Final Lab - Pipeline RISC-V Datapath and Control

Manpreet Dhindsa 15859742

March 19, 2018

1 Introduction

For this assignment I designed a Pipelined RISC-V Datapath and Control System which carried out RType, RTypeI, Load, Store, LUI, and AUIPC Operations. To begin the assignment I carefully looked through my original Single-Cycle RISC-V code (all instructions working properly) that I implemented for the Lab 2 submission in an attempt to understand the concept of a pipelined processor and to get a general idea of the changes implementing a pipelined processor would require of the single-cycle code. Upon reviewing my code and the Pipelined.pdf file I still had confusion on how to implement the pipelined processor, but after meeting Nazanin in office hours, the concept of the pipelined processor became clear and I decided to begin by implementing four new floprs into the Datapath.sv file.

2 RISCV.sv and Datapath.sv

In order to get the Pipelined Processor working properly large revisions were made to the Datapath.sv and the RISCV.sv files. I began by breaking apart my original Datapath.sv file into five different stages IF (Instruction Fetch), ID (Instruction Deocde), EXE (Execution), MEM (Memory), and WB (Write Back). Breaking the code into five different stages was not too difficult, I simply read off the Datapath figures from the Pipelined.pdf file to decide which modules go into what stage of pipelining. Once I had a general idea of how to split the modules into their individual stages I implemented 4 distinct floprs between the five stages of pipelining; one between IF and ID, another between ID and EXE, a third between EXE and MEM, and a final one between MEM and WB. I decided to implement my pipelined processor by using floprs instead of register files (professor had recommended using register files and structs to pass the information), and I decided to pass my information as a single variable of concatenated counterparts (idea received from the TA). For example, the first flopr received as input a logic variable which was assigned as a concatenation of the PC and the instruction. Then in the next stage, Instruction Decode, if I wanted to use PC I extracted it by using input[40:32]. Defining and implementing the five stages of the pipelined processor as well as the four floprs along with their inputs and outputs was not very difficult, just very tedious.

However, I did encounter some difficulty understanding how to pass the PC into the different stages (when trying to update PC with PC+imm instead of PC+4), therefore; for the purposes of my submission, I simply updated PC to PC+4 because I did not implement the Branch and Jump Instructions (I excluded the Branch and Jal instructions because I learned that these instructions were extra credit and time was running short because of finals). So my solution to not being able to pass and update PC was to simply update by PC+4 and exclude the clock muxs altogether. Another Issue I faced was understanding how to pass the instruction to the Controller and ALUController properly

since these modules are defined in the RISCV.sv file. My solution was to move these modules out of the RISCV.sv file altogether and instantiate them into the Datapath.sv file instead. By doing so I was able to make sure that the right information was being passed into the Controller and ALUController for each instruction. The output signals from the Controller such as the MemWrite and ALUSrc were all passed through the floprs as well (Idea came from the Pipelined.pdf file). This idea for defining 4 floprs and 5 stages of pipelining was completed without too much difficulty when Issues such as updating the PC to PC+4 and also passing the Controller signals was resolved.

As for the Hazards and forwarding I had a very difficult time implementing them. For forwarding I followed the Instruction from piplelinedhazard.pdf and I implemented the forwarding unit exactly as listed. Upon creating the forwarding unit, I used it to then implement muxs to use the information of ForwardA and ForwardB to correctly check and pass the correct information based of if forwarding was detected. I was unable to get this to work properly. As a result I did not move on to implement the hazards check. This was a very tedious and difficult code, but in the end, time played a major factor (I was unable to attend TA office hours for help) and therefore, I was unable to implement forwarding and hazards correctly.

3 Synthesis Results

Upon completing the synthesis of my pipelined processor, I recorded the following values: Critical Path Length = 2.13, Critical Path Slack = -0.15, and Area = 87178(Combinational). The values recorded from my single-cycle processor were the following: Critical Path Length = 5.41, Critical Path Slack = -3.42, and Area = 88999(Combinational). In comparison, the pipelined values were better.

4 Conclusion

In conclusion, I was able to get the Pipelined Processor working properly. For me personally, I had a lot of difficulty with moving the Controller and ALUController into the Datapth.sv file and then figuring out how to correctly pass the Control Signals into the different stages of the pipelined processor. However, after careful reading of the Pipelined.pdf file and many hours of tedious coding, I was able to create a pipelined processor. In conclusion, this was an extremely challenging code, I was unable to get the hazards and forwarding working properly. Below, I have attached Screenshots of the output waveform, as well as a JPEG image of the Datapath I implemented.

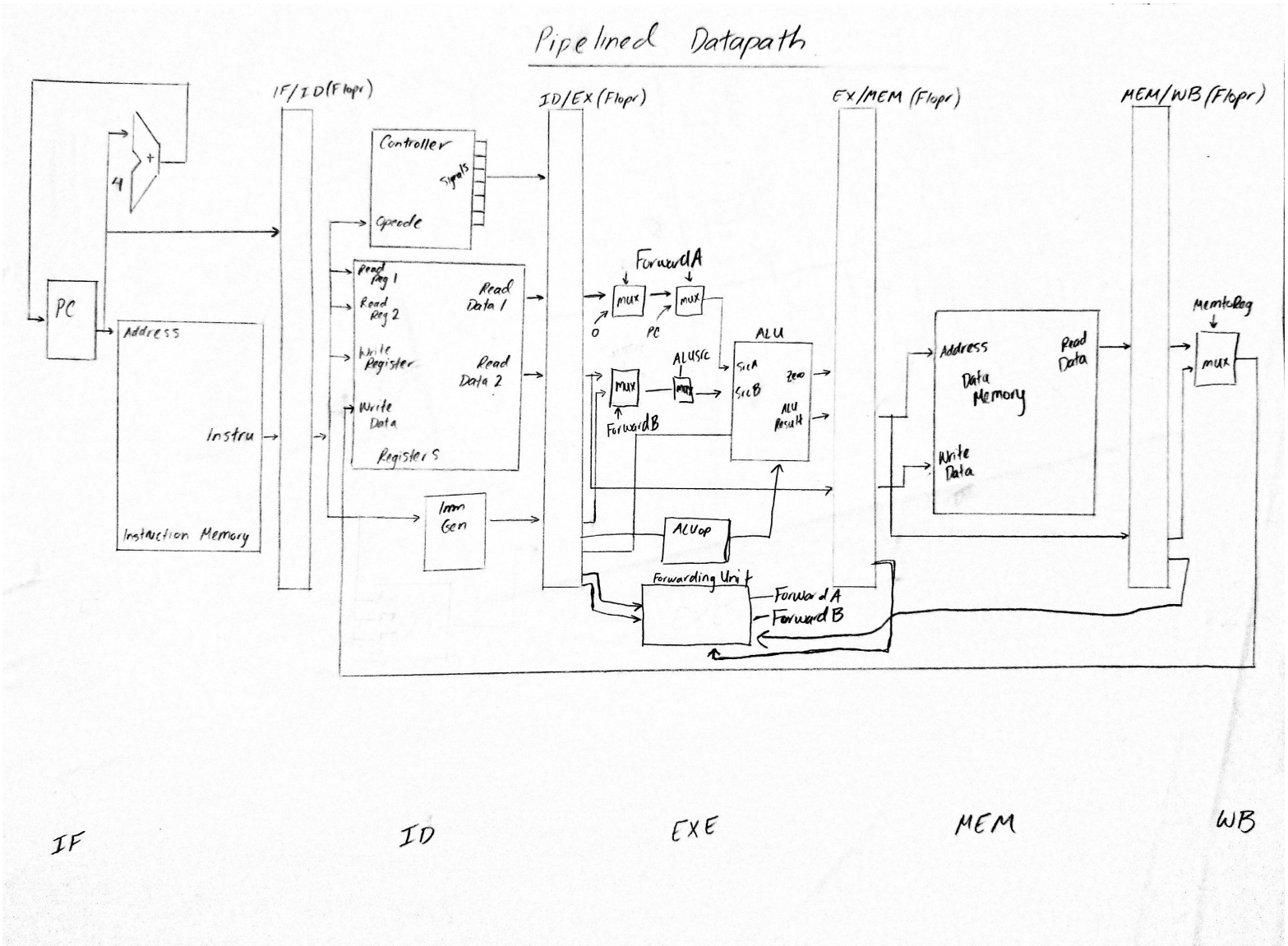

Figure 1: Pipelined Datapath.

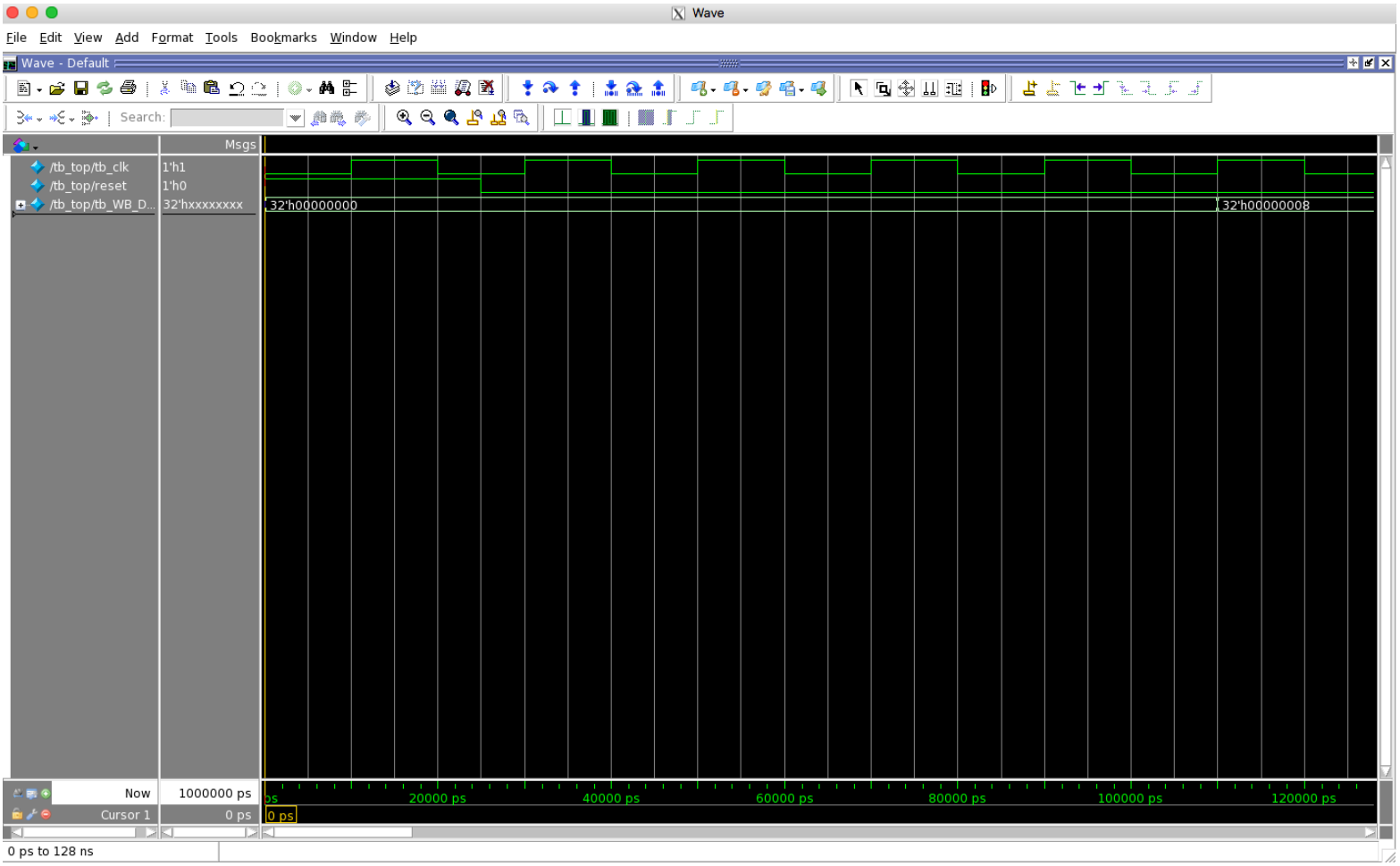
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000000
32'h00000008
Now
1000000 ps
Cursor 1
0 ps
20000 ps
40000 ps
60000 ps
80000 ps
100000 ps
120000 ps
0 ps to 128 ns

Figure 2:

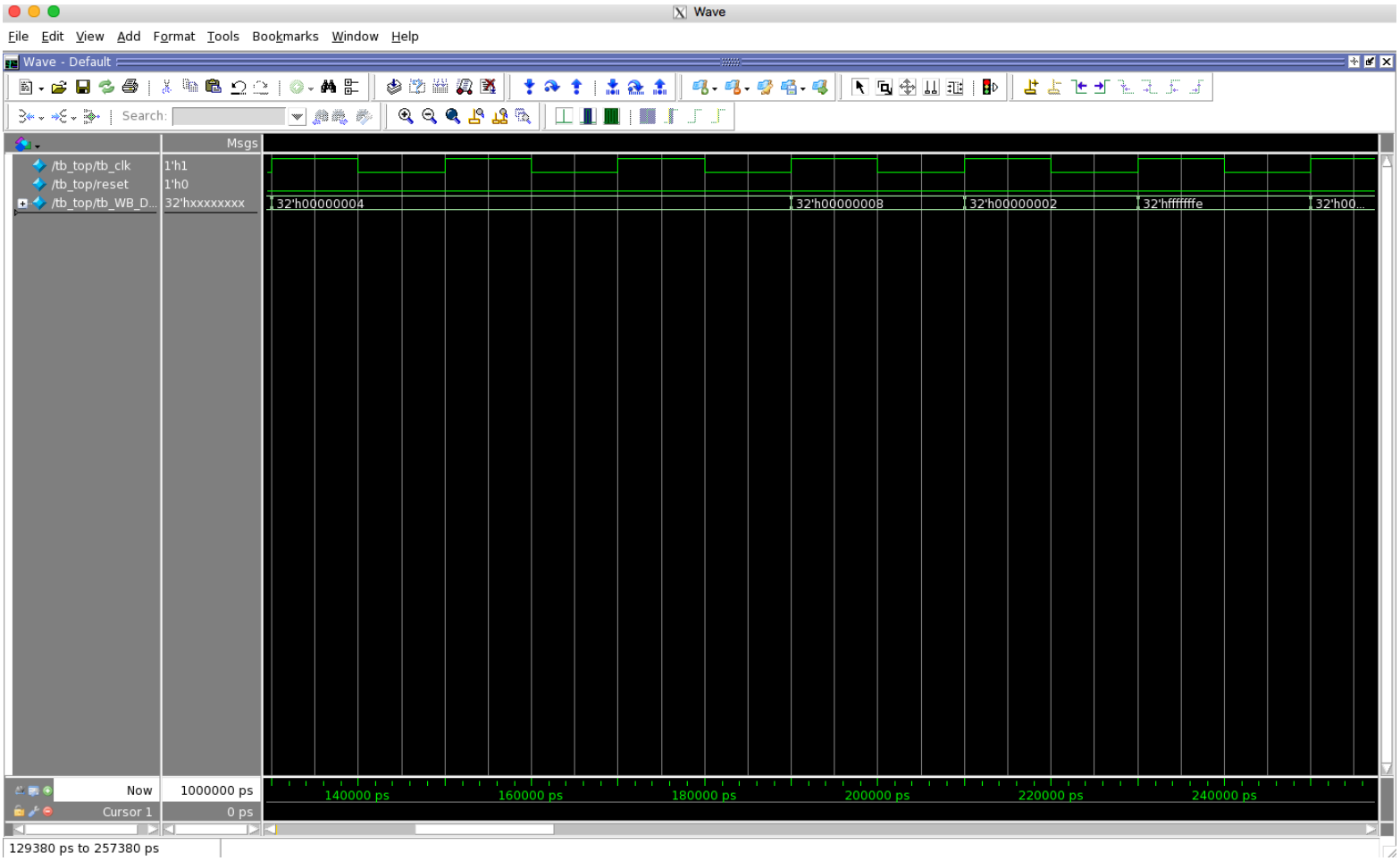
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000004
32'h00000008
32'h00000002
32'hfffffffe
32'h00...
Now
1000000 ps
Cursor 1
0 ps
140000 ps
160000 ps
180000 ps
200000 ps
220000 ps
240000 ps
129380 ps to 257380 ps

Figure 3:

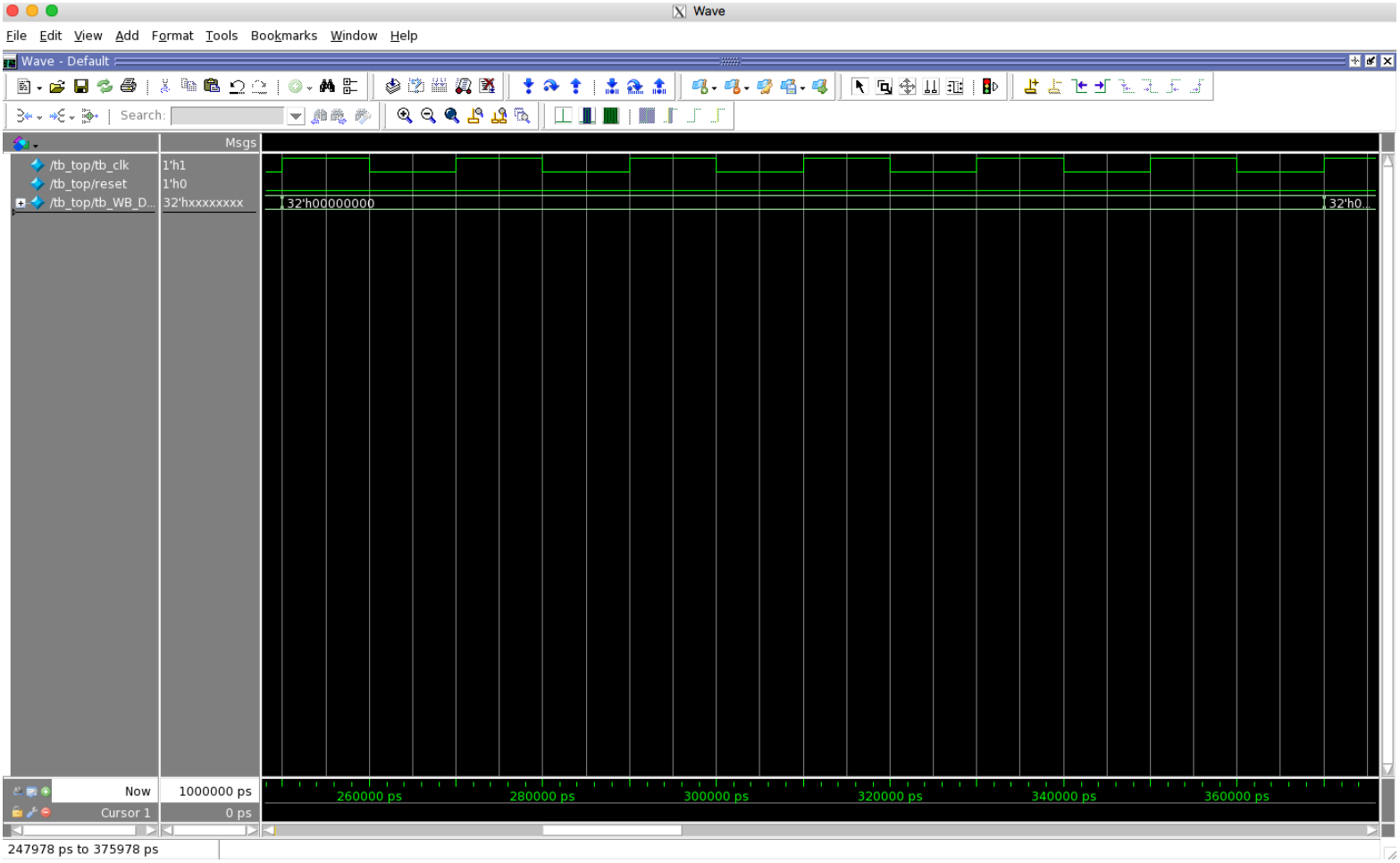
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000000
32'h0...
Now
1000000 ps
Cursor 1
0 ps
260000 ps
280000 ps
300000 ps
320000 ps
340000 ps
360000 ps
247978 ps to 375978 ps

Figure 4:

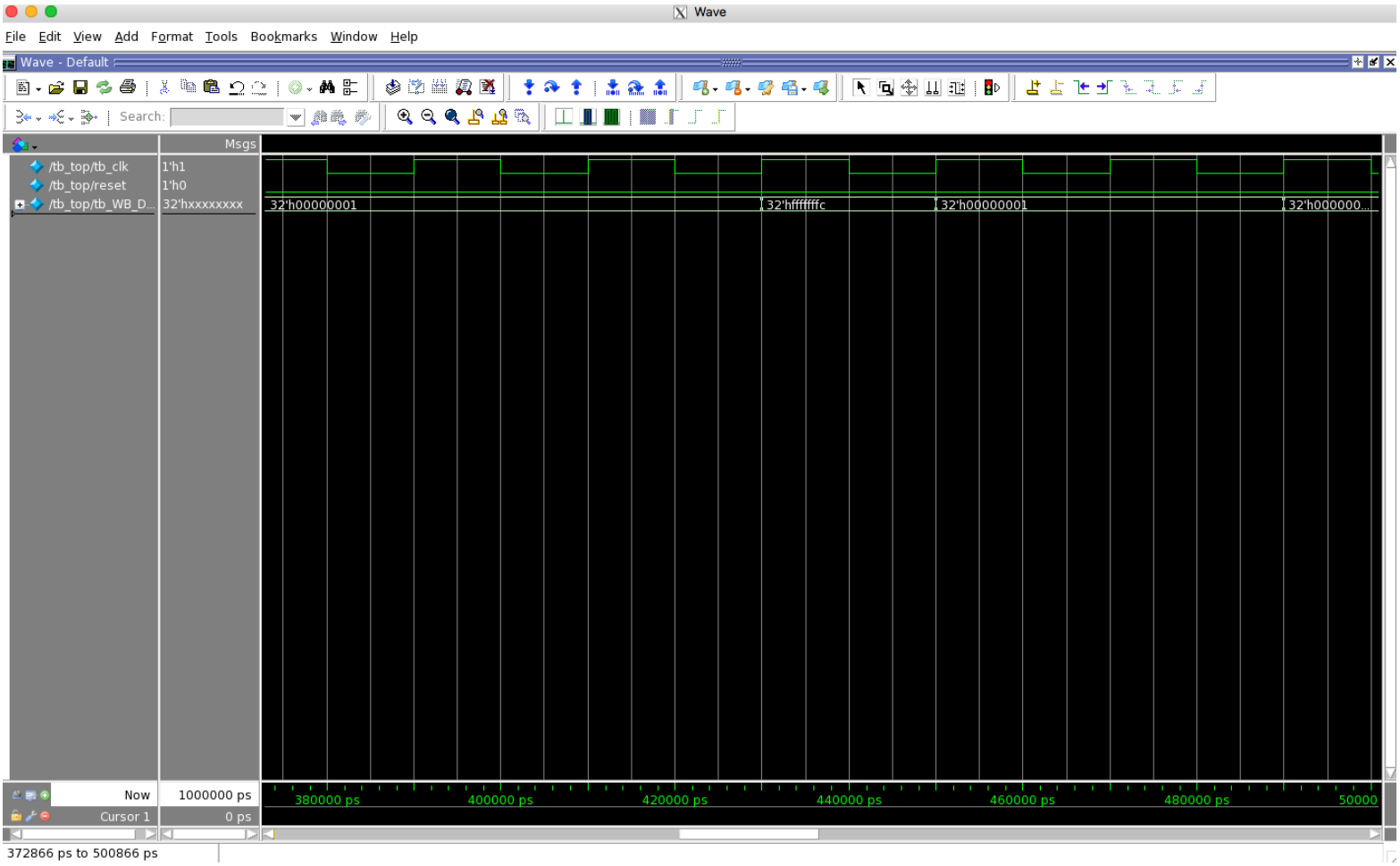
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000001
32'hfffffffc
32'h00000001
32'h000000...
Now
1000000 ps
Cursor 1
0 ps
380000 ps
400000 ps
420000 ps
440000 ps
460000 ps
480000 ps
50000
372866 ps to 500866 ps

Figure 5:

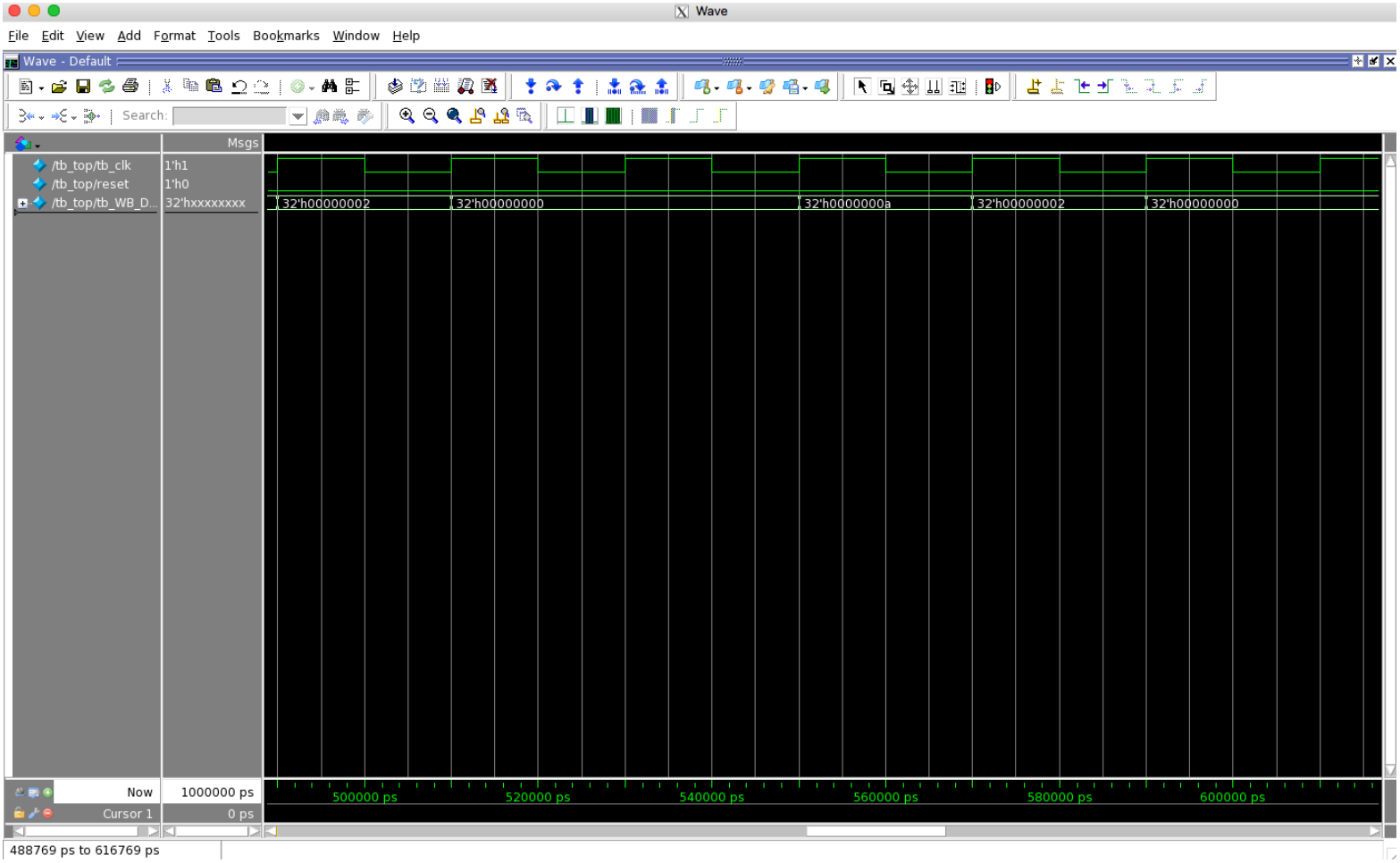
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000002
32'h00000000
32'h0000000a
32'h00000002
32'h00000000
Now
1000000 ps
Cursor 1
0 ps
500000 ps
520000 ps
540000 ps
560000 ps
580000 ps
600000 ps
488769 ps to 616769 ps

Figure 6:

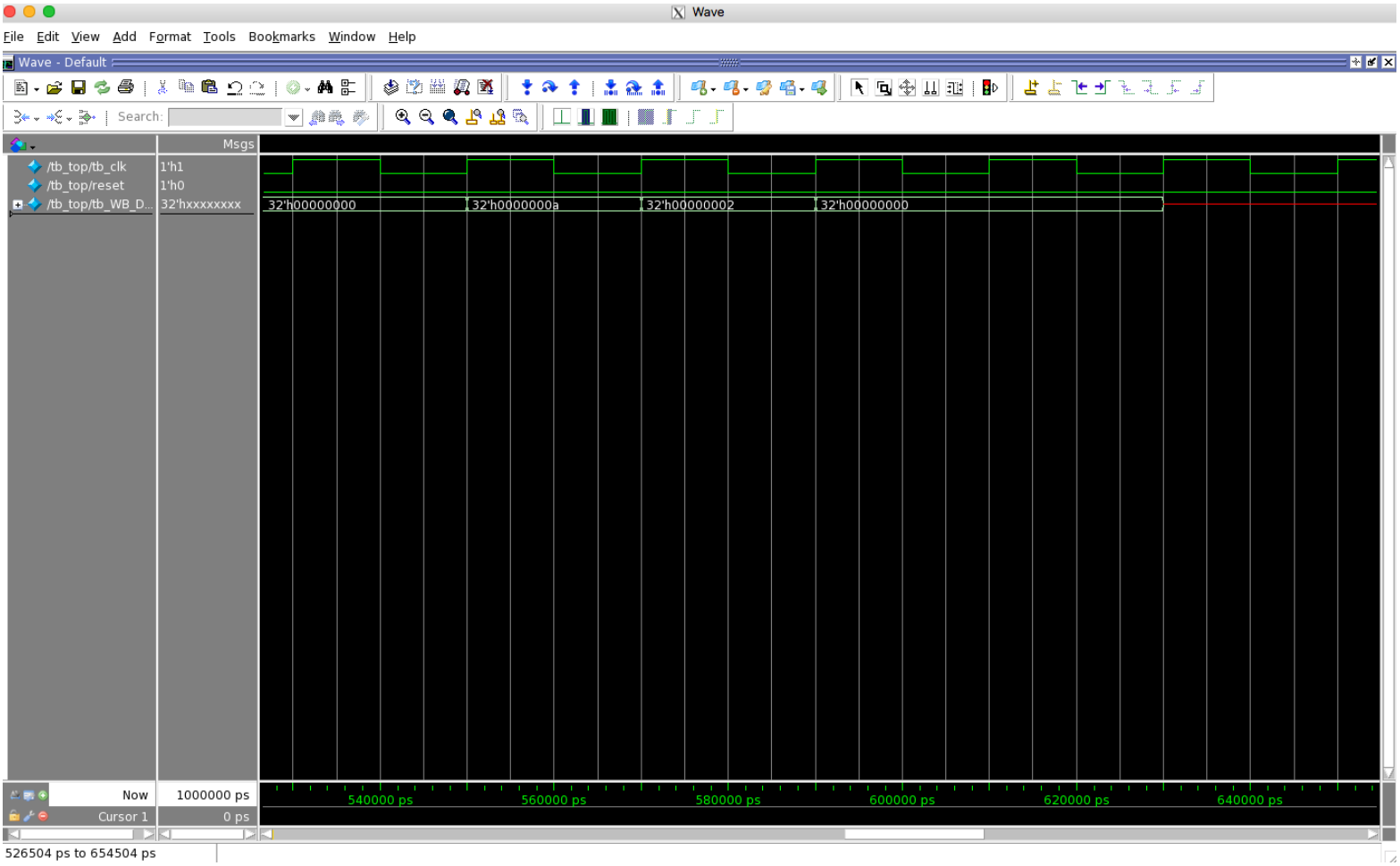
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_top/tb_clk
/tb_top/reset
/tb_top/tb_WB_D...
1'h1
1'h0
32'hxxxxxxxx
32'h00000000
32'h0000000a
32'h00000002
32'h00000000
Now
1000000 ps
Cursor 1
0 ps
540000 ps
560000 ps
580000 ps
600000 ps
620000 ps
640000 ps
526504 ps to 654504 ps

Figure 7: