CMPE 315: Principles of VLSI Design
Lab Cover Page

Project # : **1**
Project Title : **Project Submission 1**

Name : **Nem Negash and Sean Mullins**
Section : **1**

Date Submitted : **11/21/2021**

TA / Grader Use Only:

Late Submission Deduction (20% per day late):

Other Deductions:

Final Lab Grade:

Comments to student:

Table of Contents

1. Submission Description
    a. System Description
        i. The approach we took was to first start with the small pieces and work our way up. We first wanted to have a working cache where it writes and reads at the correct location using given input signals. We first started by creating a 1-bit cache that is made up of a positive level d-latch and a negative level transmission gate. This is so the bit is only written when clock goes high and reads the rest of the time. We used the 1-bit cache to create an 8-bit cache and then a single block. We used a decoder to set the write high to the specific byte during a write and a mux to send the correct output during a read. We then created the 8-block cache using the single block. Then used another decoder to decide which block to write to during a write and a mux to output from the correct read data from the blocks. We the proceeded to working on the state machine. The state machine needs a counter, so we decided to create that first. We decided to use a 19-bit shift register as the counter. We first created a single bit D-flip-flop using a master and slave latch using the Dlatch, then called it 19 times in the counter to create a 19-bit shift register. This shift register is then used in the state machine to keep track of the number of cycles during an operation. The rest of the state machine uses an encoder to decide what kind of operation is occurring to decide when to set the busy signal high or low depending on the number of cycles. Next, we created the hit or miss block which didn't really use any other entities but just simply compared the cpu_add tag and the tag of the block using an XOR and AND gate. The hit or miss also checked if the blocks valid was 1 and used those two outputs to determine if the operation is a hit or a miss. Then, we created the cache tag blocks and the cache valid blocks. These were created using the 1-bit cache entity similar to how the actual cache is made up except the cache tag was 3 bits instead of 8 and the valid was only 1 bit. Finally, we pieced all these main entities together in the chip. The chip first saves the R/W operation, cpu_data, and cpu_add using a Dlatch since they are only available for a limited time during the start of an operation. Next the chip tells the state machine to start counting the number of cycles. Using the counter to see the number of cycles the chip waits on specific cycles to set enables on entities like the cache so tell it when to write and when to read. And reading from the cache tag and cache valid to see if the operation is a hit or a miss. After an operation is done busy goes low and the state machine waits until the input start signal goes high to start the counter for the next operation and so on.
    b. Block Diagram

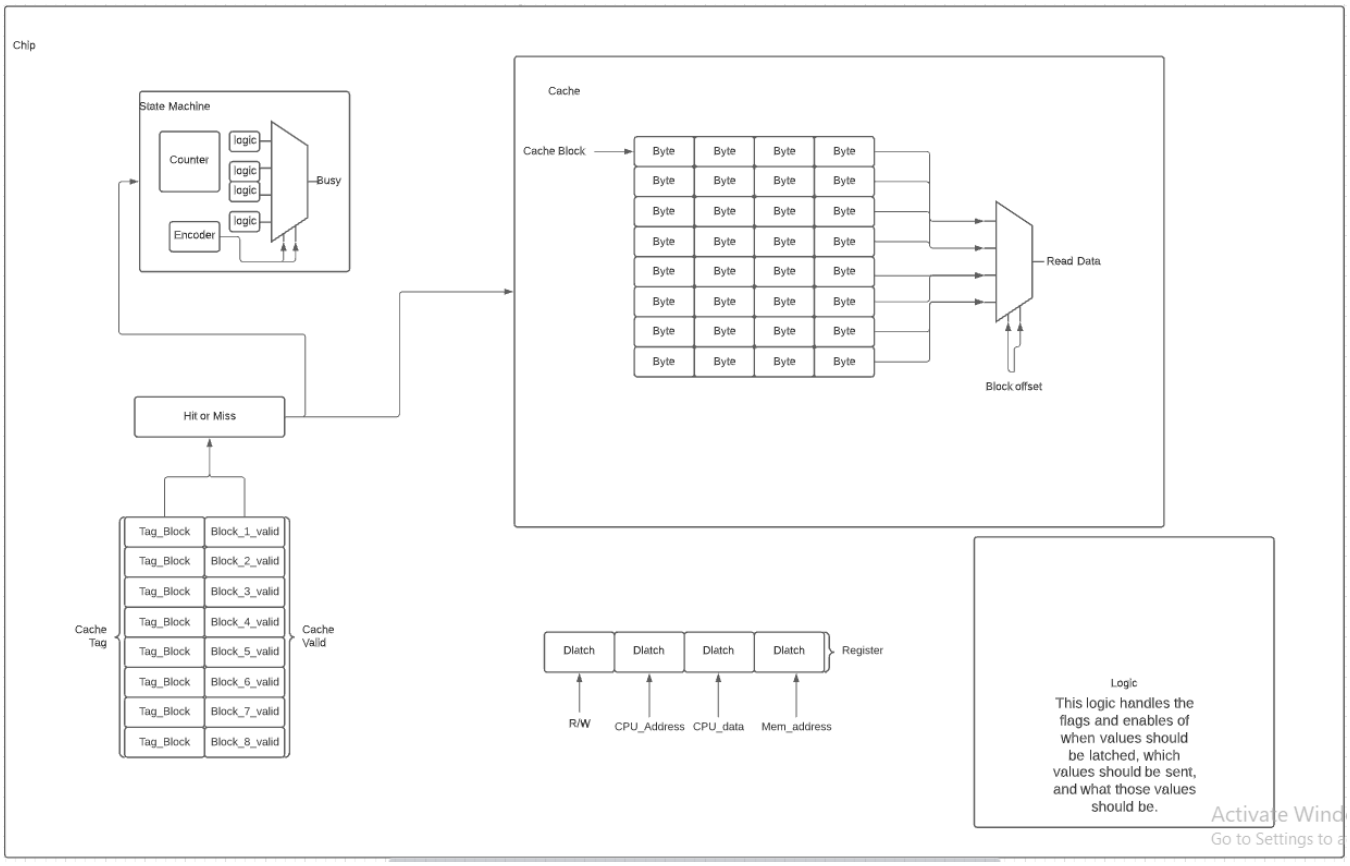

Figure 1: Block Diagram for system

2. System Breakdown
    a. Entity Architecture

*****Cascading blocks shows the other entities the entity is made up of.*****

Chip:

Hit_miss: Hit or miss block that uses logic to see if there is a hit or a miss
Dlatch: a positive level sensitive latch
Dlatch_8bit: an 8-bit positive sensitive latch
    Dlatch: a positive level sensitive latch
mux_8_bit: 4 to 1 8-bit multiplexer
    mux_4_1: 4 to 1 1-bit multiplexer
        mux_2_1: 2 to 1 1-bit multiplexer
mux2_1_8bit: 2 to 1 8-bit multiplexer
    mux_2_1: 2 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer
tran_gate: Transmission gate
Cache: main 8 blocks of cache
    Mux_8_1_8bit: 8 to 1 8-bit multiplexer
        mux_8_bit: 4 to 1 8-bit multiplexer
            mux_4_1: 4 to 1 1-bit multiplexer
                mux_2_1: 2 to 1 1-bit multiplexer
        mux_2_1: 2 to 1 1-bit multiplexer
    Decoder3_8: 3 to 8 decoder

Cache_block: a single block of the cache
Decoder2_4: 2 to 4 decoder
mux_8_bit: 4 to 1 8-bit multiplexer
mux_4_1: 4 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer
Cache_8bit: a single byte of cache
Cache_1bit: a single bit of cache
Dlatch: a positive level sensitive latch
Tran_gate: negative level transmission gate
Cache_tag: the tags of the 8 blocks of the cache
Decoder3_8: 3 to 8 decoder
Mux_8_1_3bit: 8 to 1 3-bit multiplexer
mux_3_bit: 4 to 1 3-bit multiplexer
mux_4_1: 4 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer
Cahe_tag_block: a single tag block
Cache_1bit: a single bit of cache
Dlatch: a positive level sensitive latch
Tran_gate: negative level transmission gate
Cache_valid: the valids for the 8 blocks of cache
Decoder3_8: 3 to 8 decoder
mux_4_1: 4 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer
dff: d-flip-flop
Dlatch: a positive level sensitive latch
State_machine: tells the chip if it is in busy state and what cycle it is on
Counter: a shift register counter
dff: d-flip-flop
Dlatch: a positive level sensitive latch
Encoder4_2: 4 to 2 encoder
mux_4_1: 4 to 1 1-bit multiplexer
mux_2_1: 2 to 1 1-bit multiplexer

b. Cache
   i. Cache Description
      1. The cache is made up a cache block entity that is called 8 time which is made up of a cache 8-bit entity that is called 4 times which is made up of a cache 1-bit entity that is called 8 times which is just a positive level Dlatch and a negative level transmission gate. The cache also takes in the R/W operation signal and a hit or miss signal to basically decide what to do.
   ii. Cache Test-Bench input
      10011001 --write
      11100000
      1
      1

00000000 --read
11100000
0
1
10101010 --write
11100101
1
1
00000000 --read
11100101
0
1
00000001 --write
11101010
1
1
00000000 --read
11101010
0
1
10000000 --write
11101111
1
1
00000000 --read
11101111
0
1

iii. Cache Test-Bench output

```
[xk28378@ite375pc19 proj] run_ncsim.bash -input ncsim.run -messages -cdslib cds.lib -hdlvar hdl.var cache_test
ncsim: 15.20-s035: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
Loading snapshot vhdl.cache_test:test ..................... Done
ncsim> run 400ns
 CPU_data:10011001 CPU_addr11100000 R_W:1 h_m:1 CLK:0
 rd_data:XXXXXXXX


 CPU_data:00000000 CPU_addr11100000 R_W:0 h_m:1 CLK:0
 rd_data:10011001


 CPU_data:10101010 CPU_addr11100101 R_W:1 h_m:1 CLK:0
 rd_data:XXXXXXXX


 CPU_data:00000000 CPU_addr11100101 R_W:0 h_m:1 CLK:0
 rd_data:10101010


 CPU_data:00000001 CPU_addr11101010 R_W:1 h_m:1 CLK:0
 rd_data:XXXXXXXX


 CPU_data:00000000 CPU_addr11101010 R_W:0 h_m:1 CLK:0
 rd_data:00000001


 CPU_data:10000000 CPU_addr11101111 R_W:1 h_m:1 CLK:0
 rd_data:XXXXXXXX


 CPU_data:00000000 CPU_addr11101111 R_W:0 h_m:1 CLK:0
 rd_data:10000000
```

Figure 2: Output for Cache

c. Counter

i. Counter Description

1. The counter is made up of 19 D-flip flops to account for the longest operation which is a read miss. The counter basically shifts a one across a 19-bit logic vector output after every clock cycle.

ii. Counter Test-Bench input

```
3.  --set reset high to start and then low after to see 1 shift across
4.    for i in 0 to 20 loop
5.      if(i <= 0) then
6.      rst_1 <= '1';
7.      else
8.      rst_1 <= '0';
9.      end if;
10.     wait until falling_edge(clock);
11.     print_output;
12.   end loop;
```

i. Counter Test-Bench output

ncsim> run 400ns

Clk: 0 Rst: 1

Out: 0000000000000000000

Clk: 0 Rst: 0
Out: 0000000000000000000

Clk: 0 Rst: 0
Out: 0000000000000000001

Clk: 0 Rst: 0
Out: 0000000000000000010

Clk: 0 Rst: 0
Out: 0000000000000000100

Clk: 0 Rst: 0
Out: 0000000000000001000

Clk: 0 Rst: 0
Out: 0000000000000010000

Clk: 0 Rst: 0
Out: 0000000000000100000

Clk: 0 Rst: 0
Out: 0000000000001000000

Clk: 0 Rst: 0
Out: 0000000000010000000

Clk: 0 Rst: 0
Out: 0000000000100000000

Clk: 0 Rst: 0
Out: 0000000001000000000

Clk: 0 Rst: 0
Out: 0000000010000000000

Clk: 0 Rst: 0
Out: 0000000100000000000

Clk: 0 Rst: 0
Out: 0000001000000000000

Clk: 0 Rst: 0
Out: 0000010000000000000

Clk: 0 Rst: 0
Out: 0000100000000000000

Clk: 0 Rst: 0
Out: 0001000000000000000

Clk: 0 Rst: 0
Out: 0010000000000000000

Clk: 0 Rst: 0
Out: 0100000000000000000

Clk: 0 Rst: 0
Out: 1000000000000000000

Clk: 0 Rst: 1
Out: 0000000000000000000

Clk: 0 Rst: 0
Out: 0000000000000000000

Clk: 0 Rst: 0
Out: 0000000000000000001

Clk: 0 Rst: 0
Out: 0000000000000000010

Clk: 0 Rst: 0
Out: 0000000000000000100

Clk: 0 Rst: 0
Out: 0000000000000001000

Clk: 0 Rst: 0
Out: 0000000000000010000

Clk: 0 Rst: 0
Out: 0000000000000100000

Clk: 0 Rst: 0
Out: 0000000000001000000

Clk: 0 Rst: 0
Out: 0000000000010000000

Clk: 0 Rst: 0
Out: 0000000000100000000

Clk: 0 Rst: 0
Out: 0000000001000000000

Clk: 0 Rst: 0
Out: 0000000010000000000

Clk: 0 Rst: 0
Out: 0000000100000000000

Clk: 0 Rst: 0
Out: 0000001000000000000

Clk: 0 Rst: 0
Out: 0000010000000000000

Clk: 0 Rst: 0
Out: 0000100000000000000

Clk: 0 Rst: 0
Out: 0001000000000000000

Ran until 400 NS + 0
ncsim> exit

b. State Machine
   i. State Machine Description
      1. The state machine basically uses the counter to see what state in the operation it is in. It checks where the one in the counter is to basically set busy to either high or low. It also outputs the counter to the chip to use in there.
   ii. State Machine Test-Bench input
   1 --Read Miss high
   0
   0
   0
   1 --reset high
   1 --Read miss high
   0
   0
   0
   0 --reset low
   iii. State Machine Test-Bench output
   ncsim> run 400ns
   RM:1 RH:0 WM:0 WH:0 CLK:0 rst:1

count:0000000000000000000 busy:0

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000001 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000010 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000100 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000001000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000010000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000100000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000001000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000010000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000100000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000001000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000010000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000100000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000001000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000010000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000100000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0001000000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0010000000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0100000000000000000 busy:0

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:1000000000000000000 busy:0

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000001 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0

count:0000000000000000010 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000000100 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000001000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000010000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000000100000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000001000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000010000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000000100000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000001000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000010000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000000100000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000001000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000010000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0000100000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0001000000000000000 busy:1

RM:1 RH:0 WM:0 WH:0 CLK:0 rst:0
count:0010000000000000000 busy:1

Ran until 400 NS + 0
ncsim> exit

c. Chip
   i. Chip Description
      1. The chip basically puts all the entities together. It saves the signals that are there for a limited amount of time at the beginning of an operation to a Dlatch to save for later uses in the operation states. It also uses the state machine to keep track of these states and depending on the states it enables reads and writes to the cache, cache tag, and cache valid.
   ii. Chip Test-Bench input
      1. Chip_test.vhd input
         ZZZZZZZZ
         ZZZZZZZZ
         Z
         0
         1
         ZZZZZZZZ
         ZZZZZZZZ
         ZZZZZZZZ
         Z
         0
         1
         ZZZZZZZZ
         00000000
         ZZZZZZZZ
         1
         1
         0
         ZZZZZZZZ

ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0

0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000000
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000000
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000001
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000001
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000010
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000010
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
00000011
ZZZZZZZZ
ZZZZZZZZ

Z
0
0
00000011
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
00000011
11111111
0
1
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ

00000011
ZZZZZZZZ
1
1
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
11111111
10101010
0
1
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0

0
ZZZZZZZZ
ZZZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ

2. Chip_full_test.vhd input
   The input for this test bench is way too long to put in the report but it is submitted alongside the report and all other files

iii. Chip Test-Bench output

1. Chip_test.vhd output

a. Chip waveform

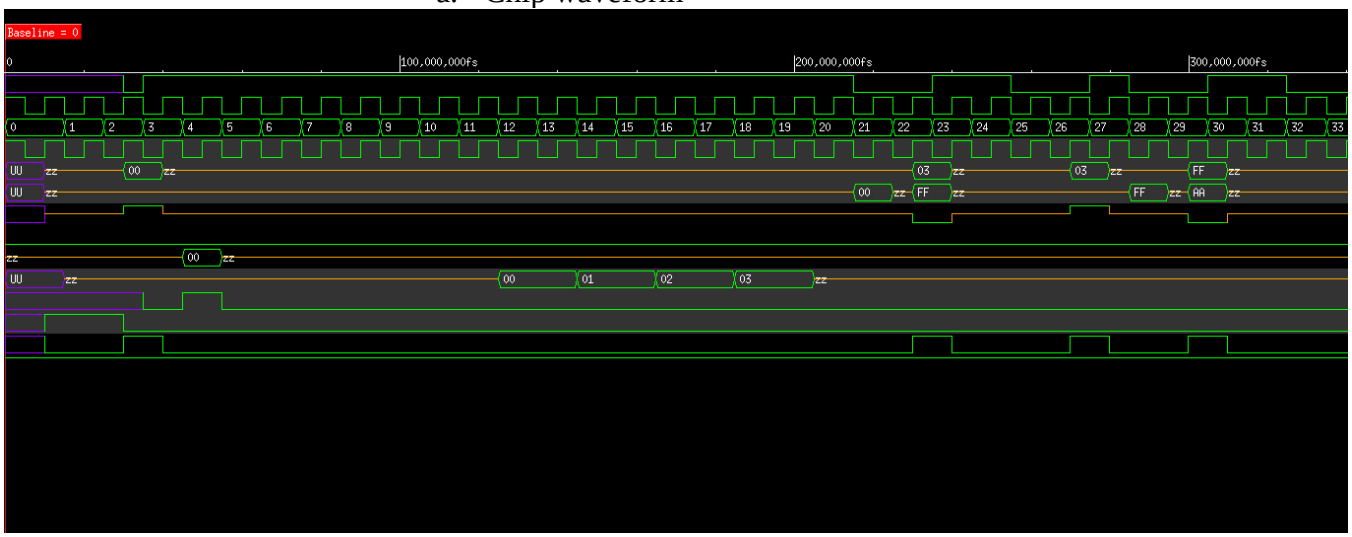

Figure 3: Chip waveform for chip_test.vhd

b. Cache waveform

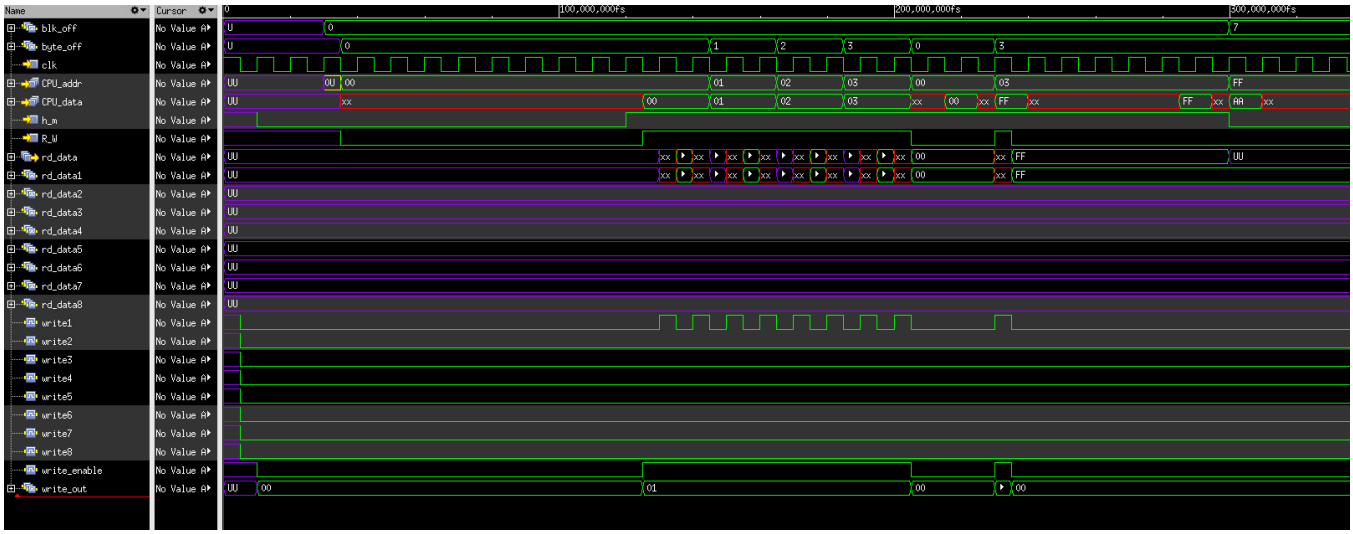

Figure 4: Cache waveform for chip_test.vhd

c. State Machine waveform

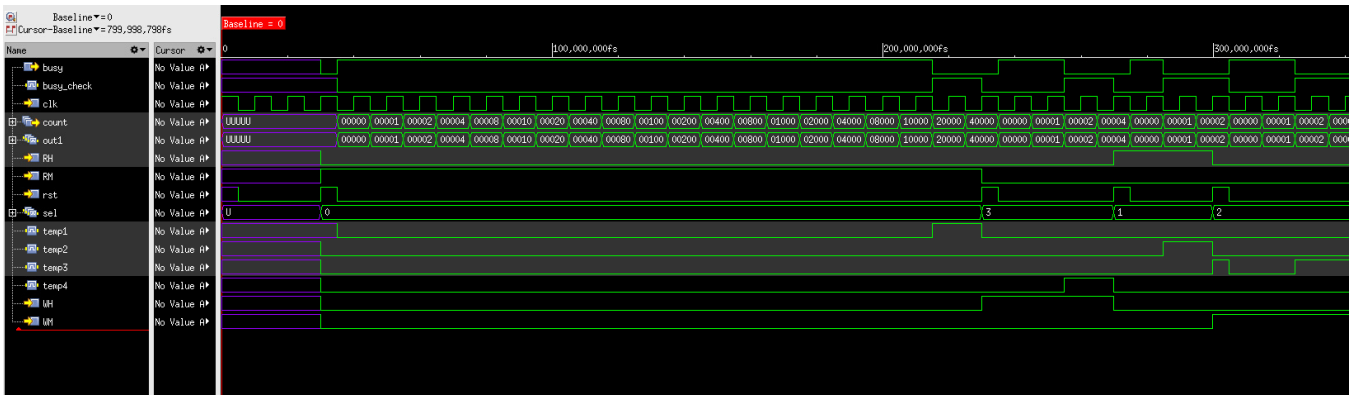

Figure 5: State Machine waveform for chip_test.vhd

d. Counter waveform

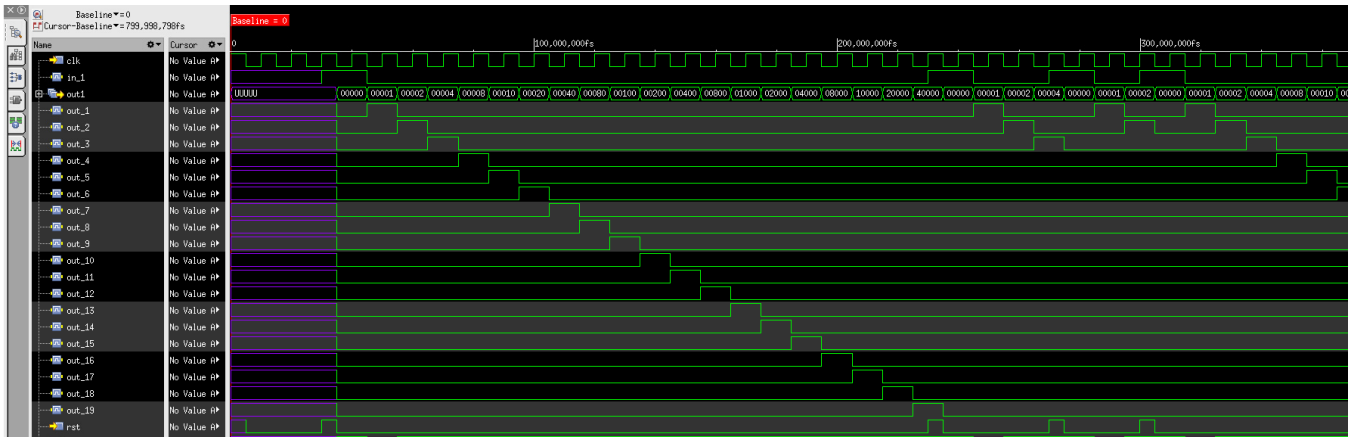

Figure 6: Counter waveform for chip_test.vhd

2. Chip_full_test.vhd output

a. Chip waveform

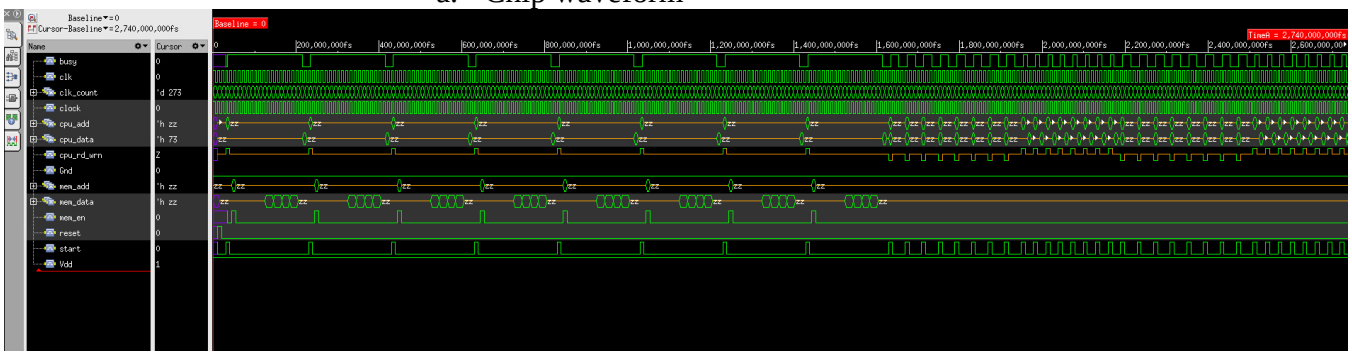

Figure 7: Chip waveform for chip_full_test.vhd

b. Cache waveform

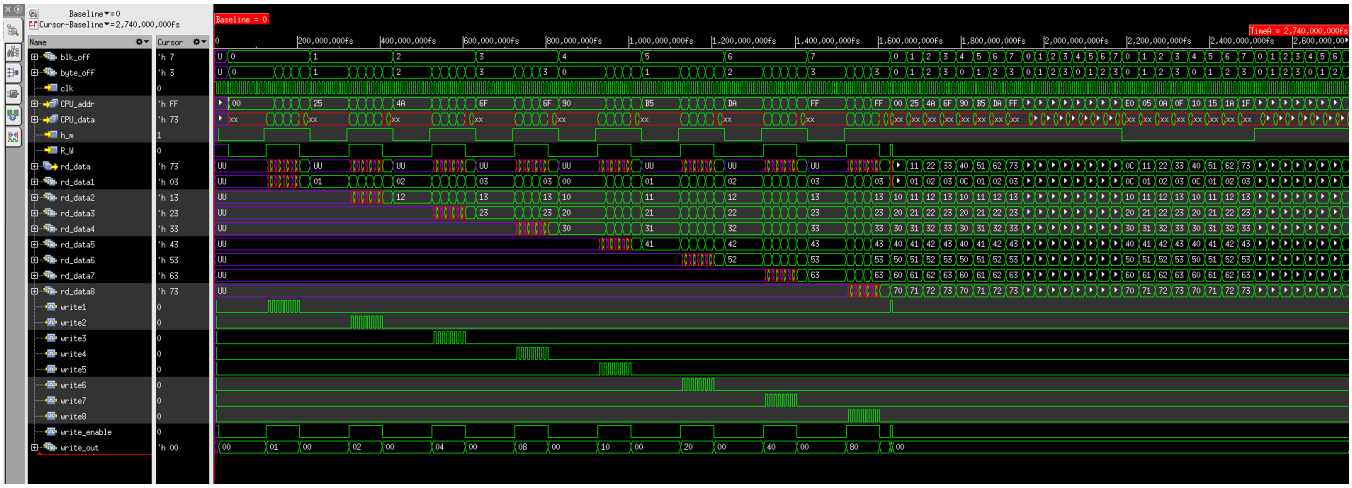

Figure 8: Cache waveform for chip_full_test.vhd

c. State Machine waveform

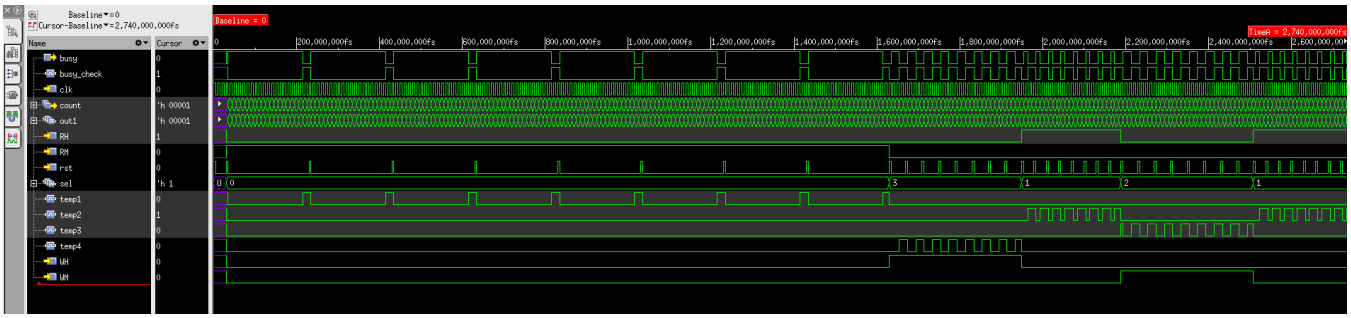

Figure 9: State Machine waveform for chip_full_test.vhd

d. Counter waveform

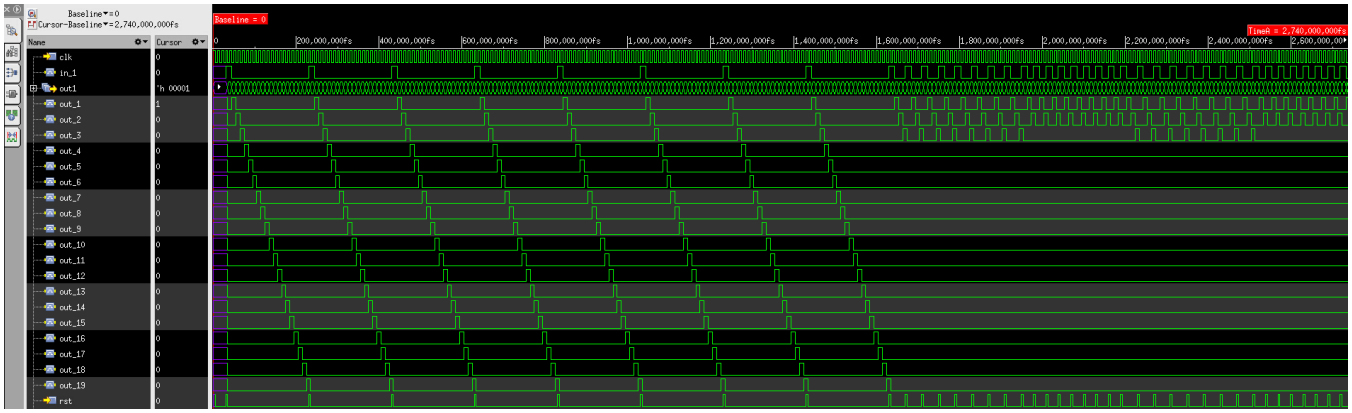

Figure 10: Counter waveform for chip_full_test.vhd

13. Work Breakdown

Team Member	Work Completed
Nem Negash	• State Machine • Chip • Encoder and Decoder

Sean Mullins	• Cache • Counter • Hit or Miss • Muxes