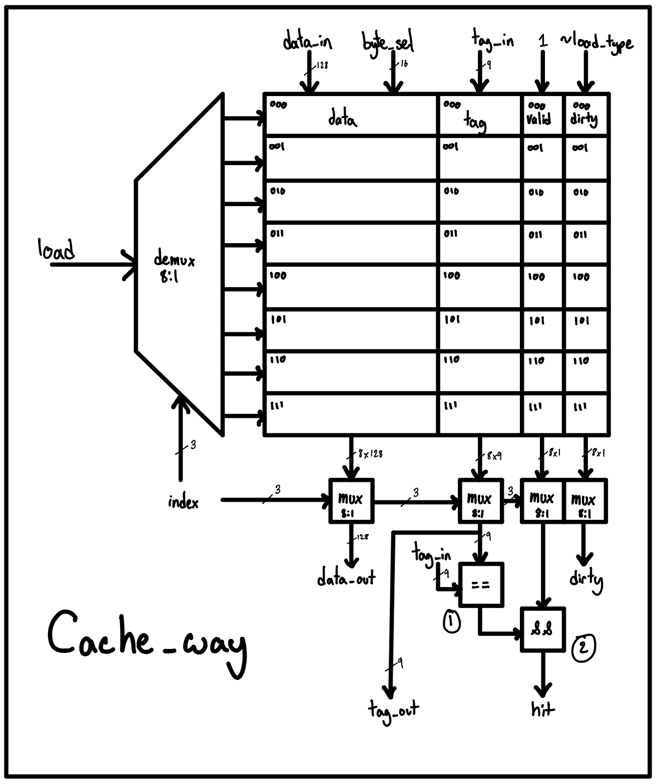
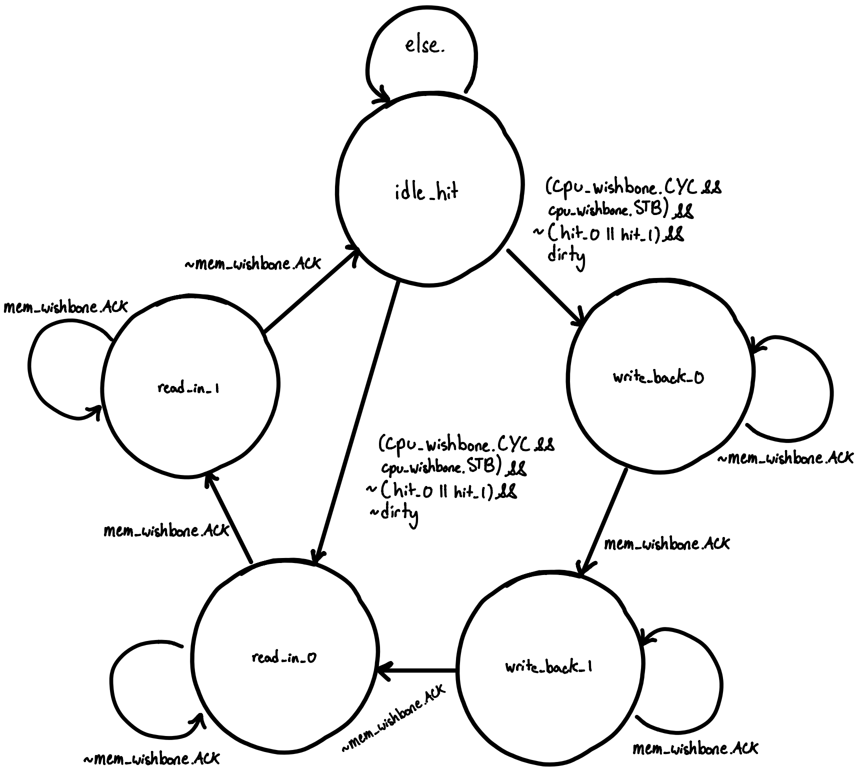
ECE 411 MP2 Report

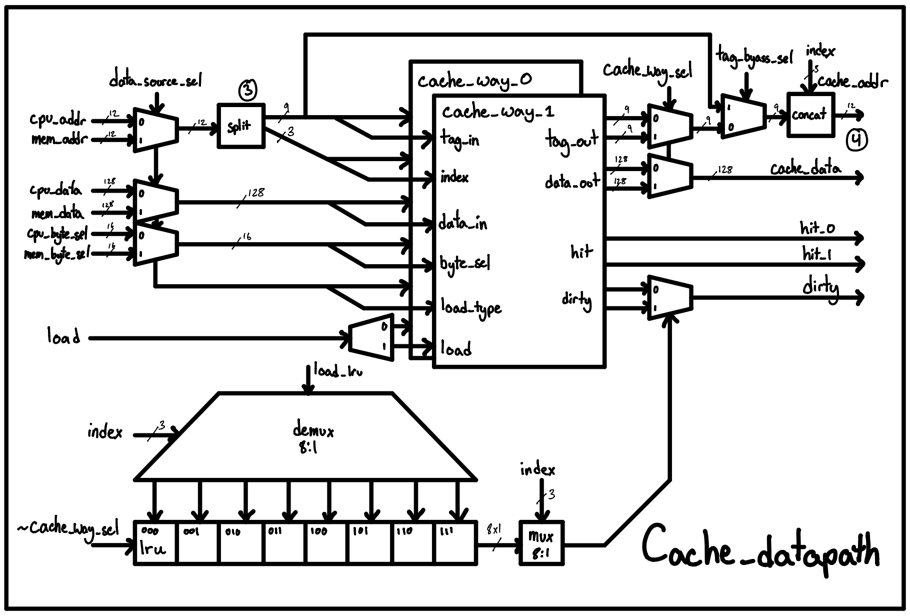
Rauhul Varma
rvarma2 — 2/26/18



data_in
byte_sel
tag_in
1
~load_type
data
tag
valid
dirty
load
demux 8:1
index
mux 8:1
data_out
tag_out
dirty
hit
==
&&
1
2
Cache_way



else.
idle_hit
(cpu_wishbone.CYC && cpu_wishbone.STB) && ~(hit_0 || hit_1) && dirty
write_back_0
~mem_wishbone.ACK
mem_wishbone.ACK
write_back_1
mem_wishbone.ACK
~mem_wishbone.ACK
read_in_0
~mem_wishbone.ACK
mem_wishbone.ACK
read_in_1
mem_wishbone.ACK
~mem_wishbone.ACK
(cpu_wishbone.CYC && cpu_wishbone.STB) && ~(hit_0 || hit_1) && ~dirty



data_source_sel
cpu_addr
mem_addr
split
3
cpu_data
mem_data
cpu_byte_sel
mem_byte_sel
load
cache_way_0
cache_way_1
tag_in
index
data_in
byte_sel
load_type
load
tag_out
data_out
hit
dirty
cache_way_sel
tag_bypass_sel
index
cache_addr
concat
4
cache_data
hit_0
hit_1
dirty
load_lru
demux 8:1
index
~cache_way_sel
lru
mux 8:1
Cache_datapath

Descriptions of Blocks

1. The **==** block takes in the currently selected tag (tag_arr[index]) and the input tag and outputs a **1** if they are equal and **0** if not.
2. The **&&** block takes in the result of the **==** block and the currently selected valid bit (valid_arr[index]) and outputs **1** if they are both 1, **0** if not. The combination of these blocks is used to define the **hit** signal, ie. if the input tag is equal to the current tag and the data is valid then there is a hit.
3. The **spilt** block take the input address and splits it into two signals, a tag and index. The **lower 3** bits are used to create the **index** bus and the **upper 9** are used to create the **tag** bus.
4. The **concat** block takes in an input tag from the desired source and the current index combines them to create an address. The address's **upper 9** bits are formed from the input **tag** and the **lower 3** bits are formed from the **index**.

All of the other blocks in the cache datapath are registers/register arrays, muxes and demuxes.

Testing Methodology

I followed a similar multistep process for mp2 as I described for mp1. I setup my development environment so that I could catch many bugs before even going into testing. For example, whenever I made changes to the cache datapath, I would ensure the relevant control logic was updated immediately. On top of that I used git diff constantly to review all changes I made.

I first updated my cpu to use the new wishbone interface and used all the tests from mp1 and earlier as regression tests. I checked that all tests completed with the correct and expected register values. I then chose a couple random points in each simulation to compare the waves against the cpu datapath before using the wishbone interface. I checked that at all of these points both implementations behaved the same way. The list of tests used is as follows:

```
jsrTest.asm
ldbstbTest.asm
ldistiTest.asm
mp1cp1.asm
mp1final.asm
trapTest.asm
```

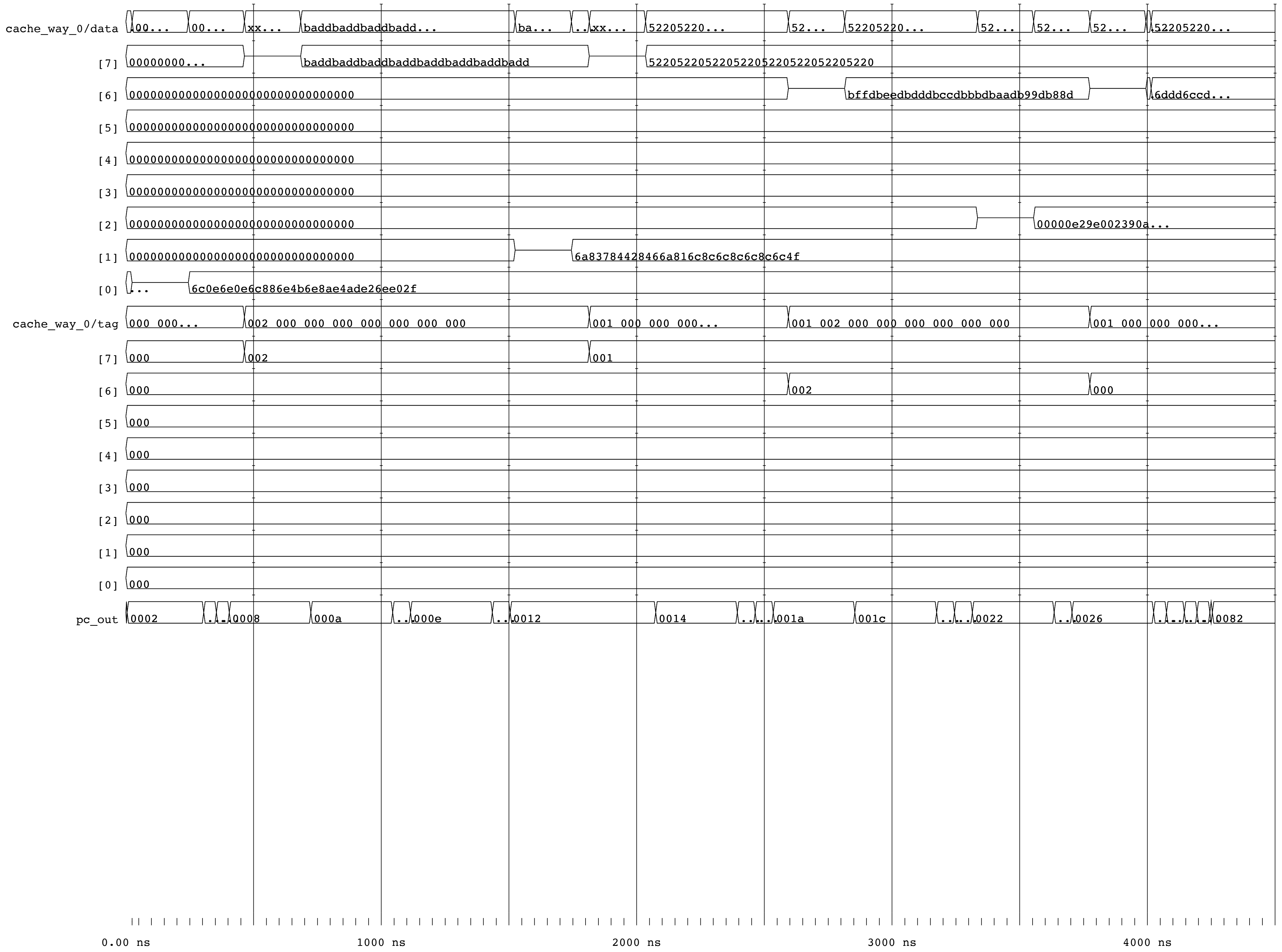
I then fully implemented the cache on paper and traced out by hand exactly what I expected to happen on every memory access, (hit, miss dirty, miss clean). With my design completed entirely on paper, I implemented the cache in verilog and as mentioned used git to ensure I only made the changes I intended to make.

I used the same methodology as I did in mp1 when testing the cache. For these tests, I used these paper diagrams once again to trace every relevant during my test case. One of the simpler test cases I used was my `ldbstbTest.asm` which is as follows:

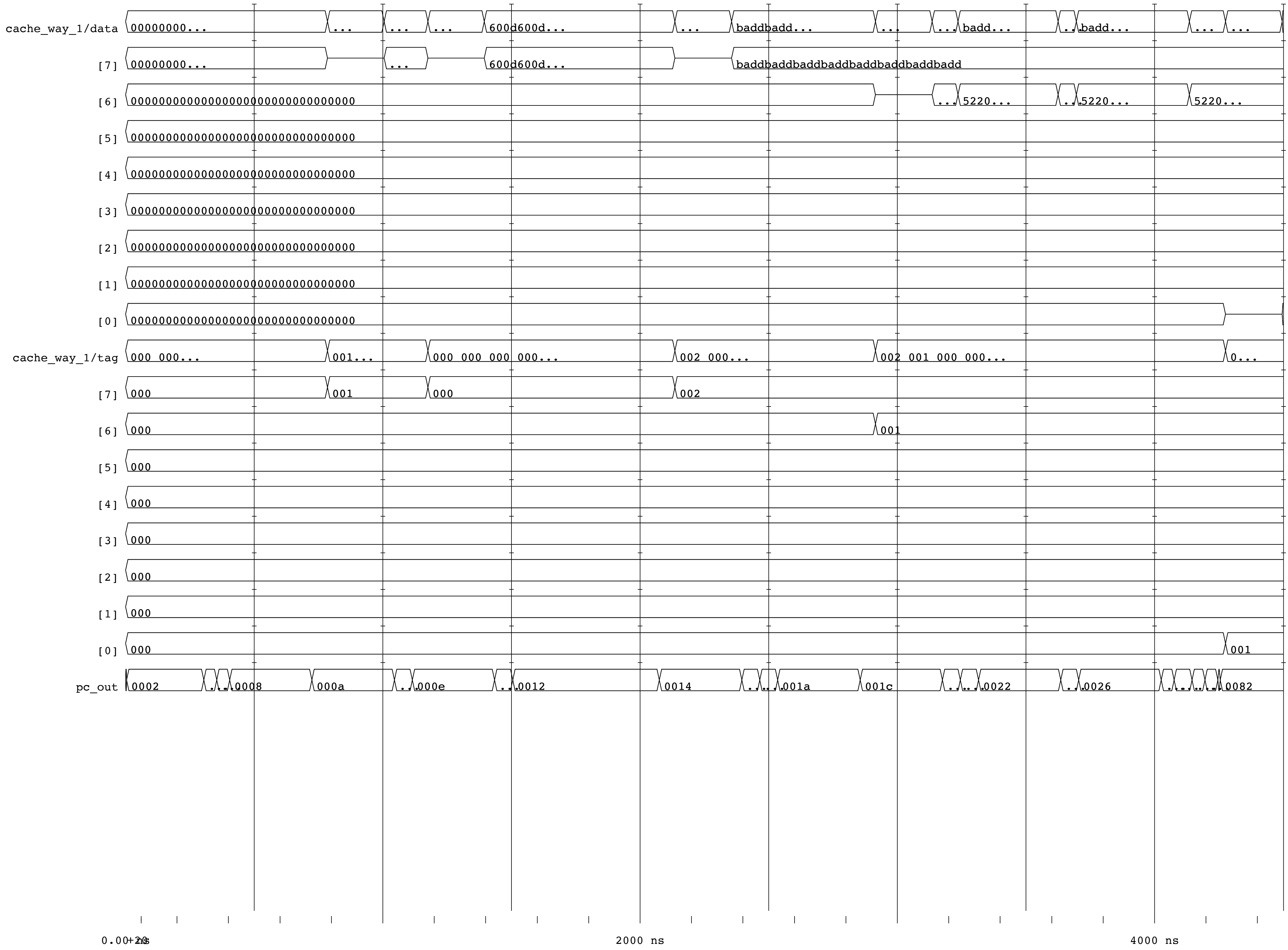
```
ORIGIN 4x0000
SEGMENT CodeSegment:
    LEA R0, L_DATA
    LDB R1, R0, 0
    LDB R2, R0, 1
    NOT R1, R1
    NOT R2, R2
    STB R1, R0, 0
    STB R2, R0, 1
HALT:             ; Infinite loop to keep the processor
    BRnzp HALT    ; from trying to execute the data below.
L_DATA:
    DATA2 4xAA55
```

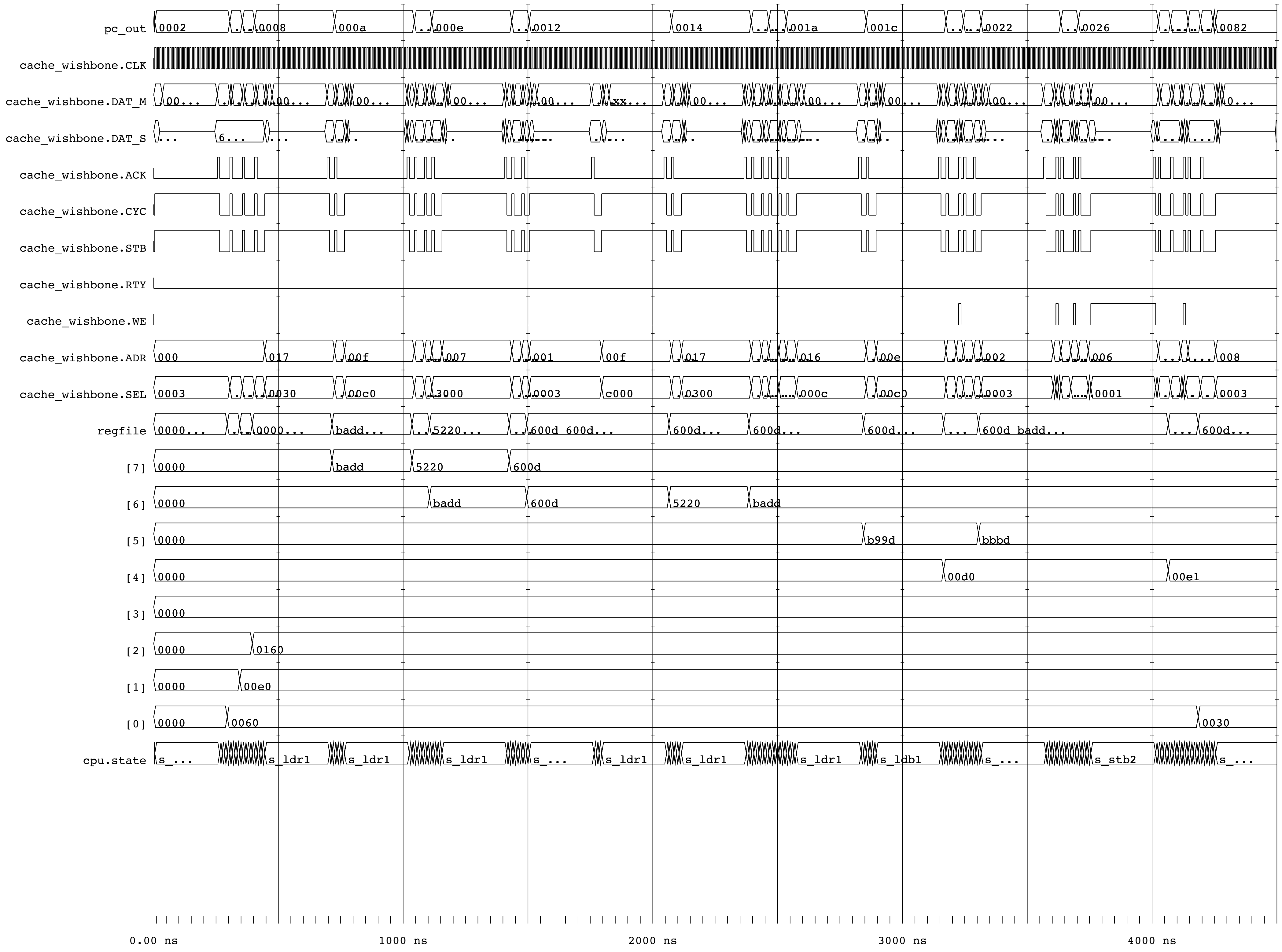
This methodology allowed to me near immediately solve a bug I encountered when first performing my the test case. I incorrectly assumed the ACK signal remained high for only one cycle and would drop by itself. This led dirty misses to **write back** correctly but immediately compete the **read in** state as the **ACK** never dropped. Since I had written down exactly what I expected to happen at each step, it was easy to find the mistake and rectify the issue.

I also went through the additional step of comparing my processor's state to the simulator's at random points, checking that all the registers matched their expected values. This set of tests and preparation made me confident the final mp2 test would work.

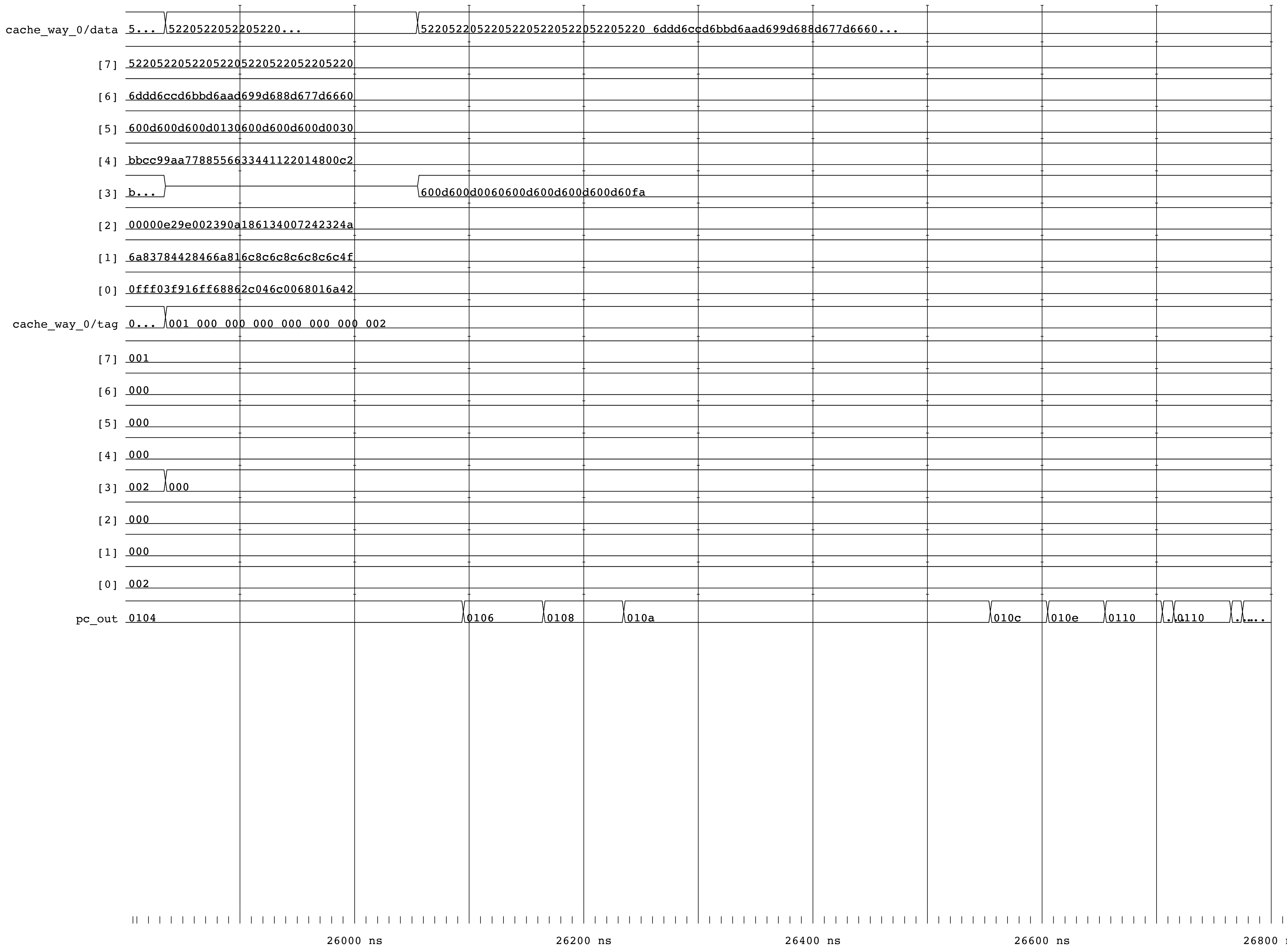


Entity:mp2_tb Architecture: Date: Sun Feb 25 14:42:15 CST 2018   Row: 1 Page: 1

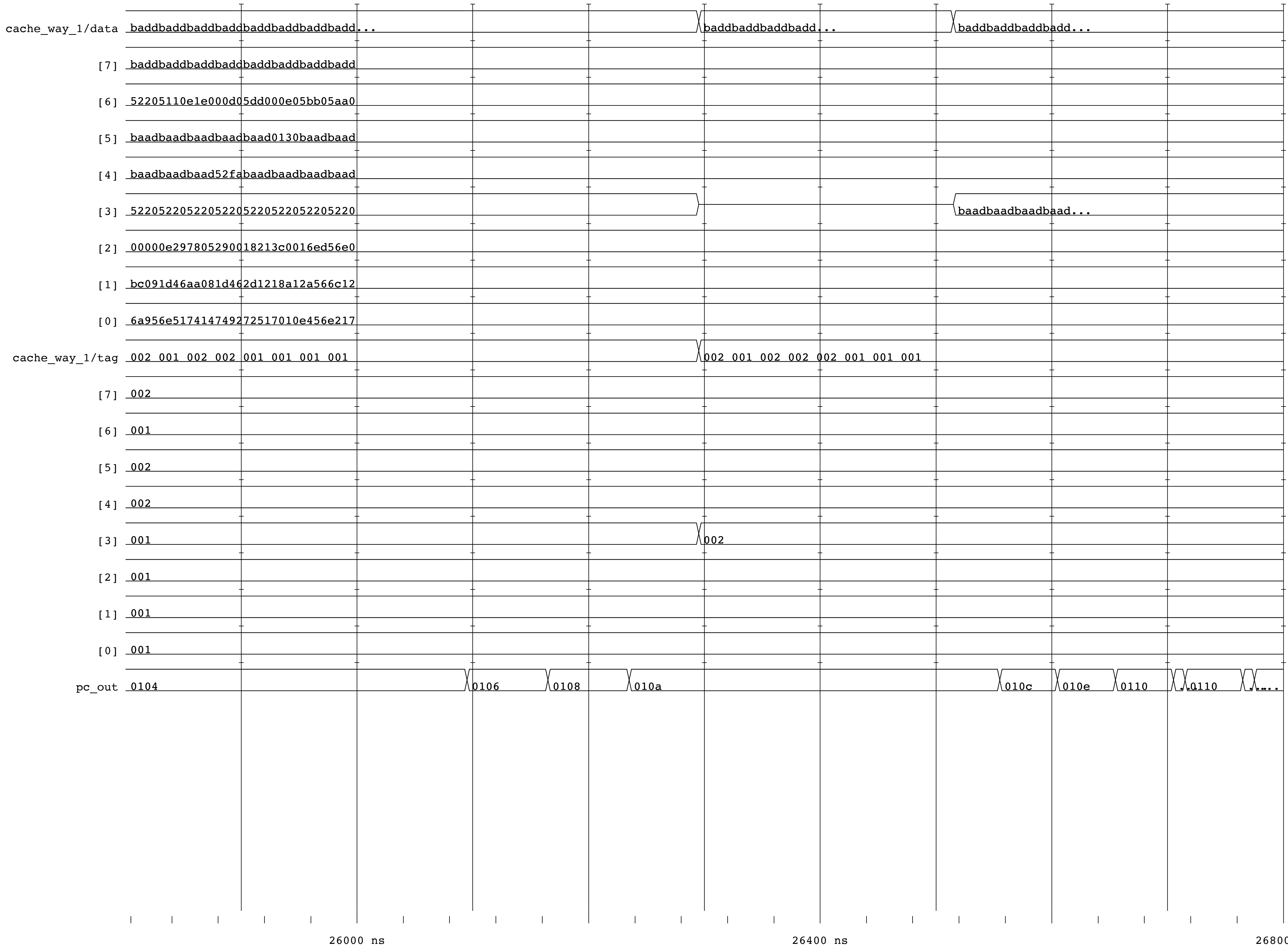

cache_way_1/data
00000000...
600d600d...
baddbadd...
badd...
[7]
baddbaddbaddbaddbaddbaddbaddbadd
[6]
00000000000000000000000000000000
5220...
[5]
[4]
[3]
[2]
[1]
[0]
cache_way_1/tag
000 000...
001...
000 000 000 000...
002 000...
002 001 000 000...
0...
000
001
002
pc_out
0002
0008
000a
000e
0012
0014
001a
001c
0022
0026
0082
0.00+20 ns
2000 ns
4000 ns



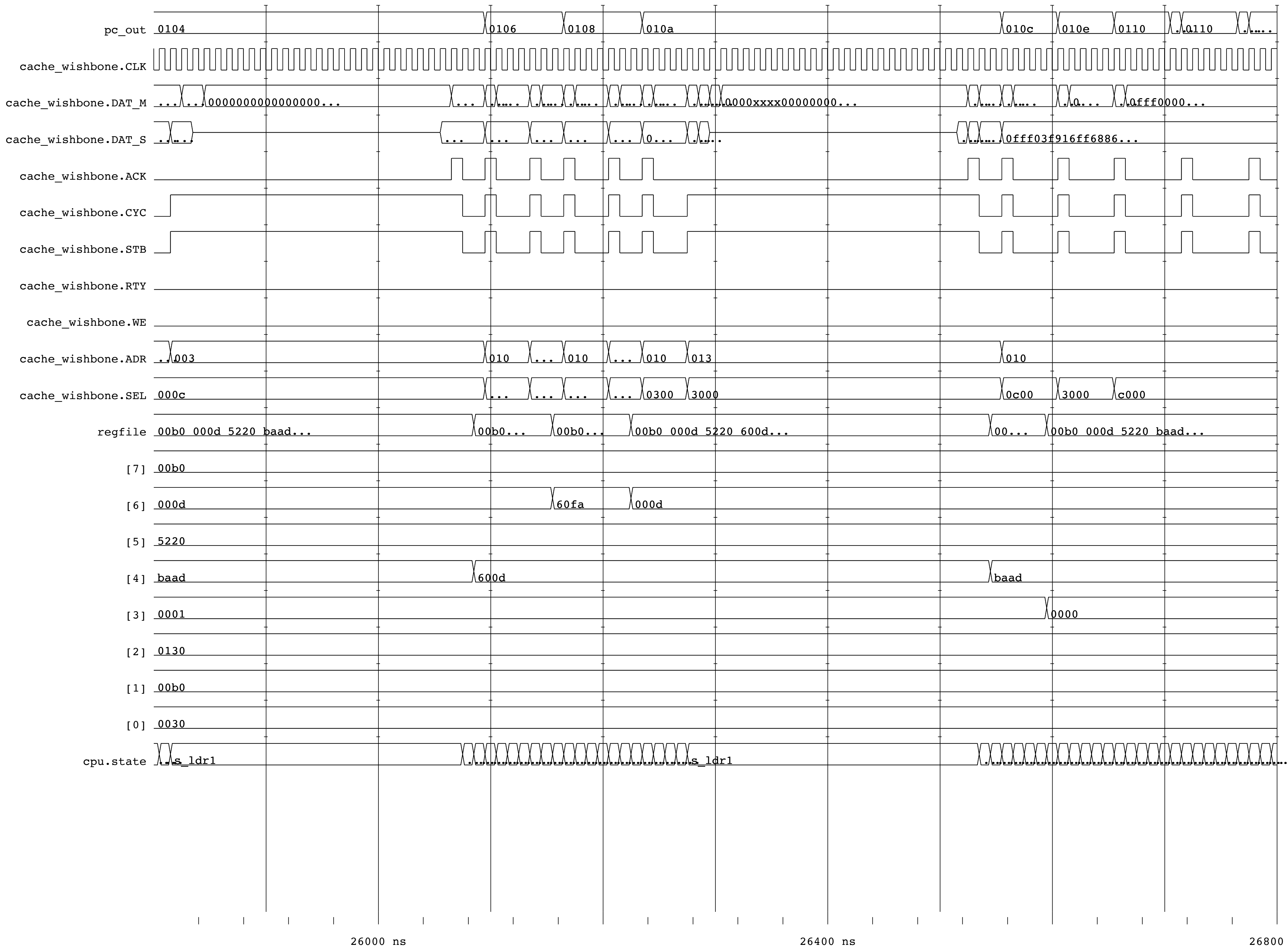
Entity:mp2_tb Architecture: Date: Sun Feb 25 15:13:49 CST 2018 Row: 1 Page: 1



Entity:mp2_tb Architecture: Date: Sun Feb 25 14:42:41 CST 2018   Row: 1 Page: 1



Entity:mp2_tb Architecture: Date: Sun Feb 25 14:44:11 CST 2018   Row: 1 Page: 1


pc_out
cache_wishbone.CLK
cache_wishbone.DAT_M
cache_wishbone.DAT_S
cache_wishbone.ACK
cache_wishbone.CYC
cache_wishbone.STB
cache_wishbone.RTY
cache_wishbone.WE
cache_wishbone.ADR
cache_wishbone.SEL
regfile
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
cpu.state
0104
0106
0108
010a
010c
010e
0110
0000000000000000...
0000xxxx00000000...
0fff0000...
0fff03f916ff6886...
003
010
013
000c
0300
3000
0c00
c000
00b0 000d 5220 baad...
00b0 000d 5220 600d...
00b0
000d
60fa
5220
baad
600d
0001
0000
0130
0030
s_ldr1
26000 ns
26400 ns
26800

List of Memory Writes

```
        ps delta memory_wishbone.WE                  memory_wishbone.ADR
                   memory_wishbone.DAT_M
         0    +0 x xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxx
         0    +1 0 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxX
   5305000    +3 1 600d600d600d600d600d600d600d0030 005
   5535000    +3 0 600d600d600d600d600d600d600d0030 005
   5865000    +3 1 baadbaadbaadbaadbaad0130baadbaad 015
   6095000    +3 0 5220522052205220522052200b05220 00d
   6415000    +7 1 baadbaadbaadbaadbaad0130baadbaad 015
   6645000    +3 0 baadbaadbaadbaadbaad0130baadbaad 015
   6975000    +7 1 5220522052205220522052200b05220 00d
   7205000    +3 0 600d600d600d0130600d600d600d0030 005
  11955000    +7 1 600d600d0060600d600d600d600d60fa 003
  12185000    +3 0 600d600d0060600d600d600d600d60fa 003
```

Timing Analysis Report

```
TimeQuest Timing Analyzer report for mp2
Sun Feb 25 15:57:02 2018
Quartus II 32-bit Version 13.1.4 Build 182 03/12/2014 SJ Full Version

---------------------
; Table of Contents ;
---------------------
  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      ; mp2                                                   ;
; 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                 ;
+----------------------+------+--------+-----------+-------+-------
+------------+-----------+-------------+-------+--------+-----------
+------------+----------+--------+--------+-------------------------+
; memory_wishbone.CLK ; Base ; 10.000 ; 100.0 MHz ; 0.000 ; 5.000 ;
;          ;           ;       ;       ;          ;           ;
;        ;        ; { memory_wishbone.CLK } ;
+----------------------+------+--------+-----------+-------+-------
+------------+-----------+-------------+-------+--------+-----------
+------------+----------+--------+--------+-------------------------+

+------------------------------------------------------------+
; Slow 1100mV 85C Model Fmax Summary                         ;
+------------+-----------------+---------------------+------+
; Fmax       ; Restricted Fmax ; Clock Name          ; Note ;
+------------+-----------------+---------------------+------+
; 108.05 MHz ; 108.05 MHz      ; memory_wishbone.CLK ;      ;
```

```
+------------+----------------+--------------------+------+
This panel reports FMAX for every clock in the design, regardless of the
user-specified clock periods.  FMAX is only computed for paths where the
source and destination registers or ports are driven by the same clock.
Paths of different clocks, including generated clocks, are ignored.  For
paths between a clock and its inversion, FMAX is computed as if the rising
and falling edges are scaled along with FMAX, such that the duty cycle (in
terms of a percentage) is maintained. Altera recommends that you always use
clock constraints and other slack reports for sign-off analysis.

+------------------------------------+
; 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: Sun Feb 25 15:56:51 2018
Info: Command: quartus_sta mp2 -c mp2
Info: qsta_default_script.tcl version: #1
Info (11104): Parallel Compilation has detected 8 hyper-threaded processors.
However, the extra hyper-threaded processors will not be used by default.
Parallel Compilation will use 4 of the 4 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: 'mp2.out.sdc'
Info: Found TIMEQUEST_REPORT_SCRIPT_INCLUDE_DEFAULT_ANALYSIS = ON
Info: Analyzing Slow 1100mV 85C Model
Info (332146): Worst-case setup slack is 0.745
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     0.745               0.000 memory_wishbone.CLK
Info (332146): Worst-case hold slack is 0.304
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     0.304               0.000 memory_wishbone.CLK
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.370
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     4.370               0.000 memory_wishbone.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 1.324
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     1.324               0.000 memory_wishbone.CLK
Info (332146): Worst-case hold slack is 0.279
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     0.279               0.000 memory_wishbone.CLK
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.372
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     4.372               0.000 memory_wishbone.CLK
Info: Analyzing Fast 1100mV 0C Model
Info (332146): Worst-case setup slack is 3.554
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     3.554               0.000 memory_wishbone.CLK
Info (332146): Worst-case hold slack is 0.180
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     0.180               0.000 memory_wishbone.CLK
Info (332140): No Recovery paths to report
Info (332140): No Removal paths to report
Info (332146): Worst-case minimum pulse width slack is 4.650
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= =================== =====================
    Info (332119):     4.650               0.000 memory_wishbone.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: 536 megabytes
    Info: Processing ended: Sun Feb 25 15:57:02 2018
    Info: Elapsed time: 00:00:11
    Info: Total CPU time (on all processors): 00:00:06
```