribute not allowed%%
%%attribute not given%%
%illegal array index%
%%illegal ptr target%%
%illegal round/trunc%
%%inappropriate attributes%%
%%inconsistent register access%%
%%index not allowed%%
%%literal or ascon too big%%
%%multiple qualifiers%%
%%REG not allowed%%
%%res too fine%%
%%subrange not allowed%%
%%undeclared operand%%
%%undeclared spectype%%
```

CRF 169 170 171 172 173 181 240 252 267

EC.DATA.2.5 TRANSFER OPERATIONS

Program name	Parm type	Parm info	Undesired events
+SET+	p1:see below;I	!+source+!	%inconsistent lengths%
++SET++	p2:see below;O	!+destination+!	%range exceeded%

Parameters

p1 and p2 must be either both real, or both timeint, or both bitstrings of the same length, or both pointers of the same specific type.

Program Effects

p2 = the value of p1 before the operation.

CRF 143 181 182

EC.DATA.2.6 NUMERIC OPERATIONS

EC.DATA.2.6.1 NUMERIC COMPARISON OPERATIONS

Program name	Parm type	Parm info	Undesired events
+EQ+	p1:see below;I	!+source+!	None
+NEQ+	p2:see below;I	!+source+!	
+GT+	p3:boolean;O	!+destination+!	
+GEQ+	p4:see below;I	!+user threshold+!	
+LT+			
+LEQ+			

Parameters

p1,p2,p4 must be either all real types or all timeint types.

Program Effects

+EQ+	p3 = (p1 = p2)*
+NEQ+	p3 = NOT (p1 = p2)*
+GT+	p3 = p1 - p2 is positive and NOT (p1 = p2)*
+GEQ+	p3 = (p1 = p2)* OR (p1 - p2 is positive)
+LT+	p3 = p1 - p2 is negative and NOT (p1 = p2)*
+LEQ+	p3 = (p1 = p2)* OR (p1 - p2 is negative)

*Definition of equality (=):

absv(p1 - p2) is less than or equal to !+user threshold+!.

EC.DATA.2.6.2 NUMERIC CALCULATIONS

Program name	Parm type	Parm info	Undesired events
+ABSV+ +COMPLE+	p1:see below;I p2:see below;O	!+source+! !+destination+!	%range exceeded%
+ADD+ +MUL+ +SUB+	p1:see below;I p2:see below;I p3:see below;O	!+source+! !+source+! !+destination+!	
+DIV+	p1:see below;I p2:see below;I p3:boolean;I p4:see below;I	!+source+! !+source+! check for success? !+destination+!	%range exceeded% %divide by zero% %%variable parm%%
+SIGN+	p1:see below;I p2:see below;I p3:see below;I	!+source+! of sign !+source+! of magnitude !+destination+!	None.

Parameters

+ADD+
+ABSV+
+COMPLE+
+SUB+
(1) all operands real, or
(2) all operands timeint

+MUL+
(1) all operands real, or
(2) one of p1 or p2 real, the other operands timeint

+DIV+
(1) p1,p2 and p4 real, or
(2) p1 and p2 timeint and p4 real, or
(3) p1 timeint, p2 real, p4 timeint;
p3 must be given by a literal or ascon.

+SIGN+
(1) p1, p2, p3 real; or
(2) p1, p2, p3 timeint.

CRF 107 167 198 232

Program Effects

+ABSV+	p2 = magnitude(p1)
+ADD+	p3 = p1 + p2
+COMPLE+	p2 = - p1
+MUL+	p3 = p1 * p2
+SIGN+	p3 = sign(p1) * absv(p2), where sign(0) is defined to be 0.
+SUB+	p3 = p1 - p2
+DIV+	If a subrange assertion (lb:ub) was given for p4 AND absv(ub) lt absv(p1/p2) then: (a) if p3=$true$ then sign(p4)=sign(p1/p2), magnitude of p4 is undefined, and the built-in program entity DIV_FAIL is invoked; (b) if p3=$false$ then p4 is undefined. Otherwise, p4 = p1/p2.

Built-in Objects

DIV_FAIL	A program variable of spectype E1 (defined in EC.PGM.2.3). DIV_FAIL has no initial value; if it is invoked before assigning a value to it, the UE %uninitialized pgm% (defined in EC.PGM.2.5) will be raised.

CRF 107 167 232 254

EC.DATA.2.6.3 OPERATIONS CONVERTING OTHER TYPES TO REALS

Program name	Parm type	Parm info	Undesired events
+R_BITS_2COMP+	p1:bitstring;I	!+source+!	%range exceeded%
+R_BITS_POSITIVE+	p2:integer;I	!+radix pt ident+!	
+R_BITS_SIGNMAG+	p3:real;O	!+destination+!	
+R_TIME_HOUR+	p1:timeint;I	!+source+!	
+R_TIME_MIN+	p2:real;O	!+destination+!	
+R_TIME_MS+			
+R_TIME_SEC+			

Program effects

+R_BITS_2COMP+ p3 = real value equivalent to p1 assuming that bitstring p1 is in a two's complement representation, bit 0 is the most significant bit, and the radix point is specified by p2.

+R_BITS_POSITIVE+ p3 = real value equivalent to p1 assuming that bitstring p1 represents a positive number, with bit 0 the most significant bit, and the radix point is specified by p2.

+R_BITS_SIGNMAG+ p3 = real value equivalent to p1 assuming that bitstring p1 is in a sign magnitude representation, bit 0 is the most significant bit, and the radix point is specified by p2.

+R_TIME_HOUR+ p2 = a real value giving the time p1 in hours.

+R_TIME_MIN+ p2 = a real value giving the time p1 in minutes.

+R_TIME_MS+ p2 = a real value giving the time p1 in milliseconds.

+R_TIME_SEC+ p2 = a real value giving the time p1 in seconds.

EC.DATA.2.6.4 OPERATIONS CONVERTING TO TIME INTERVALS

Program name	Parm type	Parm info	Undesired events
+T_REAL_MS+ +T_REAL_SEC+ +T_REAL_MIN+ +T_REAL_HOUR+	p1:real;I p2:timeint;O	!+source+! !+destination+!	%range exceeded%

Program effects

+T_REAL_MS+ p2=timeint value equivalent to p1 assuming p1 to specify the time interval in milliseconds.

+T_REAL_SEC+ p2=timeint value equivalent to p1 assuming p1 to specify the time interval in seconds.

+T_REAL_MIN+ p2=timeint value equivalent to p1 assuming p1 to specify the time interval in minutes.

+T_REAL_HOUR+ p2=timeint value equivalent to p1 assuming p1 to specify the time interval in hours.

EC.DATA.2.7 OPERATIONS FOR THE BITSTRING TYPE CLASS

Bits in all bitstring types are numbered from 0 upward. We refer to bit 0 as the leftmost bit and a shift of information from higher numbered bits to lower numbered bits as a left shift.

EC.DATA.2.7.1 BITSTRING COMPARISON OPERATIONS

Program name	Parm type	Parm info	Undesired events
+EQ+	p1:bitstring;I	!+source+!	None.
+NEQ+	p2:bitstring;I	!+source+!	
	p3:boolean;O	!+destination+!	

Program Effects

+EQ+ p3 = (p1 = p2)*
+NEQ+ p3 = NOT (p1 = p2)*

*Definition of equal (=) length(p1) = length(p2) and
for all i such that 0 lseq i lt length(p1)
bit (i) of p1 = bit (i) of p2

EC.DATA 2.7.2. BITSTRING CALCULATIONS

Program name	Parm type	Parm info	Undesired events
+AND+	p1:bitstring;I	!+source+!	%inconsistent lengths%
+CAT+	p2:bitstring;I	!+source+!	
+MINUS+	p3:bitstring;O	!+destination+!	
+NAND+			
+OR+			
+XOR+			
+NOT+	p1:bitstring;I	!+source+!	
	p2:bitstring;O	!+destination+!	
+SHIFT+	p1:bitstring;I	!+source+!	
	p2:integer;I	shift length	
	p3:bitstring;O	!+destination+!	
+REPLC+	p1:bitstring;I	!+source+!	%nonexistent position%
	p2:integer;I	source start position	
	p3:integer;I	destination start position	%inconsistent lengths%
	p4:integer;I	length	
	p5:bitstring;I	background !+source+!	
	p6:bitstring;O	!+destination+!	

Program Effects

+AND+	p3 = p1 AND p2
+CAT+	p3 = p1 followed by p2
+MINUS+	p3 = p1 AND (NOT p2)
+NAND+	p3 = NOT (p1 AND p2)
+NOT+	p2 = NOT p1
+OR+	p3 = p1 OR p2
+REPLC+	p6[p3:p3+p4-1] = p1[p2:p2+p4-1] p6[all other bits] = corresponding bits in p5
+SHIFT+	p4 = shift of p1 by p2 positions to the right (or -p2 positions to the left). The vacated bits are set to "0:B".
+XOR+	p3 = (p1 AND (NOT p2)) OR (p2 AND (NOT p1))

CRF 090 143 195 201

EC.DATA.2.7.3. OPERATIONS CONVERTING TO BITSTRING

Program name	Parm type	Parm info	Undesired events
+B_REAL_2COMP+ +B_REAL_POSITIVE+ +B_REAL_SIGNMAG+	p1:real;I p2:integer;I p3:bitstring;O	!+source+! !+radix pt ident+! !+destination+!	%left truncation%

Program Effects

+B_REAL_2COMP+ p3 = two's complement representation of p1, such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 of p3 will be the most significant. The operation truncates all bits beyond the highest numbered bit in the !+destination+! bitstring.

+B_REAL_POSITIVE+ p3 = bitstring representation of ABSV(p1), such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 of p3 will be the most significant bit. The operation truncates all bits beyond the highest numbered bit in the !+destination+! bitstring.

+B_REAL_SIGNMAG+ p3 = sign magnitude representation of p1, such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 will be the sign bit and bit 1 the most significant bit of the magnitude. The operation truncates all bits beyond the highest numbered bit in the !+destination+! bitstring.

EC.DATA.2.8 OPERATIONS FOR THE POINTER TYPE CLASS

Except for the transfer operations specified in EC.DATA.2.5, there are no operations provided for pointers.

CRF 261

EC.DATA.3 Local Type Definitions

array-init

A description of a list of constant or literal initial values for an array. The list must contain exactly as many members as there are elements in the array. If the array is numeric, all values must be within the !!range!! of the specific type of the array. If the array is bitstring, all values must be of the the same length as the specific type of the array. The syntax is:

```
array-init ::= count.prod ( elem.list )
count.prod ::= empty   OR   count.prod * count
count      ::= positive integer constant or literal
elem.list  ::= elem   OR   elem.list , elem
elem       ::= constant or literal of type appropriate to
                  the array
               OR  array-init
empty      ::=
```

An empty count has the value of 1. The meaning of count.prod * count is the product of count.prod and count. The meaning of count.prod (elem.list) is count.prod occurrences of elem.list, with the parentheses removed and the occurrences separated by commas. Examples:

```
*3(a,b)          == (a,b,a,b,a,b)
*3*2(a)          == (a,a,a,a,a,a)
*2(*3(a),*1(b))  == (a,a,a,b,a,a,a,b)
(a)              == (a)
```

attribute

An attribute for a bitstring is a positive integer specifying length.

A real or timeint attribute is a parenthesized list:

(lower bound, upper bound, !!resolution!!, EXACT_REP)

The fourth element is optional; if omitted, the third comma is also omitted. The lower bound and upper bound are often collectively called "range" (see !!range!!). If present in a type declaration, EXACT_REP specifies that results to be stored in a variable of that type may be truncated/rounded (at the discretion of the user) to a value that is a multiple of the variable's current !!resolution!!.

!!Range!! and !!resolution!! for reals must given by real entities, and by timeint entities for timeints.

A pointer attribute is the name of a previously-declared or built-in specific type.

Attributes for other typeclasses are given in EC.SMPH and EC.TIMER.

binding	Either FIX (meaning attributes do not change at run time) or VARY (meaning attributes may change at run time)
bitstring	An ordered list of values, each value represented by "0" or "1". The number of such values is called the length of the bitstring. A bitstring literal is written as a string of 0s and 1s suffixed by ":B". E.g., 0:B bitstring of length 1 1011:B bitstring of length 4
boolean	Bitstring of length 1. Where convenient, $true$ may denote "1:B", $false$ may denote "0:B".
convar	Either ASCON (meaning constant that will not change without a reassembly) or LOADCON (meaning constant that may be changed by a memory loading device while the program is not running) or VAR (meaning variable).
data_set	A group (previously declared by ++DCL_DATA_SET++) of user-defined entities that the user may rank according to desired access speed.
data-set-reln	A partial ordering on the set of all data sets, given as a !!relation!!.
indexset	A set of permissible indices. Only sets of contiguous integers may be created. The set must be specified in the following way: (si,li) where "si" denotes the smallest index and "li" denotes the largest index. Both si and li must be integer ascons or literals. For example, (7,12) indicates a six-element array indexed by the integers from 7 through 12. (-4,-4) indicates a one-element array whose index is -4.
integer	Real with !!resolution!! = 1.
name	An identifier for an object created. A name must consist only of alphanumerics or one of the following: +#%@/$()_
pointer	A type that provides indirect referencing to other declared entities. A pointer literal is given by ⟨REF,x⟩ where "x" is another non-literal entity; "x" may not be REG or an i/o data item (see EC.IO).
rank-data-relation	A !!relation!! between entity and/or array names.

CRF 054 093 136 151 181 185 195
198 240 266 267

real — An approximation to conventional real numbers. Real literals are denoted in one of the following formats:

standard decimal notation e.g. -112.345, .000234, 127

exponent notation: decimal number, followed by :En, where n is an integer, meaning that the value is the number multiplied by 10n
e.g. 1.12345:E2 (= 112.345); 2.34:E-4 (=.000234)

power of two notation 2**n, where n is an integer;
e.g., 2**3 (= 8); 2**-4 (=.0625); -2**4 (=-16)

spectype — An identifier that has been previously declared as a type in a ++DCL_TYPE++ operation, or the name of a spectype built in to the EC. The latter includes BOOLEAN (representing the built-in bitstring type "boolean"), as well as those named in EC.PGM.2 and EC.IO.

timeint — Representation of a time interval. Literals of type timeint are denoted by using the name of one of the real-to-timeint conversion programs of EC.DATA.2.6.4 and a real literal. The form is:

⟨program-name , real-literal⟩

Where a blank appears, any number of blanks (including zero) may appear. The value thus specified is that which would be returned by the named program were it called with the real as the input parameter. For example, ⟨+T_REAL_SEC+, 4.0⟩ denotes a timeint value of 4 seconds.

typeclass — Either BITS (meaning bitstring), PTR (meaning pointer), REAL, or TIMEINT (meaning time interval). Other values are SEMAPHORE (see EC.SMPH), PGM (see EC.PGM.2), and TIMER (see EC.TIMER).

version — A version name applicable to the specific type being declared. Version names and characteristics are listed in Appendix 6.

CRF 093 105 179 180 181 182 199
224 260

EC.DATA.4 Dictionary:

Term	Definition
!+destination+!	variable, register or a list of such entities; will contain results of operation.
!!destination!!	An O or IO !!actual parameter!! to an EC access program or a user-defined EC program.
!!hardest attributes!!	For bitstring types, the maximum length that an entity of this spectype will ever assume. For real or timeint types, the !!range!! and !!resolution!! such that the ratio of !!range!! to !!resolution!! is the maximum that an entity of this spectype will ever assume, where !!range!! is defined as ABSV(upper bound - lower bound). Further, if an entity of this spectype will ever have the EXACT_REP attribute, it must be specified here. The syntax for specifying !!hardest attributes!! is the same as for attributes, defined in EC.DATA.3.
!!list!!	An unordered sequence of elements. The syntax of a !!list!! is: ::= item OR (item , ... , item)
!+radix pt ident+!	Interpreting the bitstring as a binary real number with bit 0 the most significant bit, 2 raised to the !+radix pt ident+! power is the significance of the rightmost (highest numbered) bit. For instance, a value of zero means that the bitstring represents an integer.
!!range!!	The set of values between (and including) the lower bound and upper bound of a numeric data type.
!!relation!!	A set of ordered pairs. In EC, a !!relation!! is specified by giving two !!list!!s; the ordered pairs is that obtained by taking the cross-product of the !!lists!!. The syntax is: ::= (!!list!! , !!list!!)

CRF 119 154 192 206 226 267

!!resolution!!	The maximum difference between any two consecutive representatives of the values of a real or timeint data type.
!+source+!	variable, register, literal or constant; has a value to be used as input to the operation.
!!source!!	An I or IO !!actual parameter!! to an EC access program or a user-defined EC program.
!+user threshold+!	A difference that user programs specify for a comparison operation; i.e., two numbers whose difference is less than this are considered equal.

EC.DATA.5 Undesired Event Dictionary:

%assertion violation%	A variable's value was not in the [lb,ub] !!range!! specified in a subrange assertion.
%%attribute not allowed%%	A fixed-attribute entity was given using the variable-with-varying-attributes form of parameter specification.
%%attribute not given%%	An attribute qualifier was not given for an operand that is of varying-attribute spectype.
%divide by zero%	A user program attempted to divide by zero.
%illegal array index%	The index supplied in an array reference is not in the index set of the array.
%%illegal index set%%	The index set of an array is not: (a) contiguous (b) in ascending order (c) integers (d) given with ascons or literals.
%%illegal ptr target%%	A pointer may not point to REG, nor to an i/o data item.
%illegal round/trunc%	A user specified ROUND or TRUNC for a variable that does not have the EXACT_REP attribute or is not a !!destination!!, or tried to round and truncate the same numeric result.
%%inappropriate attributes%%	The attributes given are not valid for the type class at hand.

CRF 107 111 121 133 139 143 158 170
171 240 262

%%inconsistent data ranking%%	There are EC data entities x and y (not necessarily distinct) such that, if we assume the rank relation for data to be transitive, both (x,y) and (y,x) are in the accumulated relation.
%inconsistent lengths%	The length of the result of a bitstring operation differs from the length of the destination variable.
%%inconsistent register access%%	An operation that changes the value of a register has the "-SAVE" suffix.
%%index not allowed%%	An index was provided as a qualifier for an operand that is not an array.
%left truncation%	The most-significant bits are lost in a real to bitstring conversion. This results from the user specifying a radix point too close to the most significant bit in the destination bitstring.
%%length too great%%	The length of a bitstring type exceeds the maximum allowed.
%%literal or ascon too big%% %%loadcon too big%%	The value of a non-variable is greater in magnitude than that allowed for an entity of that typeclass, as given by a system generation parameter.
%mdr outside range%	A !+max div result+! was given that exceeded the !!range!! of the destination variable.
%%multiple qualifiers%%	More than one qualifier of the same type was given for a single operand.
%%name in use%%	An attempt has been made to redefine a name of one of the following: - a built-in object; - an EC access program; - an EC UE; - an EC reserved word; - an EC system generation parameter; - a user-defined spectype, entity, array, region, or program.

CRF 107 112 114 120 138 151 157 172
173 205 209 221 247 257

%nonexistent position%	A user has specified (1) a start position that does not exist in the bitstring; or (2) a start position and a length that define a substring not contained in the bitstring.
%range exceeded%	The value being stored into a variable is outside the !!range!! of the variable.
%%range too great%%	The magnitude of the declared !!range!! exceeds the maximum allowed for that typeclass, as given by a system generation parameter.
%%ranres too great%%	The ratio of the declared !!range!! to the declared !!resolution!! exceeds the maximum allowed for that typeclass, as given by a system generation parameter.
%%res too fine%%	Declared resolution (or implied resolution of a literal) was less than the minimum allowed for that typeclass, as given by a system generation parameter.
%%subrange not allowed%%	A subrange qualifier was given for a non-numeric operand.
%%undeclared operand%%	A !!source!! or !!destination!! is or refers to (a) an entity not declared previously using ++DCL_ENTITY++; and (b) not an element of a previously-declared array; and (c) not an EC built-in object or system generation parameter.
%%undeclared spectype%%	The user has supplied an undeclared spectype in an entity or array declaration, or as the attribute of a varying-attribute variable or a PTR spectype.
%%unknown initial value%%	A variable has been used as an initial value of a declared entity.
%%unknown operand in attribute%%	An attribute has been given using an entity not previously declared using ++DCL_ENTITY++ as a value.
%%variable parm%%	User supplied a variable or loadcon for an !!actual parameter!! when an ascon or literal was called for.

CRF 181 193 205 209 232

%%varying constant%%	A user sought to declare a constant of a specific type that has varying attributes.
%%wrong init value size%%	The set of initial values is not the same size as the array.
%%wrong init value type%%	A constant or literal used as an initial value is not in the domain of the type of the entity being initialized.

EC.DATA.6 System Generation Parameters

Parameter	Type	Definition
#max bits length#	integer	The maximum number of bits allowed for a bitstring.
#max real ascon# #max timeint ascon#	real timeint	Maximum allowable magnitude for a real (timeint) ascon or literal.
#max real loadcon# #max timeint loadcon#	real timeint	Maximum allowable magnitude for a real (timeint) loadcon.
#max real ranres ratio# #max timeint ranres ratio#	real timeint	Maximum allowable magnitude of the ratio of a type's !!range!! to its !!resolution!!.
#max real range# #max timeint range#	real timeint	Maximum allowable magnitude for the absv(upper bound - lower bound) for a real (timeint) type.
#min real resolution# #min timeint resolution#	real timeint	Minimum allowable resolution for a real (timeint) entity.

CRF 209 214 221

EC.IO.1 Introduction

This module implements two types of bitstring entities known as input data items and output data items, which are used to communicate between the computer and external devices. This interface also includes facilities for i/o used during channel diagnostics.

Each data item may be enabled or disabled by user programs. When enabled, communication with the outside world is possible. The values of input data items may be set by external devices. The values of output data items are transmitted to external devices. When a data item is disabled, its connection with the outside world is severed.

Although input data items are normally "read-only" and output data items are normally "write-only", a few may be both read and written when they are disabled. These may be used as storage at such times. An input (output) data item may always be used as a !!source!! (!!destination!!) in an EC statement.

User programs are able to check to see if an external communication has been successful.

Within these constraints, an input or output data item may be used exactly as other bitstring variables.

In addition to the input data items described above, some input from the outside world is handled only through semaphores. For these inputs, which correspond to transient events occurring in external devices, a semaphore is incremented when the event occurs. There are no corresponding bitstrings for these inputs.

EC.IO.2 Interface Overview

EC.IO.2.1 Access programs

Program name	Parm type	Parm info	Undesired Events
+DISABLE+	p1:dataitem;I	name of data item	%already disabled%
+ENABLE+	p1:dataitem;I	name of data item	%already enabled%
+G_SUCCESS+	p1:dataitem;I p2:boolean;O	name of data item !+i/o success+!	None.

Program effects

+ENABLE+ Enables transmission to/from the external environment. If p1 is an input data item, then external values for this input item will now become available internally as soon as practicable. If p1 is an output data item, the value is now available externally. If the item is read-write input, use of the item as a !!destination!! in an EC statement is now prohibited until disabled. If the item is read-write output, use of the item as a !!source!! in an EC statement is now prohibited until disabled.

At system-generation time, all data items are enabled.

+DISABLE+ Transmission to/from the external environment will be inhibited. If the item is read-write input, it may now be used as a !!destination!! in an EC statement. If the item is read-write output, it may now be used as a !!source!! in an EC statement.

EC.IO.2.2 Access programs for IO diagnostics

Program name	Parm type	Parm info	Undesired Events
+TEST_AC+	pl:boolean;0	!+io test result+!	None
+TEST_CSA+	pl:boolean;0	!+io test result+!	
+TEST_CSB+	pl:boolean;0	!+io test result+!	
+TEST_DC+	pl:boolean;0	!+io test result+!	
+TEST_DIOW1+	pl:boolean;0	!+io test result+!	
+TEST_DIOW2+	pl:boolean;0	!+io test result+!	
+TEST_DIOW3+	pl:boolean;0	!+io test result+!	
+TEST_XACC+	pl:boolean;0	!+io test result+!	
+TEST_YACC+	pl:boolean;0	!+io test result+!	
+TEST_ZACC+	pl:boolean;0	!+io test result+!	

Effects

These programs report the results of input/output hardware diagnostic tests. If the test is performed periodically or independent of user request, the result given will be that of the most recent test. If the test is performed on request, the command will initiate the test and report the result when the test is complete. In addition, the following effects are observable.

+TEST_AC+ This program reports the results of the AC signal converter check. It may interfere with:

output, when the data item is

//BRGDEST//	//GNDTRK//	
//RNGHND//	//RNGTEN//	//RNGUNIT//
//STEERAZ//	//STEEREL//	

+TEST_CSA+ This program reports the results of the cycle-steal channel A and serial channel 1 check. It may interfere with:

output, when the data item is

//ASAZ//	//HUDCTL//	//USOLCUAZ//
//ASEL//	//LSOLCUAZ//	//USOLCUEL//
//ASLAZ//	//LSOLCUEL//	//VERTVEL//
//ASLEL//	//MAGHDGH//	//VTVELAC//
//ASLCOS//	//MAPOR//	//XCOMMF//
//ASLSIN//	//PTCHANG//	//XCOMMC//
//AZRING//	//PUACAZ//	//YCOMM//
//BAROHUD//	//PUACEL//	
//FLTDIRAZ//	//ROLLCOSH//	
//FPMAZ//	//ROLLSINH//	
//FPMEL//		

input, when the data item is /LOCKEDON/ or /SLTRNG/.

CRF 263

+TEST_CSB+ This program reports the results of the cycle steal channel B and serial channel 2 check. It may interfere with:

output, when the data item is
//CURAZCOS// //CURAZSIN// //CURPOS//

input, when the data item is
/ANTGOOD/ /DGNDSP/ /DRFTANG/
/DRSFUN/ /DRSMEM/ /DRSREL/
/ELECGOOD/

+TEST_DC+ This program reports the results of the DC signal converter check. It may interfere with:

output, when the data item is
//FPANGL// //GNDTRVEL// //STERROR//

+TEST_DIOW1+
+TEST_DIOW2+
+TEST_DIOW3+ These programs report the results of the checks on discrete input and output word pairs 1, 2, and 3 respectively. These programs may interfere with:

output, when the data item is
//DOW1// //DOW2//

input, when the data item is
/DIW1/ /DIW2/ /DIW3/
/DIW4/ /DIW5/ /DIW6/
/ANTGOOD/ /DGNDSP/ /DRFTANG/
/DRSFUN/ /DRSMEM/ /DRSREL/
/ELECGOOD/ /LOCKEDON/ /SLTRNG/
/SINSDD/

+TEST_XACC+
+TEST_YACC+
+TEST_ZACC+ These programs report the results of checks on the accelerometer and torque registers associated with the X, Y, and Z axes of the IMS respectively. These programs may cause the IMS to lose its alignment and velocities, and may interfere with:

output, when the data item is
//XGYCOM// //YGYCOM// //ZGYCOM//

input, when the data item is
/XGYCNT/ /XVEL/ /YGYCNT/
/YVEL/ /ZGYCNT/ /ZVEL/

EC.IO.2.3 Built-in Objects

The names of all data items are listed in Appendix 5 of this document.

Undesired Events associated with Built-in Objects

The following undesired events may occur when data items are used in EC statements:

%read-write violation%
%%read/write-only violation%%

EC.IO.2.4 Events signalled by incrementing a semaphore

For some inputs, an event is signalled (by incrementing a semaphore) when a new value of an input data item has been transmitted. The event is of the form

@T(!+ x ready+!)

where "x" is the name of the data item.

Some inputs correspond to an event occurring in an external device. When such an event occurs, this module will signal a corresponding event of the form

@T(!+ x occurred)

where "x" specifies the event.

These events and their corresponding semaphores are enumerated in Appendix 5.

EC.IO.3 Local Type Definitions

dataitem	The name of any input or output data item. The data items are listed in Appendix 5 of this document. The semantics of the data items are given in Chapter 2 of [REQ].

EC.IO.4 Dictionary

Term	Definition
Any item of the form !+ x ready+!	The named data item is now available for read operations.
Any item of the form !+ x occurred+!	The named event has just occurred in an external device.
!+i/o success+!	true iff the last transmission associated with the named data item was successful.
!+io test result+!	true iff the i/o hardware passes built-in test.

EC.IO.5 Undesired Event Dictionary

%already disabled%	A user program has tried to disable a data item already disabled.
%already enabled%	A user program has tried to enable a data item already enabled.
%%read/write-only violation%%	A read-only (write-only) data item appears as a !!destination!! (!!source!!).
%read-write violation%	A program call was executed with a read-write input (output) data item as a !!destination!! (!!source!!) when that data item was enabled.

EC.IO.6 System-Generation Parameters

Parameter	Type	Explanation
#max i/o time x#	timeint	(where "x" is replaced by the name of each data item) The maximum time interval that can elapse between the beginning of the access program that reads/writes the named item, and the time it takes for the external transmission to take place.
#nbr fltrec elements#	integer	Defined in Appendix 5.

CRF 102 214 231 264 265

EC.MEM
Virtual Memory Module

EC.MEM.1 Introduction

This module provides a uniformly addressable memory for the Extended Computer, as well as hiding the particular addresses to which data and instructions are allocated.

The only part of the interface that is not hidden from EC programmers (who do not need to use it) is that which reports to the user the results of a diagnostic test on the memory hardware. The specifications for the remainder of the module appear in [VM].

EC.MEM.2 Interface Overview

EC.MEM.2.1 Memory diagnostic access programs

Program name	Parm type	Parm info	Undesired Events
+TEST_MEMORY+	p1:boolean;0	!+memory test result+!	None

Effects

Reports the result of the memory diagnostic test. If the test is performed periodically or independent of user request, the result given will be that of the most recent test. If the test is performed on request, the command will initiate the test and report the result when the test is complete.

EC.MEM.3 Local Type Definitions: None.

EC.MEM.4 Dictionary

!+memory test result+!	true iff the memory diagnostic test is passed.

EC.MEM.5 Undesired Event Dictionary: None.

EC.MEM.6 System Generation Parameters: None.

CRF 263

EC.PAR.1.1 Introduction

The process mechanism allows the definition of a set of sequential processes that will proceed in parallel and unknown relative speeds. Demand processes are activated when specific events occur. Periodic processes may be turned on or off, but are re-started at regular intervals when turned on.

EC.PAR.1.2 Interface Overview

EC.PAR.1.2.1 Access program table

Program name	Parm type	Parm info	Undesired events
++D_PROCESS++	p1:timeint;I p2:invocation;I	!+deadline+! process body	%inconsistent time parms% %%undeclared program%% %%variable timing parms%%
++P_PROCESS++	p1:timeint;I p2:timeint;I p3:semaphore;I p4:invocation;I p5:boolean;I	!+deadline+! !+period+! !+starting event+! process body !+on/off+!	%inconsistent time parms% %%illegal synch%% %%undeclared program%% %missed deadline% %%variable timing parms%%
+TEST_INTERRUPTS+	p1:boolean;O	!+interrupt test result+!	None.

Parameters

p1 must be given by a literal or an ascon.

CRF 104 146 153 168 234 263

Program effects

++D_PROCESS++ establishes a demand process that becomes active after @T(!+power up+!). The body of the process is the program named by p3. The process remains active until it is suspended as a result of a synchronization operation (see EC.PAR.2, EC.PAR.2) or executes the last statement in its body. During the interval when it is active, it will execute before p2 real time has elapsed. A process that is suspended as a result of a synchronization operation may start again. A process that executes its last statement will start again only after a system generation.

++P_PROCESS++ establishes a periodic process that becomes active after the semaphore named by !+starting event+! becomes nonnegative. The body of the process is the program named by p5. While the boolean named by p6 is true, a built in semaphore, NEXT PERIOD, will be incremented at the start of each !+period+! amount of real time. After the start of a !+period+!, the process will complete exeuction before p2 real time has elapsed. The process must perform [+DOWN+,NEXT_PERIOD] [+PASS+,NEXT_PERIOD].

If p6 is given as a variable, and that variable becomes false during execution, the process will stop when it waits for the start of its next !+period+! (by invoking [+PASS+,NEXT_PERIOD]).

If p3 is given as a variable, and that variable changes value during execution, the process will change its !+period+! within an amount of time equal to the previous value of p3.

Both If two (or more) processes simultaneously execute sequences of statements that read and/or alter the value of some data, the results are unpredictable because the executions may overlap in time. However, EC access programs are considered indivisible. If two EC access programs are executed simultaneously by two processes, the effect will be as if one of the processes executed its access program before the other; the order is not specified. Note that the invocation of a user-supplied routine is the execution of a single EC access program, but the execution of the body of that routine is a sequence of EC statements.

Program effects (continued)

+TEST_INTERRUPTS+	Reports the results of the interrupt hardware checks. If the test is performed periodically or independent of user request, the result given will be that of the most recent test. If the test is performed on request, the command will initiate the test and report the result when the test is complete. It may interfere with normal operation of timers and input/output commands in unpredictable ways.

EC.PAR.1.2.2 Built-in objects

Name	Used to Refer to
NEXT_PERIOD	semaphore variable, private to each periodic process, that will be incremented by the EC at the start of each period. Each periodic process only has access to its own NEXT_PERIOD. Semaphores are described in EC.SMPH.

EC.PAR.1.3 Local Type Definitions

invocation	An occurrence of a program invocation, as described in EC.PGM.3.2.

EC.PAR.1.4 Dictionary

Term	Definition
!+deadline+!	The maximum amount of real-time that can be allowed to elapse between the time that a process can proceed and the time that it reaches the next point of suspension.
!+interrupt test result+!	true iff the interrupt hardware passes built-in test.
!+on/off+!	the boolean whose value will be used to start/stop the periodic process in whose definition it appears. Its value must be $true$ whenever the periodic process is supposed to proceed. If it is $false$ when the process next reaches its starting point, the process will be suspended until it becomes $true$ again. Of course, the value may only be changed if the boolean was given as a variable.

CRF 090 129 153 168 234 243 263

!+period+!	The timeint whose value will be interpreted as the amount of real-time that should elapse between the beginning of one execution of a periodic process and the beginning of the next execution. If !+period+! is given as a variable, changing its value has the result of changing the period of any process for which it was used as the !+period+!.
!+starting event+!	The name of a semaphore that, when becoming nonnegative, will cause the periodic process in which it is named to become active.

EC.PAR.1.5 Undesired Event Dictionary

%%illegal synch%%	a synchronization operation other than the required +DOWN+(NEXT_PERIOD) and +PASS+(NEXT_PERIOD) (see EC.SMPH) appears in the body of a periodic process; or those required operations were omitted.
%inconsistent time parms%	the timing parameters are contradictory; e.g. !+max CPU time req+! exceeds !+deadline+!, or !+deadline+! exceeds the current value of !+period+!.
%missed deadline%	a process has missed its deadline because too many demand processes have occurred, or because its !+deadline+! was less than the CPU time required for it to execute.
%%variable timing parms%%	!+deadline+! or !+max CPU time req+! was given using a variable or a loadcon.

EC.PAR.1.6 System Generation Parameters: None

CRF 175 198 214 234

EC.PAR.2
EXCLUSION REGIONS

EC.PAR.2.1 Introduction

This module allows constraints to be placed on the potential concurrency of processes executing regions of code by defining an exclusion relation among them. Region 1 excludes region 2 if starting to execute region 2 is forbidden while region 1 is being executed. Mutual exclusion is a special case of this exclusion relation, which is based on [BELP73].

EC.PAR.2.2 Interface Overview

EC.PAR.2.2.1 Access program table

Program name	Parm type	Parm info	Undesired events
++REGION++	p1:name;I p2:statement-list;I	region name region body	%%name in use%%
++EXCLUSION++	p1:exclusion-relation;I		%%undeclared region%%

Effects

++EXCLUSION++ If the exclusion relation includes (A,B) then no process will begin to execute region B in the interval that starts when a process begins execution of region A and ends when that process completes execution of region A. The exclusion relation for all regions is composed of the exclusion relation given in each invocation of this program.

++REGION++ p1 may be used to stand for the section of code that is given in p2. If the last action before the region causes a process to wait, then the process is considered to be inside the region when it is allowed to proceed. If the last statement in p2 is a wait operation, then the process is considered to have left the region when it begins to wait. Including regions in a process will prevent the process from waking up, if doing so would result in a violation of an exclusion region.

CRF 257

EC.PAR.2.3 Local Type Definitions:

exclusion-relation	A !!relation!! on region names.
statement-list	A sequence of !!command!!s.

EC.PAR.2.4 Dictionary: None

EC.PAR.2.5 Undesired Event Dictionary

%%undeclared region%%	an exclusion relation includes regions that have not been identified in the program.

EC.PAR.2.6 System Generation Parameters: None

CRF 093 119 129 214 247 267

EC.PGM.1.1 Introduction

Using the facilities of this module, a user can construct programs composed of invocations of EC built-in access programs and user-defined programs. This is done by naming the entrances and exits of these programs and describing connections between them. Each exit is connect to one entrance. The resulting structure is called a !!constructed program!!. On completion of its execution, a program selects an exit; the next program executed will begin execution at the entrance connected to that exit.

All EC access programs have one entrance. Many have one exit, but some (see EC.PGM.1.2.1) have as many exits as there are values in the range of the output parameter. When such a program is executed, it chooses the exit that corresponds to the value it has computed.

A !!constructed program!! is a literal of the typeclass PGM. In EC.PGM.2 we describe the declaration and use of entities of that typeclass. In EC.PGM.3 we describe facilities for invoking programs as closed subroutines.

EC.PGM.1.2 Interface Overview

EC.PGM.1.2.1 Entrances and exits of EC access programs

Every EC access program has exactly one entrance.

Every EC access program that has a single output parameter has n exits, where n is the number of values that can be computed for the output parameter. The exits are named Ev, where "v" is replaced by the literal representation of the value (except that "-" is replaced by "m"). For example, an exit corresponding to the real -32.7 is named Em32.7. If the value computed is a bitstring of length 1, the exits can be Etrue and Efalse. If a program computes a value u as its result, it takes exit Eu.

All other EC access programs have exactly one exit.

EC.PGM.1.2.2 Entrances and exits of !!constructed program!!s

Program name	Parm type	Parm info	Undesired events
++ENTRANCE++	pl:name-list;I	entrance name(s)	%%duplicate name%% %%port not defined%% %%port defined twice%% %%no such entrance%%
++EXIT++	pl:name-list;I	exit name(s)	%%duplicate name%% %%port not defined%% %%port defined twice%%

Parameters

Each name in pl of ++ENTRANCE++ must be an entrance to a !!command!! in the !!constructed program!!.

Effects

++ENTRANCE++ Defines the given name(s) as entrance(s) to the !!constructed program!! in which they appear. The name(s) may be used when the !!constructed program!! in which this !!invocation!! appears is invoked (see EC.PGM.3).

++EXIT++ Defines the given name(s) as exit(s) to the !!constructed program!! in which they appear. An exit of a !!command!! in the same !!constructed program!! may be connected to one of these exits as specified in EC.PGM.1.2.3; i.e., %%not an exit%% is disabled for the given names in this !!constructed program!!.

EC.PGM.1.2.3 Connecting exits with entrances

All !!invocation!!s are preceded by a label-list of this form:

```
label-list ::=
            OR label-list label :

label      ::= name
```

The UE %%name in use%% applies to label-lists.

All !!invocation!!s are followed by an exit-connector, in this form:

```
exit-connector ::=  ( exit-list : label , ... , exit-list : label )
                OR  : label
                OR
```

The second form of the exit-connector is an abbreviation that is equivalent to pairing all exits of the program with the same label. The third (null) form is equivalent to ":L L:" where L is some name not used anywhere else.

```
exit-list      ::=  exit
                OR  ( exit, exit, ..., exit )
                OR  ( Em : En )
```

The third form is equivalent shorthand for (Em, Em+1, Em+2, ..., En-1, En) where m and n are integer literals, m not greater than n.

```
exit           ::=  name     (naming an exit of the !!command!!)

label          ::=  name     (naming a label of any !!command!! in the
                             !!constructed program!! in which this
                             !!command!! appears; or naming an exit of
                             the !!constructed program!! in which this
                             !!command!! appears.
```

After a !!command!! is executed, the next !!command!! to be executed will be the one whose label is paired in the exit-connector with the exit that the !!command!! selected.

These UEs apply to exit-connectors:

%%dest unknown%%
%%illegal exit-list%%
%nowhere to go%
%%not an exit%%

EC.PGM.1.3 Local Type Definitions

name-list	A !!list!! of names.

EC.PGM.1.4 Dictionary

!!command!!	An !!invocation!!, preceded by a label-list, and suffixed by an exit-connector.
!!constructed program!!	A sequence of !!command!!s, beginning with one !!invocation!! of ++ENTRANCE++ and ending with one invocation of ++EXIT++ specifying the entrances/exits to the sequence considered as a whole. No other invocations of ++ENTRANCE++ or ++EXIT++ may appear.
!!invocation!!	A program invocation, as defined in section EC.PGM.3.

EC.PGM.1.5 Undesired Event Dictionary

%%dest unknown%%	An exit-dest in an exit-connector was neither (a) a label preceding any other !!command!! in the !!constructed program!!; nor (b) the name of an exit to the !!constructed program!!.
%%duplicate name%%	A name appeared twice in the same list.
%%illegal exit-list%%	An exit-list of a !!command!! either (a) names the same exit twice; or (b) is of the form (Em:En) and m is greater than n.
%%no such entrance%%	The name given as the entrance of a !!constructed program!! is not used as a label preceding any !!command!! in the !!constructed program!!.

%%not an exit%%	The exit-connector contained an exit that is not an exit of the program being invoked.
%nowhere to go%	A program took an exit that was left unconnected.
%%port defined twice%%	A !!constructed program!! contained more than one !!invocation!! of ++ENTRANCE++ or ++EXIT++.
%%port not defined%%	A !!constructed program!! did not contain an !!invocation!! of ++ENTRANCE++, or did not contain an !!invocation!! of ++EXIT++.

EC.PGM.1.6 System Generation Parameters: None.

EC.PGM.2
PROGRAM ENTITIES

EC.PGM.2.1 Introduction

This module provides mechanisms for declaring entities of type program and assigning a value to them. The EC access programs are built-in program constants; see EC.PGM.2.2.3.2.

EC.PGM.2.2 Interface Overview

EC.PGM.2.2.1 Declaring a Program type

To declare specific program types, use the ++DCL_TYPE++ program specified in EC.DATA.2.1, with:

p2 = PGM;
p3 a pgm-attribute, defined in section EC.PGM.2.3;
and the other parameters as described there.

EC.PGM.2.2.2 Creating a Program

To create an entity of the program typeclass, use ++DCL_ENTITY++ (see Section EC.DATA.2.2.1) with:

p2 = a program spectype (builtin or user-declared);
p4 = a program literal, or a parameterless program constant of spectype p2;
and the other parameters as described there.

To create an array of program entities, use ++DCL_ARRAY++ (see Section EC.DATA.2.2.2) with:

p2 = a program spectype (builtin or user-declared);
p4 = an array-init (defined in EC.DATA.2.3) of program literals or parameterless program constants of spectype p2;
and the other parameters as described there.

CRF 093 100 103 106 108 115 117 119 121
129 133 134 144 145 150 161 162 165
178 182 186 247 266

0236a

EC.PGM.2.2.2 Other Operations on Program Entities

Program	Parm type	Parm info	Undesired events
++RANK_PGM++	p1:rank-pgm-relation;I		%%inconsistent pgm ranking%%
+SET+ ++SET++	p1:program;I p2:program;O	!+source+! !+destination+!	%%inconsistent pgms%%

Parameters

+SET+
++SET++ p1 and p2 must be program entities having the same attributes. Neither may have parameters.

Program Effects

++RANK_PGM++ Let A and B be program entities previously declared using ++DCL_ENTITY++, or arrays of program entities previously declared using ++DCL_ARRAY++. If (A,B) appears in the rank-pgm-relation for this invocation of the program, then the time it takes to invoke A will be no longer than the time it takes to invoke B in the same program. (If A (B) is an array, then read "all program entities in A (B)".) The rank relation for all user-defined program entities is the union of the rank-pgm-relations given in each invocation of this program. The relation is transitive, antireflexive, and antisymmetric.

+SET+
++SET++ As described in EC.DATA.2.5.

EC.PGM.2.2.3 Built-in Objects

EC.PGM.2.2.3.1 Undesired Event Programs

Every run-time UE specified in this document is a built-in uninitialized program variable, which will be invoked when the error condition corresponding to the UE definition is detected by the EC program(s) to which the UE applies. It is up to the user to assign a value to each of these variables; if one of these variables remains uninitialized at the time of its invocation by an EC access program, the UE %uninitialized pgm% is raised. The variables are of the builtin spectype E1.

For every run-time UE %x% defined in this document, there is a built-in boolean variable Bx with the following properties:

(a) its initial value is $true$;
(b) it may not be used as a !!source!!;
(c) it may only be used as a !!destination!! in the ++SET++ program; hence, ++SET++ is the only way its value may be changed.
(d) the EC will check for %x% only in code segments for which Bx is $true$.

EC.PGM.2.2.3.2 EC Built-in Access Programs

Each EC access program is a built-in constant of the program typeclass. For most programs, the spectype of the program is not named. If the access program has no parameters, its spectype is E1.

EC.PGM.2.3 Local Type Definitions

E1	A built-in specific type of the program typeclass. It is characterized by a single entrance named ENTRANCE1 and a single exit named EXIT1.
pgm-attribute	An ordered pair (!!list!! , !!list!!). The first !!list!! names the entrance(s) to programs of the type; the second !!list!! names the exit(s) of programs of the type.
program	An entity of the program typeclass previously declared via ++DCL_ENTITY++, or a member of a program array previously declared via ++DCL_ARRAY++, or a built-in EC access program, or a program literal. A program literal is a !!constructed program!! as defined in section EC.PGM.1.
rank-pgm-relation	A !!relation!! of user-declared program entities.

EC.PGM.2.4 Dictionary: None.

EC.PGM.2.5 Undesired Event Dictionary

%%inconsistent pgms%%	An assignment was attempted between entities of different attributes; or, one of the entities is not a parameterless program.
%%inconsistent pgm ranking%%	There are user-declared program entities x and y (not necessarily distinct) such that, given the transitivity of the rank relation for programs, both (x,y) and (y,x) are in the relation.
%uninitialized pgm%	An undesired event program variable has been invoked by an EC program, but that program was never given a value by the user.

EC.PGM.2.6 System Generation Parameters: None

EC.PGM.3
PROGRAM INVOCATION FACILITIES

EC.PGM.3.1 Introduction

This module provides mechanisms for invoking programs (either built-in or user-defined) as closed subroutines and, in the former case, passing parameters to those programs.

EC.PGM.3.2 Interface Overview

Syntax

The syntax for invoking a program is as follows:

invocation ::= [pgm parm-list]

For programs with no parameters, the parm-list is empty.

Effects

If an EC run-time access program is named, the effect is that which is specified for that program. If an EC system-generation-time access program is named, there is no run-time effect. If a user-defined program is named, the effect is that of executing the run-time !!command!!s in the !!constructed program!! that has been assigned to the program (either as the initial value as described in EC.PGM.2.2.1, or subsequently as described in EC.PGM.2.2.3) beginning at the entrance named.

Undesired events

The following undesired events apply to program invocation:

%%constant destination%%
%%entrance incorrectly omitted%%
%%not an entrance%%
%%parm wrong type%%
%recursive call%
%%too few parms%%
%%too many parms%%
%%undeclared program%%

CRF 247

EC.PGM.3.3 Local Type Definitions

entrance	The name of an entrance to the program that is the value of the invoked entity, previously specified by ++ENTRANCE++ for that program.
parm-list	::= OR , parm parm-list
parm	An !!actual parameter!! to the program.
pgm	::= program OR (program , entrance) The first form may be used when the program only has one entrance.

EC.PGM.3.4 Dictionary

!!actual parameter!!	An entity that appears in the parameter list of a program invocation. The forms that this may take are specified in EC.DATA.2.4.

EC.PGM.3.5 Undesired Event Dictionary

%%constant destination%%	The user has supplied a constant or a literal as a !!destination!!.
%%entrance incorrectly omitted%%	The user has filed to specify an entrance in an invocation of a program that has more than one entrance.
%%not an entrance%%	The entrance named in the invocation is not an entrance to the program being invoked.
%%parm wrong type%%	The type of an !!actual parameter!! is not of the type called for in the specification of the program.
%recursive call%	A program has invoked itself, either directly, or indirectly through an invocation of a program variable.
%%too few parms%% %%too many parms%%	The programmer supplied a different number of !!actual parameters!! than the number called for by the program's specification.
%%undeclared program%%	Program called or referenced is neither an EC access program nor a program declared by the user.

EC.PGM.3.6 System Generation Parameters: None

EC.SMPH
SYNCHRONIZATION VARIABLES AND OPERATIONS

EC.SMPH.1 Introduction

This module provides a run-time synchronization mechanism, semaphores, with associated operations. They can be used where exclusion regions cannot express the constraints. This mechanism is based on [BELP73]; the semaphore operations are a more primitive version of Dijkstra's P and V [DIJK68].

Semaphores can also be affected by timers; see EC.TIMER.

EC.SMPH.2 Interface Overview

EC.SMPH.2.1 Creating a Semaphore

To create specific semaphore types, use the ++DCL_TYPE++ program specified in EC.DATA.2, with:

p2 = SEMAPHORE;
p3 a semaphore-attribute, defined in EC.SMPH.3;
and the other parameters as described there.

Semaphore entities must be declared before they can be used. Use the ++DCL_ENTITY++ program of Section EC.DATA.2.2, with:

p4 (the initial value) given as an integer literal;
and the other parameters as described there.

To create an array of semaphores, use the ++DCL_ARRAY++ program of EC.DATA.2.2.2, with:

p4 as described there, with initial values given as integer literals;
and the other parameters as described there.

CRF 110 130 154 188 189 258 266

EC.SMPH.2.2 Access programs

Program name	Parm type	Parm Info	Undesired Events
+DOWN+ +UP+	p1:semaphore;IO		%range exceeded%
+SET+	p1:semaphore;I p2:semaphore;O		
+PASS+	p1:semaphore;I		None.

Program effects

In this section, we characterize informally the effects of the synchronization operations. For a more precise description, see the formal specifications in [TRACE]. "State(self)" means the state of the process in which the synchronization operation appears.

Terminology:

Term	Explanation
state	Either "active" or "suspended".
state(self)	the state of the process executing the operation
state(waiters)	the state of all processes in the middle of a +PASS+ operation for that semaphore

Operation	Effect on the integer equivalent of the named semaphore	Effect on process state(s)
+UP+	incremented by 1	if the semaphore gteq 0 then state(waiters) := active*
+PASS+	none	if the semaphore lt 0 then state(self) := waiting
+DOWN+	decremented by 1	none
+SET+	p2 set to value of p1	same as if the value of p2 were arrived at by the smallest possible number of consecutive operations of +UP+ and +DOWN+.

* a change in "state(waiters)" means that all the other processes in pending +PASS+ operations on that semaphore may be made active in an unspecified order. A process that becomes active may make the semaphore negative, causing any other processes in the midst of a +PASS+ to remain in the waiting state. Processes will be activated and complete the pass as long as the semaphore is nonnegative.

EC.SMPH.3 Local Type Definitions

semaphore	A run-time synchronization object created previously by a user program by calling ++DCL ENTITY++; or one of the EC's built-in semaphores listed in EC.INDEX.
semaphore-attribute	An ordered pair of integers specifying the lower bound and upper bound of the type.

EC.SMPH.4 Local dictionary: None

EC.SMPH.5 Undesired Event Dictionary: None

EC.SMPH.6 System Generation Parameters

Parameter	Type	Definition
#max semaphore ascon#	semaphore	Maximum allowable magnitude for a semaphore ascon or literal.
#max semaphore loadcon#	semaphore	Maximum allowable magnitude for a semaphore loadcon.
#max semaphore range#	semaphore	Maximum allowable value for absv(upper bound - lower bound) for a semaphore type.

CRF 119 129 187 209 214

EC.STATE
EXTENDED COMPUTER STATE

EC.STATE.1 Introduction

This module controls and reports transitions between Extended Computer states.

EC.STATE.2 Interface Overview

EC.STATE.2.1 Access programs

Program name	Parm type	Parm info	Undesired Events
+S_FAIL_STATE+	--		None

EC.STATE.2.2 Events signalled by incrementing a semaphore

Event	Semaphore incremented when event occurs
@T(!+power up+!)	ECPOWUP
@T(!+failed state+!)	ECFAILED

Program and Event effects

+S_FAIL_STATE+	The Extended computer enters its failed state, increments ECFAILED, and executes an internal shutdown procedure.
@T(!+failed state+!)	Programmers should assume that when #close down time# has elapsed after this event, no more software actions can occur.
@T(!+power up+!)	The Extended Computer has entered the operating state and is functioning correctly. All demand processes are started.

EC.STATE.3 Local Type Definitions: None.

CRF 129

EC.STATE.4 Dictionary

Term	Definition
!+power up+!	computer is in the operating state and may be assumed to be functioning properly.
!+failed state+!	computer is malfunctioning.

EC.STATE.5 Undesired Event Dictionary: None.

EC.STATE.6 System Generation Parameters:

Parameter	Type	Explanation
#close down time#	timeint	The minimum expected time interval between the moment that the extended computer enters failed state and the moment when no more software actions may occur.

EC.TIMER.1 Introduction

This module provides facilities for measuring real time intervals via timers. A timer is a timeint variable that, when running, will increment or decrement at a rate commensurate with real time.

A timer may be used anywhere a timeint variable may be used, but there are two additional operations, START_TIMER and HALT_TIMER, that may be used. START_TIMER increments or decrements the timer until a limit is reached.

When a timer is declared, the user may choose between timers that increment and timers that decrement, as well as between timers that halt when they reach their limit and timers that "wrap around". The user may also specify a semaphore that will be incremented when the timer reaches its limit.

EC.TIMER.2 Interface Overview

EC.TIMER.2.1 Declaring a Timer

Timers are a numeric type class, as described in EC.DATA.1. To declare specific timer types, use the ++DCL_TYPE++ program specified in EC.DATA.2.1, with:

p2 = TIMER;
p3 a timer-attribute, defined in section EC.TIMER.3;
and the other parameters as described there.

Timer entities must be declared before they can be used. Use the ++DCL_ENTITY++ program of Section EC.DATA.2.2.1, with:

p3 = VAR; and
p4 the initial value given as a timeint literal;
and the other parameters as described there.

To declare an array of timers, use the ++DCL_ARRAY++ program of EC.DATA.2.2.2, with:

p3 = VAR;
p4 as described there, using timeint literals as initial values;
and the other parameters as described there.

CRF 110 154 188 189 238 266

EC.TIMER.2.2 Access programs

Program name	Parm type	Parm Info	Undesired Events
++TIMER_EVENTS++	p1:timer;I p2:semaphore;I	timer name limit value event	None.
+START_TIMER+ +HALT_TIMER+	p1:timer;I	timer name	

EC.TIMER.2.3 Timer tests

Program name	Parm type	Parm Info	Undesired Events
+TEST_TIMER+	p1:boolean;O	!+timer test result+!	None

Program Effects

+HALT_TIMER+ Causes running timer p1 to halt. Halting a non-running timer has no effect.

+START_TIMER+ Causes the value of p1 to be changed in value in real time. The value will be increased or decreased according to the declaration of the specific type to which p1 belongs. According to the declaration of the specific timer type to which p1 belongs, the timer will either stop when it reaches its minimum (maximum) value, or "wraparound"; i.e., continue from its maximum (minimum) value. Starting a running timer has no effect.

++TIMER_EVENTS++ Causes an event to be signalled (by incrementing p2) every time p1 reaches its minimum range value (if p1 is a decrementing timer) or its maximum range value (if p1 is an incrementing timer).

+TEST_TIMER+ Reports the results of the timer hardware tests. If the test is performed periodically or independent of user request, the result given will be that of the most recent test. If the test is performed on request, the command will initiate the test and report the result when the test is complete. It may interfere with normal operation of timers and input/output commands in unpredictable ways.

CRF 129 175 263

EC.TIMER.3 Local Type Definitions

timer	The name of a time-keeping mechanism declared previously by a user program.
timer-attribute	An ordered 5-tuple of the form (timeint, timeint, timeint, HALT/WRAP, UP/DOWN) The first three elements specify the lower bound, upper bound, and minimum !!resolution!!, respectively, of entities of the type. The fourth element is either "HALT" (meaning that the timer should stop when it reaches a limit) or "WRAP" (meaning that the timer should wrap around when it reaches a limit). The fifth element is either "UP" (meaning that the timer increments when started) or "DOWN" (meaning that the timer decrements when started).

EC.TIMER.4 Dictionary

!+timer test result+!	true iff the timer hardware passes built-in test.

EC.TIMER.5 Undesired Event Dictionary: None.

EC.TIMER.6 System Generation Parameters

Parameter	Type	Definition
#max timer error#	real	maximum allowable error rate of all timers, given as a fraction of the time interval measured.
#max timer range#	timeint	maximum allowable value of absv(upper bound - lower bound) for a timer type.
#max timer ranres ratio	real	maximum allowable magnitude of the ratio of a timer type's !!range!! to its !!resolution!!.
#min timer resolution#	timeint	for all timers, the minimum resolution.

CRF 119 175 184 214 263

EC.INDEX INDICES TO THE DOCUMENT

This section provides the following indices to the facilities described in this document:

Access programs

Builtin objects

Events signalled by incrementing a semaphore

Types provided

Dictionary terms

Undesired events

System generation parameters

Reserved words

Access Programs

Access program	Where defined
+ABSV+	EC.DATA
+ADD+	EC.DATA
+AND+	EC.DATA
+B_REAL_2COMP+	EC.DATA
+B_REAL_POSITIVE+	EC.DATA
+B_REAL_SIGNMAG+	EC.DATA
+CAT+	EC.DATA
+COMPLE+	EC.DATA
++D_PROCESS++	EC.PAR.1
++DCL_ARRAY++	EC.DATA
++DCL_DATA_SET++	EC.DATA
++DCL_ENTITY++	EC.DATA
++DCL_TYPE++	EC.DATA
+DISABLE+	EC.IO
+DIV+	EC.DATA
+DOWN+	EC.SMPH
++ENTRANCE++	EC.PGM.1
+EQ+ (bitstring)	EC.DATA
+EQ+ (numeric)	EC.DATA
+ENABLE+	EC.IO
++EXCLUSION++	EC.PAR.2
++EXIT++	EC.PGM.1
+G_SUCCESS+	EC.IO
+GEQ+	EC.DATA
+GT+	EC.DATA
+HALT_TIMER+	EC.TIMER
+LEQ+	EC.DATA
+LT+	EC.DATA
+MINUS+	EC.DATA
+MUL+	EC.DATA
+NAND+	EC.DATA
+NEQ+ (bitstring)	EC.DATA
+NEQ+ (numeric)	EC.DATA
+NOT+	EC.DATA
+OR+	EC.DATA
++P_PROCESS++	EC.PAR.1
+PASS+	EC.SMPH
+R_BITS_2COMP+	EC.DATA
+R_BITS_POSTIIVE+	EC.DATA
+R_BITS_SIGNMAG+	EC.DATA
+R_TIME_HOUR+	EC.DATA
+R_TIME_MIN+	EC.DATA
+R_TIME_MS+	EC.DATA

CRF 167 232 247 266

Access Programs (continued)

Access program	Where defined
+R_TIME_SEC+	EC.DATA
++RANK_DATA++	EC.DATA
++RANK_DATA_SET++	EC.DATA
++RANK_PGM++	EC.PGM.2
++REGION++	EC.PAR.2
+REPLC+	EC.DATA
+S_FAIL_STATE+	EC.STATE
+SET+	EC.DATA, EC.SMPH, EC.PGM.2
++SET++	EC.DATA, EC.PGM.2
+SHIFT+	EC.DATA
+SIGN+	EC.DATA
+START_TIMER+	EC.TIMER
+SUB+	EC.DATA
+T_REAL_HOUR+	EC.DATA
+T_REAL_MIN+	EC.DATA
+T_REAL_MS+	EC.DATA
+T_REAL_SEC+	EC.DATA
+TEST_AC+	EC.IO
+TEST_CSA+	EC.IO
+TEST_CSB+	EC.IO
+TEST_DC+	EC.IO
+TEST_DIOW1+	EC.IO
+TEST_DIOW2+	EC.IO
+TEST_DIOW3+	EC.IO
+TEST_INTERRUPTS+	EC.PAR.1
+TEST_MEMORY+	EC.MEM
+TEST_TIMER+	EC.TIMER
+TEST_XACC+	EC.IO
+TEST_YACC+	EC.IO
+TEST_ZACC+	EC.IO
++TIMER_EVENTS++	EC.TIMER
+UP+	EC.SMPH
+XOR+	EC.DATA

CRF 097 118 167 182 207 232 257 266

Builtin Objects

Name	Type of object	Where defined
DIV_FAIL	program	EC.DATA.2.6.2
ECFAILED	semaphore	EC.STATE
ECPOWUP	semaphore	EC.STATE
ENBLSEM	semaphore	EC.IO
ENTSWSEM	semaphore	EC.IO
KBINTSEM	semaphore	EC.IO
MARKSEM	semaphore	EC.IO
NEXT_PERIOD	semaphore	EC.PAR.1
REG	data object	EC.DATA.1.4
All i/o data items	bitstring	EC.IO
All undesired events	program	EC.PGM.2
All EC access programs	program	EC.PGM.2
Boolean UE variables	boolean	EC.PGM.2

Events Signalled by Incrementing a Semaphore

Event	Semaphore	Where defined
@T(!+/ENTERSW/ occurred+!)	ENTSWSEM	EC.IO (Appendix 5)
@T(!+failed state+!)	ECFAILED	EC.STATE
@T(!+/KBDENBL/ occurred+!)	ENBLSEM	EC.IO (Appendix 5)
@T(!+/KBDINT/ ready+!)	KBINTSEM	EC.IO (Appendix 5)
@T(!+/MARKSW/ occurred+!)	MARKSEM	EC.IO (Appendix 5)
@T(!+power up+!)	ECPOWUP	EC.STATE

In addition, users may request timer-related events by supplying their own semaphores. See EC.TIMER

CRF 182 232 247 265

Types Provided

Type name	Where defined
array-init	EC.DATA
attribute	EC.DATA
binding	EC.DATA
bitstring	EC.DATA
boolean	EC.DATA
convar	EC.DATA
data_set_reln	EC.DATA
dataitem	EC.IO
E1	EC.PGM.1
entrance	EC.PGM.3
exclusion-relation	EC.PAR.2
indexset	EC.DATA
integer	EC.DATA
invocation	EC.PAR.1
name	EC.DATA
parm	EC.PGM.3
parm-list	EC.PGM.3
pgm-attribute	EC.PGM.2
pointer	EC.DATA
program	EC.PGM.2
rank-data-relation	EC.DATA
rank-pgm-relation	EC.PGM.2
real	EC.DATA
semaphore	EC.SMPH
semaphore-attribute	EC.SMPH
spectype	EC.DATA
statement-list	EC.PAR.2
timeint	EC.DATA
timer	EC.TIMER
timer-attribute	EC.TIMER
typeclass	EC.DATA
version	EC.DATA

In addition, Appendix 5 lists a set of builtin bitstring spectypes.

CRF 115 118 181 182 195 247 266

Dictionary Terms

Term	Where defined
!!actual parameter!!	EC.PGM.3
!!command!!	EC.PGM.1
!!constructed program!!	EC.PGM.1
!+deadline+!	EC.PAR.1
!+destination+!	EC.DATA
!!destination!!	EC.DATA
!+/ENTERSW/ occurred+!	EC.IO
!+failed state+!	EC.STATE
!+fall back value+!	EC.DATA
!!hardest attributes!!	EC.DATA
!+interrupt test result+!	EC.PAR.1
!!invocation!!	EC.PGM.1
!+io test result+!	EC.IO
!+i/o success+!	EC.IO
!+/KBDENBL/ occurred+!	EC.IO
!+/KBDINT/ ready+!	EC.IO
!!list!!	EC.DATA
!+/MARKSW/ occurred+!	EC.IO
!+max CPU time req+!	EC.PAR.1
!+max div result+!	EC.DATA
!+memory test result+!	EC.MEM
!+on/off+!	EC.PAR.1
!+period+!	EC.PAR.1
!+power up+!	EC.STATE
!+radix pt ident+!	EC.DATA
!!range!!	EC.DATA
!!relation!!	EC.DATA
!!resolution!!	EC.DATA
!+source+!	EC.DATA
!!source!!	EC.DATA
!+starting event+!	EC.PAR.1
!+timer test result+!	EC.TIMER
!+user threshold+!	EC.DATA

CRF 086 154 182 206 247 263 265 267

Undesired Events

UE name	Where defined	Also appears in
%already disabled%	EC.IO	
%already enabled%	EC.IO	
%assertion violation%	EC.DATA	
%%attribute not allowed%%	EC.DATA	
%%attribute not given%%	EC.DATA	
%%constant destination%%	EC.PGM.3	
%%dest unknown%%	EC.PGM.1	
%divide by zero%	EC.DATA	
%%duplicate name%%	EC.PGM.1	
%%entrance incorrectly omitted%%	EC.PGM.3	
%illegal array index%	EC.DATA	
%%illegal exit-list%%	EC.PGM.1	
%%illegal index set%%	EC.DATA	
%%illegal ptr target%%	EC.DATA	
%illegal round/trunc%	EC.DATA	
%%illegal synch%%	EC.PAR.1	
%%inappropriate attributes%%	EC.DATA	
%%inconsistent data ranking%%	EC.DATA	
%inconsistent lengths%	EC.DATA	
%%inconsistent pgm ranking%%	EC.PGM.2	
%%inconsistent pgms%%	EC.PGM.2	
%%inconsistent register access%%	EC.DATA	
%inconsistent time parms%	EC.PAR.1	
%%index not allowed%%	EC.DATA	
%left truncation%	EC.DATA	
%%length too great%%	EC.DATA	
%%literal or ascon too big%%	EC.DATA	
%%loadcon too big%%	EC.DATA	
%max CPU time exceeded%	EC.PAR.1	
%mdr outside range%	EC.DATA	
%missed deadline%	EC.PAR.1	
%%multiple qualifiers%%	EC.DATA	
%%name in use%%	EC.DATA	EC.PAR.2, EC.PGM.1
%%no such entrance%%	EC.PGM.1	
%nonexistent position%	EC.DATA	

CRF 102 103 107 111 119 123 127 128 143 144
145 147 158 169 170 173 182 183 186 209
221 240 247 262 264

Undesired Events (continued)

UE name	Where defined	Also appears in
%%not an entrance%%	EC.PGM.3	
%%not an exit%%	EC.PGM.1	
%nowhere to go%	EC.PGM.1	
%%parm wrong type%%	EC.PGM.3	
%%port defined twice%%	EC.PGM.1	
%%port not defined%%	EC.PGM.1	
%range exceeded%	EC.DATA	EC.SMPH
%%range too great%%	EC.DATA	
%%ranres too great%%	EC.DATA	
%%read/write-only violation%%	EC.IO	
%read-write violation%	EC.IO	
%recursive call%	EC.PGM.3	
%%REG not allowed%%	EC.DATA	
%%res too fine%%	EC.DATA	
%%subrange not allowed%%	EC.DATA	
%%too few parms%%	EC.PGM.3	
%%too many parms%%	EC.PGM.3	
%%undeclared operand%%	EC.DATA	
%%undeclared program%%	EC.PGM.3	EC.PAR.1
%%undeclared region%%	EC.PAR.2	
%%undeclared spectype%%	EC.DATA	
%%unimplemented attribute via variables%%	Appendix 4	
%%unimplemented binding%%	Appendix 4	
%%unimplemented disabling%%	Appendix 4	
%%unimplemented EXACT_REP resolution%%	Appendix 4	
%%unimplemented multi-exit EC access program%%	Appendix 4	
%%unimplemented pgm ptr%%	Appendix 4	
%%unimplemented variable period%%	Appendix 4	
%%unimplemented variable shift length%%	Appendix 4	
%%unimplemented variable substring%%	Appendix 4	
%%unimplemented varying EXACT_REP%%	Appendix 4	
%unititialized pgm%	EC.PGM.2	
%%unknown initial value%%	EC.DATA	
%%unknown operand in attributes%%	EC.DATA	
%%variable parm%%	EC.DATA	
%%variable timing parms%%	EC.PAR.1	
%%varying constant%%	EC.DATA	
%%wrong init value size%%	EC.DATA	
%%wrong init value type%%	EC.DATA	

CRF 104 107 119 165 172 175 198 205 209
212 232 247 249 252 259

System Generation Parameters

Parameter name	Data type	Where defined
#close down time#	timeint	EC.STATE
#max bits length#	integer	EC.DATA
#max i/o time (data item name)#	timeint	EC.IO
#max real ascon#	real	EC.DATA
#max real loadcon#	real	EC.DATA
#max real range#	real	EC.DATA
#max real ranres ratio#	real	EC.DATA
#max semaphore ascon#	semaphore	EC.SMPH
#max semaphore loadcon#	semaphore	EC.SMPH
#max semaphore range#	semaphore	EC.SMPH
#max timeint ascon#	timeint	EC.DATA
#max timeint loadcon#	timeint	EC.DATA
#max timeint range#	timeint	EC.DATA
#max timeint ranres ratio#	real	EC.DATA
#max timer error#	real	EC.TIMER
#max timer range#	timeint	EC.TIMER
#max timer ranres ratio#	real	EC.TIMER
#min real resolution#	real	EC.DATA
#min timeint resolution#	timeint	EC.DATA
#min timer resolution#	timeint	EC.TIMER
#nbr fltrec elements#	integer	EC.IO

CRF 209 221 222 265

Reserved Words

Word	Type of word	Where defined
ASCON	Enumerated type value	EC.DATA
BITS	Typeclass	EC.DATA
BOOLEAN	Builtin spectype	EC.DATA
DEREF	Keyword in pointer use	EC.DATA
DOWN	Attribute keyword	EC.TIMER
EXACT_REP	Attribute keyword	EC.DATA
EXIT1	Exit of certain EC programs	EC.PGM.2
$false$	Boolean value	EC.DATA
FIX	Enumerated type value	EC.DATA
HALT	Attribute keyword	EC.TIMER
INTEGER	Typeclass	EC.DATA
LOADCON	Enumerated type value	EC.DATA
NOSTORE	!!destination!! identifier	EC.DATA
PGM	Typeclass	EC.PGM.2
PTR	Typeclass	EC.DATA
REAL	Typeclass	EC.DATA
REF	Keyword in pointer literal	EC.DATA
ROUND	Operand keyword	EC.DATA
-SAVE	Keyword	EC.DATA.1.4
SEMAPHORE	Typeclass	EC.SMPH
TIMEINT	Typeclass	EC.DATA
TIMER	Typeclass	EC.TIMER
$true$	Boolean value	EC.DATA
TRUNC	Operand keyword	EC.DATA
UP	Attribute keyword	EC.TIMER
VAR	Enumerated type value	EC.DATA
VARY	Enumerated type value	EC.DATA
WRAP	Attribute keyword	EC.TIMER

CRF 157 181 182 184 195 239 247

APPENDIX 1

INTERFACE DESIGN ISSUES

EC.DATA

1. We decided to give the programmer some control over the register, so that he could take care of reducing register loads and stores by being careful with the order of operations. The alternatives we considered were notations much closer to high-level programming languages. These notations make complex expressions easier to read, but require a more sophisticated translator if we are to make efficient use of registers.

2. There is a danger with fixed point division that the results will be meaningless; this problem occurs when the numerator has more significance than the denominator. An assembly language programmer has some information that he uses to avoid this danger. The only way we can get this information is to ask the programmer to provide it, since it is dependent on the context and meaning of the division.

3. Two ways were proposed for user programs to indicate the radix of the number for a bitstring-real conversion:

   a. by giving an integer literal "i" such that the rightmost bit of the bitstring represents 2 raised to the ith power;

   b. by giving an integer literal "i" such that i is the number of the bit immediately to the right of the radix pt.

   Alternative 2 most closely resembles the scaling notation used in the current program, but we chose alternative 1 because most designers felt that it was easier for newcomers to understand and remember.

4. There are two main reasons for including variables whose attributes may vary:

   a) they can be reused at different points in a computation, thereby reducing the amount of space that must be reserved;

   b) they allow the same code to be used to manipulate values in widely differing ranges.

5. We require the programmer to specify a type for results stored into and retrieved from variables. We considered permitting, but not requiring, specification of the type of intermediate results and letting the Extended Computer determine the specific type when the programmer omitted the specification. We ruled out this alternative because it requires a run-time support package to keep track of the specific types of varying-type variables.

6. We considered several alternatives for providing registers:

   a) Having a common register for all type classes. This register can be very simply mapped to the accumulator.

   b) Having a separate register for each type class, implementing them with the single accumulator, and leaving the problem of interference between them up to the programmer. This was originally accepted because it is the simplest alternative that provides type checking for results in the register. However it gives away the underlying limitation, and imposes restrictions on the programmer that would not be needed if the underlying hardware had more registers or if there was multi-processor hardware.

   c) Having a separate register for each type class, implementing them with the single accumulator, and completely automating the problem of interference between the registers, freeing the programmer from any concern about it. This could be done by saving and restoring the accumulator contents whenever a different register is used. While it would be the most convenient alternative, the overhead would be prohibitive.

   d) Having a separate register for each type class, implementing them with the single accumulator, and partially automating the problem of interference between the registers. The programmer would have to indicate when he wants to reuse results in a particular register and when he does not care.

   We chose alternative (a) because it is the simplest and treats a register as a variable with varying attributes.

7. We felt it important that the EC implementation avoid saving contents of a register if they would never be needed and therefore put that burden on the programmer rather than try to do register usage analysis. We considered several ways to allow the programmer to specify whether or not the value in the register would be needed again. Among them:
   a. Associate the information with the name of the register.
   b. Associate the information with the name of the operation.

   We chose (b) because we did not want to have two names for the same object. Further, it allows us to localize the information in a place related to the operations (of which it is a property) rather than the registers.

8. An earlier version of this interface included operations such as squareroot, exponentiation, log, and root-sum-squared. We decided to move these operations to another module because they can be implemented in a machine-independent fashion. These concerns do not belong in the Extended Computer.

9. An earlier version of this module had two bitstring sizes, corresponding to halfwords and fullwords on the target computer. We then decided to have only one size because it results in a simpler data type. We finally decided to have bitstrings of any size because we noted that insisting on a fixed but unknown size made it difficult to write efficient but machine independent code. The present choice makes the interface unbiased with respect to word length and puts the burden for effective use of the actual hardware on the implementor of the EC.

10. We considered specifying bitstring sub ranges in terms of (starting point, length) instead of (starting point, ending point). One parameter fewer would be needed on bitstring compares and transfers, and we could avoid the unmatching lengths undesired event. However, we found that people working with bitstrings find it easier to work by identifying the boundary bits.

11. We considered having the EC monitor arithmetic operations for excessive loss of significance but decided that this was a programmer responsibility and could be done in a machine independent way. This eliminated the undesired event %too much lost significance%.

12. We considered relegating time to the application data type module and implementing it in terms of reals. We chose to include it in the EC because the concept of time is basic to the specification and implementation of real-time processes in the EC and because the representation should be that used in the hardware timers.

13. We considered allowing array declarations to be shared by several variables. We found this not particularly useful unless one has operations that take whole arrays as operands.

14. We decided not to allow array elements to be structures. We lose the ability to have arrays of arrays, but if this were necessary, it could be implemented in a machine independent way and could be provided by some other module.

15. We considered allowing index sets to be more general, but this seemed unnecessary even for future extensions. Such extensions could be done using the present arrays and the extension would be machine independent. We also considered restricting the lower array bound to be either 0 or 1. This seemed unnecessarily restrictive, especially as it may be desirable to select array indices at sysgen time.

16. We considered fixing the value of the array index set at declaration time, system generation time, or run time. Declaration time is too restrictive; it is sometimes useful for the array index set to be a system generation parameter. Run time fixing requires dynamic storage allocation, which is not needed or practical for avionics applications.

17. We rejected the option of operations that apply to arrays as units, e.g. multiplying arrays by scalars or arrays by arrays. Such operations depend on mathematical algorithms, rather than on characteristics of the computer and can be implemented in a machine-independent way. The present design is the simplest way to hide the hardware addressing mechanism. Extentions can be provided by user programs.

18. We considered not allowing arrays of variables whose attributes vary at run-time as it might simplify the implementation if all elements had the same attributes at all times. Although the implementation of arrays with varying attributes will probably be less efficient than arrays of fixed attribute elements, this feature is occasionally needed.

19. More than one reviewer asked if the Extended Computer shouldn't provide stacks as a builtin data structure. If we need stacks, they can be provided using the current EC facilities. The interface to those facilities (probably in the ADT module) would be carefully modelled after the EC. Should we transfer to stack machine, we could move the interface into the EC, and user programs would not have to change. This rationale also applies to floating point arithmetic, multi-dimensional arrays, array operators, etc.

20. Entity names are global in the EC. This is because that is what avionics computers provide; one can limit the scope of a name (if desired) in a machine-independent way (e.g., using naming conventions, or a pre-processor).

21. We recognize the need to represent data most efficiently for the operations in which it will be used. Since only users can determine how a datum will be used, the best the EC can do is provide a menu of representations and tell the users what each one is best and worst at doing. Hence, the "version" attribute in specific types.

22. We considered allowing non-homogenous arrays; that is, arrays with different specific types. However, that would mean that if an operand was a member of such an array, we couldn't discover its type until run-time. Because of this great run-time penalty, we deleted the capability.

23. The +SHIFT+ program used to take a list of bitstrings as its input parameter; it shifted the concatenation of the list. The idea was to allow the same kind of shifting that occurs in the TC-2 between adjacent registers. However, it became clear that the implementation of such a feature would simply +CAT+ the strings together first anyway, and shifting the result, and we would gain no efficiency. So for consistency, +SHIFT+ now takes a single source parameter.

24. We have made the decision to design the EC to be implemented on a machine that supports fixed-point arithmetic and not floating-point arithmetic. The resolution required in a numeric variable is expressed in our machine as a constant, independent of the magnitude of the value of the variable. In fixed-point represenatations, there is a uniform distance between values that can be represented exactly. In floating-point, the distance is small for small values, large for large values.

    Were we to go to a floating-point machine, we would need to enhance the interface, because fixed-distance representations on a floating-point machine would be very inefficient. We would let the user specify a worst-case distance between representatives as a fraction of the value of the variable. The implementation would choose a representation so that the mantissa of the number had a resolution (in the current sense) less than the fraction.

    The present interface may be considered a subset of a more complete interface in which we let the user specify resolution either in absolute form (in which case a fixed-point representation would probably be chosen) or in relative form (in which case we would represent the number using the floating-point hardware, or by simulating floating-point). The present interface reflects our decision to make EC apply to typical avionics machines, which are fixed-point.

25. The pointer typeclass is a recent addition, included when we realized that we had denied users the capability found in all von Neumann machines of indirect referencing, or postponing operand specification until run-time. Since the goal of EC was to abstract from the idiosyncrasies of particular avionics computers, yet provide the capabilities that they provide, this was clearly an appropriate addition to the EC architecture.

26. When declaring a specific type, it used to be an error to specify a version that did not exist for that particular type. We now say merely that the EC will pick one of its own choosing in that case. That is so that should an Application Data Types type ever migrate into the EC because of a change that enhances the type repertoire of the hardware, we would like for user programs to remain correct. However, the versions that we would provide in the EC for that type might be entirely different than what were provided for it in the ADT.

EC.IO

1. We considered five alternatives for handling retries of unsuccesful I/O operations:

   1) having two different commands for these two cases: one that retries, either once or until it succeeds, and one that instead of retrying returns a failure indicator;
   2) having a parameter on the command specifying how often to retry, and having the command return a failure indicator;
   3) having a failure indicator, and having the user program try again if it needs to retry transmission;
   4) having a special "retry" command, with a label operand, which the user can call to have the I/O command with the specified label retried.
   5) omitting the failure indicator for the data items where it is not currently used.

   The first and fourth alternatives yield a more complicated interface than the third and provide no extra capability. The second results in extra (non-machine dependent) programs in the EC. The fifth alternative would build knowledge of the application into the EC. The third alternative relegates decisions about retrying to the user programs, and we chose this one.

3. We have considered four alternatives for handling the discrete inputs and outputs.

   Alternative 1: Treating input and output differently, allowing user programs to use a READ command to read in entire discrete input words, but providing a special WRITEBOOL command so that user programs could write individual bits appearing in the discrete output words.

   Alternative 2: Adding a READBOOL command that would read in a discrete input word, pick out the bit for a particular discrete input data item, and return it as a boolean value. Alternative 2 was rejected because not all the data items in discrete input words have boolean values. For example, /IMSMODE/ has five values, one for each switch position.

   Alternative 3: Provide the user programs with a way to specify a range of bits within both a discrete output word that they want to write out and within an input word, so that they can request individual discrete inputs in a symmetrical fashion. Alternative 3 leaves some of the responsibility for non-interference between discrete outputs to the device interface modules, since they must specify the correct ranges.

Current: All of the above alternatives were based on the decision that the EC would sometimes identify outputs and inputs by class name rather than the individual data item name. This was done both for efficiency reasons and because it was believed that knowledge of the location of a data item within a discrete input or output word was device dependent rather than computer dependent. A much more consistent interface is achieved by always using the data item name. The EC implementer is now responsible for knowing the identity of a TC-2 I/O item, but not responsible for knowing its meaning. The efficiency problems are resolved by allowing a single command to take a list of parameters so that the EC implementation may perform operations to a single I/O word simultaneously rather than sequentially. This also eliminates special treatment of double data items.

4. We originally designed the reading of intermittent data with an access function that indicated whether or not the data were available and an undesired event if a user program tried an intermittent read operation when the data were not available. This seemed dangerous, since a slight timing difference could cause an undesired event, and the user programs could not avoid the UE. Instead, we have chosen to allow the read command at any time. If the data are not available, the success indicator returns false. This is consistent with our general policy that it should be possible to avoid UEs by correct programming.

   Because the intermittent data is read just like any other, we decided not to have a separate command name for it.

5. We considered having serial inputs identified by class names rather than by individual data item name. Interpretation of the identification bits was considered the responsibility of the associated device interface module. We decided that identification of the data item is an EC responsibility, but interpretation of the item remains the responsibility of the DIM.

6. Note that sometimes an output should go to more than one data item. We originally handled this by letting users repeat sets of parameters to i/o commands. and saying that the order was unspecified. Since we no longer have i/o commands per se, but rather use assignment (and other bitstring) operations, we have expanded our general assignment statement so that many sources and many destinations can be given at once; the assignment happens in an unspecified order.

7. We promise that an output transmission will occur when an enabled output data item is used as a destination. We do not say when an input transmission will occur. This is because we can get away with it in the latter case, but not in the former (because an output transmission has visible effects). We hide when input takes place because someday there may be direct-memory-access input, and the computer really won't be able to control when an input item changes value.

8. We did not include the names of the data items in the main document, because we wanted to emphasize the fact that the architecture of the Extended Computer's i/o operations doesn't depend on those particular names. If the design of the Extended Computer were used with the TC-2 for some other application, the names of the data items would not be part of the technology transferred.

9. We chose special names for the data items' bitstring spectypes because we felt that the representation for each was likely to change in the event of a device replacement, and probably wouldn't be the same as that of a non-data-item bitstring anyway. For instance, we might choose to represent "normal" bitstrings as contiguous and left-justified within a word, but we clearly don't have this option with most of the data items (see //FPANGL//, for instance).

10. The signal converter is tested by sending particular values to it and then reading back the results of the internal signal converter manipulation on the values. The proper relationship between the values sent out and the value read in can be characterized by a set of equations. The design issue is how much of the knowledge should be hidden within this module: both the equations and the choice of test values, just the equations, or neither. The equations are based on the behavior of the channel, and therefore belong within this module. The choice of values could be considered part of the software requirements; they affect the displays seen by the pilot, and are documented in section 4 of the requirements. However, the choice of these particular values is partly influenced by hardware characteristics. Further, if they are not hidden, the interface to this module becomes much more complex. We have chosen to hide all of the information even though it means hiding some details about the required functions in this interface. We assume that the test values are likely to change with the hardware and not for any other reason.

11. We decided to hold user programs responsible for avoiding interference between the diagnostics and the regular commands rather than build monitors into the I/O commands and diagnostics. The diagnostics are not expected to be run when the software is doing anything else. Monitors impose a run-time cost in the regular commands.

EC.MEM

1. We considered dividing memory into banks that would be tested separately, allowing partial rather than complete shutdown. We decided not to do so at this time because the system lacks the ability to exploit it and we could do so easily in the future.

2. A previous design implied that invoking the access program associated with a test actually started the test. Because future computers may have tests ongoing, or running in the background, we changed our design to indicate that invoking a program merely returns the most recent result of that test. If a future computer is required to start a test at a certain time, we can add start-test commands later. Returning the value may take a substantial amount of time in some cases. The major change this caused was in the case of the memory test. Before, there was a command to start the test, an event signalled when it was done, and a program to retrieve the result. The motive was that the invoking program would want to do something else while waiting for the test to be completed. However, some program would have to wait idly for the event to occur anyway, and so we lose nothing by letting the memory test program just take a long time to return. We gain a uniform interface, with no special cases.

Other design issues dealing with the hidden portion of the interface are contained in [VM].

EC.PAR.1

1. In earlier designs of this interface, timing constraints were associated with specially designated blocks, implying that these blocks were the scheduling units. The process mechanism was unnecessarily complicated, put too many restrictions on the internal structure of processes, and gave away more information than the one here.

2. We considered having START and STOP commands so that one process can explicitly affect the ready/waiting state of another process. The problem with a STOP command is that a process cannot be safely stopped at any arbitrary point. We fixed this by adding "homing points", but specifying homing points also cluttered up the algorithm descriptions. So we dropped the idea, relying on more conventional synchronization mechanisms instead.

3. Earlier versions made an outer "do forever" loop implicit. Thus the process would execute a "INIT" block once whenever the process was started and then repeatedly execute a "FREQ" block until the process was stopped. We have decided not to include an implicit loop because we do not want to limit the internal structure of the processes. Also, the process would be easier to read if all the control was shown explicitly. Process bodies can now be specified just as subprogram bodies are, making the overall specifications of the Extended Computer simpler.

4. At one point we had intermittent processes wait for a start event and then run until a stop condition existed. We found it simpler to define a single boolean and have the process pass its start point only when the boolean was true. This eliminated the need for the event interface in the EC and eliminated ambiguous cases such as the start event ocurring when the stop condition held.

5. At one point we had a special class of processes called init processes. We recognized these as a special case of Demand processes and decided to simplify the interface by exploiting that fact. This allows some processes to be used both as init processes and under other conditions.

6. It is possible for a programmer to write a process that runs out of statements to execute. We considered three alternatives:

   a) Stating that it is an undesired event for a process to finish, i.e., making it a requirement that each process contain an infinite loop;
   b) Assuming that a completed process is in the ready state, but that it has a null statement list to execute if it becomes running;

c) Assuming that a completed process is in a waiting state, waiting for an event that will never occur.

We rejected a) because it builds too much information into the Extended Computer and it is an unnecessary restriction. We rejected b) because there is no point in having a completed process compete for a processor. Alternative 3 is a reasonable compromise for the Extended Computer interface. If it is considered undesirable to have completed processes, this should be prohibited by programming conventions.

7. We have decided not to include relative priorities for the different processes because fixed priorities do not generally work when there are real-time constraints.

8. In an earlier version, we had no distinction between periodic and demand processes because a periodic process can be viewed as one that waits for a particular stimulus, i.e., the passage of a particular amount of time. However, one of the timing parameters needed for periodic processes is not useful for demand processes. In addition, periodic processes must have restrictions on the synchronization operators within the periodic loop because the indeterminate wait associated with synchronization operators makes it difficult to prove that the loop can be scheduled regularly as required.

9. In an earlier design, we did not explicitly distinguish intermittent periodic processes. We now distinguish them in order to increase the likelihood that we can take advantage of the intermittency in the scheduling of processes. Earlier we distinguished them by calling them intermittent, now we use the presence of the optional ON_OFF to distinguish them.

10. We considered specifying periodic processes in terms of frequency rather than in terms of time intervals. Because we wanted to specify the deadline as an interval, we decided it would be more straightforward to use two intervals. These two parameters adequately constrain the variations in regularity.

11. We have an undesired event assumptions that says there won't be too many demand processes for a periodic process to miss its deadline. The assumption is worded with that orientation because it is impossible to tell how often a demand process must run.

12. We used to allow the body of a process to be any statement list. We now restrict it to a call on a previously-declared program. In this way we maintain a clear distinction between process and program, and therefore allow future extensions to include run-time creation of processes without run-time creation of programs, vice versa. The restriction does not restrict what we can do with the current version; it merely paves the way for future extensions.

13. Previously, there was a parameter in the process definitions with which the user specified the maximum CPU time required by his process. We removed this from the high-level EC interface because (a) the user doesn't have enough information to provide it; (b) the information is machine-dependent; and (c) the length of time an operation takes can vary greatly, depending on the storage and representation of the operands, for example. This is information is now provided to the EC implementation at a lower level, where the processes are divided into scheduling blocks; it may be provided by software that examines the source code, or it may be done manually.

EC.PAR.2

1. Regions with an exclusion relation were selected for Extended Computer synchronization primitives because
   a. they allow concurrency constraints to be expressed directly rather than as an implication of run time synchronization;
   b. they express the exclusion relationships in a form that can be interpreted efficiently by a pre-run-time scheduler;
   c. there is an algorithm for generating run-time synchronization from the exclusion relations;

   This is the simplest acceptable alternative. Rejected alternatives included:
   a. disabling interruption: once an identified section of code starts executing, it must run to completion. This alternative was rejected because it is prejudiced toward a single processor: it overly restricts the parallelism by stating that no other actions can be taken simultaneously with the code section, rather than specifying which other actions may not be taken;
   b. simple mutual exclusion: specifies all exclusion relations as equivalent, i.e., a section of code that excludes any other excludes all others. This alternative still places too many restrictions on the parallelism because many of the identified code sections need not exclude each other.
   c. named regions with mutual exclusion. Rejected because it assumes that the exclusion relation is symmetric.
   d. exclusion via synchronization primitives: using synchronization primitives such as those in EC.SMPH to effect mutual exclusion. Rejected because (1) synchronization primitives that are being used for other interprocess synchronization or communication purposes cannot be distinguished from those used for exclusion without additional commentary, (2) the exclusion requirements are implicit in a solution based on synchronization primitives, rather than stated explicitly as they can be with identifiable regions, and (3) the exclusion information (implying scheduling constraints) is embedded in and scattered throughout the text. These properties of the synchronism primitives make it difficult to do pre-run-time scheduling without substantial preprocessing.

2. Many useful forms of synchronization were rejected for the Extended Computer because they do not depend on the implementation of parallel process. Application-oriented synchronization operations may be developed using the exclusion relations, and semaphores.

3. Can a region be excluded from itself? Is that useful? Yes, because in the case of non-reentrant code, this is how we will probably prevent disastrous re-invocations.

EC.PGM.1

NOTE: Design issues 1 through 7 refer to a previous control structure we had included in the EC which is documented informally in [APC]. That control structure is no longer a part of EC, because we concluded that it required too much sophistication in implementation, and that simpler constructs hid the hardware characteristics just as well.

1. Alternatives considered for the syntax of a guard are shown below.

   a. Boolean variables or constants only. All the boolean variables must be assigned values before the limited program is executed.
   b. Any sequence of statements assigning a boolean value to a special guarded command register.
   c. Allowing a limited program list as a guard.
   d. Allowing a program to define the value of a guard (defined guards).
   e. All of the above.

   Discussion: We chose (e). The semantics can easily be defined formally [ITTI2]. Defined guards save code by avoiding duplication of statement lists, which would otherwise be required because of syntactic limitations.

2. We chose to have the Extended Computer provide the IT-TI construct rather than the more common IF-THEN-ELSE, CASE, and DO-WHILE constructs because IT-TI serves for all purposes. It allows some programs to be written as one loop that would otherwise require several, thereby saving variables and predicate evaluation. IT-TI has a mathematical semantics that allows systematic construction of the program's function [ITTI1].

3. Dijkstra's guarded commands are nondeterministic; of the true guards, only one is selected, but there are no rules defining which one is selected. We chose a deterministic construct because they allow simpler guards.

4. We considered providing a FOR command (FOR I = 1 to 10 DO...) but decided against it because
   a. the same purpose can be served with the IT-TI command;
   b. many special cases and questions arise with the FOR command.

5. We considered having an implicit final LP of the form (true,SKIP). We decided not to do this in order to encourage the programmer to consider every case carefully.

6. Should statement lists be allowed to contain declarations, and what is their scope? We decided that it was harmless (from this module's point of view) to allow it. The scope of all declarations is global and items must be declared before they are used, but these are issues belonging to the EC submodules that provide the declarations.

7. In an earlier version we allowed Dijkstra's cor and cand. We have eliminated them because the same effect can be obtained with the use of defined guards.

EC.PGM.2

NOTE: Design issues 1 through 3 refer to a design that allowed user-defined programs to have parameters. Because the semantics of parameter passing do not depend upon the hardware, we concluded that users could emplov other facilities to achieve a parameter protocol (e.g., assignment before and after a program call).

1. Should actual and formal parameters be specified by type class, specific type name, or using type attributes such as range and resolution? We decided that type agreement should depend on the specification chosen by the programmer. Other alternatives would force us to write separate programs for each specific type or to include a parameter passing mechanism that would be more general than needed for most cases.

2. We added PARM_GIVEN because programs must be able to tell if an optional parameter was supplied or not. We thought about making it a built-in value that any type of variable could take on; then programmers could ask, e.g., if p1=PARM_GIVEN. However, because output parameters can be optional, we didn't want programmers checking their "value".

3. Programmers need not supply trailing commas when optional parameters at the end of a parameter list are omitted. That is, instead of +pgm1+(a,b,,,) one may write +pgm1+(a,b). Besides the obvious convenience, this will allow us to add optional parameters to the end of any access program parameter list, yet not force all calls on that program to change.

4. We added the feature of ranking programs' access speed because the current computer has the capability of doing fast subroutine linkages in certain areas of memory. Because a replacement machine may not have such a capability, we made the relationship "not-slower-than", which we can trivially implement by doing nothing. We make no firm promise about the ordering, however, because we recognize that we cannot make access to a subroutine not-slower-than access to an expanded macro that simply lives in-line.

5. We considered giving the user the ability to specify whether a program was to be invoked by subroutine linkage or in-line (macro) expansion. Howver, we realized that macro-expansion can be done independently of the host machine, and hence is not an appropriate EC facility.

EC.SMPH

1. We originally had more complex synchronization operators that met many immediate demands of our application. As we prefer the Extended Computer to be as application-independent as possible, we chose synchronization operations for the Extended Computer primitives that would be as simple as possible, but that could be used as building blocks for more specialized synchronization operators. For the more complex synchronization operations, see the specifications of the Application Data Type module [ADT].

   All of the following alternatives for the Extended Computer synchronization operations were rejected either because they are more complex than the operations selected or because they can be built, given the operations selected.

   a. P and V operations on semaphores;

   b. eventcounts [REED79]: Also rejected because we weren't sure we would need them;

   c. P and V supplemented by eventcounts;

   d. UP, DOWN, and PASS supplemented by event variables. A simple generalization of event variables, event-booleans can be implemented in the Application Data Type module in a machine independent way.

   e. V, DOWN, PASS, and eventcounts.

2. At one point, we provided a semaphore-to-integer conversion program. These were deleted when we could think of no reason to use it. If such a need arises, it would be a straightforward extension, allowing upward-compatability between programs written now and later.

3. This used to be a submodule of EC.PAR, because semaphores are used to synchronize processes. However, the secret of implementing them has nothing to do with processes, so semaphores belong in a module of their own.

EC.STATE

1. The following transitions are not included in this interface for the following reasons:

off to failed:	not relevant to user programs;
failed to off:	user programs cannot respond to anything
operating to off:	when the computer is off;

   Note that failed to operating does not occur with the current computer; it must be cycled through "off" to get back to operating from failed. However, future computers may make this transition possible (perhaps by re-booting), and so this transition is subsumed by the definition of power up.

2. There may not always be a grace period after @T(!+failed state+!). Two alternatives were considered: to leave out the grace period altogether, or to include it as a system-generation parameter. We selected the latter to allow for future use of an improved computer.

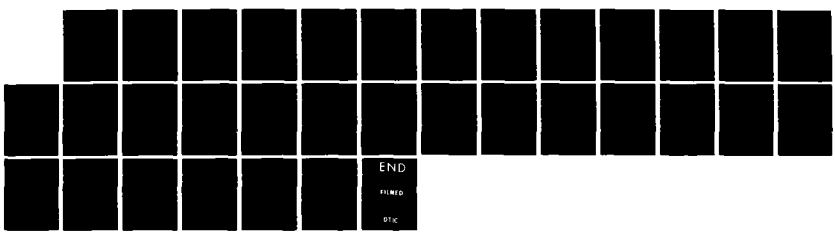
3. How do we distinguish between malfunctions that user programs must detect and handle (possibly by calling +S_FAIL_STATE+) and malfunctions that are detected inside the Extended Computer? Malfunctions are detected by this module if they are reported by the computer without software action; for example, malfunctions signalled by interrupts. Whenever a malfunction is detected because of an action dictated by the requirements, such as a diagnostic test, detection is left to a user program. The malfunctions described in various test programs (EC.PAR.1, EC.IO, EC.MEM, EC.TIMER) belong to the latter category; all others, the former.

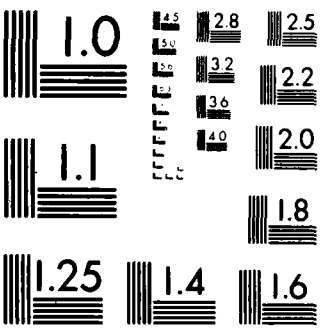
4. Future technology may make our three-state model appear oversimplified, because a system may have degraded states: that is, states without the full capability of "operating", yet not dead in the water like "failed". A degraded state may occur in a single-processor system, or in a multi-processor system where one or more processors have ceased to operate. It is important that acquiring this capability results in adding to (not revising) the present specification. We cannot add a degraded state now, because we cannot implement and programs depending on it would be not be correct. However, we can plan for the addition by assuming that there are "at least three states", etc.

5. Which module is responsible for the close-down procedures? We decided that any shutdown action that is required for every computer failure and is computer-dependent should be done by this module. If the action is device-dependent, such as setting the bomb-release output to a safe value, it should be done by the device interface module.

AD-A149 948 INTERFACE SPECIFICATIONS FOR SCR (SOFTWARE COST REDUCTION) (A-7E) EXTENDED COMPUTER MODULE REVISED(U) NAVAL RESEARCH LAB WASHINGTON DC D L PARNAS ET AL. 2/2

UNCLASSIFIED 31 DEC 84 NRL-MR-5502-REV F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

EC.TIMER

1. In earlier versions, we had clocks and timers; clocks counted up and timers counted down. They were completely distinct from timeint entities; they were declared separately and had their own set of operations. We removed the distinction as the interface grew and grew, and we realized that it would be both useful and consistent to let a clock/timer do most anything that a timeint can do.

2. We considered providing only clocks or only timers (in the sense of issue #1). Clocks are useful for measuring elapsed time; timers are useful for detecting the end of a previously specified time interval. We wanted the capabilities of both because otherwise user programs would have to use one to simulate the other. This would lead to inefficiency and possible duplicate efforts especially on a computer that provided both.

3. We considered having this module offer a special "waittime" command, instead of using the general semaphore mechanism. There seems to be no advantage in using a special mechanism for timed events.

3. We considered treating the following actions as errors:
   - starting a running timer,
   - setting a running timer,
   - stopping a non-running timer,
   - reading a non-running timer,
   - stopping a timer that has run down,
   - reading a timer that has run down,

   but these actions are not necessarily senseless.

4. We considered having a timer signal a UE if it runs past its capacity. To have it start over seems the most useful. Further, a timer might run past its usual limit, for no fault of the software. In contrast, setting timer with too large a value is a clear software error. Therefore we made exceeding the maximum capacity an undesired event in a set operation.

5. In an earlier design, there was a single maximum capacity for all clocks and a single maximum for all timers. It was pointed out that clocks and timers are used for very different purposes, some for measuring very small changes over a small period of time, and some for keeping track of a long period of time, with less concern for small changes. In order to achieve this flexibility without undue use of resources, we decided to allow programmers to specify capacity and minimum measurements for individual timers and clocks.

6. In an earlier version, clocks could only be set to zero, but this seems unnecessarily restrictive. Dwight Hill: "I believe we may need a +SET_CLOCK+ for clock corrections or for time-of-day clocks." The restriction went away when we merged timers and timeints.

7. Timers used to be a submodule of what was called the "Sequential Execution" module, because programmers would presumably want to transfer control based on the value or action of a timer. However, the secret of the timer module has nothing to do with flow of control, and so it became a module of its own.

APPENDIX 2

IMPLEMENTATION NOTES

EC.DATA

1. If the user provides a subrange assertion, the information may be used to reduce the amount of operand shifting necessary before an operation takes place.

EC.IO

1. The part of the I/O submodule that handles the relation between data item names and TC-2 instruction sequences should be a sysgen time program and should be table driven. It should be organized into submodules in accordance with the structure of the Device Interface Module, because changes are likely to be concentrated on individual devices.

EC.MEM: Contained in [VM]

EC.PAR.1: None.

EC.PAR.2: None.

EC.PGM.1: None.

EC.PGM.2

1. This module does not determine where programs are located in memory. It uses programs in the memory allocator module to request space.

2. This module uses the System Generation module to do assembly-time parameter type checking.

EC.PGM.3: None.

EC.SMPH: None.

EC.STATE: None.

EC.TIMER: None.

APPENDIX 3

ASSUMPTIONS LISTS

****
* BASIC ASSUMPTIONS *
****

EC.DATA

1. The Extended Computer can provide one private variable per process (the register) that can store values of any type. Access to a register will usually be quicker than access to other variables.

2. The attributes of a value will be known whenever a variable is used as a source or a destination. If the attributes specified for the variable when it is used as a source are not the same as were specified when its value was determined, the result may be any value.

3. The Extended Computer can store numeric quantities with any desired range and resolution. It can be expected that (a) variables with a small range-to-resolution ratio will require less actual memory space than variables with a large range-to-resolution ratio, and (b) that operations on such variables will be faster than operations on variables with a larger range-to-resolution ratio.

4. Range and resolution are adequate characterizations of a numeric variable; i.e., the needs of an application programmer can be adequately expressed by a lower bound, upper bound and guaranteed resolution.

5. The Extended Computer can store bitstring quantities of any desired length. Longer bitstring entities may require more storage than shorter ones. Operations on longer bitstring entities may require more computer time than operations on shorter ones.

6. Whenever a numeric value is stored into a variable with a resolution different from the source, the value stored should always be the closest value that can be represented in the destination. The programmer need not specify the conversions to be made; the best choice can be made by the EC implementation.

7. There is no need for operations that allow a bitstring value of one length to be assigned to a bitstring variable with a different length.

8. The operations needed for calculating new numeric values are:

   addition, multiplication, division, subtraction, absolute value, complement and conversions.

9. Division may result in a loss of all significance. This danger cannot be hidden entirely from the programmers, since they may have information that can be used to choose safe, efficient algorithms. The following division options are sufficient:

   a. The quickest division can be performed if the programmer provides an upper bound for the result. The better the bound, the more significance is preserved. If the bound is too low, all significance may be lost.

   b. A slower algorithm can be used if the programmer cannot provide an upper bound.

   c. If the programmer cannot provide an estimate of the maximum result and prefers to avoid the expense of the slower algorithm, the Extended Computer can determine whether or not division can be safely performed. The EC can return the sign of the quotient even when the operation cannot be safely performed.

10. For any variable, it is always possible to implement a uniform resolution over the entire range of that variable.

11. Whenever the program compares two numeric operands for equality, programmers need to define a threshold, such that if the difference between two numbers is less than or equal to the threshold, the numbers are considered equal.

12. It is acceptable for the results of an operation to have a larger resolution than the resolution of the destination. The approximations needed to store the result can be assumed to be acceptable for the application.

13. Only four kinds of entities are needed: variables, which can be changed at any time; ascons and literals, which can be changed by reassembling the program; and loadcons, which can be changed when the program is first loaded into the computer but not while it is running.

14. The following operations are sufficient for efficiently producing new bitstring values from existing bitstring values:

    a. AND, OR, NAND, NOT, MINUS, and XOR, defined in the usual way, operating on corresponding bits in two operands of equal length;

    b. SHIFT operation: A bitstring is shifted either right or left a specified number of bits with zeros shifted into positions vacated by the shift;

    c. REPLACE operation: A portion of a bitstring is replaced by the value found in an equal-length portion of another bitsring;

    d. CAT operation: A bitstring is formed by concatenating two previously existing bitstrings.

15. If the result of converting a real to a bitstring has more bits than the bitstring operand, the bits to the right of the rightmost bit of the destination bitstring may be ignored.

16. Arrays with dimensions that vary at run time are not needed in avionics applications.

17. Avionics applications do need arrays in which the type class is real, and the elements are variables with attributes that may vary independently of the attributes of other elements of the same array.

18. Arrays in which the indices are not a contiguous subset of the integers are not needed in avionics applications.

19. Avionics applications need to take advantage of any capability that the computer has to allow faster memory access to certain data. The Extended Computer can implement a "not-slower-than" relation for any two declared entities x and y, so that x will be accessed no slower than y. User programs can determine desired rankings at system generation time; it is not necessary to change the rankings at run-time.

EC.IO

1. The only information needed by user programs to identify inputs or outputs is the data item name given in the requirements document [REQ]. It is possible to characterize all transmissions between the Extended Computer and its associated hardware as either input or output.

2. Input data items and output data items are bitstring entities. Some can only be used as a source in a statement (read-only); some can only be used as a destination in a statement (write-only); some can be used as either (read-write). No input data item is write-only. No output data item is read-only.

3. It is possible to turn off (disable) input/output transmissions. A disabled data item has no effect on and is not affected by the external environment.

4. No application program will need the identity code and subitem identifiers in Serial Input Register Data (see [REQ]).

5. It is possible for the software to determine the success of I/O operations. (Of course, this assumption is obviously false if we consider hardware failures. However the correctness of our software is contingent on that assumption.) An unsuccessful operation may not change the value of the associated data item.

6. Some input data items are only available intermittently and the EC can notify user programs when new values for such data become available.

7. Each i/o operation can be guaranteed to complete within a fixed period of time. This worst-case timing requirement varies among data items; the time associated with each data item can be determined at system-generation time.

8. Each channel diagnostic program may interfere with a specified subset of the input/output commands. They will not interfere with any other commands.

9. Use of either the discrete diagnostics or the accelerometer-torque diagnostics may cause the IMS to lose its alignment and velocities (i.e., have the same effect as disabling the IMS temporarily).

10. The following aspects of the input/output can be tested independently:

the AC aspects of the signal converter channel,
the DC aspects of the signal converter channel,
the cycle steal channel A and serial input channel 1,
the cycle steal channel B and serial input channel 2,
discrete input word 1 and discrete output word 1,
discrete input word 2 and discrete output word 2,
discrete input word 3 and discrete output word 3,
the IMS gyro torque registers and the accelerometer accumulators.

EC.MEM

1. A memory diagnostic program can check whether portions of memory are reliable. This program does not interfere with other programs. The test may take a substantial amount of time to complete.

Basic assumptions concerning the hidden portion of the interface may be found in [VM].

EC.PAR.1

1. Processes (executions of programs) may execute in parallel with no restrictions on their relative speeds, except where they are explicitly synchronized with each other (see EC.PAR.2).

2. The number of processes need not vary at run-time. It may be set at system generation time.

3. All demand processes can start when the system is turned on (i.e., when @T(!+power up+!) occurs); some will perform initialization routines; the remaining demand processes will wait for a semaphore to become nonnegative.

4. The process mechanism will be able to detect the event @T(!+power up+!).

5. Processes are not called as subroutines by other programs and do not return control to other programs.

6. We need only distinguish two process states: active or suspended. An active process can progress. A suspended process is ineligible to progress (continue execution).

7. The state of a process changes between active and suspended only when it uses the process synchronization mechanisms described in sections EC.PAR.2 and EC.SMPH or when it has executed the last statement in its body.

8. All processes are either periodic or demand and exist throughout the life of the system;

   The bodies of periodic processes are to be executed at regular intervals (their period). The period of a process may change during system execution. A periodic process may be suspended when a specified boolean variable is false and start again when it is true.

   Demand processes wait for a semaphore to be nonnegative. They should be executed each time the semaphore is incremented. They will decrement the semaphore once per execution.

9. Demand processes can be adequately characterized by specifving the values of two timing parameters: maximum CPU time requirement and deadline for completion.

10. Periodic processes are adequately characterized by three timing parameters: maximum CPU time requirement, deadline, and period.

EC.PAR.2

1. User programs may contain contiguous sections or regions of run-time-executable statements that may not be executed concurrently. These concurrency constraints can be expressed in terms of an exclusion relation on the regions, i.e., where region 1 excludes region 2 if region 2 may not start while region 1 is executing.

2. Regions may overlap other regions or be embedded in other regions.

EC.PGM.1

1. The only sequence control constructs needed are those that choose a path based upon the results of the invocation of a program.

2. The number of entrances and exits of a program is finite, and the upper bound can be determined at system generation time.

EC.PGM.2

1. Some program entities should be invoked faster than others. Such a relation will not depend on when the programs are invoked; the relative ranking can be determined at system generation time.

2. It is not necessary to provide users with the capability to create programs that take parameters. Other mechanisms available to him (such as assignment before and after the "body" of the program) suffice.

3. It is necessary to provide facilities for recovery if a programming error is detected by a program during execution. It is up to the author of a called program to determine what programming errors his program can detect; it is up to the caller of a program to determine the action that should be taken if one of those errors occurs. It is not necessary to pass parameters to the recovery program.

EC.PGM.3

1. If a program will be reentered while already in use by another process, it is the responsibility of the programmer to make sure that local storage is saved and restored as needed. EC programs are not automatically provided with new storage when they are reentered.

2. There is no need for a mechanism to allow programs to cause the calling program to resume execution anywhere else than immediately after the call.

3. The identity of a data entity that is passed to an EC access program as an actual parameter will not be changed while the program is executing. For example, when an array element is passed as an actual parameter to a program, if that program alters the value of the variables that determined the index, the results will be undefined.

4. Parameters always fall into one of three classes: input, output, or input-output.

EC.SMPH

1. There are two process states relative to synchronization: active (which includes processes that are running and processes that are ready) and suspended (ineligible to make progress). The active processes are the only ones eligible for execution.

2. The only operations on semaphores that need to be executed in a way that guarantees non-interference with other operations on semaphores are the following:
   a. An operation that does not affect the counter value of the semaphore, but may put the process in the waiting state.
   b. An operation to decrement the semaphore counter without any effect on the state of the process that executes it.
   c. An operation to increment the semaphore counter that may put other processes in the active state.

EC.STATE

1. The Extended Computer has at least three states: off, operating, and failed. Only the following transitions between states affect user programs:

   - from off to operating
   - from operating to failed.

2. User programs cannot cause the transition into the operating state.

3. A transition from operating to failed can either be caused by user programs or occur when malfunctions internal to the Extended Computer are detected. These internal malfunctions are other than those described in test programs contained in EC.PAR.1, EC.TIMER, EC.MEM, and EC.IO. It should be assumed that after this transition occurs, user programs will have at least a short interval to execute shut-down sequences before the computer stops operating. The minimum length of the interval before shut-down can be determined at system-generation time.

4. Any actions that must be taken when a computer failure occurs are independent of the state of the user programs, and can be built into the EC.

EC.TIMER

1. Avionics programs need timers that keep track of elapsed time, and that may signal when a given time interval has elapsed. They need to be able to set a timer to a starting value, start it, stop it, and read it whether it is running or not.

2. The maximum timing capacity of a clock or a timer can be determined at system generation time.

3. If a timer runs beyond a limit specified at run time, it should either halt or start over. Sometimes it should signal that a range limit has been reached.

5 The worst acceptable error rate for all timers can be determined by users at system generation time. This error can be specified as a fraction of the running time.

6. Any number of timers can be implemented, provided that the number is known at system-generation time. There is no need to create or delete timers at run time.

7. There are diagnostic programs that can test the hardware timers and the interrupt mechanism separately, but may interfere with proper execution of other programs.

****
* ASSUMPTIONS ABOUT UNDESIRED EVENTS *
****

EC.DATA

1. User programs will not divide by zero.

2. The result of any operation will not be outside the range of the destination variable.

3. In a replace operation, user programs will not specify positions that do not appear within bitstrings or specify a substring with a start position that is higher than the stop position.

4. After converting a numeric value to a bitstring, there will be no bits to the left of the most significant bit of the destination bitstring.

5. Users will not supply a parameter in an array reference that is not in the index set of the array.

EC.IO

1. User programs will not attempt to
   - use an enabled read-write input data item as a !!destination!!; or
   - use an enabled read-write output data item as a !!source!!.

2. User programs will not disable (enable) a data item that is already disabled (enabled).

EC.MEM

None. For UE assumptions about the hidden portion of the interface, see [VM].

EC.PAR.1

1. Demand processes will not need to run so often as to cause a periodic process to miss its deadline.

2. A periodic process will have a period greater than its deadline.

EC.PAR.2

None.

EC.PGM.1

1. Every program exit that will be chosen during execution will be connected to a succeeding command.

EC.PGM.2

1. A user will not fail to assign a value to a built-in EC program variable.

EC.PGM.3

1. A program will not invoke itself.

EC.SMPH

1. There is a range of values that will suffice for all semaphores, and will not be exceeded by user programs.

EC.STATE.3

None.

EC.TIMER

None.

CRF 129 247

APPENDIX 4

UNIMPLEMENTED EXTENDED COMPUTER FACILTIES

Not all of the capabilities described in this document have been provided in the current version of the Extended Computer. A few facilities, which are not currently needed by the application program, have not been implmented. An attempt to use an absent facility will result in an undesired event in the development version. The unimplemented features are described below.

FEATURE: Periodic processes with periods that vary at run-time
WHERE DESCRIBED: EC.PAR.1
UNDESIRED EVENT: %%unimplemented variable period%%
CURRENT USE: The !+period+! parameter in the ++P_PROCESS++ must be given as an ascon or a literal.

FEATURE: Ability to enable/disable all data items
WHERE DESCRIBED: EC.IO
UNDESIRED EVENT: %%unimplemented disabling%%
CURRENT USE: Only the following data items may be disabled; attempting to +DISABLE+ or +ENABLE+ any other is prohibited.

//ASAZ//	//ASEL//	//ASLAZ//
//ASLEL//	//ASLCOS//	//ASLSIN//
//AZRING//	//BAROHUD//	//CURAZCOS//
//CURAZSIN//	//CURPOS//	//DESTPNT//
//FLTDIRAZ//	//FPMAZ//	//FPMEL//
//HUDAS//	//HUDASL//	//HUDFPM//
//HUDPUC//	//HUDSCUE//	//HUDVEL//
//HUDWARN//	//LSOLCUAZ//	//LSOLCUEL//
//MAGHDGH//	//MAPOR//	//PTCHANG//
//PUACAZ//	//PUACEL//	//ROLLCOSH//
//ROLLSINH//	//USOLCUAZ//	//USOLCUEL//
//VERTVEL//	//VTVELAC//	//XCOMMC//
//XCOMMF//	//YCOMM//	

FEATURE: Bitstrings/timeints/pointers with attributes that can vary at run-time
WHERE DESCRIBED: EC.DATA
UNDESIRED EVENT: %%unimplemented binding%%
CURRENT USE: In the ++DCL_TYPE++ program, users may not declare the binding of bitstring or timeint or pointer specific types to be VARY.

CRF 154 198 243

FEATURE:	Timers with attributes that can vary at run-time
WHERE DESCRIBED:	EC.TIMER
UNDESIRED EVENT:	%%unimplemented binding%%
CURRENT USE:	In the ++DCL_TYPE++ program for timers, users may not declare the binding of timers to be VARY.

FEATURE:	Semaphores with attributes that can vary at run-time
WHERE DESCRIBED:	EC.SMPH
UNDESIRED EVENT:	%%unimplemented binding%%
CURRENT USE:	In the ++DCL_TYPE++ program for semaphores, users may not declare the binding of semaphores to be VARY.

FEATURE:	Programs with attributes that can vary at run-time
WHERE DESCRIBED:	EC.PGM.2
UNDESIRED EVENT:	%%unimplemented binding%%
CURRENT USE:	In the ++DCL_TYPE++ program for programs, users may not declare the binding of programs to be VARY.

FEATURE:	Undesired events in the production EC
WHERE DESCRIBED:	Throughout
UNDESIRED EVENT:	none
CURRENT USE:	In the production version of the Extended Computer, no undesired events will be checked for; no undesired event handling programs will be assembled or executed. It will be assumed that user programs will invoke the EC facilities correctly.

FEATURE:	Specifying substrings of bitstrings with variables
WHERE DESCRIBED:	EC.DATA.2.7.2
UNDESIRED EVENT:	%%unimplemented variable substring%%
CURRENT USE:	In the bitstring +REPLC+ program, p2, p3, and p4 must be given by literals or ascons.

FEATURE:	Specifying the length of a bitstring shift with a variable
WHERE DESCRIBED:	EC.DATA.2.7.2
UNDESIRED EVENT:	%%unimplemented variable shift length%%
CURRENT USE:	In the +SHIFT+ program, p2 must be given by a literal or an ascon.

CRF 154 198 247

FEATURE:	Using variables to specify attributes of a specific type, or of a variable or array with varying attributes
WHERE DESCRIBED:	EC.DATA.3, EC.SMPH.3, EC.TIMER.3
UNDESIRED EVENT:	%%unimplemented attribute via variables%%
CURRENT USE:	To specify an attribute (as defined in EC.DATA.3), a timer-attribute (as defined in EC.TIMER.3), or a semaphore-attribute (as defined in EC.SMPH.3), literals or ascons must be used.

FEATURE:	Allowing the EXACT_REP attribute to vary for numeric types.
WHERE DESCRIBED:	EC.DATA.2.4, EC.DATA.3
UNDESIRED EVENT:	%%unimplemented varying EXACT_REP%%
CURRENT USE:	If a type is declared to have varying attributes, and is given an initial attribute that includes EXACT_REP then it may not later be assigned attributes that do not include EXACT_REP; the converse is also true.

FEATURE:	Using the EXACT_REP attribute for any resolution.
WHERE DESCRIBED:	EC.DATA.3
UNDESIRED EVENT:	%%unimplemented EXACT_REP resolution%%
CURRENT USE:	Whenever a type has the EXACT_REP attribute, its resolution must be an exact power of two.

FEATURE:	Checking parameter type when it is given by a pointer.
WHERE DESCRIBED:	EC.DATA
UNDESIRED EVENT:	None.
CURRENT USE:	If an !!actual parameter!! is given by naming a pointer to an entity, and that entity is not of the proper type as required by the program being invoked, the result will be unpredictable; no UE will be raised.

FEATURE:	Checking if a !!destination!! is not a variable when it is given by a pointer.
WHERE DESCRIBED:	EC.DATA
UNDESIRED EVENT:	None.
CURRENT USE:	If an !!destination!! is given by naming a pointer to an entity, and that entity is not a variable, the result will be unpredictable; no UE will be raised.

FEATURE:	Pointers pointing to programs that have parameters.
WHERE DESCRIBED:	EC.DATA, EC.PGM.2
UNDESIRED EVENT:	%%unimplemented pgm ptr%%
CURRENT USE:	A pointer may only refer to user-defined programs or to parameterless EC access programs.

FEATURE:	Defining a program with more than one entrance.
WHERE DESCRIBED:	EC.PGM.1
UNDESIRED EVENT:	%%unimplemented multi-entrance pgm%%.
CURRENT USE:	A user may not supply more than one entrance name in any invocation of ++ENTRANCE++. The UE %%entrance incorrectly omitted%% of EC.PGM.3 will not be checked for, in lieu of the above UE.

FEATURE:	EC access programs that compute a single output parameter having more than one exit, unless the parameter is real or boolean.
WHERE DESCRIBED:	EC.PGM.1
UNDESIRED EVENT:	%%unimplemented multi-exit EC access program%%
CURRENT USE:	Only EC access programs computing a single output parameter that is real or boolean have more than one exit as described in EC.PGM.1.2.1. Invocations of other EC access programs that compute a single result must be followed by an exit-list of the form :label or by a null exit-list.

CRF 247

APPENDIX 5

INPUT/OUTPUT DATA ITEM NAMES

The following table lists all data items available from the Extended Computer, gives the bitstring spectype of each, and tells whether each one is read-only (R), write-only (W), or read-write (RW). The length of each item is embedded in its spectype name.

INPUT DATA ITEMS

Data item name	R or RW	Spectype
/ACAIRB/	R	BIO1
/ADCFAIL/	R	BIO1
/AOA/	R	BAOA12
/ANTGOOD/	R	BIO1
/ARPINT/	R	BARP8
/ARPPAIRS/	R	BIO1
/ARPQUANT/	R	BARP8
/BAROADC/	R	BADC12
/BMBDRAG/	R	BIO1
/BRGSTA/	R	BTAC11
/DIMWC/	R	BIO1
/DGNDSP/	R	BDRS14
/DRFTANG/	R	BDRS14
/DRSFUN/	R	BDRS3
/DRSMEM/	R	BIO1
/DRSREL/	R	BIO1
/ELECGOOD/	R	BIO1
/FLYTOTOG/	R	BPNL2
/FLYTOTW/	R	BPNL4
/GUNSSEL/	R	BIO1
/HUDREL/	R	BIO1
/IMSAUTOC/	R	BIO1
/IMSMODE/	R	BIMS5
/IMSREDY/	R	BIO1
/IMSREL/	R	BIO1

OUTPUT DATA ITEMS

Data item name	W or RW	Spectype
//ANTSLAVE//	W	BIO1
//ASAZ//	RW	BHUD12
//ASEL//	RW	BHUD12
//ASLAZ//	RW	BHUD12
//ASLCOS//	RW	BHUD12
//ASLEL//	RW	BHUD12
//ASLSIN//	RW	BHUD12
//AUTOCAL//	W	BIO1
//AZRING//	RW	BMAP12
//BAROHUD//	RW	BHUD13
//BMBREL//	W	BIO1
//BMBTON//	W	BIO1
//BRGDEST//	W	BHSI13
//COMPCTR//	W	BIO1
//COMPFAIL//	W	BIO1
//CURAZCOS//	RW	BFLRCUR13
//CURAZSIN//	RW	BFLRCUR13
//CURENABL//	W	BIO1
//CURPOS//	RW	BFLRCUR12
//DESTPNT//	RW	BMAP12
//ENTLIT//	W	BIO1
//FIRRDY//	W	BIO1
//FLTDIRAZ//	RW	BHUD12
//FLTREC//	W	See note 3
//FPANGL//	W	BFLRFPA11
//FPMAZ//	RW	BHUD12
//FPMEL//	RW	BHUD12
//GNDTRK//	W	BHSI13

CRF 204

9159a

INPUT DATA ITEMS

Data item name	R or RW	Spectype
/KBDINT/	R	BPNL10
/LOCKEDON/	R	BIO1
/MA/	R	BIO1
/MACH/	R	BIMS12
/MAGHCOS/	R	BIMS13
/MAGHSIN/	R	BIMS13
/MFSW/	R	BMFS5
/MODEROT/	R	BPNL6
/MULTRACK/	R	BIO1
/PCHCOS/	R	BIMS13
/PCHSIN/	R	BIMS13
/PMDCTR/	R	BIO1
/PMHOLD/	R	BIO1
/PMNORUP/	R	BIO1
/PMSCAL/	R	BIO1
/PMSLAND/	R	BIO1
/PNLTEST/	R	BIO1
/PRESPOS/	R	BPNL3
/RADALT/	R	BRA12
/RE/	R	BIO1
/RNGSTA/	R	BTAC14
/ROLLCOSI/	R	BIMS13
/ROLLSINI/	R	BIMS13
/SINEVEL/	R	See note 1
/SINHDG/	R	See note 1
/SINLAT/	R	See note 1
/SINLONG/	R	See note 1
/SINNVEL/	R	See note 1
/SINPTH/	R	See note 1
/SINROL/	R	See note 1
/SLEWRL/	R	BSLEW13
/SLEWUD/	R	BSLEW13
/SLTRNG/	R	BFLRSR13
/STA1RDY/	R	BIO1
/STA2RDY	R	BIO1
/STA3RDY/	R	BIO1
/STA6RDY/	R	BIO1
/STA7RDY/	R	BIO1
/STA8RDY/	R	BIO1
/TD/	R	BIO1
/TAS/	R	BADC12
/THDGCOS/	R	BIMS13
/THDGSIN/	R	BIMS13
/UPDATTW/	R	BPNL4

OUTPUT DATA ITEMS

Data item name	W or RW	Spectype
//GNDTRVEL//	W	BFLR10
//HUDAS//	RW	BIO1
//HUDASL//	RW	BIO1
//HUDFPM//	RW	BIO1
//HUDPUC//	RW	BIO1
//HUDSCUE//	RW	BIO1
//HUDVEL//	RW	BIO1
//HUDWARN//	RW	BIO1
//IMSNA//	W	BIO1
//IMSSCAL//	W	BIO1
//KELIT//	W	BIO1
//LATGT70//	W	BIO1
//LFTDIG//	W	BIO1
//LLITDEC//	W	BIO1
//LLITE//	W	BIO1
//LLIT322//	W	BIO1
//LLITW//	W	BIO1
//LSOLCUAZ//	RW	BHUD12
//LSOLCUEL//	RW	BHUD12
//LWDIG1//	W	BPNL7
//LWDIG2//	W	BPNL7
//LWDIG3//	W	BPNL7
//LWDIG4//	W	BPNL7
//LWDIG5//	W	BPNL7
//LWDIG6//	W	BPNL7
//LWDIG7//	W	BPNL7
//MAGHDGH//	RW	BHUD11
//MAPOR//	RW	BMAP12
//MARKWIN//	W	BPNL7
//PTCHANG//	RW	BHUD12
//PUACAZ//	RW	BHUD12
//PUACEL//	RW	BHUD12
//RNGHND//	W	BMAP8
//RNGTEN//	W	BMAP8
//RNGUNIT//	W	BMAP8
//ROLLCOSH//	RW	BHUD12
//ROLLSINH//	RW	BHUD12
//STEERAZ//	W	BFLRSTR13
//STEEREL//	W	BFLRSTR13
//STERROR//	W	BADI11
//TSTADCFLR//	W	BIO1
//ULITN//	W	BIO1
//ULITS//	W	BIO1
//ULIT222	W	BIO1

CRF 204

INPUT DATA ITEMS

Data item name	R or RW	Spectype
/WAYLAT/	R	See note 2
/WAYLON/	R	See note 2
/WAYNUM1/	R	See note 2
/WAYNUM2/	R	See note 2
/WEAPTYP/	R	BASCU8
/XGYCNT/	R	BIMS2
/XVEL/	R	BIMS10
/YGYCNT/	R	BIMS2
/YVEL/	R	BIMS10
/ZGYCNT/	R	BIMS2
/ZVEL/	R	BIMS10

OUTPUT DATA ITEMS

Data item name	W or RW	Spectype
//ULIT321//	W	BIO1
//USOLCUAZ//	RW	BHUD12
//USOLCUEL//	RW	BHUD12
//UWDIG1//	W	BPNL7
//UWDIG2//	W	BPNL7
//UWDIG3//	W	BPNL7
//UWDIG4//	W	BPNL7
//UWDIG5//	W	BPNL7
//UWDIG6//	W	BPNL7
//VERTVEL//	RW	BHUD12
//VTVELAC//	RW	BHUD12
//XCOMMC//	RW	BMAP9
//XCOMMF//	RW	BMAP13
//XGYCOM//	W	BIMS4
//XSLEW//	W	BIO1
//XSLSEN//	W	BIO1
//YCOMM//	RW	BMAP11
//YGYCOM//	W	BIMS4
//YSLEW//	W	BIO1
//YSLSEN//	W	BIO1
//ZGYCOM//	W	BIMS4
//ZSLEW//	W	BIO1
//ZSLSEN//	W	BIO1

Notes

1. The spectype of this item is BSINSn, where "n" is the length of the item. The length may be determined by consulting the confidential addendum to [REQ].

2. The spectype of this item is BWISn, where "n" is the length of the item. The length may be determined by consulting the confidential addendum to [REQ].

3. //FLTREC// is an array of type BFLTREC16. The number of elements is given by the integer system generation parameter #nbr fltrec elements#.

CRF 204 222 231

The following data items have events (signalled by incrementing a semaphore) associated with them:

Event	Semaphore
@T(!+/ENTERSW/ occurred+!)	ENTSWSEM
@T(!+/KBDENBL/ occurred+!)	ENBLSEM
@T(!+/MARKSW/ occurred+!)	MARKSEM
@T(!+/KBDINT/ ready+!)	KBINTSEM

APPENDIX 6

DATA REPRESENTATION CATALOGUE

For some specific types, the Extended Computer is capable of providing more than one kind of representation. The version has no effect on the outcome of an EC operation, but some versions allow some operations to be performed more efficiently than other versions.

The following table lists the provided version names for each EC specific type which has more than one version. When declaring a specific type, users may request a particular version by using these names.

Typeclass	Specific type	Version names	Version properties
REAL	Any	R1	N/A
BITSTRING	Any	B1	N/A
TIMEINT	Any	T1	N/A
TIMER	Any	C1	N/A
SEMAPHORE	Any	S1	N/A
PGM	Any	P1	N/A

CRF 105

9244a

REFERENCES

[ADT] Clements P. C., Faulk S. R., Parnas D. L.; Interface Specifications for the SCR (A-7E) Application Data Types Module; NRL Report 8734, 23 August 1983. (AD-A132717)

[APC] Faulk, S.; "Pseudo-Code Language for the A-7E OFP", internal memorandum, April 1982

[BELP73] Belpaire, Wilmotte; "A Semantic Approach to the Theory of Parallel Processes"; in International Computing Symposium 1973.

[DIJK68] Dijkstra, E. W.; "Co-operating Sequential Processes", in Programming Languages, ed. F. Genuys; Academic Press, 1968. pp. 43-112.

[DIJK77] Dijkstra, E. W.; A Discipline of Programming; Prentice Hall, 1976.

[DIM] Parker, Heninger, Parnas, Shore; Abstract Interface Specifications for the A-7E Device Interface Module, NRL Memorandum Report 4385, November, 1980. (AD-A092-696)

[REED79] Reed, Kanodia; "Synchronization with Eventcounts and Sequencers"; Comm. of the ACM, v. 22, no. 2 (1979).

[REQ] Heninger K. L., Kallander J. W., Parnas D. L., Shore J. E.; Software Requirements for the A-7E Aircraft; NRL Memorandum Report 3876; Nov 1978. (AD-A061-751)

[SO] Clements P. C., Parker R. A., Parnas D. L., Shore J. E., Britton K. H.; A Standard Organization for Specifying Abstract Interfaces, NRL Report 8815, 14 June 1984.

[TRACE] Parnas, "Trace Specifications for D-Operations", NRL Technical Memorandum 7590-000:DP, to be published.

[VM] Alspaugh, Weiss, "Virtual Memory Interface Specifications", NRL Report in progress, draft copy 16 April 1984.

[WUER76] Parnas D. L., Wuerges H.; "Response to Undesired Events in Software Systems"; Proc. 2nd Int. Conf. Software Eng., pp. 437-446; 1976

ACKNOWLEDGMENTS

The authors gratefully acknowledge the hard work and careful reviews provided by the following people:

Naval Weapons Center, China Lake, CA:
Jack Basden
Richard Fryer
Sandra Fryer
Dawn Janney
Ray Martinusen
Jo Miller
Lee Thomson
Robert Westbrook
Richard Wolff
Janice Zenor

Vought Corporation, Dallas, TX:
Glenn Cooper
Dwight Hill

USAF A-7D/K OFP Detachment, Tucson, AZ:
Mark Jacobson
Richard Breisch

Bell Telephone Laboratories, Columbus, OH:
Don Utter

Grumman Aerospace Corp., Bethpage, NY:
Stephanie White

Computer Science and Systems Branch, Naval Research Laboratory, Washington, DC:
Tom Alspaugh
Stuart Faulk
Bruce Labaw
Larry Morell
Preston Mullen
Dr. John Shore

CRF 087 233

END

FILMED

3-85

DTIC