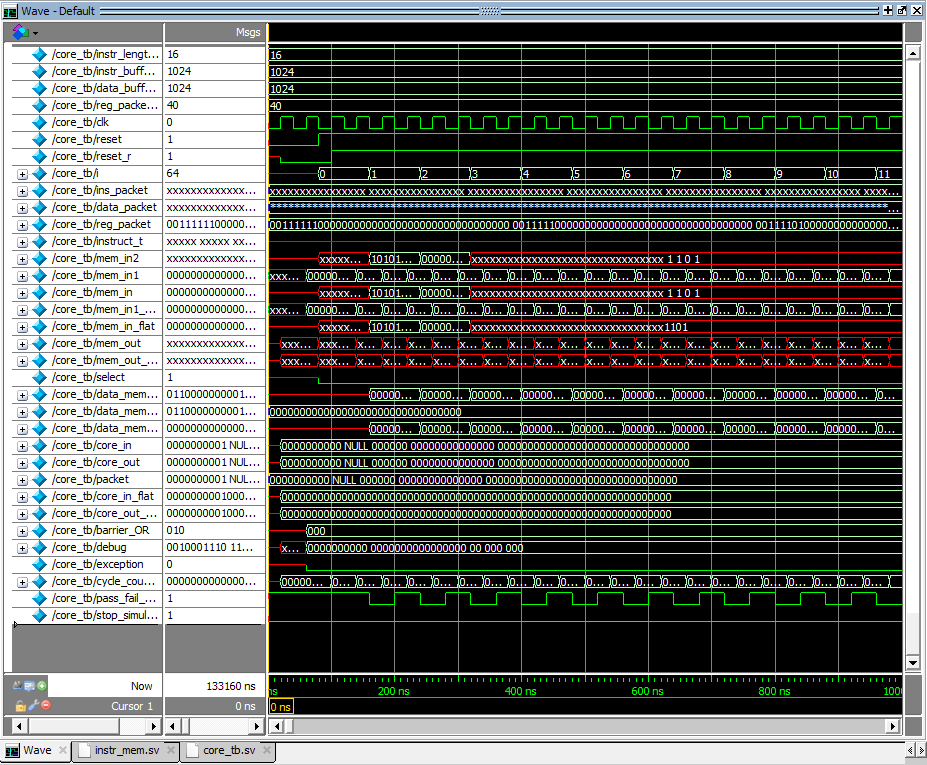
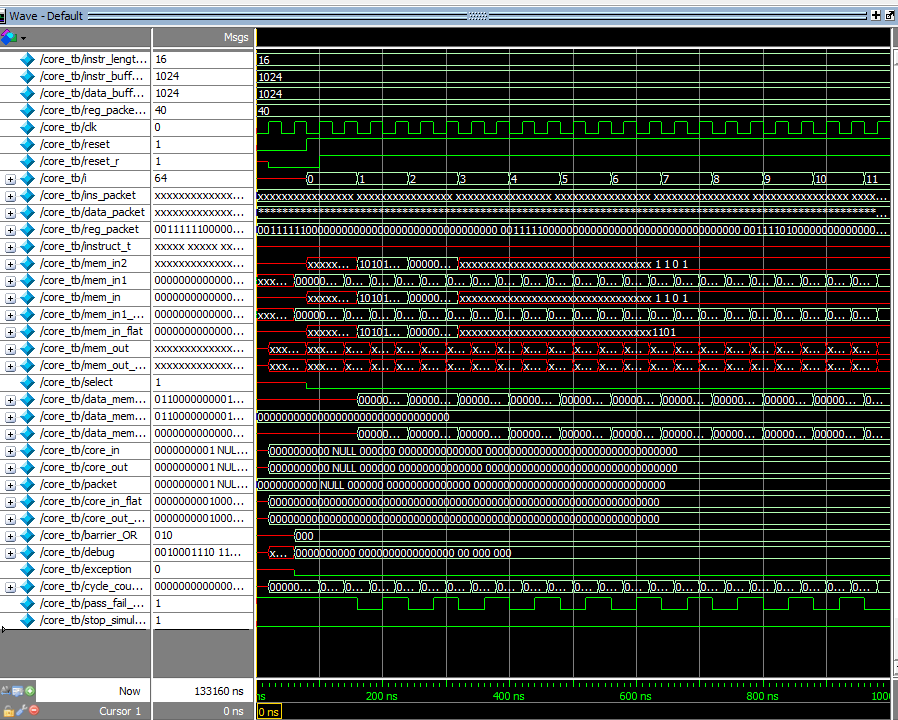
Q1)  
a) The opcodes consist of 5 bits, which can support up to a maximum of 32 instructions. According to the [instructor’s response](https://piazza.com/class/i7m4fk3czkt660?cid=274) on the Piazza, we are to only consider the opcode field. There are 22 instructions that use only this field, so the instruction set has room for 10 more instructions.  
  
b) instruction added to alu.sv on line 30, and instruction added to definitions.sv on line 46.

Q2)  
a) My Lab1 register file requires two parameters – the number of registers and their width. The Lab2 provided register file accepts one parameter – the register width (number of registers is hard coded at 31:0 which is 32 registers). The Lab2 register file also has one of the variables (rd\_addr\_i) take on a dual role for accessing registers in the array for both reading and writing data. Reading and writing in my Lab1 register file has their own variable (two). The provided register file also determines the size of the registers via exponentiation from a base of 2 from the only parameter.  
  
b) Given the dissimilarities between my register file and the Lab2 provided one, my register file had to be modified to be compatible with the Lab2 code. [Permission granted here.](https://piazza.com/class/i7m4fk3czkt660?cid=244)  
  
  
  
  
# --------VANILLA HAS BOOTED---------

# PASS: 0x00000001 1 (CYCLE 0x0000000b 11)

# PASS: 0x00000002 2 (CYCLE 0x00000012 18)

# PASS: 0x00000003 3 (CYCLE 0x00000019 25)

# PASS: 0x00000004 4 (CYCLE 0x00000020 32)

# PASS: 0x00000005 5 (CYCLE 0x00000026 38)

# PASS: 0x00000006 6 (CYCLE 0x00000034 52)

# PASS: 0x00000007 7 (CYCLE 0x0000003f 63)

# PASS: 0x00000008 8 (CYCLE 0x0000004f 79)

# PASS: 0x00000009 9 (CYCLE 0x00000056 86)

# PASS: 0x0000000a 10 (CYCLE 0x0000005c 92)

# PASS: 0x0000000b 11 (CYCLE 0x00000062 98)

# PASS: 0x0000000c 12 (CYCLE 0x0000006c 108)

# PASS: 0x0000000d 13 (CYCLE 0x00000077 119)

# PASS: 0x0000000e 14 (CYCLE 0x00000082 130)

# PASS: 0x0000000f 15 (CYCLE 0x0000008d 141)

# PASS: 0x00000010 16 (CYCLE 0x00000098 152)

# PASS: 0x00000011 17 (CYCLE 0x000000a2 162)

# PASS: 0x00000012 18 (CYCLE 0x000000ac 172)

# PASS: 0x00000013 19 (CYCLE 0x000000b6 182)

# {000000b9} barrier OR changed from 0 to 2

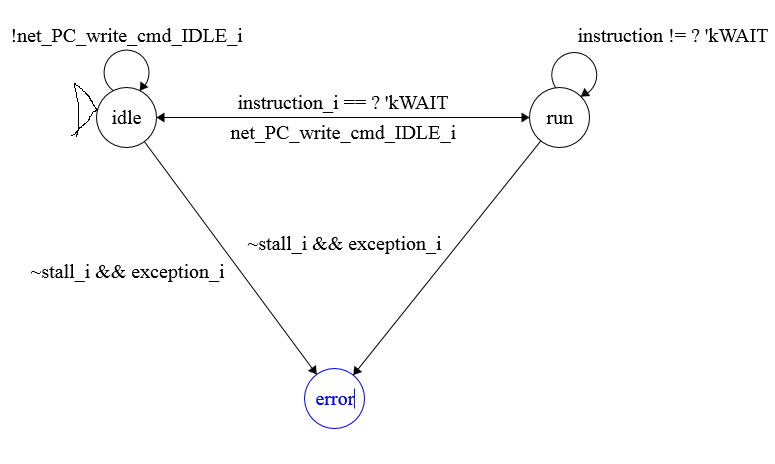
# {000000ba} barrier OR changed from 0 to 2

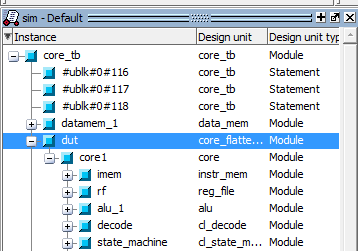
# DONE: 0x00000013 19 (CYCLE 0x000000ba 186)

# Break in Module core\_tb at C:/Users/Chris/Desktop/lab2Part1/core\_tb.sv line 258

Q3)   
a) Keeping the memory separate will prevent the possibility of one section overflowing into the other. Also, since they are separate, they can both be accessed in parallel. This addresses the issue of the memory being accessed for a data procedure while an instruction procedure is waiting for it to finish.   
  
b) Maximum size of an assembly program is 2^(address width).  
  
c) yes

Q4)  
a) *this protocol is flexible and a single request/reply pair could happen in a* ***single cycle****, or across many cycles.* It would appear that best case would be one clock cycle.

Q5)   
a)  
  
  
b) *Stall if LD/ST still active; or in non-RUN state.*  
Presumably, LD is a load operation, and ST is a store operation. If not running (idle or error), then stall.

c)   
  
  
  
d) step size for 141Core is 1, each unit being 16 bits. [source](https://piazza.com/class/i7m4fk3czkt660?cid=239)  
  
e) added them

Q6)   
a) added rotate instruction to kernel.java, recompiled VanillaAssembler  
  
b) added unit test to tester.asm, generated new hex files  
  
c) With BLR (bitwise left-rotate) testcase added to tester.asm, alu.sv, cl\_decode.sv, definitions.sv, and kernel.java.  
  
# --------VANILLA HAS BOOTED---------

# PASS: 0x00000001 1 (CYCLE 0x0000000b 11)

# PASS: 0x00000002 2 (CYCLE 0x00000012 18)

# PASS: 0x00000003 3 (CYCLE 0x00000019 25)

# PASS: 0x00000004 4 (CYCLE 0x00000020 32)

# PASS: 0x00000005 5 (CYCLE 0x00000026 38)

# PASS: 0x00000006 6 (CYCLE 0x00000034 52)

# PASS: 0x00000007 7 (CYCLE 0x0000003f 63)

# PASS: 0x00000008 8 (CYCLE 0x0000004f 79)

# PASS: 0x00000009 9 (CYCLE 0x00000056 86)

# PASS: 0x0000000a 10 (CYCLE 0x0000005c 92)

# PASS: 0x0000000b 11 (CYCLE 0x00000062 98)

# PASS: 0x0000000c 12 (CYCLE 0x0000006c 108)

# PASS: 0x0000000d 13 (CYCLE 0x00000077 119)

# PASS: 0x0000000e 14 (CYCLE 0x00000082 130)

# PASS: 0x0000000f 15 (CYCLE 0x0000008d 141)

# PASS: 0x00000010 16 (CYCLE 0x00000098 152)

# PASS: 0x00000011 17 (CYCLE 0x000000a2 162)

# PASS: 0x00000012 18 (CYCLE 0x000000ac 172)

# PASS: 0x00000013 19 (CYCLE 0x000000b6 182)

# PASS: 0x00000014 20 (CYCLE 0x000000c1 193)

# {000000c4} barrier OR changed from 0 to 2

# {000000c5} barrier OR changed from 0 to 2

# DONE: 0x00000014 20 (CYCLE 0x000000c5 197)

# Break in Module core\_tb at C:/Users/Chris/Desktop/lab2Part1/ModelSim/core\_tb.sv line 258  
  
d) [source](https://piazza.com/class/i7m4fk3czkt660?cid=235)  
*before adding rotate instruction*  
cycle time = 1/fMax = **1/54.65MHz = 0.018298**slack = **-17.297**  
flipflops used: **2086**  
total logic elements: **4823***after adding rotate instruction*cycle time = 1/fMax = **1/54.1MHz = 0.018484**slack = **-17.483**  
flipflops used: **2086**  
total logic elements: **4901**

**Instructions added**: I noticed within the miner.asm file it was making use of exclusive or and rotate right. The logic was not as efficient as it could have been. After looking on piazza, reimplementation of these two instructions in a more efficient manner fulfills the criteria of the assignment.

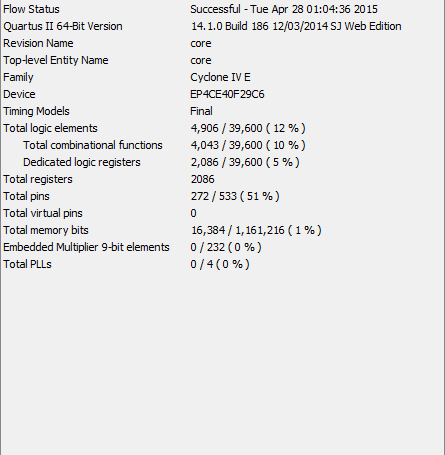
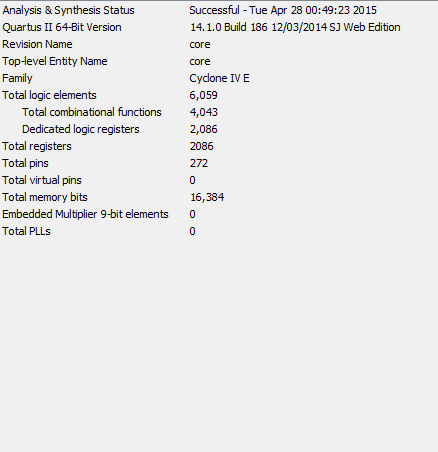
**Speedup**: 358752 / 160608 = 2.2337

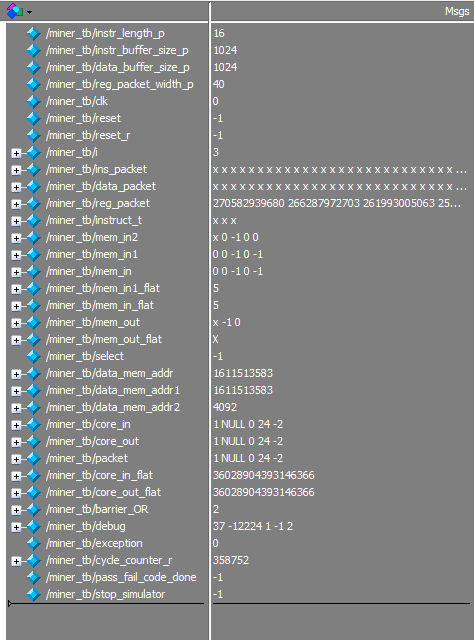
**Baseline Miner Cycles**: 358752

**Optimized Miner Cycles**: 160608

**Speedup in Cycles**: 358752 – 160608 = 198144

**Hashrate**: 9 / 160608 = 0.000056037

**Baseline Miner**  
Fmax = 53.42 MHz  
slack = -17.720, slack is .340 from hold summary  
  




**Optimized Miner**  
Fmax = 54.78 MHz  
slack = -17.225, slack is .339 from hold summary  
