Reconfigurable, Fixed-Point Processor in VHDL

DAVID STERN AND BRANDON BUSUTTIL

Overview

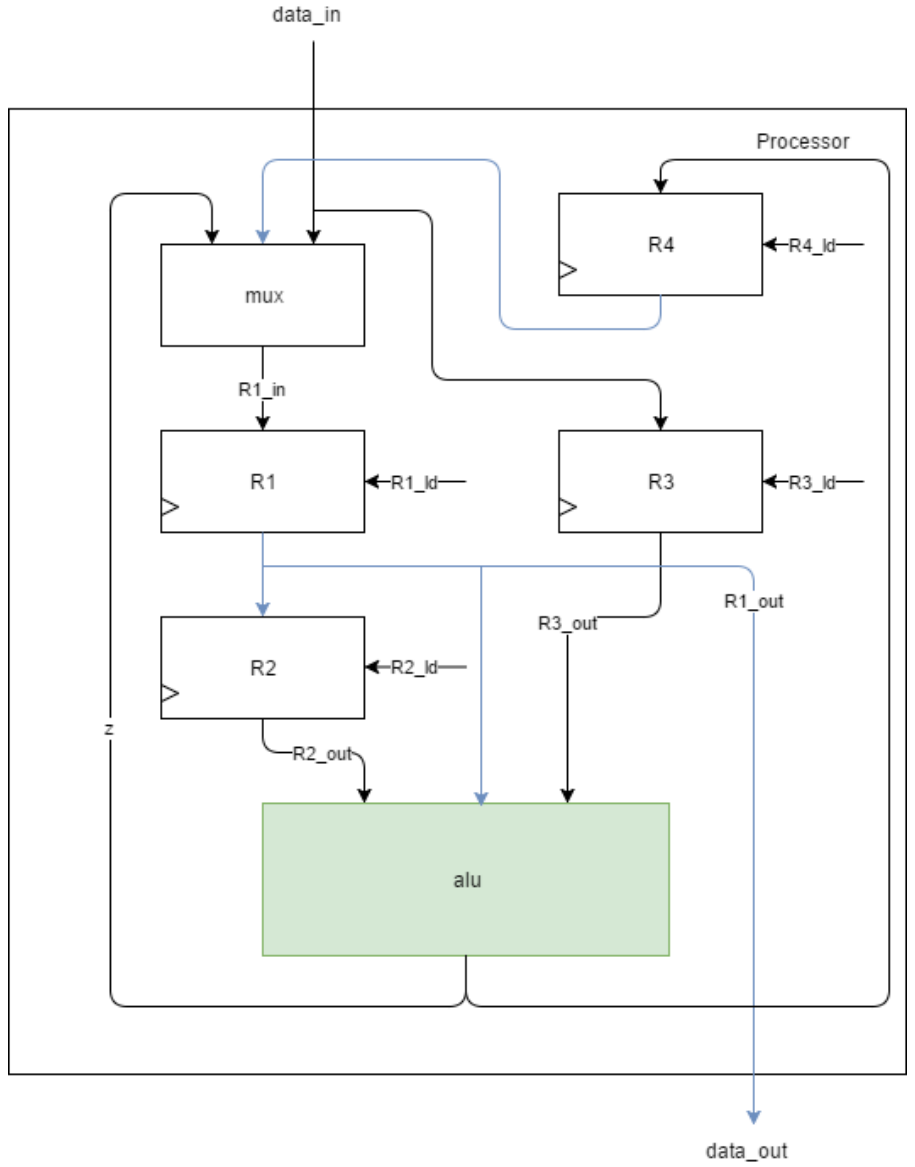
- Introduction
- Methodology
  - Hardware
  - Software
- Demonstration
- Conclusion

Introduction

- 16-bit fixed point processor with dynamically reconfigurable ALU
- Partial reconfiguration of an ALU allows your design to implement more ALU functions in a fixed amount of space
- Our design will consist of two configurations, referred to as BASIC and ADVANCED
- Basic mode will allow simple operations
- Advanced mode will allow more complex operations

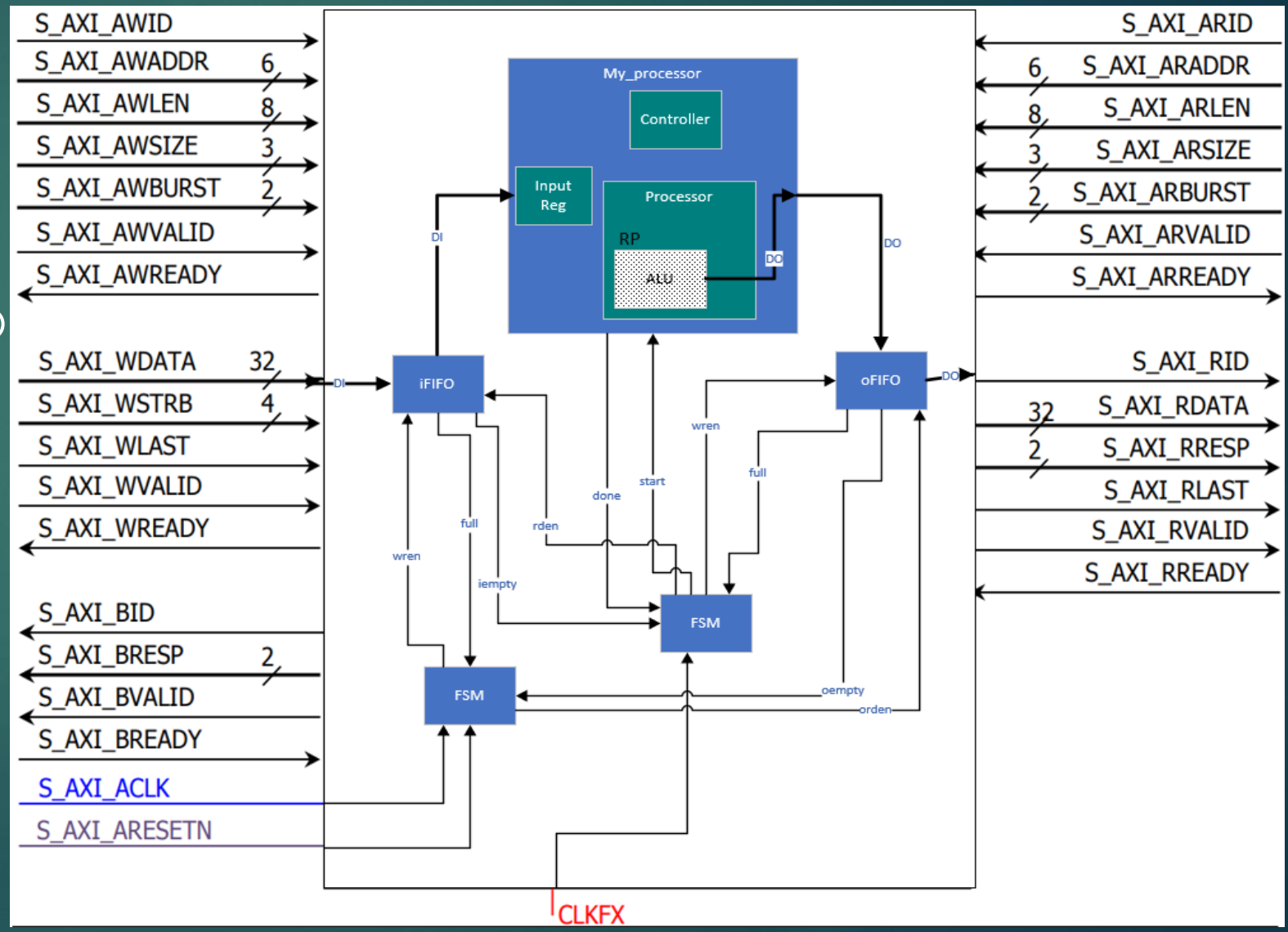
Methodology - Hardware

- The entire processor consists of a mux, four registers, ALU, and the controller
- Instructions are sent to the controller and decoded
- ALU can be reconfigured
  - Four functions per configuration



Hardware Architecture

- The processor IP core is wrapped inside an AXI Full interface.
- Data flow to the IP core is controlled through an external FSM which utilizes an input FIFO and output FIFO



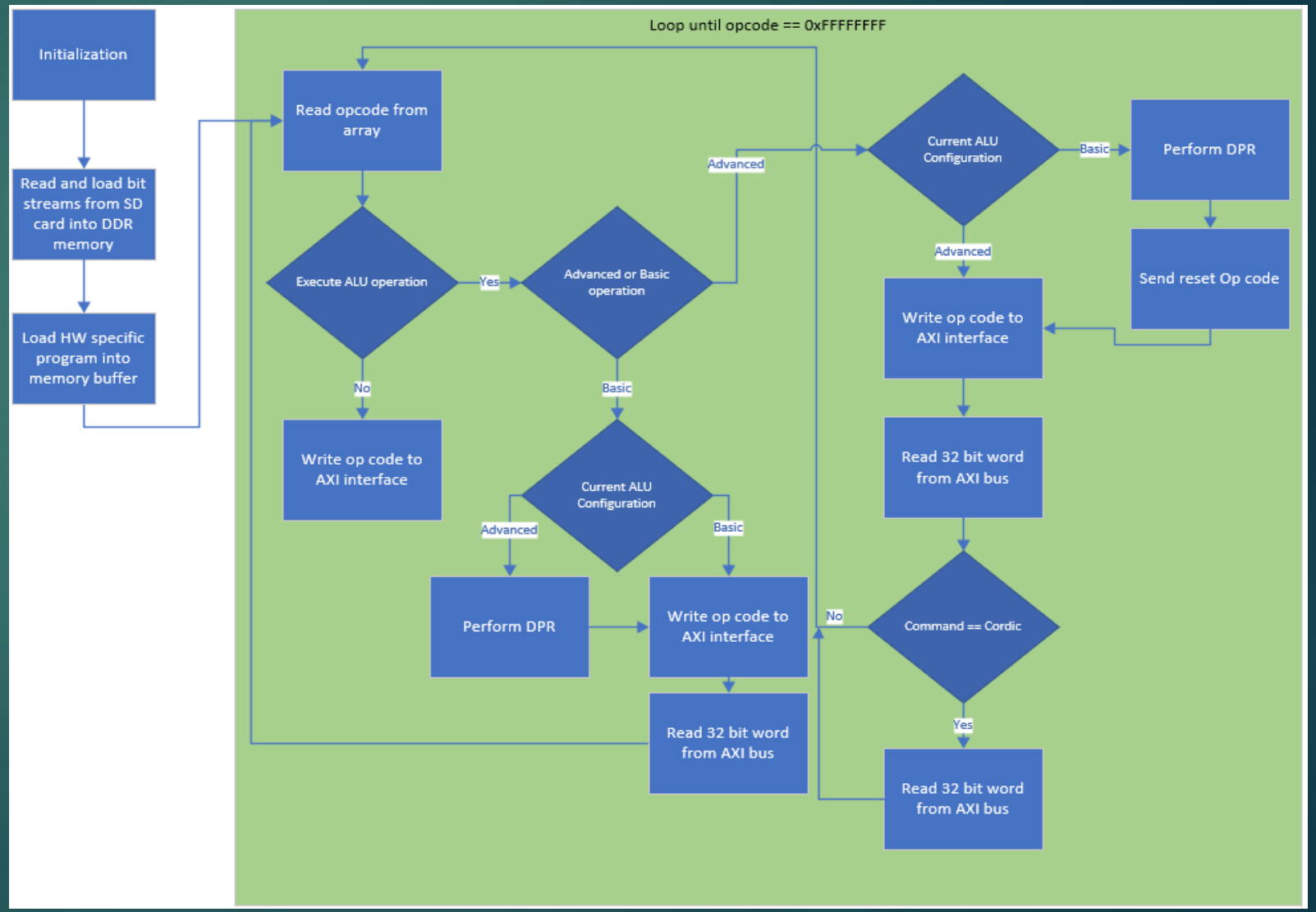
Opcodes

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	When processed by the PS, this will exit the loop and terminate the program
Basic	x + y	0xA0000000	
Basic	x – y	0xB0000000	
Basic	x * y	0xC0000000	
Basic	x / y	0xD0000000	
Advanced	cordic (x, y, z)	0xA1000000	1 Indicates the ALU must be in advanced mode
Advanced	Shift Reg 1	0xB1000000	
Advanced	Shift Reg 2	0xC1000000	
Advanced	N/A	N/A	

Methodology - Software

- Primary goal of the software is exercising the hardware to prove out the design
- Software will control data flow to and from the processor IP core
- Upon initializing:
  - The software will read two .bin files from an external SD card and load them into DDR memory
  - The software will load a predefined set of HW specific instructions into an array structure
- The array structure will be looped until the “End of Program” opcode is reached.
- Software routine keeps track of current ALU configuration. Performs DPR if necessary.

Software Architecture



Example Program

- 0x01000001: Load register 1 with 1
- 0x02000000: Duplicate register 1 into register 2
- 0x01000005: Load register 1 with 5
- 0xB0000000: Perform ALU operation Shift left X by Y
- 0x02000000: Duplicate register 1 into register 2
- 0x01000035: load register 1 with 0x0035
- 0xC0000000: Perform ALU operation X * Y
- 0x02000000: Duplicate register 1 into register 2
- 0x01001960: Load register 1 with 0x1960
- 0xA0000000: Perform ALU operation X + Y
- 0xAAAA7777: Reset ALU after reconfiguration
- 0x02000000: Duplicate register 1 into register 2
- 0x0100ECCC: Load register 1 with 0xECCC
- 0x03000000: Load register 3 with 0
- 0xA1000000: Cordic
- 0xFFFFFFFF: End of program

cordic(x, y, z)
x = .3 or 0x02000
y = -.5 or 0xECCC
z = 0 or 0x0000

Demonstration

Conclusion

- Primary goal of the software is exercising the hardware to prove out the design
- Operations implemented worked over AXI-Full and the partial bit streams
- Microprocessor design is able to handle any equation with set of operations
- Further development of functions would increase usability without needing more hardware space
- pblock placement is annoyingly difficult