ECE 5736: Reconfigurable Processor

Thomas Filarski, Kristi Stefa

Overview

A simple CPU that handles 16 bits of data and 3 instruction bits to perform operations between 3 registers

Wrapped as an AXI-4 Full Peripheral and allows PS+PL integration with C coding

Reconfigurable around one partition with two different modules, allowing for different instruction sets and architectures

Utilizes an instruction decoder so input requirements are only data+instruction

Utility of System

Useful for cases where space or signal width is limited, allowing a reconfigurable portion to be used over 2 static portions

Allows for two or more modes of operation based on the actions and instructions required

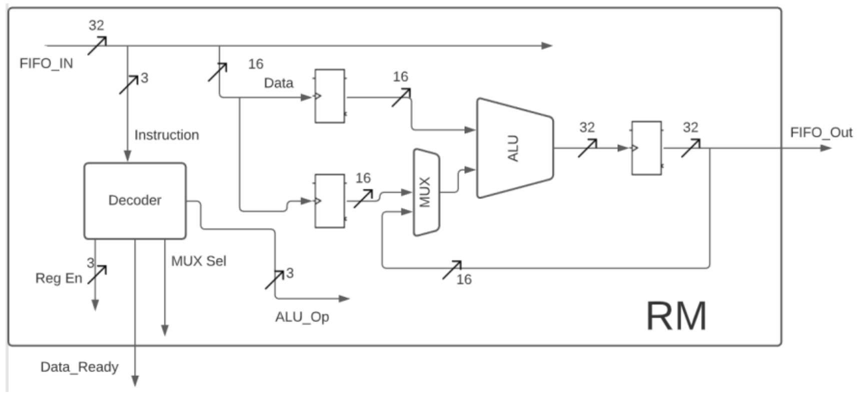
Slower than a large, static processor unit due to reconfiguration overhead but less space and the ability to be flexible at runtime

RM Design

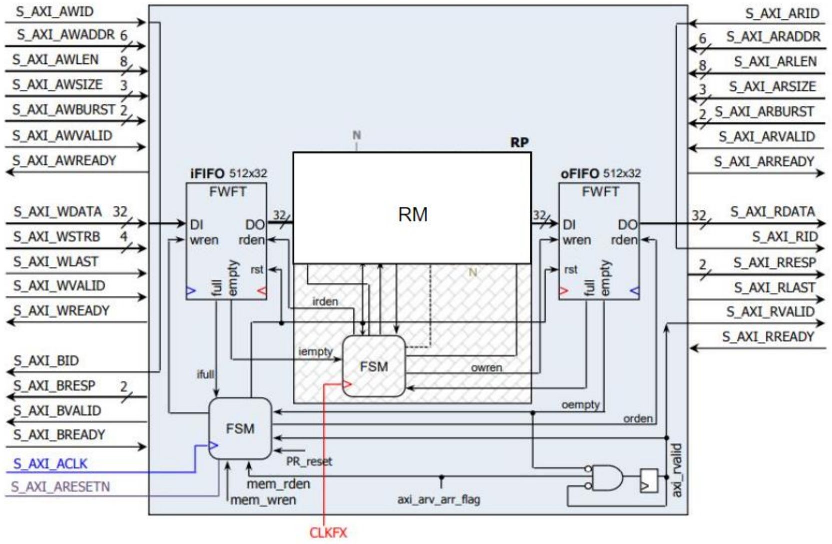
The main configurable component that holds the instruction decoder, registers, and ALU

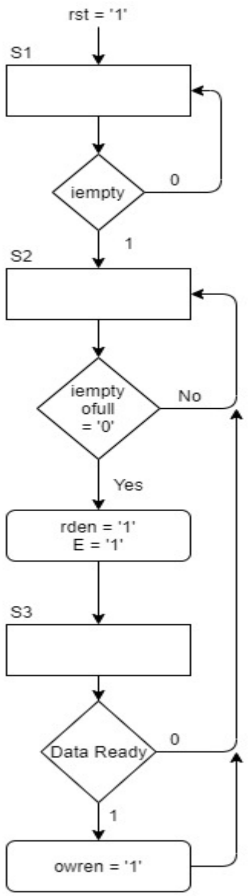
Reconfiguration comes in the decoder and ALU to handle different signals and operations

Interfaces with external FSM



Peripheral and FSM

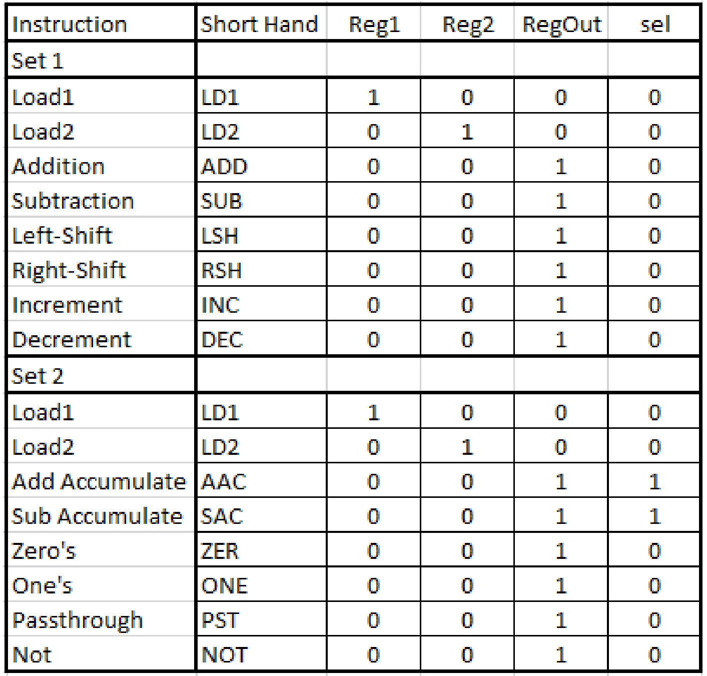




Instruction Table

The instructions were split into 2 sets and specified by their ALU operation and control signals

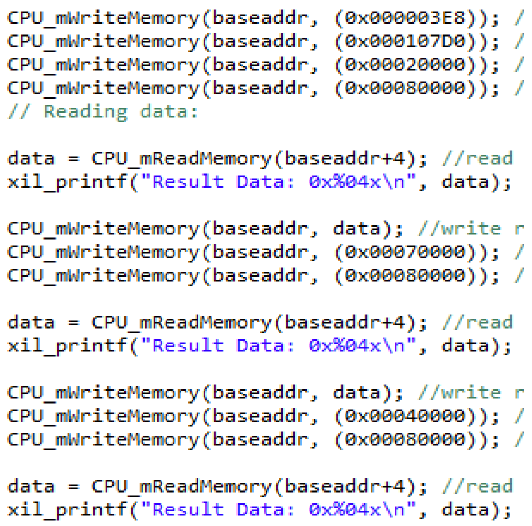
Using both sets allows for continuous transformation of data if it can be saved between reconfigs



Instruction	Short Hand	Reg1	Reg2	RegOut	sel
Set 1					
Load1	LD1	1	0	0	0
Load2	LD2	0	1	0	0
Addition	ADD	0	0	1	0
Subtraction	SUB	0	0	1	0
Left-Shift	LSH	0	0	1	0
Right-Shift	RSH	0	0	1	0
Increment	INC	0	0	1	0
Decrement	DEC	0	0	1	0
Set 2					
Load1	LD1	1	0	0	0
Load2	LD2	0	1	0	0
Add Accumulate	AAC	0	0	1	1
Sub Accumulate	SAC	0	0	1	1
Zero's	ZER	0	0	1	0
One's	ONE	0	0	1	0
Passthrough	PST	0	0	1	0
Not	NOT	0	0	1	0

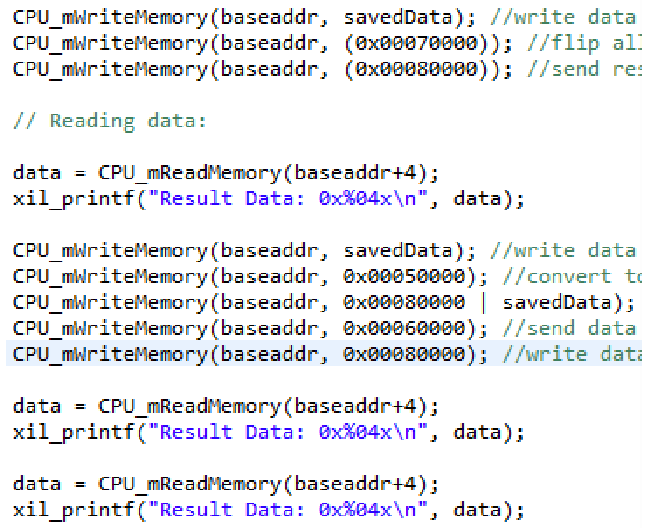
Results

First Processor



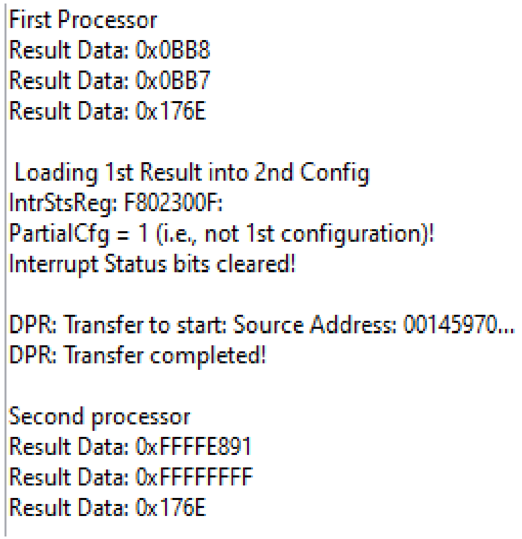
```
CPU_mWriteMemory(baseaddr, (0x000003E8)); /
CPU_mWriteMemory(baseaddr, (0x000107D0)); /
CPU_mWriteMemory(baseaddr, (0x00020000)); /
CPU_mWriteMemory(baseaddr, (0x00080000)); /
// Reading data:

data = CPU_mReadMemory(baseaddr+4); //read
xil_printf("Result Data: 0x%04x\n", data);

CPU_mWriteMemory(baseaddr, data); //write r
CPU_mWriteMemory(baseaddr, (0x00070000)); /
CPU_mWriteMemory(baseaddr, (0x00080000)); /

data = CPU_mReadMemory(baseaddr+4); //read
xil_printf("Result Data: 0x%04x\n", data);

CPU_mWriteMemory(baseaddr, data); //write r
CPU_mWriteMemory(baseaddr, (0x00040000)); /
CPU_mWriteMemory(baseaddr, (0x00080000)); /

data = CPU_mReadMemory(baseaddr+4); //read
xil_printf("Result Data: 0x%04x\n", data);
```

Second Processor



```
CPU_mWriteMemory(baseaddr, savedData); //write data
CPU_mWriteMemory(baseaddr, (0x00070000)); //flip al
CPU_mWriteMemory(baseaddr, (0x00080000)); //send re

// Reading data:

data = CPU_mReadMemory(baseaddr+4);
xil_printf("Result Data: 0x%04x\n", data);

CPU_mWriteMemory(baseaddr, savedData); //write data
CPU_mWriteMemory(baseaddr, 0x00050000); //convert t
CPU_mWriteMemory(baseaddr, 0x00080000 | savedData);
CPU_mWriteMemory(baseaddr, 0x00060000); //send data
CPU_mWriteMemory(baseaddr, 0x00080000); //write dat

data = CPU_mReadMemory(baseaddr+4);
xil_printf("Result Data: 0x%04x\n", data);

data = CPU_mReadMemory(baseaddr+4);
xil_printf("Result Data: 0x%04x\n", data);
```

Results



```
First Processor
Result Data: 0x0BB8
Result Data: 0x0BB7
Result Data: 0x176E

 Loading 1st Result into 2nd Config
IntrStsReg: F802300F:
PartialCfg = 1 (i.e., not 1st configuration)!
Interrupt Status bits cleared!

DPR: Transfer to start: Source Address: 00145970...
DPR: Transfer completed!

Second processor
Result Data: 0xFFFFE891
Result Data: 0xFFFFFFFF
Result Data: 0x176E
```

Future Work

This design could be expanded into a more complex processor with a stack to allow for less reading and writing between the software and hardware

The instruction set could be expanded from a 3-bit to a 4-bit or higher set

The software can be used with more than two processors

Demo

https://drive.google.com/file/d/1IOyDF2s5ZUMcf5NBd_SoRbNMigCVsG6_/view?usp=sharing