USE OF FIELD PROGRAMMABLE GATE ARRAY TECHNOLOGY IN FUTURE SPACE AVIONICS

Roscoe C. Ferguson, Robert Tate, United Space Alliance, LLC
Houston, Texas

Abstract

Fulfilling NASA's new vision for space exploration requires the development of sustainable, flexible and fault tolerant spacecraft control systems. The traditional development paradigm consists of the purchase or fabrication of hardware boards with fixed processor and/or Digital Signal Processing (DSP) components interconnected via a standardized bus system. This is followed by the purchase and/or development of software.

This paradigm has several disadvantages for the development of systems to support NASA's new vision. Building a system to be fault tolerant increases the complexity and decreases the performance of included software. Standard bus design and conventional implementation produces natural bottlenecks. Configuring hardware components in systems containing common processors and DSPs is difficult initially and expensive or impossible to change later.

The existence of Hardware Description Languages (HDLs), the recent increase in performance, density and radiation tolerance of Field Programmable Gate Arrays (FPGAs), and Intellectual Property (IP) Cores provides the technology for reprogrammable Systems on a Chip (SOC). This technology supports a paradigm better suited for NASA's vision. Hardware and software production are melded for more effective development; they can both evolve together over time.

Designers incorporating this technology into future avionics can benefit from its flexibility. Systems can be designed with improved fault isolation and tolerance using hardware instead of software. Also, these designs can be protected from obsolescence problems where maintenance is compromised via component and vendor availability.

To investigate the flexibility of this technology, the core of the Central Processing Unit and Input/Output Processor of the Space Shuttle AP101S Computer were prototyped in Verilog HDL and synthesized into an Altera Stratix FPGA.

1. Introduction

NASA's Constellation is the collection of systems that work together to support the new vision for the exploration of the Moon, Mars, and the outer solar system. These systems include launch vehicles, interplanetary vehicles, crew habitats, robotic systems and ground systems. They are to be implemented in a long-term development methodology. The goal of this strategy is "to develop new capabilities in a manner that is pragmatic – so that new capabilities can be developed and used to advance exploration in the near term – while also being flexible, in order to incorporate new technologies and respond with agility to scientific discoveries… NASA's acquisition strategy encourages the use of open-systems architectures that facilitate upgrades and augmentation while enabling interoperability among systems"[1].

These systems will require advanced and high performance data processing elements due to the distance of mission and the need for more advanced and autonomous operations. Furthermore, these systems must be highly reliable and evolvable. For data processing, the traditional development paradigm of using hardware with fixed processing elements followed by software development is acceptable, but provides less flexibility in support of NASA's development strategy. The primary culprit is the use of fixed processing elements and configurations that are more difficult to change and evolve after they have been established. The existence of Hardware Description Languages (HDLs), the recent increase in performance, density and radiation tolerance of Field Programmable Gate Arrays (FPGAs), and Intellectual Property (IP)

Cores provides the technology for reprogrammable Systems on a Chip (SOC). This technology provides for more flexible and evolvable designs.

This paper discusses the advantages of incorporating SOC technology into future space avionics. It also suggests some approaches to simplifying the software of systems by removing complexity from applications, and moving Real Time Operating Systems from a single processor to a collection of FPGAs. It includes a demonstration of the flexibility of this technology via the implementation of the major components of the Space Shuttle AP101S Computer in Verilog HDL and its synthesis into an Altera Stratix FPGA. It closes with a discussion of how this implementation could be incorporated into the avionics of a future launch vehicle reusing components of the existing Space Shuttle Transportation System.

2. Traditional Development Paradigms/ Platforms And NASA's New Vision

The traditional development process for embedded data processing systems begins with the purchase or fabrication of hardware boards with fixed processor and/or Digital Signal Processing (DSP) components interconnected via a standardized bus system. This is followed by the purchase and/or development of software. Development of the entire system proceeds in a stepwise fashion with the initially established hardware platform later constraining the software development effort. The inflexibility of the fixed hardware components along with the inherent bottlenecks of standardized bus systems often constrains on the adaptability and performance of the data processing system.

NASA will need to develop flexible and sustainable systems to implement its space exploration vision. This effort must provide very reliable systems that are evolvable for long-term development and maintenance. These systems must be highly automated and self-sustaining to support missions beyond low Earth orbit.

The traditional approach to embedded system development can be inefficient in a long-term development environment. Early decisions on fixed processing elements (Processors and/or DSPs) drive vendor selections and purchases drive vendor selection and purchases. A long-term marriage between these fixed processing elements and software results from the development effort, which can mean that the longevity of a system can come to depend on these vendors for both upgrades and long-term support. There is no guarantee that future versions of the fixed processing elements will be backwards compatible or that they will continue to be supported. Using high-level languages for software can remedy these dependencies; however, this is not totally true for embedded systems. These systems usually interface with low-level hardware and for performance reasons contain software written in the native language of the fixed processor elements.

The use of these fixed processing elements is usually accompanied by other components to ~~support interconnectivity using a standardized bus~~ system. This topology is a centralized architecture where a processing element communicates with peripherals and memory over a memory bus and a shared I/O bus. However, the drawback to this architecture is that the shared bus concept results in bottleneck problems [2]. This can be a hindrance for data processing systems that need to process high volumes of data.

A standardized bus system requires components to support interconnectivity in fixed processing elements. This results in a centralized architecture where a processing element communicates with peripherals and memory over a memory bus and a shared I/O bus. This traditional shared bus architecture can result in bottleneck problems [2], which is a hindrance for data processing systems that need to process high volumes of data. Such systems can end up with hardware-dependent software to overcome these bottlenecks

A standard architecture for software in embedded systems consists of application software layered on top of a real time operating system. The primary job of the real time operating system is to interface with the hardware and provide an environment where multiple applications can execute on a processor as if each were the sole owner of the processor. This architecture also helps protect the applications from each others' faults.. These systems must be designed carefully to manage against problems such as process starvation in a single processor.

The development effort of the recently cancelled Space Shuttle Cockpit Avionics Upgrade Project provides real world examples of the shortcoming of traditional stepwise development of processing systems. The goal of this project was to increase the situational awareness of the crew via increased on-board intelligence and automation, similar to what will be needed for NASA's new vision. The core-processing element for the system was the PowerPC 7455 processor. This was the best available processor in its class for speed and radiation tolerance, and the architecture of the single board computer was based on this family of processors. However, while the PowerPC 7455 cache memory contained parity protection, its cache tags did not. The project had to use software mitigation techniques to work around this problem. These techniques increased the complexity of the overall software design. As another example, the increased on-board intelligence and automation led to a need to process large amounts of data, and algorithms requiring expensive math operations. The project quickly encountered I/O bottleneck problems and had to resort to compromises. Also, some of the math intensive operations led to performance problems on one of the fastest embedded processors available on the market. The project overcame these challenges, but at the cost of schedule.

There is a high probability that NASA will face these challenges in the development of its new vision. The traditional development paradigm does not prevent success, but consideration of SOC technology can provide an easier road to success.

3. System On A Chip And FPGA Technology Primer

System on a Chip refers to the integration of varying electronic circuits onto a single chip to form a system. This differs from building an electronic product by the assembling of various chips and components on a circuit board. The tools commonly used to implement this capability consist of Hardware Description Languages (HDLs) and targeted logic devices such as Application Specific Integrated Circuits (ASICs) and Field Programmable Arrays (FPGAs). ASICs have traditionally been expensive to implement, so the rise in the density and performance of FPGAs and their low cost are making them the favorite option [3]. This paper will focus on the use of FPGAs as the target logic devices.

Hardware Description Languages are specialized programming languages used to model and design digital hardware. Large companies use them to design complex digital systems such as computer central processing units, computer peripherals and cell phones. An HDL allows the engineer to model hardware functionality as a software program. The model can then be run on a computer to see if the design will work as intended. Problems can be corrected in the model, and the corrections verified in simulation. Examples of these languages include Verilog and VHDL. An emerging trend is the use of traditional high level programming languages such as "C" to model and represent hardware

The basic development flow for creating these systems consists of providing HDL files as input to a tool set that synthesizes the design into a format to be loaded and realized on an FPGA. The process is similar in concept to the compilation and linkage of software to execute on a target processor. These tool sets normally provide support for debugging, simulation, optimization, timing analysis, and integration. Examples of such tools sets are the Embedded Development Kit and Platform Studio by Xilinx and Quartus II by Altera.

FPGAs are programmable digital logic devices. In a nutshell, their fundamental technology consists of arrays of logic elements and registers, along with a configurable interconnection matrix. Design implementation involves configuration of the interconnections. FPGAs use SRAM, Flash Memory, or antifuse programming technology to capture the configuration. SRAM and Flash Memory FPGAs use memory for device configuration. Memory cells are used to control a transistor at a crossover point on the interconnect matrix. While both SRAM and flash memory FPGAs are reprogrammable, the SRAM devices will lose their configuration when power is removed. Also, SRAM devices require a non-volatile configuration memory unit to store configuration data. This data must be loaded into the SRAM device before use. A flash FPGA does not require this additional complexity. Antifuse programming devices are not reprogrammable. The interconnect matrix is maintained using fusible material. The configuration design is "burned in" after programming [4]. Besides supporting custom logic, FPGAs are now providing support for embedded multipliers, memory, regional clock systems, and IO.

For the harsh environment of space, there are FPGAs that are radiation tolerant. Single event upsets (SEUs) occur when a charged sub-atomic particle strikes a flip-flop or an SRAM cell. This threat comes from the abundance of heavy ions and protons in the background galactic cosmic radiation and in the solar wind [5]. For FPGAs, upsets must be considered for both configuration (single event functional interrupt, or SEFI) and logic.

Radiation testing of FPGAs performed by iRoc Technologies indicates that antifuse programming and flash memory FPGAs' SEFI performance is very good [6]. Actel is a prime producer of antifuse and flash FPGAs. Its products have been proven in space applications such as Atlas II, Echostar, SBIRS-High, International Space Station, Hubble Space Telescope and the Mars Pathfinder. The subsystems involved include command and data handling, attitude reference and control, communication payload, and scientific instruments [7].

SRAM FPGAs can use mitigation techniques to manage SEFI events in radiation-heavy environments. The most common is configuration bitstream repair. This consists of cyclically reading the FPGA's bitstream configuration data and checking against a reference for discrepancies [6]. Xilinx has families of FPGAs that provide non-intrusive access to the configuration bitstream of the device [8]. Altera provides a family of FPGAs with built in cyclic configuration bitstream checking [9]. Triple module redundancy (TMR) is a technique that can be used to check for upsets to logic and memory subsystems. In TMR, elements such as flip-flops are implemented three times in parallel and a voting circuit compares the outputs [6]. Xilinx provides support tools for the seamless incorporation of TMR into designs [10]. SRAM FPGAs from Xilinx have been successfully used in the space environment. The most recent success are their use in the Spirit and Opportunity Mars Rovers missions. The Xilinx devices are used to control the pyrotechnic devices on the lander, and several motor control functions on the rover, including controllers for the wheels, steering, and antenna gimbals [11]. Another example is its use on the Optus Communications Satellite for signal processing [12].

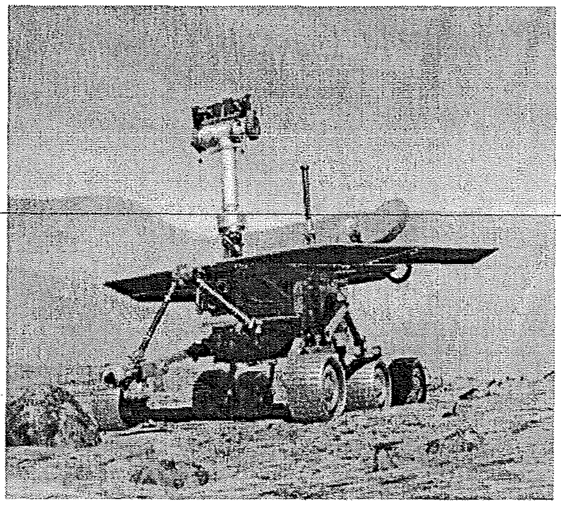

Figure 1. Artist Rendering of Mars Rover[1]

Intellectual Property Cores provide pre-packaged design components to be incorporated into SOC designs such as the examples above. Examples include embedded processors, Digital Signal Processors, bus interface units, and communication support. Engineers can pick and choose IP Cores to incorporate into designs for synthesis. "Soft" designs are purchased instead of silicon. For examples, IP cores for PowerPC 405 CPUs can be incorporated into Xilinx SOC systems and Nios RISC CPUs into Altera SOC systems.

[1] Courtesy of NASA JPL

4. Benefits Of System On A Chip And FPGA Technology In NASA's New Vision

The implementation of NASA's new vision will require evolvable and high performance data processing systems for avionics. SOC and FPGA technology can provide a design methodology better suited to support these needs.

4.1. Reduction In Risk Of Obsolescence

The hardware designs for SOC systems are represented in HDLs. These are eventually synthesized into bitstreams and are used to program the FPGA. This allows for a legacy design that can be re-targeted to future FPGAs with increased performance and densities. It also allows for legacy designs to be incorporated into new development platforms with the flexibility for design tweaks for enhanced capability. The investment made in software and the supporting infrastructure can be preserved across the lifecycle of the project, instead of the current processing platform. NASA makes huge investments in the man rating of avionics (both hardware and software) via development, maintenance, and verification costs. For upgrades, the majority of the cost can be limited to the recertification of the hardware, allowing for more efficient and safe incorporation of some new technologies.

4.2. Better Supports Long-Term Development

The representation of hardware design in HDLs and the programmability of FPGAs provide the tools for evolvable data processing systems. Long-term development can result in an iterative approach where lessons learned and bugs found on previous iterations are applied and fixed on future iterations. This concept works well for software, but not for fixed silicon hardware. To adapt to changes in fixed silicon components, either software workarounds must be provided or replacement devices must be located. However, with an SOC the designer can make design changes to HDL to adapt hardware. Modified HDL can be realized in FPGAs because they are re-programmable. SRAM FPGAs provide the most flexibility from a device point of view. However, it is even more feasible to modify HDL, program a radiation-resistant antifuse FPGA, and replace existing board components than to resort to software workarounds or fixed silicon device replacement.

4.3. Better Flexibility To Support High Performance Systems

Systems made up of fixed silicon processors, DSPs, and traditional shared bus architectures and protocols have worked well in avionics systems for decades. However, as the data processing requirements of these systems have increased, the overall performance has decreased due to serialization. HDLs and FPGAs offer the ability to design systems that utilize parallel processing to boost performance.

IO bottlenecks are one of the biggest inhibitors in the performance of traditional data processing systems. They result from the serial nature of the standard shared bus architecture. In this architecture, a single arbitrator controls communication among one or more bus masters and slaves. Since the bus is shared, only a single master may control the bus at any one time. An arbitrator is responsible for granting access. When a master has control of the bus, all other masters on the bus must wait for the completion of the transaction before it is possible to proceed with their transactions. This serial scheme can result in bottlenecks in systems requiring high data throughput. FPGA systems provide various techniques to overcome this limitation. Since FPGA systems allow for the consolidation of components into a single unit, these systems can take advantage of the internal routing structures to produce non-traditional bus systems. An example is the design of the Altera Avalon Bus. This bus uses the concept of simultaneous multi-master bus architecture. "The system does not have shared bus lines like traditional microprocessor-based systems. Instead, each master-slave pair has a dedicated connection between them. When a peripheral must accept data from multiple sources .. multiplexers (not tristates) feed the appropriate signal into the peripherals… Because master and slave peripherals are connected with dedicated paths, multiple masters can be active at the same time and can simultaneously transfer data to their slaves" [13]. Of course a bottleneck can occur in this particular scheme if multiple masters request the same source at a high frequency, but the bottleneck does not inhibit the flow of data for non-related data transfers. This flexibility allows for other design solutions around this problem as well. For example, RapidIO is a high performance interconnect technology for passing data between processors, DSPs, systems memory and peripheral devices within a system. It allows for performance for up to 10 Gigabits per second and beyond. There are IP cores available to support this technology in FPGAs [14].

FPGA systems also provide support for computation intensive operations. Advanced data processing algorithms can require a large number of multiply and add operations. These can be performed in basic processors, but DSPs have been optimized for these operations. However, the serial nature of both these devices can limit performance. FPGAs are now providing dedicated circuitry to support multiple embedded multiply and add operations. This allows for the parallel crunching of algorithms, which can provide an enormous performance boost. For example, Xilinx has families of FPGAs that can perform 256 MAC operations in a single clock cycle, vs. 256 clock cycles for a dedicated DSP [15].

HDL and FPGAs also offer other options such as implementing software algorithms in hardware. Products are available where designers can implement algorithms in the "C" language to be synthesized into a FPGA directly [16]. Tests have been performed that show algorithms implemented in hardware in an FPGA have executed 28 times faster than the same algorithms executing as software on a processor running a clock 15 times faster than the FPGA [15].

5. Increasing Fault Tolerance, Reliability, And Fault Analysis Using System On A Chip And FPGA Technology In NASA's New Vision

SOC and FPGA technology provides the flexibility to increase the reliability and fault tolerance of data processing systems. It also provides the flexibility to design systems with increased insight for fault analysis.

5.1. Increasing Fault Tolerance and Reliability

The hardware for traditional data processing systems consists of dedicated fixed components that are interfaced using a shared bus. The standard centralized architecture normally has a processor and its peripherals interconnected on a shared bus using bus controllers. Each of these devices usually exists as a separate package.

SOC and FPGA technology allows for the creation of de-centralized technologies, which can provide better fault tolerance and reliability. They allow for the consolidation of processing elements and peripherals into a single chip. This consolidation provides relief into the design of radiation tolerant boards. For example, all device components on a board should be radiation tolerant. A designer must ensure that all necessary components that are provided by the various vendors are radiation tolerant. However, a SOC design can be completely synthesized into the same radiation tolerant FPGA. A board containing multiple SOC devices can communicate using high speed chip to chip interconnect protocols such as RapidIO. Depending on one's design, this can minimize the threat of single point failures. For example, in the traditional centralized hardware architecture, a failure of a dedicated bus interface device can cripple the entire system. With a de-centralized architecture, consideration must be given to memory usage and interfacing. In a traditional system, there is usually a single memory address space used by one or multiple processing elements. However, a designer can take advantage of FPGAs with embedded memory resources to produce isolated systems that interface via a high-speed protocol. If the memory capacity is inefficient or non-existent in the FPGA, then it may be feasible to interface each FPGA with its own memory space.

A de-centralized design utilizing SOCs can also produce more reliable systems via the Keep it Simple, Stupid (KISS) paradigm. Simpler systems are less likely to fail than more complicated ones. Real Time Operating Systems are used to allow multiple software applications to share limited resources such as processors and peripherals. However, for these entities to coexist, mechanisms are needed to manage access to shared resources. For example, semaphores, tasks locks, etc. are used in software to manage the use of global objects by multiple processes. Design oversight can lead to problems such as deadlocks. Also, proper analysis must be performed during design to prevent runtime issues such as process starvation. These issues have been seen time and time again in real time systems. Using de-centralization, software can be broken up into individual simpler components that execute in parallel on multiple hardware components. This can occur either within a single SOC or on several interacting SOCs. A single SOC system can consist of multiple processors executing individual software applications. Both systems can provide interaction via message passing constructs. Software components can be partitioned to execute on self-contained units according to performance and criticality requirements. A smaller number of components can be designed to compete for the same processor resources. Also, improved fault tolerance is provided because these individually simpler interacting components are isolated by hardware. ARINC 653 standard systems provide improvement for the traditional paradigm, but a failure of the CoreOS or a single component of the hardware can disable the entire system.

5.2. Increasing Fault Analysis

Being able to detect faults is an important aspect of highly reliable systems. However, being able to analyze *why* a fault occurred is even more important. Fault analysis of software failures can be a very complicated and difficult venture. A common method of analyzing why faults occur in software is to understand software execution paths. This is normally achieved by leaving "breadcrumbs" via data logging during production runtime, or by using a debugger to "single step" through an execution path. "Breadcrumbs" usually do not leave enough information during runtime to analyze complex problems. A debugger is a post mortem tool, used by a developer attempting to recreate the problem. However, the developer may not be able re-create the exact combinations of real time scenarios that caused the problem. Designers can use the flexibility of HDLs and FPGAs to create system add-ons that can provide better information for fault analysis for both production and non-production runtime environments. As an example, a designer can modify the processor or processors of a system to interface with a component that tracks the branches of software executing in run-time without any performance degradation. This external hardware component could build and maintain execution trees in memory based on branches. There are three ways to take advantage of this system. First, it could be used to analyze the execution paths of the software to determine the root cause of problem. Next, it could be used during the software verification phase to track which software paths have been verified. The memory of verified paths could be loaded into production systems where unexpected execution paths not verified could be reported using the stored execution paths as "expected" paths. The final use could be for the indication of "wild execution paths" during production use. The system could report an error and configure itself to a safe mode. Of course, real-time software updates (patches, etc.) would have to be considered as a factor in a production environment.

6. Technology Demonstrator - Space Shuttle AP101S Computer Components In FPGAs

To investigate the promise and flexibility of SOC and FPGA technology, research based on technology from the current Space Shuttle Program was performed. The goal of the research was to design the core components of the Space Shuttle AP101S Computer in an HDL. The functionality of the HDL would then be tested via simulation and synthesis into an actual FPGA.

The AP101S Computer serves as the primary platform for the Space Shuttle control system. The design is based on the IBM System/360 mainframes. It consists of a Central Processing Unit (CPU) and an Input/Output Processor (IOP). The CPU provides instructions to support IO control, fixed point arithmetic, branching, shift operations, logical operations, floating point operations, and special operations. The IOP provides support for 24 external communication channels over 1 Mbits/s MIA Buses. It also provides instructions used for the IO management control. Both the CPU and IOP share a 1 Mbyte memory space.

The designs were implemented in Verilog HDL. The Quartus II design software by Altera Corporation was used for design, synthesis, and simulation at the device configuration file level. Icarus Verilog by Stephen Williams was used for pre-synthesis simulation (Verilog level). The Altera Nios II Development Kit, Stratix Edition was used for hardware tests. The FPGA device used in this kit was the Stratix EP1S10F780. This device provides 10,570 Logic Elements (LEs), 94 M512 Ram Blocks (32 X 18 bits), 60 M4K Ram Blocks (128 x 36 bits), 1 M-Ram Block (4K X 144 bits), 6 DSP Blocks, 48 Embedded Multipliers, 6 PLLs, and 426 IO Pins. The EP1S10F780 is the smallest device of the Stratix family. The kit also provides external components such as 1 MB, 16 bit wide SRAM, an Ethernet controller, and serial controllers [17].

To demonstrate flexibility, the CPU was redesigned from the original version, while the IOP design was based on the microcode of the original version. The re-constructed IOP microcode allowed for a design that could theoretically use the existing IOP micro programs of the AP101S.

The major components of the CPU design were the Microcode Controller, Instruction Decoder, ALU, Branching Unit, AP101 Memory Controller, and the Register File. The CPU instructions were micro-programmed and executed by the Microcode Controller. The instruction micro-programs were stored using Stratix FPGA embedded memory, which was interfaced to the Microcode Controller. The Instruction Decoder was designed to decode instructions to determine the starting address of the micro-program for the Microcode Controller. The ALU was implemented to support both Fixed and IBM Floating Point operations. It used Stratix embedded DSP blocks to implement both multiplication and division operations. The AP101 Memory Controller was designed to interface the CPU to the external 1 MB of SRAM provided by the development kit. It performed memory operations such as address expansion, AP101 effective address calculation, instruction pre-fetching, and memory access. The Branching Unit was used to update both the micro and macro program counters and the Register File stored the results for the macro instructions (Figure 2).

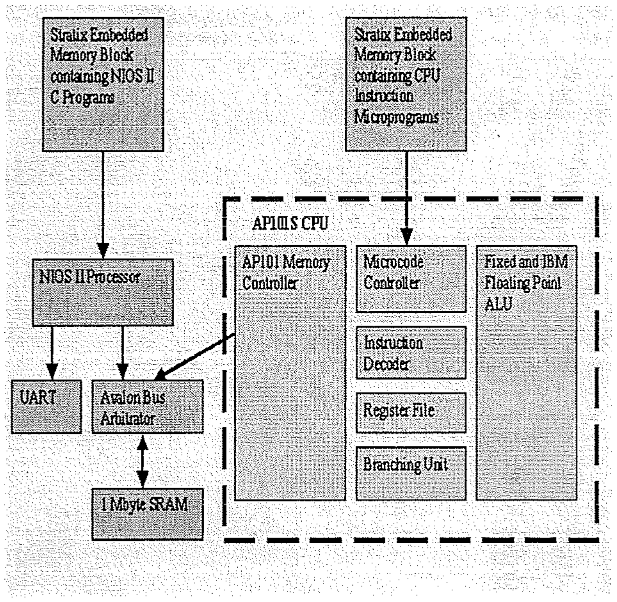
Stratix Embedded Memory Block containing NIOS II C Programs
Stratix Embedded Memory Block containing CPU Instruction Microprograms
AP101S CPU
NIOS II Processor
AP101 Memory Controller
Microcode Controller
Fixed and IBM Floating Point ALU
Instruction Decoder
Register File
Branching Unit
UART
Avalon Bus Arbitrator
1 Mbyte SRAM

Figure 2. AP101S CPU in Stratix FPGA

To provide a testing and debug interface for insight into CPU operation, an Altera Nios II processor was included in the FPGA design. The Nios II processor was interfaced to access the 1 MB memory space used the AP101S CPU. Arbitration was provided by the Avalon Bus interface. The Nios II processor was also interfaced with a serial UART device for communication with the console of an external computer.

The total synthesized size of the CPU was 5,500 LEs, 52% of the Stratix FPGA. The size of the debug interface (Nios II processor and other components) was 2,700 LEs. The total FPGA resources used were 8,200 LEs or 78% of the FPGA.

The operation of the CPU was tested using the iVerilog simulation tool and on the Stratix device in the development kit. For each case, AP101S test assembler programs were executed and monitored. The test programs covered only a subset of the instruction set. All tested instructions produced expected results.

The Quartus II Timing Analysis Tool analyzed a design clock period of 20 ns or 50 MHz. This clock signal was routed through two Phase Locked Loops (PLLs) to produce seven separate clock signals. The first clock (50 Mhz) was distributed to the Nios II processor and Avalon Bus. A 20 Mhz clock signal and five phased 4 Mhz clock signals were distributed to the AP101S CPU. The prototype CPU could match the real AP101S fastest instruction time one for one. However, the most of the prototype CPU instructions were designed to execute in fewer clock cycles. The prototype design did not implement a pipelining scheme such as the actual AP101S. A detailed timing analysis of each instruction was not performed, so no accurate timing data is available for comparison.

The FPGA version of the IOP was designed to mimic the Micro Controller design of the actual AP101S. The infrastructure based on the micro-instruction formats was implemented using specifications from the AP101S Workbook (IBM No. 85-C67-005). Test micro programs were written for the microcode controller and were simulated using the iVerilog simulation tool. The design did not provide support for the MIA, the component that is responsible for sending data from the BCE elements over the physical media. The synthesized size of the IOP components was 2500 LEs of the Stratix FPGA.

As of the time of this study, the entire system was not implemented and integrated into a complete unit based on the FPGA size (smallest in Stratix family) in the development kit. However, the study did implement core components of the AP101S in Verilog HDL and synthesized them into Stratix FPGAs. The AP101S CPU and IOP could be easily be implemented in a larger-sized Stratix device. A larger-sized device would also simplify the design; for example, the SRAM Controller could be removed by using embedded FPGA memory resources. However, the design could also be implemented by integrating more than one FPGA.

A future activity is to develop an AP101S Virtual Machine. The AP101S CPU and IOP will be implemented in software to execute on two Nios II processors embedded in the Stratix device of the development kit. Custom logic for the IBM floating-point style unit will be implemented and interfaced to one of the Nios II processing using the custom instruction interface.

7. Theoretical Application For Space Shuttle AP101S Computer In FPGAs

In the 1980's, the Space Shuttle Program upgraded its General Purpose Computer (GPC). The goal was to maintain the existing interface to minimize changes to the existing flight software, while at the same time improving the underlying technology, primarily using discrete TTL logic devices. Now, let's go back in time to this period and assume that the project modeled the GPC's CPU and IOP in an HDL to simulate the design. In fact, HDLs existed during this period as a hardware-modeling tool [18]. With this assumption that the design of the GPC existed in a HDL in the 1980's, let's move forward in time to the 2000's. Let's assume that during this period, NASA is in a bind where it needs to launch five 68 metric ton modules starting within 3 years to save the Space Station. NASA does not have an existing booster with the required heavy lift capability, and it cannot build a new one within this 3-year period. A viable option may be to use the components of the Shuttle Stack such as the SRBs, External Tank, and SSMEs as the heavy lift vehicle and develop a new payload canister (Figure 3). However, this design would require additional avionics components not compatible with the bus architecture of the current system, and the avionics must be expendable. A crew must be launched along with the payload, so the system must be man-rated and must interface with the legacy system. NASA can upgrade the avionics to support the new system, but this would also require porting the existing flight software into a new platform. This seriously hampers the project, because a rehost of the software into a new platform and its verification for a man-rated system would be hard to achieve in the timeframe.

A solution to this problem is to use the existing HDL of the GPC design to synthesize the legacy computer into FPGAs. This would allow for the reuse of the existing infrastructure and flight software. The project could then focus its effort on the development and verification of the new hardware and integration. This is compared to the development and verification of both new hardware, software, and infrastructure required for a rehost. Engineers could overcome the problem of the new bus architecture by modifying the HDL of the IOP to encapsulate the existing software from the changes. The HDL of the memory unit could be modified to create a new high-speed data interface to the payload. In general, this approach has a better chance of success when compared to the rehost of the system into new avionics and software in the 3-

year time frame. Also, the synthesis of the GPC and its new supporting components into FPGAs will allow for the advent of expendable avionics hardware. The FPGAs could be used to create a Flight Control Card to be inserted into a new avionics computer (Figure 4). This theoretical example shows how SOC and FPGA technology helps solve the problem of obsolescence for long term projects. It also shows how the flexibility of the technology can allow for a system to evolve with minimum impact.

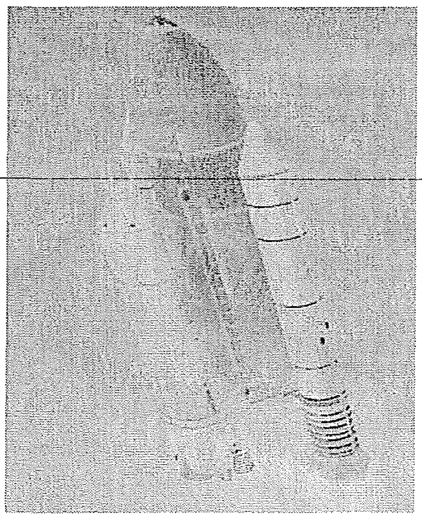

Figure 3. Artist Rendering of an SDLV[2]

8. Conclusion

SOC and FPGA technology should be considered to support the development of systems for NASA's new vision. The flexibility of this technology can reduce the risk of obsolescence, provide better support for a long-term development paradigm, provide the horsepower for high performance systems, and increase fault tolerance. The technology for FPGAs supports the harsh environment of space. A recent success story is their use in the Spirit and Opportunity Rovers currently in operation on the surface of Mars.

[2] Courtesy of NASA

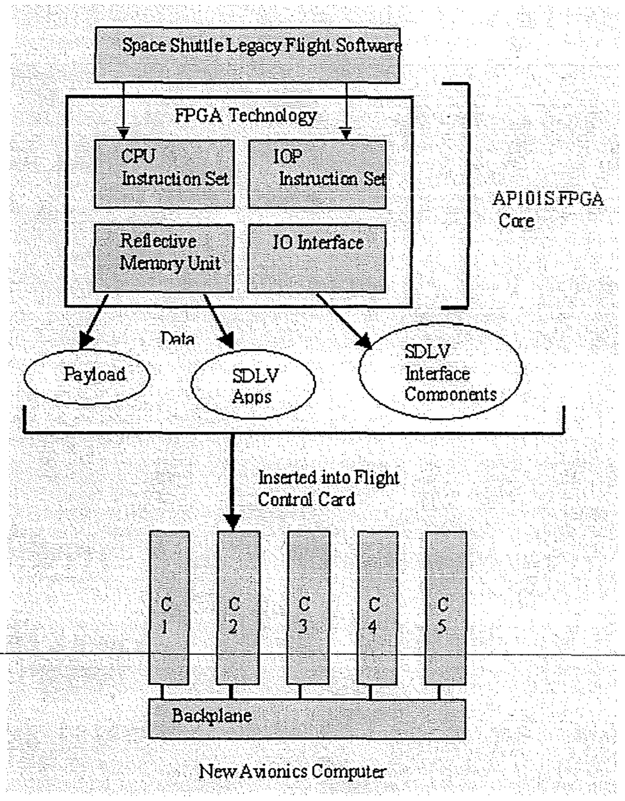

Figure 4. FPGA Technology in an SDLV

Acknowledgments

The authors wish to thank Mark Lostracco, Wendy Wilkinson, Brian Watson, Misty Wohl, Benjamin Peterson, and Christy Limero for their support of this work.

References

[1] National Aeronautics and Space Administration, March 1, 2005, Final Request for Proposal (RFP) NNT05AA01J, Crew Exploration Vehicle, Phase 1 Contract – Statement of Objectives (Attachment J-1), Washington, DC, pp. 2- 5.

[2] Mellanox, Kevin Deierling, April 2004, Advanced System Architectures Drive Choice of Switch Fabric Solution, RTC, pp. 22-25.

[3] Jaeger, Juergen, August 2004, FPGAs Flex Their Processing Muscles, COTS Journal, pp. 20-24.

[4] Wang, Mao, August 16, 2004, FPGAs Adapt to Suite Low-Power Demands, Electronic Engineering Times, p.1 of printed version. www.eetasia.com.

[5] O'Neill, Ken, August 2004, Designing Systems to Combat Single-Effect Upsets in Space and on the Ground, COTS Journal, pp. 54- 55.

[6] O'Neill, Ken, August 2004, Designing Systems to Combat Single-Effect Upsets in Space and on the Ground, COTS Journal, pp.56-57.

[7] Actel Website, The Actel Space Heritage, www.actel.com/products/aero/index.html#.

[8] Fuller, Earl, Micheal Caffrey, Anthony Salazar, Carl Carmichael, Joe Fabula, Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing, MAPLD 2000, p.4.

[9] Altera Application Note 357, Error Detection Using CRC in Altera FPGA Devices, pp.1-2.

[10] Xilinx Website, Industry First Development Tool to Automate Generation of Triple Module Redundant (TMR) Designs for Re-Programmable FPGAs, www.xilinx.com/products/milaero/tmr/index.htm.

[11] Xilinx Press Release 0412, January 22, 2004, Xilinx Chips Land on Mars, www.xilinx.com/prs_rls/design_win/0412_marsrover.htm.

[12] Xilinx Press Release 03135, September 23, 2003, Xilinx FPGAs Flying Aboard Optus Communications Satellite, www.xilinx.com/prs_rls/design_win/03135optus.htm.

[13] Altera Application Note 184, Simultaneous Multi-Mastering with the Avalon Bus, pp.1-3.

[14] Xilinx Website, RapidIO Overview, www.xilinx.com/rapidio/index.htm.

[15] Parnell, Karen, Roger Bryner, July 21, 2004, Comparing and Contrasting FPGA and Microprocessor System Design and Development, Xilinx, pp.18-19.

[16] Impulse Accelerated Technologies Website, C Programming Tools for FPGAs, www.impulsec.com.

[17] Altera Website, Nios II Development Kit, Stratix Edition, www.altera.com/products/devkits/altera/kit-nios_1S10.html.

[18] Palnitkar, Samir, 1996, Verilog HDL – A Guide to Digital Design and Synthesis, ISBN 0-13-451675-3, United States, Prentice Hall, p.4.

24th Digital Avionics Systems Conference
October 30, 2005