ECE 368 Digital Design
Spring 2014

Project: Lab3 – UMD RISC 24

Dates Performed: Wednesday, April 30 14th, 2014

Submission Date: Wednesday, May 14th, 2014

Team #2 :

Massarrah Tannous ________________________

Daniel Noyes ______________________________

Benjamin Doiron __________________________

Table of Contents

Problem Statement

This lab is considered part three of the project. The main purpose of it was to extend and build from the previous two parts of the project. For part one of the project, two components were designed as building blocks for the RISC. The first component was the FPU, the main arithmetic logic unit that processes all of the data. The next component was the control unit. This component was designed to control the FPU in the overall process of data based on the instruction given. The control entity is considered the center of the machine that controls the flow of the instruction in the pipeline. Whereas the main purpose of the FPU is to process the instruction where several registers must be used, such that general purpose registers.

For Part 2 of the project, many more components were built to give more functionality to the design. The control unit was revamped to achieve a 5 stage pipeline that will follow a five cycle routine: Fetch, Decode, Execute, Memory and Write Back. Each and every instruction will have every routine in it. For the routines, the control first will fetch the instruction. On the next cycle the control will decode the instruction to see what data is needed for the instruction. The Control will select what operand will go through the ALU, the selection can be Register A and Register B contents, immediate data or another form of data which can be from the previous ALU output or the program counter.

In the Next routine, The controller will Execute the instruction after it step up the data from the previous routine. The controller will tell the ALU what operation is needs to perform on the data. This operation can be an add, subtract, and, or, and more. These routines can be described in Appendix B . After Execution the controller will then perform data and memory access with the special instructions. The special instructions will access external and data memory. For the last routine is Write back, all data calculated will be written back into the registers. Couple additional features were required to be added for the purpose of this lab. The first is implementing branch, return and jump instructions into the control. Second is considering the data hazard caused Read After Write hazard. After these implementations, the UMD_RISC 24 comes to completion. Figure 1. shows the original design given for the UMD_RISC24.

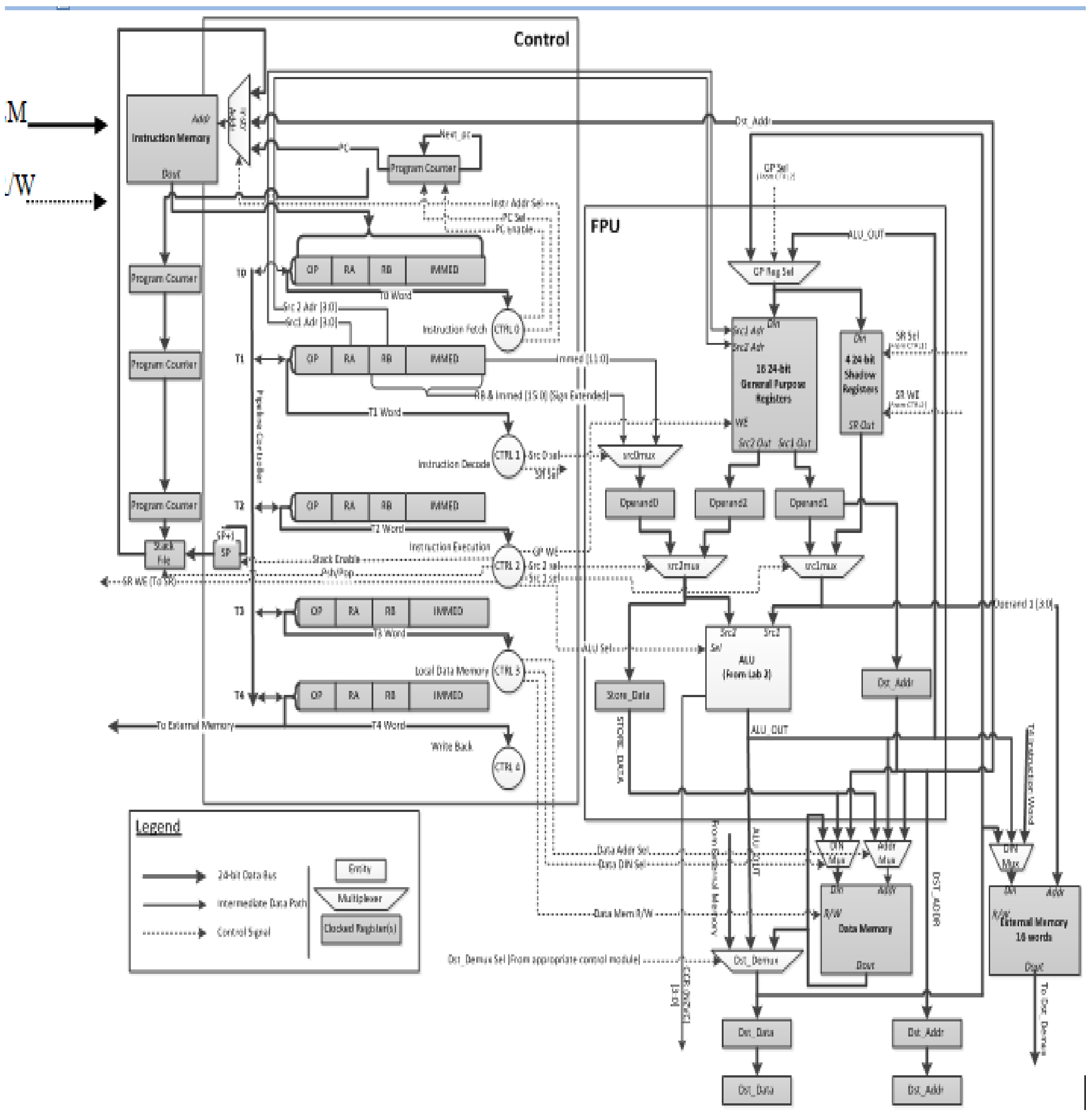

Figure 1: Pipelined FPU Data and Control Paths

Simple machine RTL block level design

(Original Designs)

The original design is very similar to the given block diagram in figure 1. Figure 2 shows the original block level design for the project. It also shows the timing diagrams at which edge (falling or rising edge) the data is latched. The original designs for the purpose of this lab are features added to previous designs (Lab1 and Lab2 of the project). The attached papers, figures 15-18, are the original hand drawn design for the control unit considering the implementation of the branch, jump and the return instructions (Figure 15). Figures 16 and 17 show the original hand drawn design for the data hazards. And figure 18 shows a top level of the original design for the debugger.

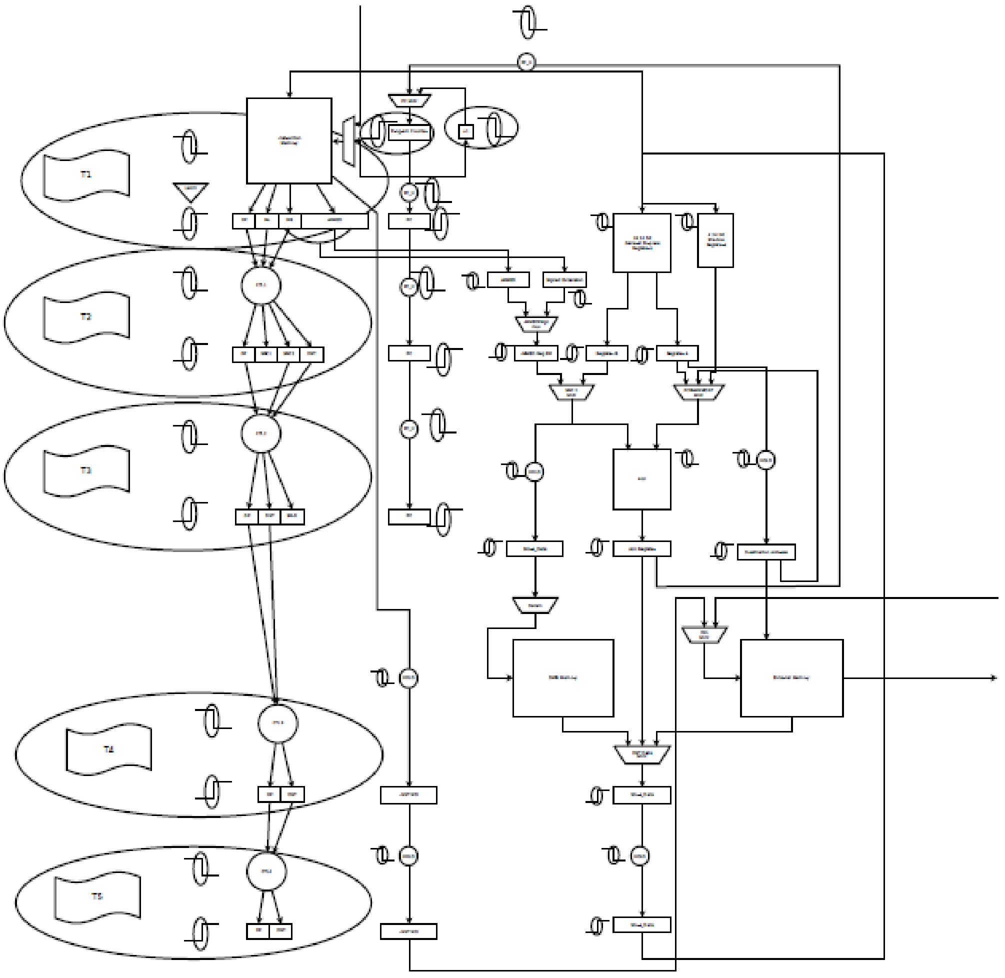

Figure 1: Original Design

VHDL component specification and schematic designs

The control unit contains major changes that as the consideration of the branch, jump and return instruction and the consideration of the data hazard. Hence, Appendix A contains the component specification and schematic designs for the updated versions of the Control Unit (Figures A1.1 through A.10), Program Counter (Figures A1.11 and A.12), and the FPU (Figures A1.13 through A.21). For the components specifications for both the VGA and the PS2 controller, please refer to Appendix A of lab3. For the debug unit components specifications please refer to Benjamin Doiron's lab2 report. The functionality of the components been the same but it is been updated as new features are added and new bugs been detected and fixed.

VHDL system specification and resulting schematic design

Based on the overall designs, figure 18 of Project- lab1, the overall system consists of several entities: PS2 controller, Debug Unit, VGA controller, and UMD-RISC24. The UMD-RISC24 Consists of several entities: Control Unit, FPU Unit, PC, Instruction Memory, Data/External Memory.

The specifications and the schematics for the PS2 Controller and the VGA are similar to Lab 3 (figure 2 and 3 of lab 3). Since the control unit has been updated with the 5 stage pipeline, the following shows the specifications and schematic (figure 3 & 4). The new specifications and the schematic of the FPU are shown in figures 5 and 6. We also show the final specifications and the schematic of the UMD RISC24 and its integration with the debug unit (Figures 7 and 8). The specifications of the debug unit are shown in figures (Figures 9 and 10)

Control Unit Entity Specification

The control entity is considered the center of the machine that controls the flow of the instruction in the pipeline. It passes signals from the debug unit into its components. It is been updated since the previous labs so that it controls data hazards and branching. The top level of the control unit is shown in figure 3 and its corresponding RTL diagram is shown in figure 4. Tables 1 and 2 show the description of the inputs and the outputs corresponding to figure 3.

Inputs	Description
CLOCK	Drives the component.
RESETN	Resets the chip constants.
ENABLE	Enable/Disables the components.
INSTRUCTION	Enables/Disables writing to the instruction memory.
PC	Address of where to write into the instruction memory
NVCZ_Compare	The result of the 4 bit comparator to compare the branch

Table 1: Description of the control unit inputs.

Outputs	Description
SR_SEL_CTRL1	Selects between the shadow register.
SRC0_SEL_CTRL1	Selects between Immediate (0) and Extended (1)
SRC1_SEL_CTRL1	Passes the RA data.
SRC2_SEL_CTRL1	Passes the RB data.
SRC1_SEL_CTRL2	Selects between register B (1) and Immediate (0)
SRC2_SEL_CTRL2	Selects between register A (0) and shadow register (1)
NVZC_MASK	OUTPUT From CRTL1 the MASK in case of branching
SIGN	Passes the signed data from the instruction
IMMED	Passes the immediate data from the instruction
ALU_SEL	Passes the opcode to the from the instruction at the CTRL 2.
PC_STACK	Passes the popped value from the stack
PC_OUT_CTRL3	Passes the program counter at the CTRL3
DATA_DIN_SEL	Selects between DST Reg(0) and STORE_DATA (1)
DATA_ADDR_SEL	Selects between DST Reg (0) and ALU (1)
DATA_RW	Selects based based on the opcode: READ(0) and Write(1)
DST_MUX_SEL	Selects between ALU(00), Data Memory (01), External Memory (10)
GP_DIN_SEL	Reg Address to write to
GP_WE	Write enable for general purpose register
SR_DIN_SEL	Shadow Register to write to
SR_WE	Write enable for shadow register
PC_SEL	An output signal to the Program Counter to select the PC input
PC_CTRL	Output signal of the program counter resulted from branch instruction

Table 2: Description of the outputs for the control unit.

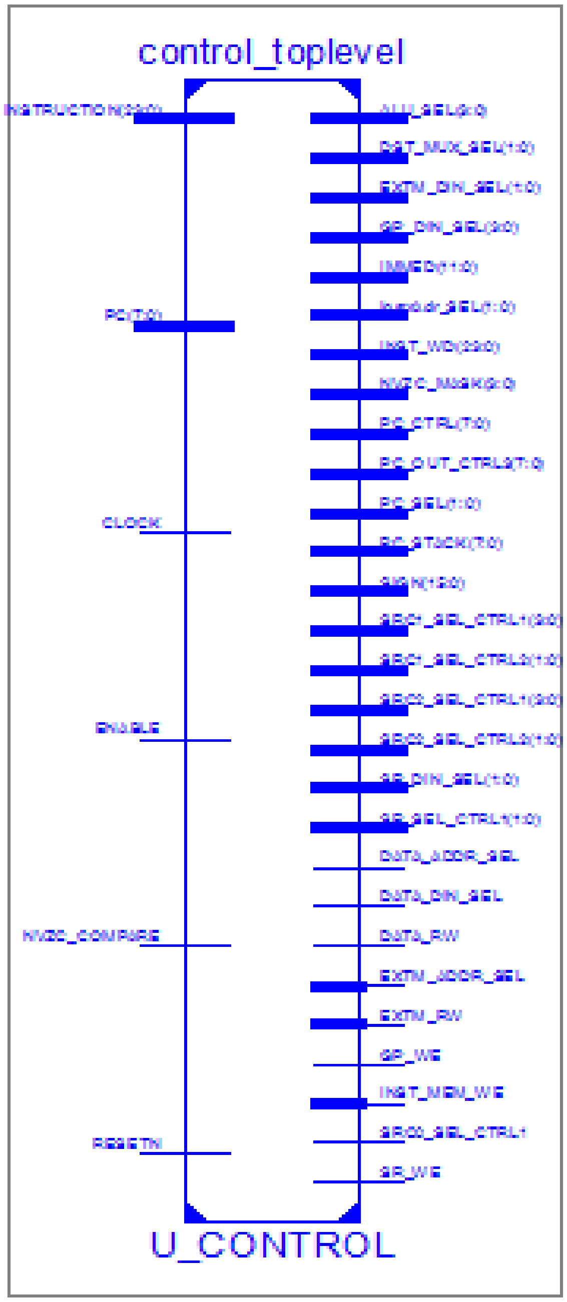

Figure 2: Top level of the control unit

Schematic

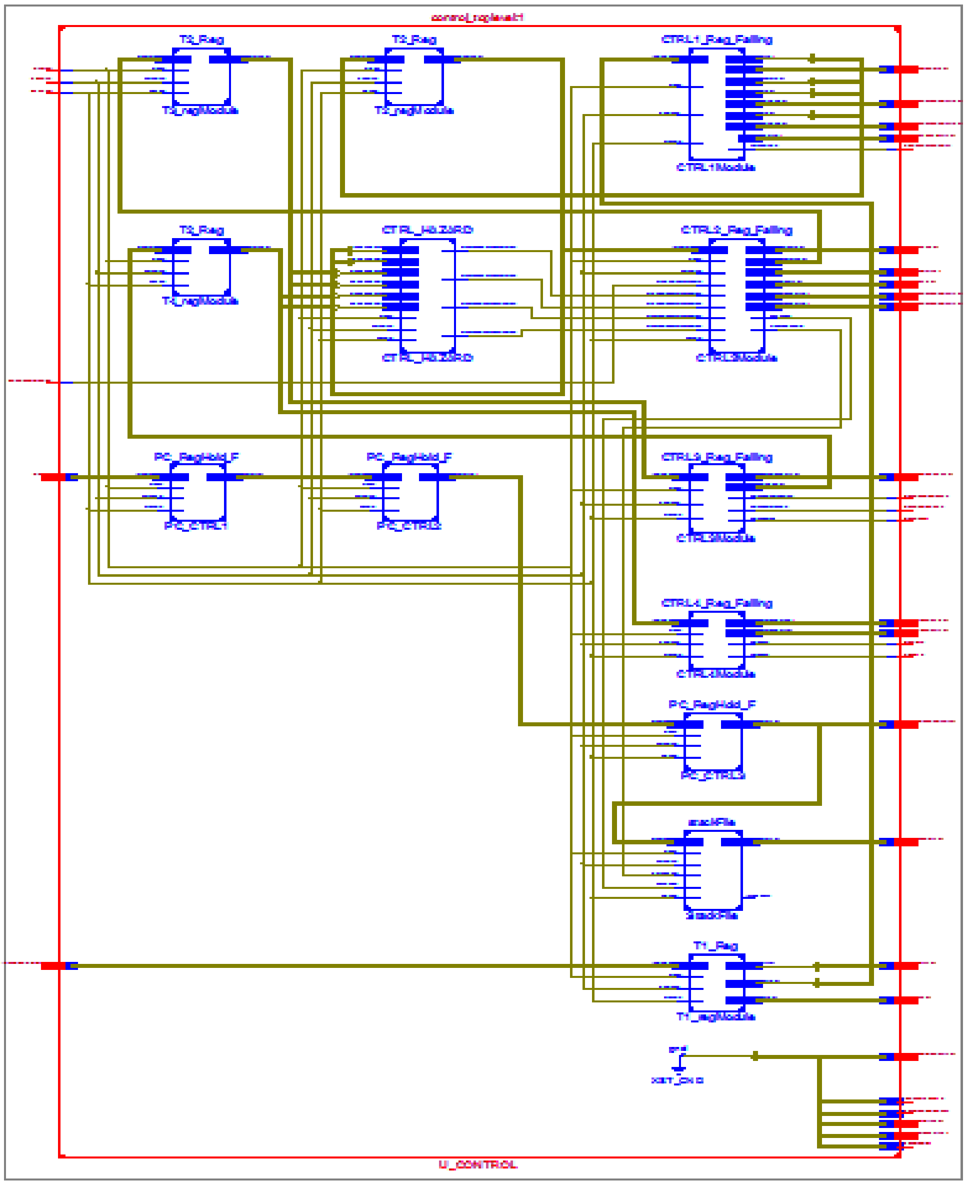

Figure 3: RTL diagram of the control unit v3.

FPU Entity

Specification

The main purpose of the FPU entity is to maintain registers and arithmetic operations based on the ALU. The top level of the FPU is shown in figure 5 and its corresponding RTL diagram is shown in figure 6. Tables 3 and 4 show the description of the inputs and the outputs corresponding to figure 5. This entity been updated after finalizing the functionality of the FPU.

Inputs	Description
CLOCK	Drives the component.
RESETN	Resets the chip constants.
ENABLE	Enable/Disables the components.
SIGN	Signed data from the instruction
IMMED	Immediate data from the instruction
REG_DATA_IN	Register Data input for general and shadow registers
SRC1	General purpose Reg A Address
SRC2	General purpose Reg B Address
GP_DIN_SEL	Reg Address to write to
GP_WE	Write enable for general purpose register
SR_SEL	Shadow Register Select
SR_DIN_SEL	Shadow Register to write to
SR_WE	Write enable for shadow register
SRC0_SEL	Select between immed and signed extension
SRC1_MUX	Select between PC, RA, SR, DST_ADDR (4 select)
SRC2_MUX	Select between immed or Reg B (2 select)
ALU_OPCODE	OPCODE for the ALU
PC	Program Counter
NVCZ_Mask	The input of the MASK from the branch instruction

Table 3: Description of the control unit inputs.

Outputs	Description
STORE_DATA	Destination Data
DST_ADDR	Destination Address
ALU_OUT	Results of the arithmetic operation
CCR	Condition Code Register (ALU)
NVZC_Compare	Returns the result of the comparison the MASK to the NVZC

Table 4: Description of the outputs for the control unit.

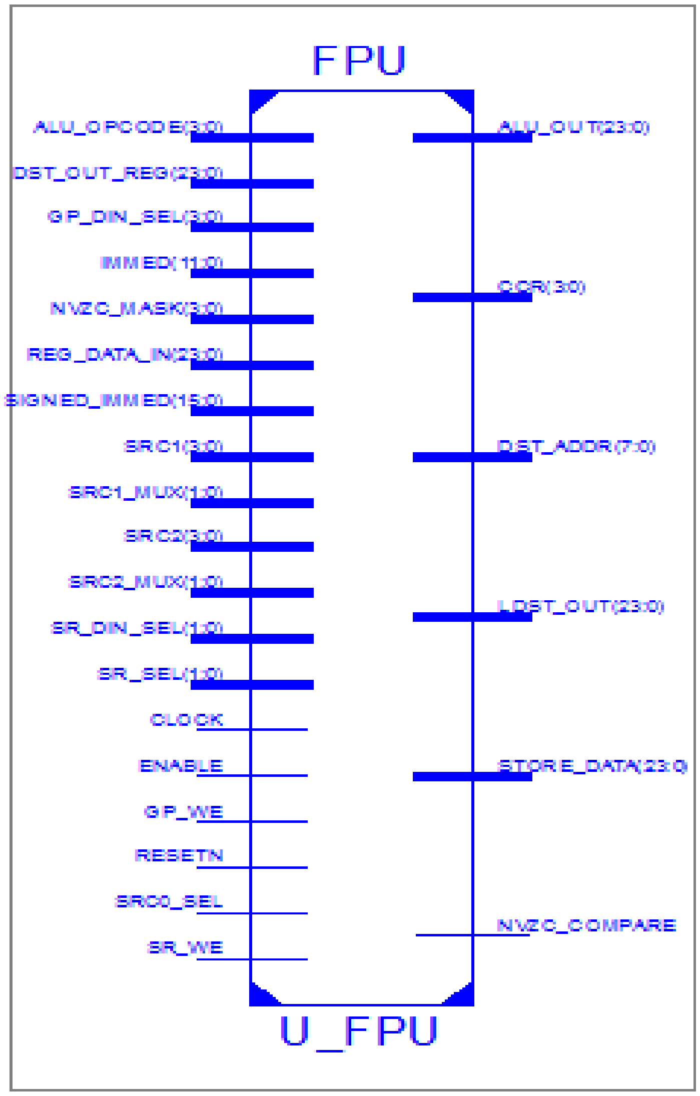

Figure 4: Top level of the FPU

Schematic

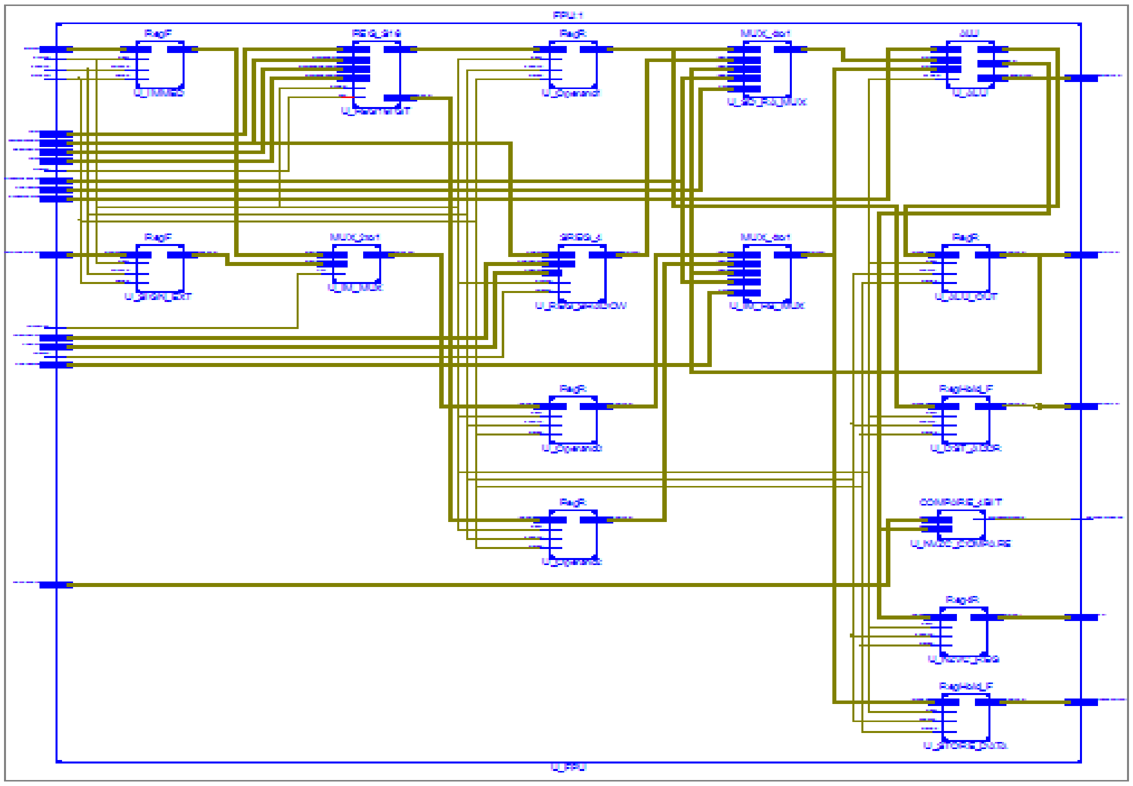

Figure 5: RTL Diagram of the FPU entity.

UMD RISC 24

Specification

This entity consists of the control entity and the FPU. It interfaces with the debug unit to get the instructions process them and then pass back the results to the debug unit. The top level of the UMD RISC24 is shown in figure 7. Its corresponding RTL diagrams are shown in figure 8 , where it shows the top level of the UMD RISC components. Tables 5 and 6 show the description of the inputs and the outputs corresponding to figure 7.

Inputs	Description
CLOCK	Drives the component.
RESETN	Resets the chip constants.
ENABLE	Enable/Disables the components.

Table 5: Description of the inputs of the UMD RISC24.

Outputs	Description
STORE_DATA	Destination Data
DST_ADDR	Destination Address
ALU_OUT	Results of the arithmetic operation
CCR	Condition Code Register (ALU)
EX_OUT	Output of the External memory
PC_OUT	Output of the program counter to the Debugger
LDST_OUT	Output the load / store on the ALU

Table 6: Description of the outputs of the UMD RISC24.

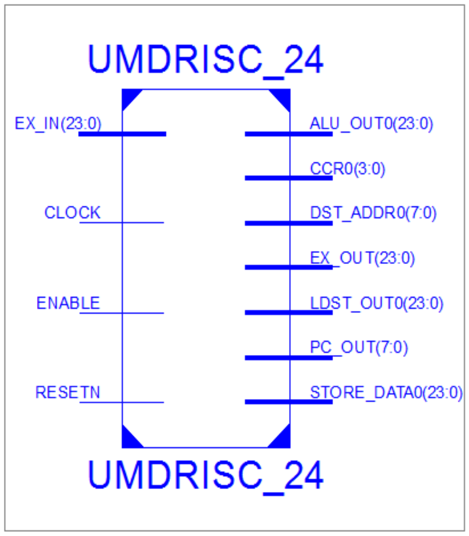

Figure 6: Top level for the UMD RISC24

Schematic

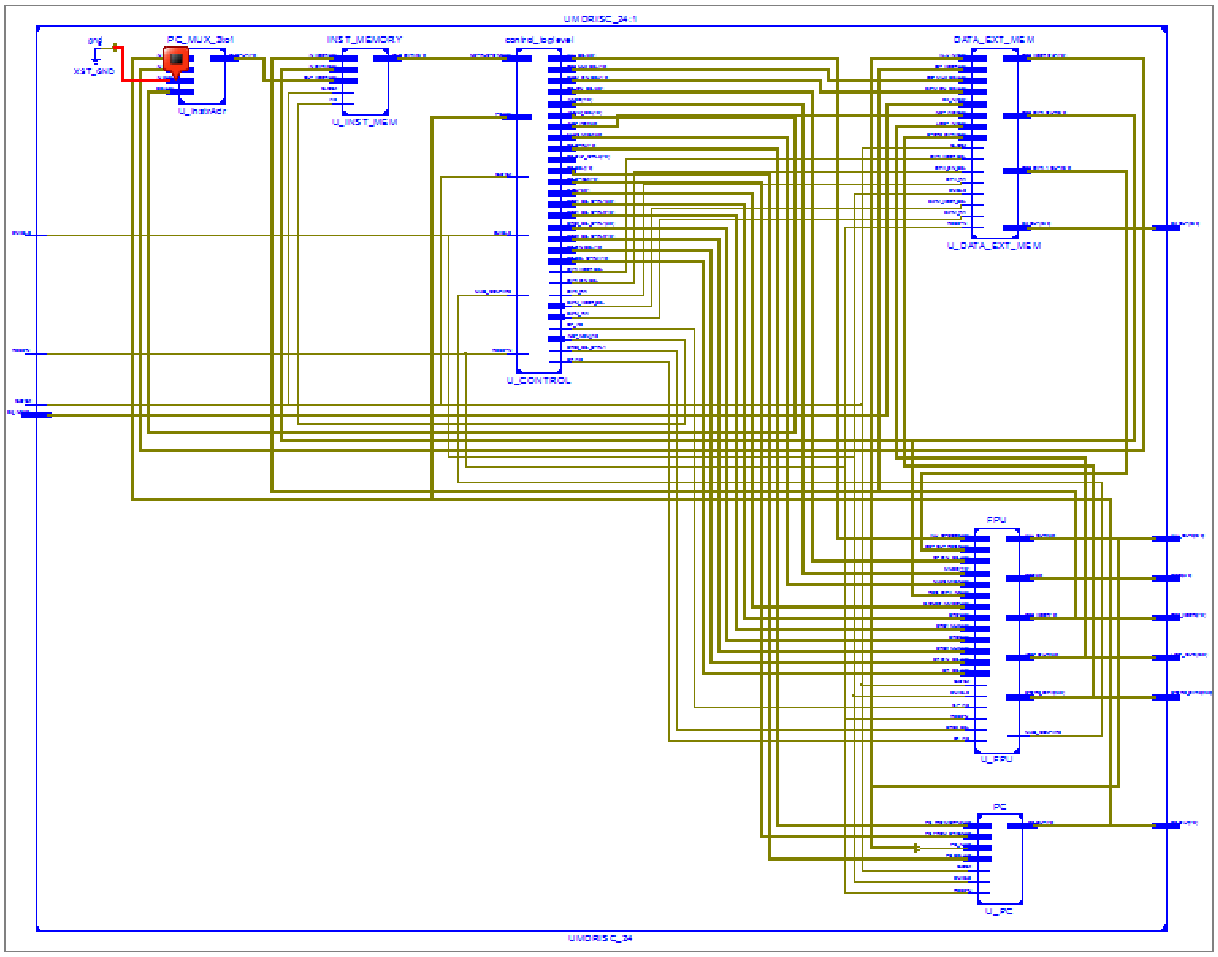

Figure 7: RTL Diagram of the the UMD RISC24 showing the toplevel of it is components.

Debug Unit

Specification

This entity consists of the debugger structural, VGA structural and the UMD-RISC24 structural. It interfaces with on the top most level to the exterior pins on the FPGA. The goal of this entity is to successfully debug the RISC machine and allows demonstration that it works on a hardware level. The top level of the debug unit is shown in Figure 8. The corresponding RTL diagram is shown in Figure 9. Table 7 show the description of the inputs and the outputs corresponding to Table 8.

Inputs	Description
CLK	Drives the component.
CLR	Resets the chip constants.
EXT_MEM	Allows memory to be pushed into the VGA
p_counter_addr	Program counter address storage to use for addressing
PS_CLK	Keyboard clock input
PS_DATA	Keyboard data input
Read signal	Signal to indicate to read the signal
RESET	Resets / Clears
RISC_ALU	ALU output from the RISC
RISC_CCU	CCU output from the RISC
RISC_DST_ADR	DST_ADR output from the RISC
RISC_STORE_DATA	STORE_DATA output from the RISC
RISC_WE	Write enable to the RISC
SEL_WE	Write enable to the

Table 7: Description of the inputs of the Video and Debugger top level.

Outputs	Description
data_output	Data to be sent to the RISC
B1	Blue VGA input
G1	Green VGA input
R1	Red VGA input
HS	Horizontal sync
VS	Vertical sync

Table 8: Description of the outputs of the Video and Debugger top level.

Schematic

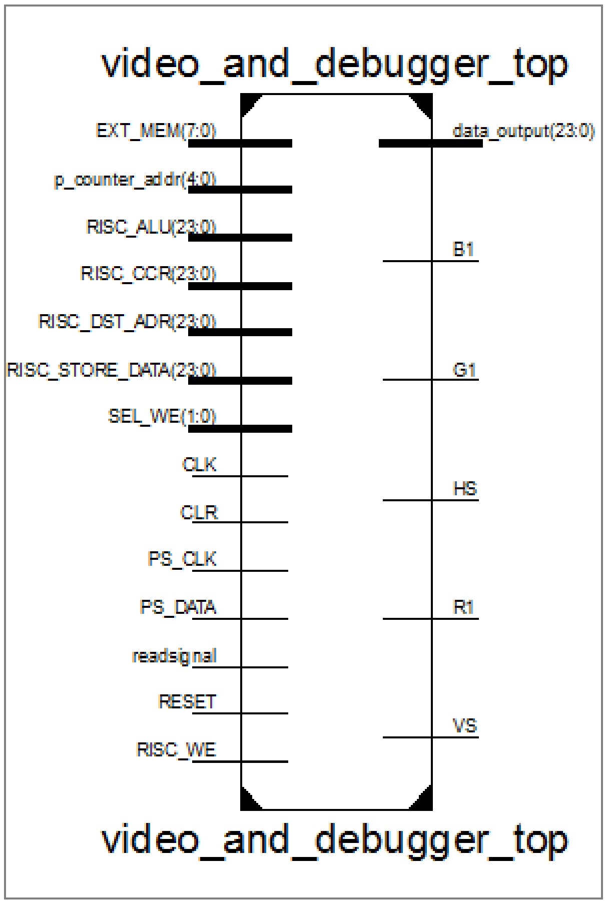

Figure 8: Video & debugger top level design

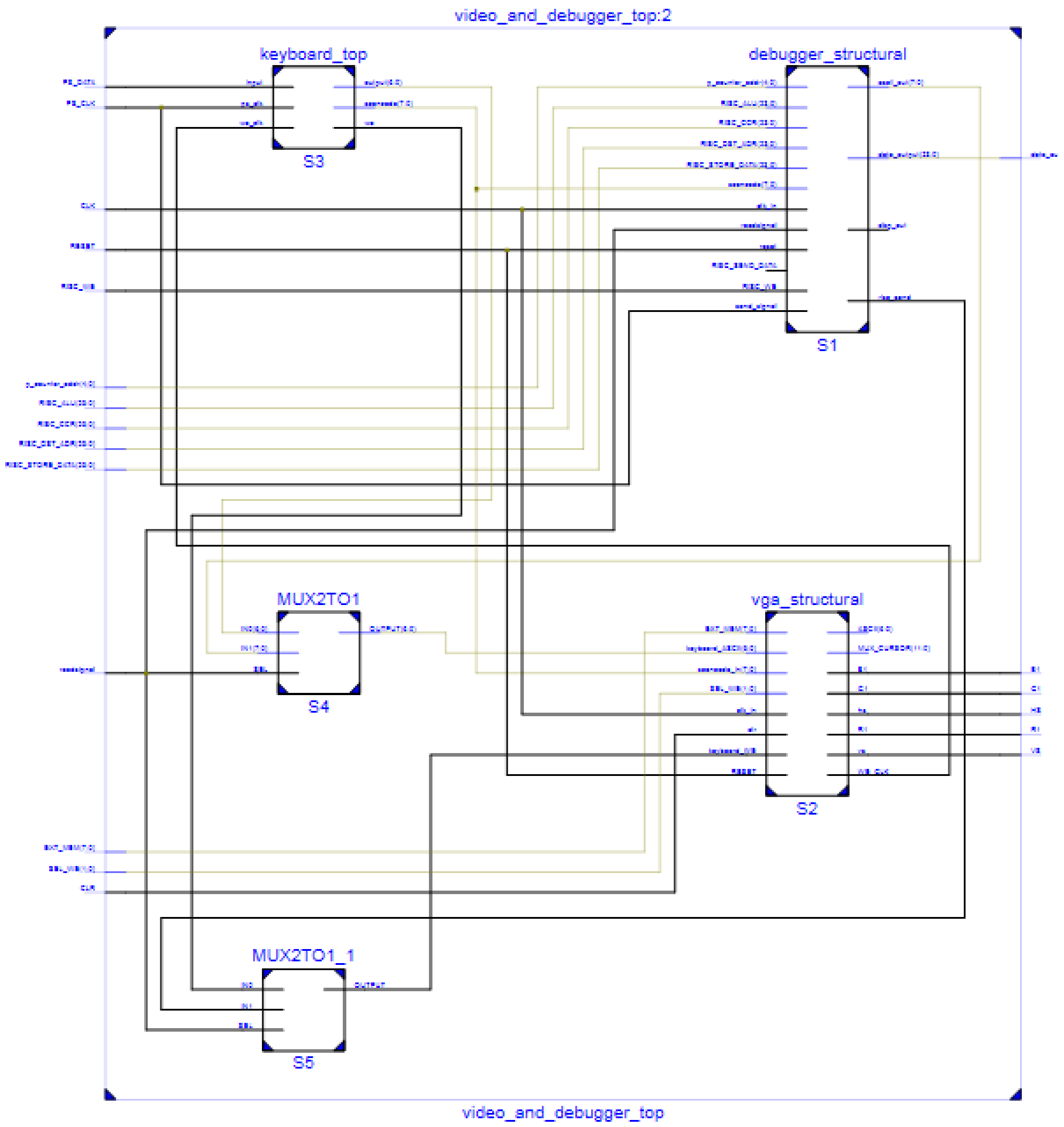

Figure 9: RTL Diagram of the the Video and Debugger showing the top-level of it is components.

VHDL Code and Test Bench Code

The VHDL code for the UMD Risc24 integrated with the debug unit and the UCF file has been emailed as a zip file to Dr. Fortier and to David Prairie.

Designs comparisons

In the original design schematics, most of the design was based upon Dr. Fortier's UMD RISC24 schematics in the Project handouts. In the First two parts of the project, the schematic was looked over and various bugs and errors were corrected. In the third part of the lab was to further develop the RISC CPU and demonstrate added functions. There were modification in the overall design to process various data hazards in a pipeline architecture. From the first two parts of the lab, most of the design is still retaining its original form. Overall the Code was modified slightly to open up more functionality in the system. The control unit was incorporated the jump/branch/and return instructions as demonstrated in the code.

In the overall RISC CPU, we been further maturing it from each of the previous labs and the project is almost mature enough. In order to further mature the RISC CPU, More routines were add which were jump and link, return from link, and Branch instruction set. Overall the controller was designed to maintain each instruction and run them when needed. The Stack was implement as an component in the control that will manage the return of the program counter. The Control unit went through multiple upgrades to achieve this objective, besides the jumping instructions. The Hazard Controller was developed in order to regulate flow of data in the FPU and to prevent data being overwritten.

Test Plan

The overall test plan for this system is through both simulation and through hardware. For the simulation, we tested the behavior of each of the new components that were added to the overall RISC. For the hardware test plan, the plan is to input assembly instructions from the keyboard and compare the expected result with output on the VGA as planed in the previous segment of the project. The Overall instructions in the simulation were used to test the output of each of the various components and how each will react. With the debugger unit still in debugging, the overall state of the device in demonstrate might have a glitch of few but the overall design is based on the proof of concept based on testing each various component in the system.

Simulation:

In the overall project, we created multiple test benches for each component in the Control entity, FPU, debug and the overall entity. Those various test benches were tested in previous reports and the update changes to the test bench can be shown in Figure 11,12,13. The original components can be shown in figures 8 through 17 of Project-Lab1 report.

Since every type of component was tested earlier, we focused on testing the integration and further development of the components. Figure 11, shows a simulation for the UMD RISC 24. This simulation demonstrate how the design will react under certain instructions. This is also elaborated in Table 9 in the Hardware Test Plan.

Hardware Test Plan:

In the Hardware Test Plan, we simulated tested the RISC device with 7 instructions to validate the proper output. Currently with the testing, there were a few bugs appeared and still in the process of ironing the bugs out. As demonstrated in both Table 9 and in Figure 10. Mainly, these bugs are due to that the data hazards that not being considered in the RISC24.

	Inputs				Output			Test Results
Clock cycle	alu_opcode		store_data	dst_addr	alu_out	ldst_out	ccr	
1	5		0	0	0	0	4	Passed
2	5		1	0	2	0	0	Passed
3	0		0	0	1	0	0	Passed
4	2		0	0	0	0	4	Failed
5	3		0	2	0	0	0	Failed
6	0		0	2	2	0	0	Failed

Table 9: Simulation result checks

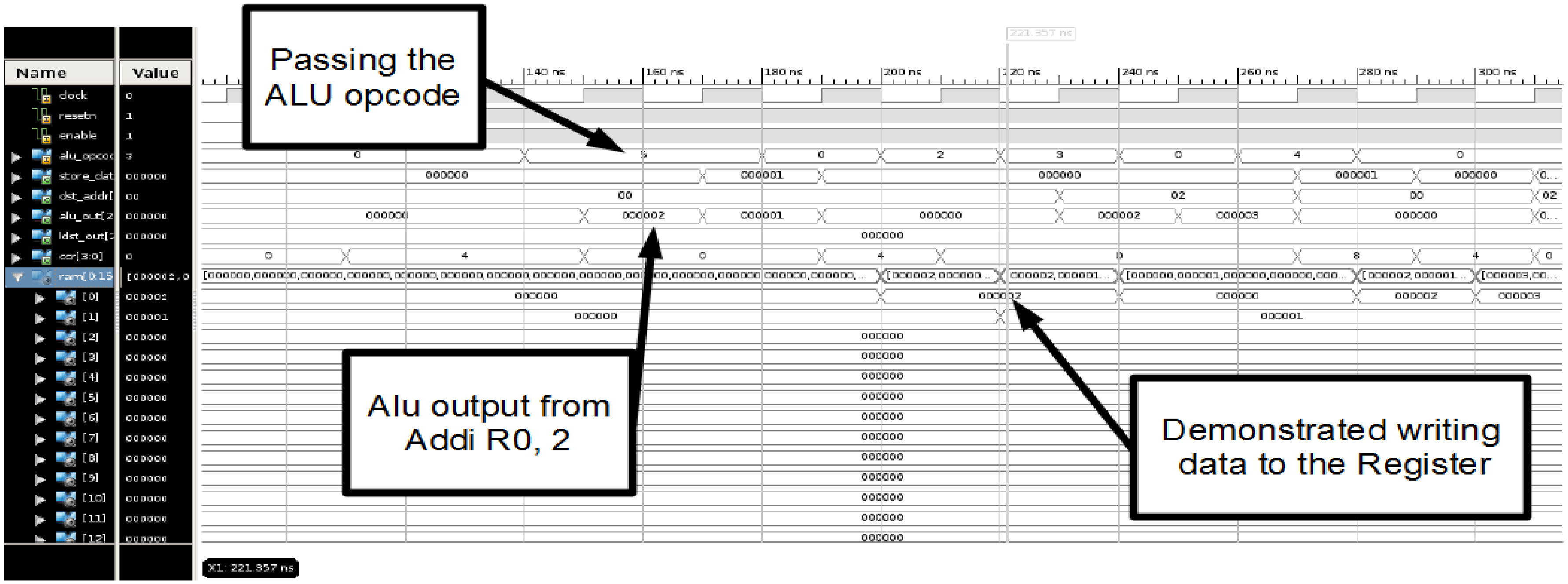

Figure 11: FPU operation, demonstrated Forwarding

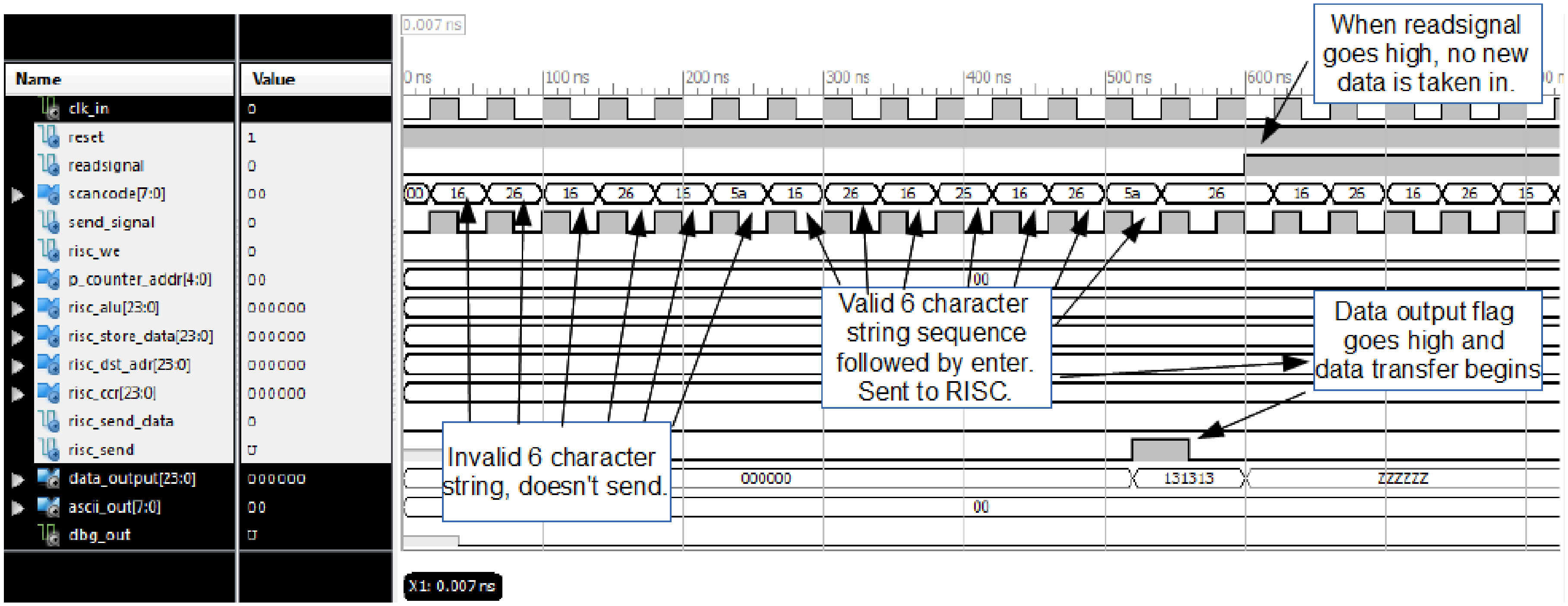

Figure 12: Top level for the video and debugger top level

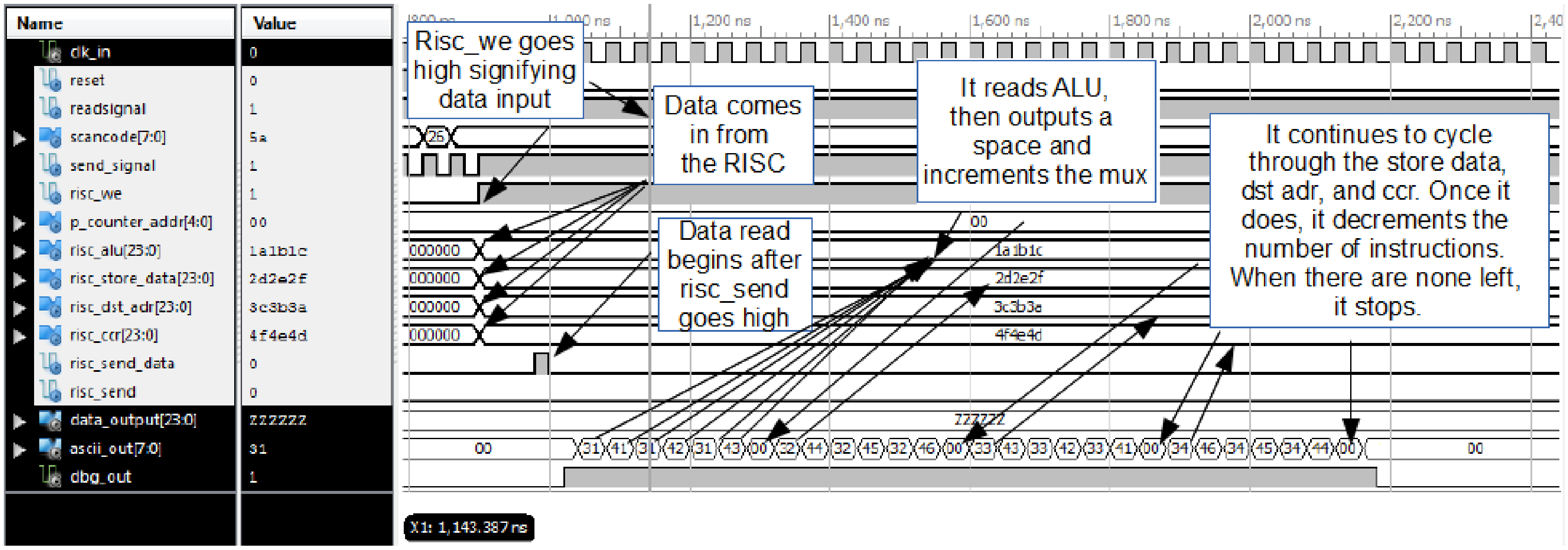
Name
Value
clk_in
reset
readsignal
scancode[7:0]
send_signal
risc_we
p_counter_addr[4:0]
risc_alu[23:0]
risc_store_data[23:0]
risc_dst_adr[23:0]
risc_ccr[23:0]
risc_send_data
risc_send
data_output[23:0]
ascii_out[7:0]
dbg_out
Risc_we goes high signifying data input
Data comes in from the RISC
It reads ALU, then outputs a space and increments the mux
It continues to cycle through the store data, dst adr, and ccr. Once it does, it decrements the number of instructions. When there are none left, it stops.
Data read begins after risc_send goes high
X1: 1,143.387 ns

Figure 13: Timing diagram of the debugger interfacing from the RISC to the VGA

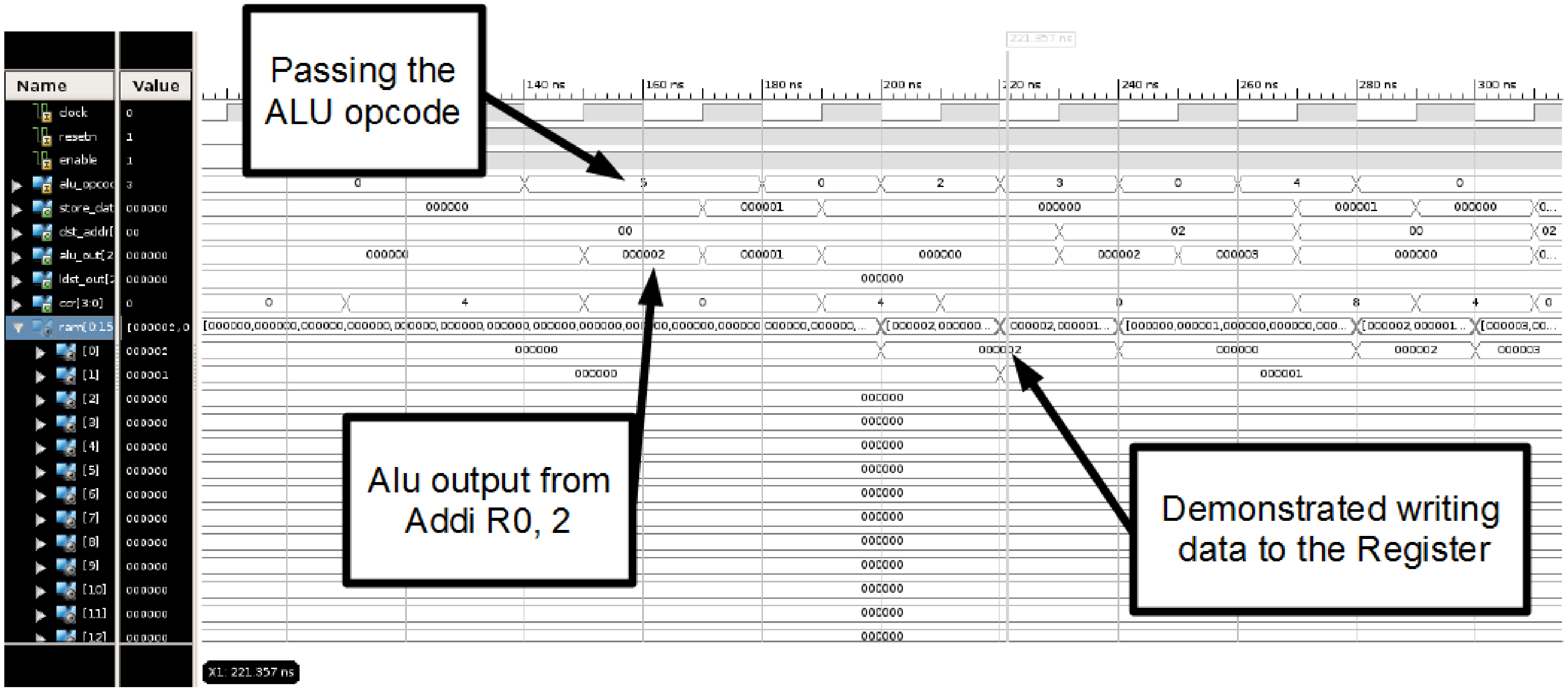

Figure 14: UMD-RISC24 simulation

UCF File

The UCF File was submitted to Dr. Fortier and David Prairie via email.

Conclusion

Through this lab we learned a lot more about each component designs and how to further develop the design. We also understood more intricate operations of the RISC machine through the various debugging routines that were created in order to further develop the knowledge of what a CPU needs in order to function. We also learned the importance of data hazards and the concepts of branch prediction. This is the Last of the three lab installments of the project, Overall each and every lab that we went through were very helpful for the progress of the entire project. This lab was helpful for the progress of the entire project where we have the basics to advance and implement more complicated instructions such as Branch Instructions.

We almost finalized each new instruction for the control as well as implemented a few routines to control the system even more. We simulated each entity though the text bench simulation. Overall the simulations were helpful to see how the overall system should react. Even though we implemented a few more instructions in the circuit, there is still the problem of data hazards and that is the reason why we developed the system with data hazards. One of the main methods of fixing any data hazard was to implement a method of staling which can be done with the NOP instruction. This can be implemented in the instruction code straight from the compiler as a temporary method of correcting the data hazard.

Reflection:

In the Overall Lab, we gain more footing and understanding in coding with VHDL. Each and every part of the project helped build up the confidence which will help each of us succeed in hardware programing with VHDL. In This lab we deviated the work up based on each expertise. Massarrah was in charge with implementing the new instructions to the system. While Benjamin focused on further improving of the debugger for the RISC machine. Daniel developed the data hazard control component and linking of each and every component in the system.

This Lab taught us a lot in begin a developer , in the previous two parts of the lab we created the RISC code from scratch and we progressed in the design and implementation of it. We can see further improvements on the overall design and coding structural. For this lab, we split the work among each other for the RISC. Massarrah proceeded to develop the control unit and the stack routine. While Daniel created the data memory and external memory control plus the program counter. We also had an honorary group member Benjamin Doiron who joined with us and developed and created a debug unit that will maintain the UMD-RISC24 on hardware testing. This project would not have been done without the effort put in by all of us.

The work we created and collaborated on helped us as a whole to in further developing the project. We discussed new ideas and gained a better understanding of the design. Right now as a whole, the RISC is partial completed(in our opinion). There is still bugs in the system that will need to be addressed. Each and every bug will be interpreted as a significant with minor bugs. Most of the current bugs are due to various data hazards that are present in the RISC device and various minor parts to the design.

This project has been both a challenge and a reward overall, we devoted much resources into the design and came out with a prototype RISC. Even though the prototype of the RISC is still buggy, it is a step in the right direction on completing the Project.

ECE 368
Digital Design
Spring 2014

Project: Lab3 – UMD RISC 24

Appendix A: Component Specification Design Layout Section