

CS150 Project Final Report

Project Functional Description and Design Requirements:

The objective of our project was to implement a functional three stage pipelined RISC-V CPU running 32-bit RISC-V instruction set at 100 Mhz, with two way set associative write-back instruction and data caching. The memory architecture was required to include a cache bypass and memory mapped IO for UART, cycle and instruction counters, and graphics acceleration. The graphics accelerator needed to have the capability to draw lines and fill a screen with a solid color. We expanded this feature to add the ability to draw circles, open and filled, on the screen. A sample image is attached (Appendix: Figure 3).

High-level organization:

Our design is split up into three main stages. We've provided a block diagram of our data path (Appendix: Figure 2) as well as the following descriptions of each stage.

Stage 1:

In stage one we receive the instruction from memory, analyze it, and generate control signals for the rest of the design. We also access the register file to get any necessary values, and handle data forwarding from stage 3 (two instructions away).

This uses the modules: Control, Stripper, ImmDecode, and RegFile.

Stage 2:

In stage two we execute arithmetic with the ALU, handle branching (and jumping) logic, and input to the memory mapped IO. Data forwarding from the previous instruction is received from stage 3 at the beginning of this stage.

This uses the modules: MemoryInterface, ALU, and Branch.

Stage 3:

In stage three, we receive the output from the memory or memory mapped IO and write to registers from either the ALU, memory, or the PC+4 according to the control signal. The data output of the stage is sent back to the first stage for data forwarding. If data forwarding from memory, a noop is inserted to reduce the critical path.

This uses the modules: MemoryInterface, and RegFile.

PC Logic:

The PC is normally incremented by 4 every cycle. On a stall, the PC value is held the same and on reset, the PC is set to 0x40000000 (the start of BIOS memory). On branches, it is set to the address computed by the ALU and there is additional logic to stall the PC when noops are inserted when data forwarding from memory.

Detailed Description of Sub-pieces:

Control:

Takes in fields of the instruction from the Stripper and outputs all control signals based on only the current instruction except address based control signals used in MemoryInterface because the memory address isn't known in the first stage.

Stripper:

Takes in an instruction and splits it into fields for opcode, rs1, etc. for other modules to use.

Also outputs the instruction type based on the opcode.

ImmDecode:

Takes in the current instruction and instruction type, and decodes the immediate of the current instruction sign extended to 32 bits. The immediate is stored in different bits for the different instruction types R, I, S, SB, U, and UJ.

RegFile:

Maintains the values of 32 32-bit registers. Stores values into a register synchronously and reads two values at a time asynchronously and always outputs zero for register zero.

ALU:

Takes in two 32-bit inputs and an operation type and then computes the result.

Branch:

Takes in two 32-bit values to compare, and the type of comparison to do (registered from Control) and outputs whether to branch or not. Always outputs true if the type is a jump.

MemoryInterface:

Takes in an address, and a read and write enable, and based on the top nibble of the address either accesses memory through different caches, or access peripherals of the cpu, based on the tables below.

Updated Memory Address Partitions

Address[31:28]	Address Type	Device	Access	Notes
4'b00x1	Data	Data Cache	Read/Write	
4'b0001	PC	Instruction Cache	Read-only	
4'b001x	Data	Instruction Cache	Write-Only	Only if PC[30]
4'b0100	PC	BIOS memory	Read-only	
4'b0100	Data	BIOS memory	Read-only	
4'b0100	Data	Cache Bypass	Write-only	
4'b1000	Data	I/O	Read/Write	

Memory Map

Address	Function	Access	Data Encoding
32'h80000000	UART transmitter control	Read	{30'b0, DataOutValid, DataInReady}
32'h80000004	UART receiver data	Read	{24'b0, DataOut}
32'h80000008	UART transmitter data	Write	{24'b0, DataIn}
32'h80000010	Cycle counter	Read	Total number of cycles
32'h80000014	Stall counter	Read	Number of cycles stalled
32'h80000018	Reset counters to 0	Write	N/A
32'h8000001c	Filler Control	Read	{31'b0, FillerReady}
32'h80000020	Filler Color	Write	{8'b0, Color}
32'h80000024	Line Control	Read	{31'b0, LE_ready}
32'h80000028	Line Color	Write	{8'b0, Color}
32'h80000030	Line x0	Write	{22'b0, Point}
32'h80000034	Line y0	Write	{22'b0, Point}
32'h80000038	Line x1	Write	{22'b0, Point}
32'h8000003c	Line y1	Write	{22'b0, Point}
32'h80000040	Triggering Line x0	Write	{22'b0, Point}
32'h80000044	Triggering Line y0	Write	{22'b0, Point}
32'h80000048	Triggering Line x1	Write	{22'b0, Point}
32'h8000004c	Triggering Line y1	Write	{22'b0, Point}

MemoryInterface also takes in the PC and outputs instructions using the instruction cache. It holds the outputs of the caches if they output while anything else is stalling, so at the end of the stall, all values are ready.

Status and Results:

Everything (2 way set associative write back cache, graphics accelerator, all instructions in the spec, UART, counters, and cache bypass) works at 100 Mhz. We fail two timing constraints, but they don't affect the ability of our cpu to run correctly at speed. We believe this is because the longest critical path is only used during a stall, so it actually has two cycles to stabilize.

Slice LUTs: 8162

LUT Flip Flop Pairs: 10095

For your convenience we've attached a copy of the report main screen (Appendix: Figure 1), in case we forgot any number you wished to see.

Conclusions:

This project was a challenge, but as predicted, the challenge was mostly in debugging the mistakes we made rather than in writing the code in the first place. Hence most of the lessons we learned were about how to better debug Verilog and other similar languages. Bugs are very often in the connections between pieces as opposed in the individual modules, so it would have been better if we'd checked these connections carefully in the beginning of our debugging process as opposed to after a week or two. Verilog is very bad at throwing errors for typos, and we should have created something that would show us only the errors that we had caused.

Appendix:

ml505top Project Status (12/11/2014 - 15:26:10)			
Configuration File:	ml505top.xreport	Parser Errors:	
Module Name:	ml505top	Implementation State:	Programming File Generated
Target Device:	5vix110tff1136-1	• Errors:	No Errors
Product Version:	ISE 14.6	• Warnings:	1765 Warnings (1765 new)
Design Goal:	data unavailable	• Routing Results:	All Signals Completely Routed
Design Strategy:	data unavailable	• Timing Constraints:	X 2 Failing Constraints
Environment:	System Settings	• Final Timing Score:	2063665


Device Utilization Summary					
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	5,569	69,120	8%		
Number used as Flip Flops	5,569				
Number of Slice LUTs	8,162	69,120	11%		
Number used as logic	8,113	69,120	11%		
Number using O6 output only	7,749				
Number using O5 output only	171				
Number using O5 and O6	193				
Number used as Memory	35	17,920	1%		
Number used as Shift Register	35				
Number using O6 output only	35				
Number used as exclusive route-thru	14				
Number of route-thrus	183				
Number using O6 output only	180				
Number using O5 output only	2				
Number using O5 and O6	1				
Number of occupied Slices	3,543	17,280	20%		
Number of LUT Flip Flop pairs used	10,095				
Number with an unused Flip Flop	4,526	10,095	44%		
Number with an unused LUT	1,933	10,095	19%		
Number of fully used LUT-FF pairs	3,636	10,095	36%		
Number of unique control sets	454				
Number of slice register sites lost to control set restrictions	956	69,120	1%		
Number of bonded IOBs	146	640	22%		
Number of LOCed IOBs	146	146	100%		
IOB Flip Flops	279				
Number of BlockRAM/FIFO	42	148	28%		
Number using BlockRAM only	30				
Number using FIFO only	12				
Number of 36k BlockRAM used	27				
Number of 18k BlockRAM used	5				
Number of 36k FIFO used	12				
Total Memory used (KB)	1,494	5,328	28%		
Number of BUFG/BUFGCTRLs	6	32	18%		
Number used as BUFPGs	6				
Number of IDELAYCTRLs	3	22	13%		
Number of BUFIOs	8	80	10%		
Number of PLL_ADVs	1	6	16%		
Average Fanout of Non-Clock Nets	4.02				

Figure 1: Screen Shot of 'make report' Front Page

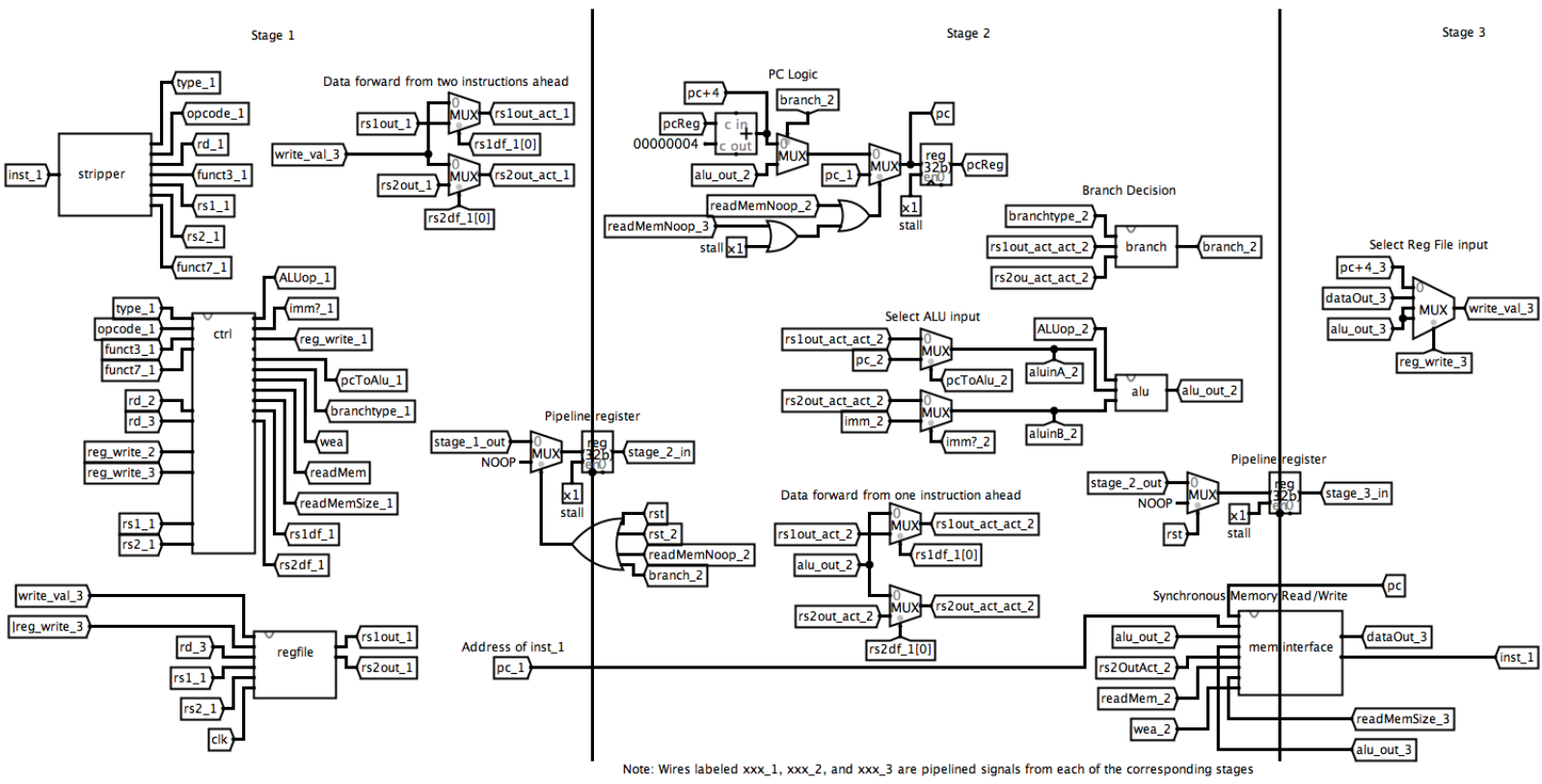


Figure 2: Block Diagram of our Datapath



Figure 3: Extra Credit Hardware Circle Acceleration