CMPE 315: Principles of VLSI Design

Lab Cover Page

Project Title : **Cache Design, Part 1**

Name : **Abenezer Wudenhe, Minhquan Tran**
Section : **01**

Date Submitted : **11/22/2017**

TA / Grader Use Only:

Late Submission Deduction (20% per day late):

Other Deductions:

Final Lab Grade:

Comments to student:

I. OBJECTIVE

The objective of this project is to design, implement, and simulate a direct-mapped cache (32-byte) using VHDL, followed by laying out a physical design using Cadence Virtuoso.

The extra credit portion requires that the cache is 2-way set associative with double the size (64-byte), following an LRU scheme for block replacement. In this project, we are submitting the structural VHDL code for a direct-mapped 32-byte cache for part 1, with the intention of developing the extra credit portion for part 2 submission.

II. PURPOSE

This document intends to outline the approach taken as we developed this project. It also serves as a manual for understanding our design choices, complete with an explanation for each and every structural module.

III. SPECIFICATION

A. System Requirements

Per design specifications listed on the project description, the cache is a 32-byte, direct-mapped, byte addressable memory that contains eight blocks, with each block holding four bytes. Figure 1 shows the organization of the described cache.

Valid	Tag	Blocks (4 bytes/block)			
1-bit	3-bit	8-bit	8-bit	8-bit	8-bit
...	...	...	...	...	...

Figure 1

In order to access a single byte within the block, we use an 8-bit CPU Address, called Ca[7:0]. Figure 2 shows the breakdown of the eight bits within Ca.

Figure 2

The three most significant bits, Ca[7:5], are used as a tag for comparison purposes. The next three most significant bits, Ca[4:2], are used to select one out of eight blocks. Lastly, the remaining two bits, Ca[1:0], are used to select one out of four bytes within the block.

B. Signals and Operations

Start: This signal goes high on the positive edge of the clock, providing half a clock cycle of setup time for other signals before other operations begin on the negative edge.

Busy: This signal indicates that the cache is currently being accessed.

Clock: A square wave synchronizing operations between cache, memory, and CPU.

Read/Write: Signals high when attempting to read and low when attempting to write.

CPU Address (Ca): An 8-bit bus carrying the address input for both read and write request; it is provided on rising edge.

CPU Data (Cd): An 8-bit bus carrying the data input when attempting to write into cache or data output when attempting to read from cache; it is provided on the rising edge.

Memory Address (Ma): An 8-bit bus carrying the memory address used to access main memory when there is a read miss.

Memory Data (Md): An 8-bit bus carrying the data input from main memory.

Read Hit/Miss: A signal that goes high when there is a read operation and the data is found within the cache. This is determined by comparing the tag bits within the CPU address to the tag bits within a cache block. If they match, it is a hit, otherwise it is a miss.

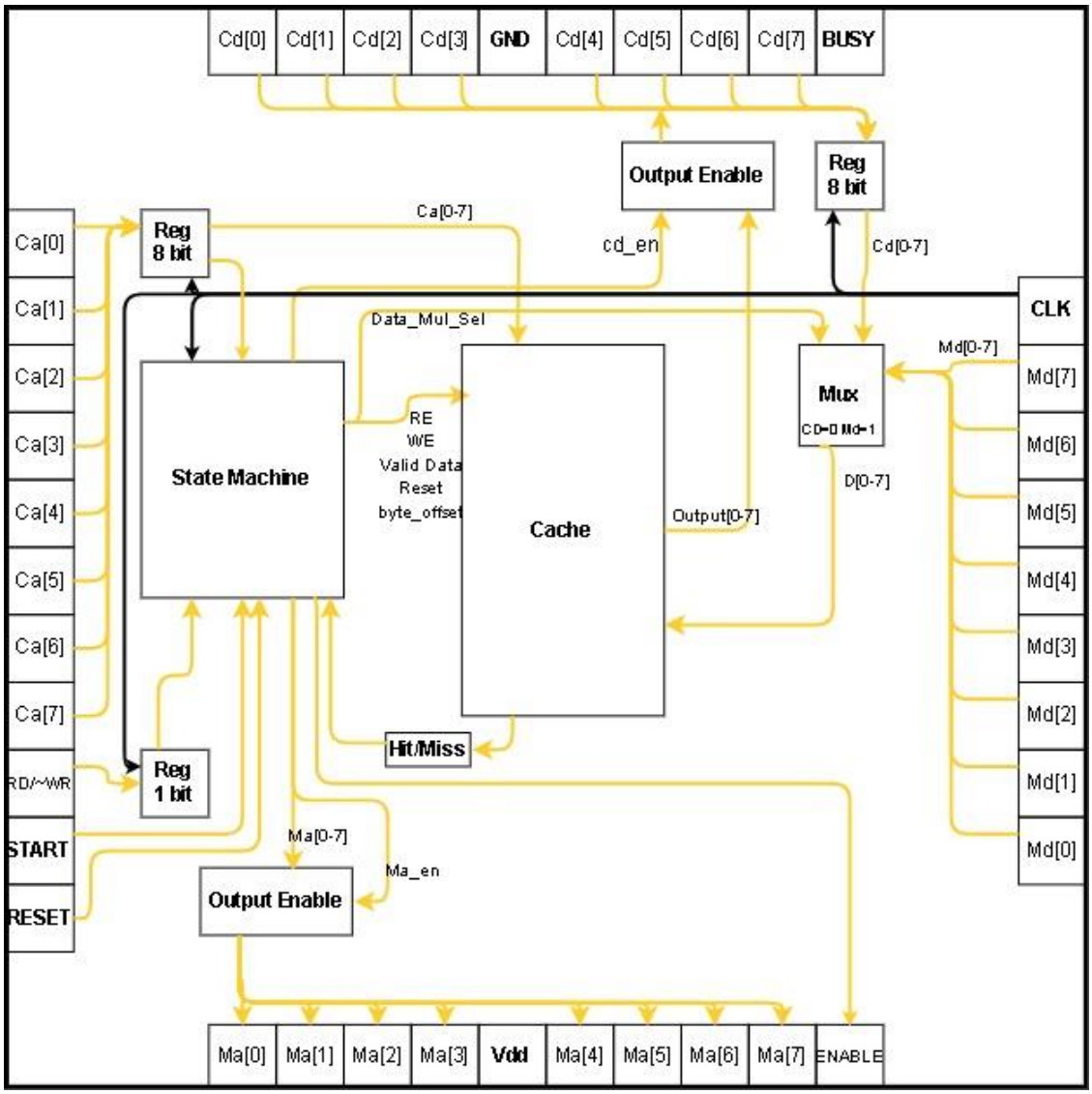
Write Hit/Miss: Similar to Read Hit/Miss, except for a write operation.

IV. DESIGN & IMPLEMENTATION

A. System Overview

The chip.vhd file contains the entire system and its implementation. The block diagram for each component is shown in the chip layout. Our approach was to make the design as simple as possible. For that we removed the use of a decoder within the chip and combined it with the cache entity within the chip. This will change the control signals will be used from the original

project description. However, we are confident that this allows for a more intuitive design.



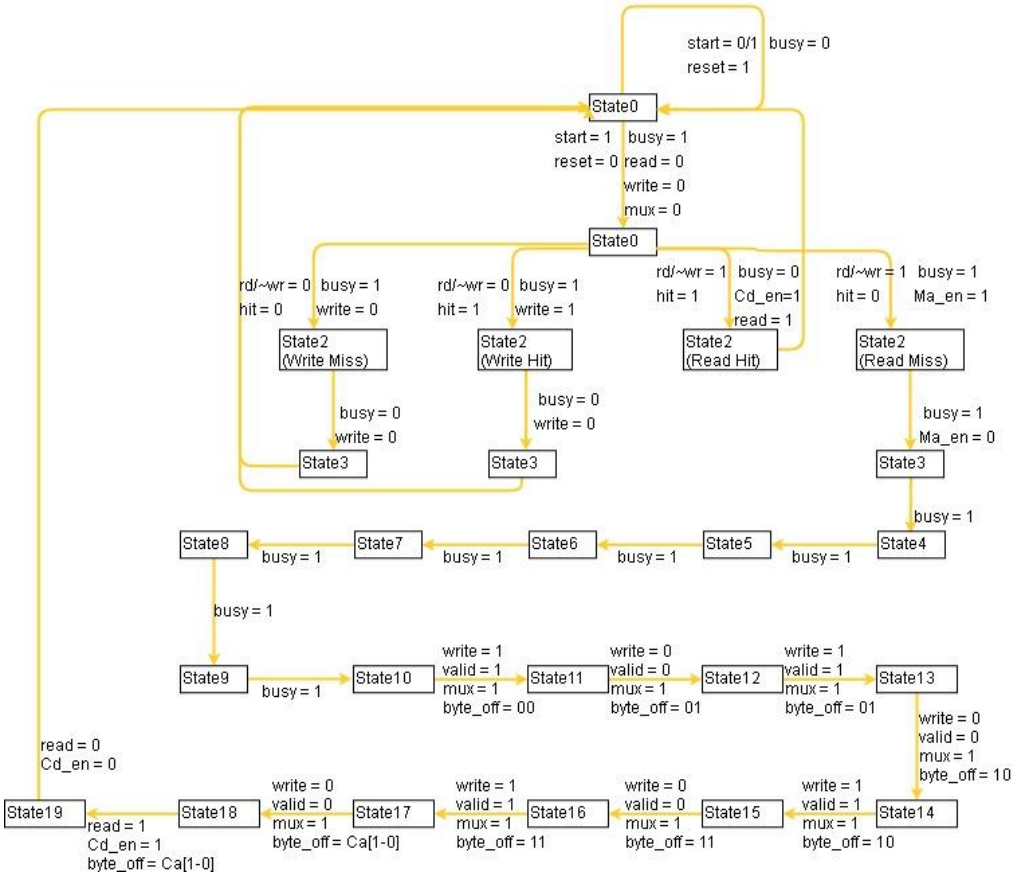
Chip: As the chip block diagram shows, a majority of the overhead for how the chip operates is handled within the state machine. This will be how which input and output control signals are used. The CPU address will be used to select the block of the cache that will be accessed, but based on the state that the chip is in, the byte offset will be determined by the state machine. Output and input enables are controlled accordingly to the modes of operation; i.e. read hit, read miss, write hit, write miss. Please see the state machine description to read more.

Included files: *chip.vhd*

B. Finite State Machine Overview

The majority of this project's logic takes place within the finite state machine (fsm). There

are a total of 20 states, starting at 0 ending at 19. The state machine is able to move between states using a 20 bit circular shift register. This allows the concept of hot states, never allowing multiple states to be on simultaneously. The inputs for the fsm include start, reset, rd/~wr, CPU address, hit/miss signal, clock. The outputs that it includes, machine address, machine address output enable (Ma_en), CPU data output enable (Cd_en), CPU address register enable (Ca_en), data mux sel (data_sel), read enable (ren), write enable (wen), valid data write (valid_data), busy signal (Busy), byte output (byte_out).



State Machine:The state machine controls all the input signal. Above is the important control signals that are relevant to each state accordingly. As you can see, state 0 is the origin for the state machine, and the default return state for each mode of the chip. Each state and identifiable mode of operation is defined and can be viewed in the figure above.

Shift Reg:: The shift register is where the current state is identified and whether the current state moves forward or stays at the current state. The shift register is a 20 bit circular shift register that when reset, will activate state 0. When enabled by the clock and the current state, the shift register will shift the current “hot” state forward. The shift register also contains a reset signal to return to state 0 early as needed.

State 0: This is known as the reset state, and is enabled when the shift register begins its count. The only way to move from this state is to have reset low and start go high. All possible state paths should return to this state.

State 1: State 1 will be handling the necessary latching of the CPU address and relevant data registers as needed. At this point, we still do not know for sure what mode we are working in until receiving the hit/miss signal. Busy is set high here to notify that the chip is currently doing work.

State 2: This state is what will decide what mode of operation we are in. Based on the response, for a read hit, we will enable the data to be read and return to state 0. Else, we will enable the relevant signals and move to state 3.

State 3: This state behaves much like the previous state in that it will return to state 0 for write miss and write hit modes of operation. Otherwise it will continue to the following state for a read miss.

States 4-9: These states are meant to hold the busy and relevant signals high as the chip waits for memory to be retrieved and placed into the cache.

States 10-16: These states handle the write enable for the cache and incrementing the byte offset.

State 17-19: Once memory has been placed into the cache, these states simply allow the read for the cache and set busy to low afterwards.

Included files: *shift_reg.vhd shift_reg20.vhd state0.vhd state1.vhd state2.vhd state3.vhd wait_state.vhd state10.vhd state11.vhd state12.vhd state13.vhd state14.vhd state15.vhd state16.vhd state17.vhd state18.vhd state19.vhd state_machine.vhd*

C. Primitive Logic Gates

In order to model the state machine transitioning in structural VHDL, we developed truth tables and formed Boolean expressions for each output. From here, we simplify each expression into primitive logic gates, such as AND, OR, NAND, NOR, etc. These logic gates allow for logic minimization as well as ease of design comprehension.

Included files: *xor2.vhd and2.vhd and3.vhd inv.vhd nand2.vhd nor2.vhd or2.vhd or3.vhd*

or4.vhd or8.vhd

D. Registers (Ca, Cd, RW)

Positive D Latch: A latch that changes the output to the input whenever the clock signal goes high.

1-bit Register: Identical to a positive D latch, except it does not output Qbar.

8-Bit Register: A set of eight 1-bit registers.

Negative D Latch: A latch that changes the output to the input whenever the clock signal goes low.

Included files: *Dlatch.vhd latch.vhd reg.vhd reg8.vhd*

E. Output Enables (Cd, Ma)

Tri-State Buffer: A buffer is a device that makes the output match the input. A tri-state buffer contains an enable signal; when enable is high, output matches input, and when enable is low, output is nothing, represented by Z, the state of high impedance.

Output Enable: A set of eight 1-bit tri-state buffers. These are used to control when to output to Cd and Ma.

Included files: *vhd tx.vhd*, *output_enable.vhd*

F. MUX (0/1, Cd/Md)

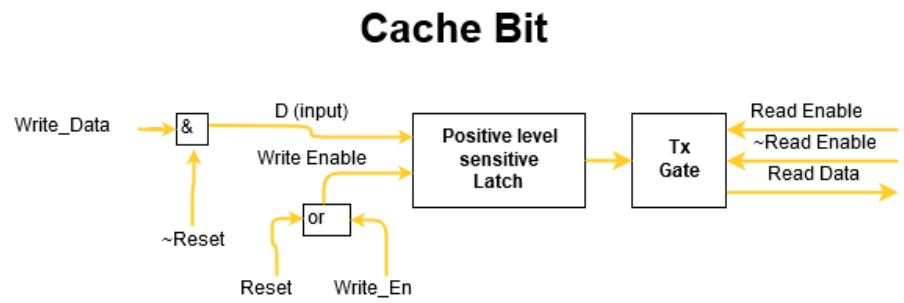
In the case of a write hit, we write Cd into the cache. In the case of a read miss, we write Md into the cache for future access. In both cases, we are writing into the cache, and we need to select which one to write with, hence the need for a 2-to-1 MUX, with select0 = Cd and select1 = Md.

Included files: *mux.vhd mux8.vhd hit_multiplex4_1.vhd multiplex4_1_test.vhd multiplex4_1.vhd multiplex8_1.vhd*

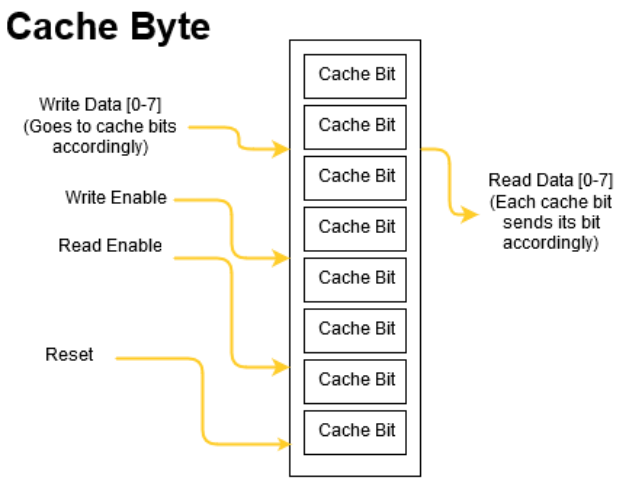
G. Cache Structure

Cache Bit: A module that takes in a reset signal, a bit write data, a write enable signal, a read

enable signal, and outputs a bit of read data. The figure below outlines the design of one cache bit, with Tx Gate being a tri-state buffer.

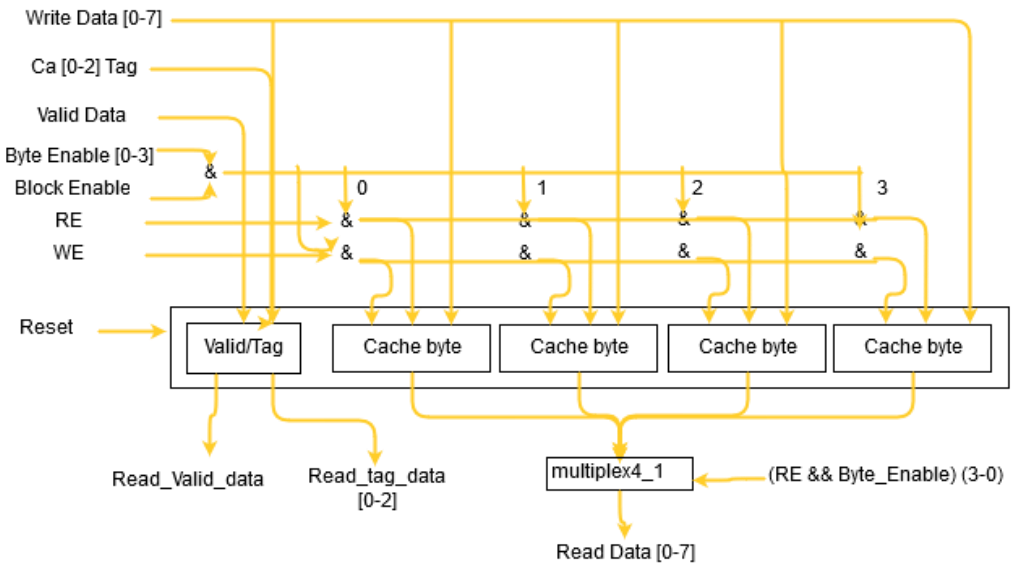


Cache Byte: A set of eight cache bits, forming a byte.



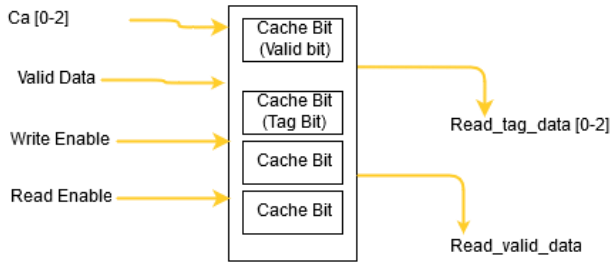
Cache Block: A set of four cache bytes, a valid bit, and a 3-bit tag.

Cache Block

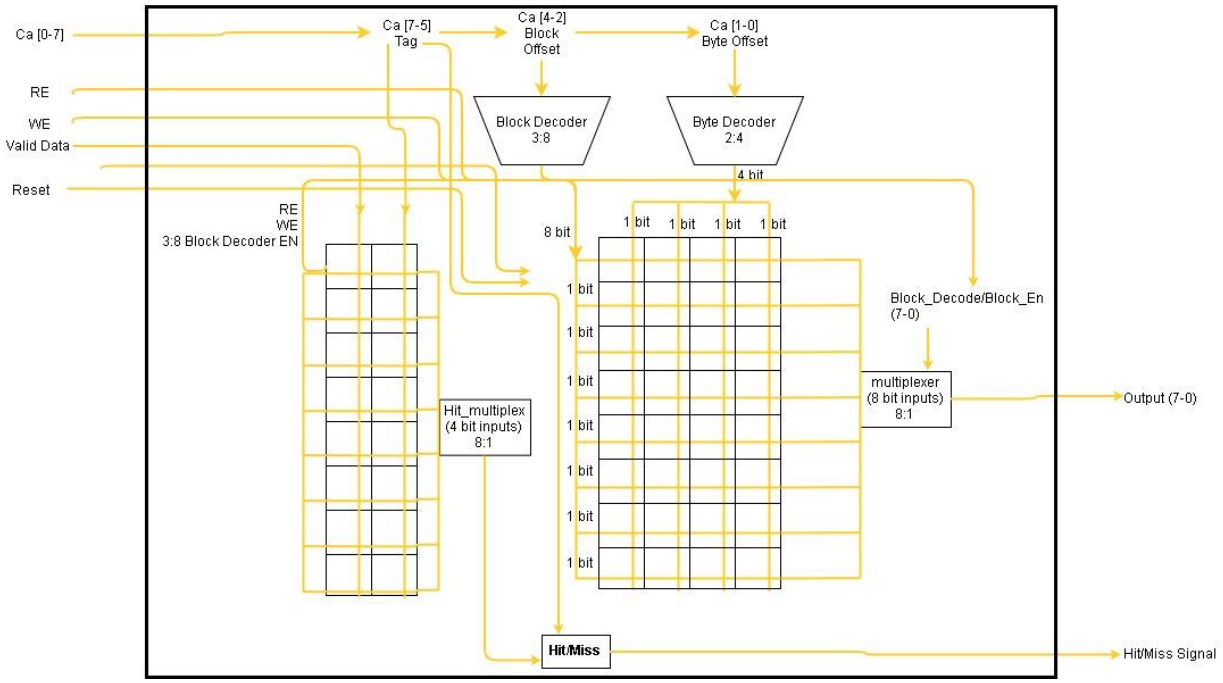


Valid/Tag: Valid bit represents whether the cache block is valid (high) or invalid (low) depending on if the three tag bits match with the tag bits found in input CPU address.

Valid/Tag 4 Bits



Cache: The designed cache is a 32-byte, direct-mapped, byte addressable cache that contains eight blocks, each block containing four bytes. To address each block, we use three bits, Ca[4:2]. To address each byte, we use two bits, Ca[1:0]. To determine validity, we use four bits; one valid bit and three tag bits for comparison. The below figure shows our approach to the cache design.



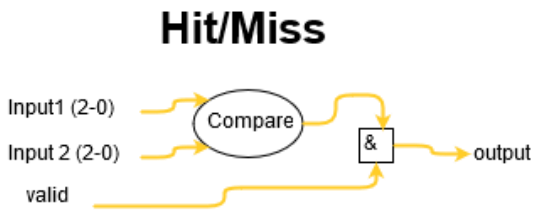
Included files: *valid_tag.vhd cache_bit.vhd cache_byte.vhd cache_block.vhd cache.vhd*

H. Cache Interface

Byte Decoder: This module is a 2-to-4 decoder. It takes in two bits, Ca[1:0], and determines which byte was selected.

Block Decoder: This module is a 3-to-8 decoder. It takes in three bits, Ca[4:2], and determines which block was selected.

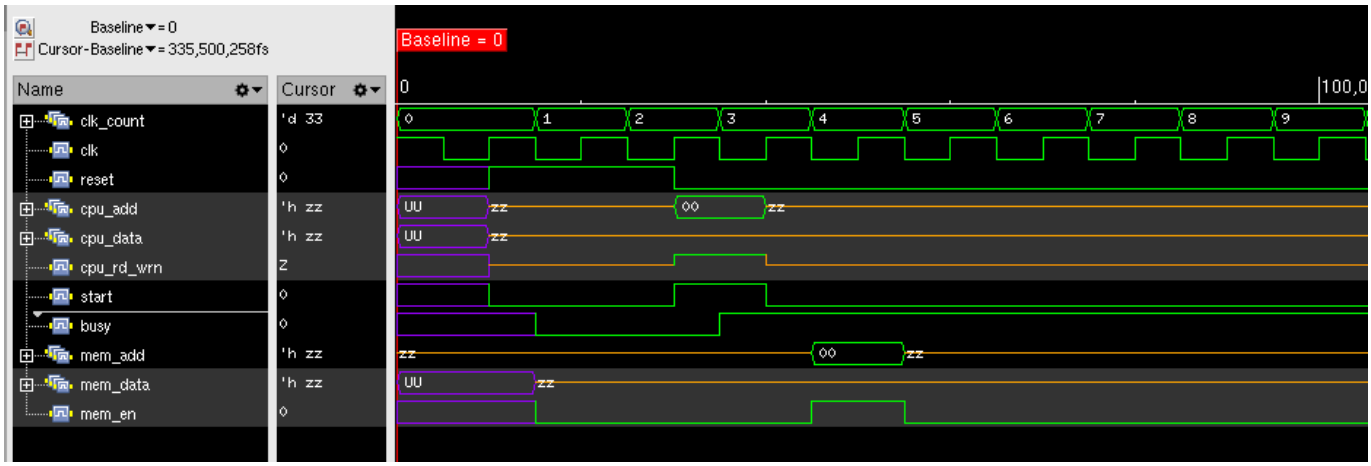
Hit/Miss: This module compares the block offset using XOR gates, and then invert the XOR outputs such that if it is matching, it outputs 1. Then, those inverted results are AND'ed together along with the valid tag. If the result is a 1, it is a miss; otherwise, it is a miss.

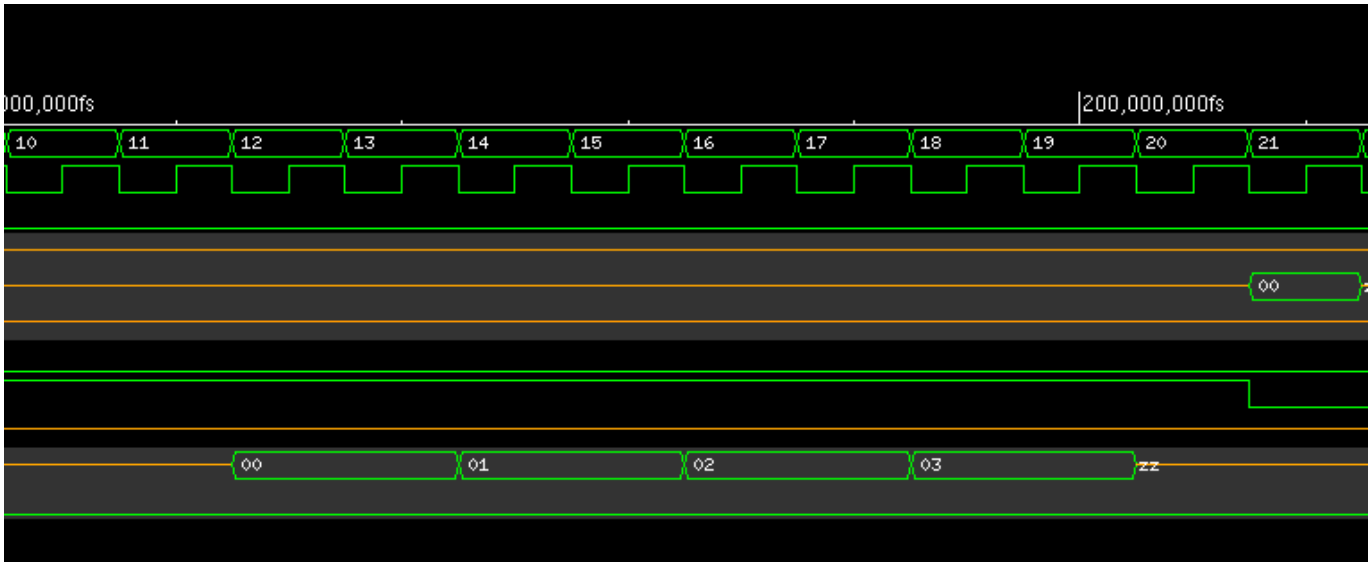


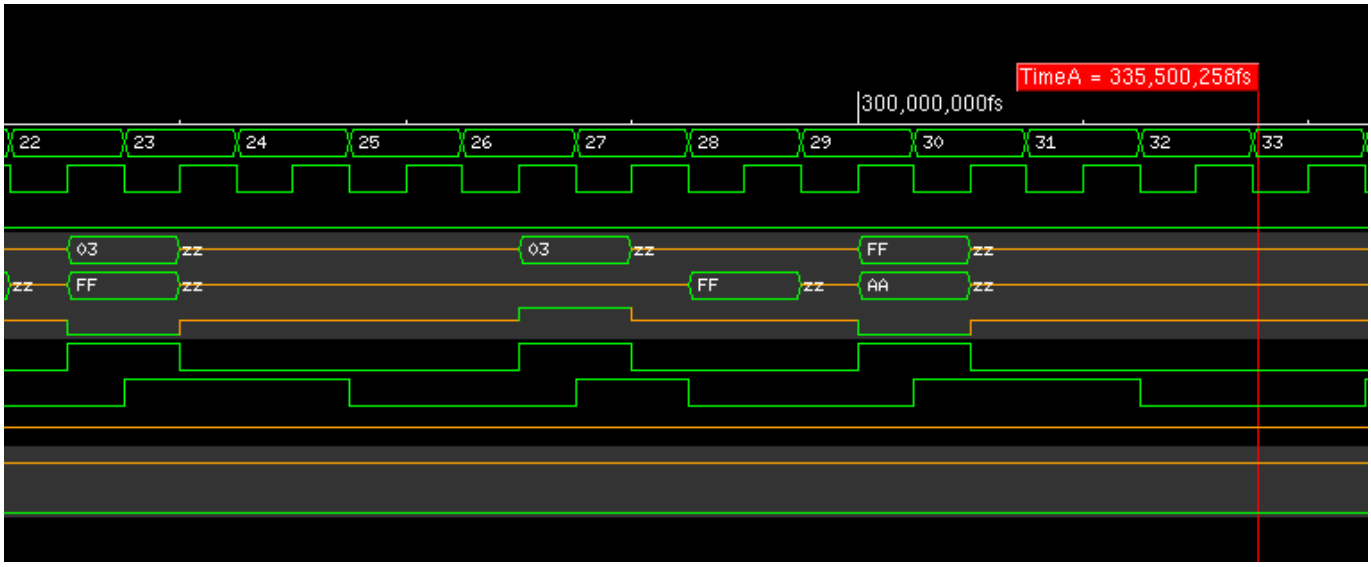
Included files: *hit_miss.vhd Byte_Decoder.vhd Block_Decoder.vhd*

V. APPENDIX

A. Waveform







B. Enscripted Code

==========

hit_miss_test.vhd

==========

```
library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity hit_miss_test is

end hit_miss_test;

architecture test of hit_miss_test is

component hit_miss
```

```
  port (
        input1 : in std_logic_vector(2 downto 0);
        input2 : in std_logic_vector(2 downto 0);
        valid  : in std_logic;
        output : out std_logic);
end component;

for hit_miss_temp : hit_miss use entity work.hit_miss(structural);

signal in1, in2 : std_logic_vector(2 downto 0);
signal v1, out1 : std_logic;

begin



        hit_miss_temp : hit_miss port map(in1, in2, v1, out1);

io_process: process

        file infile : text is in "hit_miss_in.txt";
        file outfile : text is out "hit_miss_out.txt";
        variable Input1, Input2 : std_logic_vector(2 downto 0);
        variable Valid1, Output1 : std_logic;
        variable buf : line;

begin



        while not (endfile(infile)) loop



                readline(infile, buf);
                read (buf, Input1);
                in1 <= Input1;

                readline(infile, buf);
```

```
                read(buf, Input2);
                in2 <= Input2;

                readline(infile, buf);
                read(buf, Valid1);
                v1 <= Valid1;

                wait for 10 ns;

                Output1:=out1;

                write(buf, string'(" Input 1: "));
                write(buf, in1);
                write(buf, string'(" Input 2: "));
                write(buf, in2);
                write(buf, string'(" Valid: "));
                write(buf, v1);
                writeline(outfile,buf);

                write(buf, string'(" Output: "));
                write(buf, Output1);
                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
Dlatch_test.vhd
=====================================


library IEEE;
library std;
```

```
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity Dlatch_test is

end Dlatch_test;

architecture test of Dlatch_test is

component Dlatch
  port ( d   : in  std_logic;
         clk : in  std_logic;
         q   : out std_logic;
         qbar: out std_logic);
end component;

for Dlatch_temp : Dlatch use entity work.Dlatch(structural);

signal d_1, clk_1, q_1, q_bar_1 : std_logic;

begin


          Dlatch_temp : Dlatch port map(d_1, clk_1, q_1, q_bar_1);

io_process: process

          file infile : text is in "Dlatch_in.txt";
          file outfile : text is out "Dlatch_out.txt";
          variable Input1 : std_logic_vector(1 downto 0);
          variable q_2, q_bar_2 : std_logic;
          variable buf : line;

begin
```

```
        while not (endfile(infile)) loop


                readline(infile, buf);
                read (buf, Input1);
                d_1 <= Input1(0);
                clk_1 <= Input1(1);

                wait for 10 ns;

                q_2:=q_1;
                q_bar_2:=q_bar_1;

                write(buf, string'(" D: "));
                write(buf, d_1);
                write(buf, string'(" CLK: "));
                write(buf, clk_1);
                writeline(outfile, buf);

                write(buf, string'(" Q: "));
                write(buf, q_2);
                write(buf, string'(" ~Q: "));
                write(buf, q_bar_2);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
cache_bit_test.vhd
```

```
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity cache_bit_test is

end cache_bit_test;

architecture test of cache_bit_test is

component cache_bit
  port (
    Reset      :  in std_logic;
    Write_data :  in std_logic;
    Write_en   :  in std_logic;
    Read_en    :  in std_logic;
    Read_data  :  out std_logic);
end component;

for cache_bit_temp : cache_bit use entity work.cache_bit(structural);

signal reset, wen, wdata, ren, rdata : std_logic;

begin


        cache_bit_temp : cache_bit port map(reset, wdata, wen, ren, rdata);

io_process: process
```

```
        file infile : text is in "cache_bit_in.txt";
        file outfile : text is out "cache_bit_out.txt";
        variable Input1 : std_logic_vector(3 downto 0);
        --variable Valid1, Output1 : std_logic;
        variable buf : line;

begin


        while not (endfile(infile)) loop


                readline(infile, buf);
                read (buf, Input1);
                reset <= Input1(3);
                wdata <= Input1(2);
                wen <= Input1(1);
                ren <= Input1(0);

                wait for 10 ns;


                write(buf, string'(" Reset: "));
                write(buf, reset);
                write(buf, string'(" Write Data: "));
                write(buf, wdata);
                write(buf, string'(" Write Enable: "));
                write(buf, wen);
                write(buf, string'(" Read Enable: "));
                write(buf, ren);
                writeline(outfile, buf);

                write(buf, string'(" Read Data: "));
                write(buf, rdata);

                writeline(outfile,buf);
```

```
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
cache_byte_test.vhd
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity cache_byte_test is

end cache_byte_test;

architecture test of cache_byte_test is

component cache_byte
  port (
        Reset      :  in std_logic;
    Write_data :  in std_logic_vector(7 downto 0);
    Write_en   :  in std_logic;
    Read_en    :  in std_logic;
    Read_data  :  out std_logic_vector(7 downto 0));
end component;

for cache_byte_temp : cache_byte use entity work.cache_byte(structural);
```

```
signal reset, wen, ren : std_logic;
signal wdata, rdata : std_logic_vector(7 downto 0);

begin


        cache_byte_temp : cache_byte port map(reset, wdata, wen, ren, rdata);

io_process: process

        file infile : text is in "cache_byte_in.txt";
        file outfile : text is out "cache_byte_out.txt";
        variable Input1 : std_logic_vector(2 downto 0);
        variable Input2 : std_logic_vector(7 downto 0);
        variable buf : line;

begin


        while not (endfile(infile)) loop


                readline(infile, buf);
                read (buf, Input1);
                reset <= Input1(2);
                wen <= Input1(1);
                ren <= Input1(0);

                readline(infile, buf);
                read (buf, Input2);
                wdata <= Input2;


                wait for 10 ns;
```

```
                write(buf, string'(" Reset: "));
                write(buf, reset);
                write(buf, string'(" Write Data: "));
                write(buf, wdata);
                write(buf, string'(" Write Endable: "));
                write(buf, wen);
                write(buf, string'(" Read Enable: "));
                write(buf, ren);
                writeline(outfile, buf);

                write(buf, string'(" Read Data: "));
                write(buf, rdata);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
valid_tag_test.vhd
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity valid_tag_test is

end valid_tag_test;
```

```
architecture test of valid_tag_test is

component Valid_tag
  port (
    Reset      :  in std_logic;
    Write_data :  in std_logic_vector(2 downto 0);
    Valid_data :  in std_logic;
    Write_en   :  in std_logic;
    Read_en    :  in std_logic;
    Read_valid_data : out std_logic;
    Read_tag_data  :  out std_logic_vector(2 downto 0));
end component;

for valid_tag_temp : valid_tag use entity work.valid_tag(structural);

signal reset, wen, ren, wvdata, rvdata : std_logic;
signal wtdata, rtdata : std_logic_vector(2 downto 0); --write/read tag data

begin


        valid_tag_temp : valid_tag port map(reset, wtdata, wvdata, wen, ren, rvdata, rtdata);

io_process: process

        file infile : text is in "valid_tag_in.txt";
        file outfile : text is out "valid_tag_out.txt";
        variable Input1 : std_logic_vector(2 downto 0);
        variable Input2 : std_logic;
        variable buf : line;

begin


        while not (endfile(infile)) loop
```

```
readline(infile, buf);
read (buf, Input1);
reset <= Input1(2);
wen <= Input1(1);
ren <= Input1(0);

readline(infile, buf);
read(buf, Input1);
wtdata <= Input1;

readline(infile, buf);
read (buf, Input2);
wvdata <= Input2;


wait for 10 ns;


write(buf, string'(" Reset: "));
write(buf, reset);
write(buf, string'(" Write Endable: "));
write(buf, wen);
write(buf, string'(" Read Enable: "));
write(buf, ren);
write(buf, string'(" Write Tag Data: "));
write(buf, wtdata);
write(buf, string'(" Write Valid Data: "));
write(buf, wvdata);
writeline(outfile, buf);

write(buf, string'(" Read Valid Data: "));
write(buf, rvdata);
write(buf, string'(" Read Tag Data: "));
write(buf, rtdata);
```

```
                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;====================================
multiplex4_1_test.vhd
====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity multiplex4_1_test is

end multiplex4_1_test;

architecture test of multiplex4_1_test is

component multiplex4_1
        port (
                input0 :  in std_logic_vector(7 downto 0);
        input1 :  in std_logic_vector(7 downto 0);
        input2 :  in std_logic_vector(7 downto 0);
        input3 :  in std_logic_vector(7 downto 0);
                en     :  in std_logic_vector(3 downto 0);
        output :  out std_logic_vector(7 downto 0)
                );
```

```
end component;

for multiplex4_1_temp : multiplex4_1 use entity work.multiplex4_1(structural);

signal in0, in1, in2, in3, out0 : std_logic_vector(7 downto 0);
signal en : std_logic_vector(3 downto 0);

begin


        multiplex4_1_temp : multiplex4_1 port map(in0, in1, in2, in3, en, out0);

io_process: process

        file infile : text is in "multiplex4_1_in.txt";
        file outfile : text is out "multiplex4_1_out.txt";
        variable Input1 : std_logic_vector(7 downto 0);
        variable Input2 : std_logic_vector(3 downto 0);
        variable buf : line;

begin

        readline(infile, buf);
        read (buf, Input1);
        in0 <= Input1;
        readline(infile, buf);
        read (buf, Input1);
        in1 <= Input1;
        readline(infile, buf);
        read (buf, Input1);
        in2 <= Input1;
        readline(infile, buf);
        read (buf, Input1);
        in3 <= Input1;
```

```
        while not (endfile(infile)) loop


                readline(infile, buf);
                read (buf, Input2);
                en <= Input2;

                wait for 10 ns;


                write(buf, string'(" In0: "));
                write(buf, in0);
                write(buf, string'(" In0: "));
                write(buf, in1);
                write(buf, string'(" In0: "));
                write(buf, in2);
                write(buf, string'(" In0: "));
                write(buf, in3);
                writeline(outfile, buf);

                write(buf, string'(" En: "));
                write(buf, en);
                write(buf, string'(" Output: "));
                write(buf, out0);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
cache_block_test.vhd
```

```
====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity cache_block_test is

end cache_block_test;

architecture test of cache_block_test is

component cache_block
  port (
          Reset      :  in std_logic;
    Write_data :  in std_logic_vector(7 downto 0);
    Ca_tag     :  in std_logic_vector(2 downto 0);
    Valid_data :  in std_logic;
    Byte_Enable:  in std_logic_vector(3 downto 0);
    Block_Enable: in std_logic;
    Write_en   :  in std_logic;
    Read_en    :  in std_logic;
    Read_valid_data : out std_logic;
    Read_tag_data  :  out std_logic_vector(2 downto 0);
    Read_data  :  out std_logic_vector(7 downto 0));
end component;


for cache_block_temp : cache_block use entity work.cache_block(structural);

-- Inputs
signal reset, wen, ren : std_logic;
```

```
signal wvdata, blocken : std_logic;
signal wdata  : std_logic_vector(7 downto 0);
signal wtdata : std_logic_vector(2 downto 0);
signal byteen : std_logic_vector(3 downto 0);
-- Outputs
signal rtdata : std_logic_vector(2 downto 0);
signal rvdata : std_logic;
signal rdata  : std_logic_vector(7 downto 0);

begin


        cache_block_temp : cache_block port map(reset, wdata, wtdata, wvdata, byteen, blocken, wen,
ren, rvdata, rtdata, rdata);

io_process: process

        file infile : text is in "cache_block_in.txt";
        file outfile : text is out "cache_block_out.txt";
        variable Input1 : std_logic_vector(3 downto 0);
        variable Input2 : std_logic_vector(7 downto 0);
        variable Input3 : std_logic_vector(2 downto 0);
        variable Input4 : std_logic;
        --variable temp : std_logic_vector(19 downto 0);

        variable buf : line;

begin


        while not (endfile(infile)) loop


                readline(infile, buf);
                read (buf, Input1);
                reset <= Input1(3);
```

```
wen <= Input1(2);
ren <= Input1(1);
blocken <= Input1(0);

readline(infile, buf);
read (buf, Input1);
byteen <= Input1;

readline(infile, buf);
read (buf, Input2);
wdata <= Input2;

readline(infile, buf);
read (buf, Input3);
wtdata <= Input3;

readline(infile, buf);
read (buf, Input4);
wvdata <= Input4;

wait for 20 ns;

--wen <= '0';

--wait for 40 ns;


write(buf, string'(" Reset: "));
write(buf, reset);
write(buf, string'(" Write Enable: "));
write(buf, wen);
write(buf, string'(" Read Enable: "));
write(buf, ren);
write(buf, string'(" Block Enable: "));
write(buf, blocken);
writeline(outfile, buf);
```

```
                write(buf, string'(" Byte Enable: "));
                write(buf, byteen);
                write(buf, string'(" Write Data: "));
                write(buf, wdata);
                write(buf, string'(" Write Tag Data: "));
                write(buf, wtdata);
                write(buf, string'(" Write Valid Data: "));
                write(buf, wvdata);
                writeline(outfile, buf);

                write(buf, string'(" Read Data: "));
                write(buf, rdata);
                write(buf, string'(" Read Tag Data: "));
                write(buf, rtdata);
                write(buf, string'(" Read Valid Data: "));
                write(buf, rvdata);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
Byte_Decoder_test.vhd
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
```

```
use std.env.all;

entity Byte_Decoder_test is

end Byte_Decoder_test;

architecture test of Byte_Decoder_test is

component Byte_Decoder
  port (
    Byte_Offset  : in std_logic_vector(1 downto 0);
        Output                  : out std_logic_vector(3 downto 0));
end component;


for Byte_Decoder_temp : Byte_Decoder use entity work.Byte_Decoder(structural);

signal offset : std_logic_vector(1 downto 0);
signal out1   : std_logic_vector(3 downto 0);

begin


        Byte_Decoder_temp : Byte_Decoder port map(offset, out1);

io_process: process

        file infile : text is in "Byte_Decoder_in.txt";
        file outfile : text is out "Byte_Decoder_out.txt";
        variable Input1 : std_logic_vector(1 downto 0);
        variable buf : line;

begin


        while not (endfile(infile)) loop
```

```
                readline(infile, buf);
                read (buf, Input1);
                offset <= Input1;

                wait for 10 ns;


                write(buf, string'(" Offset: "));
                write(buf, offset);
                writeline(outfile, buf);

                write(buf, string'(" Output: "));
                write(buf, out1);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
Block_Decoder_test.vhd
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;
```

```
entity Block_Decoder_test is

end Block_Decoder_test;

architecture test of Block_Decoder_test is

component Block_Decoder
  port (
    Block_Offset : in std_logic_vector(2 downto 0);
        Output                  : out std_logic_vector(7 downto 0));
end component;


for Block_Decoder_temp : Block_Decoder use entity work.Block_Decoder(structural);

signal offset : std_logic_vector(2 downto 0);
signal out1   : std_logic_vector(7 downto 0);

begin


        Block_Decoder_temp : Block_Decoder port map(offset, out1);

io_process: process

        file infile : text is in "Block_Decoder_in.txt";
        file outfile : text is out "Block_Decoder_out.txt";
        variable Input1 : std_logic_vector(2 downto 0);
        variable buf : line;

begin


        while not (endfile(infile)) loop

                readline(infile, buf);
```

```
                read (buf, Input1);
                offset <= Input1;

                wait for 10 ns;


                write(buf, string'(" Offset: "));
                write(buf, offset);
                writeline(outfile, buf);

                write(buf, string'(" Output: "));
                write(buf, out1);

                writeline(outfile,buf);
                writeline(outfile,buf);

        end loop;
        stop(0);

end process io_process;

end test;=====================================
shift_reg20_test.vhd
=====================================


library IEEE;
library std;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
use std.env.all;

entity shift_reg20_test is

end shift_reg20_test;
```

```
architecture test of shift_reg20_test is

component shift_reg20
	port (
		reset	:  in std_logic;
		clk		:  in std_logic;
		q		:  out std_logic_vector(19 downto 0);
		qbar	:  out std_logic_vector(19 downto 0));
end component;

for shift_reg20_temp : shift_reg20 use entity work.shift_reg20(structural);

signal q, qbar : std_logic_vector(19 downto 0);
signal reset, clk : std_logic;

begin


	shift_reg20_temp : shift_reg20 port map(reset, clk, q, qbar);

io_process: process

	file infile : text is in "shift_reg20_in.txt";
	file outfile : text is out "shift_reg20_out.txt";
	variable Input1 : std_logic_vector(1 downto 0);
	variable buf : line;

begin


	while not (endfile(infile)) loop


		readline(infile, buf);
		read (buf, Input1);
```

```
            clk <= Input1(1);
            reset <= Input1(0);

            wait for 10 ns;

            write(buf, string'(" Reset: "));
            write(buf, reset);
            write(buf, string'(" CLK: "));
            write(buf, clk);
            writeline(outfile,buf);

            write(buf, string'(" Q: "));
            write(buf, q);
            write(buf, string'(" ~Q: "));
            write(buf, qbar);
            writeline(outfile,buf);
            writeline(outfile,buf);




        end loop;
        stop(0);

end process io_process;

end test;
```

C. Testbench Results

Byte Decoder

Offset: 000
Output: 00000001

Offset: 001
Output: 00000010

Offset: 010
Output: 00000100

Offset: 011
Output: 00001000

Offset: 100
Output: 00010000

Offset: 101
Output: 00100000

Offset: 110
Output: 01000000

Offset: 111
Output: 10000000

Block Decoder

Offset: 00
Output: 0001

Offset: 01
Output: 0010

Offset: 10
Output: 0100

Offset: 11
Output: 1000

Cache Bit

Reset: 0 Write Data: 1 Write Enable: 0 Read Enable: 0
Read Data: Z

Reset: 0 Write Data: 1 Write Enable: 1 Read Enable: 0
Read Data: Z

Reset: 0 Write Data: 1 Write Enable: 0 Read Enable: 0
Read Data: Z

Reset: 0 Write Data: 1 Write Enable: 0 Read Enable: 1
Read Data: 1

Reset: 0 Write Data: 0 Write Enable: 1 Read Enable: 1
Read Data: 0

Reset: 1 Write Data: 0 Write Enable: 0 Read Enable: 0
Read Data: Z

Reset: 0 Write Data: 0 Write Enable: 0 Read Enable: 1
Read Data: 0

CACHE_BLOCK

Reset: 0 Write Enable: 0 Read Enable: 0 Block Enable: 1
Byte Enable: 0001 Write Data: 00000000 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: UUU Read Valid Data: U

Reset: 0 Write Enable: 1 Read Enable: 0 Block Enable: 1
Byte Enable: 0001 Write Data: 11111111 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 0 Block Enable: 1
Byte Enable: 0010 Write Data: 11111111 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 1 Read Enable: 0 Block Enable: 1
Byte Enable: 0010 Write Data: 11110000 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 0 Block Enable: 1
Byte Enable: 0100 Write Data: 11110000 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 1 Read Enable: 0 Block Enable: 1
Byte Enable: 0100 Write Data: 00001111 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 0 Block Enable: 1
Byte Enable: 1000 Write Data: 00001111 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 1 Read Enable: 0 Block Enable: 1
Byte Enable: 1000 Write Data: 11001100 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 0 Block Enable: 1
Byte Enable: 0001 Write Data: 11001100 Write Tag Data: 101 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 1 Block Enable: 1

Byte Enable: 0001 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 11111111 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 1 Block Enable: 1
Byte Enable: 0010 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 11110000 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 1 Block Enable: 1
Byte Enable: 0100 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 00001111 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 1 Block Enable: 1
Byte Enable: 1000 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 11001100 Read Tag Data: 101 Read Valid Data: 1

Reset: 0 Write Enable: 0 Read Enable: 1 Block Enable: 0
Byte Enable: 1000 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 101 Read Valid Data: 1

Reset: 1 Write Enable: 0 Read Enable: 1 Block Enable: 1
Byte Enable: 1000 Write Data: 10101010 Write Tag Data: 010 Write Valid Data: 1
Read Data: 00000000 Read Tag Data: 000 Read Valid Data: 0

CACHE_BYTE

Reset: 0 Write Data: 11100010 Write Endable: 0 Read Enable: 0
Read Data: ZZZZZZZZ

Reset: 0 Write Data: 11100010 Write Endable: 1 Read Enable: 0
Read Data: ZZZZZZZZ

Reset: 0 Write Data: 11100010 Write Endable: 0 Read Enable: 0
Read Data: ZZZZZZZZ

Reset: 0 Write Data: 11100010 Write Endable: 0 Read Enable: 1
Read Data: 11100010

Reset: 0 Write Data: 00011101 Write Endable: 1 Read Enable: 1
Read Data: 00011101

Reset: 1 Write Data: 11111111 Write Endable: 0 Read Enable: 0
Read Data: ZZZZZZZZ

Reset: 1 Write Data: 11111111 Write Endable: 0 Read Enable: 1
Read Data: 00000000

Reset: 1 Write Data: 11111111 Write Endable: 1 Read Enable: 1
Read Data: 00000000

Reset: 0 Write Data: 11111111 Write Endable: 0 Read Enable: 1
Read Data: 00000000

DLATCH

D: 0 CLK: 1
Q: 0 ~Q: 1

D: 1 CLK: 0
Q: 0 ~Q: 1

D: 1 CLK: 1
Q: 1 ~Q: 0

D: 0 CLK: 0
Q: 1 ~Q: 0

HIT_MISS_OUT

Input 1: 000 Input 2: 000 Valid: 1
Output: 1

Input 1: 111 Input 2: 111 Valid: 0
Output: 0

Input 1: 001 Input 2: 000 Valid: 1
Output: 0

Input 1: 010 Input 2: 010 Valid: 1
Output: 1

Input 1: 001 Input 2: 100 Valid: 0
Output: 0

Input 1: 001 Input 2: 100 Valid: 1
Output: 0

Input 1: 110 Input 2: 101 Valid: 1
Output: 0

Input 1: 110 Input 2: 101 Valid: 1
Output: 0

MULTIPLEX4_1

In0: 11111111 In0: 11110000 In0: 00001111 In0: 11001100
En: 0000 Output: 00000000

In0: 11111111 In0: 11110000 In0: 00001111 In0: 11001100
En: 0001 Output: 11111111

In0: 11111111 In0: 11110000 In0: 00001111 In0: 11001100
En: 0010 Output: 11110000

In0: 11111111 In0: 11110000 In0: 00001111 In0: 11001100
En: 0100 Output: 00001111

In0: 11111111 In0: 11110000 In0: 00001111 In0: 11001100
En: 1000 Output: 11001100

SHIFT_REG20_OUT

Reset: 0 CLK: 1
Q: UUUUUUUUUUUUUUUUUUUU ~Q: UUUUUUUUUUUUUUUUUUUU

Reset: 0 CLK: 0
Q: UUUUUUUUUUUUUUUUUUUU ~Q: UUUUUUUUUUUUUUUUUUUU

Reset: 1 CLK: 1
Q: UUUUUUUUUUUUUUUUUUUU ~Q: UUUUUUUUUUUUUUUUUUUU

Reset: 1 CLK: 0
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 1 CLK: 1
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 1 CLK: 0
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 1
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 0
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 0 CLK: 1
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 1 CLK: 0
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 1 CLK: 1
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 0 CLK: 0
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 1
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 0
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 0 CLK: 1
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 0 CLK: 0
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 0 CLK: 1
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 0 CLK: 0
Q: 0000000000000001000 ~Q: 1111111111111110111

Reset: 0 CLK: 1
Q: 0000000000000001000 ~Q: 1111111111111110111

Reset: 0 CLK: 0
Q: 0000000000000010000 ~Q: 1111111111111101111

Reset: 0 CLK: 1
Q: 0000000000000010000 ~Q: 1111111111111101111

Reset: 0 CLK: 0
Q: 0000000000000100000 ~Q: 1111111111111011111

Reset: 0 CLK: 1
Q: 0000000000000100000 ~Q: 1111111111111011111

Reset: 0 CLK: 0
Q: 0000000000001000000 ~Q: 1111111111110111111

Reset: 0 CLK: 1
Q: 0000000000001000000 ~Q: 1111111111110111111

Reset: 0 CLK: 0
Q: 0000000000010000000 ~Q: 1111111111101111111

Reset: 0 CLK: 1
Q: 0000000000010000000 ~Q: 1111111111101111111

Reset: 0 CLK: 0
Q: 0000000000100000000 ~Q: 1111111111011111111

Reset: 0 CLK: 1
Q: 00000000000100000000 ~Q: 11111111111011111111

Reset: 0 CLK: 0
Q: 00000000001000000000 ~Q: 11111111110111111111

Reset: 0 CLK: 1
Q: 00000000001000000000 ~Q: 11111111110111111111

Reset: 0 CLK: 0
Q: 00000000010000000000 ~Q: 11111111101111111111

Reset: 0 CLK: 1
Q: 00000000010000000000 ~Q: 11111111101111111111

Reset: 0 CLK: 0
Q: 00000000100000000000 ~Q: 11111111011111111111

Reset: 0 CLK: 1
Q: 00000000100000000000 ~Q: 11111111011111111111

Reset: 0 CLK: 0
Q: 00000001000000000000 ~Q: 11111110111111111111

Reset: 0 CLK: 1
Q: 00000001000000000000 ~Q: 11111110111111111111

Reset: 0 CLK: 0
Q: 00000010000000000000 ~Q: 11111101111111111111

Reset: 0 CLK: 1
Q: 00000010000000000000 ~Q: 11111101111111111111

Reset: 0 CLK: 0
Q: 00000100000000000000 ~Q: 11111011111111111111

Reset: 0 CLK: 1
Q: 00000100000000000000 ~Q: 11111011111111111111

Reset: 0 CLK: 0
Q: 00001000000000000000 ~Q: 11110111111111111111

Reset: 0 CLK: 1
Q: 00001000000000000000 ~Q: 11110111111111111111

Reset: 0 CLK: 0
Q: 00010000000000000000 ~Q: 11101111111111111111

Reset: 0 CLK: 1
Q: 00010000000000000000 ~Q: 11101111111111111111

Reset: 0 CLK: 0
Q: 00100000000000000000 ~Q: 11011111111111111111

Reset: 0 CLK: 1
Q: 00100000000000000000 ~Q: 11011111111111111111

Reset: 0 CLK: 0
Q: 01000000000000000000 ~Q: 10111111111111111111

Reset: 0 CLK: 1
Q: 01000000000000000000 ~Q: 10111111111111111111

Reset: 0 CLK: 0
Q: 10000000000000000000 ~Q: 01111111111111111111

Reset: 0 CLK: 1
Q: 10000000000000000000 ~Q: 01111111111111111111

Reset: 0 CLK: 0
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 1
Q: 00000000000000000001 ~Q: 11111111111111111110

Reset: 0 CLK: 0
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 0 CLK: 1
Q: 00000000000000000010 ~Q: 11111111111111111101

Reset: 0 CLK: 0
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 0 CLK: 1
Q: 00000000000000000100 ~Q: 11111111111111111011

Reset: 0 CLK: 0
Q: 00000000000000001000 ~Q: 11111111111111110111

Reset: 0 CLK: 1
Q: 00000000000000001000 ~Q: 11111111111111110111

Reset: 0 CLK: 0
Q: 00000000000000010000 ~Q: 11111111111111101111

Reset: 0 CLK: 1
Q: 00000000000000010000 ~Q: 11111111111111101111

Reset: 0 CLK: 0

Q: 0000000000000100000 ~Q: 1111111111111011111

Reset: 0 CLK: 1

Q: 0000000000000100000 ~Q: 1111111111111011111

Reset: 0 CLK: 0

Q: 0000000000001000000 ~Q: 1111111111110111111

Reset: 0 CLK: 1

Q: 0000000000001000000 ~Q: 1111111111110111111

Reset: 0 CLK: 0

Q: 0000000000010000000 ~Q: 1111111111101111111

Reset: 0 CLK: 1

Q: 0000000000010000000 ~Q: 1111111111101111111

Reset: 0 CLK: 0

Q: 0000000000100000000 ~Q: 1111111111011111111

Reset: 0 CLK: 1

Q: 0000000000100000000 ~Q: 1111111111011111111

Reset: 0 CLK: 0

Q: 0000000001000000000 ~Q: 1111111110111111111

Reset: 0 CLK: 1

Q: 0000000001000000000 ~Q: 1111111110111111111

Reset: 0 CLK: 0

Q: 0000000010000000000 ~Q: 1111111101111111111

VALID_TAG

Reset: 0 Write Endable: 0 Read Enable: 0 Write Tag Data: 111 Write Valid Data: 0
Read Valid Data: U Read Tag Data: UUU

Reset: 0 Write Endable: 1 Read Enable: 0 Write Tag Data: 111 Write Valid Data: 0
Read Valid Data: 0 Read Tag Data: 111

Reset: 0 Write Endable: 0 Read Enable: 0 Write Tag Data: 111 Write Valid Data: 0
Read Valid Data: 0 Read Tag Data: 111

Reset: 0 Write Endable: 0 Read Enable: 1 Write Tag Data: 101 Write Valid Data: 1
Read Valid Data: 0 Read Tag Data: 111

Reset: 0 Write Endable: 1 Read Enable: 1 Write Tag Data: 101 Write Valid Data: 1
Read Valid Data: 1 Read Tag Data: 101

Reset: 1 Write Endable: 0 Read Enable: 0 Write Tag Data: 100 Write Valid Data: 1
Read Valid Data: 0 Read Tag Data: 000

Reset: 1 Write Endable: 0 Read Enable: 1 Write Tag Data: 100 Write Valid Data: 1
Read Valid Data: 0 Read Tag Data: 000

Reset: 1 Write Endable: 1 Read Enable: 1 Write Tag Data: 100 Write Valid Data: 1
Read Valid Data: 0 Read Tag Data: 000

Reset: 0 Write Endable: 0 Read Enable: 1 Write Tag Data: 100 Write Valid Data: 1
Read Valid Data: 0 Read Tag Data: 000