The Design and Implementation of a Nanosatellite State-of-Health Monitoring Subsystem

Bryce Daniel Bolton

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Nathaniel J. Davis, IV, Chairman
A. Lynn Abbott
Mark T. Jones
Binoy Ravindran

December 2001
Blacksburg, Virginia

Keywords: Nanosatellite, ION-F, Virginia Tech, FPGA, VHDL.

The Design and Implementation of a Nanosatellite State-of-Health Monitoring Subsystem

Bryce D. Bolton

(ABSTRACT)

This research consists of the design of a low-power, low-cost, nanosatellite computer system solution. The proposed system solution, and design and implementation of a multiple-bus master FPGA and health monitoring space computer subsystem are described.

In the fall of 1998, the US Air Force (USAF) funded Virginia Polytechnic Institute & State University (Virginia Tech), The University of Washington (UW), and Utah State University (USU) with $100,000 each to pursue a formation-flying satellite cluster. The program specified that a cluster of three satellites would maintain radio contact through UHF cross-link communication to report relative positions, obtained through GPS, and coordinate scientific measurement mission activities. This satellite cluster, named Ionospheric Observation Nanosatellite Formation (ION-F) is presently scheduled for launch in June of 2003.

Maintaining some degree of system reliability in the error-prone space environment was desired for this low-cost space program. By utilizing high-reliability components in key system locations, and monitoring less reliable portions of the computer system for faults, an improvement in overall system reliability was achieved. The development of a one-wire health monitoring bus master was performed. A Synchronous Serial Peripheral Interface (SPI) bus master was utilized to extend the communication capabilities of the CPU. In addition, discrete I/O functions and A/D converter interfaces were developed for system health monitoring and the spacecraft Attitude Determination and Control System (ADCS).

Dedication

This work is dedicated to my parents, and my dog, Roadie.

Acknowledgements

I would like to thank my advisor, Nathaniel J. Davis, IV for his guidance. During the HokieSat project, there were many times when requirements were largely undefined or changing and Dr. Davis helped me to focus on the tangible aspects of the project. The many students and professors involved with the HokieSat project made this a meaningful learning experience. Dr. Chris Hall in particular helped to keep the project flexible and interesting.

I would also like to thank my manager, Scott Meller, for his patience with me while I earned a masters degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

NASA and US military experimental satellite systems are tending toward smaller mechanical packages, faster development times, and lower program budgets. The TechSat 21 (Technology Satellite of the 21st Century) program develops expertise in new satellite technology areas including re-configurable and redundant satellite clusters. For example, satellite clusters can be deployed with limited functionality until the entire cluster is in orbit. Also, re-configurable satellite clusters may form virtual lenses or be reusable in various missions [Das00].

Under the TechSat 21 program, several universities have been funded to develop student satellites to explore the formation-flying aspects of upcoming satellite technologies. Nanosatellite technology achieves proof of a technical concept in a light mechanical package (< 10 kg), which will be launched using the Space Shuttle. By funding multiple universities, the USAF evaluates several promising technologies and retains useful information for future satellite missions. With student involvement, space design competence increases over time in new technology areas.

In the fall of 1998, the USAF, AFOSR, DARPA, AFRL, and NASA GSFC funded Virginia Polytechnic Institute & State University (Virginia Tech or VT), The University of Washington (UW), and Utah State University (USU) with $100,000 each to design and build a three-satellite cluster. The universities collaborated by sharing satellite modules to maximize interoperability, design reuse, and to partition the formation-flying design. After the contract award, the student satellite teams named their satellites. Virginia Tech

named its satellite HokieSat. UW and USU named their satellites DawgStar and USUSat, respectively.

The program specified that the cluster of three satellites would maintain radio contact through UHF cross-link communication to track relative satellite positions and coordinate a scientific measurement mission. Using a scientific instrument payload, the cluster of satellites would record an ionospheric charge measurement at discrete satellite positions in orbit. Scientific data-points would be recorded with location information received via on-board GPS receivers. By reconstructing the scientific data from each satellite over multiple orbits, the earth's ionospheric charge would be mapped with respect to position and altitude. The satellite cluster, named *Ionospheric Observation Nanosatellite Formation* (ION-F) is presently scheduled for launch in June of 2003 [Hal01].

Nanosatellite size and weight constraints imposed electronic hardware power and size restrictions. Due to a small volume, the satellite surface area available for solar panel mounting is limited. The satellite deployment system, propulsion system jets, antennas, and electrical communications ports further consumed valuable satellite surface area. The effect of these surface area constraints was a limit on the solar cell surface area available for power generation. Due to the size and weight restrictions imposed on the satellite design, surface-mount components were preferred.

1.2 Initial Research Objectives

At the time of the contract award, in the fall of 1998, a Virginia Tech computer team began to work on HokieSat. At that time, the students were given a high level of autonomy to design a satellite computer system, which would be capable of meeting the needs of HokieSat. The team mission was to develop a computer system, which would accommodate all of the needs of the HokieSat mission, while maintaining protocol and software compatibility with the other two satellites. A custom design would be pursued, which allowed functional compatibility with the UW and USU satellites, while providing a unique computer and control subsystem for HokieSat [SCR01]. Additionally, the team was instructed that a delivery to NASA would occur in the fall of 2001. This schedule

implied that a complete hardware and software implementation would be completed within three years of the original contract award [Hal98].

In order to achieve the low mission cost and short schedule objectives, it was desirable to pursue commercial off the shelf (COTS) technology. However, several disadvantages exist with COTS assemblies. First, they are only partially radiation immune. This limits system reliability compared to more expensive radiation-hardened technologies. COTS technology possibilities could contain far fewer or more components than are desired for the mission at hand, leaving room for power and volume optimization. Because the space environment limits the materials used in electronic design, non-conforming parts used in a COTS design could disqualify a module for space flight. Strict materials and component tractability requirements associated with Space Shuttle missions may also have caused a COTS module to be disqualified. It is possible that a design capable of meeting launch environmental standards would be rejected because NASA documentation requirements could not be met. Furthermore, a COTS printed circuit board (PCB) layout and component population may not consider operation in a convection-free thermal transfer environment, rendering the PCB unreliable in space.

However, it may be possible to work with COTS technology for some subsystems. Readily available COTS technology may be coated or mechanically strengthened for space launch. Non-conforming components may be removed and replaced with space-qualified parts in some cases. An advantage of a COTS approach, for development, is that a manufacturer may provide a proven design in the form of schematics, which can be tailored toward the mission requirements. It may be possible to use a COTS product for early software development, followed by a more space-qualified redesign.

1.3 Early Design Concepts

After the ION-F contract award, a group of Virginia Tech Aerospace students drafted a conceptual specification for HokieSat. The specification stated that HokieSat would use a 2-satellite tethered system as depicted in Figure 1. The proposed tether-stabilized satellite system would use a gravity-gradient to maintain stabilization. Gravity gradient

stabilization means that a heavier satellite module assumes an orbit closer to earth compared to a lighter satellite module at the end of a connecting tether.

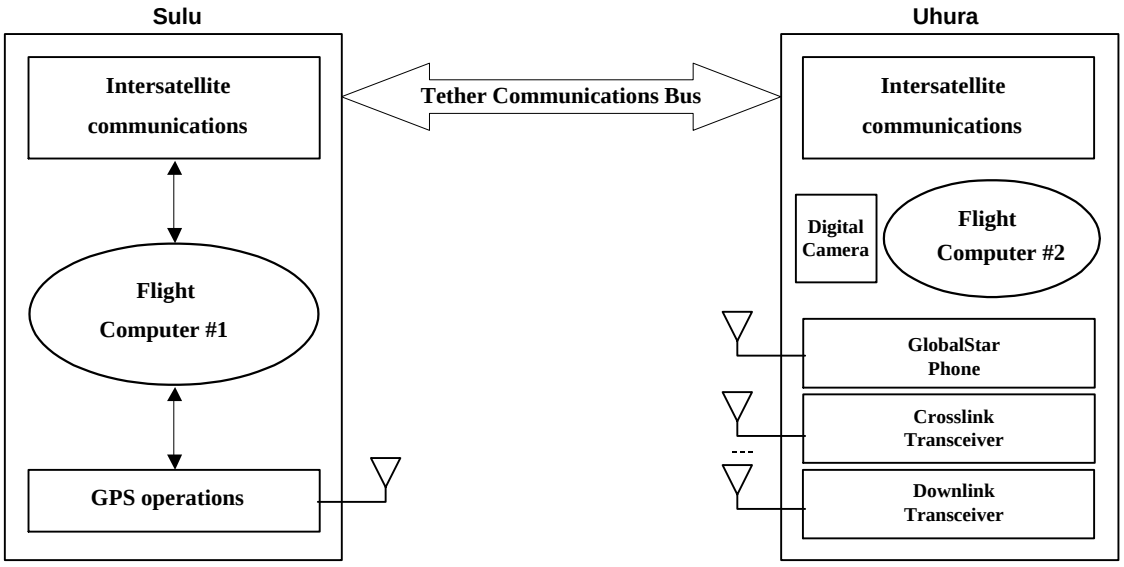


Figure 1. Original HokieSat 2-satellite system proposing a tethered connection.

This original tethered satellite configuration proposed two main subassemblies. A navigation module, named Sulu, would contain GPS and computer resources. A communication subassembly, named Uhura, would contain uplink/downlink, a GlobalStar Phone, a UHF cross-link to the other ION-F satellites, and computing resources. A tether containing a communications bus would join the satellites, and their computer resources. The Virginia Tech Aerospace students abandoned this design and pursued a single HokieSat chassis, after reconsideration of possible complications with the tether deployment mechanism [Hal98].

1.4 HokieSat CPU Selection

In the Spring Semester of 1999, the Virginia Tech computer team researched a number of low-cost, low-power processors including Motorola's M-Core series processors and Advanced RISC Machines (ARM) 32-bit microcontroller implementations. By the end of that term, the VT design team decided to pursue a low-power ARM-based microprocessor design. An ARM processor implementation, the Sharp LH77790B, was available in a $150 Arm Evaluation Board (AEB). The low cost of the AEB development

system allowed each student to borrow a development system from the team, or purchase his/her own if desired. The AEB development environment was attractive also because of a parallel-port step-by-step instruction debug interface, a low-cost simulator and compiler, an available a JTAG debug port, and a list serve support group [ABB99].

The Sharp LH77790B 32-bit microcontroller provided many attractive features for the HokieSat project. First, it was a low-power integrated circuit, capable of 3.3V operation. It was also classified as a System On a Chip (SOC) or microcontroller due to the large number of peripherals built into a single IC. Retaining as many features as possible inside the CPU meant fewer additional components would be required on the PCB-level design. This SOC approach had strong power, size, reliability, and cost savings.

Some attractive features of the Sharp LH77790B processor included the availability of three (3) UARTs, a watchdog timer, 24 bits of discrete I/O, an IrDA interface, a PWM controller, on-chip power management, a DRAM controller, and a built-in JTAG interface. The 512 MB address space, low interrupt latency, 16.7 MHz operation, floating-point multiplication capabilities, and 2kB 4-way set associative cache would provide a starting point for meeting the HokieSat computer needs [Sha98]. At the time, it was envisioned that a 16 to 32MB telemetry buffer would be utilized for offloading data from the satellite with an external telemetry FPGA. As additional requirements became available, the system analog Input/Output (I/O) capabilities, digital I/O capabilities, and communications capabilities could be increased to complement the Sharp LH77790B.

1.5 HokieSat Operating System Selection

Having selected a CPU, the HokieSat computer team investigated operating system (OS) possibilities. Because initial research showed that commercial operating systems such as VxWorks could cost as much as $20,000 per license, low-cost operating systems were investigated. Maintaining compatibility between the low cost OS and selected CPU development platform was desired to minimize software development efforts.

The team evaluated MicroC/OS-II with the AEB first. USU had previously used MicroC/OS-II and was considering it for use in their satellite. MicroC/OS-II software was also appealing because the OS source code was freely available, and applications could be developed and debugged on an Intel x86 platform [Lab99].

Another embedded design group had designed a Hardware Abstraction Layer (HAL) of MicroC/OS-II for the AEB. However, due to intellectual property protection, the implementation could not be released. All other MicroC/OS-II HALs for the AEB and similar ARM CPUs were downloaded and compiled by the HokieSat computer team, but the OS did not successfully operate. Although the instruction manual for MicroC/OS-II describes the necessary steps required for a port to any processor, the student development team was unable to achieve success with MicroC/OS-II, given the extremely tight constraints placed on the project. A lack of HokieSat funding for full-time Electrical Engineering graduate students also limited the team's ability to focus on a problem such as the operating system port for a long period of time.

Instead of continuing toward a MicroC/OS-II operating system, the team focused its efforts on another low cost operating system called Embedded Configurable Operating System (eCos) by Red Hat (formerly Cygnus). This OS also had a HAL for the AEB and used freeware tools under the Linux operating system. If required, Red Hat support could be purchased for a fee. After downloading eCos and the AEB HAL, the HokieSat computer team demonstrated a working build of the eCos OS on the AEB platform in a single evening. eCos and the AEB appeared to be a promising solution for HokieSat and possibly ION-F [Red99].

The advantages of the AEB/eCos solution proposed by Virginia Tech were overwhelming, given the cost and schedule requirements of HokieSat. Using low-cost AEB development boards, each student could work on portions of the software design long before the flight hardware was available and debugged. Allowing hardware and software development to proceed concurrently would be a major advantage for achieving the satellite launch date. Using the AEB, hardware modules could be also developed and

prototyped through the AEB's PCB headers. With a team of students working with the same type of hardware and software, code and development expertise could be developed within the team. Because the low-cost operating system imposed no license quantity or PC node-locking restrictions, many students could be involved with the effort from any location.

Unfortunately, the hardware and operating system requirements for ION-F changed after Virginia Tech's computer team decided upon the ARM and eCos operating system solution. In late 1999, NASA directed that the three ION-F universities use the VxWorks operating system, by Wind River. NASA provided additional funding so that this operating system could be purchased with one year of support per university. Prior to NASA's directive, VxWorks had been considered by the computer team but dismissed because of its high purchase price. VxWorks was a preferred NASA operating system for military and space systems because of its proven performance and reliability. Because VxWorks was prevalent in space applications, code developed for ION-F would be more portable, or at least understandable, for future missions also using VxWorks. This directive jeopardized the HokieSat CPU selection, because VxWorks had been ported to the Sharp ARM microcontroller, but with limited tool capabilities. Another low-power CPU with a suitable VxWorks Board Support Package (BSP) would need to be selected if the port to the Sharp LH77709B/AEB did not work.

Despite the disadvantages of changing the requirements to VxWorks over one year after the contract award, project leaders within all three universities agreed that utilizing VxWorks was in the best interest of the ION-F program. The availability of superior debug tools and widespread of use of VxWorks within the military and space community contributed to that decision. Some students would also benefit from having VxWorks experience when seeking employment.

Wind River was contacted to determine if VxWorks and the AEB were compatible. Although VxWorks would operate on the AEB, the toolset required a high-speed RS-232 connection. Because the Sharp LH77790B had an un-buffered UART, the toolset would

operate at a very slow speed. Unfortunately, creating a modified LH77790B solution with a 16550 8-byte FIFO buffered UART or similar interface was not considered feasible by the ION-F team. Instead, a fully custom CPU board would be developed by USU for all three universities around a new processor, the Hitachi SH7709.

A monetary loss occurred to Virginia Tech because of the change in development philosophy from individual universities pursuing computer designs to an ION-F shared computer design. At the time VxWorks was chosen, over $5000 in funding was expended on Sharp LH77790B tools including an ARM JTAG In Circuit Emulator (ICE), ARM development software, and AEBs. Software support for VxWorks would also cost each university $2500 per year.

In addition, the ability to concurrently develop hardware and software was lessened under the VxWorks operating system because low-cost development boards that supported VxWorks were not available. An obsolete $3500 development board was located for the purpose of software development with the Hitachi SH7709. Unfortunately, a JTAG interface for low-level debugging was not available for the Hitachi SH7709 and a three-day delay occurred when obtaining technical support from Japan. However, VxWorks was functional using this development platform, after UW received a considerable amount of on-site support from Wind River.

To partition the ION-F computer design, the team decided to pursue a modular computer system to lower risk and share development efforts. Under this plan, Utah State University would develop the ION-F backplane, ION-F CPU Board, ION-F Telemetry Board, ION-F Camera Board, and ION-F Science Board. Virginia Tech would develop the ION-F Input/Output Board (I/O board), the HokieSat Power Board, and HokieSat Torque Coil Board. At the time this decision was made, morale in the Virginia Tech computer team dwindled and many team members left the project. After a year of attempting to pursue a design to meet the original HokieSat cost and deadline constraints, the role of the Virginia Tech computer team was reduced to a support function. It was

clear that USU would be developing the computer system and that Virginia Tech and UW would contribute to other shared aspects of the ION-F design.

The decision to shift away from custom computer designs at each university and toward a common computer design was influenced by several factors. First was USU's close proximity and relationship with Space Dynamics Laboratory (SDL). USU and SDL have delivered the most number of student satellites in the past of any United States university. Thus, USU had an experience base in satellite design that the other universities did not. SDL personnel, clean rooms, and surface-mount assembly labs could be utilized in the design and fabrication of the satellite computer. A second reason that USU was favored for computer system development was due to additional funding for the Plasma Impedance Probe (PIP) science project, which allowed full-time Electrical Engineering graduate students to be hired to focus on the hardware and software development of the ION-F computer system.

As a result of the partitioning of the ION-F computer design and NASA's VxWorks operating system directive, the structure of the ION-F development effort changed. Some Virginia Tech computer team members remained on the project through the Spring Semester of 2000 in order to work on the Hitachi development boards with VxWorks. However, I was the only remaining hardware designer left on the Virginia Tech computer team, and I was tasked with the development of the ION-F I/O board.

1.6 Summary

The focus of this thesis is to describe the development of the ION-F I/O board and its function in the modular ION-F computer system. The I/O board will fly on each of the three ION-F satellites to serve as a health monitoring subsystem, and to extend backplane communications via an SPI bus. The ION-F computer architecture and preventive measures against radiation effects in the I/O board design will be discussed in Chapter 2[1].

[1] Although this research discusses monitoring and latch-up detection methods used to increase ION-F reliability in the space environment, it is not the purpose of this research to focus on space radiation effects.

The development of the I/O board FPGA entities will be discussed in Chapter 3. Software used for accessing the I/O board resources will be discussed in Chapter 4. Conclusions of the project will be in Chapter 5, followed by a summary of future work.

Chapter 2

ION-F Computer Design and Health Monitoring

2.1 Introduction

Several factors influenced the ION-F team to pursue a common computer system and enclosure. First, a similar design had been flown by USU on board a sounding rocket. That a similar design had survived in a harsh environment was reason to pursue a similar design methodology. USU also had close ties with the SDL, a major supplier of space optics systems with years of experience in space hardware design and development. The resources of Space Dynamics Laboratory (SDL) would be valuable in creating a successful computer for ION-F. Another benefit of the modular design was design reuse. Future programs could adopt the ION-F backplane and computer system to their needs by designing several new PCBs instead of redesigning an entire computer system.

An overview of the ION-F backplane, shown in Figure 2, follows. Three rate gyros are used for feedback control in the Attitude Determination and Control System (ADCS). The gyro board provides an interface to the rate gyros using discrete I/O from the CPU board. The CPU board is the primary satellite computer resource, which oversees all satellite Command and Data Handling (C&DH) activities. The telemetry board is used to buffer CPU-framed data for offload during a downlink session. The camera board supports up to four Fuga 15d cameras for attitude control and determination. The I/O board provides A/D conversion, discrete I/O, one-wire bus support, and Synchronous Serial Peripheral Interface (SPI) bus support for on-board health monitoring. The Plasma Impedance Probe (PIP) performs AC and DC-coupled measurements on ionospheric

charge levels for all three ION-F satellites. The Torque Coil Board allows HokieSat to perform attitude adjustments in orbit. By adjusting the satellite flight angle, the satellite drag may be varied to maintain or decrease the satellite orbit. A spare slot is available for future expansion and to accommodate the needs of other universities while maintaining a common backplane. Two power boards are located at the end of the computer enclosure to allow maximum conductive heat loss through the aluminum chassis. The I/O board, torque coil board, and power boards, were designed by Virginia Tech. All other computer system components were designed by USU and UW.

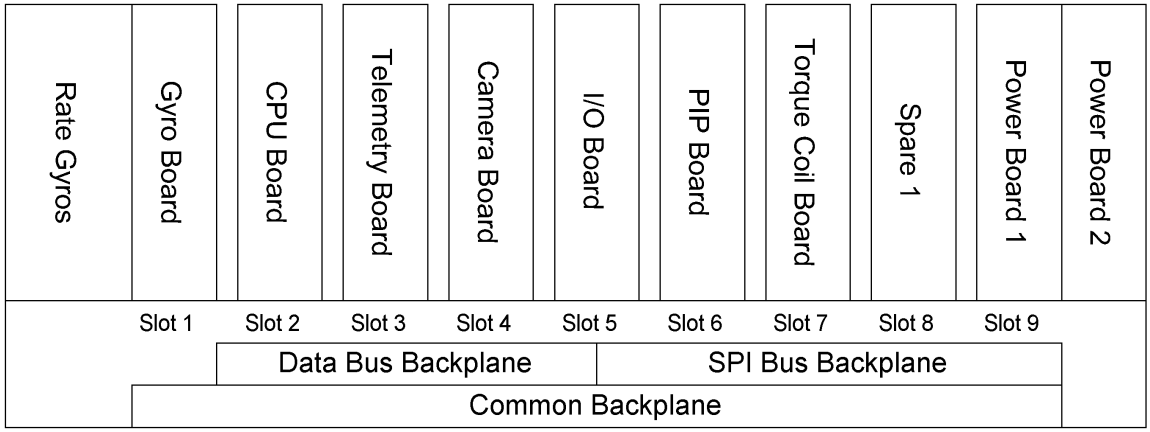


Figure 2. HokieSat Backplane Configuration.

Once the ION-F universities agreed on a common backplane approach in the Spring Semester of 2000, the computer team members met to define the details of the system. High-speed parallel communications would be available between the CPU board, telemetry board, camera board, and I/O board through an extension of the CPUs data bus. In order to simplify additional card designs, the I/O board provides an SPI bus master to communicate between Slot 5 and Slots 6, 7, 8 and 9. The backplane and the cards in slots 1 through 5 are common to all three ION-F universities. The remaining slots are flexible enough to be configured according to the needs of each university. In addition to the I/O board design, I helped define a latchup and health monitoring mechanism for the common computer design, helped define details of the backplane signals, and helped define the board-level signal interfaces in design reviews and one-on-one meetings with USU representatives.

2.2 Health Monitoring in the ION-F Computer System

Digital outputs from the CPU control power to each backplane PCB, except the power boards. Thus, the CPU board may control the system power level by switching on or off any PCB in the backplane. In the powered-down state, the boards draw the least amount of current, and are the least susceptible to radiation effects. By carefully planning which PCBs are required for various mission operations, power consumption and radiation susceptibility can be minimized.

Table 1. I/O Board Function Overview.

Function	Range	Number of Channels or Devices
16-Bit ADC	0-5V	16
12-Bit ADC	0-5V, 0-10V or +/-10V	16
One-wire bus	5V Digital I/F	100 Slaves
SPI bus	3.3V Logic I/F	32 Slaves
Digital Inputs	0, 5V TTL	8
Digital Outputs	0, 5V TTL	8

The I/O board provides a number of capabilities for health monitoring in the ION-F computer system. An overview of the I/O board functions appears in Table 1 and a description of each of the I/O board resources follows. The 100 kHz analog to digital converter (ADC) channels may be used for fast and reliable monitoring of a particular parameter on board the satellite. Digital I/O provides a capability for digital control and monitoring. The one-wire bus may be used for health monitoring, but data conversions may take as long as 750ms, depending on the type of sensor attached to the bus. Depending on the number of sensors on the bus, it may be necessary to group the sensors into high and low priority levels to achieve a desired response time. A benefit of the one-wire bus is the ability to string up to 100 sensors around the satellite for temperature profiling. The USAF requested this feature, because the ION-F satellites may be destroyed using a ground-based laser system at the end of their mission. The ability to profile the temperature is useful to improve satellite destruction methods. Unfortunately, due to sensor conversion time limitations, the one-wire bus will not be able to gather a significant amount of data unless the destruction process is very gradual.

In the ION-F system, the SPI bus serves two purposes. First, it is used for attitude control on board USUSat to control stepper motors, which move magnetic booms. In all three ION-F satellites, the SPI bus is used for access to the Plasma Impedance Probe (PIP) for scientific data gathering. The SPI bus is also used as a satellite wide health monitoring mechanism. A high-reliability Actel A42MX16 series FPGA resides on the Power Board of each satellite. Because most of the satellite electronics are not radiation hardened, it is necessary to add this reliable watchdog-timer circuit. The non radiation-hardened CPU periodically writes to the Power PCB FPGA using the SPI bus. Should the CPU fail to write to the Power Board FPGA within a certain time interval, the entire satellite power will be cycled to attempt to recover from a severe radiation upset. The Power FPGA always remains on when the satellite has battery or auxiliary power. The Power FPGA also contains a timer, which may be read to determine the total elapsed time since the system was powered on. The total number of resets, which have occurred, is also tracked using the Power FPGA. By spawning a watchdog task, the flight software may periodically update the Power FPGA watchdog timer in a relatively unobtrusive manner.

2.3 Radiation Effect Mitigation in the ION-F Computer System

In order to reduce costs in the ION-F computer system, a large number of radiation-susceptable CMOS components are being used. However, flip-flops in CMOS components may experience a single-event-upset (SEU). The SEU causes a digital flip-flop to toggle unexpectedly from a ‘0’ to ‘1’ or visa versa. An early analysis of ION-F space environment predicted that each satellite would receive up to 3 SEUs per day. In order to provide some measure of protection against radiation effects such as the SEU, high reliability Actel anti-fuse one-time-programmable FPGAs were selected for use in all of the custom circuit boards as a first defense against the SEU. The Actel A42MX16 FPGA was selected for use in the I/O board [Jen00]. The anti-fuse FPGA is a one-time-programmable (OTP) part, which burns fuses to set a configuration. The anti-fuse technology is not susceptible to reconfiguration errors, which are possible with other FPGAs that use RAM cells to hold the logic configuration [Act98a].

Actel recommends several approaches to increase design reliability using A42MX16 FPGAs. First, designs should use CC-FFs, or "Combinatorial Cell" flip-flops instead of S-FFs, or "Sequential Cell" Flip-Flops. CC-FFs are less susceptible to radiation effects than S-FFs. Triple Module Redundancy (TMR) may also be used to produce very reliable designs in a radiation environment. As long as enough room exists in the target FPGA, TMR can be applied to any Actel FPGA by using the appropriate synthesis options. An example of a TMR module appears in Figure 3 [Act98b].

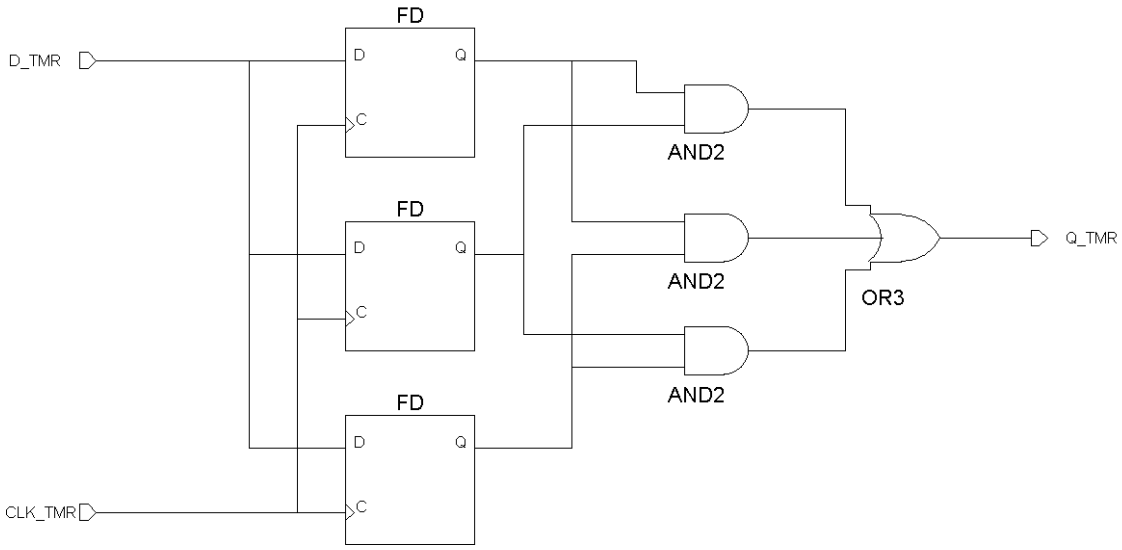


Figure 3. Triple Module Redundancy Example.

2.4. Summary

A number of preventive and active techniques have been employed in the ION-F computer system. Additional methods for mitigating radiation effects in the ION-F computer system include board-level current monitoring by the CPU board, on/off control of each board, and TMR in the main CPU memory [Jen00]. These techniques, developed during multiple e-mail and verbal discussions between ION-F computer team members and myself, have been employed to balance the use of CMOS technology with more expensive anti-fuse technology.

Chapter 3

Detailed Design of the I/O Board

3.1 Introduction

The I/O board performs health monitoring functions, attitude determination and control, science instrument control and data acquisition, temperature monitoring, and current monitoring on board all three ION-F satellites. As the I/O board designer, I was responsible for meeting the ION-F Computer Requirements Specification for the I/O board. An FPGA design and board-level architecture and component selection comprised the hardware design. In order to simplify interfacing to the I/O board, test benches were written with VHDL procedures to closely mimic the CPU interaction with the I/O board. Because there is no standard schematic and board-level tool available, the I/O board PCB layout and fabrication were performed at Space Dynamics Lab and coordinated by USU. This allows USU or SDL to maintain and enhance the design for future space missions.

As defined by the ION-F Computer Requirements Specification, the I/O board consists of several analog and digital ICs and a custom FPGA. The FPGA controls access to board-level resources as well as finite state machines (FSMs) inside the FPGA. The I/O board top connector interface provides sixteen 12-bit A/D channels, sixteen 16-bit A/D channels, 8 digital outputs, 8 digital inputs, and a 1-wire bus interface. The I/O board bottom connector provides an SPI bus interface, board-level power on/off control, current monitoring, and interrupt services. Figure 4 shows the major features of the I/O board and their interconnections.

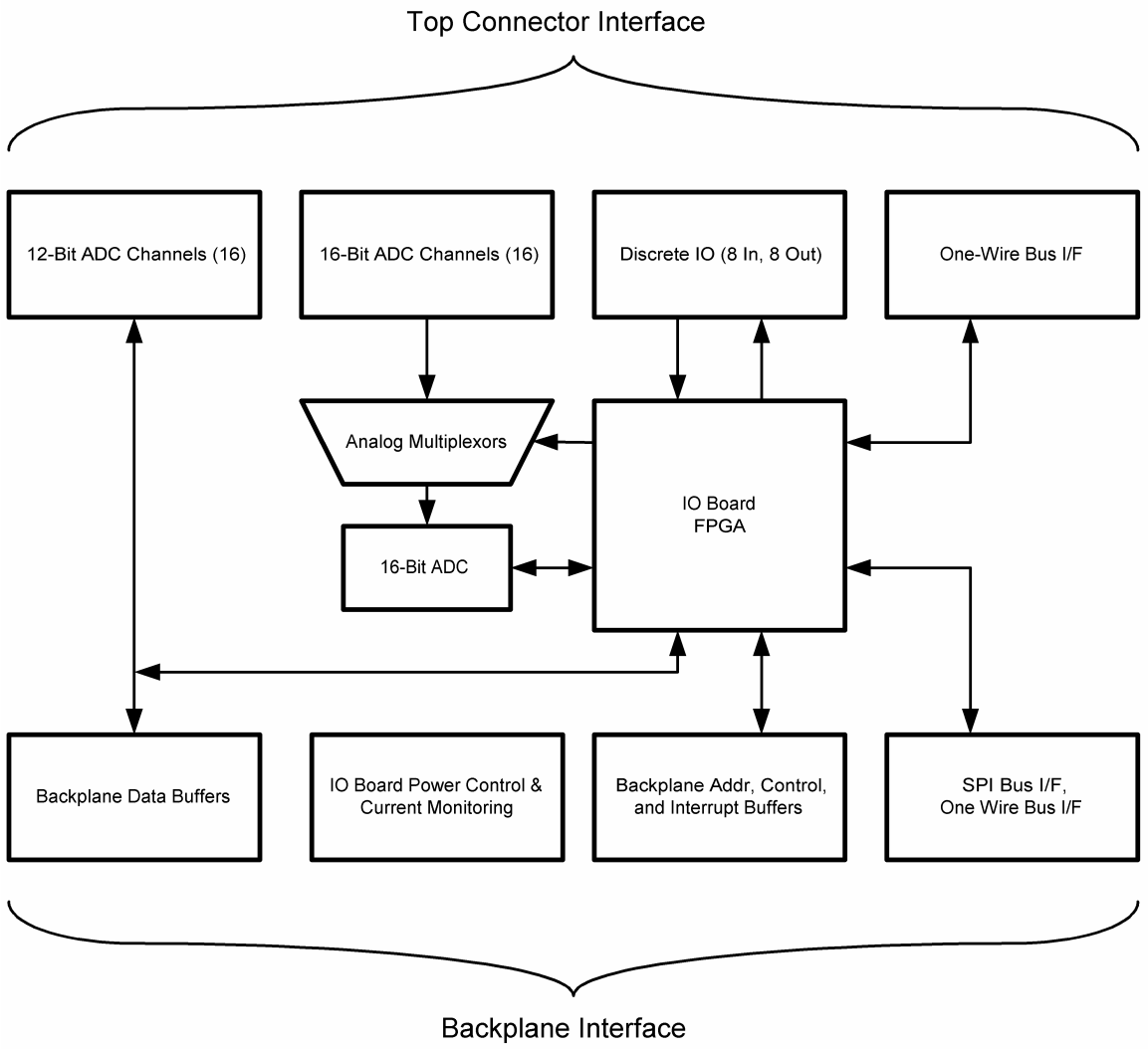


Figure 4. I/O Board Block Diagram.

3.2 The I/O Board Backplane Interface

The I/O board resides in the ION-F Common Electronics Enclosure (CEE) with parallel address/data bus access from the backplane. The CPU initiates backplane read and write operations to target either the I/O board FPGA resources or the MAX197 A/D converters using address decode logic in an FPGA. By leaving the backplane buffers in a hi-impedance state, the entire I/O board may be powered off through its power control circuitry.

The CEE specification restricts the CPU address/data bus to the first 4 PCB slots in order to minimize transmission line effects, spacecraft weight, and system complexity. Slots 5 through 9 may be accessed using discrete CPU I/O or the I/O board SPI bus. By using an SPI bus protocol, PCBs may be developed with a simple serial digital interface instead of a more complex parallel interface. Higher cost, power, and buffer chip counts are associated with a parallel interface.

I/O board interrupts inform the CPU of the completion of successful or unsuccessful I/O board data transfer sequences. Upon receipt of an I/O board interrupt, the CPU reads the I/O board STATUS and ERROR registers to service the event or log an error. I/O board power and ground connections are also accommodated through the backplane interface. The entire I/O board backplane connector signal definition is included in Appendix B.

3.3 The SPI Bus Master

A Synchronous Serial Peripheral Interface (SPI) Bus master serves as a simple low-bandwidth interface for scientific data gathering, attitude control, and power board health monitoring. This interface extends backplane communications between slots four (4) and nine (9). The SPI bus master follows the Motorola Synchronous Serial Peripheral Interface standard with clock polarity set for rising-edge gating (CPOL = '0', CPHA = '0') as shown in Figure 5 [Mot91].

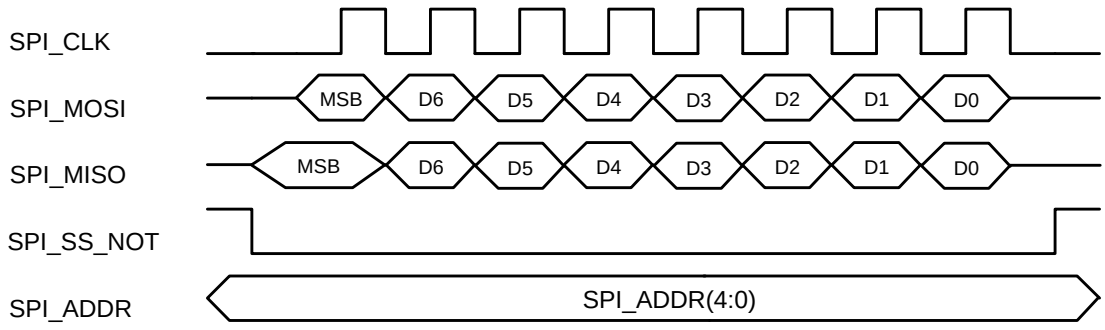


Figure 5. A typical bi-directional I/O Board SPI Bus transfer.

The I/O board SPI bus implementation supports up to 32 slave devices. An SPI slave decoding unit uses five (5) address bits and a SPI_SS_NOT signal. A local SPI_SS_NOT signal is generated for a particular SPI bus slave device when SPI_ADDR(4:0) matches the local address and SPI_SS_NOT is low (0V) on the backplane. This condition enables the SPI bus slave open-drain MISO line driver output on the shared MISO data line. ION-F SPI devices may use one or more SPI bus addresses, depending on the register definition implemented by slave board designers. All SPI bus signals use +3.3V logic, following the ION-F backplane convention. A summary of the SPI bus signals and their usage is defined in Table 2.

Table 2. SPI Bus Signal Names and Directions.

Motorola SPI Bus Name	*ION-F Backplane Signal*	*TX Device*	*RX Device*
SCK	SPI_CLK	I/O Board	Slave
MOSI	SPI_MOSI	I/O Board	Slave
MISO	SPI_MISO	Slaves*	I/O Board
SS#	SPI_SS_NOT	I/O Board	Slave
--	SPI_A4	I/O Board	Slave
--	SPI_A3	I/O Board	Slave
--	SPI_A2	I/O Board	Slave
--	SPI_A1	I/O Board	Slave
--	SPI_A0	I/O Board	Slave

* The slaves transmit using open-drain buffers for a wired-AND bus.

The ION-F SPI bus is a serial data with parallel address protocol. An address word is specified on SPI_A(4:0) by writing the slave address to the SPI_ADDR_REG(4:0). Other SPI bus slaves will not participate in a transfer until their SPI bus address is active. The SPI_SCALE_REG allows the user to specify the rate of the SPI_CLK from 625 kHz to 19 Hz according to the formula f_{SPI} = 20MHz/2^(SPI_SCALE_REG+5). The possible

SPI bus clock rates are shown in Table 3. It is necessary to scale the SPI bus clock rate because the Power Board FPGA contains a slow oscillator to reduce power consumption. A programmable SPI bus clock allows most transfers to proceed at high speeds, while accommodating slower devices.

Table 3. Possible SPI Bus clock Rates according to allowed scaling.

SPI_SCALE_REG	f_{SPI}
0	625 kHz
1	313 kHz
2	156 kHz
3	78.1 kHz
4	39.1 kHz
5	19.5 kHz
6	9.77 kHz
7	4.88 kHz
8	2.44 kHz
9	1.22 kHz
10	610 Hz
11	305 Hz
12	153 Hz
13	76 Hz
14	38 Hz
15	19 Hz

* The slaves transmit using open-drain buffers for a wired-AND bus.

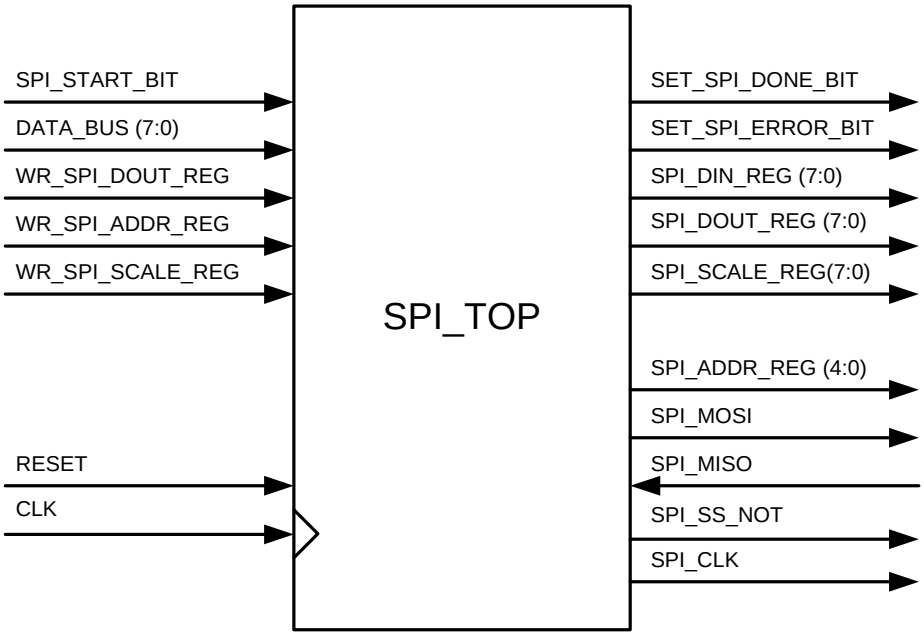


Figure 6. Top-level entity of the SPI Bus master.

The SPI_TOP entity, shown in Figure 6, receives register write strobes from the ADDRESS_DECODER module during CPU write operations. The output of each register is also available for read back verification through the READ_MULTIPLEXOR. The ability to read back each register is critical for read-modify-write and verification operations. A typical SPI bus master sequence follows. First, the CPU writes to the SPI_SCALE_REG through the DATA_BUS to configure the SPI_CLK rate for the upcoming transfer. Next, the SPI_ADDR_REG is written to specify which slave register will be accessed. The five (5) LSBs of the SPI_ADDR_REG are buffered and output directly to the backplane for receipt by SPI Slave devices. Prior to an SPI bus write operation, the SPI_DOUT_REG is written with the desired outgoing data byte. A write to the I/O board CONTROL_REG bit 0 initiates the SPI bus transfer. CONTROL_REG bit 0 sets the SPI_START_BIT, which begins the transfer. According to the SPI bus protocol, the SPI_SS_NOT signal is asserted low during the entire transfer. One byte of data is clocked out from the SPI_DOUT_REG to the Master-Out-Slave-In (SPI_MOSI) output. Simultaneously, one byte of data is clocked into the SPI_DIN_REG via the Master-In-Slave-Out SPI_MISO input. Upon completion of the data clocking, the SPI_SS_NOT output is released. The SET_SPI_DONE_BIT is asserted for one clock

cycle in order to generate an interrupt condition for the CPU and set bit 0 in the STATUS_REG. For SPI bus Read operations, the SPI_DIN_REG contains SPI bus data received from the Slave after receipt of the I/O board interrupt. For SPI bus Write operations, the operation is complete. The SPI bus interrupt condition may be cleared by writing to the STATUS_REG and additional SPI bus operations may be performed.

3.4 The Top Connector Interface

The I/O board top connector provides health monitoring functions and controls through analog sampling, digital I/O and a one-wire bus interface. Custom cables allow each ION-F satellite to be configured according to the specific needs of each university. The complete top connector pin-out appears in Appendix D.

3.5 The Discrete Output Interface

A byte-wide discrete output register is available for controlling switching functions via the I/O board top connector. This interface provides a +5V memory-mapped output register, which is buffered with inverters on the board level.

The CPU writes to the Discrete Output Register (DISC_OUT_REG) by de-referencing a pointer to the register address. The DISC_OUT_REG may be read back through the READ_MULTIPLEXOR for verification and read-modify-write purposes. The Discrete Output Entity is shown in Figure 7.

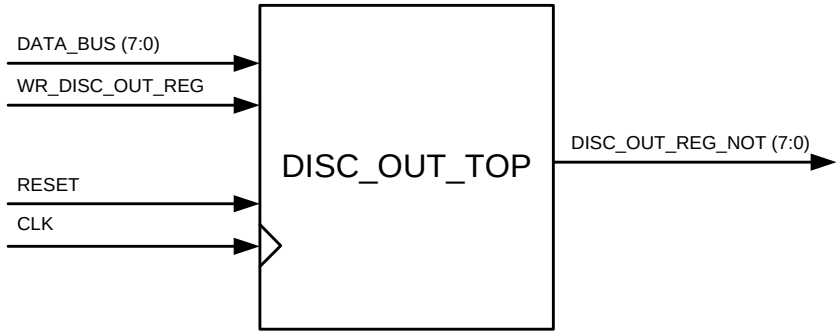


Figure 7. Top-level entity of the Discrete Output Register.

3.6 The Discrete Input Interface

A byte-wide discrete input register is available for monitoring +5V TTL or CMOS logic levels via the I/O board top connector. This interface provides a memory-mapped read-only register, which is buffered on the board level.

The CPU reads the Discrete Input Register (DISC_IN_REG) by de-referencing a pointer to the correct address. The DISC_IN_REG is read through the Register Read Multiplexor in a manner similar to the other I/O board registers. Power consumption of the DISC_IN_REG is a combination of current drain of its associated sequential and combinational units. Because the DISC_INPUTS signal is asynchronous, it is necessary to register those inputs to ensure a stable CPU read operation. The Discrete Input Entity is shown in Figure 8.

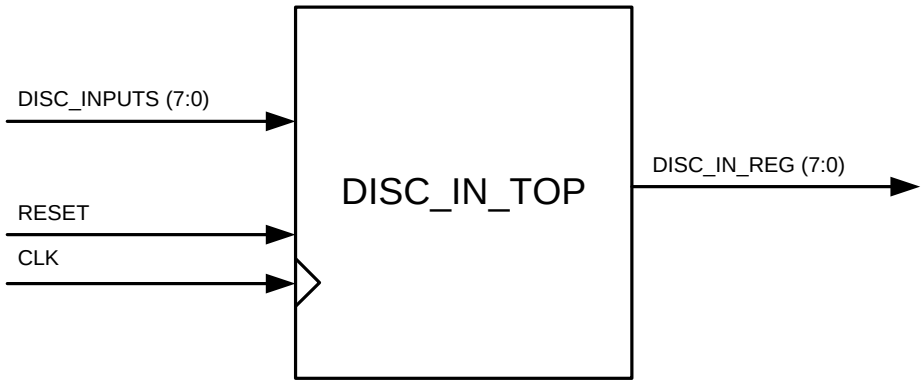


Figure 8. Top-level entity of the Discrete Input Register.

3.7 The One-Wire Bus Master

A one-wire bus master serves as a flexible low-bandwidth asynchronous serial interface for temperature and voltage monitoring as well as power regulation and satellite health monitoring. This interface extends along the CEE backplane so that PCB temperatures may be monitored. On the top connector of the I/O board, the one-wire bus interface is available to provide monitoring of HokieSat subsystems.

Use of the one-wire bus varies according to each ION-F satellite configuration. The University of Washington plans to employ as many as 100 one-wire bus devices for

satellite state-of-health monitoring. A benefit of the one-wire bus architecture is its high degree of configuration flexibility. However, it is also less robust than a dedicated A/D line from the I/O board top connector; a single-point failure could disable the entire one-wire bus.

The one-wire bus follows an entirely asynchronous serial protocol. However, timing parameters on the bus are well defined. All serial operations may be categorized into four types: Reset, Write Slot 0, Write Slot 1, or Read slot.

A one-wire bus Reset operation precedes all other operations and has the timing characteristics shown in Figure 9. The master asserts a reset pulse for a minimum of 480 us, then releases the bus for 15 to 60 us. If present, all slave devices assert a Presence Pulse by pulling the data line low for 60 to 240 us. If no slaves are present, the one-wire bus data line will remain high after the reset pulse is asserted. At a minimum, a 1 us recovery period elapses after the Presence Pulse before another operation is issued.

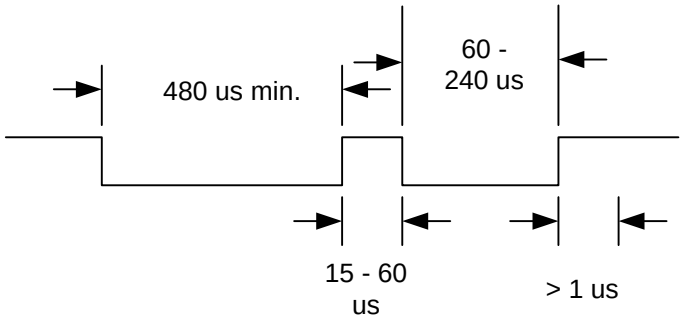


Figure 9. One-wire bus Reset Timing.

When sending data to a slave device, the one-wire bus master issues Write One Slots and Write Zero Slots. The Write Zero Slot is shown in Figure 10. The bus master asserts a low pulse on the data line for 60 to 120 us, then releases the bus for a minimum of 1 us. The slave device typically samples the data line 30 us after the falling edge.

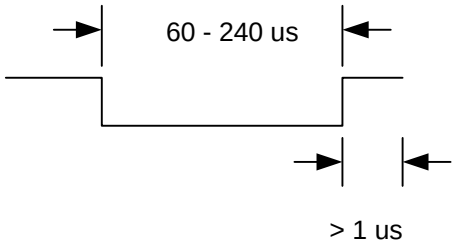


Figure 10. One-wire bus Write Zero Slot Timing.

The Write One Slot is shown in Figure 11. The bus master asserts a low pulse on the data line for 1 to 2 us, then releases the bus for a minimum of 1 us. The slave device typically samples the data line 30 us after the falling edge.

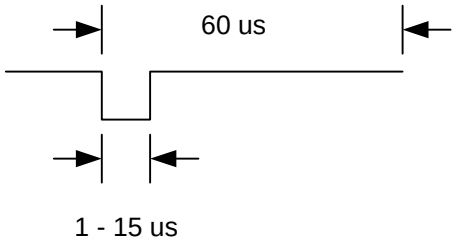


Figure 11. One-wire bus Write Zero Slot Timing.

A typical one-wire bus Read One Slot is shown in Figure 12. The bus master requests a bit of data by asserting a low pulse on the data line for at least 1 us. After 15 us from the falling edge of the data line, the bus master samples for slave data. If the data line is high, as shown, a one is read. If the line is low, as shown in Figure 13, a zero is read. A 1 us recovery period occurs at the end of the read slot in either case.

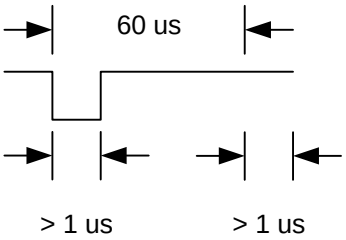


Figure 12. One-wire bus Read One Slot.

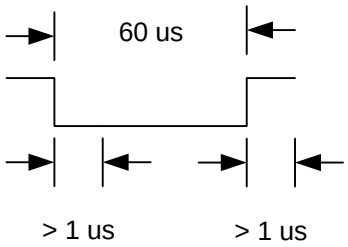


Figure 13. One-wire bus Read Zero Slot.

The I/O board one-wire bus master allows a user to perform one of four operations, as shown in Table 4. Byte Write operations are formed by sending eight (8) of the Write slots with data contained in the ONE_DOUT_REG. During a byte Read operations, 8 Read Slots are performed to fill the ONE_DIN_REG. All transfers occur from the LSB to the MSB.

Table 4. One-wire bus operation configuration with the ONE_CONFIG_REG.

Bits(1:0)	Bus Operation
"00"	Byte Write
"01"	Byte Read
"10"	ROM Search
"11"	Bus Reset

A typical one-wire Reset sequence follows: First, the CPU writes to the ONE_CONFIG_REG value “11” to configure the bus for a Reset. Next, the CPU writes a ‘1’ to the CONTROL_REG bit 2, initiating the bus reset. The DS_DQ_OUT line is asserted low as shown in Figure 9. After the elapsed reset time, the bus is released and the slave presence pulse is captured. SET_ONE_DONE_BIT pulses high for one cycle, causing the IO_INT_NOT line to be asserted on the backplane. The CPU responds to the interrupt via an ISR. The CPU reads the STATUS_REG and the completion of a one-wire bus operation is detected if a '1' appears in bit 2. If a slave presence pulse was received, the ONE_STATUS_REG LSB will contain a ‘1’. If a slave presence pulse was not received, a ‘0’ will appear in the LSB. At the end of the ISR, the STATUS_REG bit 2 is cleared, with a CPU write, to acknowledge receipt of the interrupt. The CPU may proceed with additional one-wire bus operations after a return from the ISR.

A typical One-Wire Byte Write sequence follows: First, the CPU writes to the ONE_CONFIG_REG value “00” to configure the bus for a Write. The outgoing bus data is written to the ONE_DOUT_REG. Next, the CPU writes a ‘1’ to the CONTROL_REG bit 2, initiating the bus write. Data is shifted out onto the DS_DQ_OUT line from LSB to MSB using Zero or One Write slots corresponding to the data in the ONE_DOUT_REG. When the bus operation completes, SET_ONE_DONE_BIT pulses high for one cycle, causing the IO_INT_NOT line to be asserted on the backplane. The CPU responds to the interrupt via an ISR. The STATUS_REG is read, and then STATUS_REG bit 2 is cleared to acknowledge receipt of the interrupt. The CPU may proceed with additional one-wire bus operations after a return from the ISR.

A typical One-wire Byte Read sequence follows: First, the CPU writes to the ONE_CONFIG_REG value “01” to configure the bus for a Read. Next, the CPU writes a ‘1’ to the CONTROL_REG bit 2, initiating the bus read. The slave device shifts data into the FPGA on the DS_DQ_IN line from LSB to MSB. The bus master uses Zero or One Read slots and stores the incoming data in the ONE_DIN_REG. When the bus operation completes, SET_ONE_DONE_BIT pulses high for one cycle, causing the IO_INT_NOT line to be asserted on the backplane. The CPU responds to the interrupt via an ISR. The STATUS_REG is read, then STATUS_REG bit 2 is cleared to acknowledge receipt of the interrupt. The CPU may proceed with additional one-wire bus operations after a return from the ISR.

A typical One-wire ROM Search sequence follows. This sequence describes the operations required to perform a single byte transfer. A single pass of the ROM Search process requires eight (8) byte transfers. This type of sequence would be used to query the one-wire bus in the case where the Slave IDs of the bus devices were unrecorded prior to construction of the bus. This is not recommended due to the potential difficulty associating Slave IDs with their associated devices. First, the CPU writes to the ONE_CONFIG_REG value “10” to configure the bus for a Search. The CPU writes the ONE_SRCH_O_REG with outgoing ROM comparison data to be written to the bus during the transfer. Next, the CPU writes a ‘1’ to the CONTROL_REG bit 2, initiating

the bus search. Data is shifted out onto the bus from LSB to MSB. For each bit position, the master performs a Write Slot using data from the ONE_SRCH_O_REG. Slaves whose address bit under consideration matches the written bit will respond. Other slave devices will not participate in ROM Search activities until they are reset. The master then performs two read slot operations. The first Read Slot fills one bit in the ONE_SRCH_I_NON_REG with a non-inverted address bit. The next Read Slot fills one bit in the ONE_SRCH_I_INV_REG with an inverted slave address bit. The bus master continues this sequence for all eight bits in the ONE_SRCH_O_REG. When the bus operation completes, SET_ONE_DONE_BIT pulses high for one cycle, causing the IO_INT_NOT line to be asserted on the backplane. The CPU responds to the interrupt via an ISR. The STATUS_REG is read, and then STATUS_REG bit 2 is cleared to acknowledge receipt of the interrupt. The CPU reads the ONE_SRCH_I_NON_REG and ONE_SRCH_I_INV_REG to determine the path to be taken for future ROM Search operations. The CPU may proceed with additional one-wire bus operations after a return from the ISR.

As shown in Figure 14, the one-wire bus master register interface is similar to the SPI bus master register interface. The ONE_WIRE_TOP entity receives nine (9) register write strobes from the ADDRESS_DECODER module during CPU write operations. The output of each register is available for read back verification through the READ_MULTIPLEXOR entity. During initialization of the I/O board, the ONE_DELAY_REG is written to scale the one-wire bus timing logic to the system clock. It is possible to adjust the one-wire bus timing to account for different CPU clock rates, and variations in bus timing arising from wire capacitance. It is only necessary to write the ONE_DELAY_REG upon I/O board initialization.

Another benefit of the ONE_DELAY_REG is that it allows the one-wire bus master to be ported to future designs. The VHDL code for this project could be either fully or partially reused in the future. And, because the one-wire bus master is system clock independent, it could be easily ported to another FPGA, with a different system clock rate, without hardware modification of the clock timescale.

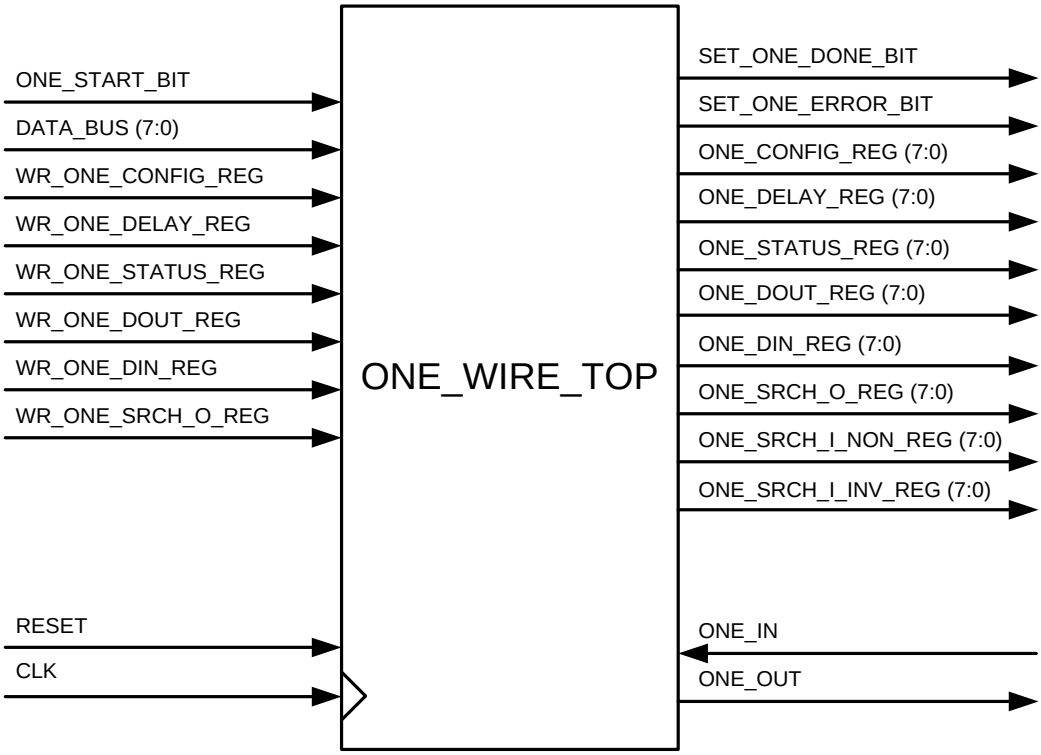


Figure 14. Top-level entity of one-wire bus master.

3.8 The 16-Bit A/D Bus Master

A 16-bit A/D Bus master residing in the I/O board FPGA allows the satellite CPU to proceed with normal operations during a data acquisition. The 16-bit Burr Brown ADS8344 accepts eight (8) 0.0 Volt to 5.0 Volt inputs. The total number of 16-bit A/D channels is increased to 16 with two MAX382 multiplexors. The 16-bit A/D interface serves as a 100 kHz general-purpose interface for state-of-health monitoring. On the top connector of the I/O board, the 16 bit A/D interface is available to provide monitoring of HokieSat subsystems. The 16 bit A/D interface follows a synchronous serial protocol as shown in Appendix G. The bit definitions shown in the Appendix G diagram correspond to signals in the Burr-Brown ADS8344 data sheet.

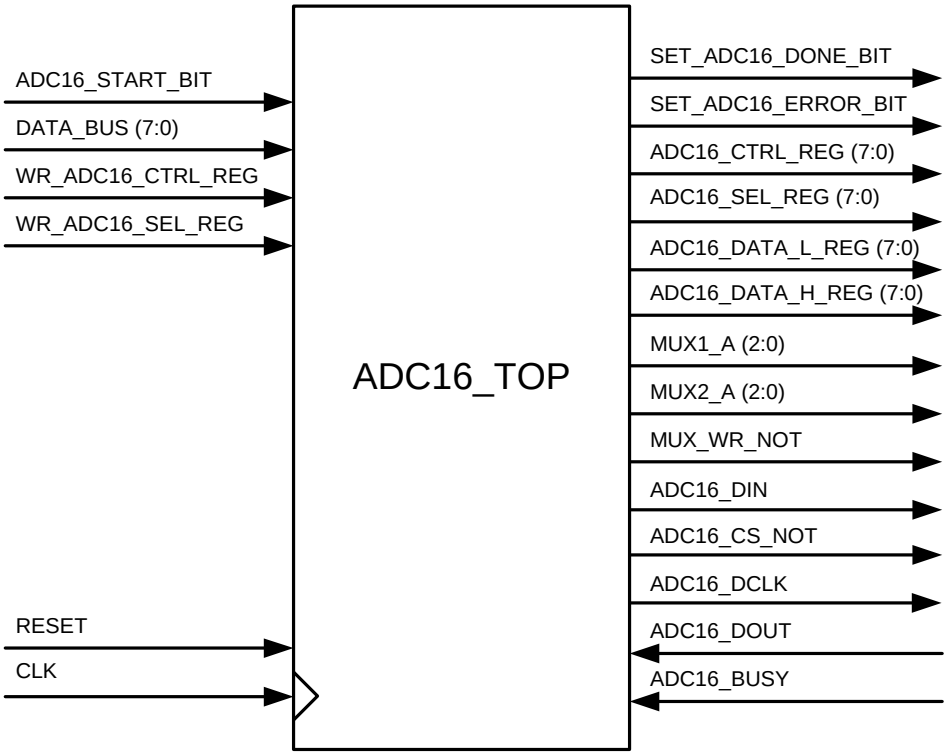


Figure 15. Top-level entity for the 16 Bit A/D Converter Bus Master.

Shown in Figure 15, the 16-bit A/D bus master register interface is similar to the SPI bus master register interface. The ADC16_TOP entity receives register write strobes from the ADDRESS_DECODER module during CPU write operations. The output of each register is available for read back verification through the READ_MULTIPLEXOR entity.

A typical 16-bit A/D conversion sequence follows: First, the CPU writes to the ADC16_SEL_REG to configure the channel expansion multiplexors. Next, the CPU writes to the ADC16_CTRL_REG to configure the ADS8344 conversion type. The control word is written to the A/D to configure the conversion parameters as shown the Burr Brown ADS8344 specification.

An ADC16 bus transfer is initiated by writing to CONTROL_REG bit 1. When the transfer is complete, the SET_ADC16_DONE_BIT pulses high for one cycle, causing the IO_INT_NOT line to be asserted on the backplane. The CPU responds to the interrupt via an ISR. The STATUS_REG is read by the CPU and the completion of a 16-bit A/D operation is detected. The ADC16_DATA_H_REG and ADC16_DATA_L_REGs are concatenated to form a 16-bit result. At the end of the ISR, the STATUS_REG bit 1 is cleared to acknowledge receipt of the interrupt. The CPU may proceed with additional operations after a return from the ISR.

3.9 The 12-Bit A/D Interface

Direct byte access to two MAX197 12-bit A/D converters is provided through decoding logic in the I/O board FGPA. The CPU writes the ADC12_CTRL0_REG to initiate a conversion according to the Maxim semiconductor MAX197 specification. Upon receipt of an I/O board interrupt, STATUS_REG bit 3 is set. The CPU concatenates ADC12_DH0_REG and ADC12_DL0_REG to form a 12-bit result. Unlike the other I/O board resources, writing to the STATUS_REG does not clear this interrupt condition. Instead, the condition is cleared when the ADC12_DH0_REG or ADC12_DL0_REG is read or when a new conversion is initiated. A similar set of operations is performed when converting with the second 12-bit A/D converter.

3.10 The FPGA Control And Status Registers

The I/O board control and status registers provide a structure under which FPGA resources may be operated independently. For example, one may initiate all of the I/O board functions simultaneously by writing ‘1’ to each bit in the CONTROL_REG and waiting for the results of each operation to cause I/O board interrupts. In general, interrupt conditions are cleared by writing a ‘1’ to set positions in the STATUS_REG and ERROR_REG. An exception to this rule is for the 12-bit A/D converters, which are accessed directly via a shared data bus. The CONTROL_REG, STATUS_REG, and ERROR_REG are contained in the CONTROL_TOP entity, shown in Figure 16.

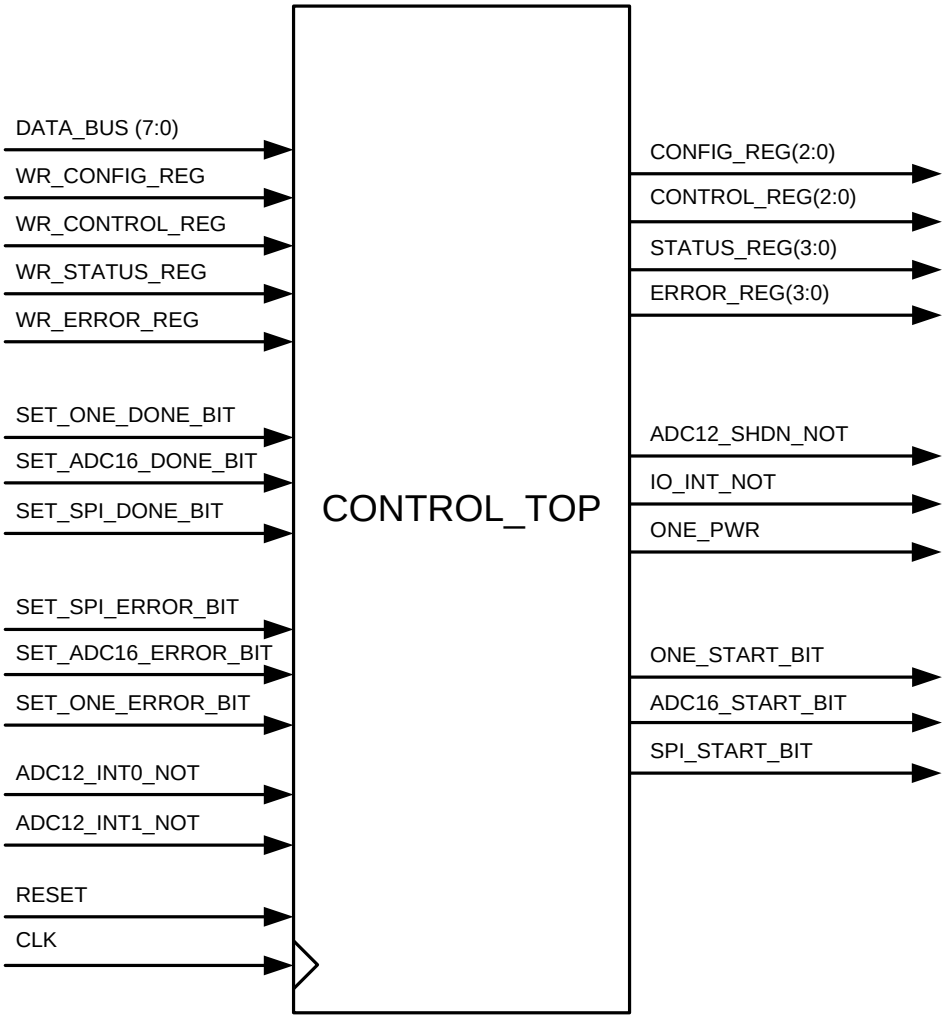


Figure 16. I/O Board Control entity block diagram.

The CONTROL_TOP entity synchronizes activities between the CPU and I/O board finite state machines (FSMs). When the CONTROL_REG is written, CONTROL_TOP generates ONE_START_BIT, ADC16_START_BIT, or SPI_START_BIT pulses as appropriate to initiate operations in corresponding FSMs. If errors are detected on the FSM-level during an operation or between operations, a corresponding error bit is set. If an FSM properly completes a sequence, a corresponding done bit is set. To allow debugging and read-modify-write operations, register values are output from the CONTROL_TOP entity. The CONFIG_REG also allows the CPU to disable the one-wire bus, mask I/O board interrupts, and disable the one-wire bus power.

3.11 I/O Board Interrupt Generation

The I/O board may generate two interrupt types. Status interrupts occur to alert software of a normally completed operation. Error interrupts occur when any I/O board FSM enters an invalid state. After an ERROR_REG bit is set, the affected FSM returns to an idle state. So, it is possible that only an error interrupt or a status interrupt be present after any given I/O board operation.

Either type of interrupt condition notifies the CPU that an event requires service from the CPU. Figure 17 shows the logic for generation of an active I/O board interrupt. When a status register or error register bit is set (1), the I/O board interrupt line will be asserted to the low state. The CPU performs a read operation of the STATUS_REG to determine if an operation has completed normally. Set bits in the STATUS_REG are cleared by writing a '1' to set bit positions. The CPU next reads the ERROR_REG to determine if an error occurred. Set bits in the ERROR_REG are cleared by writing a '1' to set bit positions. It is possible that an error would occur during a requested FPGA operation or as a result of a radiation event. A software error handler records a specific error type, allowing corrective action to be taken by a higher software process.

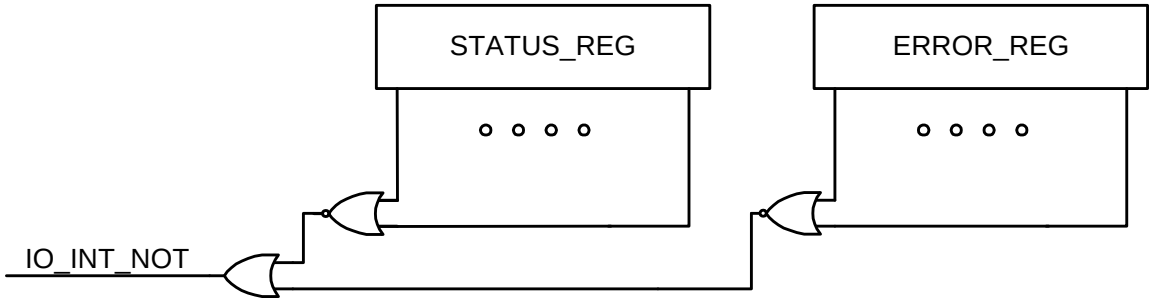


Figure 17. I/O Board Interrupt Generation Logic.

A benefit of this simple design is that the CPU can perform the fewest number of read and write operations during an ISR.

3.12 I/O Board Read Multiplexor

Figure 18 shows the I/O board read multiplexor top entity inputs and outputs. Because the I/O board uses only even addresses to access registers, the LSB of the CPU address is ignored. CPU address bits 6 through 1 are used to decode the I/O board registers to the byte-wide READ_DATA_BUS. Appendix A contains the I/O board address space.

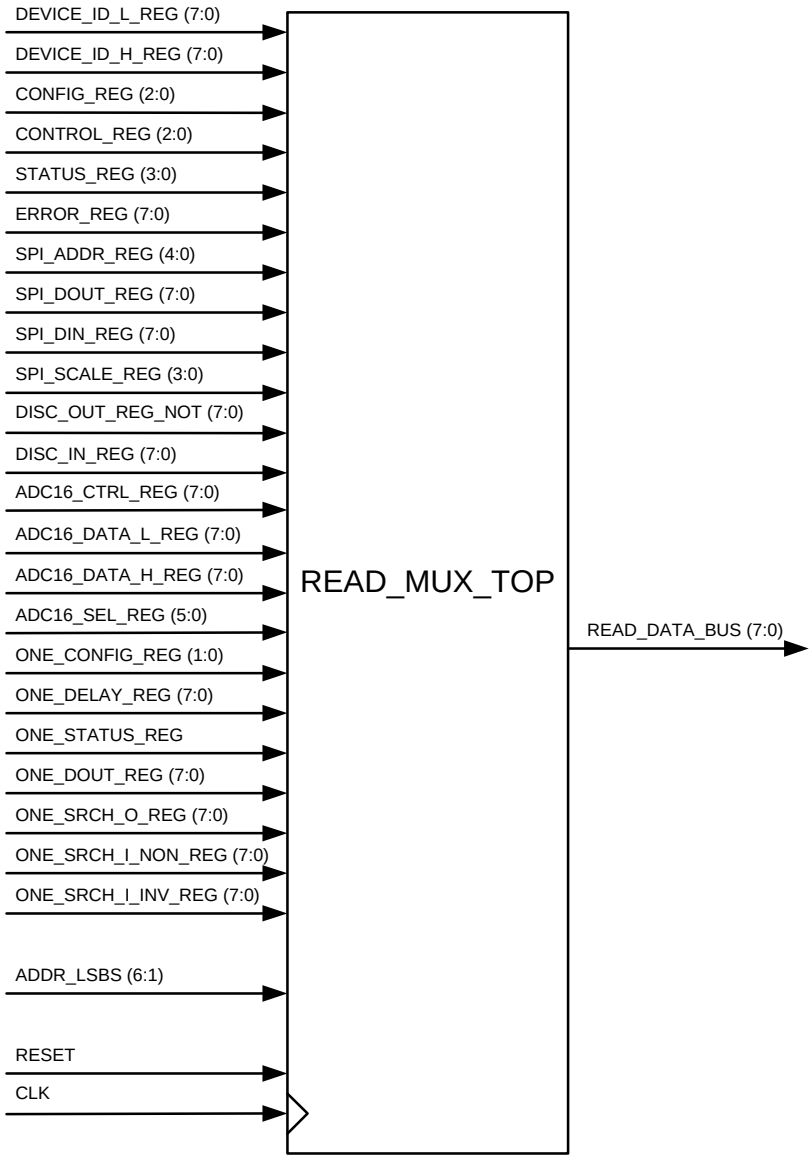


Figure 18. I/O Board Read Multiplexor entity block diagram.

In order to perform register read operations, an internal data bus must either allow tri-stating from tri-state capable register outputs, or an internal read multiplexor must be used. Because the Actel 42MX series FPGA does not contain an internal tristate buffer capability, a read multiplexor was required. A read multiplexor requires a significant amount of combinatorial logic compared to an internal HiZ bus, but it is a more portable design because not all FPGAs support internal HiZ data busses.

3.13 I/O Board CPU Interface Decoder

A purely combinatorial logic decoder entity, shown in Figure 19, controls CPU access to the I/O board. This entity translates the CPU read, write, and control signals into a local

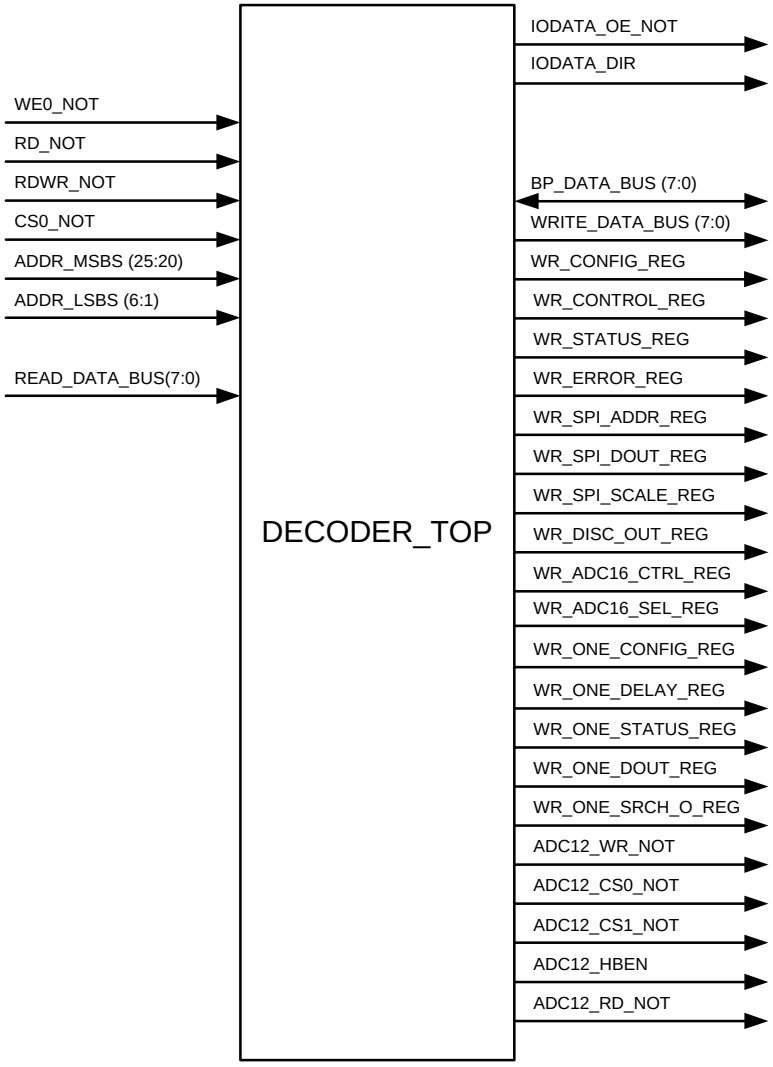


Figure 19. I/O Board Address Decoder.

simplified, set of signals used internally to the I/O board. During a CPU read operation, the READ_DATA_BUS is driven to the backplane data bus (BP_DATA_BUS). During a CPU write operation, the BP_DATA_BUS is written to an internal register whose write enable is asserted. The CPU address is partially decoded to save FPGA I/O lines when generating register read and write control signals. Because two 12-bit A/D converters are located externally to the FPGA, the decoder entity generates control signals for the CPU to directly access those devices.

3.14 I/O Board FPGA Development Tools and Verification

Once the individual I/O board entities were designed using VHDL, I combined them into a top-level entity called IO_BOARD_TOP. The top level VHDL code listing for the I/O board FPGA is shown in Appendix F. This top-level structural VHDL model was useful for tracing signal flow within the FPGA design, while encapsulating the details of lower level entities in separate files.

Although VHDL is quite portable, it was not always easy to see the overall architecture without a graphical tool. Using the Synplify Pro VHDL tool, by Synplicity, a graphical block diagram of the FPGA architecture was generated and appears in Appendix H. In hindsight, a tool which allowed graphical entry for structural models while allowing behavioral VHDL for lower level modules might have offered the best design environment. The I/O board design was entered entirely with a text-editor.

After successfully synthesizing the top level I/O board VHDL model with Synplify Pro, behavioral simulation was performed. ModelSim, by Model Technology, was utilized for all simulations. Several VHDL procedures were written and stored in a package to mimic the CPU interacting with the I/O board. A procedure called *write_byte* encapsulated the signal control and timing associated with a CPU write operation. A procedure called *read_byte* encapsulated the signal control and timing associated with a CPU read operation. By building test benches around these VHDL procedures, I generated 122 test-cases to verify functionality of the I/O board FPGA.

After behavioral simulation was complete, I synthesized the design for an Actel 16,000 gate A42MX16 FPGA. The design successfully synthesized with only 60% FPGA utilization and exceeded the system clock requirement of 20 MHz.

3.15 Summary

The structural VHDL model development of the I/O board FPGA has been documented in this chapter. The entity input and output signals and the access methods were described. By interconnecting several low level VHDL entities into a single entity, a model was generated to meet the specified requirements of the ION-F I/O board.

The I/O board FPGA architecture allowed the ION-F CPU to offload some satellite health monitoring and interface functionality to a secondary module. By implementing the one-wire bus, SPI bus, and 16 bit A/D converter interfaces in independent hardware, the bus timing was guaranteed and separate from software timing limitations in the CPU and OS environment. By using interrupt driven control to the CPU with status and error registers, an efficient interface was implemented, simulated, and synthesized. A complete listing of the I/O board FPGA signal names appears in Appendix C.

Chapter 4

I/O Board Software Architecture

4.1 Introduction

Health monitoring and input/output (I/O) operations in the ION-F computer system may require a substantial amount of time compared to the capabilities of the satellite CPU. The use of supplemental hardware for frequently accessed and timing-specific data transfers frees the CPU hardware and software of some of the subsystem overhead. Instead of dedicating discrete inputs or outputs from the CPU, the I/O board provides memory-mapped output bits for general-purpose use. The relatively slow SPI bus is an example of a resource, which would have consumed CPU counter or timer as well as software complexity if it were implemented by the CPU instead of using the I/O board dedicated hardware.

The purpose of this chapter is to discuss real-time software device drivers for the VxWorks operating system to help ensure that the efficiency gained with custom hardware is not lost with inefficient software practices. Hardware resources must be coupled with software protection in a real-time operating system environment to prevent resource conflicts and optimize performance. The use of semaphores and interrupt locking for protecting critical sections of code is discussed in this chapter. By first identifying which resources require protection, and designing a protection mechanism appropriate for each resource, a guideline is generated here for future programming efforts.

4.2 I/O Board Resource Access

Protected resources, such as those using Finite-State-Machines (FSMs), those requiring more than one CPU Write cycle to complete an operation, or those with several registers used by the CPU to execute an operation, must be guarded in software. This is done using semaphores under VxWorks. A summary of the I/O board hardware resources and the required protection for each resource is shown in Table 5.

Table 5. I/O Board Hardware Resource Protection Summary.

I/O Board Hardware Resource	Protected Resource?	Protection Method
12-Bit A/D Converters	Yes	Semaphore
16-Bit A/D Converters	Yes	Semaphore
SPI Bus	Yes	Semaphore
One-wire bus	Yes	Semaphore
Discrete Output Register	Yes	Semaphore
Discrete Inputs	No	None

If a protected resource were not guarded in the real-time-operating system environment, it would be possible for two tasks to simultaneously access the resource. A conflict could occur when control and data registers received information from two tasks for different operations if the resources were not protected. It is quite possible that one task would interfere with another and that an erroneous operation would occur. In the I/O board architecture, the protected resources are the SPI bus master, the one-wire bus master, the 16-Bit bus master, the 12-Bit A/D converters, and the Discrete Output Register. It is not necessary to protect the Discrete Input resource, because the CPU may not access that resource except for a single read operation. Thus, it is not possible for a multitasking conflict to occur with the unprotected resources.

Semaphore protection is proposed for I/O board hardware resource protection. VxWorks allows creation of binary semaphores, counting semaphores, and mutual exclusion or Mutex semaphores. For hardware with multiple channels or several similar module types, a counting semaphore may be most appropriate. For resources, which may be accessed by at most one task at a time, the binary or Mutex semaphore type is most appropriate. All I/O board resources should be protected using mutual exclusion or

binary semaphores. A second method for protecting resources under VxWorks is the use if interrupt locking. Locking system interrupts prevents an OS task context switch, which allows a resource to be briefly accessed as a critical section of code. After re-enabling system interrupts, normal multitasking resumes. The use of semaphores versus interrupt locking depends on the amount of time that a resource will use and whether it is acceptable to suspend either a task or the entire OS during a hardware operation.

Several steps are required for device driver initialization. First, binary semaphores are created for each resource using the VxWorks *semBCreate()* function. The semaphores may be either full or empty, but they are created as full using the SEM_FULL parameter in the *semBCreate()* function. Next, a C-language ISR is mapped to the interrupt vector table. The *intHandlerCreate()* function generates an function pointer to the interrupt service routine to be registered. The ISR function pointer is mapped to the interrupt vector table using the *intVecSet()*. I/O board interrupts are next enabled by setting the LSB of the I/O board CONFIG_REG. When the CPU enables its I/O board interrupt using the *intEnable()* command, backplane I/O board interrupts will call the I/O board ISR via a lookup mechanism in the interrupt vector table. One may verify that the I/O board ISR is properly installed by viewing the interrupt vector table. This is done by displaying a CPU memory block following the *sysIntTable* address [Win99a].

Device driver access to a protected resource begins with a *semTake()* command, which is an attempt to acquire the protected resource. If the resource is available, the semaphore is taken and the device driver operation continues. If the resource is not available, the requesting task is blocked until the semaphore becomes available. When the I/O board interrupt service routine (ISR) is called, the results of an operation are read. After the results of the device driver operation are stored, a semaphore is released by the ISR using the VxWorks *semGive()* command. At this time, tasks suspended due to pending *semTake()* operations are given the opportunity to acquire the protected resource and become unblocked [Win99b]. OS ceiling protocols prevent priority inversion.

An I/O board interrupt indicates the completion of a FSM sequence by either normal termination or due to a hardware error condition based on the values in the I/O board STATUS_REG and ERROR_REG. The I/O board ISR reads the STATUS_REG and ERROR_REG in order to determine the source of the interrupt condition. If the STATUS_REG contains any set bits, corresponding semaphores are released for the set bit positions. If the ERROR_REG contains any set bits, corresponding errors are logged to a message queue for the set bit positions. At the end of the ISR, serviced STATUS_REG and ERROR_REG bit positions are cleared and the return from ISR command is issued.

Some special cases require additional programming considerations. First, it is possible to turn off any HokieSat backplane board to save power and provide additional radiation protection. Before that done, it is necessary to first disable I/O board interrupts at the CPU level using the VxWorks *intDisable()* command. Before the I/O board is powered back on, all I/O board semaphores should made available using the *semGive()* command. This ensures that semaphores are not accidentally left in a taken (SEM_EMPTY) state because the I/O board was powered off before all pending operations were completed. If the I/O board is powered off without considering pending device driver operations, it is possible that device driver *semTake()* functions will reach their timeout limits. In order to prevent the I/O board from being powered off while a device driver operation is in progress, a counting semaphore is created with a count of four (4). Each I/O board resource will take the counting semaphore prior to performing an operation. Only when the counting semaphore is full are all of the I/O board resources free, meaning it is safe to power off the I/O board.

For proper access to the I/O board resources, some programming considerations are necessary. Because a C compiler may attempt to optimize variables into CPU registers to simplify executable code, it may be necessary to use the ‘volatile’ keyword. This forces the compiler to perform I/O board register access in a memory-mapped fashion, instead of to the general-purpose CPU register file.

4.3 Device Driver Sequences for I/O Board Resources

Access to the I/O board discrete output register (DISC_OUT_REG) does not require protection if the register is being initialized or if the entire register byte is updated simultaneously. However, if the register is allocated to several different satellite functions, protection is required because a read-modify-write operation will be required to avoid disruption to bit positions used in other functions. The VxWorks *intLock()* and *intUnlock()* functions allow the user to briefly disable all CPU interrupts, perform a read-modify-write operation, then re-enable CPU interrupts. This prevents task context switching during the read-modify-write operation. If the DISC_OUT_REG were not protected with *intLock()* and *intUnLock()*, it is possible that task context switch could occur after the read operation. Another read-modify-write operation to the register could occur by an interrupting task, and the original task could overwrite its DISC_OUT_REG value. Although this is an unlikely scenario, it could occur without proper resource protection. Semaphore protection is recommended, as good practice, instead of interrupt locking.

Access to the 12-bit A/D converters is protected with a semaphore called SEM_ADC12. Upon device driver initialization, SEM_ADC12 is available since the resource is not taken. After SEM_ADC12 is taken, the device driver may perform a 12-bit A/D conversion. First, the ADC12_CTRL_REG is written to initiate the sequence. When the conversion completes, the STATUS_REG bit 3 is set and an I/O board interrupt is generated. In the I/O board ISR, the A/D conversion results are read from the ADC12_DH_REG and ADC12_DL_REG and concatenated into a single 12-bit value. These registers are read directly from the MAX197 A/D converter on the I/O board. The ISR next performs a VxWorks *semGive()* operation on the SEM_ADC12 semaphore, allowing additional tasks to perform A/D conversions.

Access to the 16-bit A/D converters is protected with a semaphore called SEM_ADC16. Upon device driver initialization, SEM_ADC16 is available since the resource is not taken. After SEM_ADC16 is taken, the device driver may perform a 16-bit A/D conversion. First, the ADC16_CTRL_REG is written to set up the Burr-Brown

ADS8344 A/D converter control word. A write to the CONTROL_REG bit 1 initiates the conversion. When the conversion completes, the STATUS_REG bit 1 is set and an I/O board interrupt is generated. In the I/O board ISR, the A/D conversion results are read from the ADC16_DATA_H_REG and ADC16_DATA_L_REG and concatenated into a single 16-bit value. The ISR next performs a VxWorks *semGive()* operation on the SEM_ADC16 semaphore, freeing the resource.

Access to the SPI bus is protected with a semaphore called SEM_SPIBUS. Upon device driver initialization, SEM_SPIBUS is available since the resource is not taken. After SEM_SPIBUS is taken, the device driver may perform an SPI bus transfer. First, the SPI_ADDR_REG is written to specify the target SPI bus slave device. For write operations, the SPI_DOUT_REG is pre-loaded with outgoing data. A write to the CONTROL_REG bit 0 initiates the transfer. When the transfer completes, the STATUS_REG bit 0 is set and an I/O board interrupt is generated. In the I/O board ISR, the SPI_DIN_REG value is stored to a local variable. It is returned in to the user when an SPI bus read operation is performed. The ISR next performs a VxWorks *semGive()* operation on the SEM_ADC16 semaphore, freeing the resource.

Access to the one-wire bus is protected with a semaphore called SEM_ONEWIRE. Upon device driver initialization, SEM_ONEWIRE is available since the resource is not taken. After SEM_ONEWIRE is taken, the device driver may perform a one-wire bus transfer. The setup steps for a one-wire bus transfer vary with respect to the type of transfer. These steps are fully documented in Chapter 3. A write to the CONTROL_REG bit 2 initiates the transfer. When the transfer completes, the STATUS_REG bit 2 is set and an I/O board interrupt is generated. In the I/O board ISR, the ONE_DIN_REG value is stored to a local variable. It is returned to the user when a one-wire bus read operation is performed. The ISR next performs a VxWorks *semGive()* operation on the SEM_ONEWIRE semaphore, freeing the resource.

4.4 Summary

The software architecture for accessing I/O board resources in the VxWorks real-time OS environment was presented in this chapter. Several steps were followed during the device driver development process. First, the I/O board hardware resource types were identified and classified according to the type of protection required. Second, a general hardware initialization scheme was developed, including remote hardware initialization as well as OS-specific interrupt service routine registration and enabling. Next, specific details for accessing each hardware resource were documented. By properly designing a software protection mechanism around custom hardware, the high efficiency of a real-time OS and the underlying hardware can be realized. A recommended set of functions for accessing all I/O board hardware features appears in Appendix E.

Chapter 5

Summary

5.1 Summary of Research

In the fall of 1998, a contract was awarded to Virginia Tech to develop one of three student satellites in the formation-flying cluster known as ION-F. The student-led computer team initially evaluated low-cost and low-power CPUs. The computer team also evaluated low-cost and available operating systems to preserve the program budget and achieve delivery for a launch date of the fall of 2001. Anticipating a lengthy software integration process and environmental testing, an ARM processor, the Sharp LH77790B, and a downloadable operating system solution were proven functional by the Fall Semester of 1999.

However, a NASA directive forced the Virginia Tech computer team to use the VxWorks operating system, increasing the system cost and schedule. Due to incompatibilities with VxWorks, the Sharp LH77790B processor was abandoned in favor of a Hitachi SH7709 processor. At that time, Utah State University (USU) also assumed responsibility for developing a common, modular computer system for all three ION-F universities. Interest among members of the Virginia Tech computer team dwindled and I was left to design the I/O board, a health monitoring module to be flown on board all three ION-F flight computers. The common computer architecture, health monitoring methods, and radiation protection schemes were described in Chapter 2.

Chapter 3 described the I/O board hardware, which allowed the CPU to offload four major functions to secondary hardware. An SPI bus extended backplane communications and provided a simple interface for satellite science, power, and attitude-determination

and control hardware modules. An important feature of the SPI bus was the updating of a system-wide watchdog timer in the power board. Failure to reset the system wide watchdog timer resulted in a power cycling to recover from radiation events. A one-wire bus master provided ION-F with a flexible and configurable health monitoring solution inside the satellite. The bus was configured for each satellite according to their specific needs. Analog to digital converter entities provided additional sensing capabilities inside the satellites. These were useful for health monitoring as well as feedback for control algorithms. Last, discrete input and output registers were designed with TTL-level drive capability.

Although I began a board-level design at Virginia Tech using the Eagle PCB layout editor and then OrCad's Layout product, it was decided to move that work to USU to make the I/O board design maintainable. Issues such as adopting the proper space PCB layout standards, as well as setting up the PCB design files for a pick and place machine were also factors for moving the PCB design to USU.

To support the I/O board custom hardware, I proposed a set of VxWorks device drivers in Appendix E, and provided details for their implementation in Chapter 4. It was necessary to first identify all of the hardware resources, and categorize them into groups based on the required protection mechanisms to ensure good performance in the a real-time operating system environment. Next, specific OS functions were identified for use in implementing the I/O board device drivers. The device driver initialization and usage was described.

In conclusion, I have developed a hardware/software health monitoring solution for the ION-F computer system. 122 FPGA hardware simulations have been performed and the design was synthesized to meet device size and timing specifications for an Actel A42MX16 FPGA operating at 20 MHz. The I/O board serves as a reusable design module in future missions after ION-F. That two of the major design modules, the one-wire bus and the SPI bus, are implemented in VHDL will greatly contribute to the possibility of design reuse.

5.2 Future Work

The ION-F computer architecture and implementation represents a reusable design, which could be suitable for future nanosatellite missions. In hindsight, although the ION-F modular computer design took longer and cost more than the originally proposed Virginia Tech computer team solution, it raised the overall chances of success within the mission by increasing the amount of common hardware and software in the ION-F satellite cluster.

However, improvements could be made to the way that distributed projects are coordinated, especially regarding requirement definitions. The Virginia Tech computer team was originally instructed to pursue a custom hardware design, with a very tight delivery date. This caused a 5 to 10% budget loss, at least one year of lost development time, and a loss of team membership and interest. If the computer requirements were well defined and documented to the team prior to beginning the mission, the goal could have been pursued more efficiently. Although properly defining, documenting, and disseminating the project goals would appear to be a major portion of a $300,000 NASA program, letting students figure out the requirements definition of a project also had educational value. Initially in the HokieSat project, there was a high level of resistance to documenting the design, but as students left the project, the importance of documentation increased to minimize confusion.

One significant tool for communicating information in the ION-F project was the Internet. Web sites for each university design group stored information so that when graduating students left the teams, their contributions were not lost. An e-mail list-serve was also used for ION-F, but participation on the listserve decreased throughout the project. It might have been better to use an online message forum instead of e-mail so that incoming team members could read old message posts to become familiar with the project.

Logistically, the nanosatellite technology could be improved by making the ION-F design information fully available to future design teams. This is especially true for large efforts

such as the computer design. Enough specific details about the satellite modules should be made available to allow them to be reconstructed. This could be done with a read-only database accessible through the Internet. In the ION-F project, the design was largely kept internal to the institutions developing technologies, making it more difficult to access the design information. This is especially true for space fabrication techniques and the equipment required to properly design, populate, coat, and test space-qualified circuit boards.

Design reuse is a major incentive to keep design costs low. Nanosatellites, weighing 10 kg or less, will soon be replaced by smaller picosats and femtosats. In order to maximize the amount of a design, which may be ported to a smaller future technology, the design must be well documented and available to future design groups. The use of standards, such as VHDL and C, along with meaningful documentation would aid the portability process. However, the availability of a set of industry standard tools such as PCB layout software and the software operating system should be considered just as important for design portability and reuse. One major factor, which caused additional work in ION-F was the lack of an industry standard or at least team-wide PCB layout package.

Nanosatellite technology and the TechSat 21 program suggest a promising future for satellite missions and technologies. Clusters of satellites may be used for image enhancement looking down toward earth in military satellite applications, or looking outward into space for exploration. The capability of reusing and reconfiguring satellite clusters is also promising from a cost and capability standpoint. A significant amount of research and development could be performed on re-configurable satellite applications and algorithms.

Bibliography

[Act98a] Actel Corporation. *Benefits of the MX Family of Devices Application Note.* Page 1. September, 1998.

[Act98b] Actel Corporation. *Ehanced Tools for Minimizing Single Event Upset Effects (Included in the Designer Series RS-1998 Software).* Pages 1-4,6. 1998.

[ABB99] Matt Adams, Bryce Bolton, Jason Bright, Shital Chheda, Earle Clubb, Meghan Everett, Stephen Hauge, Kevin Martin. *HokieSat Computer Team Flight Computer Preliminary Design Report.* May 6, 1999.

[Das00] A. Das. Choreographing Affordable, Next-Generation Space Missions Using Satellite Clusters. *AFRL Technology Horizons*, 1(3): 15-16, September 2000.

[Hal98] Dr. Christopher Hall. *Virginia Tech Ionospheric Scintillation Measurement Mission (HokieSat).* Kickoff Meeting Presentation, Kirtland Air Force Base, NM. 1998.

[Hal01] Dr. Christopher Hall. Private Communication October 22, 2001.

[Jen00] John D. Jensen. *The Design of the Command and Data Handling Subsystem Used by the Ionospheric Observation Nanosatellite Formation.* Utah State University Master's Thesis. 2000.

[Lab99] Jean J. Labrosse. *MicroC/OS-II The Real-Time Kernel.* R&D Books. Page 11, 1999. Operating system downloads were obtained from www.ucos-ii.com.

[Mot91] Motorola, Inc. *M68HC11 Reference Manual Rev 3*, pages 8-1 and 8-2, 1991.

[Red99] Red Hat. *http://www.redhat.com/embedded/technologies/ecos/.* 1999.

[SCR01] Jana Schwartz, Bryce Carpenter, Christopher Rayburn. *Systems Engineering for the Ionospheric Observation Nanosatellite Formation Project.* SmallSat Systems Conference. Logan Utah, August 2000.

[Sha98] Sharp Microelectronics of the Americas. *LH7790 A/B Embedded Microcontroller User's Guide.* Version 1.0, 1998.

[Win99a] Wind River Systems. *VxWorks Programmer's Guide, 5.4*, Edition 1, March 1999.

[Win99a] Wind River Systems. *VxWorks Reference Manual, 5.4*, Edition 1, April 1999.

Appendix A

I/O Board Address Space

This appendix contains the I/O board address space and register definition. Registers may be accessed from the ION-F CPU using 16-bit (unsigned short) access in the C programming language unless otherwise specified. The I/O board base address is 0x00F0_0000. Thus, ORing the base address with a register offset listed below forms a complete I/O board register address.

Offset	Register Name	Description
0x00	**DEVICE_ID_L_REG**	Read-Only This register may be read by the CPU to verify that the I/O board is powered on and the FPGA has been properly programmed. **Bits 7:0**: Reset Value = 0x10
0x02	**DEVICE_ID_H_REG**	Read-Only Reset Value = 0xB0 This register may be read by the CPU to determine the FPGA revision. This may be useful for tracking hardware/software compatibility. **Bits 7:0**: 0xB0 = I/O board FPGA Rev 0 0xB1 = I/O board FPGA Rev 1 0xB2 = I/O board FPGA Rev 2
0x04	**CONFIG_REG**	Read/Write Reset Value = 0x00 The CPU writes to this register to control various I/O board configuration options. This register may be read back for verification purposes. **Bits 7:3**: RESERVED. **Bit 2**: 1-Wire Bus power. **Bit 1**: ADC12 Shutdown# bit. **Bit 0**: I/O board interrupts enabled if ‘1’.

0x06 **CONTROL_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to initiate FSM activities in the I/O board FPGA. This register may be read back for verification purposes. One must not clear the bits in this register; the appropriate bits are cleared following a CPU write to the STATUS_REG.

Bits 7:3 RESERVED.
Bit 2: Write '1' to initiate a 1-wire bus operation.
Bit 1: Write '1' to initiate a 16-bit ADC read.
Bit 0: Write '1' to initiate a SPI bus operation.

0x08 **STATUS_REG**

Read/Write
Reset Value = 0x00
The CPU reads this register following an I/O board interrupt. Writes clear bit positions in which write data bits = '1'. All FSM operations end with an I/O board interrupt, followed by a STATUS_REG read.

Bits 7:4 RESERVED.
Bit 3: A 12-bit interrupt is pending.
Bit 2: A 1-Wire bus operation completed normally.
Bit 1: A 16-bit ADC operation completed normally.
Bit 0: An SPI bus operation completed normally.

0x0A **ERROR_REG**

Read/Write
Reset Value = 0x00
The CPU reads this register following an I/O board interrupt. A CPU Write clears bit positions in which write data bits = '1'. All FSM operations end with an I/O board interrupt, followed by an ERROR_REG read.

Bit 7: The last 16-bit ADC operation failed to report a done strobe.
Bit 6: The last 12-bit ADC operation failed to report a 'done' strobe.
Bit 5: RESERVED.
Bit 4: RESERVED.
Bit 3: RESERVED.
Bit 2: An invalid 1-Wire control state was entered.
Bit 1: The 16-bit ADC FSM entered an invalid state.
Bit 0: The SPI bus FSM entered an invalid state.

0x0C **SPI_ADDR_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to specify the SPI bus address, which will be used for SPI bus operations until this register is modified.

Bits 7:5: RESERVED
Bits 4:0: 5-bit SPI bus address.

0x0E **SPI_DOUT_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to specify outgoing SPI bus data, which will be used for the SPI bus operations until this register is modified.

Bits 7:0: SPI bus data out.

0x10 **SPI_DIN_REG**

Read Only
Reset Value = 0x00
This register receives SPI bus data during each SPI bus operation. The CPU reads this register following an I/O board interrupt.

Bits 7:0: SPI bus data in.

0x12 **SPI_SCALE_REG**

Read/Write
Reset Value = 0x06
This register contains a scale factor used to specify the SPI clock data rate. Some SPI Slave devices may require a slow data rate, and this register allows those devices to be accommodated while preserving the ability to more rapidly configure faster devices if so desired.
f_{SPI_CLK} = 20MHz/2^(n+5), where n is an integer from 0 to 15, inclusive.

Bits 7:4: RESERVED.
Bits 3:0: SPI clock scale

0x14 **DISC_OUT_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to specify the discrete output data byte, which is written to the top connector of the I/O board.

Bits 7:0: Discrete data out.

0x16 **DISC_IN_REG**

Read Only
Reset Value = 0x00
This register contains the discrete input byte, which is read from the top connector of the I/O board. The CPU may read this register at any time.

Bits 7:0: Discrete data in

0x18 **ADC16_CTRL_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to specify the data conversion control byte according to the Burr-Brown ADS8344 specification.

Bits 7:0: Discrete data out.

0x1A **ADC16_DATA_L_REG**

Read Only
Reset Value = 0x00
Upon receipt of an I/O board interrupt with a completed ADC16 operation indicated in the STATUS_REG, this register contains the lower 8 bits of the last 16-bit ADC conversion.

Bits 7:0: 16-bit ADC results.

0x1C **ADC16_DATA_H_REG**

Read Only
Reset Value = 0x00
Upon receipt of an I/O board interrupt with a completed ADC16 operation indicated in the STATUS_REG, this register contains the upper 8 bits of the last 16-bit ADC conversion.

Bits 7:0: 16-bit ADC results.

0x1E **ADC16_SEL_REG**

Read/Write
Reset Value = 0x00
The CPU writes to this register to specify which ADC16 channel is selected through 8x1 multiplexors.

Bits 7:6: RESERVED
Bits 5:3: Address for multiplexor 1.
Bits 2:0: Address for multiplexor 0.

0x20 **ONE_CONFIG_REG**

Read/Write
This register specifies the type of the next one wire bus transfer.
Bits 7:3: RESERVED.
Bits 1:0: "00" Initiates a Write.
"01" Initiates a Read.
"10" Initiates a ROM_SEARCH.
"11" Initiates a Reset.

0x22 **ONE_DELAY_REG**

Read/Write
Range 0 to 0xFF
This is a scaling factor to set the rounded number of clock cycles required to achieve a 500 ns time base for the one-wire bus. For a 20.000 MHz clock, ONE_DELAY_REG = 10 (decimal).

Bits 7:0: One-wire bus timer scale.

0x24 **ONE_STATUS_REG**

Read/Write
Range 0 to 0xFF
Bit 0: If '1', then a presence pulse was received after the RESET command was issued. If '0', then a presence pulse was not received after the RESET command.

Bits 7:1: RESERVED.
Bit 0: One-wire bus presence pulse.

0x26 **ONE_DOUT_REG**

Read/Write
Range 0 to 0xFF
This register contains a command word or data word to be written to a one-wire device during a write operation initiated by the CONTROL register.

Bits 7:0: One-wire bus outgoing data.

0x28 **ONE_DIN_REG**

Read Only
Range 0 to 0xFF
This register contains a data word received from a one-wire bus device during a read operation initiated by the CONTROL register.

Bits 7:0: One-wire bus received data.

0x2A **ONE_SRCH_O_REG**

Read/Write
Range 0 to 0xFF
This register contains ROM comparison 8 bits of an address arbitration word issued to a one-wire device during a SEARCH_ROM operation initiated by the CONTROL register. 8 SEARCH_ROM commands are issued to arbitrate for the entire 64 bit one-wire address. If there is only one 1-wire device on the bus, or if DEVICE_IDs are known for all devices on the 1-wire bus, then a ROM SEARCH is not required; A SKIP_ROM command may be issued instead.

Bits 7:0: One-wire bus SEARCH_ROM data.

0x2C **ONE_SRCH_I_NON_REG**

Read Only
Range: 0 to 0xFF
This register contains the most recent 8 bits of non-inverted arbitration data received from the 1-wire bus. Eight (8) SEARCH_ROM commands are issued to arbitrate for an entire 64-bit one-wire bus address. This register is read 8 times, once on completion of each SEARCH_ROM command. If there is only one 1-wire device on the bus, or if DEVICE_IDs are known for all devices on the 1-wire bus, then a SEARCH_ROM sequence is not required. A SKIP_ROM command can be issued instead. See address 0x2E for the ONE_SRCH_I_INV_REG corresponding data.

Bits 7:0: One-wire bus incoming SEARCH_ROM non-inverted data.

0x2E **ONE_SRCH_I_INV_REG** Read Only
Reset Value = 0x00
This register contains the most recent byte of inverted arbitration data received from the 1-wire bus.

Bits 7:0: One-wire bus incoming SEARCH_ROM inverted data.

0x30 **ADC12_CTRL0_REG** Read/Write
Reset Value = 0x00
This register specifies the first MAX197 control byte according to the MAX197 specification.

Bits 7:0: Control word data.

0x32 **ADC12_DL0_REG** Read Only
Reset Value = 0x00
This register contains the lower 8 bits of the last 12-bit ADC operation results for the first MAX197.

Bits 7:0 Lower byte of most recent 12-bit conversion.

0x34 **ADC12_DH0_REG** Read Only
Reset Value = 0x00
This register contains the upper 4 bits of the last 12-bit ADC operation results for the first MAX197.

Bits 7:4: RESERVED
Bits 3:0: 12-bit results bits 11:8.

0x36 **ADC12_CTRL1_REG** Read/Write
Reset Value = 0x00
This register specifies the first MAX197 control byte according to the MAX197 specification.

Bits 7:0: Control word data.

0x38 **ADC12_DL1_REG** Read Only
Reset Value = 0x00
This register contains the lower 8 bits of the last 12-bit ADC operation results for the first MAX197.

Bits 7:0 Lower byte of most recent 12-bit conversion.

0x3A **ADC12_DH1_REG**

Read Only
Reset Value = 0x00
This register contains the upper 4 bits of the last 12-bit ADC operation results for the first MAX197.

Bits 7:4: RESERVED
Bits 3:0: 12-bit results bits 11:8.

Appendix B

I/O Board Backplane Pin-out

Table 6. This table identifies the pin-out of the I/O board backplane connector.

Row 1 Pin #	Signal	Row 2 Pin #	Signal	Row 3 Pin#	Signal
1	D0	**42**	NC	**82**	D8
2	D1	**43**	NC	**83**	D9
3	D2	**44**	NC	**84**	D10
4	D3	**45**	PWR_IO	**85**	D11
5	D4	**46**	NC	**86**	D12
6	D5	**47**	NC	**87**	D13
7	D6	**48**	NC	**88**	D14
8	D7	**49**	DGND	**89**	D15
9	DGND	**50**	/RESETOUT	**90**	DGND
10	SYS_CLK	**51**	NC	**91**	A25
11	1-WIRE	**52**	NC	**92**	A24
12	/CS0	**53**	NC	**93**	A23
13	/CS5	**54**	LM_IO	**94**	A22
14	/CS6	**55**	NC	**95**	A21
15	/BS	**56**	NC	**96**	A20
16	/WE0	**57**	NC	**97**	A19
17	/WE1	**58**	NC	**98**	A18
18	/RD	**59**	SPI_SS#	**99**	A17
19	RDWR	**60**	SPI_A4	**100**	A16
20	NC	**61**	SPI_A3	**101**	A15
21	NC	**62**	SPI_A2	**102**	A14
22	PTB[2]	**63**	SPI_A1	**103**	A13
23	PTB[3]	**64**	SPI_A0	**104**	A12
24	PTB[4]	**65**	NC	**105**	A11
25	PTB[5]	**66**	NC	**106**	A10
26	PTB[6]	**67**	IRQ_IO	**107**	A9
27	PTB[7]	**68**	NC	**108**	A8
28	PTC[7]	**69**	NC	**109**	A7
29	PTC[6]	**70**	IRQ_OVERCUR	**110**	A6
30	PTC[5]	**71**	SPI_CLK	**111**	A5

31	PTC[4]	**72**	SPI_MOSI	**112**	A4
32	PTC[3]	**73**	SPI_MISO	**113**	A3
33	PTC[2]	**74**	-UNREG	**114**	A2
34	-UNREG	**75**	+UNREG	**115**	A1
35	+UNREG	**76**	AGND	**116**	A0
36	AGND	**77**	-12V	**117**	AGND
37	-12V	**78**	12V	**118**	-12V
38	12V	**79**	-5V	**119**	12V
39	-5V	**80**	5V	**120**	-5V
40	5V	**81**	3.3V	**121**	5V
41	3.3V			**122**	3.3V

Appendix C

I/O Board FPGA Signals

Table 7. I/O board FPGA signals, directions, types, and reset values.

Signal Name	FPGA Pin Type	Reset Value	Description
SPI_ADDR_4	Output	0	SPI Address out.
SPI_ADDR_3	Output	0	SPI Address out.
SPI_ADDR_2	Output	0	SPI Address out.
SPI_ADDR_1	Output	0	SPI Address out.
SPI_ADDR_0	Output	0	SPI Address out.
SPI_SS_NOT	Output	1	SPI chip select.
SPI_MISO	Input	-	SPI Data to FPGA.
SPI_MOSI	Output	0	SPI Data from FPGA.
SPI_CLK	Output	0	SPI bus serial clock.
DISC_OUT_NOT_7	Output	1	Discrete output.
DISC_OUT_NOT_6	Output	1	Discrete output.
DISC_OUT_NOT_5	Output	1	Discrete output.
DISC_OUT_NOT_4	Output	1	Discrete output.
DISC_OUT_NOT_3	Output	1	Discrete output.
DISC_OUT_NOT_2	Output	1	Discrete output.
DISC_OUT_NOT_1	Output	1	Discrete output.
DISC_OUT_NOT_0	Output	1	Discrete output.
DISC_IN_7	Input	-	Discrete input.
DISC_IN_6	Input	-	Discrete input.
DISC_IN_5	Input	-	Discrete input.
DISC_IN_4	Input	-	Discrete input.
DISC_IN_3	Input	-	Discrete input.
DISC_IN_2	Input	-	Discrete input.
DISC_IN_1	Input	-	Discrete input.
DISC_IN_0	Input	-	Discrete input.
ONE_OUT	Output	0	1-Wire data out.
ONE_IN	Input	0	1-Wire bus data in.
ONE_PWR	Output	0	Default power off.
ADC12_CS0_NOT	Output	1	Chip select for 12 bit ADC 0.
ADC12_INT0_NOT	Input	1	Operation completed interrupt.
ADC12_CS1_NOT	Output	1	Chip select for 12 bit ADC 1.
ADC12_INT1_NOT	Input	1	Operation completed interrupt.

ADC12_HBEN	Output	0	Address select for 12 bit ADCs.
ADC12_RD_NOT	Output	0	Read operation from an ADC.
ADC12_WR_NOT	Output	0	Write operation from an ADC.
ADC12_SHDN_NOT	Output	0	ADC12 low-power mode control.
MUX1_A2	Output	0	Select 16 Bit ADC Channel.
MUX1_A1	Output	0	Select 16 Bit ADC Channel.
MUX1_A0	Output	0	Select 16 Bit ADC Channel.
MUX2_A2	Output	0	Select 16 Bit ADC Channel.
MUX2_A1	Output	0	Select 16 Bit ADC Channel.
MUX2_A0	Output	0	Select 16 Bit ADC Channel.
MUX_WR_NOT	Output	1	Write to MUX1 Address register.
ADC16_CS_NOT	Output	1	Chip select for 16 Bit ADC
ADC16_DIN	Output	0	Data to 16 Bit ADC
ADC16_DOUT	Input	-	Data from 16 Bit ADC
ADC16_DCLK	Output	0	Clock to 16 Bit ADC
ADC16_BUSY	Input	-	Conversion status from 16 Bit ADCs
LOCAL_RESET_NOT	Input	0	I/O Board local reset signal.
SYS_RESET	Input	0	Backplane reset signal.
SYS_CLK	Input	-	Backplane 20 MHz clock
IODATA_OE_NOT	Input	-	Backplane Buffer Output enable.
WE0_NOT	Output	1	Buffer direction control.
RD_NOT	Input	-	Bus transfer control signal.
RDWR_NOT	Input	-	Bus transfer control signal.
CS0_NOT	Input	-	Bus transfer control signal.
IODATA_DIR	Output	-	Bus chip select. (Default is read)
IO_INT_NOT	Output	1	Transceiver data direction.
ADDR_MSBS_25	Input	-	Signal CPU of I/O board interrupt.
ADDR_MSBS_24	Input	-	Signal CPU of I/O board interrupt.
ADDR_MSBS_23	Input	-	Signal CPU of I/O board interrupt.
ADDR_MSBS_22	Input	-	Signal CPU of I/O board interrupt.
ADDR_MSBS_21	Input	-	Signal CPU of I/O board interrupt.
ADDR_MSBS_20	Input	-	Signal CPU of I/O board interrupt.
ADDR_LSBS_6	Input	-	CPU Address bus for board select.
ADDR_LSBS_5	Input	-	CPU Address bus for board select.
ADDR_LSBS_4	Input	-	CPU Address bus for board select.
ADDR_LSBS_3	Input	-	CPU Address bus for board select.
ADDR_LSBS_2	Input	-	CPU Address bus for board select.
ADDR_LSBS_1	Input	-	CPU Address bus for board select.
BP_DATA_BUS_7	In/out	Z	CPU Data bus.
BP_DATA_BUS_6	In/out	Z	CPU Data bus.
BP_DATA_BUS_5	In/out	Z	CPU Data bus.
BP_DATA_BUS_4	In/out	Z	CPU Data bus.
BP_DATA_BUS_3	In/out	Z	CPU Data bus.
BP_DATA_BUS_2	In/out	Z	CPU Data bus.
BP_DATA_BUS_1	In/out	Z	CPU Data bus.
BP_DATA_BUS_0	In/out	Z	CPU Data bus.

Appendix D

I/O Board Top Connector Pin-out

Table 8. I/O board Top Connector signals, pin directions, types, and voltages.

Signal Name	I/O Board Signal Type	Reset Value	Description
ADC16_0	In	0	16-Bit 0 to 5V Input
ADC16_1	In	0	16-Bit 0 to 5V Input
ADC16_2	In	0	16-Bit 0 to 5V Input
ADC16_3	In	0	16-Bit 0 to 5V Input
ADC16_4	In	0	16-Bit 0 to 5V Input
ADC16_5	In	0	16-Bit 0 to 5V Input
ADC16_6	In	0	16-Bit 0 to 5V Input
ADC16_7	In	0	16-Bit 0 to 5V Input
ADC16_8	In	0	16-Bit 0 to 5V Input
ADC16_9	In	0	16-Bit 0 to 5V Input
ADC16_10	In	0	16-Bit 0 to 5V Input
ADC16_11	In	0	16-Bit 0 to 5V Input
ADC16_12	In	0	16-Bit 0 to 5V Input
ADC16_13	In	0	16-Bit 0 to 5V Input
ADC16_14	In	0	16-Bit 0 to 5V Input
ADC16_15	In	0	16-Bit 0 to 5V Input
GND	Power	0	Ground
ADC12_0	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_1	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_2	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_3	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_4	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_5	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_6	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_7	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_8	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_9	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_10	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_11	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_12	In	0	12-Bit Multiple-configuration -10 to 10V

ADC12_13	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_14	In	0	12-Bit Multiple-configuration -10 to 10V
ADC12_15	In	0	12-Bit configurable ADC
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
AGND	Pwr	0	Analog Ground
DIN_0	In	1	Discrete +5V input
DIN_1	In	1	Discrete +5V input
DIN_2	In	1	Discrete +5V input
DIN_3	In	1	Discrete +5V input
DIN_4	In	1	Discrete +5V input
DIN_5	In	1	Discrete +5V input
DIN_6	In	1	Discrete +5V input
DIN_7	In	1	Discrete +5V input
DOOUT_0	Out	0	Discrete +5V output
DOOUT_1	Out	0	Discrete +5V output
DOOUT_2	Out	0	Discrete +5V output
DOOUT_3	Out	0	Discrete +5V output
DOOUT_4	Out	0	Discrete +5V output
DOOUT_5	Out	0	Discrete +5V output
DOOUT_6	Out	0	Discrete +5V output
DOOUT_7	Out	0	Discrete +5V output
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground

DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
DGND	Pwr	0	Digital Ground
TEMP_DQ	Bidirectional	1	1 Wire data
TEMP_PWR	Power	OFF (0)	1 Wire bus power
TEMP_GND	Power	GND	1 Wire bus ground

Appendix E

I/O Board Device Driver Prototypes

The device driver format shown here is similar to the style outlined by Wind River in the VxWorks Reference Manual.

---

ioBoardOpen()

NAME ioBoardOpen

SYNOPSIS STATUS ioBoardOpen (void)

DESCRIPTION Initialize static variables used by I/O board device drivers. Connect interrupts to I/O board ISR and enable I/O board interrupts. Initialize semaphores for device driver use. To prevent system failures, I/O board device driver calls will not function if this routine has not been called.

Call ioBoardOn() prior to ioBoardOpen().

RETURNS OK, if the initialization completes successfully.
ERROR, otherwise or if the I/O board is powered off.

SEE ALSO ioBoardClose, ioBoardOn, ioBoardOff

ioBoardClose()

NAME ioBoardClose

SYNOPSIS STATUS ioBoardClose (void)

DESCRIPTION This function disables I/O board interrupts. I/O board driver calls will not function after this routine has been called. Any semaphores which are pending completion will be freed.

RETURNS OK for successful operation.
ERROR, otherwise.

SEE ALSO ioBoardOn, ioBoardOpen, ioBoardOff

ioBoardOn()

NAME ioBoardOn

SYNOPSIS
```
STATUS ioBoardOn
    (
    UINT ui_msDelay      /* ms delay after power switched. */
    )
```

DESCRIPTION Turn power on to the I/O board via the ION-F backplane and delay ui_msDelay milliseconds prior to addressing the I/O board with other device driver function calls. This allows power-on transients to settle to prevent unwanted interrupts or unavailability of the I/O board by a driver call.

RETURNS OK, if the I/O board ID register can be read after the specified delay.
ERROR, otherwise.

SEE ALSO ioBoardOff, ioBoardOpen, ioBoardClose

---

ioBoardOff()

NAME ioBoardOff

SYNOPSIS
```
STATUS ioBoardOff
    (
    UINT ui_msDelay      /* ms delay after power switched. */
    )
```

DESCRIPTION Remove power from the I/O board via the ION-F backplane.

RETURNS OK, if the I/O board ID register is not available after ui_msDelay.

-2, if the I/O board ID register is available after the specified delay.

-3 if ioBoardClose() has not been called prior to this routine.

SEE ALSO ioBoardOn, ioBoardOpen, ioBoardClose

---

ioBoardSPISetAddr()

NAME ioBoardSPISetAddr

SYNOPSIS

```
STATUS ioBoardSPISetAddr
        (
        UCHAR uc_spiAddr  /* 5 bit SPI Address in bits (4:0)   */
        )
```

DESCRIPTION Set the SPI bus address on the flight computer backplane. Up to 32 SPI bus devices may be present on the backplane. The default device after ioBoardOpen() is 0x00. The SPI bus address does not change unless this routine is called.

RETURNS OK, if the value read back from the SPI bus address register matches the value written.
ERROR, otherwise.

SEE ALSO ioBoardSPIWrite

ioBoardSPIWrite()

NAME ioBoardSPIWrite

SYNOPSIS

```
STATUS ioBoardSPIWrite
    (
    UCHAR uc_MOSI     /* Master Out Slave In          */
    UINT  ms_timeOut  /* ms delay for Interrupt wait.*/
    UCHAR *uc_MISO    /* Master In Slave Out          */
    )
```

DESCRIPTION Perform a MOSI transfer from the SPI bus master to a backplane card slave device. Simultaneously perform a MISO transfer of data from an SPI bus slave to the I/O board.

RETURNS OK if an operation complete interrupt was received prior to the timeout parameter. In this case, uc_MISO, contains the value read from the slave unit while the master wrote out MOSI.

ERROR if a timeout occurred.
Also returns ERROR if the I/O board is not turned on or enabled.

SEE ALSO ioBoardSPISetAddr

ioBoardRegRead()

NAME ioBoardRegRead

SYNOPSIS

```
UCHAR ioBoardRegRead
        (
        UCHAR uc_offset      /* Offset in I/O board space */
        )
```

DESCRIPTION Read from the specified I/O board internal register address. Use this driver call to access the 8 discrete inputs on the top connector. Alternatively, one may de-reference pointers to the I/O board registers. This is recommended in the driver functions.

RETURNS The value read from the I/O board memory address requested.

SEE ALSO ioBoardRegWrite

ioBoardRegWrite()

NAME ioBoardRegWrite

SYNOPSIS

```
STATUS ioBoardRegWrite
    (
    UCHAR uc_offset     /* Offset in I/O board space */
    UCHAR uc_data       /* Data to write to I/O board */
    )
```

DESCRIPTION Write data to the specified I/O board internal address.

RETURNS OK if the I/O board is turned on and drivers are enabled.
Also returns ERROR if the I/O board is not turned on or enabled.

SEE ALSO ioBoardRegRead

ioBoard1WireReset()

NAME ioBoard1WireReset

SYNOPSIS

```
STATUS ioBoard1WireReset
    (
    USHORT us_timeout /* Timeout in VxWorks ticks  */
    )
```

DESCRIPTION Drive DS_DQ low for 300us then drive it high (+5V), resetting all devices on the bus.

Because this is a semaphore-driven operation, the I/O board must be open and a timeout is necessary in case the hardware does not respond.

RETURNS OK, if at least one slave responded with a presence pulse.
ERROR, if the I/O board fails to complete the reset within the specified timeout.
Also returns ERROR if the I/O board is not turned on or enabled.

SEE ALSO ioBoard1WireRead, ioBoard1WireWrite

ioBoard1WireWrite()

NAME ioBoard1WireWrite

SYNOPSIS

```
STATUS ioBoard1WireWrite
    (
    UCHAR uc_data       /* Byte to write to a 1-wire device */
    UCHAR us_timeout    /* Timeout in VxWorks ticks         */
    )
```

DESCRIPTION This operation allows the user to write 8-bits of data at a time to the one-wire bus.

If data integrity is critical, the user should read the entire memory of a device and calculate the 8-bit CRC to check for communications errors.

Because this is a semaphore-driven operation, the I/O board must be open and a timeout is necessary in case the hardware does not respond.

RETURNS OK, if the function did not time out.
ERROR, otherwise.

SEE ALSO ioBoard1WireReset, ioBoard1WireRead

ioBoard1WireRead()

NAME ioBoard1WireRead

SYNOPSIS

```
STATUS ioBoard1WireRead
    (
    UCHAR *uc_data      /* Data read from 1-wire device */
    UCHAR us_timeout    /* Timeout in VxWorks ticks     */
    )
```

DESCRIPTION This function returns the data read from a specific one-wire device, which has been addressed using previous write ROM setup commands per the one-wire device protocol.

If data integrity is critical, the user should read the entire memory of a device and calculate the 8-bit CRC to check for communications errors.

Because this is a semaphore-driven operation, the I/O board must be open and a timeout is necessary in case the hardware does not respond.

RETURNS OK, if the function did not time out.
ERROR, otherwise.

SEE ALSO ioBoard1WireReset, ioBoard1WireWrite, ioBoard1WireArbitrate

ioBoard12bitADC()

NAME ioBoard12bitADC

SYNOPSIS

```
STATUS ioBoard12BitADC
    (
    UCHAR uc_channel   /* 12-bit ADC channel to sample */
    UCHAR uc_mode      /* Conversion mode              */
    USHORT *us_data    /* Conversion result            */
    UCHAR us_timeout   /* Timeout in VxWorks ticks     */
    )
```

DESCRIPTION This function returns an analog input sample data read from one of 16 channels using the specified mode.

uc_channel ranges from 0..15.

uc_mode typdefs:
MODE0: 0-5V conversion
MODE1: +/-5V conversion
MODE2: 0-10V conversion
MODE3: +/-10V conversion

Because this is a semaphore-driven operation, the I/O board must be open and a timeout is necessary in case the hardware does not respond.

RETURNS OK, if the conversion completes successfully. In this case, *us_data contains the conversion result in bits 11:0.

ERROR, if the operation times out.
Also returns ERROR if the I/O board is not turned on or enabled.

SEE ALSO ioBoard16BitADC

ioBoard16bitADC()

NAME ioBoard16bitADC

SYNOPSIS

```
STATUS ioBoard16BitADC
        (
        UCHAR uc_channel   /* 16-bit ADC channel to sample */
        USHORT *us_data    /* Conversion result            */
        UCHAR us_timeout   /* Timeout in VxWorks ticks     */
        )
```

DESCRIPTION This function returns an analog input sample data read from one of 16 channels using the specified mode. All data is returned in 16-bit precision.

uc_channel ranges from 0..15.

Because this is a semaphore-driven operation, the I/O board must be open and a timeout is necessary in case the hardware does not respond.

RETURNS OK, if the function did not time out. In this case, *us_data contains the conversion result in bits 15:0.

ERROR, if the operation times out.
Also returns ERROR if the I/O board is not turned on or enabled.

SEE ALSO ioBoard12BitADC

Appendix F

I/O Board Top Level Structural VHDL Model

The following is the I/O board top-level VHDL code listing.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;
--------------------------------------------------------------------------
---
-- TOP-LEVEL ENTITY for entire IO board FPGA.
--
-- This is the top level IO_BOARD entity.
--
-- bdb   7-1-2001
--------------------------------------------------------------------------
---
entity IOBOARD_TOP is
    port
    (
      SPI_ADDR           : out STD_LOGIC_VECTOR(4 downto 0);
      SPI_SS_NOT         : out STD_LOGIC;
      SPI_MISO           : in  STD_LOGIC;
      SPI_MOSI           : out STD_LOGIC;
      SPI_CLK            : out STD_LOGIC;

      DISC_OUT_NOT       : out STD_LOGIC_VECTOR(7 downto 0);
      DISC_IN            : in  STD_LOGIC_VECTOR(7 downto 0);

      ONE_IN             : in  STD_LOGIC;
      ONE_OUT            : out STD_LOGIC;
      ONE_PWR            : out STD_LOGIC;

      ADC12_CS0_NOT      : out STD_LOGIC;
      ADC12_INT0_NOT     : in  STD_LOGIC;
      ADC12_CS1_NOT      : out STD_LOGIC;
      ADC12_INT1_NOT     : in  STD_LOGIC;
      ADC12_HBEN         : out STD_LOGIC;
      ADC12_RD_NOT       : out STD_LOGIC;
      ADC12_WR_NOT       : out STD_LOGIC;
      ADC12_SHDN_NOT     : out STD_LOGIC;

      MUX1_A2            : out STD_LOGIC;
      MUX1_A1            : out STD_LOGIC;
      MUX1_A0            : out STD_LOGIC;
```

```
      MUX2_A2            : out STD_LOGIC;
      MUX2_A1            : out STD_LOGIC;
      MUX2_A0            : out STD_LOGIC;
      MUX_WR_NOT         : out STD_LOGIC;
      ADC16_CS_NOT       : out STD_LOGIC;
      ADC16_DIN          : out STD_LOGIC;
      ADC16_DOUT         : in  STD_LOGIC;
      ADC16_DCLK         : out STD_LOGIC;
      ADC16_BUSY         : in  STD_LOGIC;

      LOCAL_RESET_NOT    : in  STD_LOGIC;
      SYS_RESET          : in  STD_LOGIC;
      SYS_CLK            : in  STD_LOGIC;
      IODATA_OE_NOT      : out STD_LOGIC;
      WE0_NOT            : in  STD_LOGIC;
      RD_NOT             : in  STD_LOGIC;
      RDWR_NOT           : in  STD_LOGIC;
      CS0_NOT            : in  STD_LOGIC;
      IODATA_DIR         : out STD_LOGIC;
      IO_INT_NOT         : out STD_LOGIC;
      ADDR_MSBS          : in  STD_LOGIC_VECTOR(25 downto 20);
      ADDR_LSBS          : in  STD_LOGIC_VECTOR(6 downto 1);
      BP_DATA_BUS        : inout STD_LOGIC_VECTOR(7 downto 0)
   );
end IOBOARD_TOP;

architecture IOBOARD_TOP_ARCH of IOBOARD_TOP is


component GLUE_LOGIC_TOP
   port
   (
      RESET              : out STD_LOGIC;
      LOCAL_RESET_NOT    : in  STD_LOGIC;
      SYS_RESET          : in  STD_LOGIC
   );
end component;


component DECODER_TOP
   port
   (
      IODATA_OE_NOT      : out STD_LOGIC;
      WE0_NOT            : in  STD_LOGIC;
      RD_NOT             : in  STD_LOGIC;
      RDWR_NOT           : in  STD_LOGIC;
      CS0_NOT            : in  STD_LOGIC;
      IODATA_DIR         : out STD_LOGIC;
      ADDR_MSBS          : in  STD_LOGIC_VECTOR(25 downto 20);
      ADDR_LSBS          : in  STD_LOGIC_VECTOR(6 downto 1);
      BP_DATA_BUS        : inout STD_LOGIC_VECTOR(7 downto 0);
      READ_DATA_BUS      : in  STD_LOGIC_VECTOR(7 downto 0);
      WRITE_DATA_BUS     : out STD_LOGIC_VECTOR(7 downto 0);
      WR_CONFIG_REG      : out STD_LOGIC;
      WR_CONTROL_REG     : out STD_LOGIC;
      WR_STATUS_REG      : out STD_LOGIC;
      WR_ERROR_REG       : out STD_LOGIC;
```

```
      WR_SPI_ADDR_REG    : out STD_LOGIC;
      WR_SPI_DOUT_REG    : out STD_LOGIC;
      WR_SPI_SCALE_REG   : out STD_LOGIC;
      WR_DISC_OUT_REG    : out STD_LOGIC;
      WR_ADC16_CTRL_REG  : out STD_LOGIC;
      WR_ADC16_SEL_REG   : out STD_LOGIC;
      WR_ONE_CONFIG_REG  : out STD_LOGIC;
      WR_ONE_DELAY_REG   : out STD_LOGIC;
      WR_ONE_STATUS_REG  : out STD_LOGIC;
      WR_ONE_DOUT_REG    : out STD_LOGIC;
      WR_ONE_SRCH_O_REG  : out STD_LOGIC;
      ADC12_WR_NOT       : out STD_LOGIC;
      ADC12_CS0_NOT      : out STD_LOGIC;
      ADC12_CS1_NOT      : out STD_LOGIC;
      ADC12_HBEN         : out STD_LOGIC;
      ADC12_RD_NOT       : out STD_LOGIC
   );
end component;


component ADC16_TOP
   port
   (
      RESET              : in  STD_LOGIC;
      CLK                : in  STD_LOGIC;

      ADC16_CS_NOT       : out STD_LOGIC;
      ADC16_DIN          : out STD_LOGIC;
      ADC16_DOUT         : in  STD_LOGIC;
      ADC16_DCLK         : out STD_LOGIC;
      ADC16_BUSY         : in  STD_LOGIC;

      MUX1_A2            : out STD_LOGIC;
      MUX1_A1            : out STD_LOGIC;
      MUX1_A0            : out STD_LOGIC;
      MUX2_A2            : out STD_LOGIC;
      MUX2_A1            : out STD_LOGIC;
      MUX2_A0            : out STD_LOGIC;
      MUX_WR_NOT         : out STD_LOGIC;

      SET_ADC16_ERROR_BIT: out STD_LOGIC;
      DATA_BUS           : in  STD_LOGIC_VECTOR(7 downto 0);
      SET_ADC16_DONE_BIT : out STD_LOGIC;
      ADC16_START_BIT    : in  STD_LOGIC;
      WR_ADC16_CTRL_REG  : in  STD_LOGIC;
      WR_ADC16_SEL_REG   : in  STD_LOGIC;
      ADC16_CTRL_REG     : out STD_LOGIC_VECTOR(7 downto 0);
      ADC16_DATA_L_REG   : out STD_LOGIC_VECTOR(7 downto 0);
      ADC16_DATA_H_REG   : out STD_LOGIC_VECTOR(7 downto 0);
      ADC16_SEL_REG      : out STD_LOGIC_VECTOR(5 downto 0)
    );
end component;
```

```
component ONE_WIRE_TOP
   port
   (
      RESET              : in  STD_LOGIC;
      CLK                : in  STD_LOGIC;
      ONE_IN             : in  STD_LOGIC;
      ONE_OUT            : out STD_LOGIC;
      ONE_START_BIT      : in  STD_LOGIC;
      SET_ONE_DONE_BIT   : out STD_LOGIC;
      SET_ONE_ERROR_BIT  : out STD_LOGIC;
      DATA_BUS           : in  STD_LOGIC_VECTOR (7 downto 0);
      WR_ONE_CONFIG_REG  : in  STD_LOGIC;
      ONE_CONFIG_REG     : out STD_LOGIC_VECTOR (1 downto 0);
      WR_ONE_DELAY_REG   : in  STD_LOGIC;
      ONE_DELAY_REG      : out STD_LOGIC_VECTOR (7 downto 0);
      WR_ONE_STATUS_REG  : in  STD_LOGIC;
      ONE_STATUS_REG     : out STD_LOGIC;
      WR_ONE_DOUT_REG    : in  STD_LOGIC;
      ONE_DOUT_REG       : out STD_LOGIC_VECTOR (7 downto 0);
      WR_ONE_DIN_REG     : in  STD_LOGIC;
      ONE_DIN_REG        : out STD_LOGIC_VECTOR (7 downto 0);
      WR_ONE_SRCH_O_REG  : in  STD_LOGIC;
      ONE_SRCH_O_REG     : out STD_LOGIC_VECTOR (7 downto 0);
      ONE_SRCH_I_NON_REG : out STD_LOGIC_VECTOR (7 downto 0);
      ONE_SRCH_I_INV_REG : out STD_LOGIC_VECTOR (7 downto 0)
   );
end component;

component CONTROL_TOP
   port (
      RESET              : in  std_logic;
      CLK                : in  std_logic;
      DATA_BUS           : in  std_logic_vector(7 downto 0);
      WR_CONFIG_REG      : in  std_logic;
      WR_CONTROL_REG     : in  std_logic;
      WR_STATUS_REG      : in  std_logic;
      WR_ERROR_REG       : in  std_logic;
      CONFIG_REG         : out std_logic_vector(2 downto 0);
      CONTROL_REG        : out std_logic_vector(2 downto 0);
      STATUS_REG         : out std_logic_vector(3 downto 0);
      ERROR_REG          : out std_logic_vector(3 downto 0);
      SET_SPI_ERROR_BIT  : in  std_logic;
      SET_ADC16_ERROR_BIT: in  std_logic;
      SET_ONE_ERROR_BIT  : in  std_logic;
      ONE_PWR            : out std_logic;
      IO_INT_NOT         : out STD_LOGIC;
      ADC12_SHDN_NOT     : out std_logic;
      ADC12_INT0_NOT     : in  STD_LOGIC;
      ADC12_INT1_NOT     : in  STD_LOGIC;
      ONE_START_BIT      : out std_logic;
      SET_ONE_DONE_BIT   : in  std_logic;
      ADC16_START_BIT    : out std_logic;
      SET_ADC16_DONE_BIT : in  std_logic;
      SPI_START_BIT      : out std_logic;
      SET_SPI_DONE_BIT   : in  std_logic
   );
end component;
```

```
component DISC_IN_TOP
    port
    (
      RESET              : in  STD_LOGIC;
      CLK                : in  STD_LOGIC;
      DISC_IN            : in  STD_LOGIC_VECTOR(7 downto 0);
      DISC_IN_REG        : out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;


component DISC_OUT_TOP
    port
    (
       RESET             : in STD_LOGIC;
       CLK               : in STD_LOGIC;
       DATA_BUS          : in STD_LOGIC_VECTOR(7 downto 0);
       WR_DISC_OUT_REG   : in STD_LOGIC;
       DISC_OUT_REG_NOT  : out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;


component READ_MUX_TOP
    port
    (
      ADDR_LSBS          : in  STD_LOGIC_VECTOR(6 downto 1);
      READ_DATA_BUS      : out STD_LOGIC_VECTOR(7 downto 0);
      DEVICE_ID_L_REG    : in  STD_LOGIC_VECTOR(7 downto 0);
      DEVICE_ID_H_REG    : in  STD_LOGIC_VECTOR(7 downto 0);
      CONFIG_REG         : in  STD_LOGIC_VECTOR(2 downto 0);
      CONTROL_REG        : in  STD_LOGIC_VECTOR(2 downto 0);
      STATUS_REG         : in  STD_LOGIC_VECTOR(3 downto 0);
      ERROR_REG          : in  STD_LOGIC_VECTOR(3 downto 0);
      SPI_ADDR_REG       : in  STD_LOGIC_VECTOR(4 downto 0);
      SPI_DOUT_REG       : in  STD_LOGIC_VECTOR(7 downto 0);
      SPI_DIN_REG        : in  STD_LOGIC_VECTOR(7 downto 0);
      SPI_SCALE_REG      : in  STD_LOGIC_VECTOR(3 downto 0);
      DISC_OUT_REG_NOT   : in  STD_LOGIC_VECTOR(7 downto 0);
      DISC_IN_REG        : in  STD_LOGIC_VECTOR(7 downto 0);
      ADC16_CTRL_REG     : in  STD_LOGIC_VECTOR(7 downto 0);
      ADC16_DATA_L_REG   : in  STD_LOGIC_VECTOR(7 downto 0);
      ADC16_DATA_H_REG   : in  STD_LOGIC_VECTOR(7 downto 0);
      ADC16_SEL_REG      : in  STD_LOGIC_VECTOR(5 downto 0);
      ONE_CONFIG_REG     : in  STD_LOGIC_VECTOR(1 downto 0);
      ONE_DELAY_REG      : in  STD_LOGIC_VECTOR(7 downto 0);
      ONE_STATUS_REG     : in  STD_LOGIC;
      ONE_DOUT_REG       : in  STD_LOGIC_VECTOR(7 downto 0);
      ONE_DIN_REG        : in  STD_LOGIC_VECTOR(7 downto 0);
      ONE_SRCH_O_REG     : in  STD_LOGIC_VECTOR(7 downto 0);
      ONE_SRCH_I_INV_REG : in  STD_LOGIC_VECTOR(7 downto 0);
      ONE_SRCH_I_NON_REG : in  STD_LOGIC_VECTOR(7 downto 0)
    );
end component;
```

```
component SPI_TOP
   port
   (
      RESET              : in  STD_LOGIC;
      CLK                : in  STD_LOGIC;
      SPI_START_BIT      : in  STD_LOGIC;
      SET_SPI_DONE_BIT   : out STD_LOGIC;
      SET_SPI_ERROR_BIT  : out STD_LOGIC;
      SPI_DIN_REG        : out STD_LOGIC_VECTOR(7 downto 0);
      DATA_BUS           : in  STD_LOGIC_VECTOR(7 downto 0);
      WR_SPI_DOUT_REG    : in  STD_LOGIC;
      SPI_DOUT_REG       : out STD_LOGIC_VECTOR(7 downto 0);
      WR_SPI_ADDR_REG    : in  STD_LOGIC;
      SPI_ADDR_REG       : out STD_LOGIC_VECTOR(4 downto 0);
      WR_SPI_SCALE_REG   : in  STD_LOGIC;
      SPI_SCALE_REG      : out STD_LOGIC_VECTOR(3 downto 0);
      SPI_MISO           : in  STD_LOGIC;
      SPI_SS_NOT         : out STD_LOGIC;
      SPI_MOSI           : out STD_LOGIC;
      SPI_CLK            : out STD_LOGIC
   );
end component;


-------------------------------------------------------------------------
-- Signals are defined for internal interconnects.
-- A new signal is not required for top-level I/O.
-------------------------------------------------------------------------
   signal DEVICE_ID_L_REG : STD_LOGIC_VECTOR(7 downto 0);
   signal DEVICE_ID_H_REG : STD_LOGIC_VECTOR(7 downto 0);
   signal RESET           : STD_LOGIC;
   ------------------------------
   -- For DECODER_TOP.
   ------------------------------
   signal WR_CONFIG_REG       : STD_LOGIC;
   signal WR_CONTROL_REG      : STD_LOGIC;
   signal WR_STATUS_REG       : STD_LOGIC;
   signal WR_ERROR_REG        : STD_LOGIC;
   signal WR_SPI_ADDR_REG     : STD_LOGIC;
   signal WR_SPI_DOUT_REG     : STD_LOGIC;
   signal WR_SPI_SCALE_REG    : STD_LOGIC;
   signal WR_DISC_OUT_REG     : STD_LOGIC;
   signal WR_ADC16_CTRL_REG   : STD_LOGIC;
   signal WR_ADC16_SEL_REG    : STD_LOGIC;
   signal WR_ONE_CONFIG_REG   : STD_LOGIC;
   signal WR_ONE_DELAY_REG    : STD_LOGIC;
   signal WR_ONE_STATUS_REG   : STD_LOGIC;
   signal WR_ONE_DOUT_REG     : STD_LOGIC;
   signal WR_ONE_SRCH_O_REG   : STD_LOGIC;
   ------------------------------
   -- For ADC16_TOP.
   ------------------------------
   signal SET_ADC16_ERROR_BIT : STD_LOGIC;
   signal DATA_BUS            : STD_LOGIC_VECTOR(7 downto 0);
   signal SET_ADC16_DONE_BIT  : STD_LOGIC;
   signal ADC16_START_BIT     : STD_LOGIC;
   signal ADC16_CTRL_REG      : STD_LOGIC_VECTOR(7 downto 0);
```

```
    signal ADC16_DATA_L_REG    : STD_LOGIC_VECTOR(7 downto 0);
    signal ADC16_DATA_H_REG    : STD_LOGIC_VECTOR(7 downto 0);
    signal ADC16_SEL_REG       : STD_LOGIC_VECTOR(5 downto 0);
    ------------------------------
    -- For ONE_WIRE_TOP.
    ------------------------------
    signal ONE_START_BIT       : STD_LOGIC;
    signal SET_ONE_DONE_BIT    : STD_LOGIC;
    signal SET_ONE_ERROR_BIT   : STD_LOGIC;
    signal ONE_CONFIG_REG      : STD_LOGIC_VECTOR(1 downto 0);
    signal ONE_DELAY_REG       : STD_LOGIC_VECTOR(7 downto 0);
    signal ONE_STATUS_REG      : STD_LOGIC;
    signal ONE_DOUT_REG        : STD_LOGIC_VECTOR(7 downto 0);
    signal WR_ONE_DIN_REG      : STD_LOGIC;
    signal ONE_DIN_REG         : STD_LOGIC_VECTOR(7 downto 0);
    signal ONE_SRCH_O_REG      : STD_LOGIC_VECTOR(7 downto 0);
    signal ONE_SRCH_I_NON_REG  : STD_LOGIC_VECTOR(7 downto 0);
    signal ONE_SRCH_I_INV_REG  : STD_LOGIC_VECTOR(7 downto 0);
    ------------------------------
    -- For CONTROL_TOP.
    ------------------------------
    signal CONFIG_REG          : STD_LOGIC_VECTOR(2 downto 0);
    signal CONTROL_REG         : STD_LOGIC_VECTOR(2 downto 0);
    signal STATUS_REG          : STD_LOGIC_VECTOR(3 downto 0);
    signal ERROR_REG           : STD_LOGIC_VECTOR(3 downto 0);
    signal SET_SPI_ERROR_BIT   : STD_LOGIC;
    signal SPI_START_BIT       : STD_LOGIC;
    signal SET_SPI_DONE_BIT    : STD_LOGIC;
    ------------------------------
    -- For DISC_IN_TOP.
    ------------------------------
    signal DISC_IN_REG         : STD_LOGIC_VECTOR(7 downto 0);
    ------------------------------
    -- For SPI_TOP.
    ------------------------------
    signal SPI_DIN_REG         : STD_LOGIC_VECTOR(7 downto 0);
    signal SPI_DOUT_REG        : STD_LOGIC_VECTOR(7 downto 0);
    signal SPI_SCALE_REG       : STD_LOGIC_VECTOR(3 downto 0);
    ------------------------------
    -- For READ_MUX_TOP.
    ------------------------------
    signal READ_DATA_BUS      : STD_LOGIC_VECTOR(7 downto 0);
    signal DISC_OUT_REG_NOT   : STD_LOGIC_VECTOR(7 downto 0);
    signal SPI_ADDR_REG       : STD_LOGIC_VECTOR(4 downto 0);

begin

-- Modify DEVICE_ID_L_REG to 0xB1, 0xB2 if the rev changes.
    DEVICE_ID_L_REG <= "00010000";  -- 0x10
    DEVICE_ID_H_REG <= "10110000";  -- 0xB0
```

```
GLUE_LOGIC_TOP_0 : GLUE_LOGIC_TOP
   port map
   (
      RESET              => RESET,
      LOCAL_RESET_NOT    => LOCAL_RESET_NOT,
      SYS_RESET          => SYS_RESET
   );


DECODER_TOP_0 : DECODER_TOP
   port map
   (
      IODATA_OE_NOT      => IODATA_OE_NOT,        -- External signals.
      WE0_NOT            => WE0_NOT,
      RD_NOT             => RD_NOT,
      RDWR_NOT           => RDWR_NOT,
      CS0_NOT            => CS0_NOT,
      IODATA_DIR         => IODATA_DIR,
      ADDR_MSBS          => ADDR_MSBS,
      ADDR_LSBS          => ADDR_LSBS,
      BP_DATA_BUS        => BP_DATA_BUS,
      ADC12_WR_NOT       => ADC12_WR_NOT,
      ADC12_CS0_NOT      => ADC12_CS0_NOT,
      ADC12_CS1_NOT      => ADC12_CS1_NOT,
      ADC12_HBEN         => ADC12_HBEN,
      ADC12_RD_NOT       => ADC12_RD_NOT,

      WR_CONFIG_REG      => WR_CONFIG_REG,        -- Internal signals.
      WR_CONTROL_REG     => WR_CONTROL_REG,
      WR_STATUS_REG      => WR_STATUS_REG,
      WR_ERROR_REG       => WR_ERROR_REG,
      WR_SPI_ADDR_REG    => WR_SPI_ADDR_REG,
      WR_SPI_DOUT_REG    => WR_SPI_DOUT_REG,
      WR_SPI_SCALE_REG   => WR_SPI_SCALE_REG,
      WR_DISC_OUT_REG    => WR_DISC_OUT_REG,
      WR_ADC16_CTRL_REG  => WR_ADC16_CTRL_REG,
      WR_ADC16_SEL_REG   => WR_ADC16_SEL_REG,
      WR_ONE_CONFIG_REG  => WR_ONE_CONFIG_REG,
      WR_ONE_DELAY_REG   => WR_ONE_DELAY_REG,
      WR_ONE_STATUS_REG  => WR_ONE_STATUS_REG,
      WR_ONE_DOUT_REG    => WR_ONE_DOUT_REG,
      WR_ONE_SRCH_O_REG  => WR_ONE_SRCH_O_REG,
      READ_DATA_BUS      => READ_DATA_BUS,
      WRITE_DATA_BUS     => DATA_BUS
   );


ADC16_TOP_0 : ADC16_TOP
   port map
   (
      RESET              => RESET,
      CLK                => SYS_CLK,              -- External signals.
      MUX_WR_NOT         => MUX_WR_NOT,
      MUX1_A2            => MUX1_A2,
      MUX1_A1            => MUX1_A1,
      MUX1_A0            => MUX1_A0,
      MUX2_A2            => MUX2_A2,
```

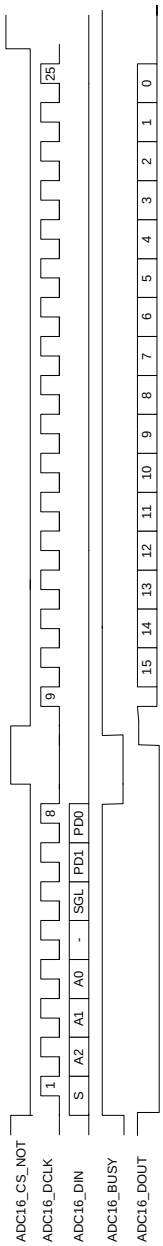
```
      MUX2_A1            => MUX2_A1,
      MUX2_A0            => MUX2_A0,
      ADC16_CS_NOT       => ADC16_CS_NOT,
      ADC16_DIN          => ADC16_DIN,
      ADC16_DOUT         => ADC16_DOUT,
      ADC16_DCLK         => ADC16_DCLK,
      ADC16_BUSY         => ADC16_BUSY,

      SET_ADC16_ERROR_BIT=> SET_ADC16_ERROR_BIT,  -- Internal signals.
      DATA_BUS           => DATA_BUS,
      SET_ADC16_DONE_BIT => SET_ADC16_DONE_BIT,
      ADC16_START_BIT    => ADC16_START_BIT,
      WR_ADC16_CTRL_REG  => WR_ADC16_CTRL_REG,
      WR_ADC16_SEL_REG   => WR_ADC16_SEL_REG,
      ADC16_CTRL_REG     => ADC16_CTRL_REG,
      ADC16_DATA_L_REG   => ADC16_DATA_L_REG,
      ADC16_DATA_H_REG   => ADC16_DATA_H_REG,
      ADC16_SEL_REG      => ADC16_SEL_REG
    );


ONE_WIRE_TOP_0 : ONE_WIRE_TOP
   port map
   (

      CLK                => SYS_CLK,              -- External signals.
      ONE_IN             => ONE_IN,
      ONE_OUT            => ONE_OUT,

      RESET              => RESET,                -- Internal signals.
      ONE_START_BIT      => ONE_START_BIT,
      SET_ONE_DONE_BIT   => SET_ONE_DONE_BIT,
      SET_ONE_ERROR_BIT  => SET_ONE_ERROR_BIT,
      DATA_BUS           => DATA_BUS,
      WR_ONE_CONFIG_REG  => WR_ONE_CONFIG_REG,
      ONE_CONFIG_REG     => ONE_CONFIG_REG,
      WR_ONE_DELAY_REG   => WR_ONE_DELAY_REG,
      ONE_DELAY_REG      => ONE_DELAY_REG,
      WR_ONE_STATUS_REG  => WR_ONE_STATUS_REG,
      ONE_STATUS_REG     => ONE_STATUS_REG,
      WR_ONE_DOUT_REG    => WR_ONE_DOUT_REG,
      ONE_DOUT_REG       => ONE_DOUT_REG,
      WR_ONE_DIN_REG     => WR_ONE_DIN_REG,
      ONE_DIN_REG        => ONE_DIN_REG,
      WR_ONE_SRCH_O_REG  => WR_ONE_SRCH_O_REG,
      ONE_SRCH_O_REG     => ONE_SRCH_O_REG,
      ONE_SRCH_I_NON_REG    => ONE_SRCH_I_NON_REG,
      ONE_SRCH_I_INV_REG    => ONE_SRCH_I_INV_REG
   );
```

```
CONTROL_TOP_0 : CONTROL_TOP
   port map

   (
      RESET              => RESET,                -- External signals.
      CLK                => SYS_CLK,
      ADC12_SHDN_NOT     => ADC12_SHDN_NOT,
      ADC12_INT0_NOT     => ADC12_INT0_NOT,
      ADC12_INT1_NOT     => ADC12_INT1_NOT,
      ONE_PWR            => ONE_PWR,
      IO_INT_NOT         => IO_INT_NOT,

      DATA_BUS           => DATA_BUS,             -- Internal signals.
      WR_CONFIG_REG      => WR_CONFIG_REG,
      WR_CONTROL_REG     => WR_CONTROL_REG,
      WR_STATUS_REG      => WR_STATUS_REG,
      WR_ERROR_REG       => WR_ERROR_REG,
      CONFIG_REG         => CONFIG_REG,
      CONTROL_REG        => CONTROL_REG,
      STATUS_REG         => STATUS_REG,
      ERROR_REG          => ERROR_REG,
      SET_SPI_ERROR_BIT  => SET_SPI_ERROR_BIT,
      SET_ADC16_ERROR_BIT=> SET_ADC16_ERROR_BIT,
      SET_ONE_ERROR_BIT  => SET_ONE_ERROR_BIT,
      ONE_START_BIT      => ONE_START_BIT,
      SET_ONE_DONE_BIT   => SET_ONE_DONE_BIT,
      ADC16_START_BIT    => ADC16_START_BIT,
      SET_ADC16_DONE_BIT => SET_ADC16_DONE_BIT,
      SPI_START_BIT      => SPI_START_BIT,
      SET_SPI_DONE_BIT   => SET_SPI_DONE_BIT
   );


DISC_IN_TOP_0 : DISC_IN_TOP
   port map
    (
      CLK                => SYS_CLK,              -- External signals.
      DISC_IN            => DISC_IN,

      RESET              => RESET,                -- Internal signals.
      DISC_IN_REG        => DISC_IN_REG
    );


DISC_OUT_TOP_0 : DISC_OUT_TOP
   port map
    (
      RESET              => RESET,                -- External signals.
      CLK                => SYS_CLK,
      DISC_OUT_REG_NOT   => DISC_OUT_REG_NOT,

      DATA_BUS           => DATA_BUS,             -- Internal signals.
      WR_DISC_OUT_REG    => WR_DISC_OUT_REG
    );
```

```
SPI_TOP_0 : SPI_TOP
   port map
   (
      RESET              => RESET,                -- External signals.
      CLK                => SYS_CLK,
      SPI_ADDR_REG       => SPI_ADDR_REG,
      SPI_MISO           => SPI_MISO,
      SPI_SS_NOT         => SPI_SS_NOT,
      SPI_MOSI           => SPI_MOSI,
      SPI_CLK            => SPI_CLK,

      SPI_START_BIT      => SPI_START_BIT,        -- Internal signals.
      SET_SPI_DONE_BIT   => SET_SPI_DONE_BIT,
      SET_SPI_ERROR_BIT  => SET_SPI_ERROR_BIT,
      SPI_DIN_REG        => SPI_DIN_REG,
      DATA_BUS           => DATA_BUS,
      WR_SPI_DOUT_REG    => WR_SPI_DOUT_REG,
      SPI_DOUT_REG       => SPI_DOUT_REG,
      WR_SPI_ADDR_REG    => WR_SPI_ADDR_REG,
      WR_SPI_SCALE_REG   => WR_SPI_SCALE_REG,
      SPI_SCALE_REG      => SPI_SCALE_REG
   );


READ_MUX_TOP_0 : READ_MUX_TOP
   port map
    (
      ADDR_LSBS          => ADDR_LSBS,
      READ_DATA_BUS      => READ_DATA_BUS,
      DEVICE_ID_L_REG    => DEVICE_ID_L_REG,
      DEVICE_ID_H_REG    => DEVICE_ID_H_REG,
      CONFIG_REG         => CONFIG_REG,
      CONTROL_REG        => CONTROL_REG,
      STATUS_REG         => STATUS_REG,
      ERROR_REG          => ERROR_REG,
      SPI_ADDR_REG       => SPI_ADDR_REG,
      SPI_DOUT_REG       => SPI_DOUT_REG,
      SPI_DIN_REG        => SPI_DIN_REG,
      SPI_SCALE_REG      => SPI_SCALE_REG,
      DISC_OUT_REG_NOT   => DISC_OUT_REG_NOT,
      DISC_IN_REG        => DISC_IN_REG,
      ADC16_CTRL_REG     => ADC16_CTRL_REG,
      ADC16_DATA_L_REG   => ADC16_DATA_L_REG,
      ADC16_DATA_H_REG   => ADC16_DATA_H_REG,
      ADC16_SEL_REG      => ADC16_SEL_REG,
      ONE_CONFIG_REG     => ONE_CONFIG_REG,
      ONE_DELAY_REG      => ONE_DELAY_REG,
      ONE_STATUS_REG     => ONE_STATUS_REG,
      ONE_DOUT_REG       => ONE_DOUT_REG,
      ONE_DIN_REG        => ONE_DIN_REG,
      ONE_SRCH_O_REG     => ONE_SRCH_O_REG,
      ONE_SRCH_I_NON_REG => ONE_SRCH_I_NON_REG,
      ONE_SRCH_I_INV_REG => ONE_SRCH_I_INV_REG);

   DISC_OUT_NOT <= DISC_OUT_REG_NOT;
   SPI_ADDR     <= SPI_ADDR_REG;
end IOBOARD_TOP_ARCH;
```

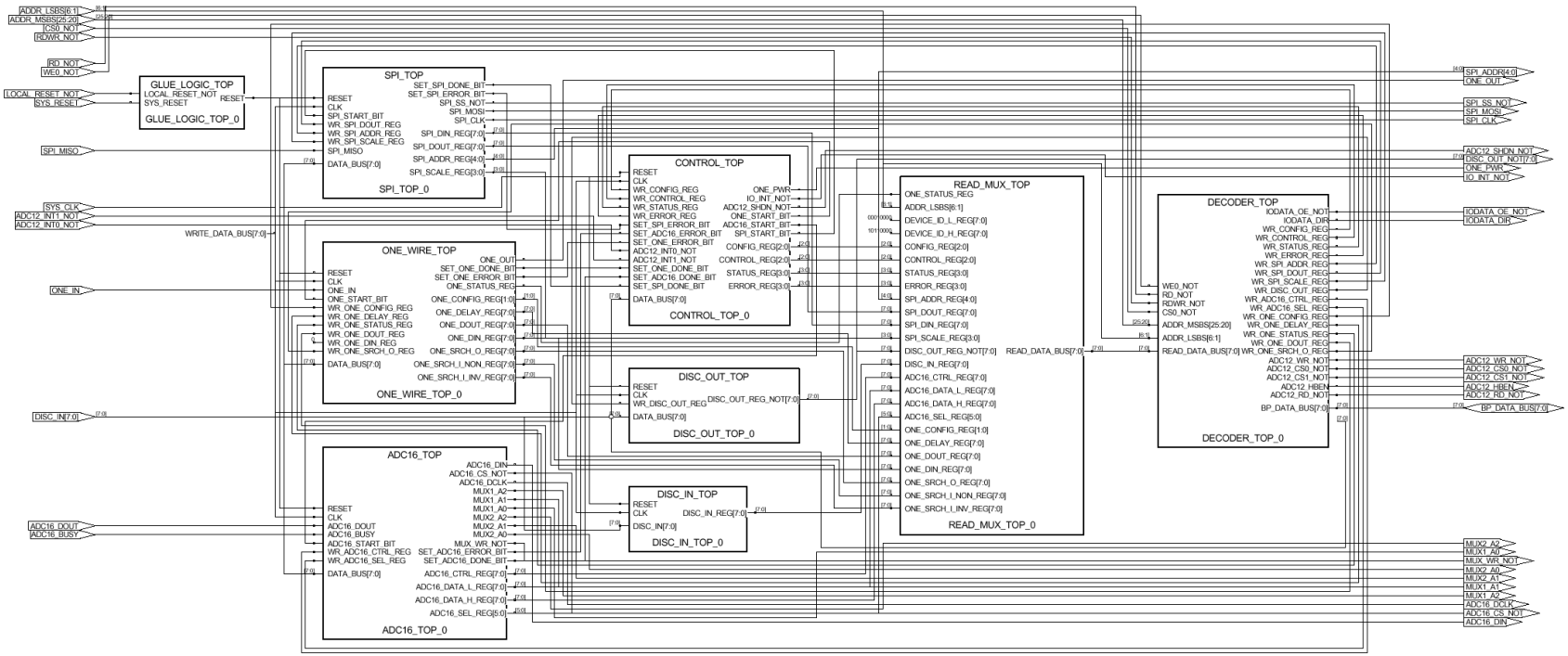
Appendix G

ADS834416-Bit A/D Converter Protocol.



Appendix H

I/O Board FPGA Top Level Structural Diagram.


ADDR_LSBS[6:1]
ADDR_MSBS[25:20]
CS0_NOT
RDWR_NOT
RD_NOT
WE0_NOT
LOCAL_RESET_NOT
SYS_RESET
GLUE_LOGIC_TOP
GLUE_LOGIC_TOP_0
SPI_MISO
SYS_CLK
ADC12_INT1_NOT
ADC12_INT0_NOT
WRITE_DATA_BUS[7:0]
ONE_IN
DISC_IN[7:0]
ADC16_DOUT
ADC16_BUSY
SPI_TOP
SPI_TOP_0
ONE_WIRE_TOP
ONE_WIRE_TOP_0
ADC16_TOP
ADC16_TOP_0
CONTROL_TOP
CONTROL_TOP_0
DISC_OUT_TOP
DISC_OUT_TOP_0
DISC_IN_TOP
DISC_IN_TOP_0
READ_MUX_TOP
READ_MUX_TOP_0
DECODER_TOP
DECODER_TOP_0
SPI_ADDR[4:0]
ONE_OUT
SPI_SS_NOT
SPI_MOSI
SPI_CLK
ADC12_SHDN_NOT
DISC_OUT_NOT[7:0]
ONE_PWR
IO_INT_NOT
IODATA_OE_NOT
IODATA_DIR
ADC12_WR_NOT
ADC12_CS0_NOT
ADC12_CS1_NOT
ADC12_HBEN
ADC12_RD_NOT
BP_DATA_BUS[7:0]
MUX2_A2
MUX1_A0
MUX_WR_NOT
MUX2_A0
MUX2_A1
MUX1_A1
MUX1_A2
ADC16_DCLK
ADC16_CS_NOT
ADC16_DIN

VITA

Bryce D. Bolton

Bryce Bolton was born in Jacksonville Florida on October 11, 1972. As part of a Navy family, he lived in Florida, California, Okinawa Japan, Madison Wisconsin, and Virginia Beach. After graduation from Kempsville High School in Virginia Beach, he entered the engineering program at the main Virginia Tech campus in Blacksburg, Virginia. During his undergraduate studies, Bryce was the president of the Virginia Tech Amateur Radio Association, and was also a member of the VPI Cave Club. While pursuing his undergraduate degree, he worked for IBM Federal Systems Manassas, IBM Air Traffic Control/Loral Federal Systems in Gaithersburg Maryland, and Data Measurement Corporation, Inc. in Gaithersburg, MD as part of the Virginia Tech Co-op program.

After receiving a Bachelors degree in Electrical Engineering from Virginia Tech, Bryce followed his love of the outdoors for 1800 miles of the Appalachian Trail in 1995 from Maine to Asheville, North Carolina. After returning home due to bad weather, he went to work for approximately 18 months with Lockheed Martin Federal Systems in Manassas, VA. In the space systems software group, he wrote device drivers and test software for Space Missile Tracking System (SMTS), GPS Block IIF, and Mars program RAD6000 processors.

Missing a small town lifestyle, Bryce returned to Blacksburg to pursue a Masters degree and work for Fiber and Sensor Technologies, Inc. At Fiber and Sensor Technologies, he wrote the flight software for the Shape Memory Alloy Thermal Tailoring Experiment (SMATTE), which successfully flew as a payload on the MightySat 2.1 Air Force Satellite. Later, he worked on wireless low-power fiber-optic data acquisition systems for the paper mill industry. He presently works for Luna Technologies, Inc. as an embedded design engineer and completed the Appalachian Trail in the summer of 1999.