Designing a Microcontroller Using RISC-V in Triple Modular Redundancy for NASA'S Caution & Warning System

Authors & Designers:

James Thomas,

Sam Bagley,

Michael Ashford,

Max Bakes,

Zac Carico

Index

I. List of Figures
II. Acronyms
III. Mission Statement
IV. Design Specifications
    a. Required Specifications
    b. Optional Specifications
V. Project Report
    a. Microcontroller Design
        i. LVDS Full-Duplex and Normal UARTs
        ii. SPI
        iii. I2C
        iv. GPIO
        v. RISC-V Implementations
    b. PCB Design
    c. Drivers and UI Design
VI. Conclusion

List of Figures

Figure 1: Base LVDS UART Module

Figure 2: SPI Module

Figure 3: Core I2C Module

Figure 4: Bi-directional Buffer for SDA and SCL

Figure 5: I2C SDA Output

Figure 6: Core GPIO Module and GPIO Top Module

Figure 7: Bi-directional Buffer for GPIO

Acronyms

CWS- Caution and Warning System

EMU- Extravehicular Mobility Unit

FIFO- First In First Out

FMC- FPGA Mezzanine Card

FPGA- Field Programmable Gate Array

HPC- High Pin Count

IP- Intellectual Property

LPC- Low Pin Count

LVDS- Low Voltage Differential Signaling

NASA- National Aeronautics and Space Administration

PLSS- Portable Life Support System

RISC- Reduced Instruction Set Computer

TMR- Triple Modular Redundancy

xEMU- Exploratory EMU

Mission Statement

Create and test a microcontroller using TMR with various configurations of the RISC-V open source instruction set on Microsemi's PolarFire radiation tolerant FPGA. This is to test the possibility of using RISC-V architecture in space, specifically the possibility of implementing this system in the CWS in NASA's newest space suit, the xEMU.

Design Specifications

Required Specifications

- Use VHDL, not Verilog/System Verilog
- Multiple configurations of a RISC-V core (at least 3)
- Create a program to benchmark and test RISC-V cores
- Microcontroller architecture with the following features:
    - 10 LVDS Full-Duplex UARTs
    - TMR
    - GPIO pins
    - SPI
    - I2C
- PCB that connects to the PolarFire's FMC HPC with the following features:
    - LCD screen
    - 3-position toggle switch
    - Ports for Full-Duplex LVDS UART using the provided FIFOs
    - LEDs and Switches for testing
    - ADC

Optional Specifications

- Program to test all microcontroller and PCB features
- Interface with program over FPGA's micro USB port
- Microcontroller architecture:
    - PWM signals
    - Normal UART
    - UART over FPGA's micro USB port
    - GPIO interrupts
- PCB:
    - Pressure and humidity sensor
    - Heartrate sensor
    - Accelerometer
    - 12v, 5v, and 3.3v ports
    - SPI, I2C, UART ports

Project Report

Unfortunately, there were some unforeseen setbacks, hindering the amount of time we could devote to designing and testing. The computers needed more memory to synthesize our design, Sophos anti-virus software and Windows Defender constantly removed files Libero needed to synthesize the design and run place and route. The school was also closed due to a pandemic.

We were, however, able to create a solid foundation for next semester's students to be successful in continuing to design and test the hardware, software, and PCB.

Microcontroller Design

Libero's smart design was used for majority of the project, due to the available IPs we could use. Using a tutorial on how to implement a RISC-V core on the PolarFire FPGA, we created a basic core, implemented TMR, and started to implement the features needed to create one completed and fully featured microcontroller. To answer any questions about how the core was configured, or how to connect to the microcontroller over the micro USB port, see the tutorial we followed here. The tutorial also goes over how to program the microcontroller and set it up for debugging in Soft Console.

With what time we were able to spend designing the microcontroller, a great deal of progress was made, but not fully tested. We were able to get one complete core working in TMR. Implement SPI, I2C, and UART over the FPGA's micro USB port, GPIO, and somewhat LVDS UART. Some of these features were tested using 6 GPIO pins located on the PolarFire development board, but very few features could be completely tested.

LVDS Full-Duplex and Normal UARTs

The UART over the FPGA's micro USB port is fully functional. Using PuTTY, you can connect to the microcontroller.

As for the LVDS Full-Duplex UART, we created a design that could possibly work. We tried to implement the FIFO into the bus interface, but don't know if we were able to insert it correctly. To create the UART, we used a normal UART IP with a LVDS IP specific to the PolarFire FPGA (see figure below).

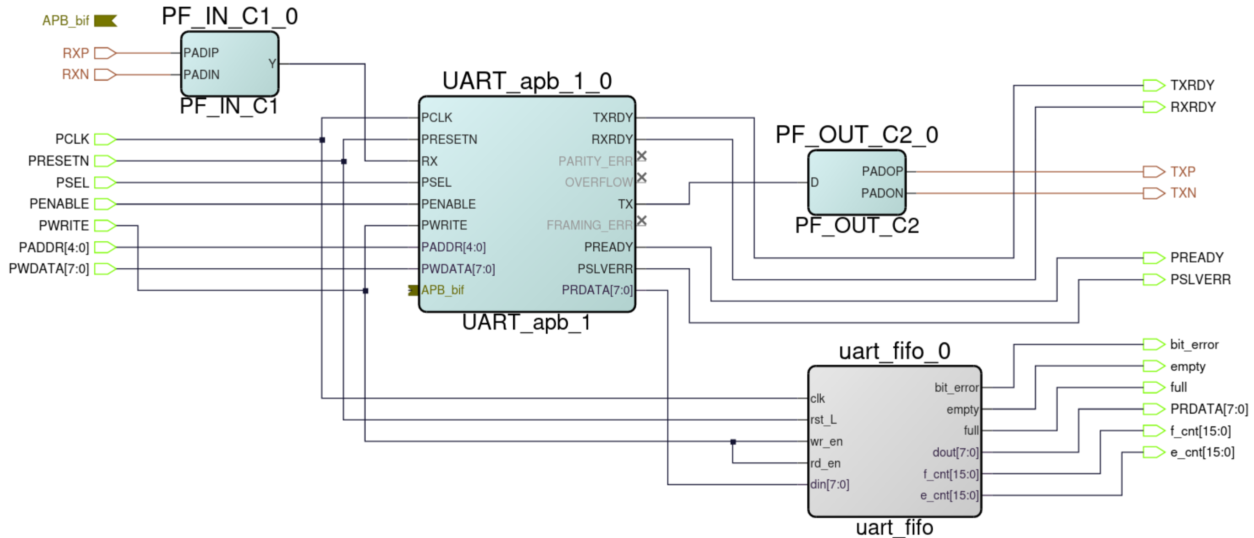

Figure 1: *Implementation of a single LVDS UART Module*

Unfortunately, the LVDS IP doesn't have the ability to pick what port it outputs to on the FPGA. This kept us from being able to test this feature.

SPI

The SPI module is an IP that was implemented when following the tutorial. It has been changed to include five extra SPI select pins to be used for a couple SPI devices on the PCB, and for some external SPI connections.

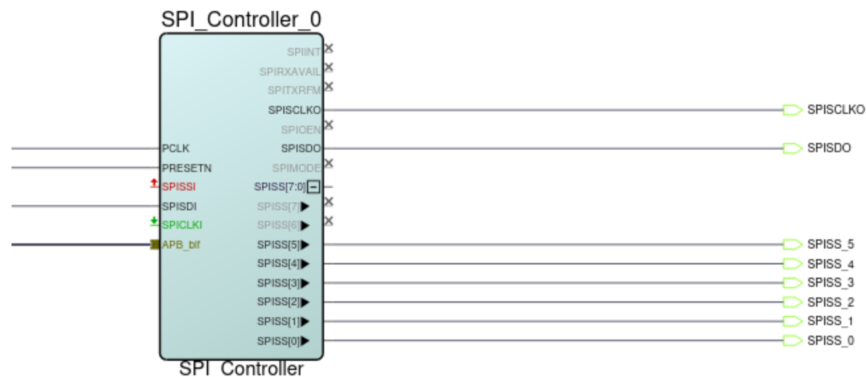

Figure 2: *Smart Design implementation of the SPI module*

"SSISS[0]" in the figure above connects to the FPGA's SPI FLASH, which is what is flashed with the program for the processor to run. Limited testing was done for the SPI, but "SPISCLKO" would activate when sending data over SPI.

I2C

The I2C communication capabilities have been added into CPU design using a CoreI2C module. This is connected to the processor via the Advanced Peripheral Bus (APB). This module currently uses a single I2C channel set to Full Master RX/TX mode.

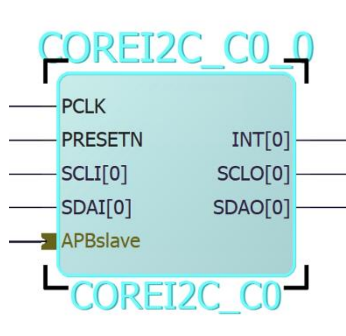

Figure 3: *CORE I2C IP Module added to the project*

Because this module has a separate line for input and output of the serial clock and the serial data, bi-directional buffers are used to combine them. This allows for bi-directional communication using only 2 pins.

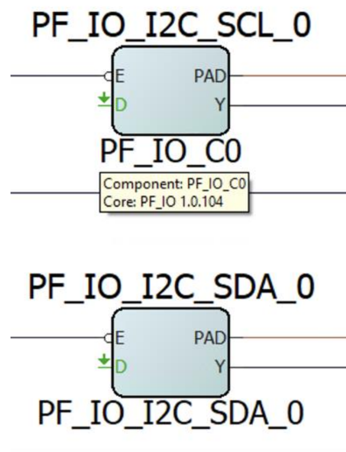

Figure 4: Bi-directional buffers combining inputs and output lines of I2C SDA and SCL

This I2C implementation was able to be tested to a small degree. It was verified to produce output on the SDA and SCL pins that it was assigned to. This is the extent that was able to be tested before the quarantine went into effect, so more work will probably be needed to properly configure the I2C module to communicate with the sensors.

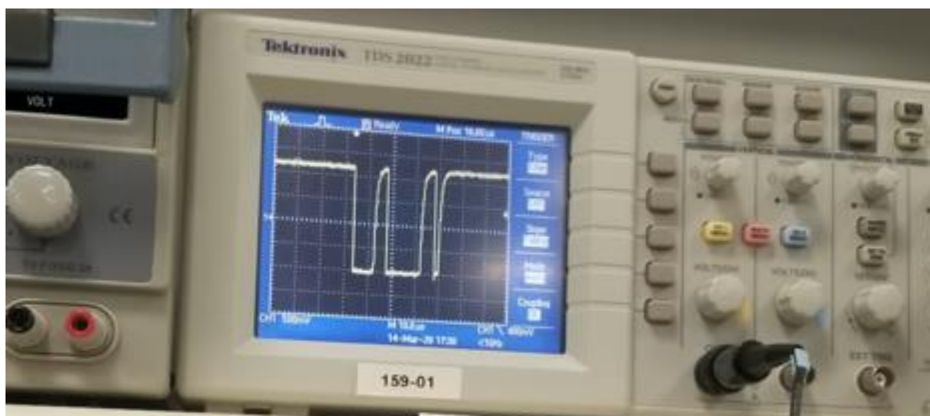

Figure 5: I2C SDA output viewed on an oscilloscope

GPIO

The GPIO pins initially came from the tutorial also, but have been modified to include 32 GPIO ports, both input and output (so they can be configured in software), be able to send interrupts signals to the ISR.

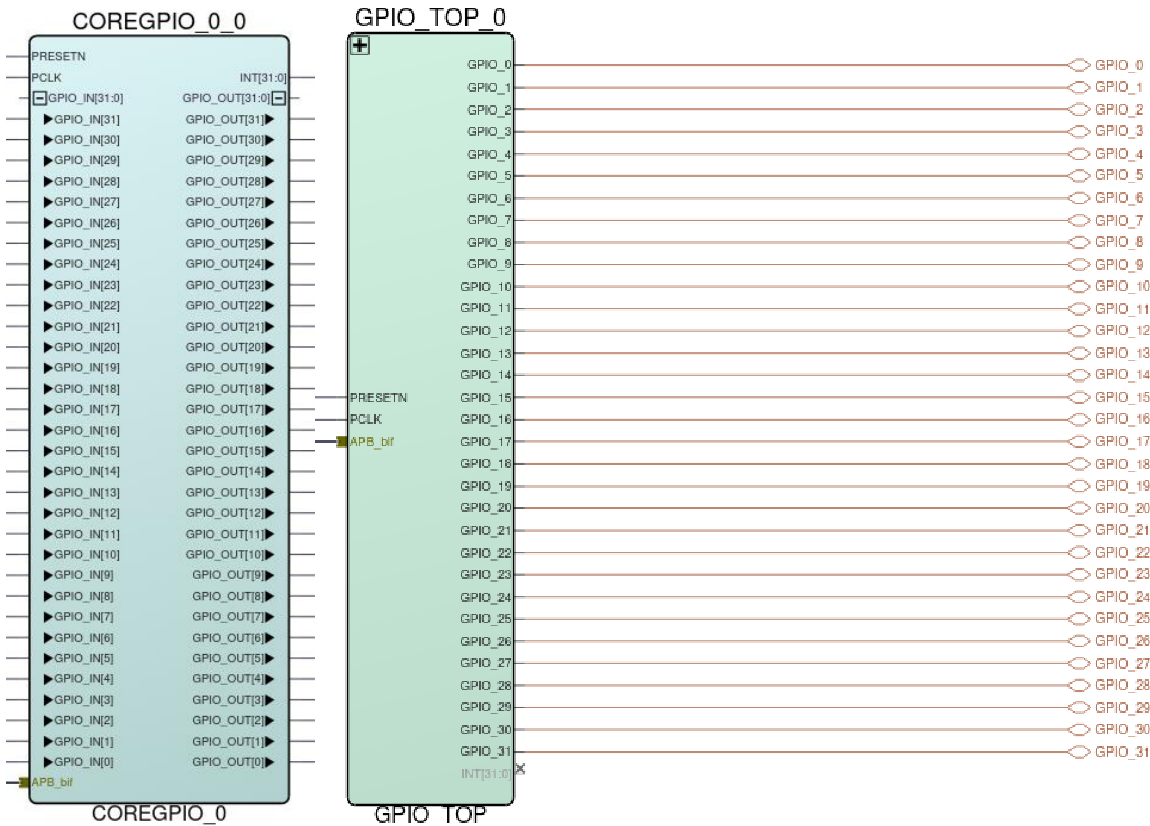

Figure 6: *CORE GPIO IP Module (left) and GPIO Top Module (right)*

The GPIO pins were set to both input and output and the interrupt settings were set in the settings of the Core GPIO IP module. To give them the ability to be both an input and an output, a bi-directional buffer had to be added to each GPIO port (this buffer is also used for the I2C IP). To make things look cleaner in the top Smart Design, all of this was implemented in a sub-module call GPIO_TOP.

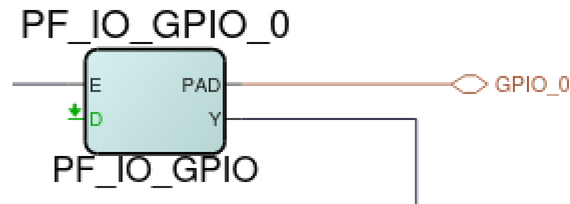

Figure 7: *Bi-directional Buffer IP*

Not all aspects of the GPIO ports were tested. The tutorial had them set as output only pins with no interrupts attached. This worked in Software, but no testing has been done to see if the changes listed still work as expected.

RISC-V Implementations

We have the solution for the tutorial we followed implemented as one of our cores. Once we had a solid grasp on the CPU design process, we started from this solution and made the Solution – Copy project. That copy was fully modified to interface with the PCB and should serve as a point of reference if any problems arise. The finished copy was then saved as a separate project for more concise documentation. It was named Solution – MIV_AXI. The three other processor designs are based on this Solution – MIV_AXI. Solution – CoreRISCV replaces the processor in Solution – MIV_AXI with the CoreRISCV processor. MIV_FP and MIV_AHB are alternate processors substituted into Solution – MIV_AXI with one key difference, these two use AHB memory controllers and needed some adaptation to work with the onboard memory (they are close to completion but do require more effort yet to finish them off). All four of these last designs (the three MIV and the one CoreRISCV processors) need to be tested and then they can be benchmarked to see which one works the best.

A more thorough explanation can be found here.

PCB Design

The design of the PCB began with a selection of different sensors that we wanted to test. The sensors that were chosen are a heartrate, pressure and humidity, and accelerometer sensors. Along with the sensors there were some additional necessities to add to the design. The addition of 3.3, 5, and 12 Voltage ports and ports for SPI, I2C, and UART for communication. In order to use information gained from the sensors an ADC was also needed. Also needed are GPIO pins, an LCD screen, switches with an associated LED, an FMC connector, and a toggle switch.

With the components picked, the next step was to find the models and footprints to import into Altium. When all the components had their designated footprint, we could begin putting together all the needed schematics. By reviewing the specifications for a given part, interconnecting the different parts was quite easy. When each part was fully connected the last thing to be done was the routing.

The process of routing the PCB was not an easy one. While adding more layers to the board can make routing significantly easier, it also makes it more expensive. We were determined to route everything on 2 layers, leaving 2 for a power and ground plane. One issue that was prominent came from the FMC connector. With 400 pins and several and limited space, careful design was needed to get all the traces to connect. With routing complete the only steps left were to create a solder mask and create a pour. When these steps were complete, we shipped the board to get milled.

Drivers and UI Design

Once a RISC-V core is built and loaded onto the FPGA, we can use SoftConsole, a software package to program softcores, to run our custom drivers and software on our cores. This is how we set up the interface with the PCB and how we planned on benchmarking the processors.

We started by building a general user interface that worked over the JTAG debugging UART to send messages back and forth with a Putty window. On FPGA startup, we could see the intro message displayed by our processor and then it would display a menu of possible options. We created drivers to interface with each of our custom sensors as well as any communications protocols. We created drivers that would allow our processors to communicate over SPI and I2C with sensors already on the board, as well as any additional sensors or devices added later on using the external pins on our PCB. We created code so that the LCD screen could be initialized and then display custom messages or a simple test image.

Some of these driver modules need to be completed still, and all of them need to be tested with the PCB. No code has been included to leverage the ADC on the PCB, but individual sensor drivers could be added that take the output from the ADC and convert it into intelligible information. Doxygen code has been added so that our code becomes clearer. Some of the documentation outlines which parts are yet to be done and need some tender love and care.

These efforts were mostly setting up the stage for a later user interface we had planning on designing. NASA wanted us to make a simple interface using the LCD screen and the paddle switch. The FPGA and the PCB together would form a standalone unit. The LCD would display a sensor name and the value currently measured by it. Flipping the paddle switch up or down would send interrupt signals to the processor, prompting it to switch which sensor was currently displayed. The sensors could, for example, be placed into a list that would be iterated through using the paddle switch. The values for each could be held by registers that were updated by the processors. Every small-time interval, those registers could all be updated to their current values, and then whichever sensor in the list was currently selected, its real time value could be refreshed by the LCD so that all displayed information was up to date.

Conclusion

The goal of this project was to implement RISC-V cores on the PolarFire FPGA, set them up in a TMR configuration, and test them. The testing would be done both by a benchmarking program and by a PCB with sensors using the different communication protocols implemented.

Although there were a few setbacks, we were able to get a major portion of the project done. One complete core set up with TMR was created with two other cores close to completion. A PCB was designed and manufactured and can quickly be ready for testing with the FPGA. The software is ready to be benchmarked.

With the groundwork we have laid, future teams will now be able to fully test the system with minimal effort and then improve the designs we have made. The additional RISC-V core configurations can also be completed and tested to provide data sets for each configuration.