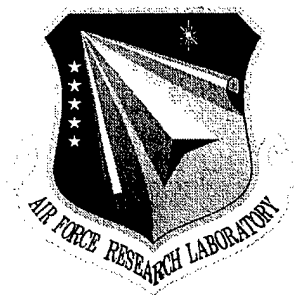


AFRL-IF-RS-TR-2000-19
Final Technical Report
March 2000



DEVELOPMENT OF AN ADAPTIVE COMPUTING SYSTEM WITH REMOTE PODS FOR DATA ACQUISITION AND SENSOR INTEGRATION

Acquisition Systems, LLC

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F227

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

20000420 149

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-19 has been reviewed and is approved for publication.

APPROVED:



RALPH KOHLER
Program Manager

FOR THE DIRECTOR:



NORTHROP FOWLER
Technical Advisor
Information Technology Division

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTC, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

DEVELOPMENT OF AN ADAPTIVE COMPUTING SYSTEM WITH REMOTE
PODS FOR DATA ACQUISITION AND SENSOR INTEGRATION

Bruce Pirger

Contractor: Acquisition Systems, LLC
Contract Number: F30602-97--0285
Effective Date of Contract: 16 September 1997
Contract Expiration Date: 15 September 1998
Program Code Number: 62301E
Short Title of Work: Development of an Adaptive Computing
System with Remote Pods Data Acquisition
and Sensor Integration
Period of Work Covered: Sep 97 – Sep 98
Principal Investigator: Bruce Pirger
Phone: (607) 255-5892
AFRL Project Engineer: Ralph Kohler
Phone: (315) 330-2016

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This research was supported by the Defense Advanced Research
Projects Agency of the Department of Defense and was monitored
by Ralph Kohler, AFRL/IFTC, 26 Electronic Pky, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE MARCH 2000	3. REPORT TYPE AND DATES COVERED Final Sep 97 - Sep 98		
4. TITLE AND SUBTITLE DEVELOPMENT OF AN ADAPTIVE COMPUTING SYSTEM WITH REMOTE PODS FOR DATA ACQUISITION AND SENSOR INTEGRATION		5. FUNDING NUMBERS C - F30602-97-C-0285 PE - 62301E PR - D002 TA - 02 WU - P9		
6. AUTHOR(S) Bruce Pirger				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Acquisition Systems, LLC 26 Lake Street Trumansburg NY 14886		8. PERFORMING ORGANIZATION REPORT NUMBER N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency Air Force Research Laboratory/IFTC 3701 North Fairfax Drive 26 Electronic Pky Arlington VA 22203-1714 Rome NY 13441-4514		10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2000-19		
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Ralph Kohler/IFTC/(315) 330-3016				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This project consisted of the development of an adaptive computing system with an emphasis on remote sensor control, data acquisition, and sensor data processing. During the project, two hardware platforms were designed and developed and software was modified and extended to the new platforms. The developed hardware was then integrated into an existing infrared focal plan array sensor package and used for sensor control and data acquisition. The use of reconfigurable hardware has extended the capabilities of the sensor platform considerably and has proven very successful.				
14. SUBJECT TERMS Remote Pod, CompactPCI Bus, Infrared Focal Plane Array Control and Data Acquisition, Hardware Platform Development			15. NUMBER OF PAGES 56	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1	Project Goals/Achievements.....	1
1.1.	Hardware/Software Development.....	1
1.2.	Infrared Focal Plane Array Control and Data Acquisition.....	2
2.	Hardware Platform.....	3
2.1.	Altera, CompactPCI, Cypress HOTLink, ZBT SRAM, AMCC 5933 PCI Interface, SHARC 21060 DSP, Daughter Card Format.....	3
2.1.1.	Altera.....	3
2.1.2.	CompactPCI.....	4
2.1.3.	Cypress Semiconductor HOTLink.....	4
2.1.4.	Memory Selection.....	5
2.1.5.	AMCC 5933 PCI Interface.....	5
2.1.6.	SHARC 21060 DSPs.....	6
2.1.7.	Daughter Card Format.....	6
2.2.	Remote Pod.....	6
2.2.1.	10K50 FPGA.....	9
2.2.2.	SRAM.....	9
2.2.3.	Cypress HOTLink Interface.....	9
2.2.4.	I/O Pins.....	10
2.2.5.	Clock Distribution.....	10
2.2.6.	Boot EPROM.....	10
2.2.7.	JTAG.....	10
2.3.	Base Platform.....	11
2.3.1.	10K250AA FPGAs.....	13
2.3.2.	5933 PCI Interface.....	13
2.3.3.	ZBT Synchronous SRAM.....	14
2.3.4.	Dual Port SRAM.....	14
2.3.5.	21060 SHARC DSP.....	15
2.3.5.1.	Dual Port SRAM.....	15
2.3.5.2.	Link Ports.....	16
2.3.6.	Programmable Oscillator.....	16
2.3.7.	Daughter Card Interface.....	16
2.3.8.	Debugging Port.....	17
2.4.	Daughter Card Fiber Interface.....	17
3.	Software Platform Extension.....	20
3.1.	ASbridge Host Interface.....	20
3.1.1.	General Program Design.....	20
3.1.2.	Platform-Specific Extensions.....	21
3.1.3.	Software Scatter/Gather.....	23
3.1.4.	SHARC/Host Communication.....	23
3.2.	Hardware Promela Extension.....	23
3.2.1.	Hardware Interface Library.....	24
3.2.2.	Software Interface Library.....	25
3.2.3.	Extensions to this platform.....	28
3.2.3.1.	Fiber Optic Channels.....	29
3.2.3.2.	ZBT Synchronous SRAM.....	29
3.2.3.3.	SHARC Communication.....	30
3.2.3.4.	DMA channels.....	30

Table of Contents

3.2.3.5.	Summary of Hardware Promela Extensions.....	32
4.	Integration with Infrared Focal Plane Array Sensor System.....	34
4.1	Existing System.....	34
4.1.1.	Focal Plane Array Control and Data Acquisition.....	35
4.1.2.	Communication Link with Host.....	35
4.1.3.	Stepper Motor Control.....	36
4.2.	Physical Integration of Remote Pod.....	37
4.3.	FPGA Coding and System Operation.....	38
4.4.	New Capabilities.....	39
5.	Platform Evaluation.....	40
5.1.	Remote Pod.....	41
5.2.	Base Platform.....	42
5.3.	Software Interfaces.....	42
5.3.1.	Asbridge.....	42
5.3.2.	Hardware Promela.....	42
5.4.	The Next Step.....	44

List of Figures

Fig 1	Quad Chart.....	2
Fig 2	Remote Pod.....	7
Fig 3	Base Platform.....	11
Fig 4	Fiber Interface Daughter Card.....	18
Fig 5	FPGA Coding and System Operation.....	38

List of Photos

Photo 1	Remote Pod.....	8
Photo 2	Base Platform.....	11
Photo 3	Fiber Daughter Card.....	19

1. Project Goals/Achievements

This project consisted of the development of an adaptive computing system with an emphasis on remote sensor control, data acquisition, and sensor data processing. During the project, two hardware platforms were designed and developed and software was modified and extended to the new platforms. The developed hardware was then integrated into an existing infrared focal plane array sensor package and used for sensor control and data acquisition. The use of reconfigurable hardware has extended the capabilities of the sensor platform considerably and has proven very successful.

This is the Final Report for this development effort. The remainder of this document will describe the developed hardware platforms (Section 2), system software (Section 3), and discuss the successful integration with an existing sensor system (Section 4). In Section 5 we will provide our overall system evaluation, lessons learned, and suggested next steps.

A separate document will serve as the User's Documentation for further development with this platform, including both hardware and software documentation.

The remainder of Section 1 will provide a brief introduction to the project.

1.1. Hardware/Software Development

The project's quad chart is on the following page in Figure 1. It summarizes the project and its goals very well. The goal of the project was to extend reconfigurable hardware directly to sensor platforms. Previous R&D efforts have focussed on large scale computing platforms, striving to achieve the first Tera-op scale computing platform in a compact size by allowing custom configuration of reconfigurable resources. Acquisition Systems desired to extend the utility of reconfigurable hardware not only to large scale computing, but also to direct integration and control with external sensor and actuator systems. To accomplish this, we designed a small, compact reconfigurable system we call the remote pod which is intended to be directly integrated into existing sensor platforms or used as a foundation for sensor platform development. This small remote pod is linked via a fiber optic interface to a host platform we call the base platform. This base platform utilizes the CompactPCI bus. Hence the entire platform, consisting of the base, remote pod, and host CompactPCI computer can provide the complete foundation for development of advanced sensor platforms.

In order for large scale reconfigurable computing systems to be widely adopted and therefore deemed successful, they must become much easier to program. Furthermore, most current systems require programming at the hardware design level,

excluding nearly all software programmers from utilizing the capabilities of such a platform. Many researchers are trying to develop high level languages which can be used by software programmers to develop custom applications. During this effort we ported one such high level language to the base platform.

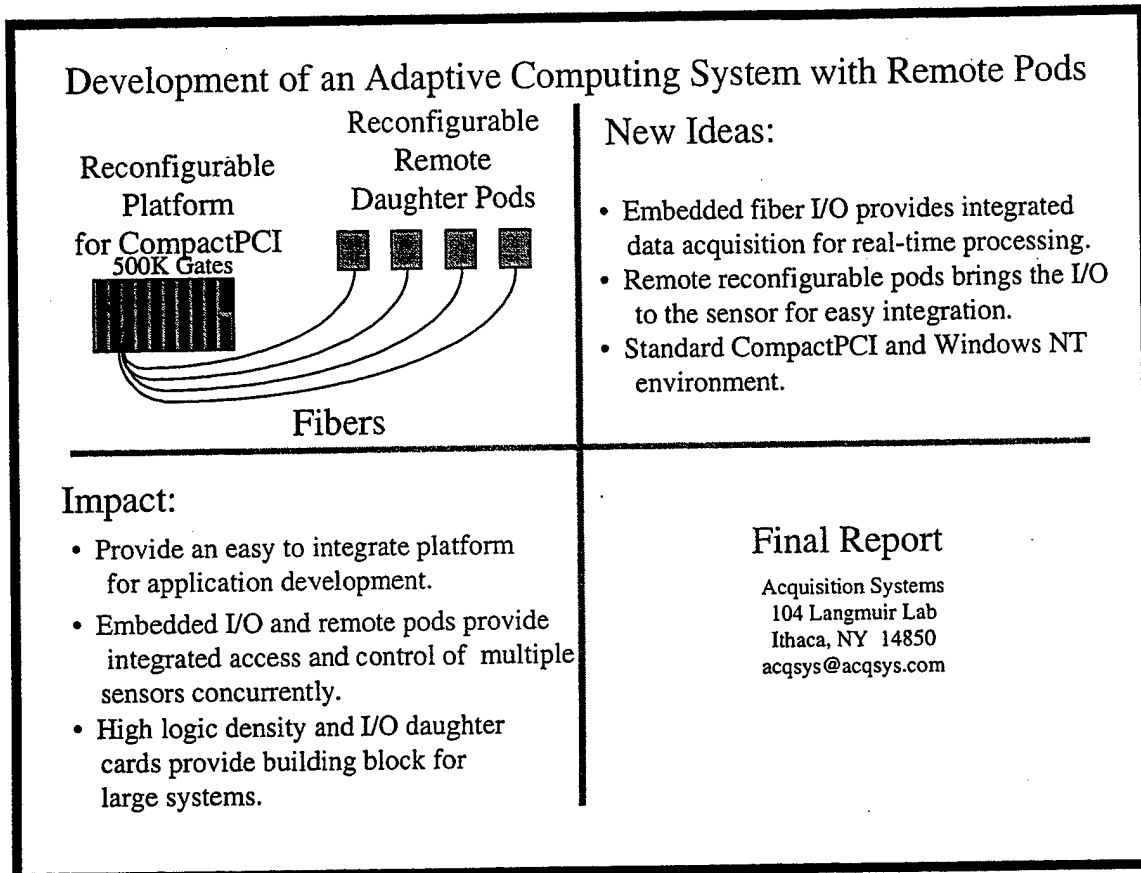


Figure 1. Quad Chart

1.2. Infrared Focal Plane Array Control and Data Acquisition

During the effort, the remote pod was integrated into an existing sensor platform. The pod was used to control an instrument built by Cornell University for infrared astronomy. The pod completely controls a HgCdTe 1024x1024 near infrared focal plane array, the data acquisition system used to digitize the sensor pixels, the communication link to/from the host computer, and controls six motors used within the instrument to select various filters and optical configurations. The platform replaced an expensive, inflexible array controller platform and resulted in excellent noise performance, critical to signal starved astronomical applications.

2. Hardware Platform Development

The project's goal was to extend reconfigurable hardware directly into sensor platforms, not just customizable large scale computing applications. Reconfigurable hardware allows for both reconfigurable logic resources for computing *and* I/O pin control. Hence, for the first time, it is now possible to interface many different sensor and actuator platforms to the same controlling hardware. Furthermore, reconfiguring the hardware to control a different sensor, or perhaps the same sensor in a different fashion as dictated by changing conditions (wide field target search as opposed to narrow field target tracking), can be accomplished in-circuit in milliseconds. The potential for reduced volume, reduced mass, reduced cycle and re-fit time, reduced inventory and maintenance is considerable when using reconfigurable hardware. For example, UAV systems of the future may offer quickly interchangeable sensor platforms in a very simple bolt-and-go manner or perhaps various sensors can be selected in-flight as needed and controlled using the same embedded controller, simply be reconfiguring the controller.

Acquisition Systems also developed a state-of-the-art adaptive computing platform which provides significant logic resources for large scale computing applications in a low-cost, standard environment. The base platform provides 500,000 gates of user configurable logic on a CompactPCI 6U card. Furthermore, while not in our initial proposal (or budget!), we integrated two Analog Devices SHARC 21060 DSPs into the base platform.

With the combination of an embedded sensor or actuator controller, and a large scale processing platform, direct processing of real-time sensor data and actuator control is possible using the developed adaptive computing hardware.

2.1. Altera, CompactPCI, Cypress HOTLink, ZBT SRAM, AMCC 5933 PCI Interface, SHARC 21060 DSP, Daughter Card Format

Before discussing the design of each hardware platform in detail, we will first discuss why we made some of the various chip and platform choices.

2.1.1. Altera

We chose Altera for the following reasons.

1. The 10K architecture (the 20K family was not yet announced) provides Embedded Array Blocks (EABs) which are "large" blocks of SRAM, easily made into FIFOs and other extended memory structures.
2. The 10K family offers "cross-chip" routing resources which we believe is superior for non-floorplanned logic design (which will result when designing on this platform with high level languages).
3. We were familiar with the 10K architecture.

2.1.2. CompactPCI

We chose CompactPCI for the following reasons.

We wanted to develop a platform which offered considerable resources and "off-the-shelf" utility. Many previously developed systems were somewhat esoteric stand-alone platforms which were not easily adopted by others. Desktop PCI platforms are somewhat limited in mechanical ruggedness, cooling, and external I/O space. We investigated using VME as the host platform, but preferred the lower cost of CompactPCI, the emerging CompactPCI technology, and higher performance of the established CompactPCI technology vs. moving target VME specifications. Furthermore, we had experience with PCI. CompactPCI also allowed for the use of a "dual PCI bus" architecture. In summary, the base platform, a 6U Compact PCI card, has two completely isolated compact PCI busses available over the host backplane, for increased I/O to other CompactPCI hardware.

2.1.3. Cypress Semiconductor HOTLink

We chose the Cypress Semiconductor HOTLink interface for the following reasons:

Integrated I/O was very important to our overall design concept. It was clear that the link between the base and remote pods was integral to our design. Fiber was chosen for its speed and transmission line characteristics. The remote pod and base platform can be separated by hundreds of meters without concern over the high speed communication link between the remote pod and base platform (except for the obvious increased latency). Communication link bandwidth was traded off with ease-of-use, power dissipation, and physical size. We investigated Gigabit Link Module (GLM) technology, but found it would drive all end-user designs to operate at sufficiently high speeds. Meeting the roughly 52 MHz 16-bit word size data rate to support the GLM technology for applications developed with a high level compiler was *not* likely to be successful. Furthermore, our intended applications are imaging applications with data rates more often in the 10-30 MByte/second range, not 100 MByte/sec. Our platform was also to support multiple fiber links (four) to allow for use with multiple simultaneous remote pods. Physical size and power dissipation of GLM's was excessive. Cypress Semiconductor's HOTLink chipset was found to be satisfactory. It also held the added promise of a fully CMOS low-power version, but this has failed to appear from Cypress Semiconductor.

2.1.4. Memory Selection

The type of memory used in the base and remote pod platforms was driven by numerous issues. SDRAM was desired for its large size and low cost. However, use of SDRAM required either a dedicated SDRAM controller or use of logic resources within the platforms reconfigurable resources to support the SDRAM. This latter was deemed unacceptable for two reasons (absolute requirement of resources and unknown affect of end user design on SDRAM controller speed). SDRAM's inherent refresh requirement also implies the requirement of wait states in memory access. This was undesirable, as it would add a complication to all end user programs. SRAM is desirable for its high speed and static nature. However it is limited in density and is expensive. It was determined that the use of newly available Zero Bus Turnaround (ZBT) Synchronous SRAM was the best choice for the base platform. While at the time of hardware design SSRAM density was limited, the hardware platform was designed to accommodate the expected higher density components when available. The base platform was designed and fabricated prior to the availability of SSRAM and hence asynchronous SRAM was used. Furthermore, the requirement to operate with large format infrared focal plane array detectors (1024x1024) demanded large memories for the remote pod.

2.1.5. AMCC 5933 PCI Interface

This design selection received extensive review. The PLX 9080 was released during the design cycle for the base platform. The 9080 was extensively considered for the platform. In fact, a complete base platform design was investigated using 9080 PCI interfaces and shared memory (using either dual ports or more likely bus switched designs) between the 9080 and reconfigurable resources. This design was abandoned however due to the significantly increased complexity and resulting risk to the on-budget success of the project. The initial proposal called for 5933 interfaces, with which we were familiar and satisfied. It is clear however that a shared (bus switched) interface with the 9080 and platform FPGAs would offer many advantages.

We quickly dismissed using the reconfigurable resources as the PCI interface. End user designs would certainly affect the timing requirements of the PCI interface within the device. This complication is not desired, although some interesting opportunities for tailored PCI interfaces would be possible. Using a second FPGA, not part of the end user reconfigurable resources, was also investigated. However developing a PCI interface within an FPGA is not the design goal of this project or the end users of this platform.

An off the shelf ASIC provides an excellent interface to the platform. The 5933 also provides many forms of communication between the reconfigurable

resources and PCI bus, which proved useful for housekeeping design (configuration of the resources, communication with DSPs, etc.)

2.1.6. SHARC 21060 DSPs

During initial work on the project, it became clear that floating point operations would suffer in the reconfigurable resources. Furthermore, wide fixed-point multiplication operations are also large (and hence slow) in FPGAs. While not in the original proposal (or budget) for this project, we investigated the inclusion of floating point DSPs into the platform. The TI C40 and Analog Devices 2106x families were compared, and the SHARC family was found to be superior. Its large internal memory would allow for the expected method of interface between the platforms FPGAs and DSPs. The newer C62xx family was not yet available at the time of design, and hence was not selected.

2.1.7. Daughter Card Format

The base platform daughter card format was investigated. It was suggested that the format could be the PCI Mezzanine Card (PMC) format. This was considered and investigated but rejected for a few reasons. One was space constraints, as this would have demanded clearance heights and undesired component placement. We also were opposed to using a physical form factor specification identical to a popular standard but *NOT* that standard. It was conceivable that a PCI interface could have been created within the reconfigurable resources of the platform and hence end users could use off-the-shelf PMC hardware directly, however development of such an interface would interfere with end user custom logic configurations and add undesired difficulty. A high quality connector was selected and 90 I/O pins are available on the connector directly from the FPGA, as well as +3.3V and +5V. In the original proposal, the fiber optic links were to be fixed to the base platform and not on daughter cards. However, it was realized that end users might desire to use other I/O formats and not communicate directly with the remote pods. For example, a FPDP daughter card could be readily designed and allow for integration into other systems via the FPDP standard.

2.2. Remote Pod

The remote pod is a standalone printed circuit board which was designed to be embedded into sensor or actuator control systems and link back to the base platform via a fiber optic link. An FPGA has access to memory, a communication port back to the host, and numerous I/O pins to the outside world. The FPGA boots from a local EPROM at power up and can be reconfigured over the fiber at any time. An onboard PLD is used to configure the FPGA from the fiber receiver.

It is the I/O pins which give the remote pod its versatility. It was intended that the end user would design the remote pod into a developing or existing system by accommodating the remote pod's footprint into the new design. The remote pod then "plugs in" via sockets in the developing system or can be permanently soldered. The remote pod's I/O pins can then be routed to various components in the developing system, for example digital control signals, analog to digital converters, digital to analog converters, etc. The FPGA is then configured as desired to control the system. The FPGA also has access to the memory and fiber interface on the remote pod, proving a compact yet powerful and flexible control platform. The same remote pod can be attached to a different sensor platform, reconfigured, and control a completely separate device.

The board is physically small, measuring 3.875" x 5.125" and requires a single +5V supply to operate all functionality. The design could be more compact, but is limited by the large SIMMs and low density connectors. The design block diagram is in Figure 2 below. I/O pins are available from the FPGA via two 100 pin connectors. These connectors are low density .1"x.1" connectors with Ground-Signal-Signal-Ground pin configurations. Power can be supplied via these connectors or through a separate screw-terminal post for bench top development.

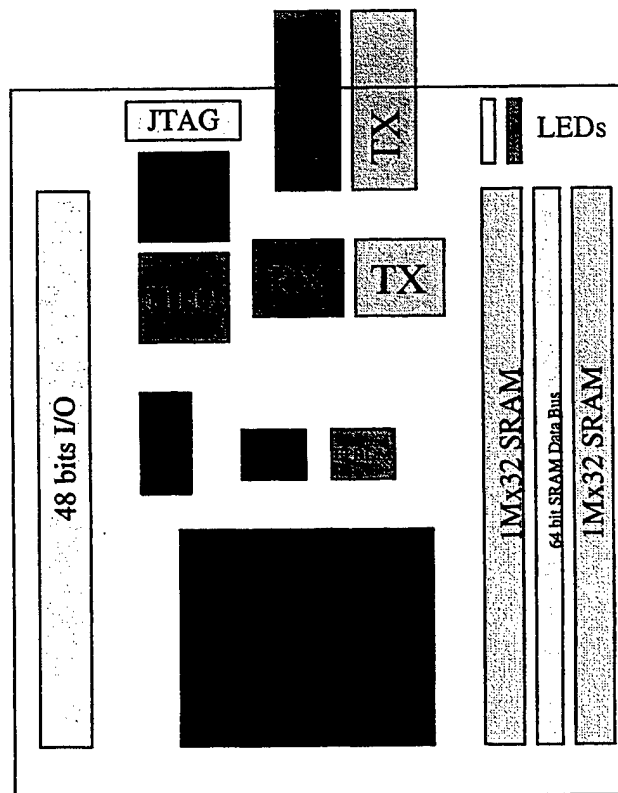


Figure 2. Remote Pod

Photo 1 below is a photo of the remote pod. The fibers can be seen exiting to the top of the photo. Power is being supplied in this stand-alone mode via the wires entering the bottom of the photo. When integrated into a system, power is supplied via the 100 pin connectors.

The design is simple. A single Altera 10K50 FPGA has dedicated access to 1Mx64 bit wide SRAM via two 72 pin SIMM sockets, a Cypress Semiconductor HOTLink chipset, and 48 bits of dedicated I/O are available on the daughter card connector. All components on the pod are clocked from a single clock source. Each component is discussed in detail below.

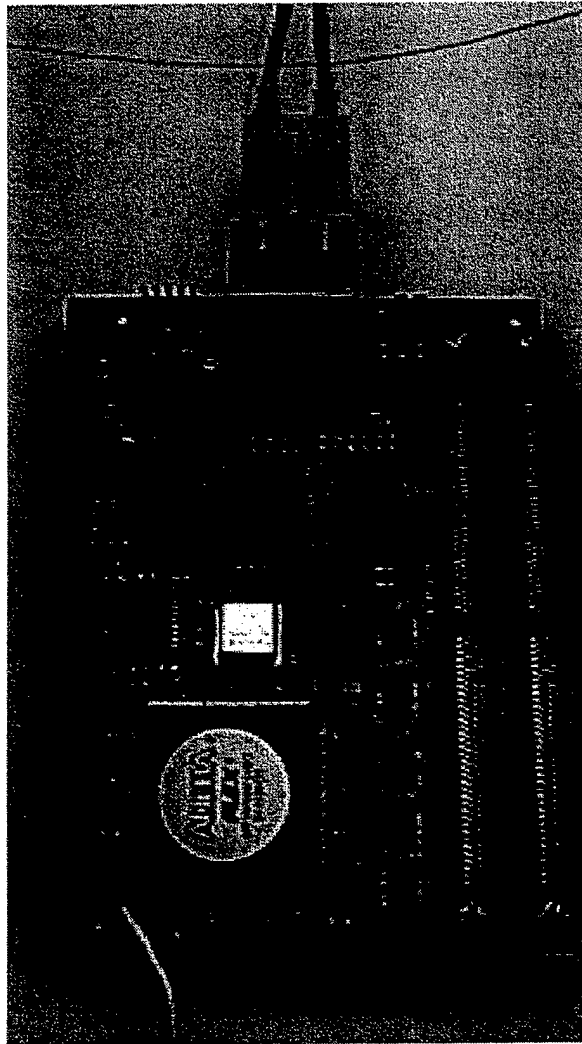


Photo 1. Remote Pod

2.2.1. 10K50 FPGA

An Altera 10K50 (50,000 nominal logic gates) is used as the remote pod FPGA. The 240 RQFP footprint will accept a wide range of compatible FPGAs, ranging from the 10K30 (30,000 gates) to the 10K70 (70,000 gates) without board modifications. The FPGA is configured at power up from the local EPROM and then may be reconfigured at any time over the fiber interface.

2.2.2. SRAM

The FPGA has direct and dedicated access to 1Mx64 bit wide SRAM via two 72 pin SIMM sockets. The 64 bit data bus to this memory is also available on a 100 pin low density connector (.1"x.1"). This allows the 64 I/O pins from the FPGA to be used for other functions besides SRAM data bus pins if the SRAM is not required. It is also possible to populate only 1 SIMM socket or populate the SIMM sockets with other memories or custom hardware. The SRAM address and control lines are not brought to the SRAM data bus header connector. SRAM SIMMs are available off-the-shelf in various speed grades. It is also possible to build custom memory modules in the SIMM sockets.

2.2.3. Cypress HOTLink Interface

The Cypress Semiconductor HOTLink interface provides an 8 bit communication link. The chipset employs internal 8B/10B encoding and serializes an 8-bit word into a 10-bit stream. This is then transmitted via the fiber link to the receiver, where the 10-bit stream is parallelized into an 8-bit byte. The HOTLink chipset therefore is a byte wide link. It also supports a number of "special characters" which can be used to establish command sequence or communication protocols. A ninth bit is written to the chipset which indicates whether the output byte is a data character or special character. Only a handful of special characters are valid, so this is not a true 9 bit link. The HOTLink receiver provides a DATARDY strobe when it has valid data ready. The receiver also indicates if the incoming byte is a special character or data character.

A Cypress Semiconductor HOTLink interface is available to the FPGA. The HOTLink transmitter is directly connected to the FPGA. However incoming data from the HOTLink receiver is first intercepted by the PLD. The PLD writes incoming data to a FIFO which is then connected to the FPGA. The PLD is used to both configure the FPGA over the fiber (when the FPGA is obviously not able to control the incoming data stream) and also to allow for FPGA resets (setting the FPGA into an un-configured state, ready for reconfiguration over the fiber). It should be noted that this PLD can be programmed in-circuit using the JTAG interface header on the remote pod and the Altera ByteBlaster. End users can customize the PLD receiver interface if desired.

It is assumed the end user programming at the hardware level will be familiar with the operation of the HOTLink chipset or will refer to the Cypress data sheets. The chipset is easy to use and a quick summary has been provided here. The transmitter is clocked synchronously with the FPGA clock and a transmit enable pin is driven low to transmit a byte over the fiber link. The receiver data is available to the FPGA from a synchronous FIFO. The FIFO status flags are available to the FPGA.

As delivered, the PLD interprets a special character 02 as the command to reset the FPGA (drive it into the non-configured state). Any following received bytes are expected to be configuration data for the FPGA and are written to the FPGA as configuration data. When the FPGA is again configured, the PLD writes incoming data to the FIFO for FPGA reception.

2.2.4. I/O Pins

48 I/O pins are available on a 100 pin connector for integration into an external platform. These I/O pins are direct from the FPGA. They are unbuffered and can be used as bi-directional pins. Care should be taken when driving external loads with these unbuffered pins directly. They should be treated as FPGA I/O pins.

2.2.5. Clock Distribution

The remote pod is clocked by a single clock source. This source can be the on-board oscillator or an externally supplied clock. In either case, a clock distribution chip is used to distribute the CLK and CLK/2 (the input clock divided by two) across the platform. Solder jumpers located on the pod printed circuit board select where CLK or CLK/2 is used for the HOTLink interface. The FPGA is always provided with both CLK and CLK/2. The on board oscillator clock is also available on the 100 pin connector. This allows for a design utilizing multiple remote pods to run off the same clock or the entire external system to run synchronous to the remote pod.

2.2.6. Boot EPROM

At power up the remote pod will configure from a local EPROM. This is an Altera EPC1 EPROM which is one-time programmable. Altera or third party programmers must be used to program this EPROM. End users must program an EPC1 device with an initial power up FPGA configuration. Reconfiguration via the fiber link is available any time after initial power up. Note that in conditions of brown out the FPGA may be configured from the EPROM.

2.2.7. JTAG

The Altera FPGA can be configured via the JTAG port using the Altera ByteBlaster cable interface. This method of configuration can be very useful

during development of a stand alone application where the fiber is not used. The onboard PLD can also be programmed via the JTAG port.

2.3. Base Platform

The base platform is a 6U CompactPCI card which provides considerable logic resources for adaptive computing applications and an interchangeable daughter card port intended for I/O applications. When used with the developed HOTLink fiber interface daughter cards, a single base platform can access four remote pods concurrently.

The base platform block diagram is show in Figure 3 below. The design is divided into two nearly identical halves, named I and II. Each half consists of an Altera 10K250A (250,000 gates) FPGA, an AMCC 5933 PCI interface, two independent banks of 128Kx36 ZBT SSRAM, a 64Kx36 bank of dual port SRAM with a SHARC 21060 DSP sharing the SRAM, daughter card interface, and a debugging port. Each half has access through a AMCC 5933 to a PCI bus. The platform was designed using Ziatech Corporation's dual PCI bus specification.

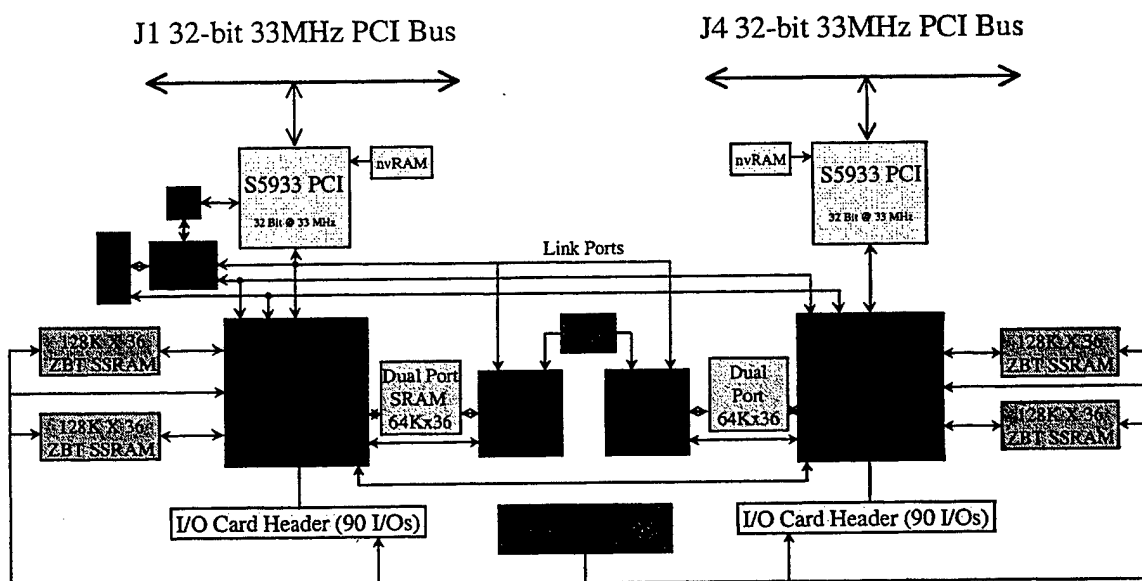


Figure 3. Base Platform

FPGA configuration, DSP programming, oscillator programming, and platform house keeping functions are accomplished through 5933 I. Pass thru regions 3 and 4 of 5933 I are decoded by the on-board PLD. The PLD passes the data to the proper location, whether it be FPGA configuration data, DSP programming or communication, or oscillator frequency programming. FPGA I does not have access to pass thru regions 3 and 4 but has access to all remaining 5933 I functionality.

The two FPGAs are connected together through 40 I/O pins. The two FPGAs have identical pin configurations and therefore an FPGA configuration may run unchanged on either FPGA. The limitation to this is the 40 pin interconnection between the two FPGAs. The end user must be certain that these I/O pins will not drive in conflict between the two FPGAs. Also, pass thru region 3 and 4 are not available to FPGA I as they are used by the on-board PLD. Furthermore, the user must be certain the attached daughter cards are identical (the daughter card connector pins are identical).

A programmable oscillator (programmable through the PCI bus) provides the synchronous clock used by the entire system. Each FPGA, the ZBT SSRAM, and the daughter card ports are provided copies of this programmable clock. All of these resources run synchronous to this clock. The FPGA can interface with the AMCC 5933 either synchronous to the PCI bus clock (33MHz) or asynchronously. To enable a PCI synchronous interface, the PCICLK is available to the FPGA (buffered by the 5933, called BPCLK). An asynchronous interface between the FPGA and 5933 (asynchronous with respect to the PCI clock) runs synchronous to the platform distributed clock (called CLKIN). The PCI synchronous clock provides for higher PCI throughput (achieving PCI bus mastering throughputs of over 100 Mbytes/sec) but requires the synchronization within the FPGA of the PCICLK and the platform distributed clock.

The design is a mixed +5V and +3.3V design, with nearly all components running at +3.3V. The AMCC 5933 PCI interface chips and platform boot PLD are the only +5V components. The Altera 10K250A FPGA's are +3.3V core devices with +5V tolerant I/O pins.

Photo 2 below is a photograph of the base platform. Note that the fiber daughter card is attached to FPGA I. The daughter card port for FPGA II is not populated in the photo. The fiber daughter card can be used on either (or both) ports.

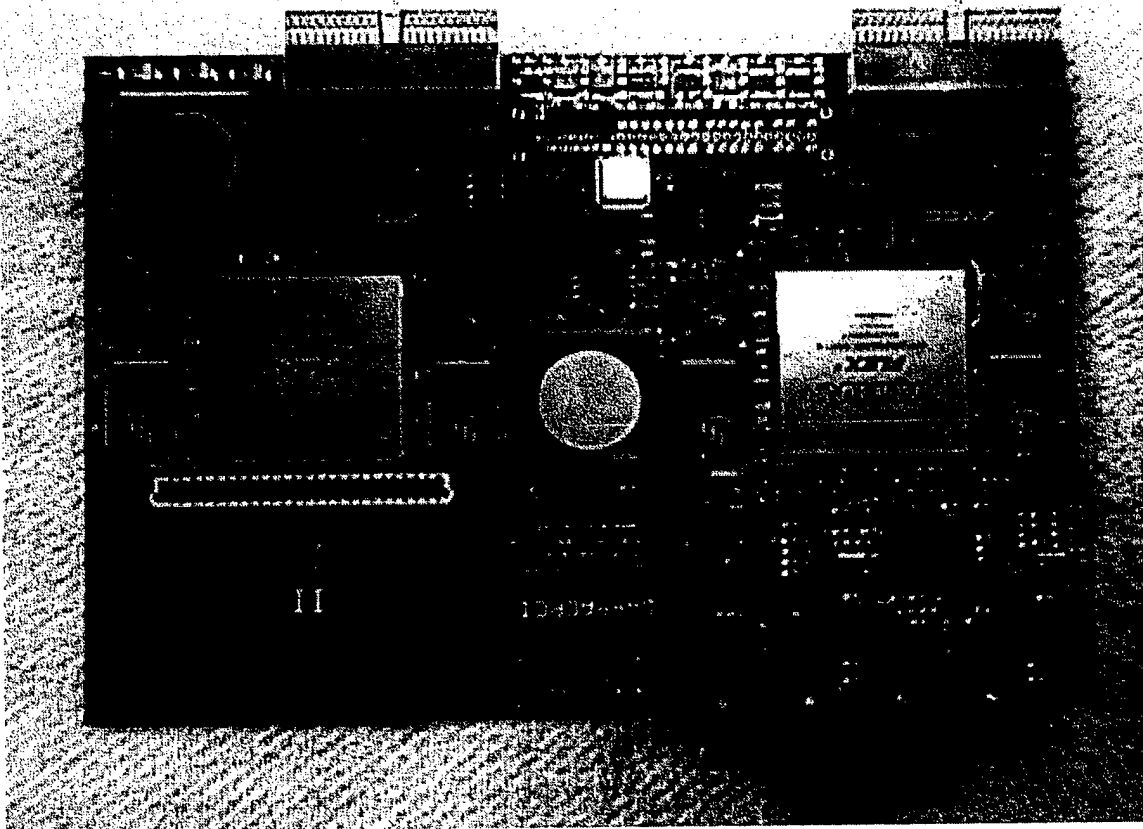


Photo 2. Base Platform with fiber daughter card attached.

The remainder of this section (2.3) will discuss the individual components of the platform in more detail.

2.3.1. 10K250A FPGAs

Altera 10K250ABC-600 FPGAs have been selected for this platform in a 600 ball BGA package. The devices use a +3.3V core voltage and are +5V-tolerant. The devices were the highest density logic devices soon to be available during the design phase of this hardware platform. Offering 250,000 gates of "nominal" logic each FPGA provides 12,160 logic elements (4 input LUT and register), 20 Embedded Array Blocks (totaling 40,960 memory bits), and over 400 I/O pins.

2.3.2. 5933 PCI Interface

The base platform was designed to work with Ziatech Corporation's 6U CompactPCI Dual PCI Bus architecture. In this specification, the J1 and J4 connectors on the CompactPCI backplane each provide their own independent PCI bus. This allows the host single board computer to access PCI peripherals on

two independent PCI busses, providing for more PCI peripherals and higher system bandwidth.

The base platform has access to each PCI bus using an AMCC 5933 PCI interface. The 5933 provides a flexible interface to the PCI bus. Each FPGA has direct connections to the 5933 add-on bus (backside) control pins. The FPGA is programmed by the end user to use the desired 5933 functions. The 5933 is capable of bus mastering at speeds exceeding 100 Mbytes/sec into PC system memory. The 5933 sports a bi-directional FIFO interface to the PCI bus, mailbox registers, pass thru regions (allowing memory mapping into the host computers address space), and various interrupt generation/notification schemes. It is expected that the end user developing on the platform from a hardware level will be familiar with the AMCC 5933.

2.3.3. ZBT Synchronous SRAM

Each FPGA has access to two independent banks of 128Kx36 bit Zero Bus Turnaround Synchronous SRAM memories. These memories operate synchronously with the system clock, called CLKIN. Read and write operations can be performed on every clock cycle, including back to back read and writes without any wait states.

The ZBT banks, referred to as ZBTA and ZBTB, are completely independent. Each FPGA has dedicated access to these memories.

The base platform was designed to accommodate ZBT memories up to 512Kx36 in density. Two extra address pins have been provided in the platform design.

2.3.4. Dual Port SRAM

Each FPGA has access to a bank of 64Kx36 dual port asynchronous SRAM. This SRAM serves as the primary interface between the FPGA and an Analog Devices 21060 SHARC DSP. The asynchronous SRAM is directly connected to the FPGA on one port and a SHARC DSP on the other.

The dual port memory has an interrupt generation capability between the two users of the SRAM. When the FPGA writes to a specific location, the SHARC DSP will receive an interrupt. Likewise, when the SHARC writes to a specific location, the FPGA will receive an interrupt.

The primary intended interface arrangement between the FPGAs and SHARCs will be discussed in detail in Section 2.3.5.1 below. However, the dual port memory provides a shared memory between the FPGA and DSP and the end user can use this relationship in any manner.

The use of synchronous dual port SRAM was desired. However, during the design cycle for the base platform no synchronous SRAMs were available which met the timing requirements of the SHARC DSPs.

2.3.5. 21060 SHARC DSP

FPGAs are not very well suited for high speed, wide word multiplication or floating point arithmetic. Numerous logic resources within the FPGA are consumed and speed is slow. On the other hand, multipliers and floating point units are often required in many signal processing applications. This requirement has led to the development of ASIC circuits dedicated to multiplication, division, floating point operations, etc. In order to provide the resources for optimal system throughput, we decided to include a dedicated ASIC circuit to perform these calculations.

We opted to use a SHARC DSP as a "programmable ASIC". The SHARC DSP is a 32-bit floating point processor with 4Mbits of dual ported internal SRAM. It performs single cycle memory fetches in parallel with single cycle multiplications. We designed the SHARC into the system interfacing with the FPGAs through a dual port SRAM for primary data exchange. Dual port interrupts are used to synchronize the FPGA and DSP.

Programming of the SHARCs occurs over PCI bus I from the host computer via the SHARC Link Port. SHARC programs are written using the Analog Devices DSP development software.

2.3.5.1. Dual Port SRAM

The primary data interface between the FPGA and the DSP is the dual port SRAM. SHARC Link Ports also connect between the FPGA and the SHARC, but Link Ports are only 4 bits wide and are NOT intended as a primary data transfer mechanism. They are discussed more in section 2.3.5.2 below.

The primary synchronization method between the FPGA and SHARC is use of the dual port interrupt. When the FPGA writes to dual port location FFFF, the SHARC will receive an interrupt (SHARC IRQ0). The SHARC clears this interrupt by reading dual port location FFFF. When the SHARC writes to dual port location FFFE, the FPGA will receive an interrupt signal on FPGA input pin DPnINT. The FPGA clears this interrupt by reading location FFFE.

We intend to use the SHARCs as programmable arithmetical ASICs. A program will be running on the SHARCs. This program is to be fetched completely from SHARC internal memory. Data will be downloaded into the dual port memory and the SHARC or FPGA will be signaled of the availability of the data by writing to the dual port interrupt generating register.

For example, we imagine writing 64K dwords from the FPGA into dual port memory and then reading the products of these dwords. It is equally easy to write samples to the dual port and read back a completed FFT from the dual port, allowing the SHARCs to perform the DSP using the optimized ASIC. The FPGA resources could then be used for pattern matching on the power spectrum, perhaps looking for a specific radar signature.

This combined FPGA and DSP architecture provides the end user with the ability to run algorithms on the proper hardware. In the example above, the SHARC DSP is more likely superior in executing a 32 bit floating point FFT whereas the FPGA resources are better suited to a bit-level pattern matching routine in identifying the measured power spectrum.

2.3.5.2. Link Ports

Link Ports are SHARC communication ports. Six 4 bit link ports are part of the 21060 DSP. Link ports are intended to be used for multiple SHARC processor farms for processor to processor communication and additional external I/O.

Link ports are used in the base platform to allow for communication between the SHARC and FPGA, FPGA and SHARC, PCI and SHARC, SHARC and PCI, and SHARC to SHARC. The link ports are assigned as:

SHARC LINK PORT	DESTINATION
0	OTHER SHARC PORT 1
1	OTHER SHARC PORT 0
2	FPGA TO SHARC communication
3	SHARC to FPGA communication
4	PCI to SHARC communication
5	SHARC to PCI communication

It is not intended to use the FPGA and PCI link ports as primary data transfer routes as they are not intended for maximum throughput. However they do provide very useful communication and synchronization pathways between the various components of the system.

An enhanced design using the link ports was considered for this platform, but was not pursued to budgetary constraints. Inclusion of DSPs into the base platform was not in the original proposal or budget.

2.3.6. Programmable Oscillator

The base system clocks synchronously to a distributed clock, named CLKIN. The rate of this clock is programmable by the end user over the PCI bus. The rate of this clock is selected by the end user to comply with the timing requirements of

the specific design. The clock range can be from approximately 300 kHz to 100 MHz in fine, high resolution steps.

2.3.7. Daughter Card Interface

The daughter card interface is a 100 pin, high quality connector. The connector used employs internal ground planes which are carried continuously from base platform to daughter card board. This provides a high quality signal environment as well as allows all 100 pins to be used for signaling.

90 of the connector's pins are directly connected to FPGA I/O pins. One pin is clocked at the system distributed clock and one pin is an FPGA dedicated input pin. Power is supplied to the daughter card via 4 pins at +5V and 4 pins at +3.3V. This is summarized below.

Number of Pins	Function
90	Direct I/O pins from FPGA
1	Distributed system clock CLKIN
1	Dedicated input to FPGA
4	+5V
4	+3.3V

2.3.8. Debugging Port

An additional header was designed into the base platform accessible from the CompactPCI front panel. This header has 8 pins connected directly to 8 I/Os on each of FPGA I and FPGA II. The header also supplies GND and +5V. This header is intended to serve as a debugging port, allowing end users easy access to these 8 pins for logic analyzer or oscilloscope probing. It is also possible to build a small indicator board to use this header as a status monitor for end users, for example a segmented display or simple LEDs.

Test point vias are located throughout the base platform at strategic places. Probing of these test points requires operation of the base platform on extender cards out from the host CompactPCI chassis. A list of these test points is located in the User's Guide documentation.

2.4. Daughter Card Fiber Interface

During this effort a fiber optic daughter card was developed. The fiber daughter card provides two independent fiber transmit and receive channels, each utilizing the Cypress Semiconductor HOTLink chipset. Each fiber daughter card can interface to

two remote pods. The base platform can accept two daughter cards, allowing one base platform to communicate with up to four remote pods.

Figure 4 below is the fiber daughter card block diagram. Each channel of the fiber interface appears as two synchronous FIFOs to the base platform FPGAs. The FPGA interface simply reads or writes to the synchronous FIFOs (synchronous with the base distributed clock CLKIN).

Each channel on the daughter card is identical to the other. The TX FIFO is read by the PLD and the data is transmitted out the transmitter. The RX FIFO is written by the PLD when data is received from the receiver. Note that each channel is clocked by its own oscillator. This permits the two channels to operate at a different clock frequency if desired. Note that a HOTLink requirement is that both ends of the fiber optic cable must run at the same clock speed. Therefore with this arrangement two remote pods can be controlled at different speeds with respect to each other and a different speed from the base platform.

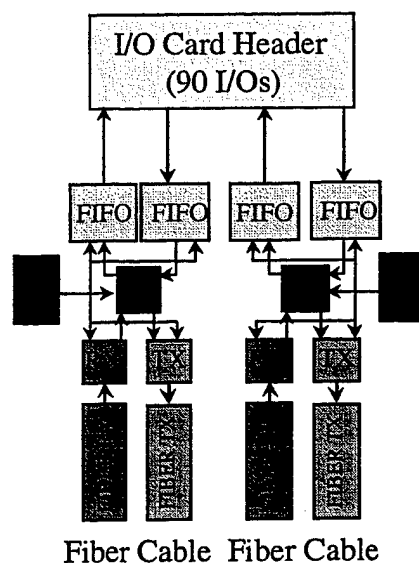


Figure 4. Fiber Interface Daughter Card

Oscillator 1 and 2 can operate up to 27 MHz in speed. This limitation is due to the electro-optical TX/RX used on the daughter card. If these devices were upgraded to a higher speed grade, operation up to 40 MHz is possible. This requires the fastest speed grade HOTLink chipset, and correspondingly faster FIFOs and PLDs.

Throughput of the daughter card as designed with 270 Mbits/sec electro-optical components is 27 Mbytes/sec in and 27 Mbytes/sec out. The channels both operate in full duplex and in parallel, providing up to 108 Mbytes/sec of sustained I/O per daughter card. Note that this is not limited by distance from remote pod to base platform, up to a fiber optic cable length of a few hundred meters.

Photo 3 below is a photo of the fiber daughter card.

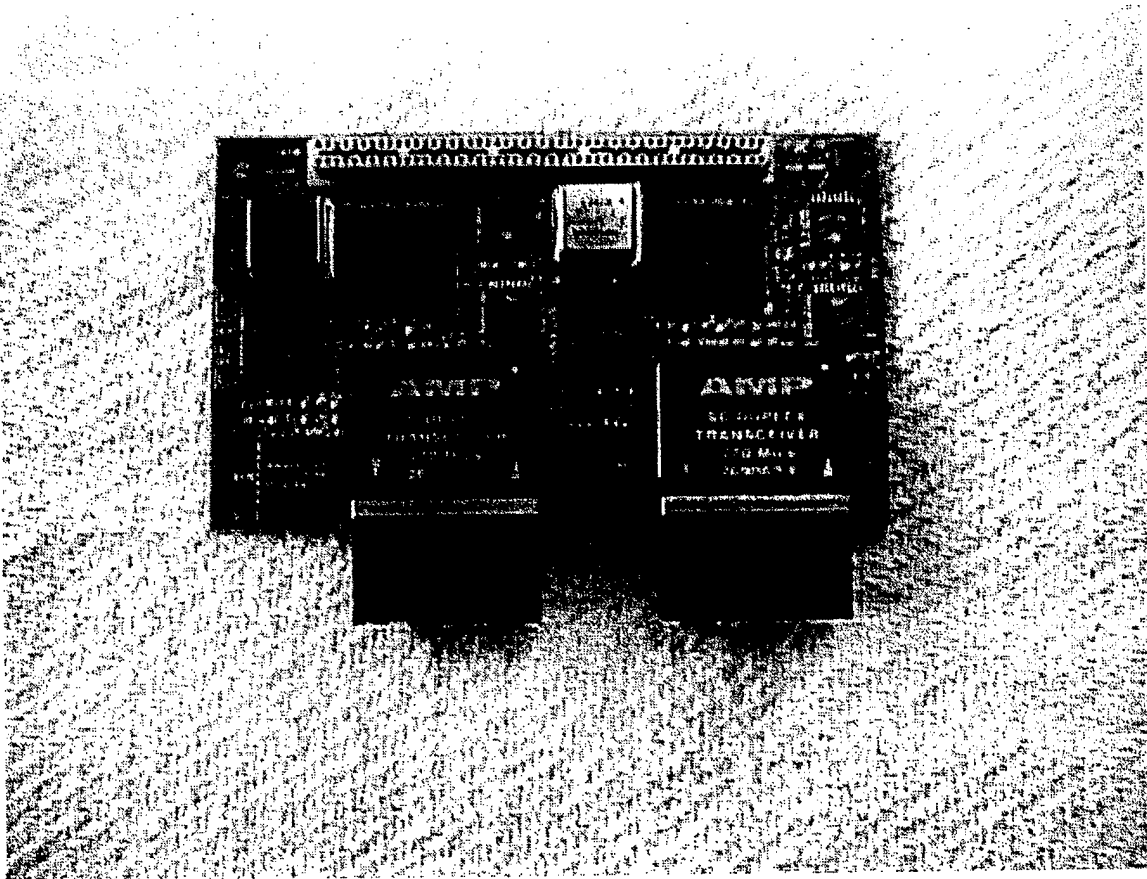


Photo 3. Fiber Daughter Card.

The PLD is not necessary in the above design, but was used to allow for maximum end user flexibility when using the daughter card interfaces. The PLDs can be programmed in-circuit using Altera's ByteBlaster cable and the daughter card JTAG interface port. For example, the PLD can be used to implement a hardware "stop sending" and "start sending" scheme (by monitoring the FIFOs status flags and using HOTLink special characters), independent of the base platform FPGAs. There are also unused I/O pins directly from the daughter card's control PLDs to the base platform's FPGA, via the daughter card connector, bypassing the FIFOs.

3. Software Platform Extension

Project software development involved two areas: host system hardware control and high level programming of reconfigurable hardware.

When the project began, Acquisition Systems had developed an appropriate graphical user interface and interactive interpreter environment for Windows. For this project, core additions to the user interface application were made to support the new hardware platform.

A reconfigurable hardware compiler, under license from Cornell University, was ported for use with this hardware platform.

3.1. ASbridge Host Interface

ASbridge is a Win32 application that communicates with Windows NT device drivers for hardware access, interrupt handling, and other host system resource control. Graphical components provide intuitive functionality such as PCI interface register views, FPGA and DSP configuration download, and PCI interface nvRAM configuration. For programmatic interaction with the hardware, a comprehensive command line interpreter is a central component.

The base platform resides on a CompactPCI system with two (2) independent PCI busses. As described elsewhere, the base platform has 2 similar FPGA's, each with an associated DSP, daughter card, etc. These FPGAs interface to the separate PCI busses. Host software provides symmetric access to the PCI side of each of these 2 base platform PCI interfaces.

The overall intent is to provide a user environment for hardware application development and testing.

3.1.1. General Program Design

Host platform hardware resides in a CompactPCI system. A host computer supporting a standard operating system interacts with this hardware through PCI interfaces, in this case AMCC S5933 PCI interface devices. ASbridge runs under the Windows NT operating system and is designed to handle multiple S5933 PCI interfaces on multiple PCI busses.

There are two (2) NT device drivers. One provides functionality related to PCI Configuration Space, such as the "plug&play" detection of multiple Acquisition Systems devices on multiple PCI busses. The other driver is handed a list of devices which it then uses internally. The application and this second device driver then use an index into this device list to refer to the individual devices. All

driver functionality used by the ASbridge program is exposed and documented for end-user custom application development (see the files [asriocfg.txt](#) and [asriopci.txt](#) for reference documentation for the two NT device drivers).

All device driver calls are through the DeviceIoControl function using custom control codes exclusively. Asynchronous communication from the device driver, optionally initiated in a hardware-generated interrupt service routine, is via user-mode Asynchronous Procedure Calls (APC).

The S5933 contains an nvRAM (non-volatile random-access memory) component that is used on power-up for setting certain device parameters. ASbridge makes use of otherwise unused portions of the nvRAM for user-specified identification of the device. In particular a user-settable byte-wide device "ID" allows programmatic access to each S5933 in the system unambiguously, even after changing the devices' physical locations in a PCI bus/slot hierarchy.

The versatility of ASbridge lies in its command line interpreter. This is a FORTH-like pseudo byte-code compiled language interpreter. It is FORTH-like, and not a true byte-code language, in that compiled byte-code is not transferable to another environment, i.e. the compiled code uses non re-locatable addressing. It is compiled for speed of execution only.

3.1.2. Platform-Specific Extensions

The command line interpreter (as well as the graphical user interface) has functionality common to any S5933 PCI interface, and also functionality particular to this project's base platform hardware. For base platform FPGA-specific functionality, such as status register interpretation, download of 10K250 FPGA configurations, etc., a family of commands with the word "base" in their names was developed. Similarly, for base platform DSP-specific functionality, the commands include "sharc" in their names.

Following are listings of these ASbridge interpreter commands. Programmable clock control is listed as FPGA-related since this clock's output is available to both FPGAs. It is used, for instance, to drive Hardware Promela applications (Sect 3.2).

FPGA-related command	functionality
basettf	read a 10K250 Altera Tabular Text File into one of the sixteen host memory buffers
baserbf	read a 10K250 Altera Raw Binary File into one of the sixteen host memory buffers
baseshowconfigs	display current status of all non-empty configuration buffers, including source file name and download (FPGA configuration) status
baseconfig1	configure FPGA1 from one of the sixteen host memory buffers
baseconfig2	configure FPGA2 from one of the sixteen host memory buffers
baseconfigboth	configure both FPGA's simultaneously from the same host memory buffer
basereset1	reset (un-configure) FPGA1
basereset2	reset (un-configure) FPGA2
baseresetboth	reset (un-configure) both FPGA's simultaneously
basesetclk	set one of three registers in the programmable oscillator to specify a specific frequency
baseuseclk	set one of the three frequency registers of the programmable oscillator to be active, i.e. clock begins output at registered frequency
basestep	single-step the programmable oscillator. assumes the frequency has been set to zero (0)
basestatus	read and interpret base platform status register

SHARC-related command	functionality
sharcldr	load an Analog Devices ldr file into one of 16 SHARC configuration (program) host memory buffers
sharcconfig1	configure (program) SHARC1 from one of the SHARC configuration buffers
sharcconfig2	configure (program) SHARC2 from one of the SHARC configuration buffers
sharcshowconfigs	display current status of all non-empty configuration buffers, including source file name and download (SHARC programmed) status
sharcreset1	reset (unprogram) SHARC1
sharcreset2	reset (unprogram) SHARC2
sharcwlp1	write dword to SHARC1 link port

sharcwlp2	write dword to SHARC2 link port
sharcwlpboth	write dword to both SHARCs' link port simultaneously

3.1.3. Software Scatter/Gather

The S5933 does not support scatter/gather bus mastering so software scatter/gather capability was designed into the NT device driver. This works by creating a list of variable-length contiguous memory regions, each of which can support a single bus-mastering transfer. As each bus-mastering transfer completes, a hardware interrupt is generated and the interrupt service routine in the driver sets up and initiates a bus-mastering transfer for the next region in the list. We find that this software scatter/gather technique achieves good performance, as long as there is enough hardware data buffering to allow for the interrupt latencies. For large transfers with just a few sub-regions, performance is virtually indistinguishable from the equivalent single large transfer. In a somewhat extreme case of small sub-regions, a 256 Kbyte transfer, consisting of 16 separate bus-mastering transfers, is completed at an overall rate of 104 MBytes/sec. while the comparable single large bus-mastering transfer takes place at 128 MBytes/sec.

3.1.4. SHARC/Host Communication

SHARC DSP programming is handled by the host similarly to FPGA configurations. The host software (ASbridge) maintains a number of variable-length RAM buffers to contain programs compiled for the SHARC by Analog Devices' compilers. The compilers (C-compiler and Assembler) run on Intel hardware under NT. The compiled code is downloaded on demand to one of the two DSPs via a SHARC link port ("link port booting"). Once programmed, the SHARC has access to PCI data through another of its link ports and the ASbridge interpreter provides for writing 32-bit data to these link ports (one for each of the two SHARCs). The four SHARC programmable "flags" are brought out to the PCI interface and show up as individual bits in one of the PCI interface registers.

3.2. Hardware Promela Extension

Hardware Promela is a high level language capable of generating Altera Hardware Description Language (AHDL) code for Altera FPGA's. Developed at Cornell University by Dr. Geoffrey Brown, it is licensed from Cornell by Acquisition Systems. The language is a variant of the protocol description language Promela, designed by Gerard Holzmann (Design and Validation of Computer Protocols, 1991). Hardware design languages are relatively inaccessible to large numbers of computer

programmers. Hardware Promela is intended to provide a configurable hardware design environment more familiar to the typical programmer, like that provided by the "C" and similar programming languages.

One aim of this project was to port Hardware Promela to this hardware platform. This involved extending Hardware Promela to incorporate specific hardware features in the Hardware Promela programming environment.

3.2.1. Hardware Interface Library

Hardware Promela source code is run through the Hardware Promela Compiler (HPC) to generate equivalent AHDL. The resultant AHDL file, named "promela.tdf", is then input, along with related Hardware Promela files (see below), to the Altera tools MAX+Plus II to compile, synthesize, and place and route the configuration for a particular Altera device. The resulting configuration file can be downloaded directly to the FPGA.

In the Hardware Promela (HP) system, hardware platform specifics reside in a module referred to as the hardware interface library. This hardware interface library references the sub-design "promela" which has an effect similar to an "extern" declaration in C. The effect is to make the interface defined in "promela.tdf" (which was generated by HPC) available to the rest of the hardware interface library. The interface library is responsible for hardware access, synchronization, etc. In order to add new hardware functionality, the hardware interface library is extended, and users' Hardware Promela code can then refer to the new entities in the interface library.

For example, if an FPGA output pin were referred to as PINOUT, it could be made available to Hardware Promela by setting *PINOUT* = *prom.Vw_pinout_data_* in the hardware interface library and declaring *externo int: 1 pinout* in the HP source file. In the body of the HP source, the statement *pinout = 1* would set the physical PINOUT output high. This declaration ("externo int:1") results in the entry "Vw_name_data_" in the Interface section of promela.tdf. If the integer width is greater than one, the AHDL format is *Vw_name_data_[n]*, where n is the bit width. In general, a Hardware Promela external declaration generates multiple signals in the hardware description depending on its type and data width.

Hardware Promela "channels" are an integral part of the hardware interface library. They provide synchronization between elements of a user's Hardware Promela program or between the user's program and the hardware interface library. In the latter case, channels must be instantiated in the hardware library interface. A corresponding declaration in Hardware Promela user code results in the generation of the required Promela-side channel interfaces. Each external Promela channel exposes nine (9) I/Os to the hardware interface library. These include send- and receive-side request- and acknowledge-signals. Each end of the

channel specifies its own clock input. The Promela side uses the one clock specified for all of Promela user code. On this platform the Hardware Promela clock is the programmable oscillator output.

For example, for use on this hardware platform, where Altera FPGAs are wired to the AMCC S5933 PCI interface with its 8 dword FIFO, the hardware interface library monitors PCI output control signals and FIFO status registers to read and write the FIFO in concert with Hardware Promela user code. The hardware end of the channel runs off the PCI clock for accesses synchronous to the PCI bus, while the Promela end of the channel is clocked at whatever rate Promela is being driven. In the Hardware Promela user code, the statements:

```
externi chan fifo in of {int : 32};  
externo chan fifo out of {int : 32};
```

define input and output channels to read and write the (32-bit wide) S5933 FIFO.

Another component of the hardware interface library provides RAM access. The statement :

```
extern ram:32 DP[65535];
```

provides array-style access to the 64 Kdword dual port memory. The implementation involves no handshaking nor timing constructs, and thus is useful only with asynchronous memories.

Although any given Hardware Promela resource that is handled in the interface library does not require actual use of that resource in a Hardware Promela program, the Hardware Promela program file must declare the resource in order to generate all the various interface components.

3.2.2. Software Interface Library

For platforms such as this one, where the FPGA can communicate with a host processor, some intermediate software is needed to supply an interface between code run by the host processor, and Hardware Promela code on the FPGA. In one sense, ASbridge itself is such a software library (and user interface) because it provides easy host-side programmatic access to the hardware. Within ASbridge, though, one can build software modules to provide specific functionality designed for use with specific Hardware Promela programs. As a platform for application development, ASbridge interpreter code interfacing to Hardware Promela can be an effective tool.

For example, suppose we want to send data out one of the platform's fiber channels and verify the data at the remote site. We want to fill one of the ZBT

memory banks with known data, and may wish to use various bit patterns according to various tests. We write a Hardware Promela program to accept test data from the host and place it in local memory (a ZBT memory bank connected to the FPGA), and then on command from the host output some or all of that data to the fiber interface. A small amount of coding of Hardware Promela for FPGA configuration and of ASbridge interpreter code is all that's needed. All necessary code is shown below.

The fiber optic hardware provides a byte wide channel, and for simplicity the Hardware Promela code below just uses the low 8 bits of 32-bit data. To transfer all 32 bits, Promela code could easily transfer the multiple bytes sequentially. Or, for increased efficiency, the hardware interface library could accept 32 bits from Promela and write the individual bytes sequentially itself.

See the accompanying document [ASPromela.doc](#) for information necessary to write and compile Hardware Promela programs. The following Hardware Promela examples use channels and certain flow control operators. Output to an output channel named "chan_out" looks like: *chan_out ! value* . Similarly, the syntax for input from an input channel named "chan_in" to a variable named "var" is: *chan_in ? var* , spaces not required. The Promela "if .. fi" construction executes the first enclosed list that begins with a statement that evaluates to "true" (lists are preceded with double semicolons). A channel access evaluates to "false" or "true" depending on whether it would or would not block, respectively. The Promela "do .. od" construction is equivalent to "if .. fi" except that it is re-evaluated and executed repeatedly. Also note that the statement separators ";" and "->" are completely equivalent and are used alternately just for readability.

See section 3.2.3.2 for details of Hardware Promela handling of ZBT memory, as used in the example below. The accompanying file [fiber.p](#) contains the listing below as well as the unused external declarations necessary to compile this program. The complete external listing also appears in section 3.2.3.5.

Hardware Promela program for fiber output

```
// File: fiber.p
// Hardware Promela program to store host-supplied data
// and output it on demand over fiber channel

externi chan fifo in of {int : 32}; // fifo input
externi chan apt in of {int : 32}; // PT1 in (1st passthru region)
externi chan bpt in of {int : 32}; // PT2 in (2nd passthru region)
externi chan azbt in of {int : 32}; // memory read bank A, increments address
externo chan azbt out of {int : 32}; // zbt memory write bank A, increments address
externo chan azbt base of {int : 17}; // set base address
externi chan afiber in of {int : 9}; // data input, channel A
externo chan afiber out of {int : 9}; // data output, channel A

<unused external declarations (see Section 3.2.3.5) left out for brevity>

int: 17 address; // define variable to hold address
int: 17 count; // variable to hold transfer dword count
int: 32 newdata;
int: 9 data;

par // execute following lists ("do..od" blocks) in parallel
:: do // repeat
  :: apt in ? address -> // get ZBT base address over channel (passthru region)
    azbt base ! address // set base address for ZBT bank A
  od // end repeat block
:: do
  :: fifo in ? newdata -> // get data over channel (fifo)
    azbt out ! newdata // write one dword to ZBT bank A, increments address
  od
:: do
  :: bpt in ? count -> // get count for transfer from ZBT to FIBER
    do
      :: azbt in ? data -> // read ZBT, increments address
        afiber out ! data; // send byte out fiber
        count = count - 1; // decrement count
      if
        :: count == 0 -> break // if count is zero, break out of do
        :: skip // else nothing
      fi
    od
  od
od
```

rap

// -----

Corresponding ASbridge interpreter program to supply data and control fiber output

```
=====
# ASbridge macros to support interactive control
# of the above Hardware Promela configuration
hex                # set default numeric radix to hexadecimal
array ax           # define dword array named "ax"
20000 ax fix       # allocate contiguous memory equal to ZBT bank size,
                  # requires 17-bit addressing
macro setbase { PT1 0 put } # takes number from stack and writes Promela channel
                          # aptin, which alters ZBT memory "current" address
macro emit { PT2 0 put }   # takes number from stack and writes Promela channel
                          # bptin, which specifies number of
                          # bytes to transmit on fiber...
                          # also serves as a command to start this transmission
macro useax { 0 setbase   # set ZBT memory base address to zero
              ax sizeof 0 # place zero and size of array ax (20000 hex) on the stack
              bmo }       # write all data in ax, starting at offset 0,
                          # to Promela input channel fifoin, which Promela then
                          # transfers to ZBT memory
```

Interactive use of ASbridge interpreter to communicate with Hardware Promela

```
=====
# With the above programs loaded, interactive use of ASbridge could be as below
# (without the comments):
0 ax sizeof 1 - loop{ i ff & ax i put } # put ascending byte values 0,1,2,... into array
# (filling the array could be done by reading in a file, instead)
useax                # transfer this data to ZBT memory
0 setbase            # set ZBT base address to zero (0)
ax sizeof emit       # transmit data in ZBT bank A over fiber link
100 setbase          # set ZBT base address to 100
10 emit              # transmit 10 bytes over fiber link
# -----
```

See section 3.2.3.5 for a listing of the various I/O channels defined for Hardware Promela on this platform. See the ASbridge Software Reference Guide for complete details of the ASbridge interpreter.

3.2.3. Extensions to this platform

The Hardware Promela hardware interface library was modified and extended for

use with the current hardware platform. An Altera Assignment and Configuration file was written and the AHDL hardware interface library modified according to the FPGA I/O connectivity. Besides the address/data, status, and control lines of the S5933 PCI interface, device accesses from a single FPGA include fiber optic channels, multiple banks of random access memories, FPGA-to-FPGA ties, host-interrupt, dual-ported-memory interrupt, and SHARC- shared-memory-addressing status. All of these channels have been implemented in Hardware Promela and are discussed below.

3.2.3.1. Fiber Optic Channels

Each of the base platform's two 10K250 FPGA's has access to two pairs of fiber optic channels, each pair providing one input and one output channel. The fiber optic transceivers are connected through dedicated FIFO's to the FPGA.

Each fiber channel's dedicated FIFO has multiple status lines. It would be possible to simplify use of these fiber channels in Promela programs by handling the FIFO status information in the hardware interface library, so that Promela code could then send and receive on these channels without regard to FIFO status. The hardware library would block (temporarily stall) the process when necessary. However it is desirable to be able to monitor data transfer status in the user-level (Promela) code, especially when communicating with external systems. Therefore all fiber FIFO status flags are made available to the Promela program. These flags include *almost full* and *full* flags for each output channel and *almost empty* and *empty* flags for input channels. Input and output FIFO *reset* controls are also present for each of the two fiber I/O pairs.

3.2.3.2. ZBT Synchronous SRAM

Each of the base platform's two 10K250 FPGA's has access to two banks of Zero Bus Turnaround synchronous random access memory. This SRAM architecture was chosen for its synchronous operation and the fact that sequences that involve alternate read and write access ("turnaround") do not incur a performance penalty. With an eye to optimizing Hardware Promela's access to this memory, the usual "device[address]" syntax was not attempted.

It is assumed that where speed is desired, sequential addressing can be used, i.e. a packed block of data will be transferred with increasing sequential addressing. We provide three separate Promela channels for reading, writing, and addressing ZBT memory. The address-channel allows Promela code to set a "base address" for a block transfer. Using a read- or write-channel has the side effect of incrementing the address after the memory access. This way the hardware addressing can take place in parallel with Promela data access and performance is maximized.

This ZBT memory access method could determine the best way to allocate ZBT memory in a particular Hardware Promela application. For instance, instead of placing blocks of data that are to be alternately read and written into distinct (separate) memory ranges, it may be possible to interleave the two blocks so that alternate reads and writes access the alternate data sets. However, it seems likely that at some point one would prefer each data set to appear contiguous, e.g. for wholesale transfer over some external channel. With this in mind, it would be straightforward to add another Promela ZBT channel that would set the address increment. With such an arrangement, the Hardware Promela programmer could adjust the memory "granularity" at will. For example, if the desired data set were distributed in every fourth location in ZBT memory, setting the "granularity" to four would allow sequential access to this data set without any performance penalty.

3.2.3.3. SHARC Communication

Each FPGA shares a bank of dual-ported SRAM with a SHARC 21060 DSP, and the high address bit being used by the SHARC is made available to Hardware Promela via the *sharchimem* external variable. This allows the Promela code to monitor SHARC access of shared memory and determine when it is in the upper half of available memory. Another channel into Promela, *dpint*, comes from the dual-port-memory interrupt. This line is triggered when the SHARC writes any value to a specific address at the high end of the dual port memory address range. Similarly, SHARC programs can be interrupted when the FPGA (Promela) writes to its specific address at the high end of the dual port memory range. Cooperative programming between FPGA and SHARC can use these methods to insure program synchronization.

3.2.3.4. DMA channels

The speed at which any particular Hardware Promela program can be run is generally unpredictable until it is completely synthesized and routed. This is a general feature of such high-level approaches to reconfigurable hardware design. (See section 5.3.2 for more on this topic).

In order to provide high speed data transfer capability to Hardware Promela programs, we designed a more complex Hardware Promela "channel" to clock data transfers at a higher rate than the Promela clock. Just as DMA hardware on a PC transfers data independently of the central processor, our "dma" channel, being designed in the hardware interface library, acts independently of Promela per se. We use a Hardware Promela external integer to specify the memory base address, and a channel interface for count-initialization and completion synchronization.

One of two channels is used for transferring in a given direction, one for PCI to DP (dual port memory), the other for DP to PCI transfers. The hardware interface side of the channels and the data transfer circuitry run off the 33 MHz PCI clock, while the Promela side of the channels operate with the rest of Promela at the Promela clock rate (programmable clock). An example program using these channels is given here.

Hardware Promela program using "dma" channels

```
=====
// Hardware Promela program for storing 256 KBytes of host-supplied data,
// automatically signaling DSP that data are present in shared memory,
// then transferring data to host upon DSP-done-signal.
// Entire process repeats indefinitely.

externo chan fifo_dp of {int: 16}; // dma PCI to DP
externo chan dp_fifo of {int: 16}; // dma DP to PCI
externo int:16 dpbase;           // DP base address for dma

<unused external declarations left out for brevity>

int 1: junk; // needed to receive during channel synchronization, value not used
do
    // repeat
    :: dpbase!0 -> // set Dual Port (shared with DSP) memory base for "dma" transfer
    fifo_dp!65536; // do transfer from PCI host to DP memory; DSP gets interrupt
    // when high memory address is written
    dpin?junk;    // wait for DP interrupt when DSP writes its high memory address,
    // presumably after processing data
    dpbase!0 -> // set DP memory base for next transfer
    dp_fifo!65536 // do transfer from DP to PCI host
od
// -----
```

Corresponding ASbridge interpreter interactive use to supply data for DSP processing

```
=====
# ASbridge interactive communication
# with the preceding Hardware Promela configuration

decimal      # set default numeric radix to decimal
array ax      # define dword array named "ax"
65536 ax fix   # allocate contiguous memory equal to transfer desired
ax sizeof 0 rdec c:/data/data_file # read data into array from ASCII decimal file on disk
ax sizeof 0 bmo # transfer all data into fifo, to be written by Promela to DP memory
ax sizeof 0 bmi # transfer all (processed) data back to the original array
ax sizeof 0 wdec d:/processed/proc_file # write to disk
```

#

This sort of interaction could continue indefinitely

On the current platform, this type of "dma" channel can be applied only to the asynchronous dual port (DP) memory, since the ZBT memories and fiber optic FIFOs are all synchronous with the Promela clock. Experimental implementation of these channels also entailed the removal of the Hardware Promela "ram" interface to DP memory. For these reasons, these "dma" channels are not present in the demonstration version of Hardware Promela.

3.2.3.5. *Summary of Hardware Promela Extensions*

We have ported Hardware Promela in a straightforward manner for use on this hardware platform. We have experimented with platform and application specific extensions such as providing high- speed DMA-type data transfer to the user Hardware Promela code.

Hardware Promela programs can run on either (or both) of the platform's two 10K250 FPGAs. On either FPGA, Hardware Promela has read/write access to the DP memory shared with a SHARC DSP. It can also tell when the SHARC is accessing the upper half of shared memory, as well as detecting interrupt notification from this shared memory which can be generated by a SHARC access. On either FPGA it also has access to the two banks of ZBT memory and four separate fiber optic ports (2 read, 2 write). Hardware Promela can interrupt the host processor via the PCI interface, and fiber optic channel FIFO flags are available to Hardware Promela so that programs can monitor data transfer status with external devices.

Below is a table of hardware channels available to Hardware Promela on the base platform. Although FPGA-to-FPGA communication is possible and has been tested, this requires that *different* Promela configurations be run on the two FPGAs. One version must treat the relevant I/O's as inputs while the other configures them as outputs. Furthermore, mistakenly setting both to outputs could result in physical damage to the platform hardware. For this reason, FPGA-to-FPGA communication is not included in the demonstration version of Hardware Promela.

The list below conforms to the syntax of Hardware Promela declarations. This list is necessary in any Promela program that is to link with the supplied dcpcihp.tdf hardware interface library. The "externi" declaration indicates INPUT (read access) and "externo" indicates OUTPUT (write access). The number in curly braces is the channel bit-width.

Hardware Promela hardware channels on Base Platform

```

externi chan fifoin of {int : 32}; // S5933: FIFO input
externo chan fifoout of {int : 32}; // S5933: FIFO output
externi chan aptin of {int : 32}; // S5933: PT1 input (1st passthru region)
externo chan aptout of {int : 32}; // S5933: PT1 output
externi chan bptin of {int : 32}; // S5933: PT2 in (2nd passthru region)
externo chan bptout of {int : 32}; // S5933: PT2 out
externo chan inthost of {int : 1}; // S5933: interrupt host
externi chan dpint of {int : 1}; // DP w/ SHARC: interrupt from dual port memory
extern ram:32 DP[65535]; // DP w/ SHARC: dual port memory read/write
externi chan azbtin of {int: 32}; // ZBT: read memory bank A (current address, see text)
externo chan azbtout of {int: 32}; // ZBT: write memory bank A (current address, see text)
externo chan azbtbase of {int: 17}; // ZBT: set base address bank A (see text)
externi chan bzbtin of {int: 32}; // ZBT: read memory bank B (current address, see text)
externo chan bzbtout of {int: 32}; // ZBT: write memory bank B (current address, see text)
externo chan bzbtbase of {int: 17}; // ZBT: set base address bank B (see text)
externi chan afiberin of {int: 9}; // FIBER: data input, channel A
externo chan afiberout of {int: 9}; // FIBER: data output, channel A
externi chan afiberaf of {int: 1}; // FIBER: almost-full flag (TX FIFO), channel A
externi chan afiberff of {int: 1}; // FIBER: full flag (TX FIFO), channel A
externi chan afiberae of {int: 1}; // FIBER: almost-empty flag (RX FIFO), channel A
externi chan afiberef of {int: 1}; // FIBER: empty flag (RX FIFO), channel A
externo int: 1 afiberreset; // FIBER: reset (RX and TX FIFO), channel A
externi chan bfiberin of {int: 9}; // FIBER: data input, channel B
externo chan bfiberout of {int: 9}; // FIBER: data output, channel B
externi chan bfiberaf of {int: 1}; // FIBER: almost-full flag (TX FIFO), channel B
externi chan bfiberff of {int: 1}; // FIBER: full flag (TX FIFO), channel B
externi chan bfiberae of {int: 1}; // FIBER: almost-empty flag (RX FIFO), channel B
externi chan bfiberef of {int: 1}; // FIBER: empty flag (RX FIFO), channel B
externo int: 1 bfiberreset; // FIBER: reset (RX and TX FIFO), channel B
externi int: 1 sharchimem; // SHARC: high address bit active

```

4. Integration with Infrared Focal Plane Array Sensor System

During the effort a remote pod was integrated into a HgCdTe 1024x1024 near infrared (1.0-2.5 μ m) focal plane array control and data acquisition system. This system is used by Cornell University astronomers to operate the focal plane array in a near infrared camera at the Palomar Observatory 200" telescope.

A single remote pod was used to control the entire camera. The remote pod performed the following functions:

- Generation of focal plane array clocking patterns, including various sampling techniques and sub-frame clocking.
- Convert pulse generation for analog to digital converters and data collection from 4 ADC channels.
- Communication link over fiber optic lines (HOTLink) with a host computer. Link was used for uploading commands and downloading data.
- Index control (step generation and limit switch monitoring) of six stepper motors.

The remote pod was physically integrated into the system by development of a "motherboard" to accept the remote pod.

This remainder of this section will discuss in more detail the integration of the remote pod in to the camera controller system.

4.1. Existing System

The remote pod was used to control and infrared camera/spectrograph used for astronomical observation by Cornell astronomers. The pod controls the array clocking and data acquisition, communication link with the host computer (in this case a Sun workstation), and six stepper motors. Each of these subsystems was completely implemented in the Altera 10K50 FPGA on the remote pod, using less than 30% of the FPGAs resources. This section will discuss each of these subsystems in more detail.

The focal plane arrays may be clocked differently for different types of observations. For example, for some observations perhaps only a subset of pixels from the focal plane array is desired. For others, multiple sampling techniques to achieve lower device read noise are required. This places a requirement of flexibility on the array controller system. The system must be capable of providing these various clocking schemes and resulting signal acquisition.

It is also necessary that the controlling system be fundamentally flexible. Infrared focal plane array development is an ongoing research effort at Cornell University.

Array controller systems must be flexible enough to adapt to the changing requirements of various focal plane arrays.

4.1.1. Focal Plane Array Control and Data Acquisition

A 1024x1024 HgCdTe infrared focal plane array, manufactured by Rockwell and named HAWAII, is used in the Cornell infrared instrument (named PHARO, Palomar High Angular ResOlution camera). The instrument is a diffraction limited spatial resolution camera/spectrograph designed to work with an adaptive optics (AO) system on the 200" Mt. Palomar telescope. The camera combined with the new AO system will carry the 200" telescope into the next century and permit leading edge science to continue with the famous 200" telescope.

Infrared focal plane arrays require digital clocking patterns to clock through the individual pixels in the detector array. The HAWAII detector has four parallel analog outputs and requires five digital clocks for proper operation. As the digital clock patterns are presented to the focal plane array, the various pixel values (analog signals) are presented on the outputs. These analog signals are then digitized with 16-bit analog to digital converters at up to 1 MHz, yielding a digital pixel rate of 4 Mpixels/second (four outputs, each digitized at 1 MHz), or 8 MBytes/sec. The controller system must generate the properly timed convert pulses to the analog to digital converters, read the data values from the converters, and process this data stream. In the case of this effort, the processing of the data stream consisted only of relaying the pixel values back to the host computer via the fiber optic link.

The Cornell instrument used analog array electronics from the University of Hawaii. Cornell opted not to use the same digital controller system used by Hawaii due to excessive cost, poor flexibility, and a poor noise performance. The digital system used at Hawaii consisted of multiple boards, each using a dedicated Motorola DSP to generate digital timing patterns, collect data, and communicate to a host computer. The Hawaii digital system was based on a VME architecture, requiring a VME host computer. This system was reported by astronomers at the University of Hawaii to introduce increased noise into their imaging systems. This is not acceptable in astronomical applications, as the detection of the faintest possible objects is paramount.

4.1.2. Communication Link with Host

The remote pod is the sole interface between the PHARO instrument and its host control computer, a Sun workstation in this case. A commercial S-Bus card, using the Cypress Semiconductor HOTLink chipset, was purchased by Cornell University to communicate with the remote pod. The link consists of a single full-duplex fiber optic connection operating at 200 MBits/sec (20 MBytes/sec/direction). All communication between the instrument and host

computer is achieved through this single fiber connection. This includes commands to the instrument, status information from the instrument, and data from the instrument. Six stepper motors and associated home sensors or limit switches are also controlled through this single fiber link. The host computer is located in the control room with the astronomer working directly on this workstation to operate the instrument. The fiber optic cable is snaked through the observatory conduit between the host computer and instrument. The length of this fiber is approximately 100m.

This is in great contrast to previous array controller systems at Cornell. Previous systems have required a PC to be mounted within 20 feet of the instrument on the telescope directly. Copper ribbon cables (two 37 pin cables) were linked between the instrument and this PC. These cables were observed to produce electrical interference with the previous array controller system and were limited to 20' in length due to transmission line characteristic. Data rates over these cables were limited to less than 1 MByte/sec. The astronomer operated the instrument from a workstation in the telescope control room via ethernet communication to the PC on the telescope. Other cables from the PC are required for motor control or other miscellaneous functionality.

During this effort, a simple communication protocol was developed for commanding the remote pod from the host workstation. The host computer would transmit out a command, consisting of 16 bytes of data. The remote pod would echo each byte as it received it. The host computer would then receive these echoes and compare with the sent command. If in agreement, a checksum was then transmitted by the host. The remote pod would compare this checksum with an internally calculated checksum and if equivalent the checksum was echoed back to the host and the command was executed. If the checksums did not agree, the remote pod would discard the errant command.

A number of commands were developed for instrument control. The astronomer can command the acquisition of data in one of several modes and control the integration time (effectively the exposure time) of the specific observation. All stepper motors can be moved in the selected direction a specified number of steps. Home sensors in the instrument could also be monitored to stop the motors at the home position. The remote pod provided all indexer functionality to external stepper motor drivers and monitored the motor home sensors and limit switches when available.

4.1.3. Stepper Motor Control

The PHARO instrument contains six stepper motors to select different optical configurations for astronomical observations. Previous Cornell systems used a separate I/O board in a controlling PC to control motion devices.

These motors were all controlled from the remote pod. Each of the motors was monitored for position either with a home sensor (which is not a physical limit to motion, rather a reference point) or limit switches (which represented the end of

allowable motor travel). The workstation would issue a command to the remote pod to move a particular motor a certain number of steps in a certain direction. The remote pod would then generate the proper digital control signals to the motor drivers. It is possible to move the motors and ignore the home/limit switches, or move the motors and stop when reaching the home/limit switches. In either case, at the end of the motor move, the remote pod would return status information to the host computer.

4.2. Physical Integration of Remote Pod

The remote pod was designed as a stand alone PCB which communicates to a host (if desired) via a fiber link. It was intended that the remote pod would be designed into a system by providing for its footprint (two 100 pin .1"x.1" connectors). For the integration with the PHARO instrument, a separate "motherboard" printed circuit card was developed which accepted the remote pod. The remote pod controlled various subsystems of the array controller (analog to digital converters, array clock generation, and stepper motor controllers). Each of these subsystems had various connector and drive electronic requirements. The motherboard was used to route the I/O pins from the remote pod to the pins of the various connectors for the subsystem. The interfaces to the analog to digital converters and stepper motor controllers were also completely optically isolated to provide a low noise environment. These optical isolators were all placed on this motherboard.

A photograph of the completed array electronics assemble is displayed below. Note the remote pod and motherboard mounted in the lower left section of the electronics box. The pod mates to the motherboard via sockets and can be removed if necessary. However, since it can be reconfigured in-circuit via the fiber link, removal has not been necessary.

In the photo below (Figure 5), four connectors can be seen on the motherboard (white and oblong, D type connectors). These connectors are used to interface with the various subsystems of the array controller. The empty sockets on the motherboard are stuffed with optical isolators are digital buffers to drive the various subsystems. Note that the FPGA pins are NOT used to directly drive cables to other subsystems, but are buffered on the motherboard. The analog to digital converters are located in the upper left section. The digital array clocks are delivered to the signal conditioner board in the lower right section. The stepper motors are controlled through a cable external to this electronics box. The stepper motor cable bundle can be seen under development on the left side of the box. The hardware in the upper right corner is the analog signal preamplifier.

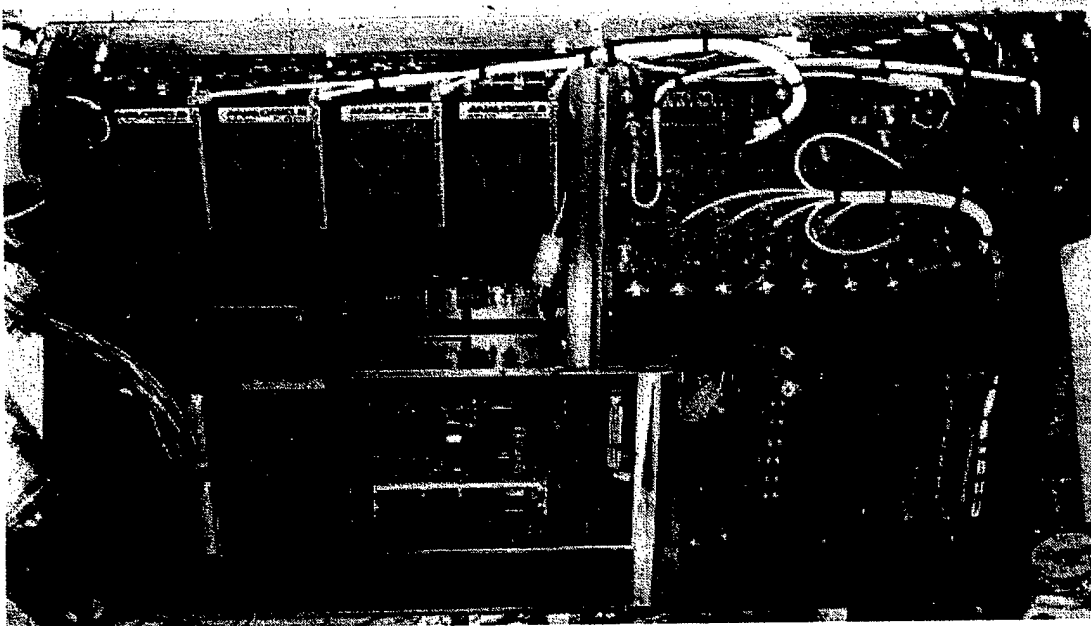


Figure 5. Remote pod and motherboard in PHARO (lower left corner).

4.3. FPGA Coding and System Operation

The Altera 10K50 FPGA on the remote pod provides all control functions for the camera using less than 30% of available resources. The remote pod design was completed entirely in AHDL (Altera HDL) and was not optimized in any fashion.

The design consists primarily of three state machines: PARSER, DETCLOCK, and DATATX. The PARSER machine waits for an incoming command from the host workstation. When it arrives, the command is verified against the checksum. As the command bytes are read by the PARSER machine, they are directly latched into the control registers (i.e. parsed) for the various commands. If the command's checksums match, the command is then executed. For motor moves, the PARSER command pauses until the movement is complete (or home sensor/limit switch has been detected) and then returns to the host computer the instrument status after the movement. If the command is to acquire data, the PARSER machine sets flags in the DETCLOCK machine to acquire data and returns to its idle state.

The DETCLOCK machine actually generates the digital clocking patterns for the focal plane array. Successive states of the machine are cycled through and the digital clocking bits are toggled as required. When a convert pulse is required, the DATATX machine is activated by the DETCLOCK machine. DETCLOCK continues to run *at all times*, always clocking out the focal plane array, even if data is not being acquired. This state machine has parameters passed to it from the astronomer to select various sampling methods and sub-regions within the 1024x1024 pixels as desired.

The DATATX machine reads the data from the analog to digital converters and returns the data over the HOTLink fiber interface to the host workstation. The DATATX machine reads the four analog to digital converters sequentially as individual 16 bit words. These data values are then written one byte at a time over the fiber interface to the host workstation. When complete, it returns to its idle state and awaits another convert pulse from DETCLOCK.

There are a few issues that are critical to point out. Infrared focal plane arrays are very sensitive to small changes in temperature. Thermally generated dark current and other detector properties are affected by even small changes in the clocking pattern. It is therefore critical that the focal plane arrays remain clocked uniformly at all times. This includes when motors are in motion, data is being returned, the telescope is being moved, and other times when the astronomer is not actually collecting data with the focal plane array.

It is critical that the array control system be a reproducible and guaranteed timing system. Use of DSPs or other programmable processors is not a guaranteed reliable method of clock pattern generation unless very careful attention is paid during programming. Small program changes can result in significant timing changes, even when programming directly in low level assembler. Pipeline flushes and other processor issues can wreak havoc on guaranteed timing systems.

On the other hand, FPGAs excel at exactly this requirement. While it is true that the system timing parameters may change with system modifications, as long as the user is not attempting to clock a design at the highest speeds of the devices, timing is a guaranteed parameter. In all work done under this effort with the PHARO integration, the remote pod was operated at the targeted speed of 20 MHz with the slowest speed grade FPGA.

4.4. New Capabilities

The remote pod has brought a revolution to the capabilities of the array controller system at Cornell University. The revolution can be discussed on two significant fronts, pure performance and widespread ease-of-use.

For the first time, it is now possible for all instruments at Cornell to use the same array controller hardware. A wide range of requirements is placed upon the controller due to a range of instrument specialties. Wide band thermal infrared imaging requires raw speed to handle data rates approaching 20 Million/pixels per second (16 bit pixels) whereas detector research and development requires highly flexible controllers to adopt new testing schemes and sampling techniques as they are conceived.

Requiring both high speed performance and high flexibility has not been possible prior to the use of reconfigurable hardware. The high speed specifies a custom hardware solution and flexibility of course demands software based modifications.

The remote pod provides a solution which can be embedded into the array control platform and modified as required as the observation mode demands.

The remote pod offers pixel coadding at 20 Million pixels/second, a factor of two increase over the earlier platform. The remote pod offers a simple, elegant fiber optic interface to the host computer capable of over 20 MByte/sec. The previous system only provided low speed (1 Mbyte/sec) copper ribbon cables, requiring a PC to be mounted within 20' of the instrument. Clocking patterns and complete system operation can be altered in less than one second with the downloading of a new FPGA configuration file using the remote pod. The previous system required eeprom changes to alter array clocking and system operation was completely non-flexible. The previous system was a dedicated hardware controller and it was difficult to integrate new functionality into the system. Integration of external control apparatus, for example motor controls, was not simple.

The single remote pod replaces six 6.5"x4.5" printed circuit cards, a factor of nearly 9 in printed circuit board area, while adding higher performance and an integrated fiber optic link.

5. Platform Evaluation

In this section, we will summarize our findings when developing and working with this platform.

The first general comment to be made is the requirement for digital design expertise in developing with these platforms. During the integration effort with Cornell and the remote pod, it was clear that the scientists did not have the digital design experience to develop FPGA coding internally. Therefore the overall utility of the remote pod to the astronomers would be restricted without the collaborative efforts during this project. While it is clear the hardware provides a new opportunity for them, it is not clear that without the proper high level interface they will be able to develop their applications on the hardware. This refers to relatively simple applications on the remote pod.

We believe AHDL or VHDL design on the base platform presents a challenge even for the experienced digital designer. The platform provides numerous resources, some of them potentially shared. Possible asynchronous clocking boundaries exist (using a synchronous interface with the 5933). Developing "general purpose" template programs for end users to use to begin their design from has proven to be very challenging, as the possible combinations of system utilization are difficult to generalize. Perhaps placing more restrictions on the platform (requiring a AMCC 5933 asynchronous interface for example) would allow easier development of these general purpose templates.

A general comment can be made about the incorporation of leading edge technology. First, it is never available when initially suggested by the manufacturer. Secondly, by the time you get prototypes up and running, next generation components are

appearing on the near horizon. A designer of such a research based developmental system is tempted to incorporate the latest technology, but care must be made in selection of components based on schedule as well as performance. This effort suffered a large schedule slip due to the non-availability of the large 10K250 FPGAs from Altera.

5.1. Remote Pod

The remote pod has been highly successful and shows the utility of reconfigurable hardware for data acquisition and systems control. The in-circuit reconfigurability of a high performance, custom hardware solution has been very well received by astronomers at Cornell.

The architecture of the remote pod is simple. No resources are shared among various processors. The entire design runs synchronous to a single clock (the on-board oscillator or externally supplied clock). The I/O pins can be configured as necessary and routed via a motherboard to external TTL/CMOS interface devices. End users can develop applications on the remote pod very quickly starting from an example AHDL design file. We think the choices that were made for the architecture of the remote pod and its intended applications have proven to be good choices and have resulted in a flexible platform offering unique capabilities.

Some implementation decisions regarding the pod were made that would be modified. One design change that would be desired with the remote pod would be the choice of connector for interfacing with the external system. .1"x.1" low density connectors were used due to their simplicity and immediate availability. At the time of design, a tight time schedule was presented for first light of the Cornell instrument. For signal integrity reasons, the connector pin assignments are in a GND-SIGNAL-SIGNAL-GND configuration, using 1/3 of the pins for GND or power propagation. A quality, high density connector such as that used for the base platform daughter card would be preferred. This connector was investigated during the design cycle for the remote pod but was rejected due to lack of availability at that time.

Another design change would be choice of FPGA package. The 240 RQFP package was selected during the design cycle to assure simplicity of assembly given very tight first light schedule restrictions. However, a redesign of the pod would utilize the BGA package for the 10K50 FPGA. This is to acquire more I/O pins. It is also possible to consider a lower voltage core component, however it is unclear if requiring a mixed +3.3V and +5.0V supply is worth the benefit. An onboard DC-DC converter or regulator could be used to generate this +3.3V core, but then considerations of noise generation (especially with the DC-DC converter) must be pursued.

With the increased I/O pins from the larger BGA package, it would be feasible to consider adding a second independent bank of SRAM for the FPGA. In many data acquisition applications, it is desirable to have two independent banks of SRAM for

data buffering. This is an architectural change, however it was mandated by the lack of available I/O pins on the 240 pin package.

5.2. Base Platform

The base platform provides considerable logic resources for adaptive computing applications coupled with flexible I/O capabilities. The base platform has performed as designed during this effort.

It is difficult to suggest any definite architectural design changes with the base platform without specific consideration of the application objectives. It would be easy to conceive of a preferred shared memory interface between the PCI bus and the host FPGAs (as was discussed in section 2 previously), using for example the 9080 PLX chip and bus switched memories.

5.3. Software Interfaces

5.3.1. ASbridge

As expected, the ASbridge user interface for application development was found to be appropriate and versatile in the verification of hardware functionality. Its programmability requires learning its unique set of keywords and syntax, but they are all very straightforward and easy to pick up by those used to programming in any language. The ability to easily get accurate reports of event timing, such as bus mastering transfers and even latencies due to program execution, is always valuable. The ability for the hardware designer to have interactive access to the developing application has proven extremely beneficial. The hardware designer can modify the host platform interface quickly and easily without the need of a Windows programmer.

5.3.2. Hardware Promela

The trouble with a "C"-like language compiler for reconfigurable hardware is its unpredictable and potentially slow performance. Hardware Promela is no exception. Achieving the goal of putting hardware design into the hands of the typical programmer (rather than hardware design engineer) means that any hardware-related optimizations must be built into the language compiler system itself. The nature of general purpose programming languages, and their explicit sequential operation, make for a complex hardware configuration that is difficult to optimize for speed. The ease of programming in such an environment would be removed if the programmer had to return to hardware design considerations regarding speed of execution. The typical programmer might be used to estimating program cycles for a conventional processor, but in this new environment will not be able to control, or even discover, the particular signal delays that may be preventing acceptable performance.

Just as the "C" language cannot standardize hardware I/O across hardware platforms, neither can a language for reconfigurable hardware. On any particular hardware, a "C" programming environment may expose certain hardware features, such as access to the I/O address space on an Intel platform. Due to the goal of making "C" platform-independent, only general functionality (such as I/O address availability) is provided. The programmer is still left to control hardware according to particular hardware interfaces, such as the DMA controller on an Intel platform.

In porting a compiler system to a reconfigurable hardware platform we may provide any degree of low-level behavioral complexity to the high-level programming environment. A case in point is the "DMA" channels we constructed for Hardware Promela. Use of these channels conforms to the use of all Hardware Promela channels (trivial to use from the Promela side) yet its function is to control high-speed block-data transfer independent of the user Promela code.

The significance of this method of achieving high speed data throughput is similar to that of FPGA "cores". Once the core, or in this case the "channel", is developed and in place, user-developed programs can make use of its complex functionality through a simple interface. FPGA cores are either compute-oriented or hardware I/O specific. An FFT core provides optimized processing, while a PCI core provides hardware interfacing. With Hardware Promela we have been focused on data acquisition and flow. Further application of Hardware Promela, on this and other platforms, will be with full consideration of this potential for easy overall system control and development with Promela combined with optimized use of hardware platform features. We find Hardware Promela, with its built-in "channels", to be a convenient system for this sort of architecture development.

Supplying complex functionality this way requires platform-specific programming in the underlying hardware description language. The choice of functionality can be application-driven as well, and then application development becomes a mix of HDL and the high level language. Once the desired functionality is in place, further development can take place entirely in the high level language accessible to the typical programmer.

Hardware Promela is seen to be unpredictable in performance, regarding achievable speed of execution, and improving this situation while maintaining the completely generalized approach of a C-like language appears challenging. Although some other basic design might improve this runtime performance, it does not seem likely that any design could begin to approach the dependable instruction execution rate of a conventional processor. In the realm of data acquisition, where performance requirements are determined by external applications, speed predictability is a general requirement. Where application speed requirements tend to increase with time, users of Hardware Promela would

have to depend on FPGA hardware improvements to keep up, rather than any ability to further optimize their code. If Hardware Promela is to be made useful for astronomy data acquisition, a means of using it as a process controller – rather than a central processor – must be found. We have a developing sense of how this can be done, but doing so will make each port of Hardware Promela more platform, and perhaps application, specific.

5.4. *The Next Step*

As discussed elsewhere, Hardware Promela holds the promise of putting at least some reconfigurable hardware development into the hands of the typical real-time computer programmer. We will be developing Hardware Promela hardware interface functionality to allow hardware-level performance for certain application-specific functions on various hardware. With experience gained in this, we will be considering whether such a version of Hardware Promela could succeed in giving the Cornell University astronomy scientists developmental control over their own reconfigurable data acquisition system.

RALPH L. KOHLER
AFRL/IFTC
26 ELECTRONIC PKWY
ROME NY 13441-4514

2

ACQUISITION SYSTEMS, LLC
26 LAKE STREET
TRUMANSBURG NY 14886

3

AFRL/IFOIL
TECHNICAL LIBRARY
26 ELECTRONIC PKY
ROME NY 13441-4514

1

2

ATTENTION: DTIC-OCC
DEFENSE TECHNICAL INFO CENTER
8725 JOHN J. KINGMAN ROAD, STE 0944
FT. BELVOIR, VA 22060-6218

1

4

DEFENSE ADVANCED RESEARCH
PROJECTS AGENCY
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

1

7

ATTN: NAN PFRIMMER
IIT RESEARCH INSTITUTE
201 MILL ST.
ROME, NY 13440

1

12

AFIT ACADEMIC LIBRARY
AFIT/LDR, 2950 P-STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765

1

17

ATTN: SMDC IM PL
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

1

24

TECHNICAL LIBRARY D0274(PL-TS)
SPAWARSSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001

1

26

COMMANDER, CODE 4TLOOOD
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

1

27

CDR, US ARMY AVIATION & MISSILE CMD
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

2

31

REPORT LIBRARY
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

1

33

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

1

38

USAF/AIR FORCE RESEARCH LABORATORY
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

1

44

ATTN: EILEEN LADUKE/D460
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

1

45

OUSD(P)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

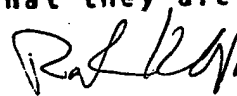
1

46

*Total Number of Copies is:

21

I have verified that this address list is correct and complete
also checked the mailing labels to see that they are correct a
for use.



Signature

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.