Reconfigurable, Fixed-Point Processor in VHDL

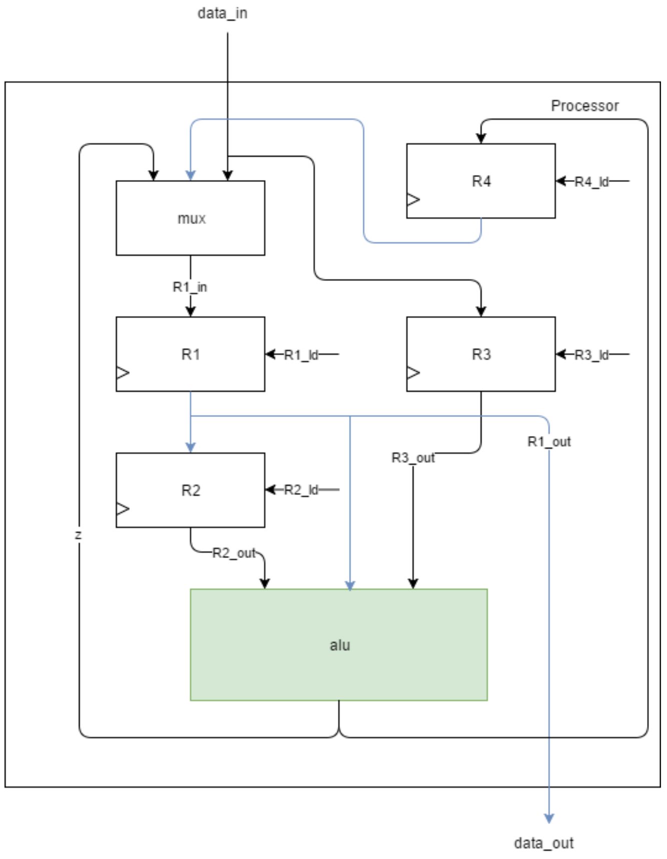
David Stern, Brandon Busuttil
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
destern@oakland.edu, bbusuttil@oakland.edu

I. INTRODUCTION

Partial reconfiguration of hardware presents many benefits such as the following: shorter reconfiguration times, increased system performance, the ability to change hardware, and more [1]. These benefits are important for applications that have limited resources such as hardware space or available power. This paper will outline a hardware implementation that demonstrates some of these benefits. The functionality of the hardware is a 16-bit, fixed-point processor. The processor's ALU will be partially reconfigurable so that it is capable of up to four instructions at a time. Only being capable of a small handful of instructions at a time, hardware space for the processor will be limited. Instructions and data will be received and sent between the processor and software using an AXI-Full interface.

II. METHODOLOGY

Hardware design and implementation were done using Vivado 2018.2. The software used for programming the FPGA and the on-chip processor was SDK 2018.2.



A. Hardware Implementation

The most important design components for the hardware implementation is the processor (**Figure 1**) and controller components. Combined, these components handle incoming data by performing different functions and then outputting results. This processor component will be wrapped with an AXI interface and packaged as an IP core.

Depending on the opcodes received, the controller component will control the data path in the processor component. The design of the processor component allows for a wide variety of functions to be implemented. The functions that were implemented can be seen in **Table 1**. For a given function to be used, the ALU must be set to the configuration that has the given function included. The ALU will be the reconfigurable module in the design. It will have two configurations, ADVANCED and BASIC, denoted A and B. In the BASIC configuration mode, the operations are rather simple such as A + B, A – B, A * B, or A / B, and in ADVANCED configuration the ALU will except the same OPCODES but perform different operations such as Cordic and shift operations.

Figure 1: Block diagram of the processor component. The block shaded green is partially reconfigurable.

Config	Function	Opcode	Comment
Advanced & Basic	Load Reg X	0x0100xxxx	Lower 16 data
Advanced & Basic	Load Reg Y	0x02000000	X => Y
Advanced & Basic	Load Reg Z	0x030xxxxx	Cordic only. Bit 16 denotes rotational or vectoring mode.
Advanced & Basic	Reset RM	0xAAAA7777	Reset ALU after DPR
Advanced & Basic	End program	0xFFFFFFFF	Command indicating end of program
Basic	x + y	0xA0000000	
Basic	x – y	0xB0000000	
Basic	x / y	0xC0000000	
Basic	x * y	0xD0000000	
Advanced	cordic (x, y, z)	0xA1000000	
Advanced	Shift Reg 1	0xB0000000	
Advanced	Shift Reg 2	0xC0000000	
Advanced	N/A	N/A	

Table 1: Instructions implemented in the processor component.

After the controller and processor are packaged together in a top-level file, an IP will be generated. This IP will be added to a block design that utilizes an AXI-Full interface.

The ALU portion of the processor will contain a special generic string parameter to indicate what type of circuit it needs to generate. Two implementations will be made, and a check will be done on the string parameter to determine which configuration to generate. When developing a system with partial reconfiguration, it is important to synthesize a static design which contains a black box of the reconfigurable partition. For our design, it will be imperative that when the static design is generated it is generated with the ALU in ADVANCED configuration. This is because the ADVANCED mode contains a much larger circuit to implement the Cordic functionality.

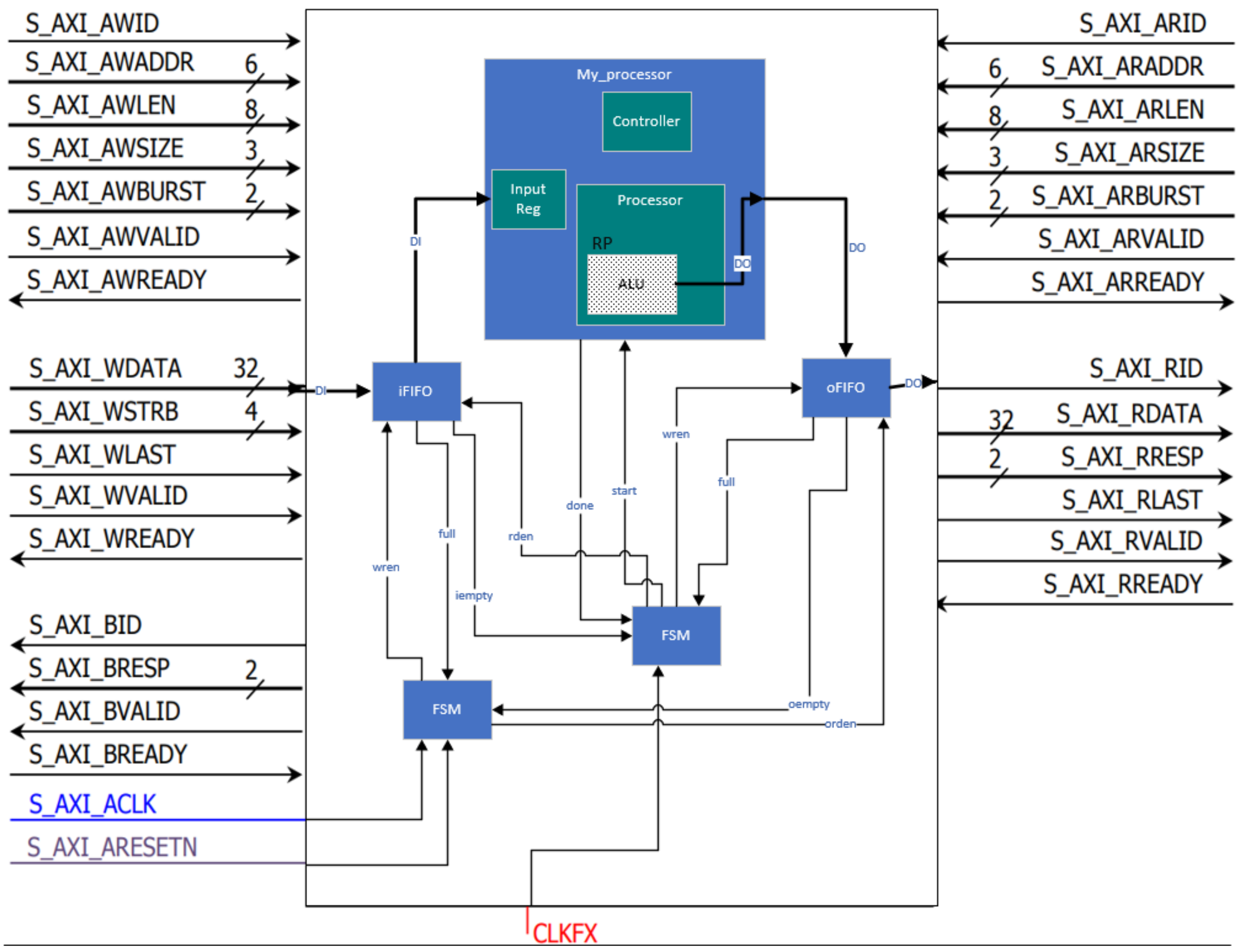


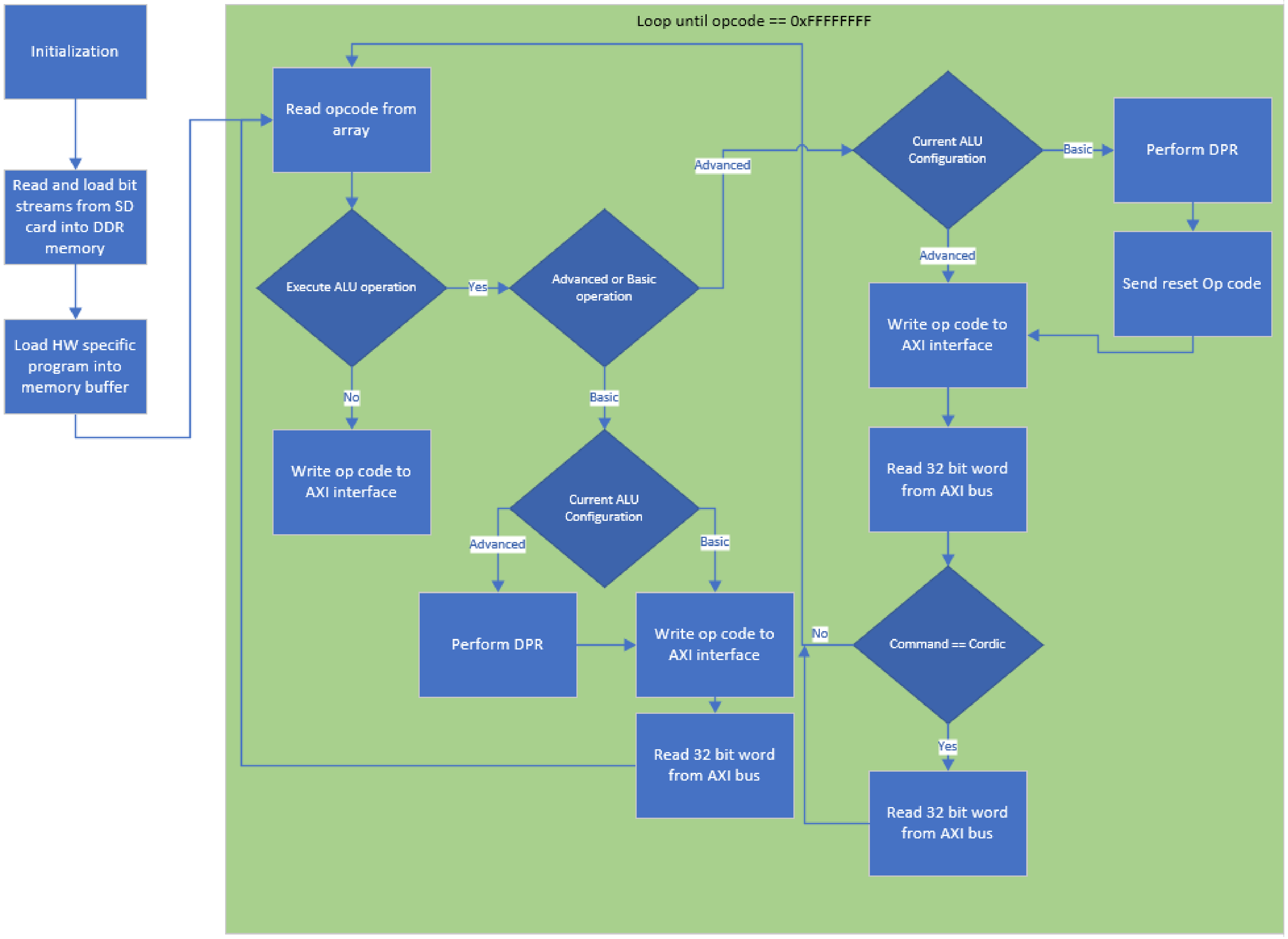
Figure 2: Block diagram of the AXI Full IP Core

B. *Software Implementation*

The primary goal of the software implementation will be exercising the hardware and prove out the design. Therefore, the software will be a simple flow of data exchanges between the PL and PS that execute the various operations on the processor. The flow of data exchanges will be pre-defined in the form of a manually generated HW specific program loaded into program memory.

Input data and instructions will be sent to the FPGA. 16-bit values will be sent to the processor one at a time. The first value to expect would be loading register one with some data. Afterwards the instructions are sent in an order that makes sense. Instructions can be sent from the software either all at once or one at a time because they will be buffered into an input FIFO on the FPGA. The controller will grab instructions and data from the input FIFO when it is ready for the next instruction. the output of the circuit is also buffered into an output FIFO which is read over the AXI bus. It is important that the software is conscious of how many and how fast it writes instructions to the processor as to not overload the input FIFO. The software also must be aware of when it should expect data. For example, after instructing the FPGA to perform an ALU computation, the software must read the data from the output FIFO over AXI bus as to not let the output FIFO overload with data.

The software will keep track of the current configuration of the ALU in hardware. On powerup the system will be configured for BASIC mode. Before sending an instruction to the processor that includes using the ALU, there is a check to make sure the function is currently in the ALU. If a function is not currently in the ALU, the software will initiate a dynamic partial reconfiguration (DPR) of the ALU to the correct mode that has the desired operation. Upon initialization of the software, two bit streams will be loaded into DDR memory from an external SD card. These two bit streams will be the ADVANCED and BASIC configurations of the ALU. Various FFS and external C libraries will be leveraged to read the data from the SD card into DDR memory. **Figure 2** shows an architectural view of the software design.



Initialization
Read and load bit streams from SD card into DDR memory
Load HW specific program into memory buffer
Loop until opcode == 0xFFFFFFFF
Read opcode from array
Execute ALU operation
Yes
No
Write op code to AXI interface
Advanced or Basic operation
Advanced
Basic
Current ALU Configuration
Advanced
Basic
Perform DPR
Write op code to AXI interface
Read 32 bit word from AXI bus
Current ALU Configuration
Basic
Perform DPR
Send reset Op code
Advanced
Write op code to AXI interface
Read 32 bit word from AXI bus
Command == Cordic
No
Yes
Read 32 bit word from AXI bus

Figure 2: Software Architectural Design

III. Experimental Setup

Experimental tests will be performed iteratively throughout the development phase of the project.

The first milestone will be to test and verify our processor IP core. Using a Vivado Test Bench we can simulate the necessary data and signals to the processor and verify the output.

The next milestone will be to test and verify our AXI IP core using another Vivado Test Bench. After our processor core has been imported into a much larger design which implements an AXI Full interface, we will need to verify that the data can be written and retrieved properly.

After our AXI Full IP core has been validated and we are confident the hardware is implemented without any bugs, the next step will be to export the bit stream and develop an SDK project to test the AXI IP core on the target hardware. Both ADVANCED and BASIC configurations will be verified here before moving onto partial reconfiguration.

Finally, the setup that will be used to verify the proper functionality of the reconfigurable FP processor will consist of a Zybo board, USB cable, and laptop computer running Xilinx's SDK 2018.2 integrated development environment. Using this setup, we will be able to read and write over the AXI stream and control the configuration of the hardware. The SDK will be configured properly to support dynamic reconfiguration of the FPGA. A C program will be developed that exercises the full functionality of our design. To verify a specific functionality, the C program will write known inputs to the processor and we will expect to receive the proper outputs over the AXI bus. Outputs will be viewed in debug mode and/or printed to the console window of the SDK.

IV. Results

List all results you obtained, For example: audiovisual results, results in an oscilloscope, etc. You can include pictures and/or links to video of your project functioning.

Include some discussion of your findings and relate them to the topic learnt in class. Were the results what you expected? In what cases are the results explainable, and in what cases unexplainable (if any)?

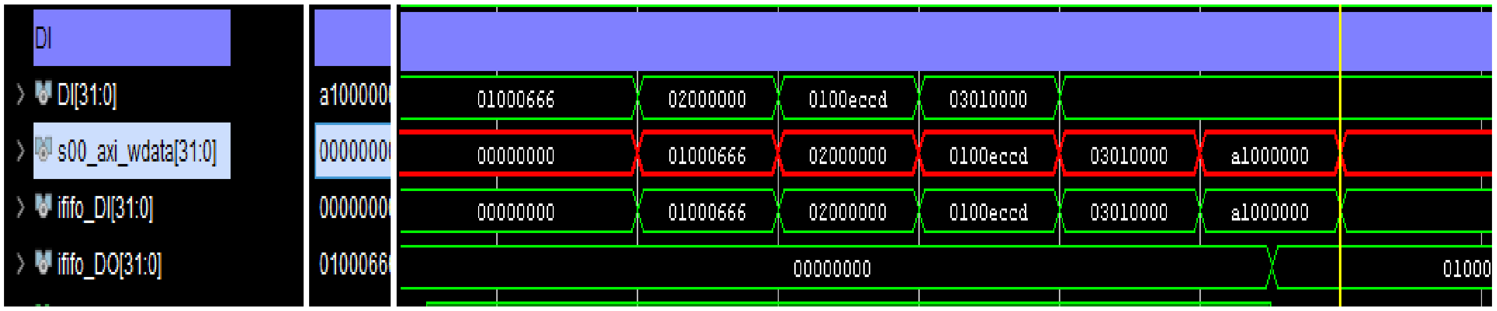


Figure 3: AXI Testing Results, s00_axi_wdata

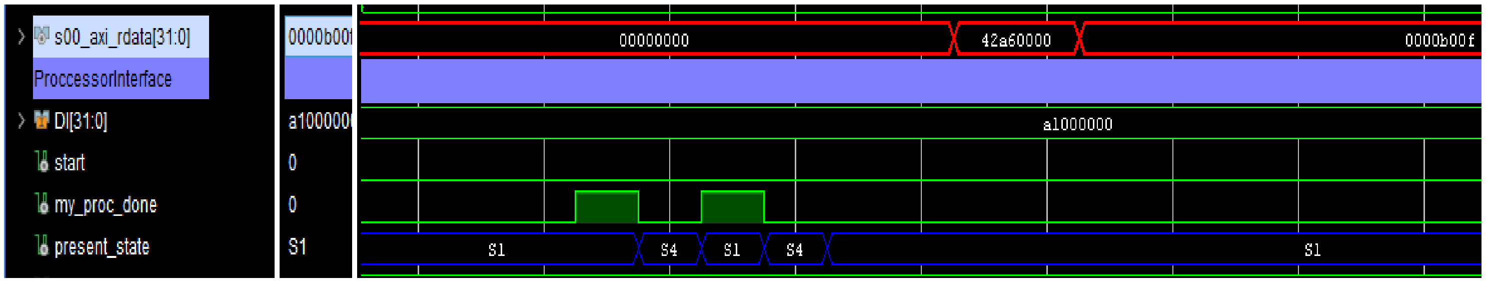


Figure 4: AXI Testing Results, s00_axi_rdata

Conclusions

Designing and implementing this 16-bit fixed point reconfigurable processor was a nice challenge for the team that really exercised some of the concepts learned throughout the semester, including some of the concepts we did not have time to implement in lab such as a hardware reset initiated after DPR by the software. Designing a simple processor, controller and data path, was something we felt moderately confident in accomplishing and was an interesting challenge for the team. Identifying and understanding our reconfigurable partition was one of the better take-aways from this project, which was different from working on the labs, where the reconfigurable partitions were already identified. In its simplest form, the processor is a great example of how partial reconfiguration can be leveraged on an embedded system where PL fabric space is very limited. Constrained by the size of your circuit, you might not have enough room to implement a generic processor, so you design a special purpose processor with a reconfigurable ALU to cover all the operations you need.

References

[1] Kao, Cindy. "Benefits of partial reconfiguration." *Xcell journal*55 (2005): 65-67.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.