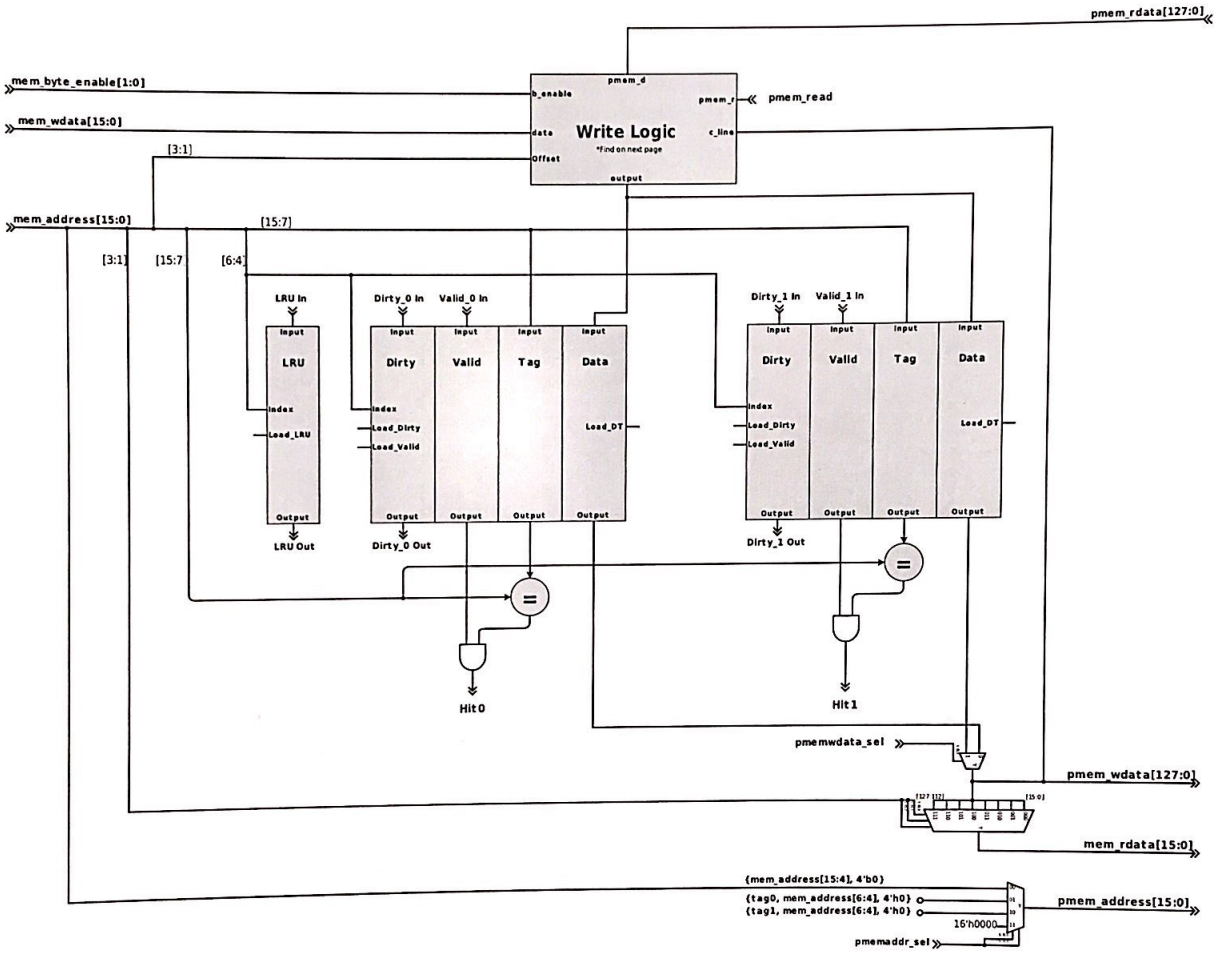
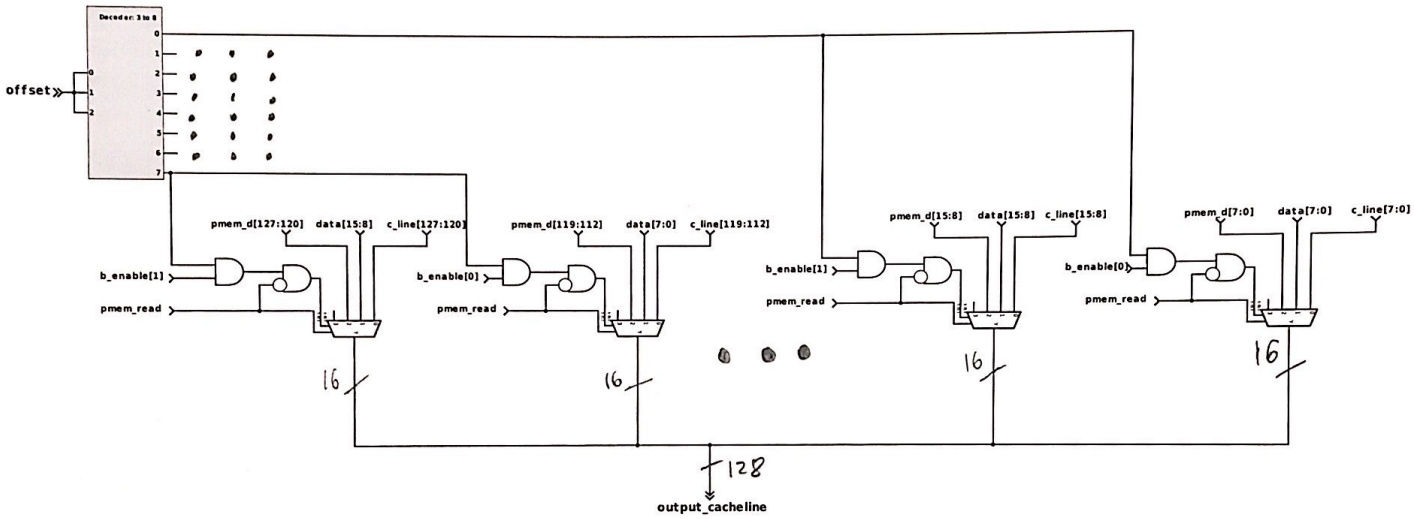
Cache Datapath

Rishi Thakkar (rrthakk2)
MP2: CP3
ECE 411
2/26/2017



Write Logic

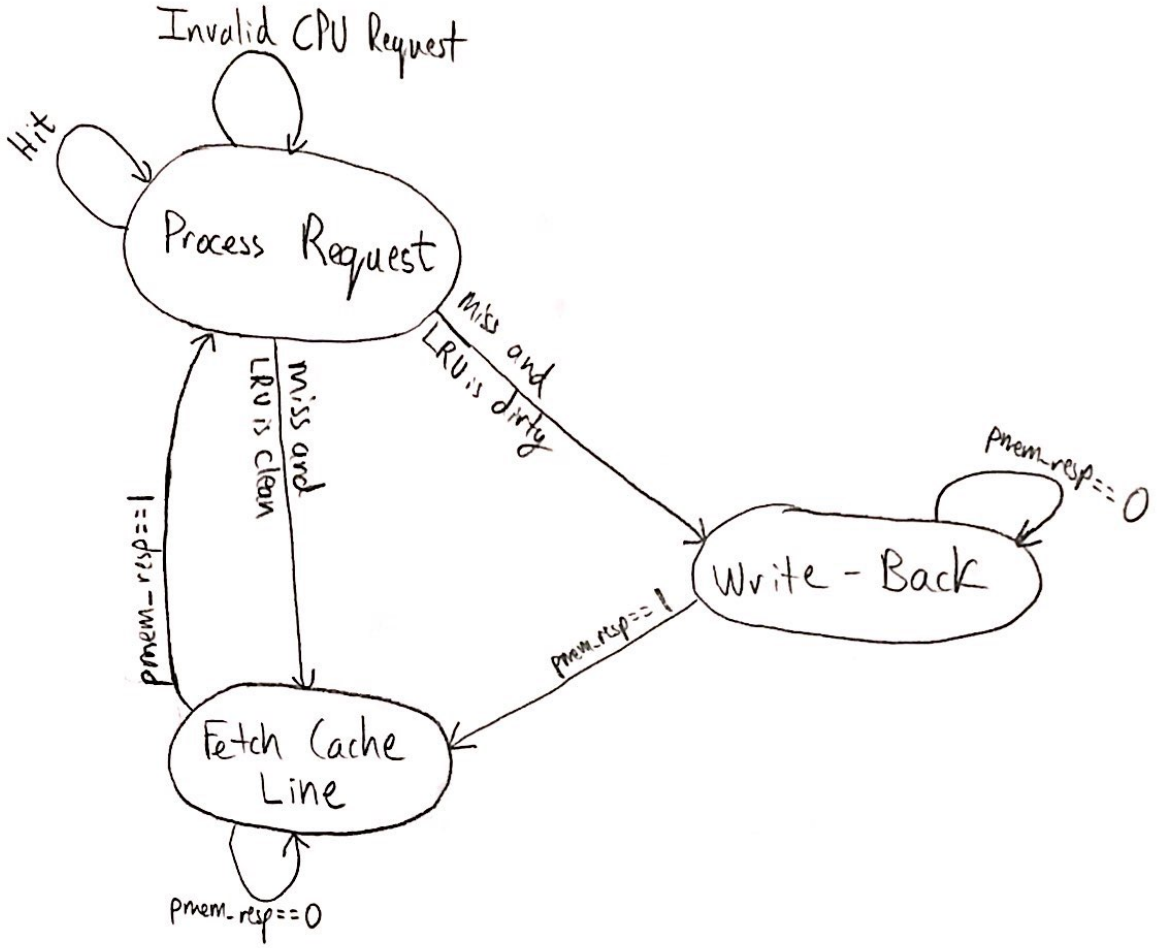
Rishi Thakkar (rrthakk2)
MP2: CP3
ECE411



The dot-dot-dot in the schematic above is used to signify the repetition in the structure of the mux usage. There will be 12 more muxes required to account for all the bytes in a cache line. Inputs into the muxes will be assigned using the same pattern seen above.

Cache Controller

Rishi Thakkar (rvthakk2)
MP2 : CP3
ECE 411



States:

1) Process Request: In this state, the controller checks for the validity of a CPU request. If valid, then it checks and processes a hit within one cycle. If there is a miss and LRU is dirty, then the controller will transition to the Write-Back state. If there is a miss and LRU is clean, then the controller goes to the Fetch Cache Line state.

2) Write-Back: This state is used as a part of the replacement policy to write back the least recently used way. The controller uses the input of the LRU and sets the pmemwdata_sel and pmemaddr_sel accordingly to pick the correct way to write back. Once pmem_resp becomes 1, the controller transitions to the Fetch Cache Line state.

3) Fetch Cache Line: In this state, the controller fetches a cache line directly from physical memory and writes it into set specified by the index and the way specified by the LRU bit. Once pmem_resp becomes 1, the controller transitions to the Process Request state to process the hit.

Note: Process Request is process_request, Write-Back is write_back, and Fetch Cache Line is fetch_cline

In MP2, we designed a 2-way set associative cache to reduce the memory latency in the LC3 design. The modules that were used in this datapath of cache are listed and explained below:

way module:
The way module was created as a wrapper around all of the arrays associated with a given way. This module takes in the inputs of clk, load_d, load_v, load_TD, index, d_in, v_in, tag_in, and data_in. Its outputs are d_out, v_out, tag_out, and data_out. It takes the inputs and relays them directly to the submodules within it and takes the output from those arrays and brings them into the datapath of the cache. The main purpose of this module is to reduce the clutter in the datapath module and modularly bundle together related components.

cache_writelogic module:
The cache_writelogic module is used to pick the input value into the cache data arrays. The inputs into this module are pmem_read, mem_byte_enable, offset, mem_wdata, pmem_rdata, and cur_cacheline. The output is output_cacheline. The module consists of 16 4 to 1 muxes and one 3 to 8 decoder module. The offset is passed into the decoder module which decodes the value and selects one of the 8 output words. mem_byte_enable, pmem_read, and the output from the decoder are used combinationaly to set the select signals for all of the muxes. The 16 muxes get the 16 different bytes from pmem_rdata and the cur_cacheline and the muxes alternate between the low and high bytes of the mem_wdata. Through the use of the control signals described earlier, this module is able to splice and pass the inputs as required to produce a desirable cache line input to the data arrays.

decoder3 module:
The decoder3 module is a 3 to 8 decoder that is used as a submodule within the cache_writelogic module. It takes in a 3 bit input and produces a 8 bit output. The output consists of a single bit that has been set to 1. The location of this bit is specified by the 3 bit input. The output from this module was used as a part of combinational control logic within the cache_writelogic module to select the word being written to.

wayselector_mux:
The wayselector_mux is a 2 to 1 mux which is used to pick between the data outputs from the 2 ways. The pmemwdata_sel signal from the the control unit selects between these two values.

mem_rdata_mux:
The mem_rdata_mux is a 8 to 1 mux which is used to select the word from within the cache line that needs to be sent to the CPU. Bits [3:1] of the mem_address are used to choose this word.

pmemaddr_mux:
The mem_rdata_mux is a 4 to 1 mux which is used to select the address that will be sent to the physical memory. By default, the mem_address is directly relayed to the physical memory. However, when there is a miss that requires a write-back, the address needs to be a concatenation of the stored tag bits, index bits, and 4 zeros. The pmemaddr_sel signal from the the control unit is used to select between these 3 values.

Verification is one of the most important parts of the design process. Due to this, a design must be put through various tests to make sure that it functions properly. My testing strategy consists of 2 phases. The first phase is a visual verification phase. During this phase, I made sure that all of the modules in the cache datapath and mp2 top-level module were connected correctly. I used the RTL viewer functionality that is built into Quartus to do this. This simple visual check allowed me verify that the design did not have have any connection related issues. Through the use of visual verification, I was able to fix few simple bugs and speed up the testing process.

Once the visual verification phase had ended, the code based verification phase began. Since I had already tested the LC3b extensively during mp1, I decided to test the cache separately through the use of a testbench (cache_tb.sv). This allowed me to modify signals as I needed to and test very specific situations. As I started to write the testbench, I realized that testing a partially used cache would allow me to test many of the edge cases that exist. To give the illusion of a partially used cache, I commented out the line of code that initialized the cache arrays to 0 and then used a do file (cache_test.do) to load the cache arrays with preset values. My testbench consisted a total of 18 different tests. These tests are were carried out in the following order:

1. Read miss, clean, way 0, set 0
2. Read hit, clean, way 0, set 0
3. Write hit, clean, way 0, set 0, high byte
4. Write hit, way 0, set 0, low byte
5. Write miss, clean, way 0, set 1
6. Read hit, dirty, way 0, set 1
7. Read miss, dirty, way 0, set 2
8. Write miss, dirty, way 0, set 3
9. Write hit, dirty, way 0, set 3
10. Read miss, clean, way 1, set 0
11. Read hit, clean, way 1, set 0
12. Write hit, clean, way 1, set 0, high byte
13. Write hit, way 1, set 0, low byte
14. Write miss, clean, way 1, set 1
15. Read hit, dirty, way 1, set 1
16. Read miss, dirty, way 1, set 2
17. Write miss, dirty, way 1, set 3
18. Write hit, dirty, way 1, set 3

I order to carry out these tests I also wrote a very simple LC3 program (memory.asm) to load the memory with a few data values. I then modified mem_read, mem_write, mem_address, mem_byte_enable, and mem_wdata directly while waiting for mem_resp. After executing the testbench I compared final values in the cache and memory to the theoretical results that I had calculated. I also checked to make sure that hits take one clock-cycle to process while misses go to physical memory. Since all of the values for the 18 tests matched the expected results, I believe that I can safely claim that my cache design is able to function perfectly.

Locations of files used for testing:
Preset Values: mp2/simulation/modelsim/cache_presets/
cache_test.do: mp2/simulation/modelsim/
memory.asm: mp2/testcode/
cache_tb.sv: mp2/

```
1   module cache_tb;
2
3   timeunit 1ns;
4   timeprecision 1ns;
5
6   logic clk;
7   logic pmem_resp;
8   logic pmem_read;
9   logic pmem_write;
10  logic [15:0] pmem_address;
11  logic [127:0] pmem_rdata;
12  logic [127:0] pmem_wdata;
13  logic mem_read, mem_write, mem_resp;
14  logic [1:0] mem_byte_enable;
15  logic [15:0] mem_address, mem_wdata, mem_rdata;
16
17  /* Clock generator */
18  initial clk = 0;
19  always #5 clk = ~clk;
20
21  initial
22  begin: TEST_SIGNALS
23     mem_read = 0;
24     mem_write = 0;
25     mem_byte_enable = 0;
26     mem_address = 0;
27     mem_wdata = 0;
28
29     /********************* Way 0 Tests **********************/
30     // Read miss, clean, way 0, set 0
31     #5
32        mem_read = 1;
33     wait(mem_resp == 1);
34     #10
35        mem_read = 0;
36
37     // Read hit, clean, way 0, set 0
38     #10
39        mem_address = {9'h0, 3'h0,3'h1,1'b0};
40        mem_read = 1;
41     wait(mem_resp == 1);
42     #10
43        mem_read = 0;
44
45     // Write hit, clean, way 0, set 0, high byte
46     #10
47        mem_address = {9'h0, 3'h0,3'h1,1'b0};
```

```
48          mem_wdata = 16'h600D;
49          mem_byte_enable = 2'b10;
50          mem_write = 1;
51        wait(mem_resp == 1);
52        #10
53          mem_write = 0;
54
55        // Write hit, way 0, set 0, low byte
56        #10
57          mem_address = {9'h0, 3'h0,3'h1,1'b0};
58          mem_wdata = 16'h600D;
59          mem_byte_enable = 2'b01;
60          mem_write = 1;
61        wait(mem_resp == 1);
62        #10
63          mem_write = 0;
64
65        // Write miss, clean, way 0, set 1
66        #10
67          mem_address = {9'h0, 3'h1,3'h0,1'b0};
68          mem_wdata = 16'h600D;
69          mem_byte_enable = 2'b11;
70          mem_write = 1;
71        wait(mem_resp == 1);
72        #10
73          mem_write = 0;
74
75        // Read hit, dirty, way 0, set 1
76        #10
77          mem_address = {9'h0, 3'h1,3'h0,1'b0};
78          mem_read = 1;
79        wait(mem_resp == 1);
80        #10
81          mem_read = 0;
82
83        // Read miss, dirty, way 0, set 2
84        #10
85          mem_address = {9'h0, 3'h2,3'h0,1'b0};
86          mem_read = 1;
87        wait(mem_resp == 1);
88        #10
89          mem_read = 0;
90
91        // Write miss, dirty, way 0, set 3
92        #10
93          mem_address = {9'h0, 3'h3,3'h0,1'b0};
94          mem_wdata = 16'h600D;
```

```
95          mem_byte_enable = 2'b11;
96          mem_write = 1;
97        wait(mem_resp == 1);
98        #10
99          mem_write = 0;
100
101       // Write hit, dirty, way 0, set 3
102       #10
103         mem_address = {9'h0, 3'h3,3'h1,1'b0};
104         mem_wdata = 16'h600D;
105         mem_byte_enable = 2'b11;
106         mem_write = 1;
107       wait(mem_resp == 1);
108       #10
109         mem_write = 0;
110
111       /********************* Way 1 Tests **********************/
112       // Read miss, clean, way 1, set 0
113       #10
114         mem_address = {9'h2, 3'h0,3'h0,1'b0};
115         mem_read = 1;
116       wait(mem_resp == 1);
117       #10
118         mem_read = 0;
119
120       // Read hit, clean, way 1, set 0
121       #10
122         mem_address = {9'h2, 3'h0,3'h1,1'b0};
123         mem_read = 1;
124       wait(mem_resp == 1);
125       #10
126         mem_read = 0;
127
128       // Write hit, clean, way 1, set 0, high byte
129       #10
130         mem_address = {9'h2, 3'h0,3'h1,1'b0};
131         mem_wdata = 16'h600D;
132         mem_byte_enable = 2'b10;
133         mem_write = 1;
134       wait(mem_resp == 1);
135       #10
136         mem_write = 0;
137
138       // Write hit, way 1, set 0, low byte
139       #10
140         mem_address = {9'h2, 3'h0,3'h1,1'b0};
141         mem_wdata = 16'h600D;
```

```
142         mem_byte_enable = 2'b01;
143         mem_write = 1;
144       wait(mem_resp == 1);
145       #10
146         mem_write = 0;
147
148       // Write miss, clean, way 1, set 1
149       #10
150         mem_address = {9'h2, 3'h1,3'h0,1'b0};
151         mem_wdata = 16'h600D;
152         mem_byte_enable = 2'b11;
153         mem_write = 1;
154       wait(mem_resp == 1);
155       #10
156         mem_write = 0;
157
158       // Read hit, dirty, way 1, set 1
159       #10
160         mem_address = {9'h2, 3'h1,3'h0,1'b0};
161         mem_read = 1;
162       wait(mem_resp == 1);
163       #10
164         mem_read = 0;
165
166       // Read miss, dirty, way 1, set 2
167       #10
168         mem_address = {9'h2, 3'h2,3'h0,1'b0};
169         mem_read = 1;
170       wait(mem_resp == 1);
171       #10
172         mem_read = 0;
173
174       // Write miss, dirty, way 1, set 3
175       #10
176         mem_address = {9'h2, 3'h3,3'h0,1'b0};
177         mem_wdata = 16'h600D;
178         mem_byte_enable = 2'b11;
179         mem_write = 1;
180       wait(mem_resp == 1);
181       #10
182         mem_write = 0;
183
184       // Write hit, dirty, way 1, set 3
185       #10
186         mem_address = {9'h2, 3'h3,3'h1,1'b0};
187         mem_wdata = 16'h600D;
188         mem_byte_enable = 2'b11;
```

```
189             mem_write = 1;
190         wait(mem_resp == 1);
191         #10
192             mem_write = 0;
193
194     end
195
196     cache DUT
197     (
198         .clk,
199
200         .mem_read, .mem_write, .mem_byte_enable,
201         .mem_address, .mem_wdata,
202         .mem_resp, .mem_rdata,
203
204         .pmem_resp, .pmem_rdata,

205         .pmem_read, .pmem_write, .pmem_address, .pmem_wdata
206     );
207
208     physical_memory memory
209     (
210         .clk,
211         .read(pmem_read),
212         .write(pmem_write),
213         .address(pmem_address),
214         .wdata(pmem_wdata),
215         .resp(pmem_resp),
216         .rdata(pmem_rdata)
217     );
218
219     endmodule : cache_tb
220
```

```
ORIGIN 0

;;************************** Way 0 **************************
;; This segment is used to test read miss/hit and write hit with with a
;; clean cache line in way 0
SEGMENT 0 TEST1:
DATA2 4x600D
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;; This segment is used to test write miss with with a clean and then read
;; hit with a dirty cache line in way 0
SEGMENT 16 TEST2:
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;; This segment is used to test read miss with with a dirty cache line in way 0
SEGMENT 16 TEST3:
DATA2 4xCAFE
DATA2 4xBABE
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;; This segment is used to test write miss/hit with with a dirty
;; cache line in way 0
SEGMENT 16 TEST4:
DATA2 4xBAAD
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;;************************** Way 1 **************************
;; This segment is used to test read miss/hit and write hit with with
;; a clean cache line in way 1
SEGMENT 208 TEST5:
DATA2 4x600D
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
```

```
;; This segment is used to test write miss with with a clean and
;; then read hit with a dirty cache line in way 1
SEGMENT 16 TEST6:
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;; This segment is used to test read miss with with a dirty cache line in way 1
SEGMENT 16 TEST7:
DATA2 4xCAFE
DATA2 4xBABE
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000

;; This segment is used to test write miss/hit with with a
;; dirty cache line in way 1
SEGMENT 16 TEST8:
DATA2 4xBAAD
DATA2 4xBAAD
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
DATA2 4x0000
```

LRU:
0
0
0
0
0
0
0
0

Way 0 - Tag:	Way 0 - Data:
001	00000000000000000000000000000000
001	11111111111111111111111111111111
001	22222222222222222222222222222222
001	33333333333333333333333333333333
001	44444444444444444444444444444444
001	55555555555555555555555555555555
001	66666666666666666666666666666666
001	77777777777777777777777777777777

Way 1 - Tag:	Way 1 - Data:
003	88888888888888888888888888888888
003	99999999999999999999999999999999
003	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
003	BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
003	CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
003	DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
003	EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
003	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

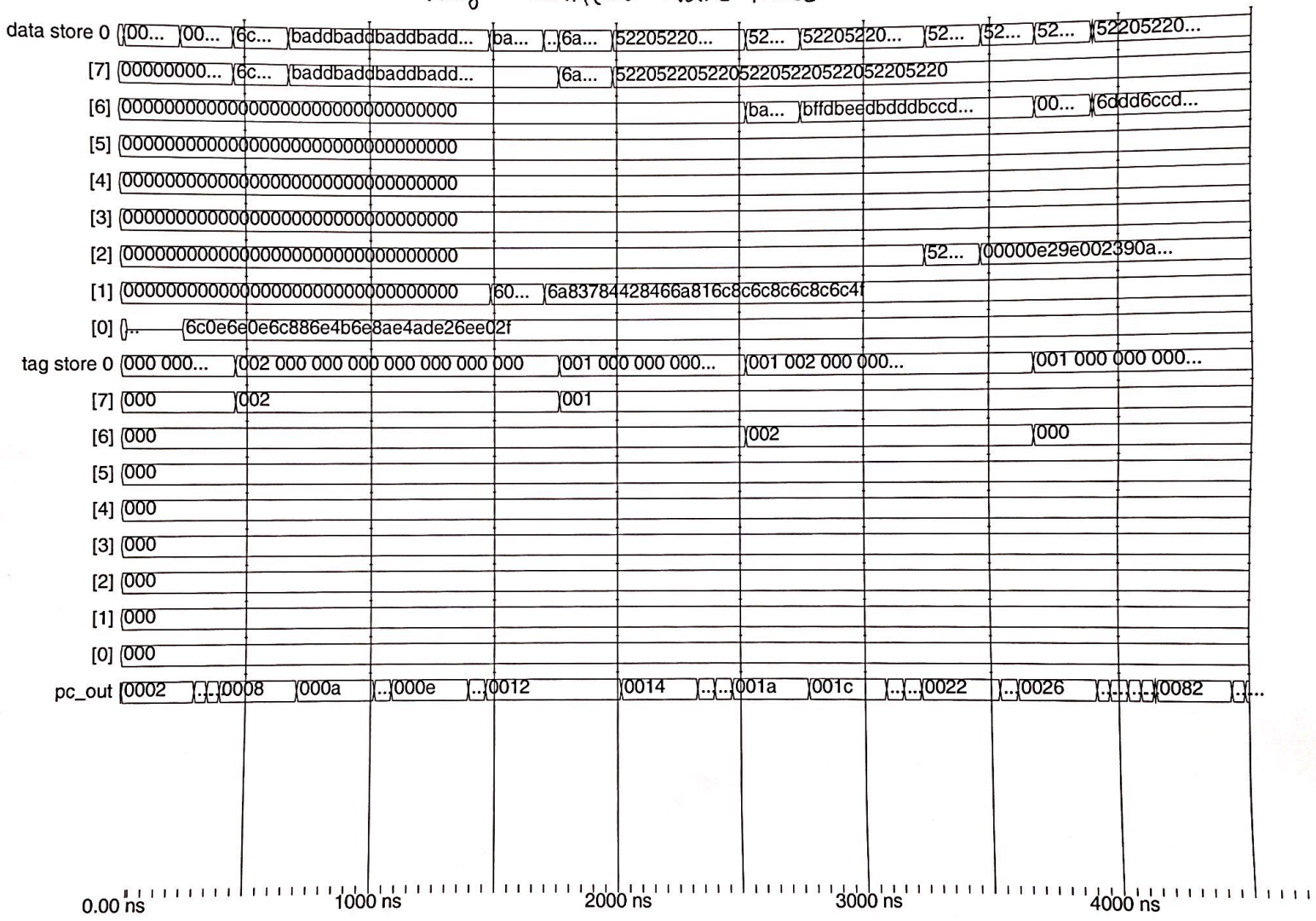
Way 0 and Way 1 - Dirty:
0
0
1
1
0
0
0
0

Way 0 and Way 1 - Valid:
0
0
1
1
0
0
0
0

Note: The preset values for each array are listed from index 0 to index 7.

Way 0: Initial Wave Trace

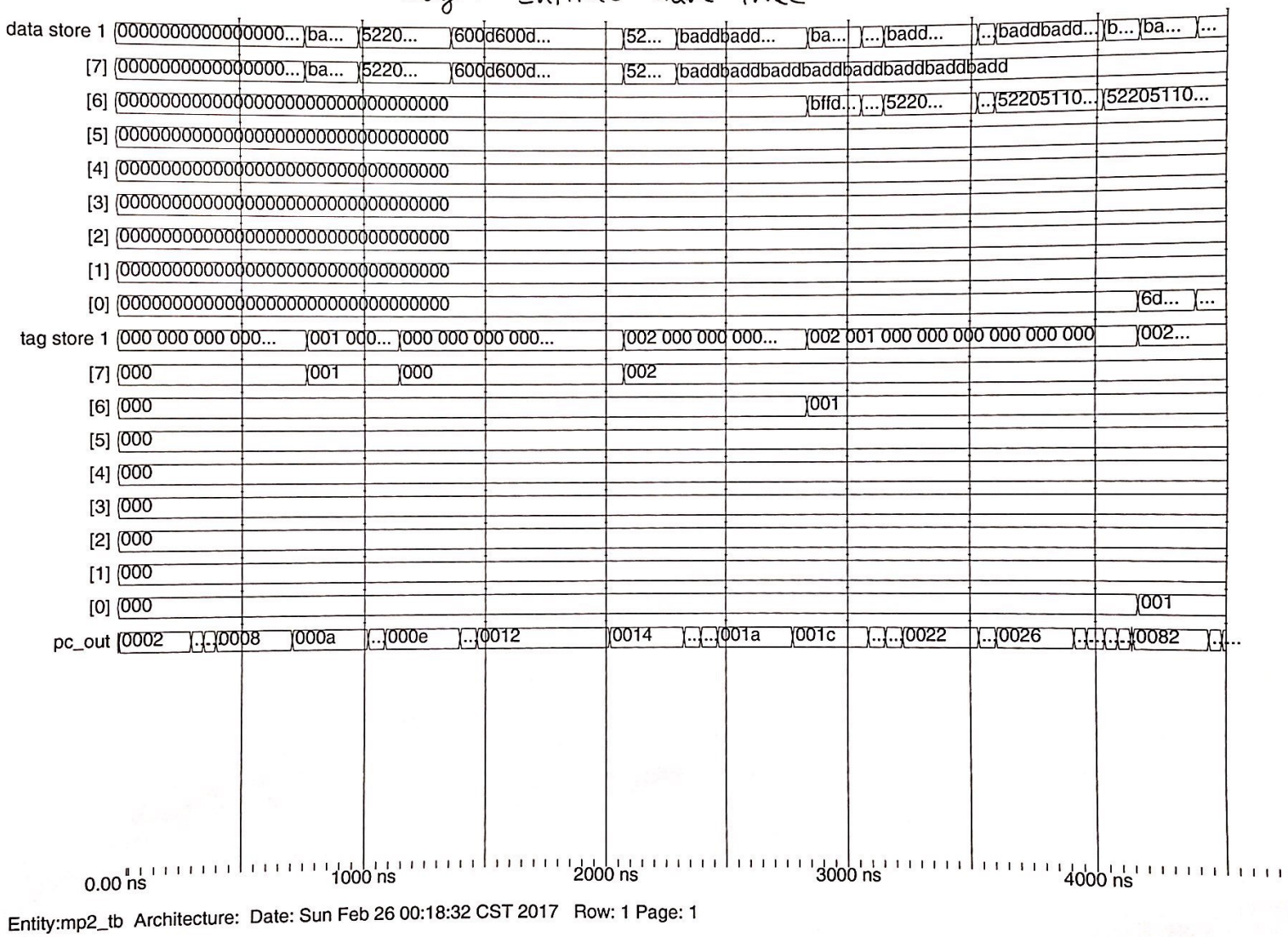
Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:10:04 CST 2017 Row: 1 Page: 1

Way 1: Initial Wave Trace

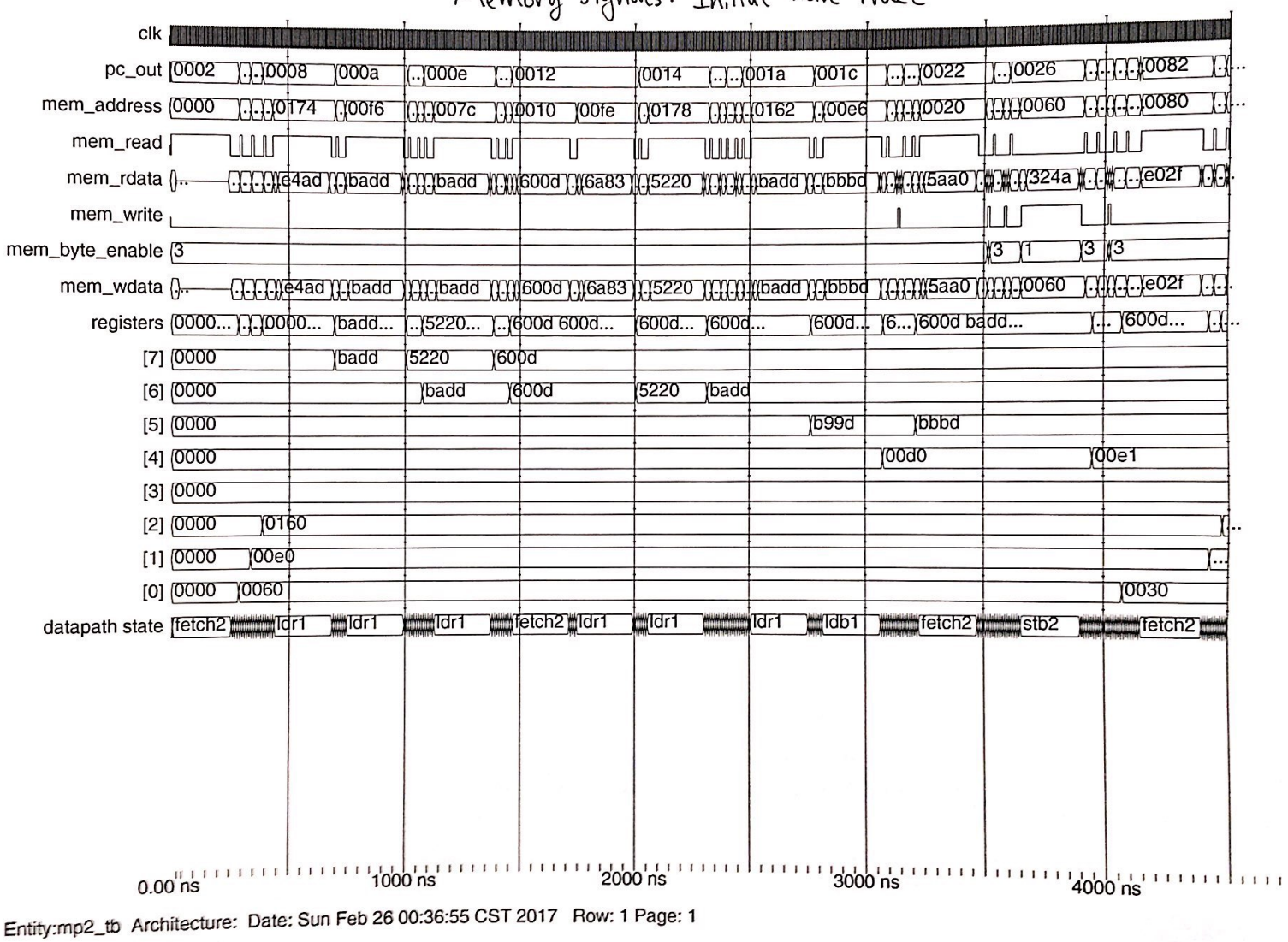
Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:18:32 CST 2017 Row: 1 Page: 1

Memory Signals: Initial Wave Trace

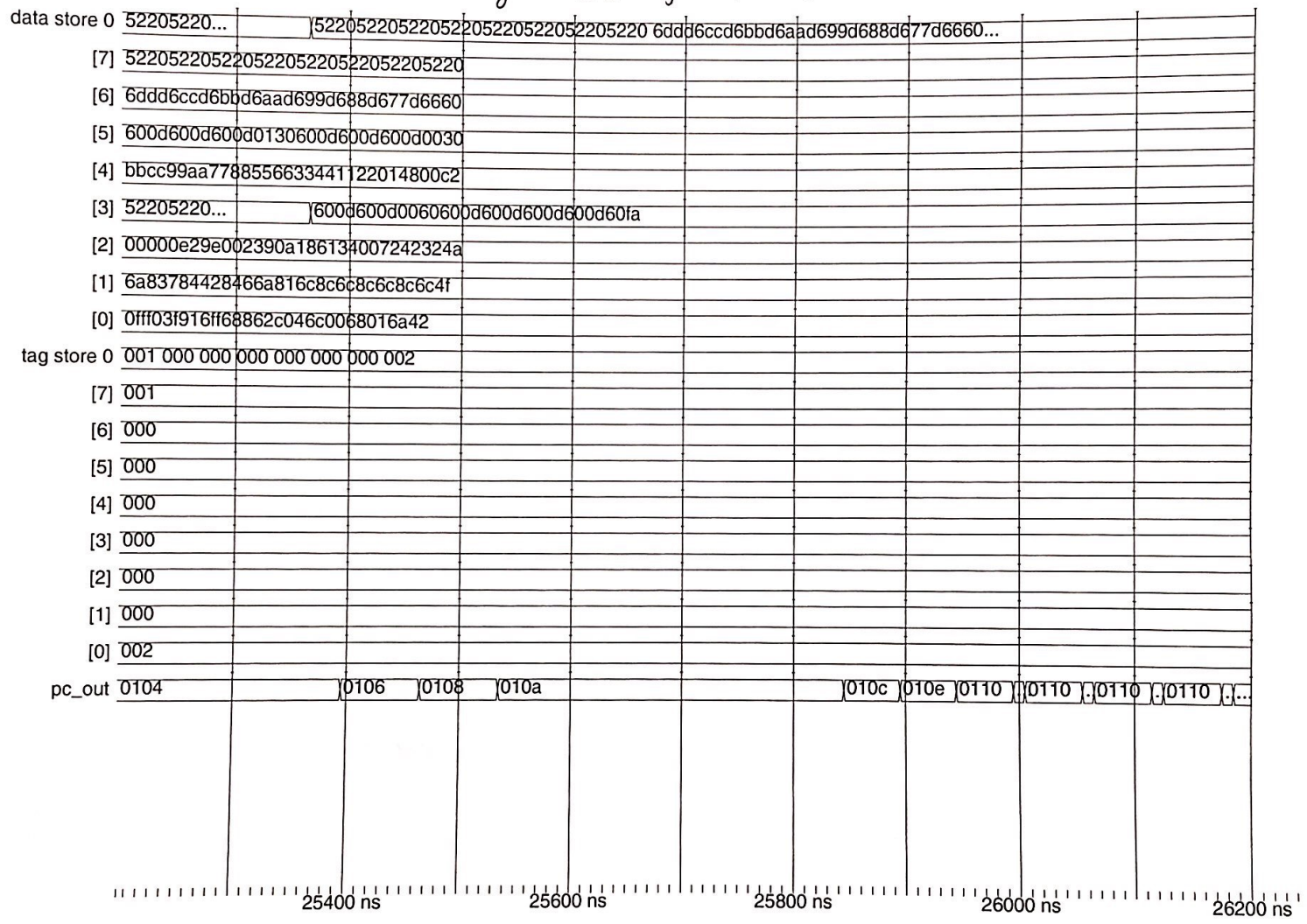
Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:36:55 CST 2017 Row: 1 Page: 1

Way 0: End of Test Code

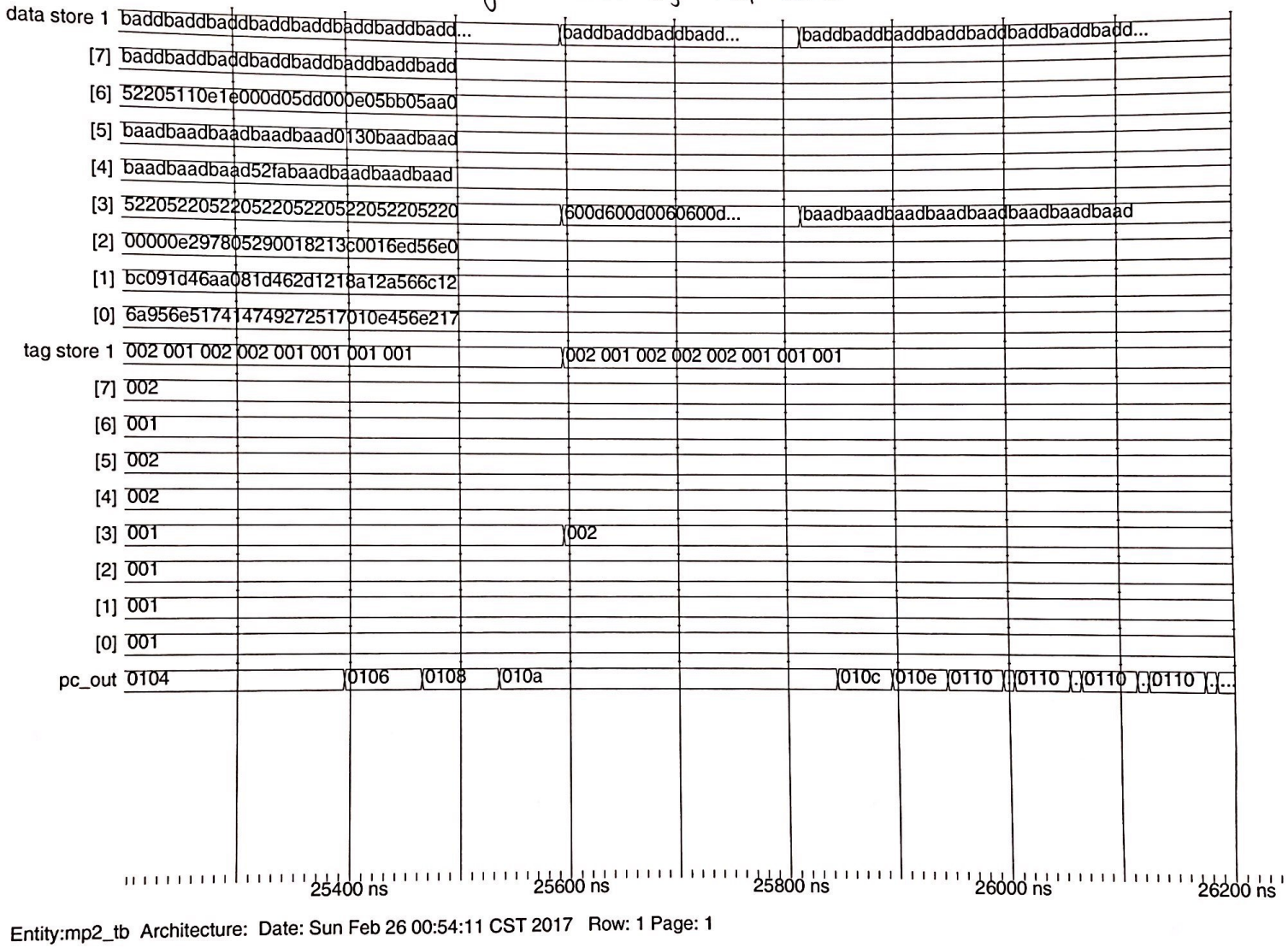
Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:45:38 CST 2017 Row: 1 Page: 1

Way 1: End of Test Code

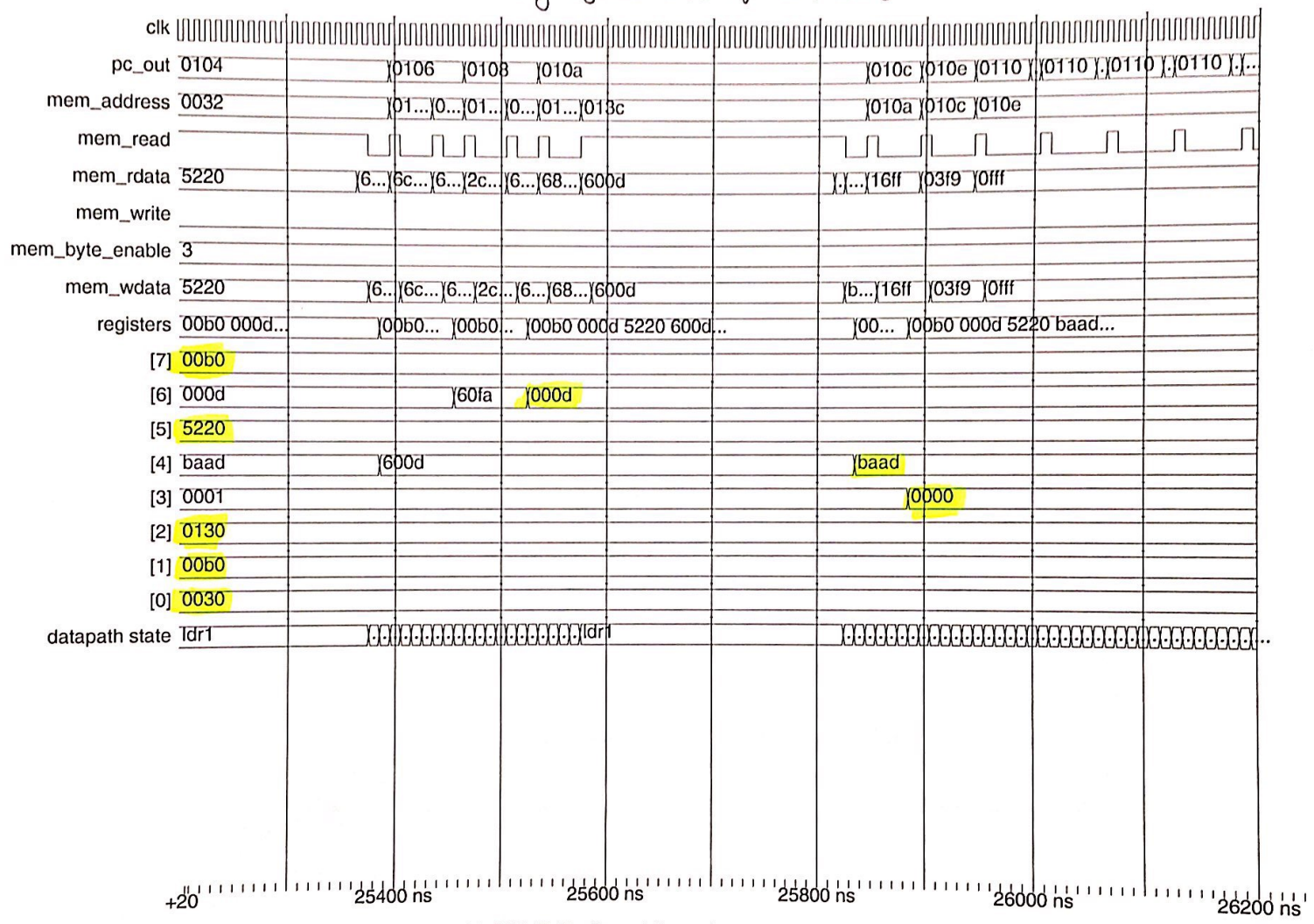
Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:54:11 CST 2017 Row: 1 Page: 1

Memory Signals: End of Test Code

Rishi Thakkar



Entity:mp2_tb Architecture: Date: Sun Feb 26 00:36:33 CST 2017 Row: 1 Page: 1

List of all Memory Writes

ps	delta	pmem_wdata	pmem_address	pmem_write
0	+0	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxx	x
0	+1	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxx	0
5165000	+3	600d600d600d600d600d600d600d0030	0150	1
5395000	+4	600d600d600d600d600d600d600d0030	0050	0
5705000	+3	baadbaadbaadbaadbaad0130baadbaad	0050	1
5935000	+4	52205220522052205220522000b05220	00d0	0
6235000	+6	baadbaadbaadbaadbaad0130baadbaad	00d0	1
6465000	+4	baadbaadbaadbaadbaad0130baadbaad	0150	0
6775000	+6	52205220522052205220522000b05220	0150	1
7005000	+4	600d600d600d0130600d600d600d0030	0050	0
11615000	+6	600d600d0060600d600d600d600d60fa	00b0	1
11845000	+4	600d600d0060600d600d600d600d60fa	0030	0

```
+--------------------------------------------------------------------------+
; 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  ; [...] ; 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 ;
+-----------+-----------------+------------+----+
; 109.0 MHz ; 109.0 MHz       ; 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 26 00:59:05 2017
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.sdc'
Info: Found TIMEQUEST_REPORT_SCRIPT_INCLUDE_DEFAULT_ANALYSIS = ON
Info: Analyzing Slow 1100mV 85C Model
Info (332146): Worst-case setup slack is 0.826
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.826               0.000 clk
Info (332146): Worst-case hold slack is 0.306
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
```

```
    Info (332119):     0.306               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 4.114
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.114               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 1.407
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     1.407               0.000 clk
Info (332146): Worst-case hold slack is 0.282
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.282               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 4.104
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.104               0.000 clk
Info: Analyzing Fast 1100mV 0C Model
Info (332146): Worst-case setup slack is 3.668
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     3.668               0.000 clk
Info (332146): Worst-case hold slack is 0.190
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     0.190               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 4.193
    Info (332119):     Slack       End Point TNS Clock
    Info (332119): ========= ==================== ====================
    Info (332119):     4.193               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: 519 megabytes
    Info: Processing ended: Sun Feb 26 00:59:11 2017
    Info: Elapsed time: 00:00:06
    Info: Total CPU time (on all processors): 00:00:04
```