# Methodology for Fault Tolerant System Design Based on FPGA into Limited Redundant Area Mohamed El-Hadedy, Michael Bishara, Jinrun Liang, Juan Marquez, Arturo Murillo

***Abstract—This paper discusses the issue of Single Event Upsets (SEUs) that damage traditional devices that are sent into orbit on space missions. For this case, a Field Programmable Gate Array (FPGA) is the device that will be affected by radiation from the SEU resulting in the damage/loss of data. Triple Modular Redundancy (TMR) and a redundancy algorithm will be implemented as the basis of repair to allow the Static Random-Access Memory to hold the correct data despite radiation particles disrupting the natural timing of the logic. Simulations of faults will be run on the FPGA in order to test the resilience of the algorithm and the operationality of TMR in the given system. Assuming adequate testing, the FPGA should be able to self-correct any faults that happen in data flow as a result of a SEU.***

***RISC-V architecture will be implemented to PolarFire and Nexys 4 FPGA to test the functionality and find the difference of other ISA. RISC-V ISA is able to handle the most important features in a CPU. For example, it can be configured standalone as a simplified general-purpose computer. It is compatible for a CPU to include an extension of a different instruction set. The load-and-store architecture allows operands and destination sets in registers instead of in memory, which achieve orthogonality to achieve better memory maintenance.***

1. **Introduction**

As mechanical systems become increasingly more technologically advanced, it becomes difficult to manage all of the aspects of the device when evaluating defects with the device. Field Programmable Gate Arrays (FPGAs) are examples of these devices as they have what is called Partial Dynamic Reconfiguration (PDR) [1]. PDR allows this device to act as if it were multiple pieces of hardware due to it having programmable Complex Logic Blocks (CLBs) which are configurable blocks of logic that are described by programmed input data. This data can be separated into configuration bits and user bits. Configuration bits are to remain constant as their purpose is to describe a particular hardware circuit design. The user bits are meant to be the inputs of the user to change the values of the hardware described by the configuration bits [2]. Since the FPGA needs a place to store the programmed data, they have Static Random-Access Memory (SRAM) implemented onto the board so it may act as configurable hardware.

Since this project is in collaboration with The National Aeronautics and Space Administration (NASA) and Jet Propulsion Laboratory (JPL), this device is intended to be used in space travel. One notable event that has a significant impact on devices that are put into orbit are radiation effects known as Single Event Upset (SEU) [3]. This event occurs whenever alpha and neutron particles hitting the surface of the FPGA cause a change in the configuration bits by changing the state of the memory [2]. This creates a state of error within the timing of the machine, which will cause it to either stop transferring data or transfer incorrect data between that FPGA and the target device. Desynchronization of timing in communication between two devices results in a loss of data bits being transferred or the transmission of inaccurate data.

One notable way to prevent the ration effects on this device is to reinforce the actual hardware of the FPGA. Although expensive, radiation-hardened FPGAs are available for purchase and provide one solution to the SEU phenomenon. Notably, the radiation-hardened FPGA is very costly to the point where it is not preferable to use. The alternative to purchasing an expensive FPGA is to utilize Triple Modular Redundancy (TMR) within the programming of a standard FPGA. TMR branches off of the idea that the SRAM in the FPGA is creating combinational and sequential logic using lookup tables in the memory state. Lookup tables act as blocks of data that hold program memory that can be accessed by the program that utilizes it [4]. Expanding off of this idea, TMR acts as a way of hardening the circuit logic and flip-flops being formed by the combinational bits in the FPGA. Not only is the feedback path made up of SRAM, but the routing connection is also made up of SRAM cells, so those must be hardened as well [5]. TMR can take two different forms in an FPGA. At a top-down level, TMR acts as two replicas of the original system where the output becomes the majority output of the three available outputs. This is called module redundancy because if one of the three systems fails or provides an incorrect output, the output will still provide the correct information given that the other two replica systems are not damaged by a SEU. The other use TMR has is that it can apply just to a lookup table in an SRAM cell, which means that instead of just having a singular lookup table for the program to reference from, it will instead have three lookup tables. Consequently, TMR will fail if two out of the three systems are damaged by one or more SEUs, so it is not quite the most reliable form of redundancy. This concept is very similar to how Redundant Array of Independent Disks (RAID) operates for computer hard disk drives. RAID works by dividing data into separate chunks that are stored on different hard disk drives. Depending on the type of RAID, data can either be mirrored and copied, or rebuilt from data that is held on other drives [6].

Utilizing this idea of RAID, it is possible to make an effective redundancy algorithm using TMR in combination with the idea of radiation hardened SRAM components. Thus, leads to the topic of Partial Dynamic Reconfiguration (PDR). PDR is the concept that an FPGA can take the form of many different types of hardware without the need for re-fabrication through the use of Partial Reconfigurable Modules (PRM). After a full BIT file configures the FPGA, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured. This gives the ability to time multiplex hardware dynamically on a single FPGA [7]. Module redundancy ensures that the PRMs will not fail and can be restored to their previous states, similar to a RAID setup. Below is a demonstration of how a reconfigurable block will constantly change with a stream of input bits while allowing the FPGA to remain functioning as an overall system [Fig. 1].

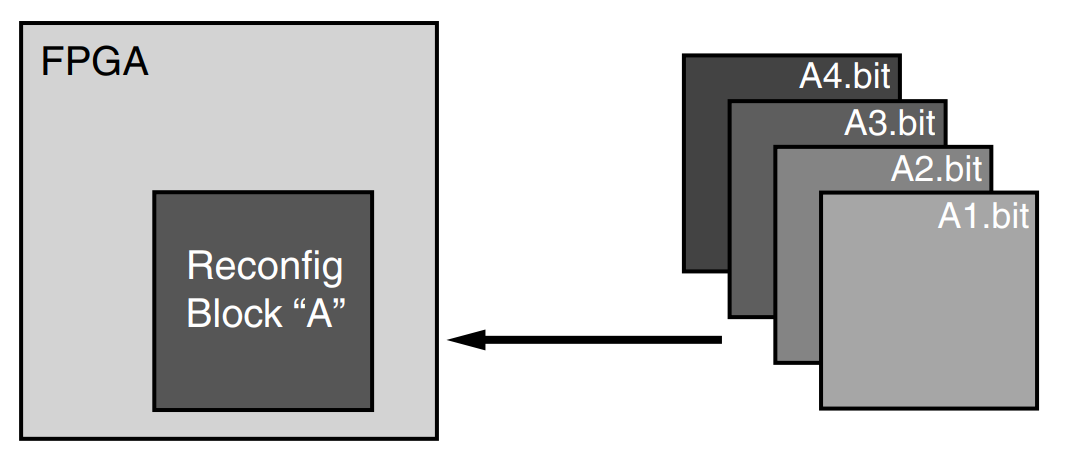


Fig. 1 Partial Reconfigurable Module Layout

Notably, there are also ways to add redundancy to the FPGA through means of hardware architecture also. Since the interconnects of an FPGA are set up in rows and columns fed to and from logic blocks, modification can be made to ensure there is no loss of data. For instance, the decoder can logically shift bits that are in adjacent rows of interconnects while disabling the defective interconnect. This process involves “lesser-blown poly fuses” since the FPGA is an SRAM-based FPGA and done by placing selectors after a row of decoders [8]. Contrary to the redundancy algorithm, the architecture is purely circuit based rather than programming based as hardware is physically implemented onto the FPGA to improve redundancy. As a result, this process is more expensive than the TMR alternative.

Additionally, there is another method meant to compensate for any errors that lie within the interconnects rather than the actual memory itself. This method is the cache coherency protocol, which essentially acts as a synchronizer between the cache/memory and the interconnect so that the redundancy does not have to assume the interconnect is functioning as intended. Notably, this implementation also includes algorithms meant to combat cases of deadlock and livelock, which are situations where the program is unable to function because each process is waiting for another process or variable before continuing execution [9].

1. **Requirements and Specifications**

Microsemi's [PolarFire](https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas) Splash Kit provides general purpose interfaces for evaluation and development. The kit connections include Gigabit Ethernet RJ45, PCIe, USB and LPC FMC connector. A PolarFire 300K LE FPGA along with on-board DDR4 and SPI-Flash memory enable general purpose evaluations.

* 300K LE [PolarFire](https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas) FPGA in a FCG484 Package (MPF300TS-1FCG484EES)
* PCI Express (x4) Edge Connector
* FMC Connector (LPC)
* x32 DDR4
* RJ45 port with 1 Gigabit Ethernet [VSC8541](https://www.microsemi.com/product-directory/gigabit-ethernet-phys/3886-vsc8541) 1-port PHY with RGMII interface
* [NX7102IDETR](https://www.microsemi.com/existing-parts/parts/80964) and [LX7167CLD](https://www.microsemi.com/existing-parts/parts/978) switching regulators
* On-board Power Monitoring
* USB for UART Interface & programming
* 1Gb SPI Flash Memory
* JTAG and SPI programming interface

In addition to this, we are required to use Libero to establish a “floor plan” for how the internal circuit layout should function [Fig. 2]. Libero is a native interface to the Polarfire splash kit and allows for the importation of both System Verilog (SV) and Verilog Hardware Description Language (VHDL). These two description languages help to establish the states of memory as well as the general clock timing functions and logic.

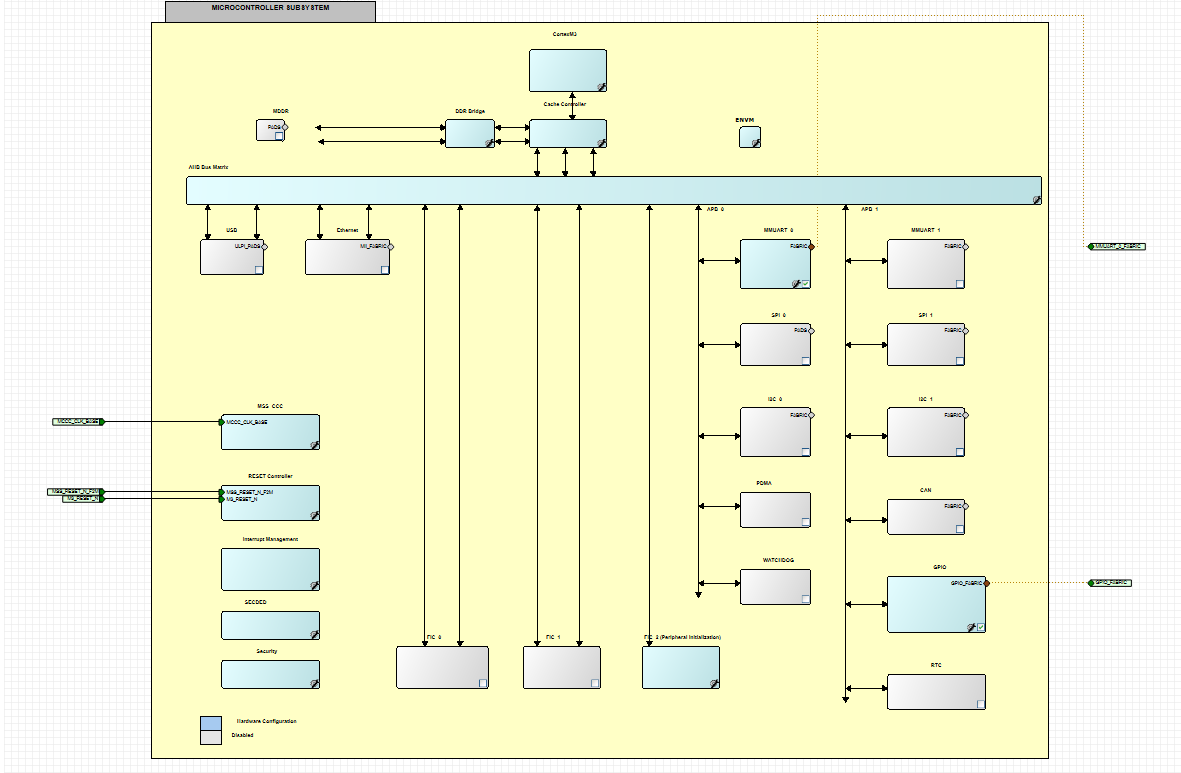


Fig. 2 Libero Circuit Floor Plan of Circuit Design

Once the layout of the floor plan is complete, the components can then be coded using either VHDL or SV. Furthermore, Serial Rapid Input Output (SRIO) communication is required so that the FPGA is able to communicate with other devices. Reduced Instruction Set Architecture (RISC) will be the architecture of choice in order to have a minimal instruction set operating on this FPGA while having it be easy to decode and simple to address.

1. **Design**

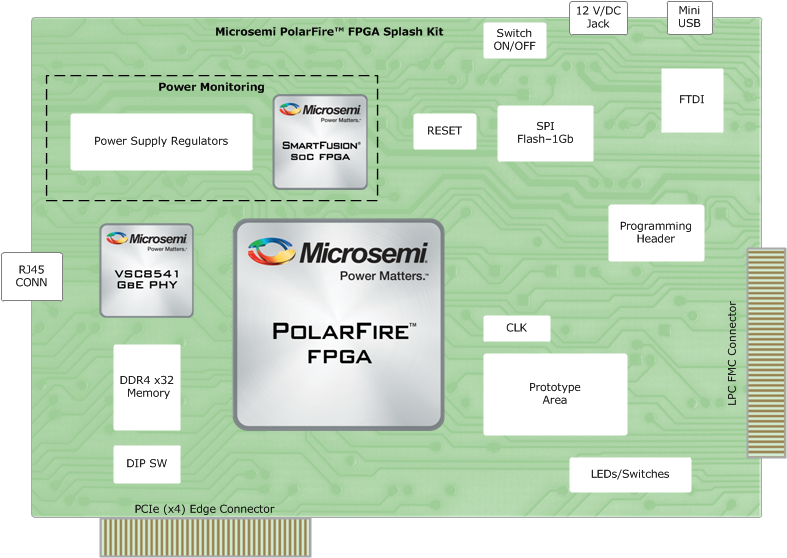
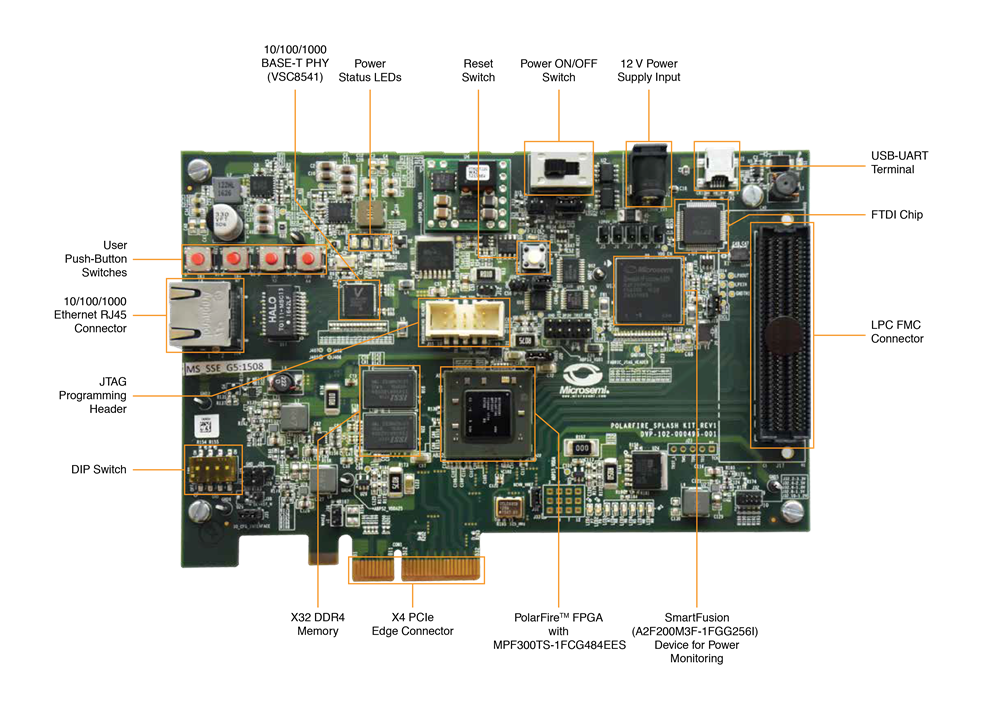


Fig. 3 PolarFire Splash Kit Board with MPF300TS-1FCG484EES Device and Hardware Schematic

The preliminary design for this project involves a Polarfire FPGA implemented with TMR, SRIO and the redundancy algorithm [Fig.3]. The state machine that holds the redundancy algorithm is still being developed, but the goal of the device is to be able to detect faults from SEUs purely through simulation. Since radiation cannot be used directly to test this device, SRIO communication will allow for the bitstream sharing between this device and another device receiving serial communication. From there, data will be transferred between the Polarfire and the “control” FPGA until the control purposefully sends a fault through a clock desynchronization. When this happens, the Polarfire’s algorithm should be able to repair the states of the SRAM so the timing of the received data will match the timing of the memory. Then, the TMR should be able to utilize multiple instances of output data to finalize the correct value using lookup tables [Fig. 4].

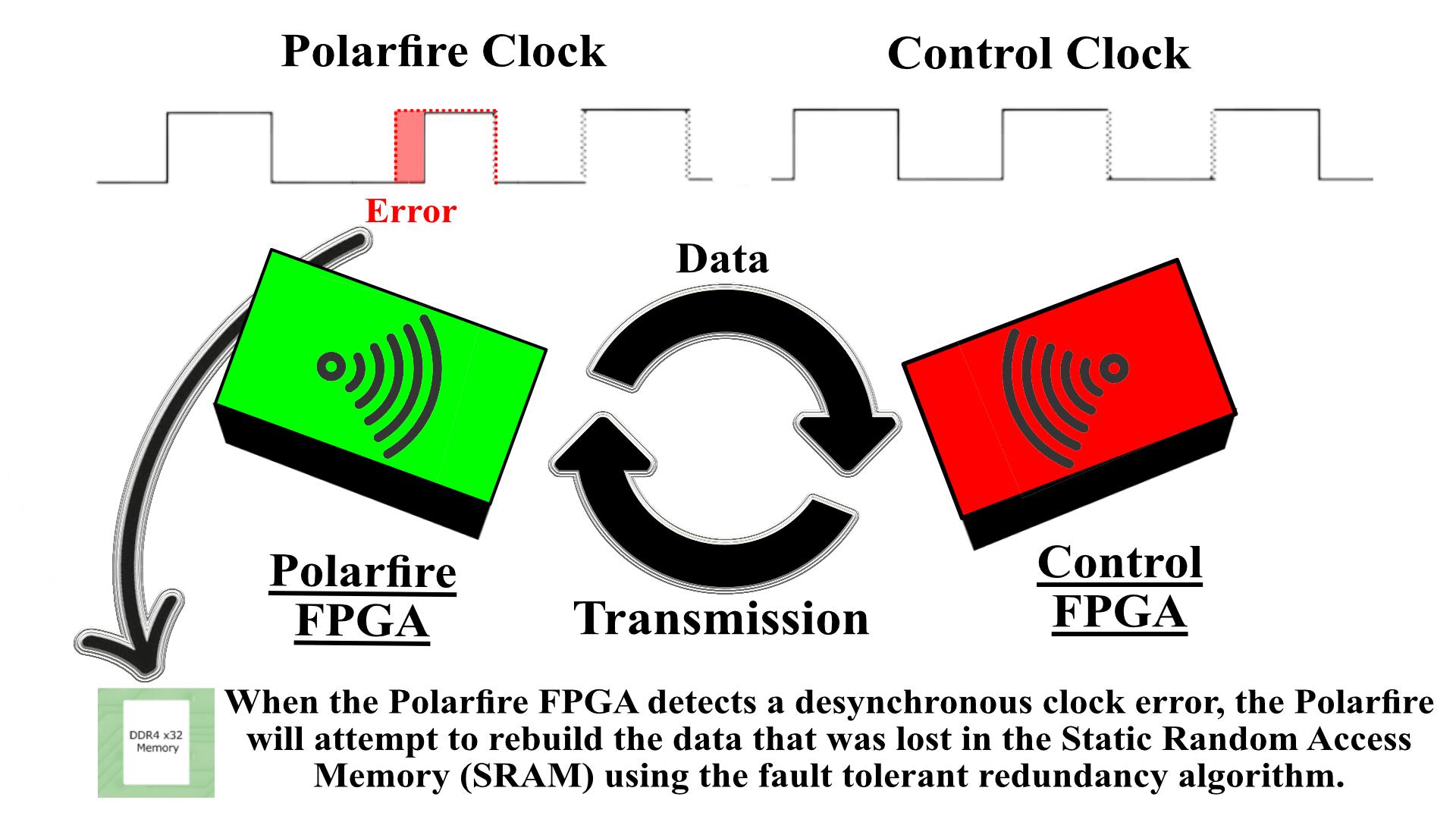


Fig. 4 Polarfire FPGA Design and Testing Post Implementation

1. **Standards and Constraints**

The two standards that were followed most closely for this project were the ASTM and the DO2 standards. The ASTM Energy Standards describes that natural resources are becoming scarce relative to energy consumption and that there are rising concerns of climate change. In an effort to conserve energy, we have considered placing solar panels to power the FPGA board in the final design as well as attempting to optimize the redundancy algorithm so that it consumes as little energy as possible whilst continuing to function as intended.

Additionally, the DO2: A Proven Global Reputation standard explains the steps to ensure statistical quality assurance among products tested in laboratories and the methods used to conduct tests that provide statistical data. In an effort to ensure that our product is working as intended, we are attempting to effectively simulate faults and other damages to mirror that of natural occurrences in orbit. Given the smaller scale and limitations that come with our project, we must research effective methods of simulation environmental factors rather than directly exposing the equipment to the environment that it will be in so ensure quality assurance.

Among the primary constraints for this project lies the constraints of accessibility, interoperability, maintainability, marketability, and usability. Accessibility will be an issue in this project as the FPGA board being used is no longer being sold to the public since it has been discontinued. As such, we are waiting on NASA/JPL (the partnered company for this project) to send us the board so that implementation may begin. Interoperability will also be an issue in this project because, in theory, the project is designed to be in orbit and constantly transmitting data back to devices on the ground. Since we have no way of putting the device into orbit, we must work with a much smaller communication range. Furthermore, because of the timeframe that we are working with, we are unable to test the devices maintainability over a long period of time to see if the device will maintain its function after being in space for example. Another issue is the marketability in this case. The device is intended to be sent to space and repair faults while in orbit. Yet very few companies in the world will be able to send a device like this to space, so it will be hard to market. Finally, usability may prove to be the most crucial constraint for this project since we cannot replicate any of the damages the device will take while it is in orbit. Rays from the sun and damage from space debris cannot be practically tested, so it must be simulated by signal scrambling instead.

1. **RISC-V**

RISC-V is an open-source Instruction Set Architecture (ISA) with origins at UC Berkeley. Directed by Prof. David Patterson, the RISC-V ISA was a part of the Parallel Computing Laboratory at UC Berkeley started by Prof. Krste Asanović and graduate students Yunsup Lee and Andrew Waterman. Although not the first open-source ISA, RISC-V has attracted a lot of attention and support from many companies like Intel and Microsoft. The goal of a fully developed RISC-V ISA is to provide open-source platforms that will not be limited by proprietary regulations. Smaller companies and academia are currently unable to access the current development tools necessary to design their own processors because the use of ISA’s provided by Intel, ARM, Wave Computing, etc. are restricted by expensive licensing and regulatory mandates. David Patterson argues that “Moore’s law is ending” and that “improvements in cost and/or performance are not coming from the semiconductor manufacturers, but they are coming from architectural innovation” [10]. This statement indicates the importance of the development for an open-source ISA. The semiconductor is nearing its limit to how small a transistor can get, so new processor developments have to arise from another place. This is where open-source RISC-V ISA will be of benefit. The ISA could be fitted to a particular application allowing improvements to the necessary areas of the chip. Improvements on performance could be achieved by customization and allowing chip variants to exist.

1. **Implementations of Polarfire and Nexys 4 FPGA**
2. **Polarfire**

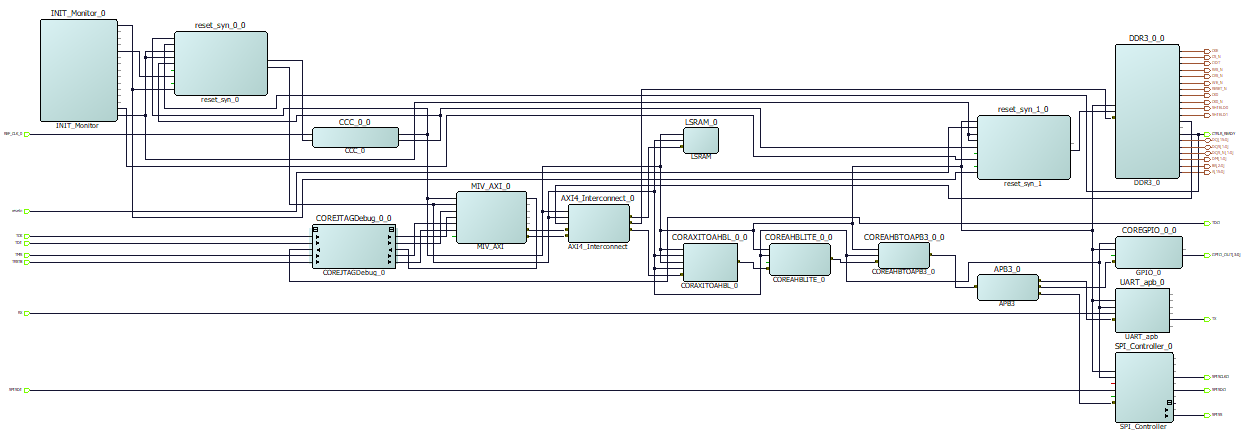


Fig. 5 PolarFire RISC-V Subsystem Connected

Microsemi PolaFire FPGA offers a comprehensive suite of software tools chains and IP cores to design RISC-V architecture. In order to build the RISC-V using the polarFire FPGA some requirements and prerequisites are needed before getting started. As we mention above the [PolarFire](https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas) Splash Kit will be needed to implement the hardware subsystem, as well Libero SoC design suit v12.5, and SoftConsole to run and implement the software. In addition to this, downloading and installing the design files are crucial to complete [Fig.5] RISC-V subsystem.

Furthermore, a quick overview of the taken process to build the RISC-V on libero. The first step is to create a project in libero and pick the right device selection for the polar fire board which is Part Number: MPF300TS-1FCG1152I. Then a new top smart design is added to the project. Using the catalog, the MI-V processor IP is added. This processor's main memory has to be on the Mi-V AXI memory interface whose memory-mapped address ranges from 0x80000000 to 0x8FFFFFFF. To continue adding the requirement components to connect subsystems the AXI interconnect Bus IP is added with three numbers of slaves and a width of four. The next component is the on-chip SRAM with a memory depth of 8192 and a width of five. From the catalog instantiate the DDR3 memory controller is crucial to set CCC PLL Clock Multiplier to six and width to sixteen, if not this is going to create an error when building it. The AXI3 TO SHB-LITE bridge, This will allow the CoreAXItoAHBL IP to connect an AXI bus to an AHB-Lite bus, enabling an AXI master to communicate with an AHBL slave/subsystem. The system will also need a uart controller therefore we need to instantiate a Uart controller form the catalog. Moreover, the general-purpose input and output is selected from the catalog under coreGPIO. The PolarFire includes two SPI Flash memories, therefore the CoreSPI IP is used to interface with the other SPI Flash, which is connected to the fabric I/Os. Another important component is the PolarFire clock conditioning circuitry (CCC). This clock will generate a 111.11 MHz clock to the Processor we created before. The monitor will be used to get the status of device initialization including the LSRAM initialization, the monitor is done by creating a component called INIT\_Monitor from the catalog. This needs to be configured by enabling the BANK1\_CALIB\_STATUS and BANK6\_CALIB\_STATUS. Both banks need to be enabled if not when running the timing analysis in libero it will output an error. Finally, two more components are needed, one the corereset\_pf and the coreJTAGDebug to complete the whole RISC-V subsystem. The next step before running the synthesis, some constraint files need to be imported from the design files downloaded earlier, in this case the timing\_user\_contrainis.sdc has to be selected.

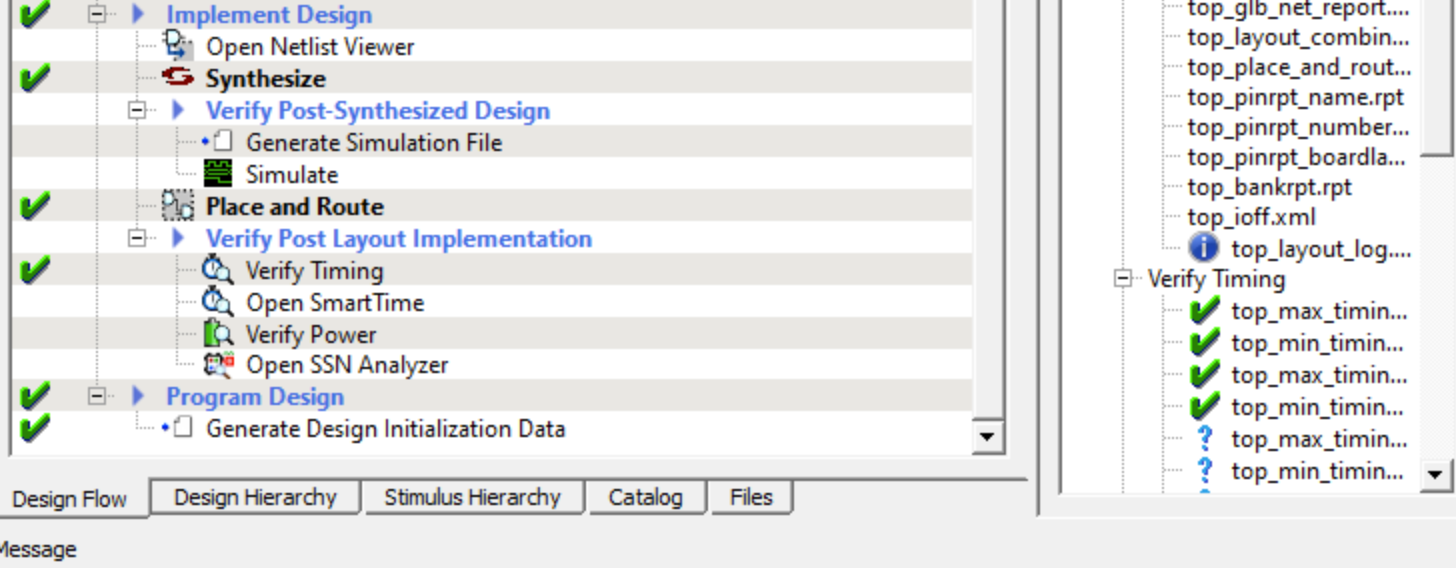
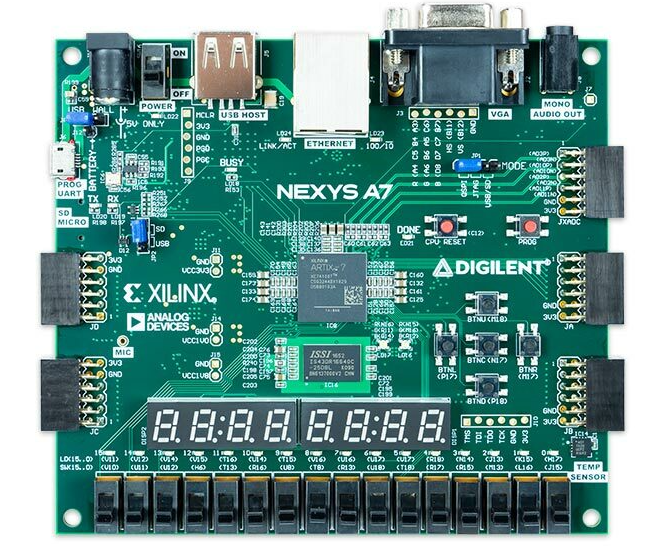


Fig. 6 Libero Design Flow

As shown on [Fig.5] this will now be allowed to run the Synthesis, Place and Route, and Timing Verification. To continue another important process that needs to be taken before generating a bitstream, is to configure the design data and memories. For this final step the MIV\_uart\_blinky.hex needs to be imported which is found under the design files download earlier and the storage types need to be SPI-flash. These previous steps must be done under the edit fabric Ram client dialog box. For the design initialization data is very important to select SPi-flash-No-binding plaintext because this ensures that the SPI Clock divider value is set to six. After done all these steps are ready to generate the bitstream and get a green check on the design flow.

1. **Nexys 4 A7**

The project would build upon the device Nexys 4 – 100T board, which contains a Xilinx XC7A100T FPGA [Fig 7], providing an excellent platform of softcore processors. The final goal of this development would be building a RISC-V processor using low-RISC implementation and fully operational in Nexys 4 DDR board.

Fig. 7 Nexys A7 Board

As the RICS-V processor is designated to compile, generate the bitstream, and upload the development in the Linux machine, visual machine and dual boot are the options to access the Linux environment. There is one software required in this project, Vivado 2018.1. In order to maintain the stability and efficiency, at least 50GB is required for the partition of the Linux machine. Before starting the process, many environments are needed to install, such as Git and Device Tree Compiler. We download the core files using Git and cloning its repository. After we get the necessary files, we need to clean the Micro SD card for partition and reformat the partition. The reason we need to do that is because the disk is required to set up the partition and establish the file system for the further use of programming on Nexys 4. Then, we connect the FPGA board with the Linux machine, and put in commands to start programming onto the device with the repository files. In the next step, the SD card that is partitioned would be inserted to the Nexys 4. We connect the serial console to the FPGA using Microcom since Linux is in use. After setting up the FPGA, we turn on the device as a result of succeeding in creating the lowRISC boot program. LowRISC contains the feature of a configured standalone that comes from SPI flash memory. Other than that, PC mimicking and remote serial console are the features of lowRISC.

**References**

[1] L. Miculka, M. Straka and Z. Kotasek, "Methodology for Fault Tolerant System Design Based on FPGA into Limited Redundant Area," 2013 Euromicro Conference on Digital System Design, Los Alamitos, CA, 2013, pp. 227-234.

[2] G.-H. Asadi and M. B. Tahoori, "Soft error mitigation for sram-based fpgas," in Proceedings of the 23rd IEEE Symposium on VLSI Test, ser. VTS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 207-212.

[3] A. Gavros, H. H. Loomis and A. A. Ross, "Reduced Precision Redundancy in a Radix-4 FFT implementation on a Field Programmable Gate Array," 2011 Aerospace Conference, Big Sky, MT, 2011, pp. 1-15.

[4] T. Wilmshurst, “Building Assembler programs,” Designing Embedded Systems with PIC Microcontrollers, pp. 107–144, 2010.

[5] R. Yao, Q. Chen, Z. Li, and Y. Sun, “Multi-objective evolutionary design of selective triple modular redundancy systems against SEUs,” Chinese Journal of Aeronautics, vol. 28, no. 3, pp. 804–813, 2015.

[6] F. Yan, Y. Tan, Q. Zhang, F. Wu, Z. Cheng and J. Zheng, "An Effective RAID Data Layout for Object-Based Deduplication Backup System," in Chinese Journal of Electronics, vol. 25, no. 5, pp. 832-840, 9 2016,

[7] C. Rossmeissl, A. Sreeramareddy and A. Akoglu, "Partial Bitstream 2-D Core Relocation for Reconfigurable Architectures," 2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, 2009, pp. 98-105.

[8] F. Hatori et al., "Introducing redundancy in field programmable gate arrays," Proceedings of IEEE Custom Integrated Circuits Conference - CICC '93, San Diego, CA, USA, 1993, pp. 7.1.1-7.1.4.

[9] F. Verbeek and J. Schmaltz, “Towards the Formal Verification of Cache Coherency at the Architectural Level,” ACM Transactions on Design Automation of Electronic Systems, 2012, pp. 1-16.

[10] M. D. Hill, D. Christie, D. Patterson, J. J. Yi, D. Chiou

and R. Sendag, "Proprietary versus Open

Instruction Sets," in IEEE Micro, vol. 36, no. 4, pp.

58-68, July-Aug. 2016, doi: 10.1109/MM.2016.61.