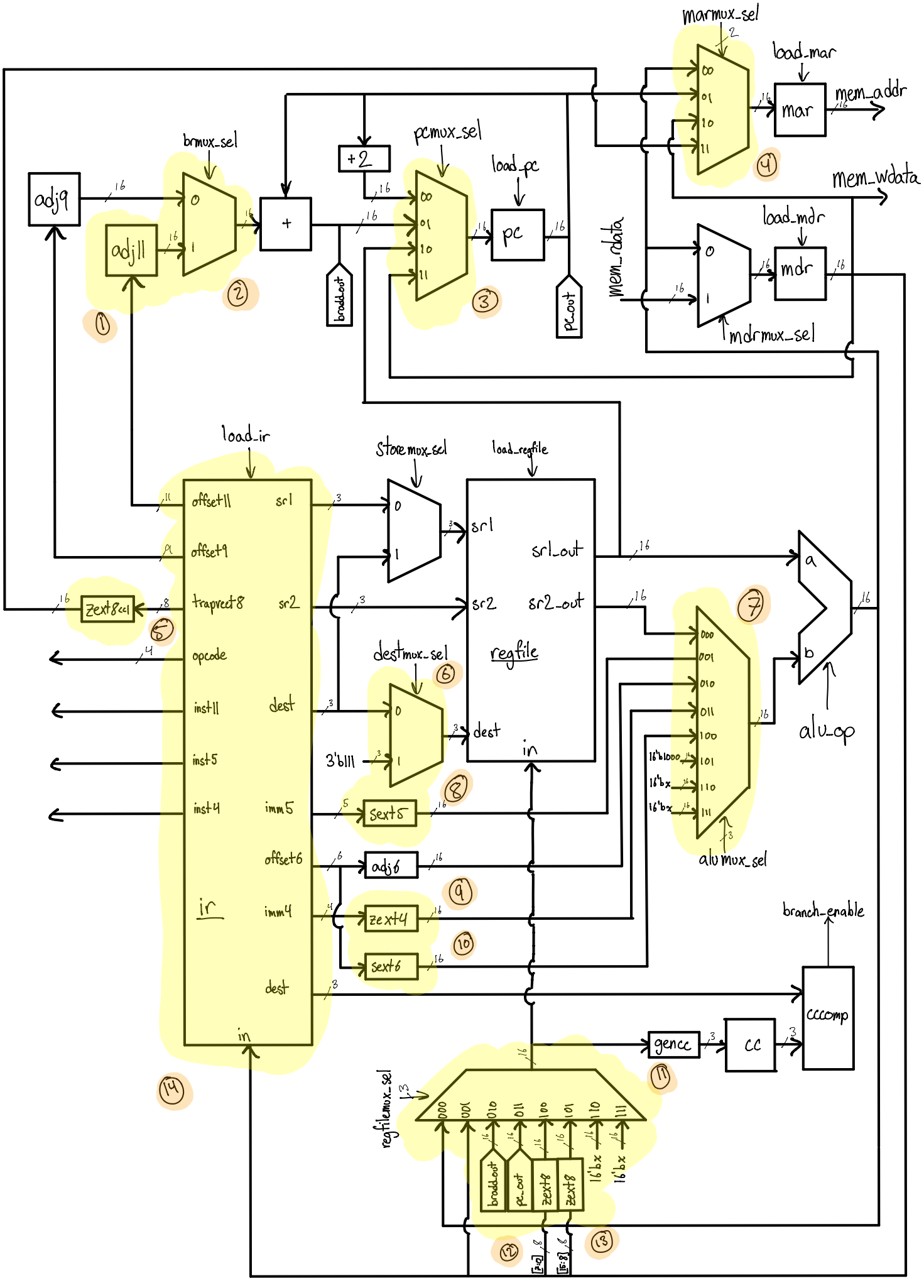
ECE 411 MP1 Report

Rauhul Varma
rvarma2 — 2/4/18

1. Datapath diagram
2. Descriptions of highlighted blocks
3. Testing methodology
4. Wave traces
5. List of memory writes
6. Timing analysis report


marmux_sel
load_mar
mar
mem_addr
mem_wdata
load_mdr
mdr
mdrmux_sel
mem_rdata
brmux_sel
adj9
adj11
+
+2
pcmux_sel
load_pc
pc
braddout
pc_out
load_ir
offset11
offset9
trapvect8
opcode
inst11
inst5
inst4
sr1
sr2
dest
imm5
offset6
imm4
ir
in
zext8<<1
storemux_sel
destmux_sel
3'b111
sr1
sr2
dest
in
sr1_out
sr2_out
regfile
sext5
adj6
zext4
sext6
alumux_sel
alu_op
a
b
branch_enable
cccomp
gencc
cc
regfilemux_sel
braddout
pc_out
zext8
zext8
16'bx
16'bx
[7:0]
[15:8]

Descriptions of highlighted blocks

1. (new) **adj11** uses **ir.offset11** as its input, left shifts it and sign-extends the result to 16 bits. The output is used during the **JSR** instruction as the amount to increment the **pc** by.

2. (new) **brmux** allows the control logic to select which shifted and sign-extended offset to increment the **pc** by. It is used during the **BR**, **JSR**, and **LEA** instructions.

3. (modified) **pcmux** is extended to allow the **pc** to be loaded directly from **regfile.sr1_out** and **mdr_out**.
    1. **regfile.sr1_out** is used during the **JMP**, and **JSR** instructions so the **pc** can be directly loaded from the desired register.
    2. **mdr_out** is used during the **TRAP** instruction so the **pc** can be loaded directly from the result of the memory access to the address specified by **zext8<<1**.

4. (modified) marmux is extended to allow the **mar** to be loaded directly from **mdr_out** and **zext8<<1_out**.
    1. **mdr_out** is used during the **LDI**, and **STI** instructions so the **mar** can be loaded directly from **mdr** so the second memory access can be made.
    2. **zext8<<1_out** is used during the **TRAP** instruction so the **mar** can be loaded with the computed trapvector.

5. (new) **zext8<<1** uses **ir.trapvect8** as its input, left shifts it and zero-extends the result to 16 bits. The output is used during **TRAP** instruction as the new value for **mar**.

6. (new) **destmux** uses **ir.dest** and **3'b111** as its inputs. It allows the control logic to override which register in the **regfile** is loaded. It is used during the **JSR** and **TRAP** instructions, so **regfile.data.7** can be loaded with the **pc**.

7. (modified) **alumux** is extended to allow the **alu** to use **sext5_out**, **zext4_out**, **sext6_out**, and **16'b1000** as inputs to b. (Note: the unused inputs are tied to 16'bx.)
    1. **sext5_out** is used during the **ADDi** and **ANDi** instructions as the value to add/and **regfile.sr1_out** with.
    2. **zext4_out** is used during the **SHF** instruction as the amount to shift **regfile.sr1_out** by.
    3. **sext6_out** is used during the **LDI**, **LDB**, **LDR**, **STB**, **STI**, and **STR** instructions as the value to add to **regfile.sr1_out** to form the address to load/store data to/from memory.
    4. **16'b1000** is used during the **STB** instruction to left shift **regfile.sr1_out** by 8 bits to enable writing to only the high bits of an address.

8. (new) **sext5** uses **ir.imm5** as its input and sign-extends it to 16 bits. The output is used during the **ADDi** and **ANDi** instructions.

9. (new) **zext4** uses **ir.imm4** as its input and zero-extends it to 16 bits. The output is used during the **SHF** instruction.

10. (new) **sext6** uses ir.offset6 as its input and sign-extends it to 16 bits. The output is used during the **LDI**, **LDB**, **LDR**, **STB**, **STI**, and **STR** instructions.

11. (modified) **regfilemux** is extended to allow the **regfile** to be loaded directly from **bradd_out**, **pc_out**, **mdr_zext8_l_out**, and **mdr_zext8_h_out**. (Note: the unused inputs are tied to 16'bx.)
    1. **bradd_out** is used during the **LEA** instruction so the address of the desired label/offset can be loaded into the specified register, **regfile.dest**.
    2. **pc_out** is used during the **JSR** and **TRAP** instructions so **regfile.data.7** can be loaded with the current pc for a return at a later time as needed.
    3. **mdr_zext8_l_out**, **mdr_zext8_h_out**, are used during the LDB instruction so the upper/lower byte of the specified memory location can be loaded into the specified register, **regfile.dest**.

12. (new) **mdr_zext8_l** uses the low bits of **mdr_out**, [7:0], and zero-extends them to 16 bits. The output is used during the **LDB** instruction.

13. (new) **mdr_zext8_h** uses the high bits of **mdr_out**, [15:8], and zero-extends them to 16 bits. The output is used during the **LDB** instruction.

14. (modified) **ir** is extended to output **imm4**, **imm5**, **trapvect8** and **offset11** to various locations in the data path, the uses of these outputs in the ISA are explained above. It is also extended to output **inst4**, **inst5**, and **inst11** to the control logic.
    1. **inst4** is used during the **SHF** instruction to determine which type of shift to perform.
    2. i**nst5** is also used during the **SHF** instruction to determine which type of shift to perform. It is additionally used during the **ADD** and **AND** instructions to determine which type of add/and to perform (from a register or from an immediate value).
    3. **inst11** is used during the **JSR** instruction determine which source to load the **pc** from.

Testing methodology

I used a multistep process for testing and design. While implementing each instruction I had both the data path and control logic open side by side so I could update the control logic as the datapath changed. I also used git diff extensively before committing any code so I could review all the changes. This allowed me to catch many errors before even running my tests.

For each instruction I made a corresponding LC3 test case to ensure everything was working before moving on to the next instruction. I also tried to make the test cases fairly wide in order to increase code coverage. For example my **SHF** test case is as follows:

```
ORIGIN 4x0000
SEGMENT CodeSegment:
    LDR   R1, R0, VAL1
    LSHF  R2, R1, 3
    RSHFL R3, R1, 3
    RSHFA R4, R1, 3
    LDR   R1, R0, VAL2
    LSHF  R2, R1, 3
    RSHFL R3, R1, 3
    RSHFA R4, R1, 3

HALT:                   ; Infinite loop to keep the processor
    BRnzp HALT          ; from trying to execute the data below.

VAL1:
    DATA2 4xAAAA
VAL2:
    DATA2 4x5555
```

In this test case I wanted to ensure that all forms of the shift worked: left, right logical, and right arithmetic. However I also wanted to ensure that regardless of the leading bit/value being shifted the operations occurred correctly and the cc bits were set correctly. This prompted me to perform 2 sets of shifts one with 4xAAAA and 4x5555. I followed every exercised path for each shift of the 6 shifts ensuring each bus had the correct value and all registered loaded the expected values.

I used the same methodology when testing more complicated instructions like **LDB** and **STB**. For these instructions I made extra care to carefully diagram what the datapath/state machine looked like before implementing any of the design in verilog. Again I used git to ensure I made all the changes I specified in my diagrams and didn't make unintended changes. I then used these diagrams once again to trace every value during my test case for these instructions. The test case is as follows:

```
ORIGIN 4x0000
SEGMENT CodeSegment:
    LEA R0, L_DATA
    LDB R1, R0, 0
    LDB R2, R0, 1
    NOT R1, R1
    NOT R2, R2
    STB R1, R0, 0
    STB R2, R0, 1
```

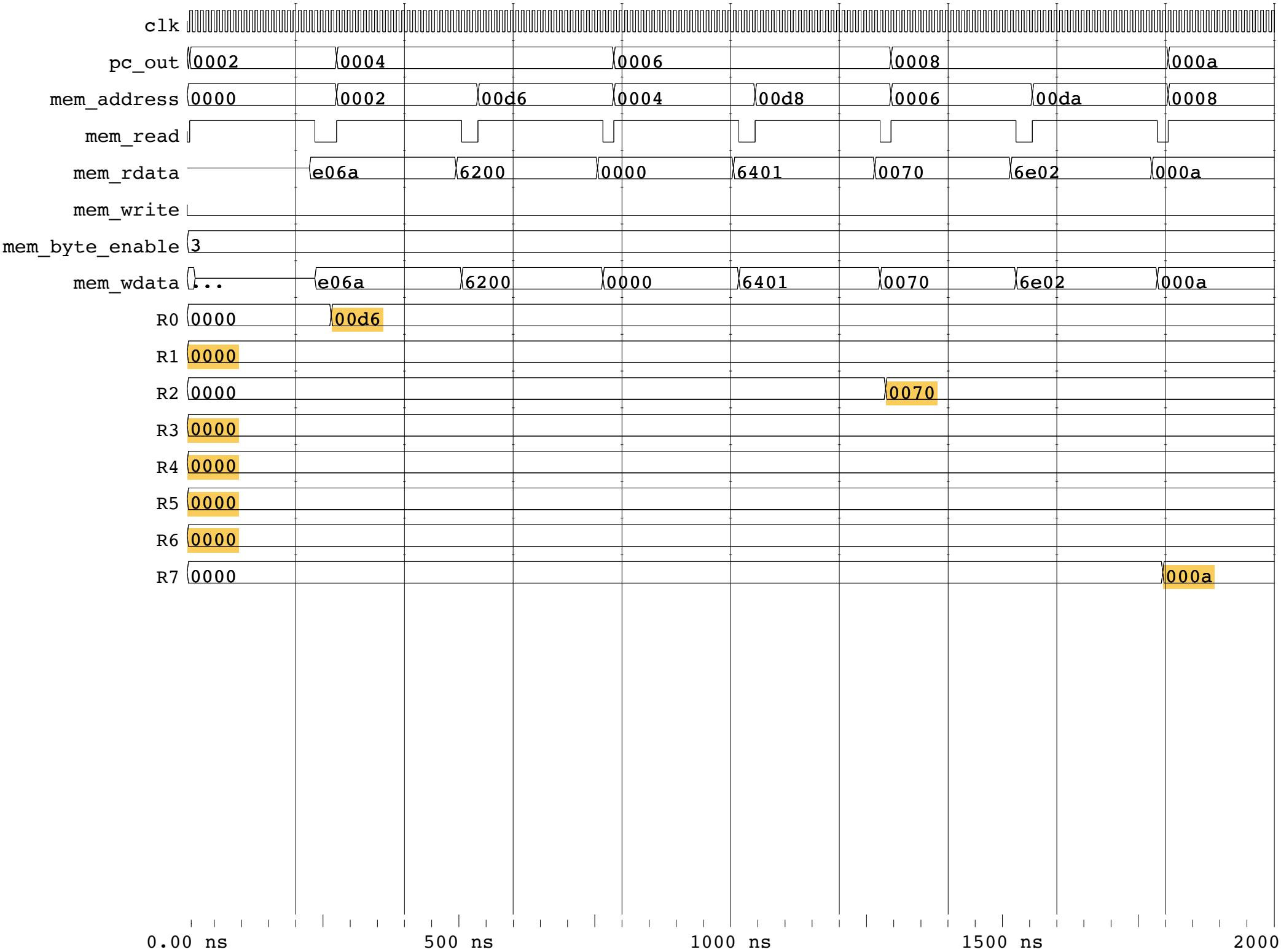
```
HALT:             ; Infinite loop to keep the processor
    BRnzp HALT    ; from trying to execute the data below.

L_DATA:
    DATA2 4xAA55
```

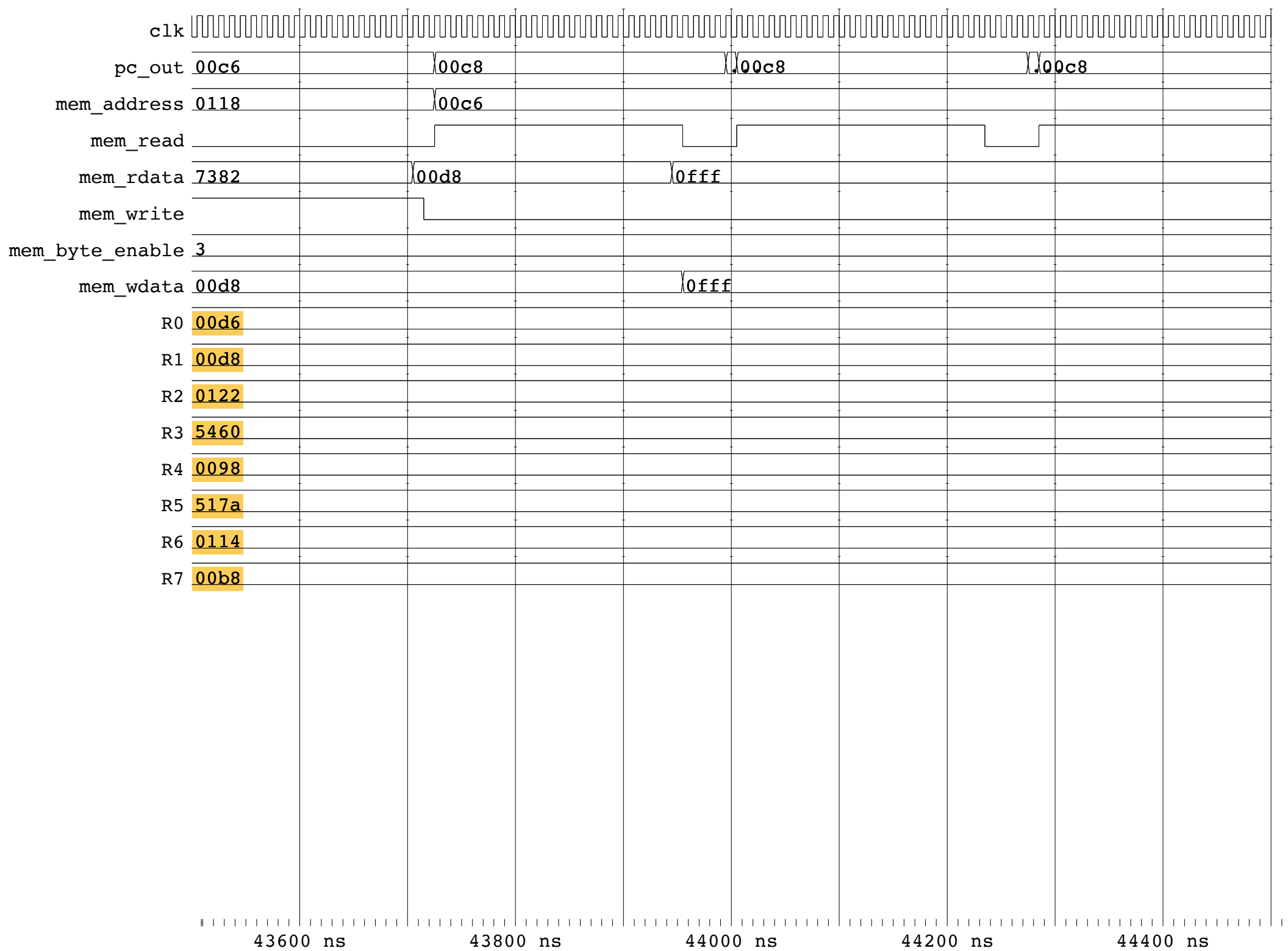
This mythology allowed to me near immediately solve a bug I encountered when first performing my the test case. I had accidentally specified a right shift during STB (high) which was caused 4x00XX to be written to memory. Since I had written down exactly what I expected to happen at each step, it was easy to find the mistake.

Additionally, after completion an instruction and passing it's test case. I reran every single test case I previously wrote to ensure no instructions had broken due to my additions.

This set of tests and preparation made me confident the final mp1 test would work. I also went through the additional step of comparing my processor's state to the simulator's at random points, checking that all the registered matched their expected values.



Entity:mp1_tb Architecture: Date: Sat Feb 03 13:39:05 CST 2018 Row: 1 Page: 1



Entity:mp1_tb Architecture: Date: Sat Feb 03 13:37:49 CST 2018 Row: 1 Page: 1

List of memory writes

ps	delta	mem_address	mem_write	mem_byte_enable	mem_wdata
0	+0	xxxx	x	x	xxxx
0	+1	0000	0	3	0000
3155000	+5	00f4	1	3	01e0
3385000	+4	00f4	0	3	01e0
5225000	+5	00f6	1	3	000a
5455000	+4	00f6	0	3	000a
6515000	+5	00f8	1	3	4537
6745000	+4	00f8	0	3	4537
10205000	+5	00fa	1	3	a342
10435000	+4	00fa	0	3	a342
10715000	+5	00fc	1	3	01a1
10945000	+4	00fc	0	3	01a1
11225000	+5	00fe	1	3	fe62
11455000	+4	00fe	0	3	fe62
12005000	+5	0102	1	3	0100
12235000	+4	0102	0	3	0100
13025000	+5	0100	1	3	fe6f
13255000	+4	0100	0	3	fe6f
21365000	+5	0102	1	3	001a
21595000	+4	0102	0	3	001a
23195000	+5	0104	1	3	ff9b
23425000	+4	0104	0	3	ff9b
24515000	+5	0106	1	3	000d
24745000	+4	0106	0	3	000d
27095000	+5	0108	1	3	517a
27325000	+4	0108	0	3	517a
29435000	+5	010a	1	3	0133
29665000	+4	010a	0	3	0133
30755000	+5	010d	1	2	9000
30985000	+4	010d	0	3	9000
31265000	+5	010c	1	1	00ac
31495000	+4	010c	0	3	00ac
32285000	+5	010a	1	3	90ac
32515000	+4	010a	0	3	90ac
33065000	+5	0110	1	3	00f2
33295000	+4	0110	0	3	00f2
34325000	+5	010e	1	3	0646
34555000	+4	010e	0	3	0646
36155000	+5	00a2	1	3	126c
36385000	+4	00a2	0	3	126c
36935000	+5	0112	1	3	000c
37165000	+4	0112	0	3	000c
38495000	+5	0114	1	3	ae85
38725000	+4	0114	0	3	ae85
41885000	+5	0116	1	3	600d
42115000	+4	0116	0	3	600d
43485000	+5	0118	1	3	00d8
43715000	+4	0118	0	3	00d8

Timing analysis report

```
TimeQuest Timing Analyzer report for mp1
Sat Feb  3 14:04:50 2018
Quartus II 32-bit Version 13.1.4 Build 182 03/12/2014 SJ Full Version

  2. TimeQuest Timing Analyzer Summary
  5. Clocks
  6. Slow 1100mV 85C Model Fmax Summary
 53. TimeQuest Timing Analyzer Messages

+------------------------------------------------------------------------+
; TimeQuest Timing Analyzer Summary                                      ;
+--------------------+---------------------------------------------------+
; Quartus II Version ; Version 13.1.4 Build 182 03/12/2014 SJ Full Version ;
; Revision Name      ; mp1                                               ;
; Device Family      ; Stratix III                                       ;
; Device Name        ; EP3SE50F780C2                                     ;
; Timing Models      ; Final                                             ;
; Delay Model        ; Combined                                          ;
; Rise/Fall Delays   ; Enabled                                           ;
+--------------------+---------------------------------------------------+

+------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------+
; Clocks
;
+------------+------+--------+-----------+-------+-------+------------
+-----------+-------------+-------+--------+-----------+------------
+----------+--------+--------+---------+
; Clock Name ; Type ; Period ; Frequency ; Rise  ; Fall  ; Duty Cycle ;
Divide by ; Multiply by ; Phase ; Offset ; Edge List ; Edge Shift ;
Inverted ; Master ; Source ; Targets ;
+------------+------+--------+-----------+-------+-------+------------
+-----------+-------------+-------+--------+-----------+------------
+----------+--------+--------+---------+
; clk        ; Base ; 10.000 ; 100.0 MHz ; 0.000 ; 5.000 ;            ;
;            ;      ;        ;           ;           ;          ;        ;
; { clk } ;
+------------+------+--------+-----------+-------+-------+------------
+-----------+-------------+-------+--------+-----------+------------
+----------+--------+--------+---------+

+-------------------------------------------------+
; Slow 1100mV 85C Model Fmax Summary              ;
+------------+-----------------+------------+------+
; Fmax       ; Restricted Fmax ; Clock Name ; Note ;
+------------+-----------------+------------+------+
; 113.68 MHz ; 113.68 MHz      ; clk        ;      ;
+------------+-----------------+------------+------+

+-----------------------------------+
```

```
; TimeQuest Timing Analyzer Messages ;
+-----------------------------------+
Info: *******************************************************************
Info: Running Quartus II 32-bit TimeQuest Timing Analyzer
    Info: Version 13.1.4 Build 182 03/12/2014 SJ Full Version
    Info: Processing started: Sat Feb  3 14:04:41 2018
Info: Command: quartus_sta mp1 -c mp1
Info: qsta_default_script.tcl version: #1
Info (11104): Parallel Compilation has detected 24 hyper-threaded processors.
However, the extra hyper-threaded processors will not be used by default.
Parallel Compilation will use 12 of the 12 physical processors detected
instead.
Info (21077): Core supply voltage is 1.1V
Info (21077): Low junction temperature is 0 degrees C
Info (21077): High junction temperature is 85 degrees C
Info (332104): Reading SDC File: 'mp1.out.sdc'
Info: Found TIMEQUEST_REPORT_SCRIPT_INCLUDE_DEFAULT_ANALYSIS = ON
Info: Analyzing Slow 1100mV 85C Model
Info (332146): Worst-case setup slack is 1.203
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     1.203               0.000 clk
Info (332146): Worst-case hold slack is 0.294
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.294               0.000 clk
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.377
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.377               0.000 clk
Info: Analyzing Slow 1100mV 0C Model
Info (334003): Started post-fitting delay annotation
Info (334004): Delay annotation completed successfully
Info (332146): Worst-case setup slack is 1.858
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     1.858               0.000 clk
Info (332146): Worst-case hold slack is 0.272
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.272               0.000 clk
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.374
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.374               0.000 clk
Info: Analyzing Fast 1100mV 0C Model
Info (332146): Worst-case setup slack is 4.242
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.242               0.000 clk
```

```
Info (332146): Worst-case hold slack is 0.179
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.179               0.000 clk
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.654
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.654               0.000 clk
Info (332101): Design is fully constrained for setup requirements
Info (332101): Design is fully constrained for hold requirements
Info: Quartus II 32-bit TimeQuest Timing Analyzer was successful. 0 errors, 0
warnings
    Info: Peak virtual memory: 582 megabytes
    Info: Processing ended: Sat Feb  3 14:04:50 2018
    Info: Elapsed time: 00:00:09
    Info: Total CPU time (on all processors): 00:00:06
```