Organization of Digital Computer Lab
EECS112L/CSE 132L

Assignment 3
Single-cycle MIPS - Complete

prepared by: Team CART

Student name	Student ID
Raquel Fallman	26316814
Arash Nabili	37684183
Christine Srun	15386050
Tyler Stevens	40051117

EECS Department
Henry Samueli School of Engineering
University of California, Irvine

February 13, 2016

Contents

List of Figures

1 Introduction

In this lab, we completed the design for our simple single-cycle MIPS processor. Building on our previous design, unconditional branch (jump), conditional branch, shift, and memory instructions for loading/storing are now supported in the controller and datapath.

2 Processor

Most of the processor design has been carried over from the previous project. This implementation is divided into two major units: the controller and datapath. The processor is broken down into basic functional blocks and each component is verified with its own testbench. The components that have been modified for this design is the program counter, controller, ALU, and RAM. We have also added two new components: the adder and shifter.

The processor ports are shown below.

Processor Port Description			
Port name	Port size	Port Type	Description
ref_clk	1	IN	clock signal
reset	1	IN	reset to normal state

CPU operation types		
op	Operation Type	Comments
000000	R-type instruction	The funct field will be sent to the ALU
001XXX	I-type instruction	The controller will determine the ALU function code
00001X	J-type instruction	The controller will determine the ALU function code
100011	Load Word	ALU operation will be addi
101011	Store Word	ALU operation will be addi
Others	nop	ALU function code will be set to nop

2.1 Design Schematic

Below is the design schematic for our implementation of the processor. Previously, the program counter was implicitly incremented and now it is incremented by an adder. We have also changed the processor from word-addressable to byte-addressable. The black box with white text are labels for the components as used in our processor.vhd file. The wires are also labeled with names of signals used in the design and has the label for the associated bit size.

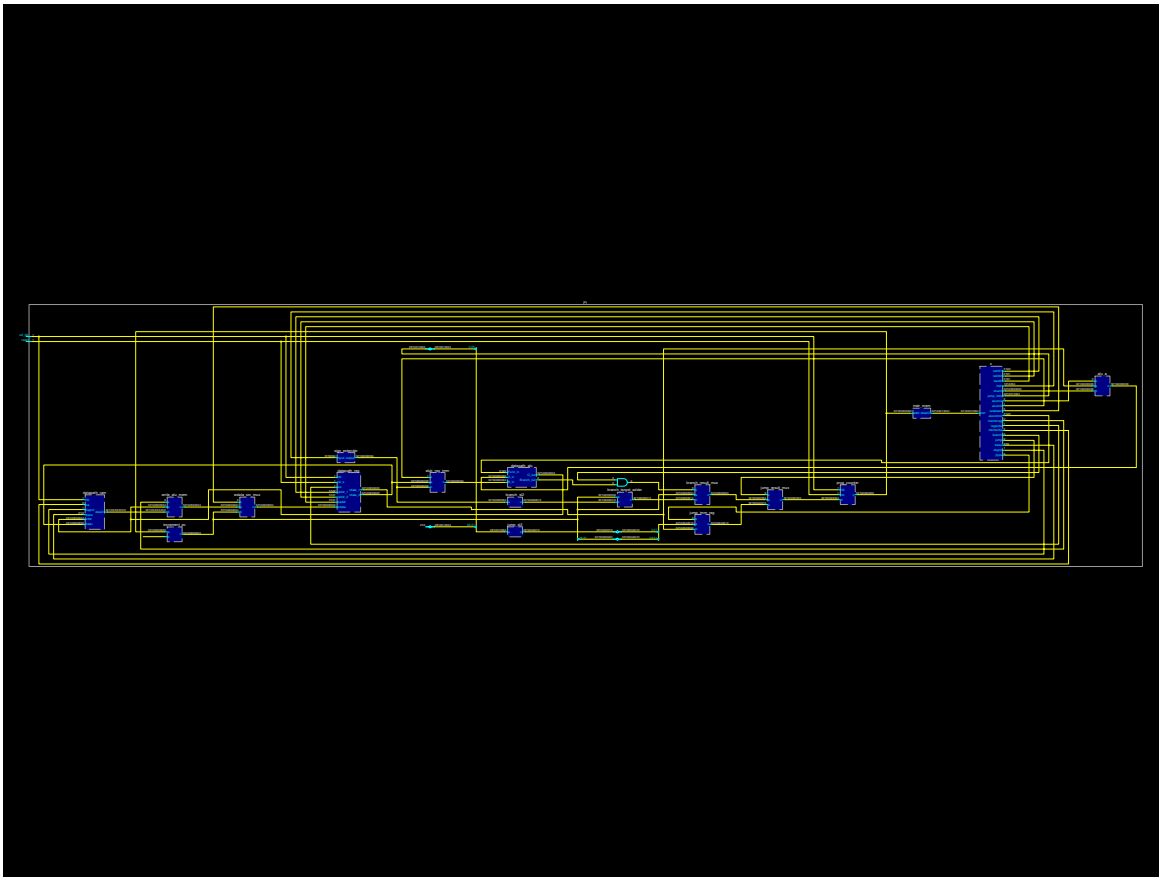


Figure 1: Design Schematic of the Processor generated from QuestaSim.

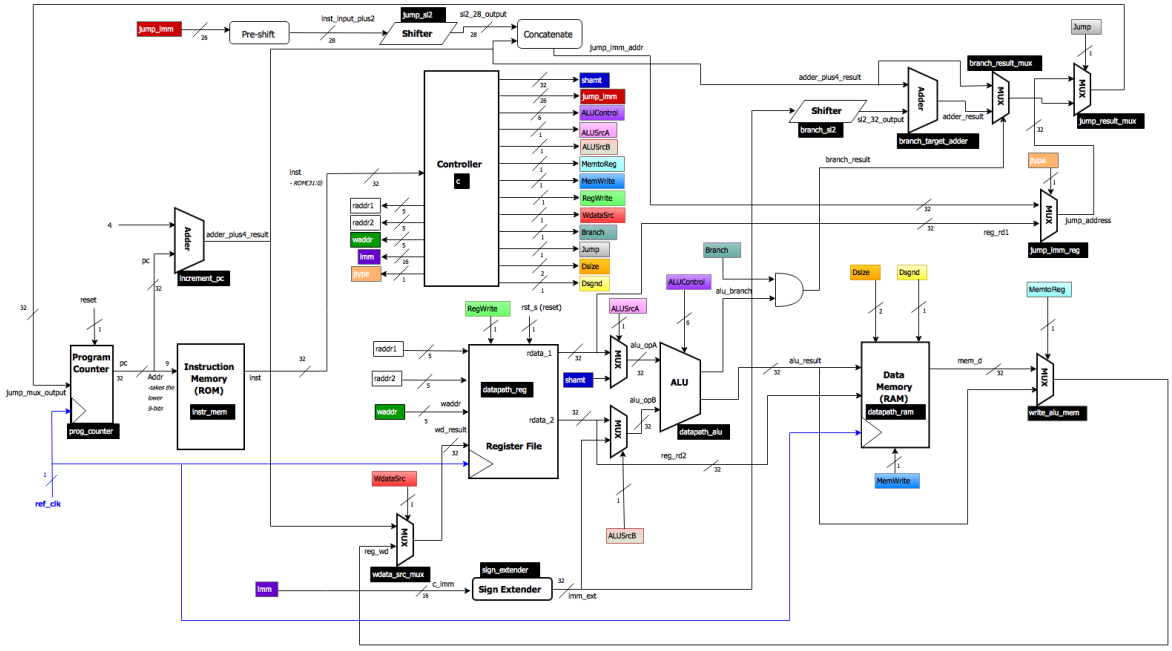


Figure 2: Created diagram for schematic.

2.2 Simulation

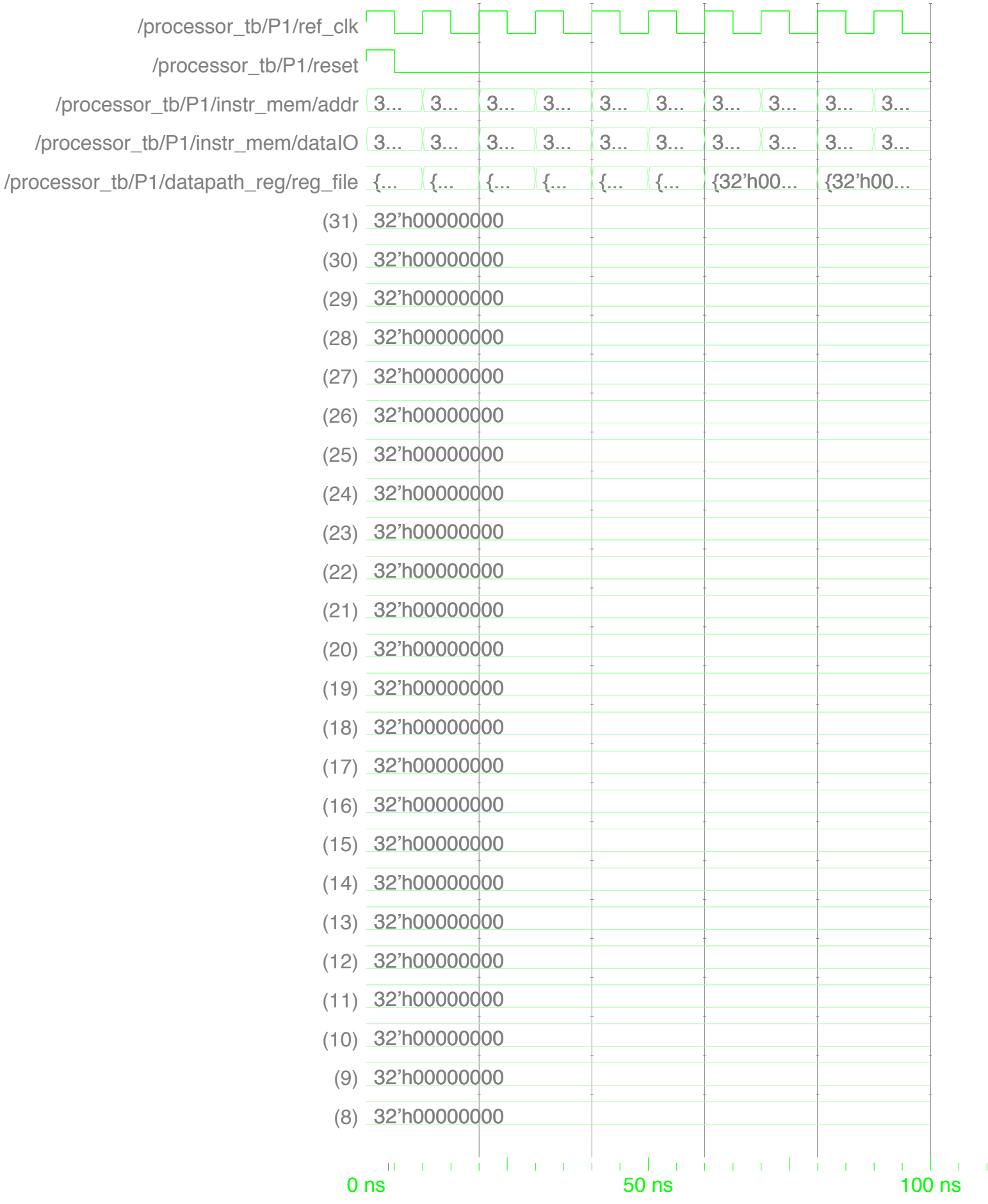
To simulate the processor, we created a testbench and used the provided MIPS code to verify the design. The code is preloaded into the instruction memory. It tests R-type, I-type and J-type instructions. The following operations are supported:

Supported Instructions			
Arithmetic	Shift	Branch	Memory Operation
ADD	SLL	BEQ	LB
ADDU	SRL	BNE	LH
SUB	SRA	BLTZ	SB
SUBU	SLLV	BGEZ	SH
AND	SRLV	BLEZ	LBU
OR	SRAV	BGTZ	LHU
XOR		JUMP	
NOR		JR	
SLT		JAL	
SLTU		JALR	
ADDIU			
ANDI			
ORI			
XORI			
LUI			
SLTI			
SLTIU			

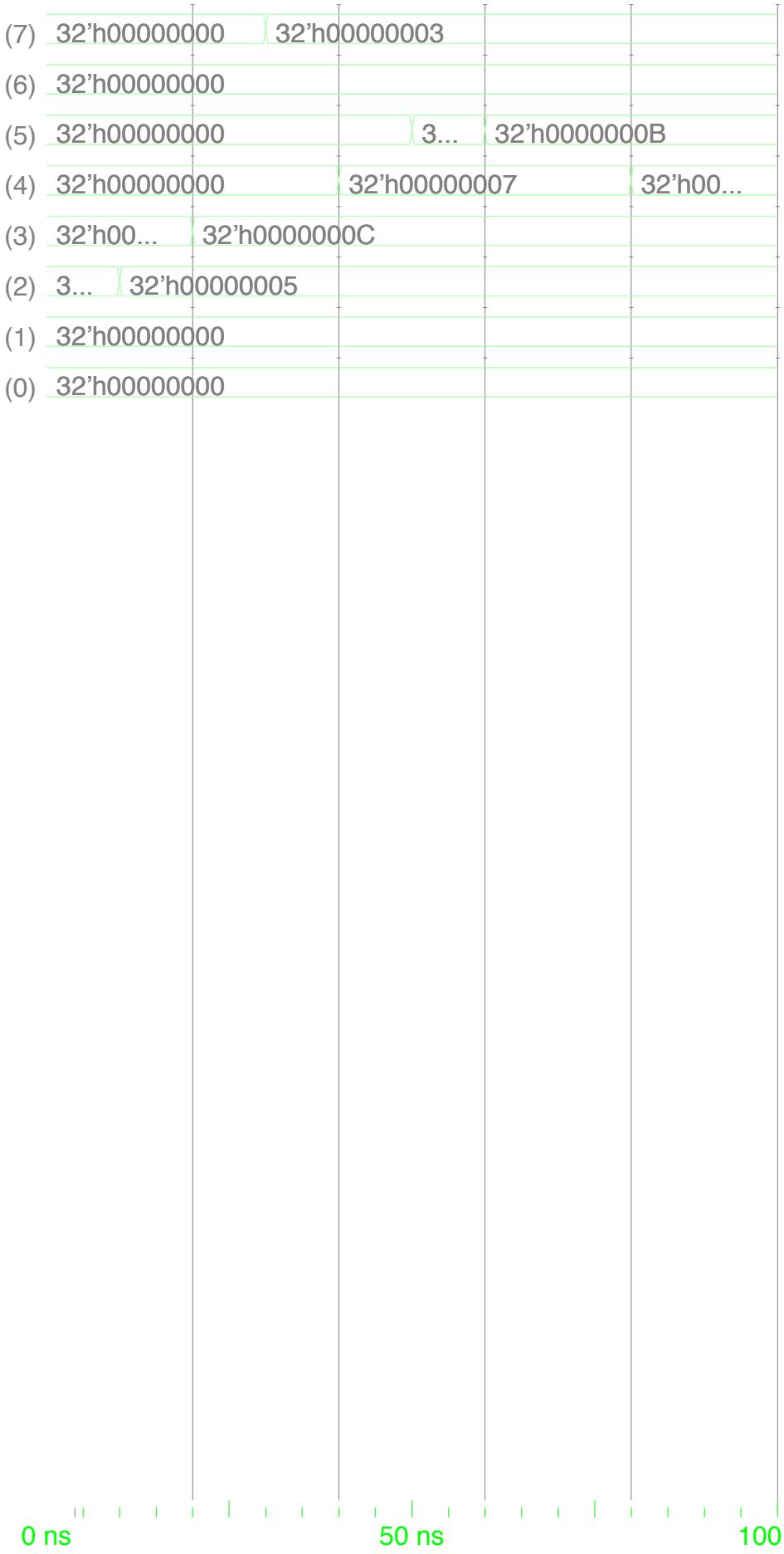
The first set of waveforms were generated from the sample code provided by the professor. The second set of waveforms were generated from our own test code to make sure that all the operations were working properly.

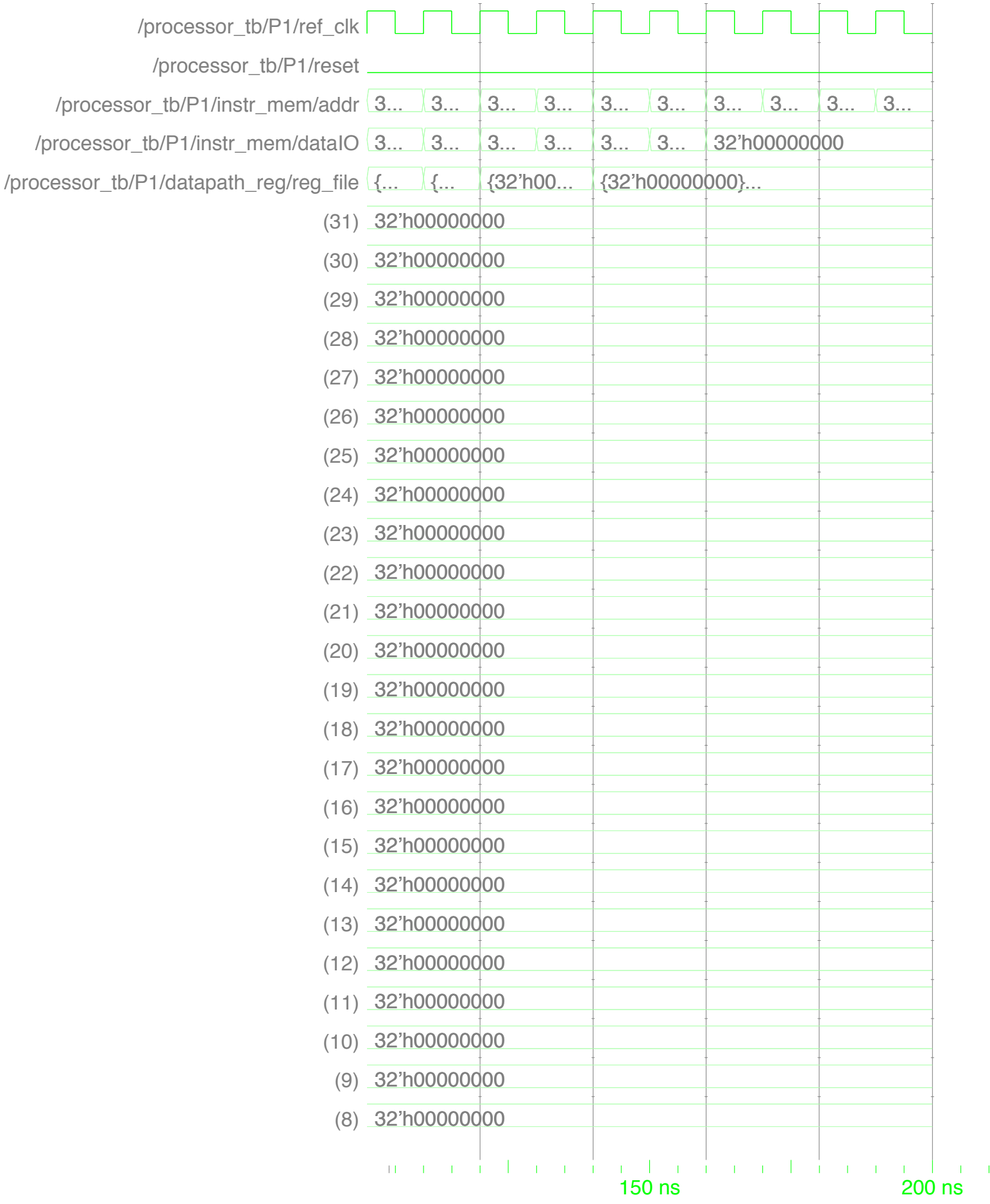
2.2.1 Waveform 1: Using provided sample code.

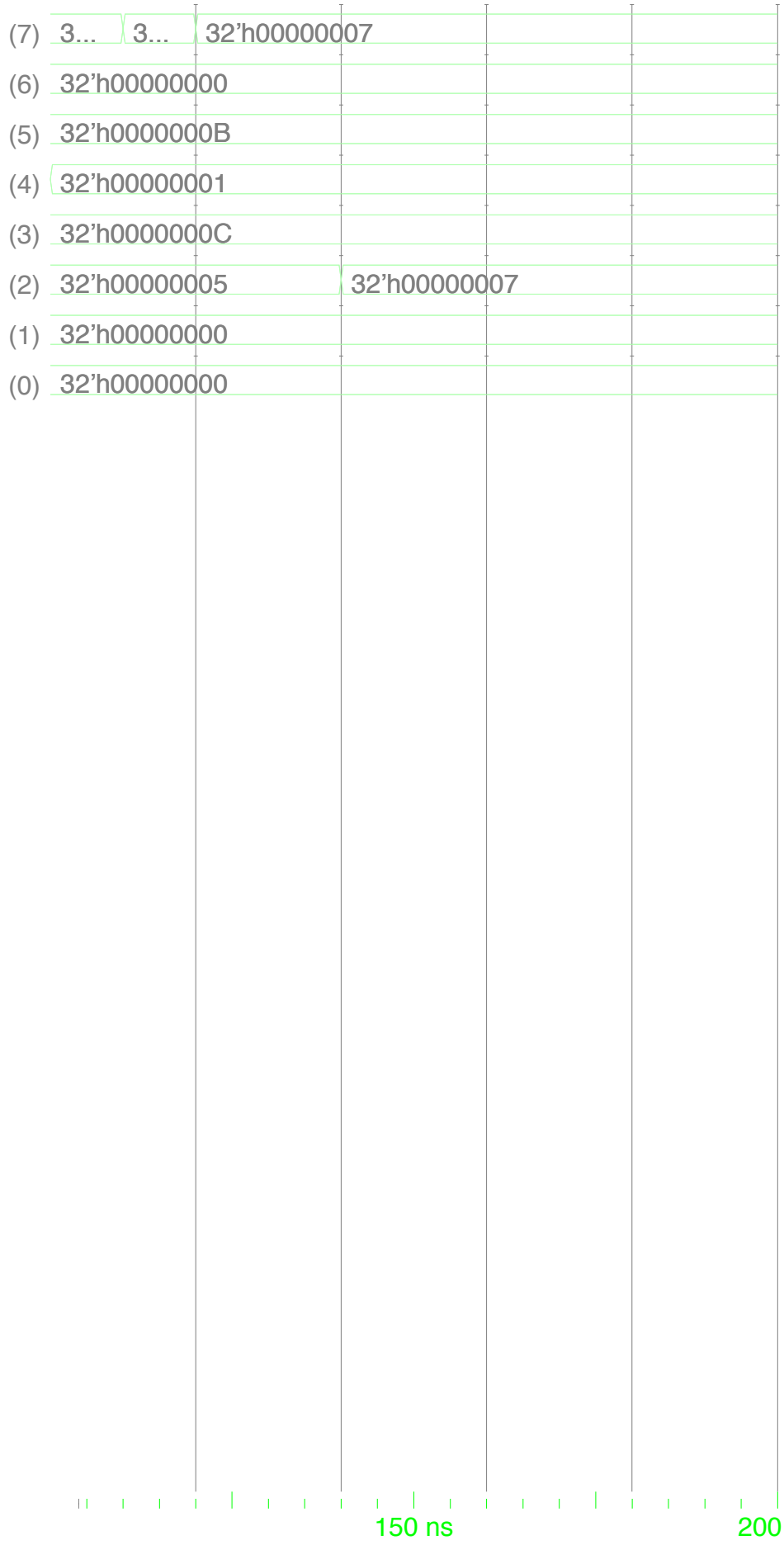
The final output generated from the sample code is: 12 in register 3, 1 in register 4, 4 in register 5 and 0 in r0, r2, r7, mem[80], and mem[84]. We also broke down the code to verify that these values were correct as shown in a separate file called lab3_professor-code.txt.



Entity: Architecture: Date: Sun Feb 14 18:17:19 PST 2016 Row: 1 Page: 1

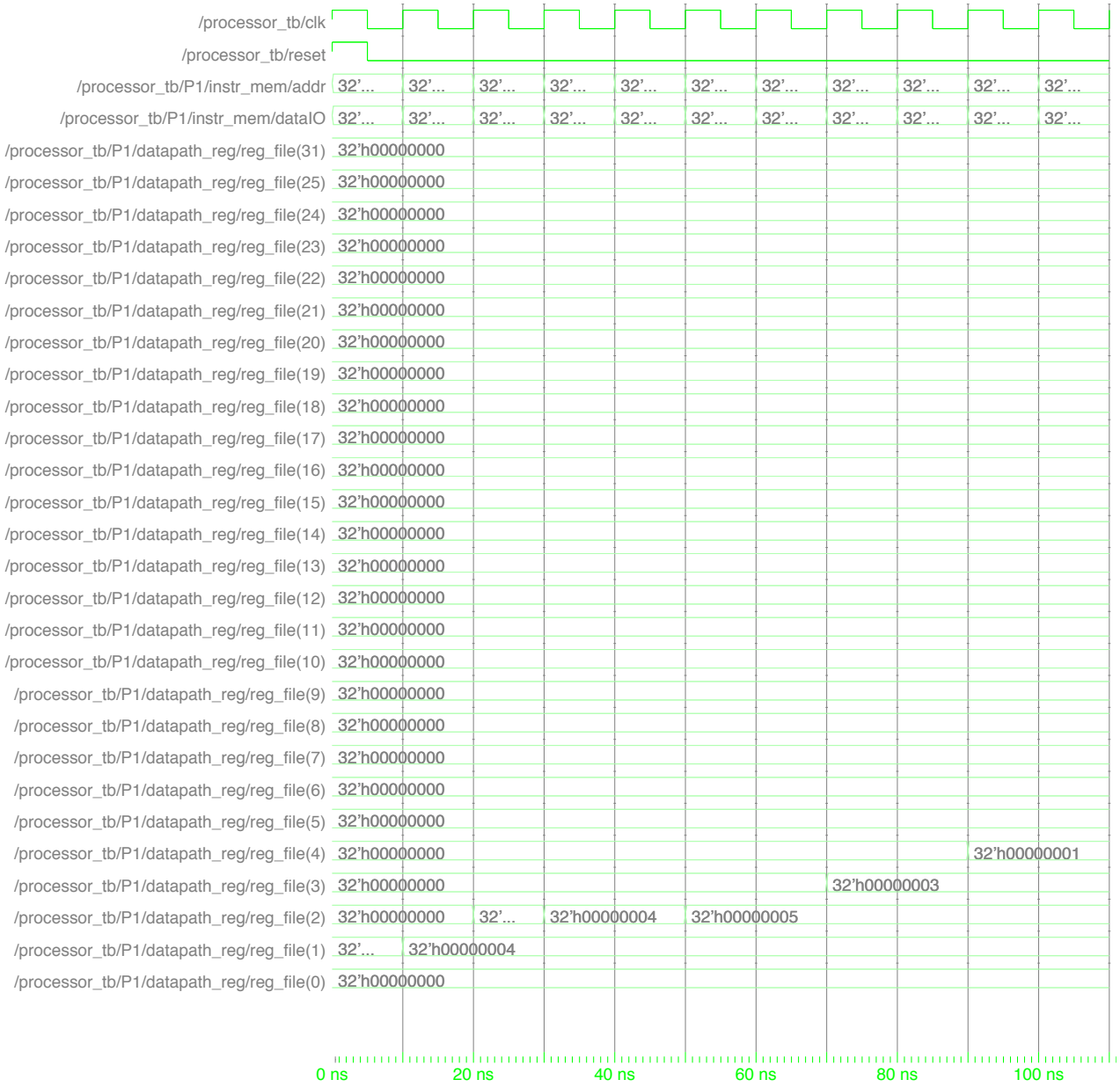

(7) 32’h00000000 32’h00000003
(6) 32’h00000000
(5) 32’h00000000 3... 32’h0000000B
(4) 32’h00000000 32’h00000007 32’h00...
(3) 32’h00... 32’h0000000C
(2) 3... 32’h00000005
(1) 32’h00000000
(0) 32’h00000000
0 ns
50 ns
100 ns


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file
(31) 32'h00000000
(30) 32'h00000000
(29) 32'h00000000
(28) 32'h00000000
(27) 32'h00000000
(26) 32'h00000000
(25) 32'h00000000
(24) 32'h00000000
(23) 32'h00000000
(22) 32'h00000000
(21) 32'h00000000
(20) 32'h00000000
(19) 32'h00000000
(18) 32'h00000000
(17) 32'h00000000
(16) 32'h00000000
(15) 32'h00000000
(14) 32'h00000000
(13) 32'h00000000
(12) 32'h00000000
(11) 32'h00000000
(10) 32'h00000000
(9) 32'h00000000
(8) 32'h00000000
150 ns
200 ns

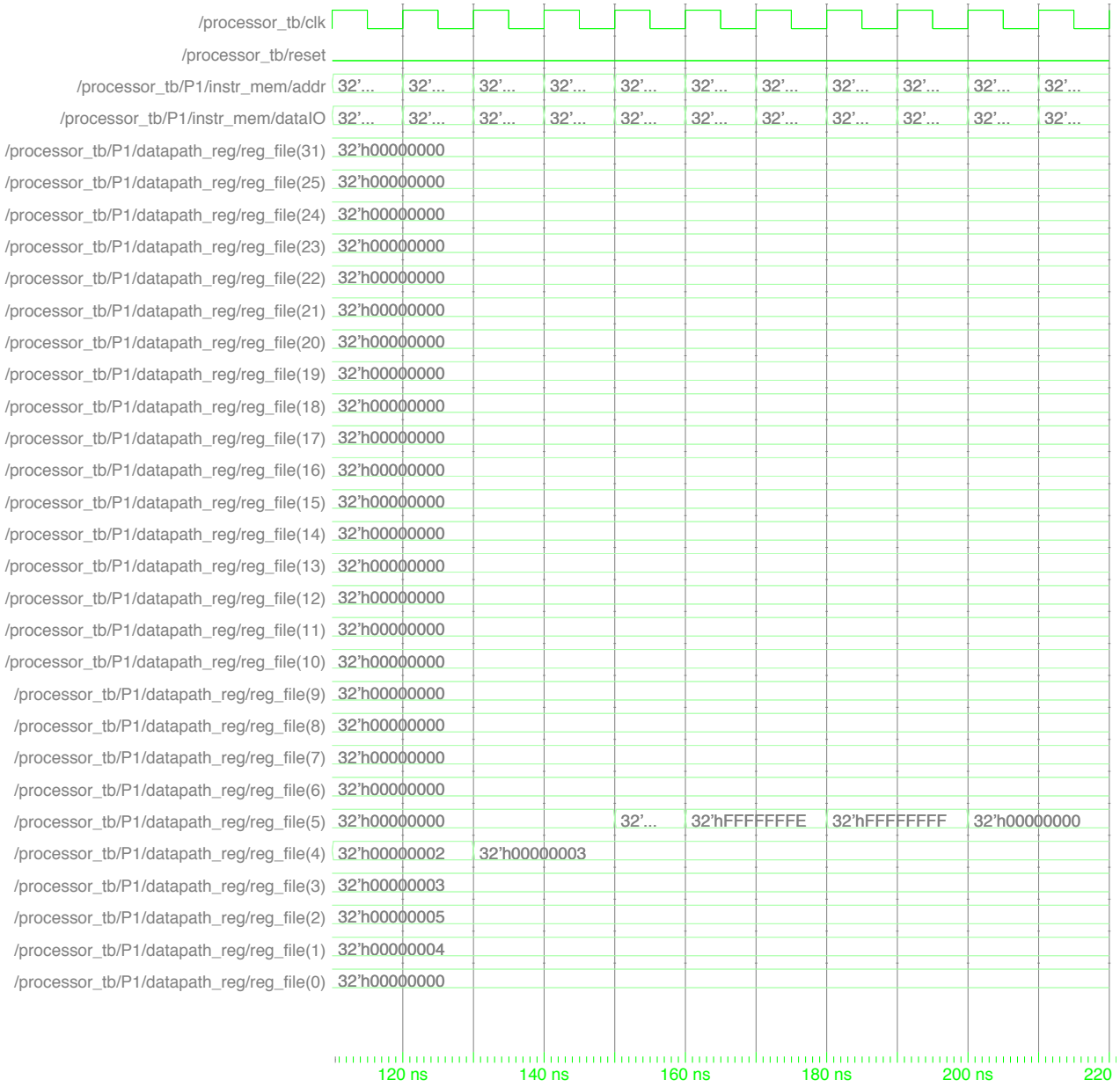

(7) 3... 3... 32'h00000007
(6) 32'h00000000
(5) 32'h0000000B
(4) 32'h00000001
(3) 32'h0000000C
(2) 32'h00000005 32'h00000007
(1) 32'h00000000
(0) 32'h00000000
150 ns
200 ns

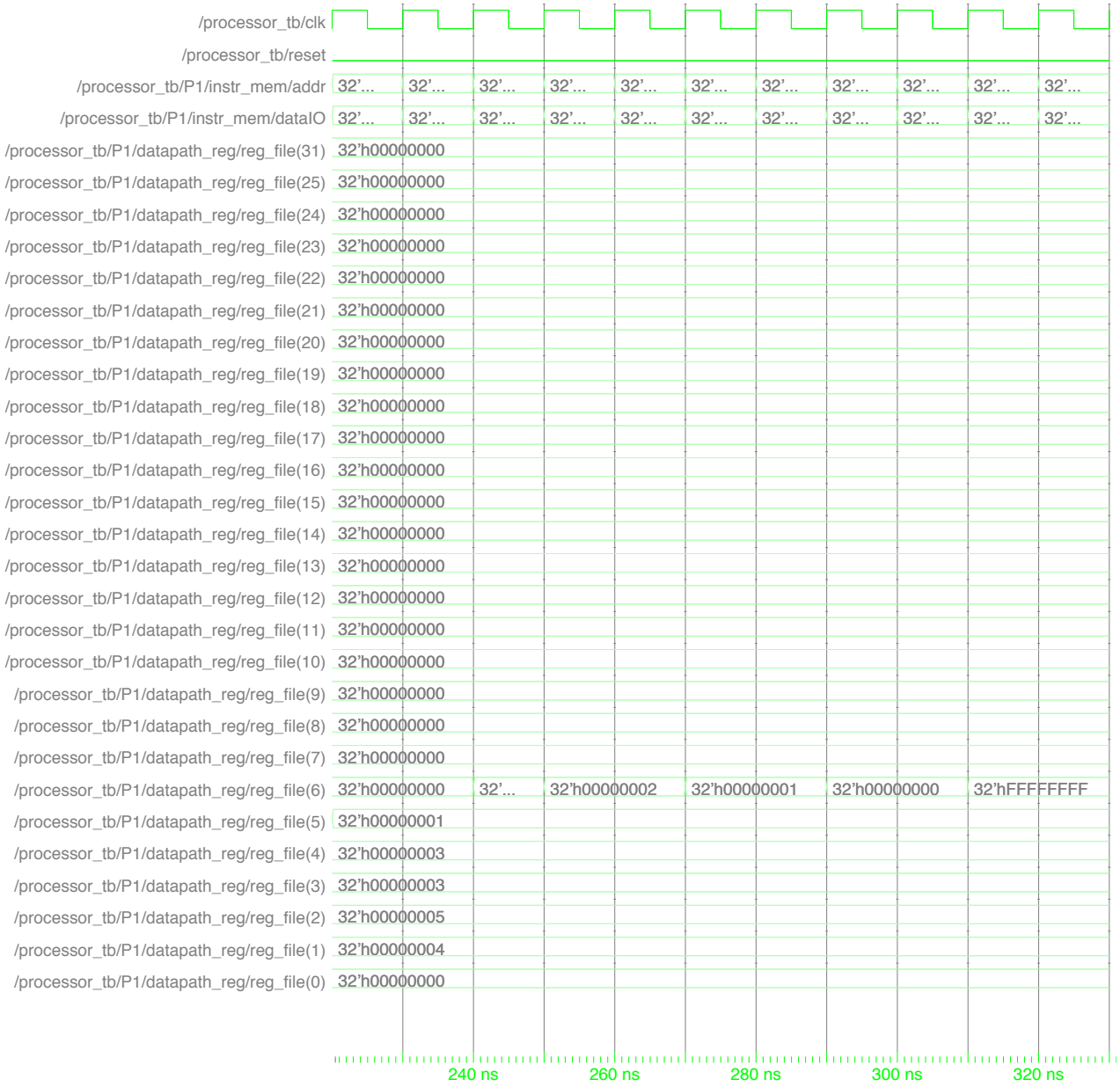
2.2.2 Waveform 2: Using self generated test code.

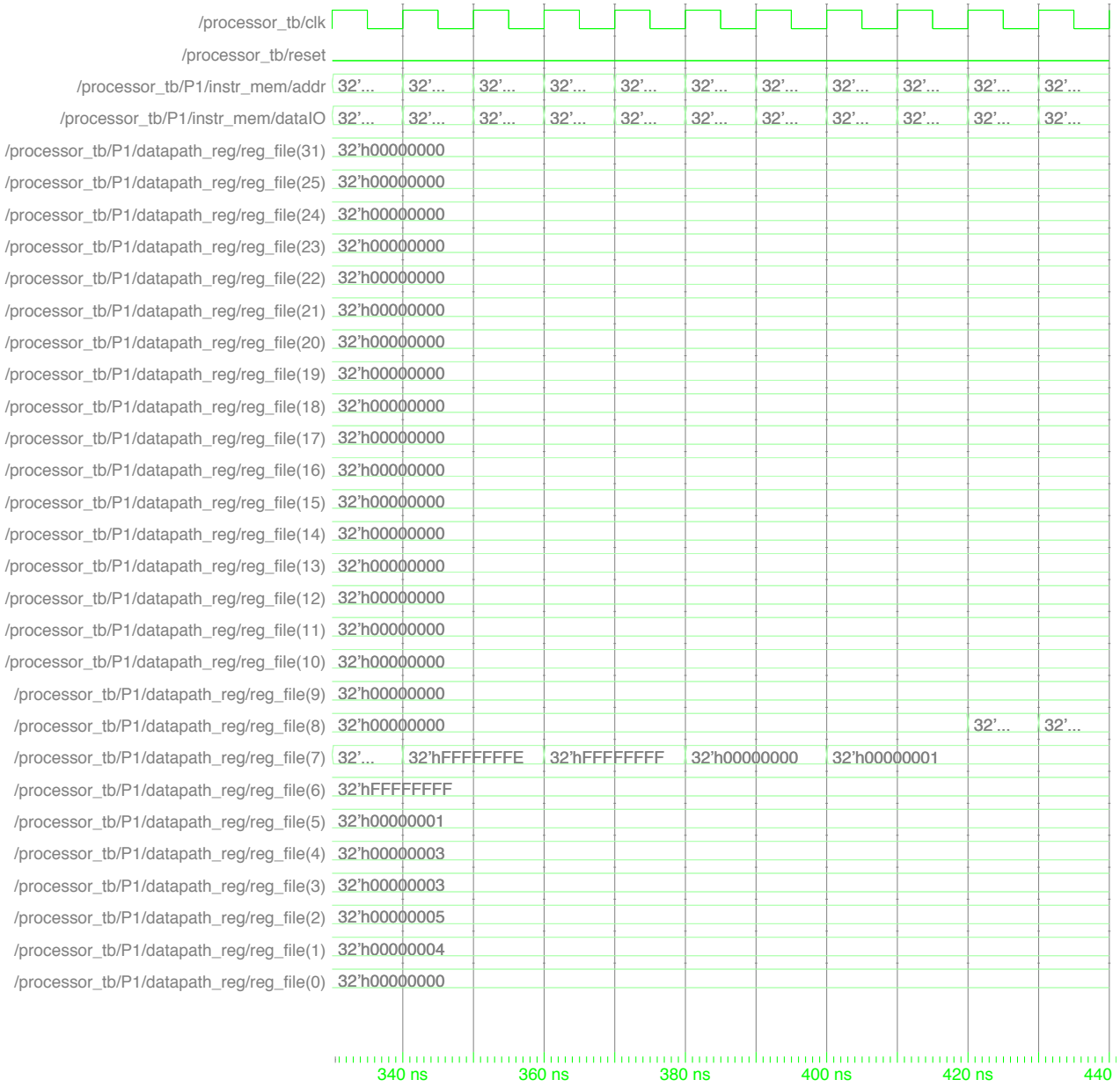
A separate file called lab3.asm provides the MIPS equivalent of the binary code.

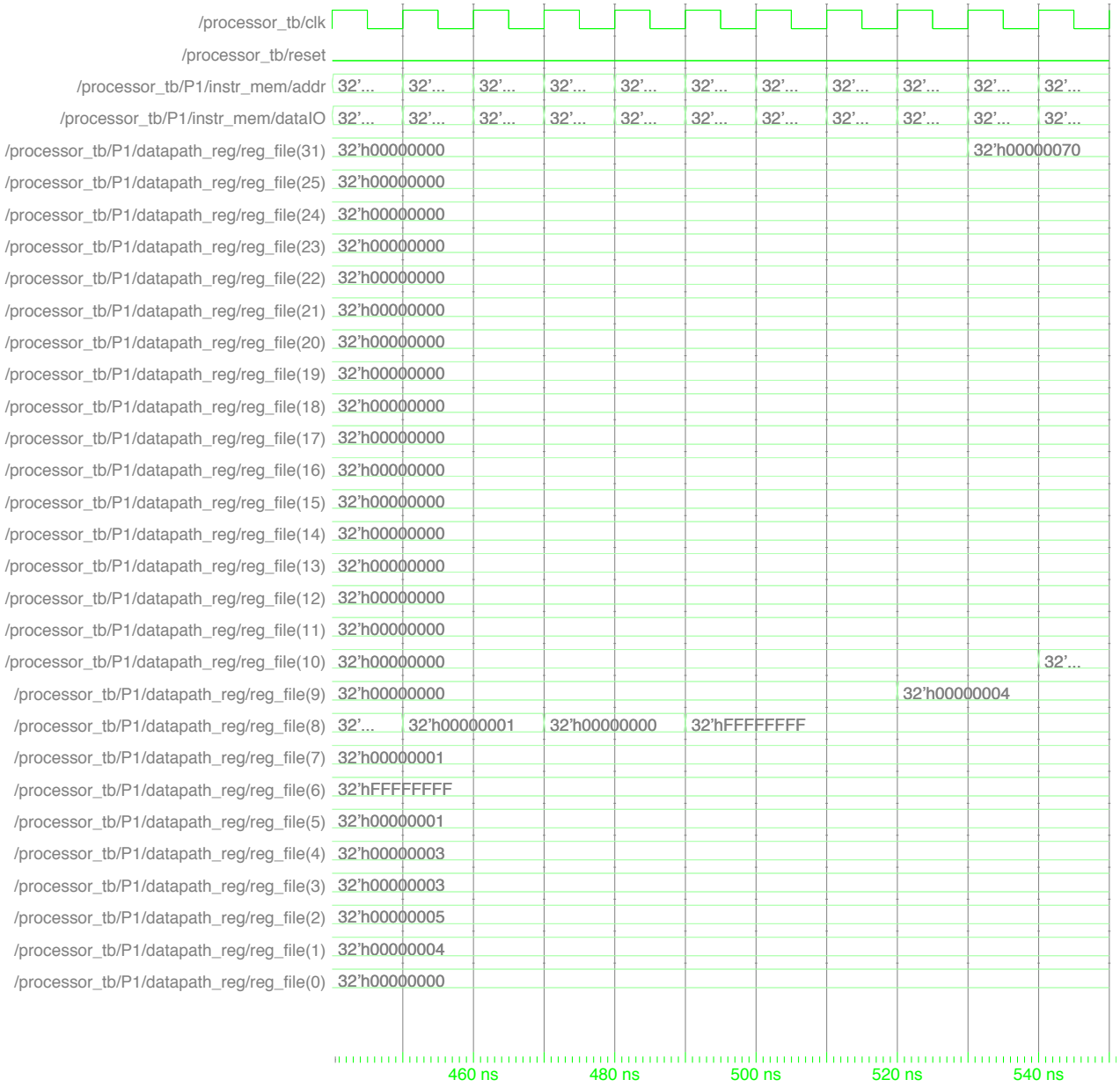


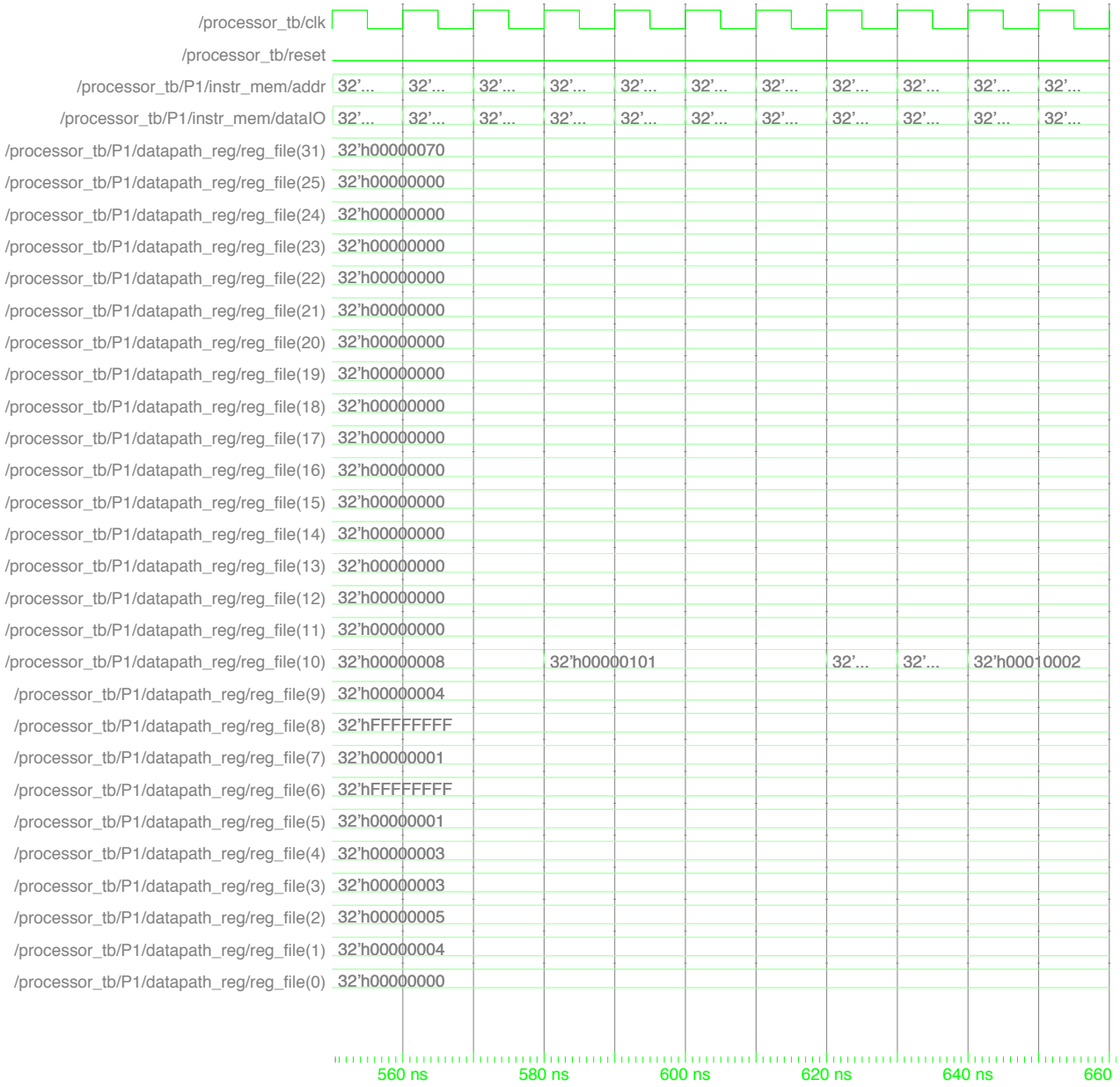
Entity: Architecture: Date: Sun Feb 14 16:49:34 PST 2016 Row: 1 Page: 1

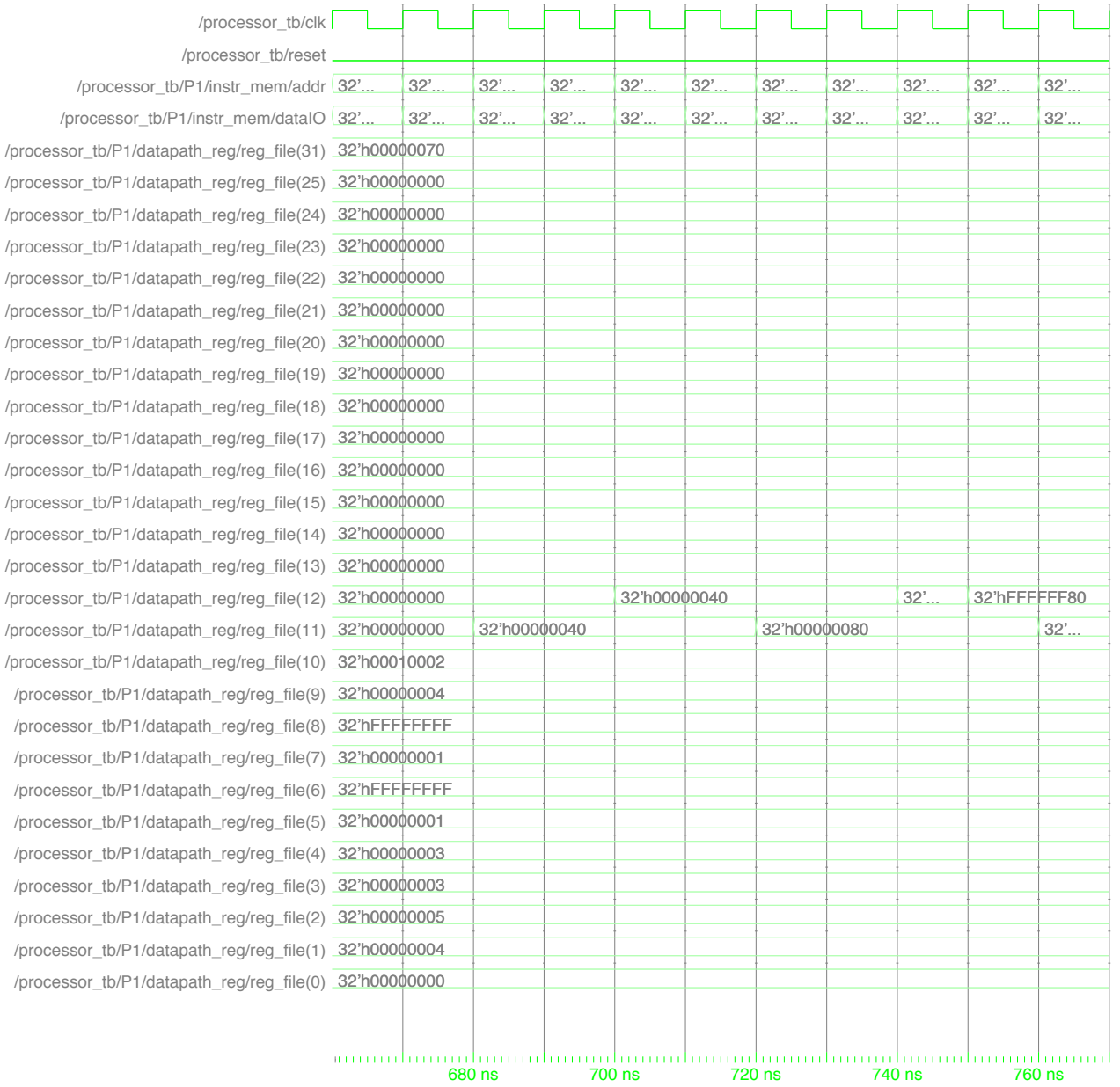

/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000000
32'hFFFFFFFE
32'hFFFFFFFF
32'h00000002
32'h00000003
32'h00000005
32'h00000004
120 ns
140 ns
160 ns
180 ns
200 ns
220

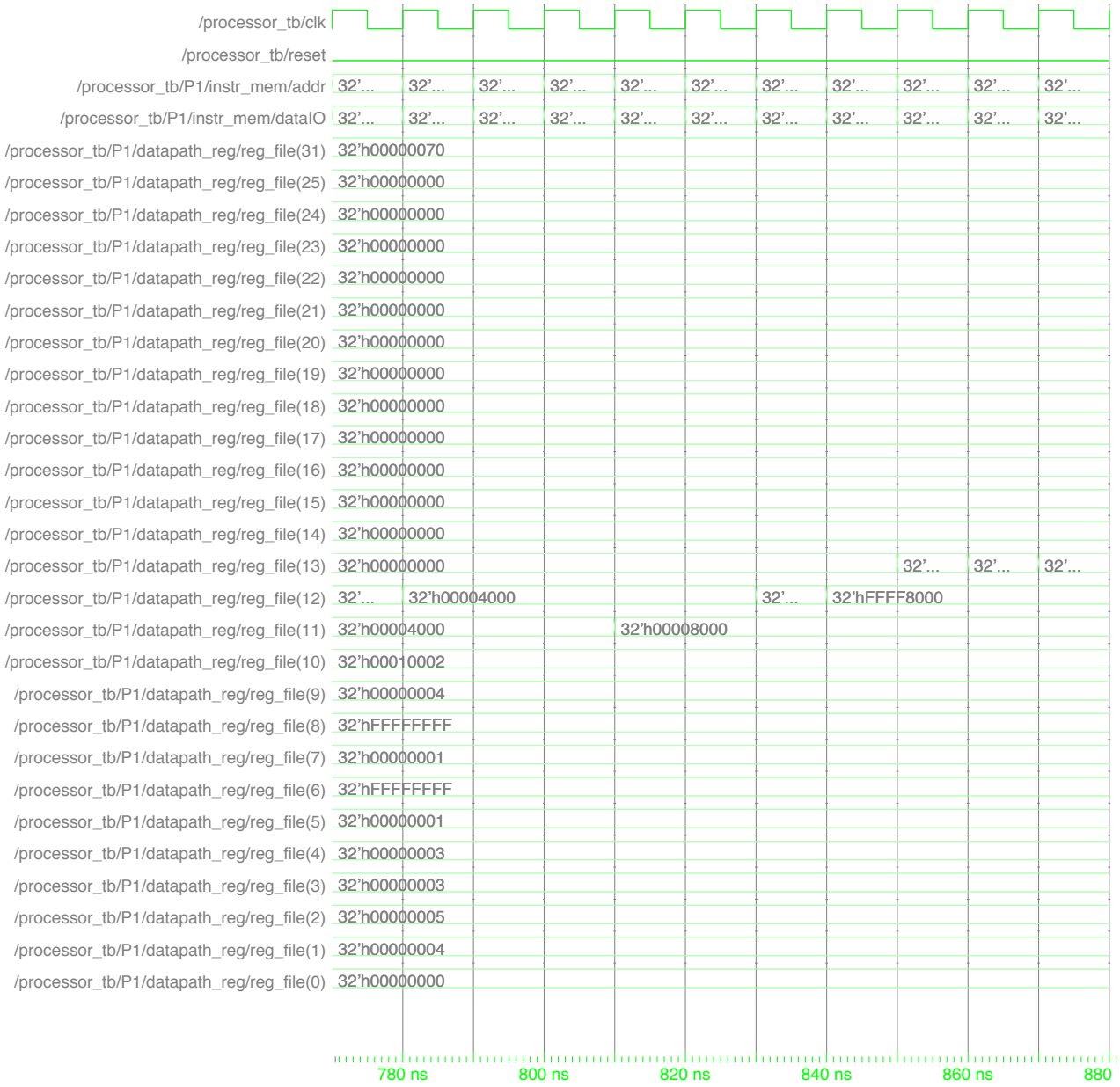

/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000000
32'h00000002
32'h00000001
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000005
32'h00000004
240 ns
260 ns
280 ns
300 ns
320 ns

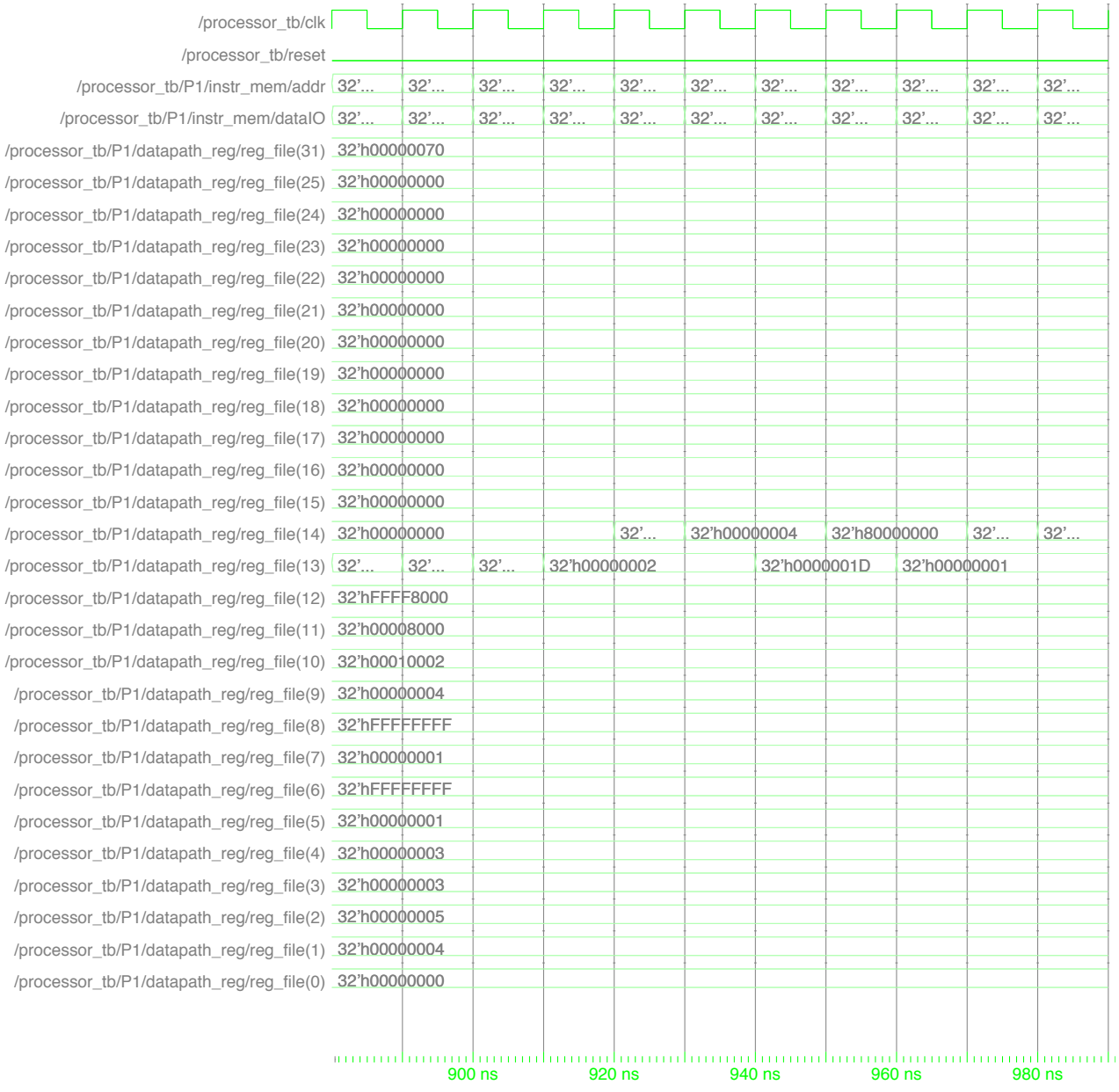

/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000000
32'hFFFFFFFE
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000005
32'h00000004
340 ns
360 ns
380 ns
400 ns
420 ns
440

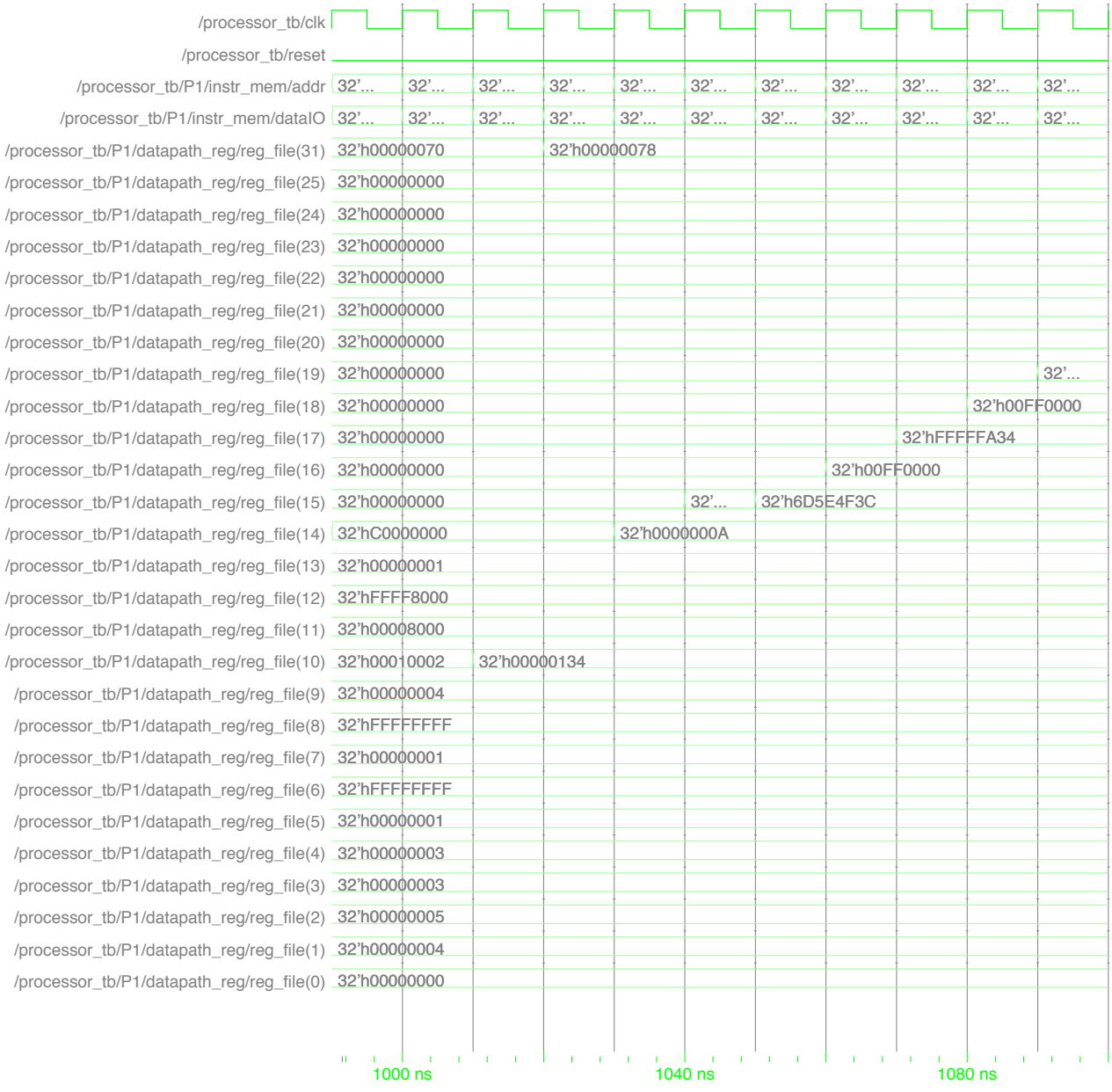

/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000000
32'h00000070
32'h00000001
32'h00000000
32'hFFFFFFFF
32'h00000004
460 ns
480 ns
500 ns
520 ns
540 ns

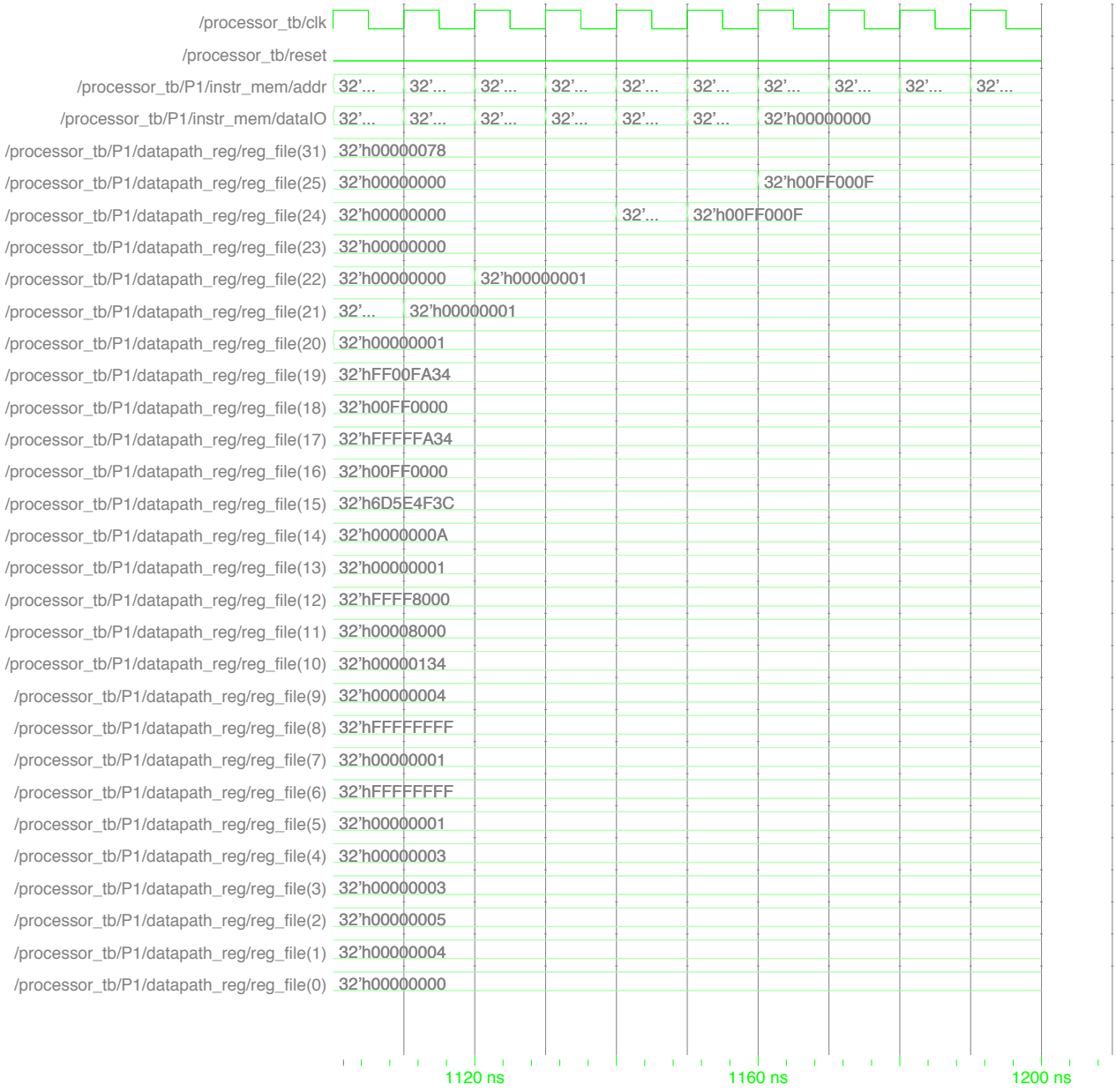

/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000070
32'h00000000
32'h00000008
32'h00000101
32'h00010002
32'h00000004
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000005
560 ns
580 ns
600 ns
620 ns
640 ns
660


/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000070
32'h00000000
32'h00000040
32'h00000080
32'hFFFFFF80
32'h00010002
32'h00000004
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000005
680 ns
700 ns
720 ns
740 ns
760 ns


/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000070
32'h00000000
32'h00004000
32'h00008000
32'h00010002
32'h00000004
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000005
32'hFFFF8000
780 ns
800 ns
820 ns
840 ns
860 ns
880


/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000070
32'h00000000
32'h00000004
32'h80000000
32'h0000001D
32'h00000001
32'h00000002
32'hFFFF8000
32'h00008000
32'h00010002
32'h00000004
32'hFFFFFFFF
32'h00000001
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000003
32'h00000005
32'h00000004
32'h00000000
900 ns
920 ns
940 ns
960 ns
980 ns


/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'h00000070
32'h00000078
32'h00000000
32'hFFFFFA34
32'h00FF0000
32'h6D5E4F3C
32'hC0000000
32'h0000000A
32'h00000001
32'hFFFF8000
32'h00008000
32'h00010002
32'h00000134
32'h00000004
32'hFFFFFFFF
32'h00000001
32'hFFFFFFFF
32'h00000001
32'h00000003
32'h00000003
32'h00000005
32'h00000004
32'h00000000
1000 ns
1040 ns
1080 ns


/processor_tb/clk
/processor_tb/reset
/processor_tb/P1/instr_mem/addr
/processor_tb/P1/instr_mem/dataIO
/processor_tb/P1/datapath_reg/reg_file(31)
/processor_tb/P1/datapath_reg/reg_file(25)
/processor_tb/P1/datapath_reg/reg_file(24)
/processor_tb/P1/datapath_reg/reg_file(23)
/processor_tb/P1/datapath_reg/reg_file(22)
/processor_tb/P1/datapath_reg/reg_file(21)
/processor_tb/P1/datapath_reg/reg_file(20)
/processor_tb/P1/datapath_reg/reg_file(19)
/processor_tb/P1/datapath_reg/reg_file(18)
/processor_tb/P1/datapath_reg/reg_file(17)
/processor_tb/P1/datapath_reg/reg_file(16)
/processor_tb/P1/datapath_reg/reg_file(15)
/processor_tb/P1/datapath_reg/reg_file(14)
/processor_tb/P1/datapath_reg/reg_file(13)
/processor_tb/P1/datapath_reg/reg_file(12)
/processor_tb/P1/datapath_reg/reg_file(11)
/processor_tb/P1/datapath_reg/reg_file(10)
/processor_tb/P1/datapath_reg/reg_file(9)
/processor_tb/P1/datapath_reg/reg_file(8)
/processor_tb/P1/datapath_reg/reg_file(7)
/processor_tb/P1/datapath_reg/reg_file(6)
/processor_tb/P1/datapath_reg/reg_file(5)
/processor_tb/P1/datapath_reg/reg_file(4)
/processor_tb/P1/datapath_reg/reg_file(3)
/processor_tb/P1/datapath_reg/reg_file(2)
/processor_tb/P1/datapath_reg/reg_file(1)
/processor_tb/P1/datapath_reg/reg_file(0)
32'...
32'h00000000
32'h00000078
32'h00000000
32'h00FF000F
32'h00000001
32'h00000001
32'hFF00FA34
32'h00FF0000
32'hFFFFFA34
32'h6D5E4F3C
32'h0000000A
32'hFFFF8000
32'h00008000
32'h00000134
32'h00000004
32'hFFFFFFFF
32'h00000003
32'h00000005
1120 ns
1160 ns
1200 ns

3 Newly Added Components

3.1 Adder

The adder takes 32-bit inputs and adds them together to create an 32-bit sum. This adder supports signed values.

Adder Port Description			
Port name	Port size	Port Type	Description
a	32	IN	addend a
b	32	IN	addend b
c	32	OUT	sum

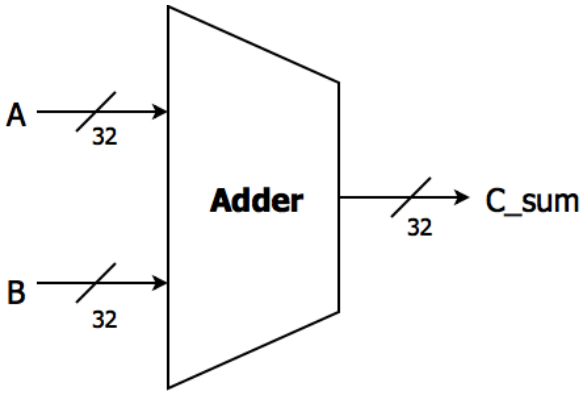


Figure 3: A diagram of an adder with two 32-bit inputs and one 32-bit output.

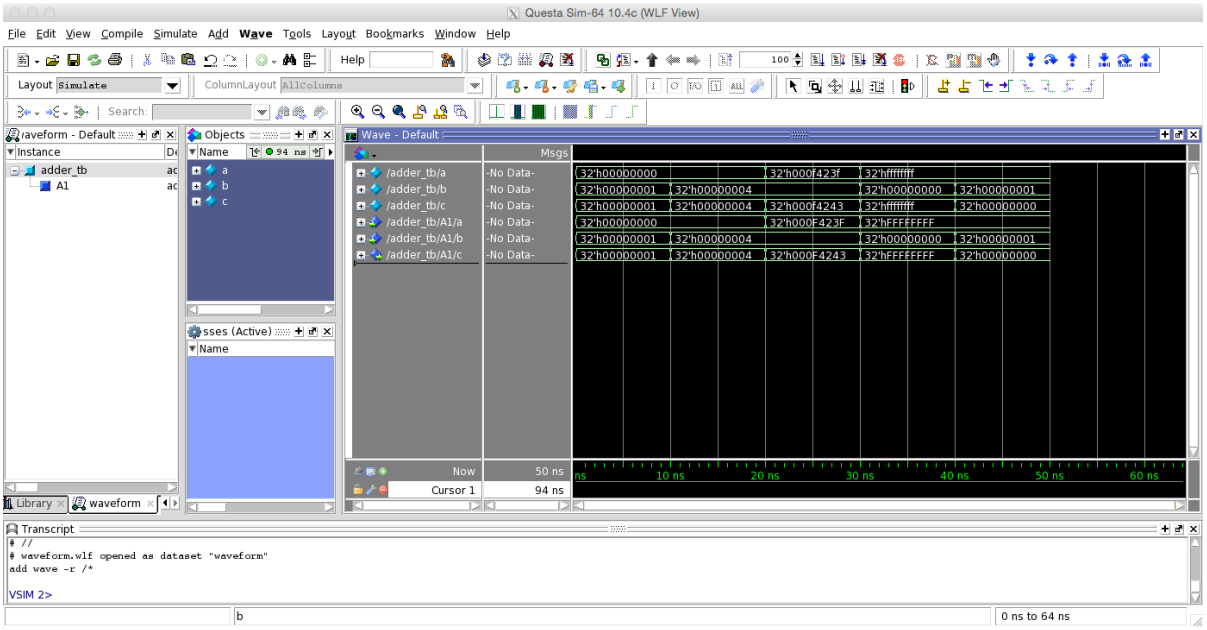


Figure 4: Adder waveform generated from a testbench to verify the design.

3.2 Shifter

The shifter shifts the input value 2 places to the left where zeros will be shifted in the least significant bit positions. The size of the input and output are the same.

Shifter Port Description			
Port name	Port size	Port Type	Description
a	32	IN	input value
o	32	OUT	shifted value

Figure 5: A diagram of a shifter that shifts inputs 2 places to the left.

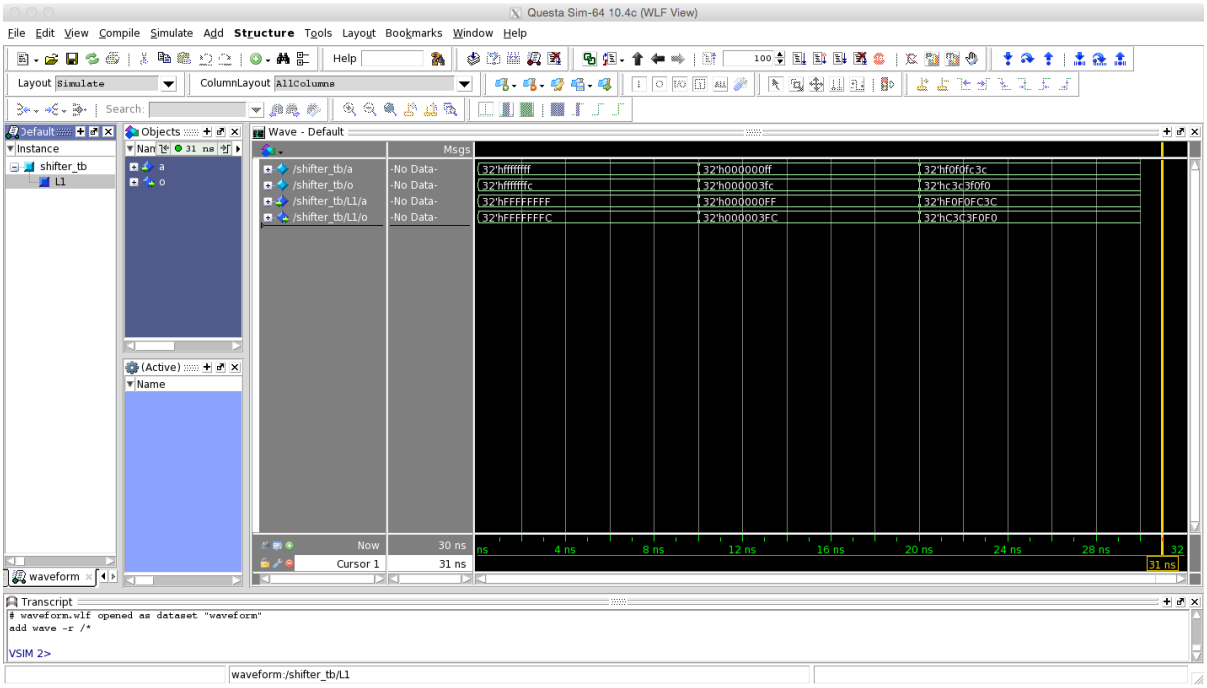


Figure 6: Shifter waveform generated from a testbench to verify the design.

4 Modified Components

The components in this section have been updated to support the branch, shift, and memory instructions.

4.1 Program Counter

The program counter is a register that stores the address of the instruction being executed. The address in the program counter is passed to the instruction memory. After the instruction is fetched, the counter gets incremented to go to the next address. In our case, the program counter is incremented by 4 using an adder as our design is byte-addressable. Our implementation of the program counter, takes in a clock signal, reset, and input. It outputs a 32-bit address. The program counter was changed from the previous design so that it no longer increments its output by itself. Instead, an adder takes the output

of the program counter.

Program Counter Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
rst	1	IN	reset bit
input	32	IN	incremented value
output	32	OUT	outputs an address

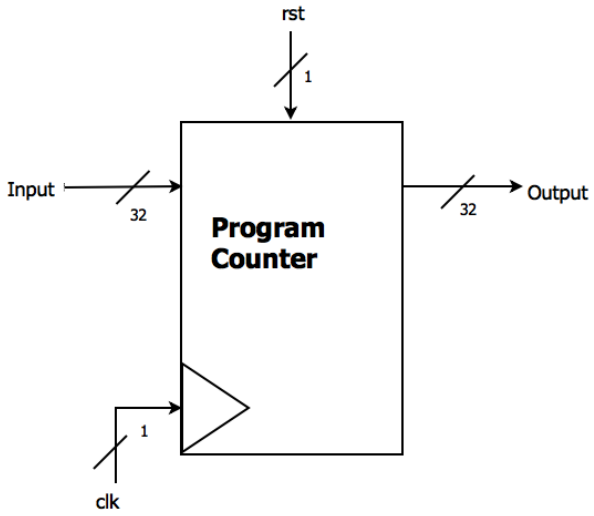


Figure 7: Program Counter Diagram

4.2 Controller

The controller design has changed quite a bit from the previous design and is one of the most important blocks, if not the most important block, in ensuring that the processor works correctly. It now takes the whole instruction from the Instruction Memory and not just the opcode and function value. The controller determines raddr1, raddr2, and wdata values that goes into the Register File. It will also pass the immediate value to the sign extension unit. Out of the 7 multiplexers in the design, the controller provides 6 of multiplexers with the proper select lines.

In this assignment, the controller logic has grown significantly, as the controller has to ensure that the register file and data memory do not accidentally get overwritten in the presence of an opcode. As such, the controller enables the write signal for the register file and data memory only for opcodes where writing to those blocks is necessary. Also, the controller makes heavy use of the opcode and func fields of the instruction to determine the correct function code for the ALU, ALU operands, and the select values for the various multiplexers in the design.

Controller Port Description			
Port name	Port size	Port Type	Description
Input	32	IN	Current instruction
ALUControl	6	OUT	ALU function code
ALUSrcA	1	OUT	Selecting ALU Operand A
ALUSrcB	1	OUT	Selecting ALU Operand B
MemtoReg	1	OUT	Selecting data to write to regfile
MemWrite	1	OUT	Enable memory write
RegWrite	1	OUT	Enable regfile write
WdataSrc	1	OUT	Used for jal/jalr
Branch	1	OUT	controls branch instruction
Jump	1	OUT	control for jump instruction
Dsize	2	OUT	goes to RAM
Dsgnd	1	OUT	goes to RAM
shamt	32	OUT	shift amount
jump_imm	26	OUT	jump immediate
raddr1	5	OUT	goes to register file
raddr2	5	OUT	goes to register file
waddr	5	OUT	goes to register file
imm	16	OUT	immediate value goes to sign extension unit
jtype	1	OUT	Choose between register/immediate for jump

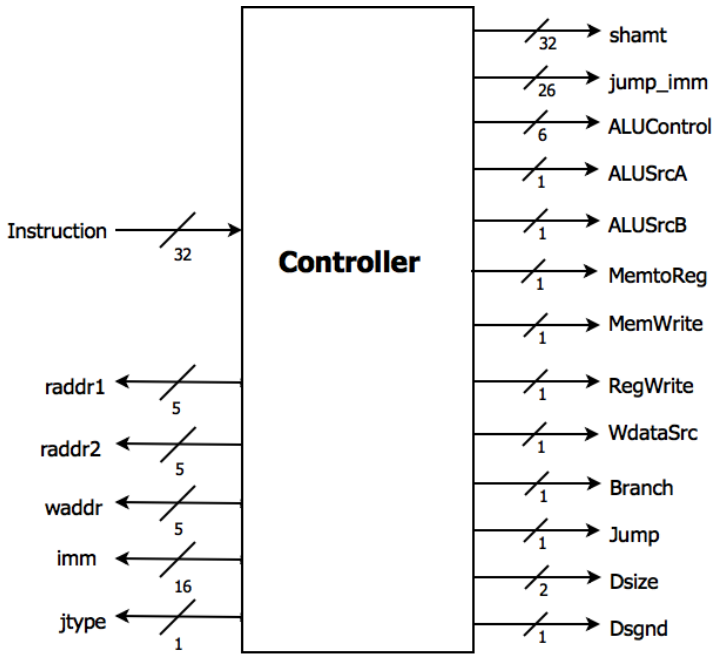


Figure 8: Diagram of the controller

4.3 ALU

The ALU performs arithmetic and logic operations on two 32-bit operands and produces a 32-bit output. It also provides a one-bit value for the branch instruction.

ALU Port Description			
Port name	Port size	Port Type	Description
Func_in	6	IN	opcode from controller
A_in	32	IN	operand A
B_in	32	IN	operand B
O_out	32	OUT	output from ALU
Branch_out	1	OUT	for branch instruction

ALU Function Description			
Instruction	Description	Function Code	Comments
nop	nothing	"000000"	
add/addi	rd ← rs + rt / rd ← rs + immediate	"100000"	
addu/addiu	rd ← rs + rt / rd ← rs + immediate	"100001"	
sub	rd ← rs - rt / rd ← rs - immediate	"100010"	
subu	rd ← rs - rt / rd ← rs - immediate	"100011"	
and/andi	rd ← rs AND rt / rd ← rs AND immediate	"100100"	
or/ori	rd ← rs OR rt / rd ← rs OR immediate	"100101"	
xor/xori	rd ← rs XOR rt / rd ← rs XOR immediate	"100110"	
nor	rd ← rs XOR rt / rd ← rs NOR immediate	"100111"	
slt/slti	Set rd if rs < rt / set rd if rs < immediate	"101000"	If the condition is satisfied set else reset destination
sltu/sltiu	Set rd if rs < rt / set rd if rs < immediate	"101001"	If the condition is satisfied set else reset destination
sll	B << A	000X00	
srl	B >> A	000X10	
sra	B >>> A	000X11	
bltz	A < 0	111000	
bgez	A >= 0	111001	
beq	A == B	111100	
bne	A ! = B	111101	
blez	A <= 0	111110	
bgtz	A > 0	111111	
nop	nothing	others	Any other function code does nothing

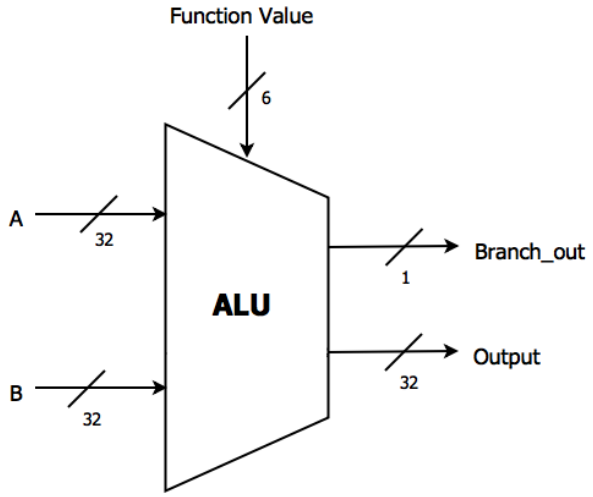


Figure 9: The ALU has 3 inputs and two outputs. There are two 32-bit inputs that act as the operands and a 6-bit input that lets the ALU know which operation to perform. The result is then sent to the output and a value for the branch.

4.4 RAM

The data memory is 512 lines with one word per line. It has a single read/write port. On the rising edge of the clock, if write enable is 1, it writes data into the input address. On the falling edge of the clock, it reads the data from the input address.
The main change to the data memory is that it now receives a two-bit input, called dsize, for determining the size of the data to be read or written. This is because we now support loading and storing bytes and half-words, in addition to words. The data memory uses bits 0 and 1 of the address to determine where to store a byte within a 32-bit line, and bit 1 to determine where to store a half-word within a 32-bit line. Bits 2 through 10 specify the word, or line, where the data is to be written. Another addition is a 1-bit input called dsgnd, which specifies whether the byte or half-word to be loaded is signed or unsigned. If signed, the byte or half-word is sign-extended.

RAM Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
we	1	IN	write enable
dsgnd	1	IN	for LB and LH instructions
dsize	2	IN	for SB and SH instructions
addr	32	IN	address
dataI	32	IN	input data
dataO	32	OUT	output data

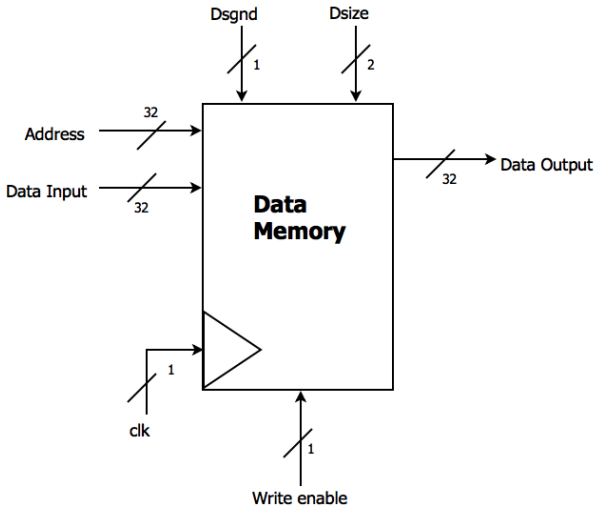


Figure 10: Random Access Memory

5 Unmodified Components

These components have not been modified from the previous design and thus the waveforms associated with the testbenches are not included.

5.1 Instruction Memory

The instruction memory for our design is implemented as a ROM (read-only memory). It is preloaded with instructions provided in the rom.dat file. The address is sent from the program counter as 32-bits, but the instruction memory will only take the lower 9-bits. This makes our instruction memory size $2^9 - 1$ and each line is 32-bits long. Initially, we set the size to $2^{32} - 1$. However, we encountered errors where QuestaSim and Cadence would not allow an array size greater than $2^{30} - 1$. Since we were not going to use that many registers, we shrunk the size.

Instruction Memory Port Description			
Port name	Port size	Port Type	Description
addr	32	IN	address for the location of instruction, only takes the lower 9-bits
dataIO	32	INOUT	outputs an instruction

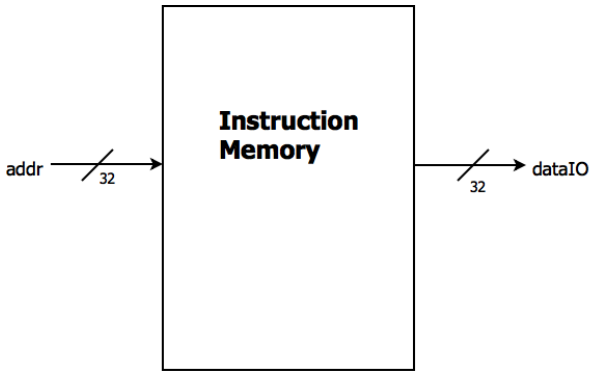


Figure 11: The instruction memory has one 32-bit input and one 32-bit output.

5.2 Register File

The register file has 32 registers, each 32 bits wide. There are 2 read ports and one write port. The read ports are asynchronous and the write port is synchronous. The register file has a synchronous reset signal and a write enable signal.

Register Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
rst_s	1	IN	synchronous reset
we	1	IN	write enable
raddr_1	5	IN	read address 1
raddr_2	5	IN	read address 2
waddr	5	IN	write address
rdata_1	32	OUT	read data 1
rdata_2	32	OUT	read data 2
wdata	32	IN	write data

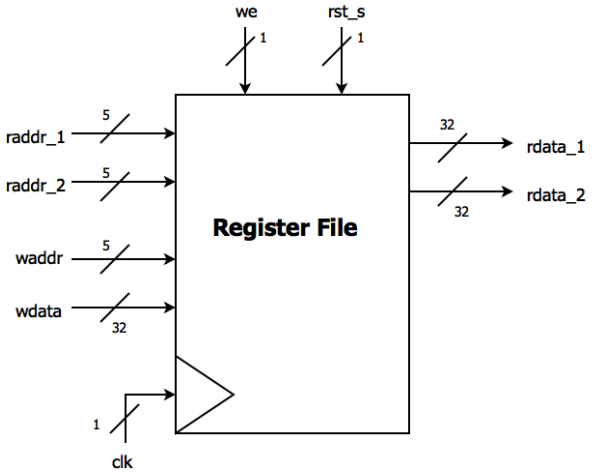


Figure 12: Register file

5.3 Multiplexer

In general, the multiplexer chooses an output from several possible inputs based on the value of the select signal. Our design uses seven 2:1 multiplexers where 6 out of the 7 select signal inputs come from the controller block. The signals from the controller are: WdataSrc, ALUSrcA, ALUSrcB, MemtoReg, jtype, and jump.

1. WdataSrc chooses what value should be written in the register file.
2. ALUSrcA decides whether operand A into the ALU will be a shifted value or value attained from the register file.
3. ALUSrcB chooses from either the register file input or sign immediate input as the output for operand B in the ALU.
4. MemtoReg picks between the ALU result for R-type instructions or read data input from data memory for the load word instruction.
5. Jtype determines which value would be use for the jump address.

6. Jump signal chooses which address the program counter will go to next to attain the next instructions.

7. The seventh multiplexer is controlled by the branch result attained from an AND operation.

Multiplexer Port Description			
Port name	Port size	Port Type	Description
s	1	IN	select line
a	32	IN	data 1
b	32	IN	data 2
o	32	OUT	selected data

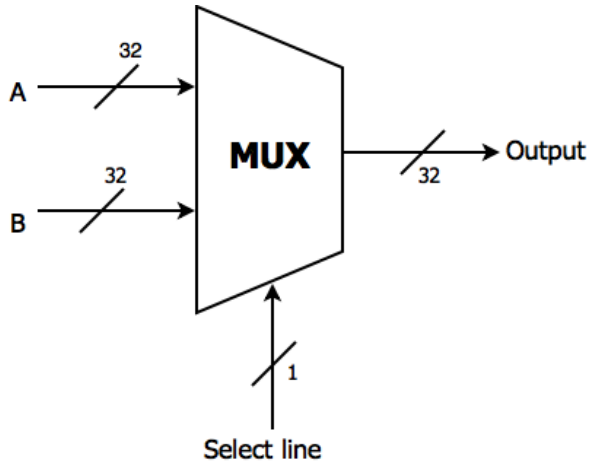


Figure 13: 2-to-1 MUX

5.4 Sign Extension Unit

The sign extension unit performs sign extension on the 16-bit immediate value provided by the controller to make it 32-bits wide, so that it can be used in as a second operand in the ALU.

Sign Extender Port Description			
Port name	Port size	Port Type	Description
input	16	IN	input value
output	32	OUT	extended value to 32-bits

Figure 14: Takes in a 16-bit value and extends it to 32-bits.

6 Synthesis

In the process of synthesizing our processor design, we ran into a number of issues. First, we had to use the std_logic_unsigned package instead of the numeric_std_unsigned package, and as a result, had to use conv_integer instead of to_integer. Second, we had to use the built-in SRAM library, through the provided example, instead of our own design for the data memory, because of issues with using variable

array indices inside a process block. Third, we had to change the name of the processor's clock port to clk from ref_clk, as the synthesis tool did not accept any other name. Also, we had to hardcode instructions into the instruction memory, rather than preloading it, and the instruction memory could not be empty.

After fixing these issues, we were able to synthesize our design successfully and obtain the following metrics for our design:

- Area
  - Total area: 76249 square microns
  - Area for black box regions: 50667 square microns
  - Area for combinational components: 10734 square microns.
- Power
  - Total power: 17 mW
  - Leakage power: 13.3 mW
  - Internal power: 3.57 mW
  - Switch power: 108 uW
- Maximum Frequency
  - Minimum clock period: 2.67 ns
  - Maximum frequency: 375 MHz

7 Conclusion

By extending the simple single-cycle processor design from the previous lab, we were able to see the significant increase in complexity of the entire system. The number of components and wires in the processor block grew significantly, and the logic for the various outputs in the controller became much more complex.

After synthesizing our processor design, we were able to evaluate the performance and efficiency of our design, in terms of speed, size, and power consumption. Synthesis of the design also helped us gain an understanding of how actual processors are composed, and how we can change our design in order to reduce power consumption and chip area, and increase speed.