




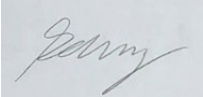
**Department of Electrical
& Computer Engineering**
Faculty of Engineering & Architectural Science

Course Title:	Digital Systems Engineering
Course Number:	COE758
Semester/Year (e.g.F2016)	F2019

Instructor:	Lev Kirischian
--------------------	-----------------------

<i>Assignment/Lab Number:</i>	Project 1
<i>Assignment/Lab Title:</i>	Cache Project

<i>Submission Date:</i>	Nov. 3 rd , 2019
<i>Due Date:</i>	Nov. 3 rd , 2019

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Chen	Jie	500715242	2	
Zou	Yun Peng	500628304	5	

Reset Form

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: <http://www.ryerson.ca/senate/current/pol60.pdf>

Table of Contents

1.0 - Abstract.....	2
2.0 - Introduction.....	2
3.0 - Specifications.....	2
4.0 - Device Description.....	4
4.1 - Symbols.....	5
4.2 - Block Diagram.....	6
4.3 - State Diagram.....	6
4.4 - Process Diagram.....	7
5.0 - Results.....	9
6.0 - References.....	10
7.0 - Appendix.....	10
Cache Controller VHDL.....	10
SDRAM Controller VHDL.....	16

1.0 - Abstract

The objective of this laboratory project is to learn the functionalities of a cache controller and how does it work with the block memory and SDRAM-controllers, as well as its behaviour and time performance. The implementations and simulations of the complete cache system are performed on the software *Xilinx ISE CAD* based on VHDL-coding with a hardware evaluation platform *Xilinx Spartan-3E FPGA*. The major finding is that whenever a CPU requests a write or read, the cache controller will compare the tag portion of the address word issued by the CPU and check the valid and dirty bits, then fetch or write out the whole data block from or to the main memory. After the comparison, the cache controller will send the request to the SDRAM Controller and the final data block will be read from or write to the local BlockRAM memory. During the whole process, the cache controller has to go through several states and clock cycles in order to complete the whole request.

2.0 - Introduction

This lab introduces the functions of a cache controller based on VHDL-coding implementations and emulations of the FPGA hardware platform. Nowadays, the efficiency of memory access has become a great focus on computer designs. Cache Memory allows to fast store and access small amounts of data that are recently accessed without going through the slower Main Memory. These data are stored as blocks of word addresses with fields of index, tag, valid bits and dirty bits. The cache controller is responsible of identifying and comparing the data blocks when the CPU issues a read or write request. The purpose of this laboratory project is to design such cache system that utilizes the small and fast SRAM Memory Block to access or store data blocks from or to the Main Memory. This report states the process of the cache system and specifies the components/devices needed and each of their functionalities, as well as analyzes the performance of the Cache Memory accessing.

3.0 – Specifications

CPU

The CPU issues a 16-bit word address to the cache controller. The address contains an 8-bit upper tag, 3-bit index tag, 5-bit offset in that order. The CPU store data and addresses in a 24-bit vector with the upper 16 bits containing the address while the lower 8 bit contains the data. The cache controller will issue a signal to CPU when it needs to replace a block. The CPU only stores data when the cache controller gives a ready signal which is TRIG_R to the CPU which signals that the transaction is complete.

Cache Controller

The cache controller is the top-level entity. The cache controller gets write and read request from the CPU. Based upon the signals that it stores internally and based on the request, the cache controller will act based upon the request. The cache controller is a finite state machine with 5 states. There is no pre-set state for the controller so the first request is based upon which signals came first in the board so it may be corrupted. The five states are:

State 0:

It updates the signals inside the cache. It acts as mux to the cache and decides where data out of the cache goes towards and data in goes towards. After it sets, the state signal to state 3.

State 1:

It loads data from the main memory in the CPU. It sets the offset signal to 0 and loads the entire block data from CPU memory with a counter. It sets the W_R signal to 0 which is a read for the SDRAM memory. It waits every other cycle as it must wait for the memory to finish executing which takes one cycle. It sets the state to zero.

State 2:

It writes data back to main memory in the CPU. It sets offset signal to 0 and replaces the entire block data from cache memory with the block data from CPU. It sets the W_R signal to 1 which is a write for the SDRAM memory. It waits every other cycle as it must wait for the memory to finish executing which takes one cycle. It sets the state to 1 and sets the corresponding d bit to 0.

State 3:

This state is basically an idle state. It sets the CPU ready signal to 1 and it waits for CPU strobe to equal 1. When the strobe equals to one, it sets the state to state 4.

State 4:

This state is the compare state. It sets all the signals such as CPU_READ, CPU_TAG to their corresponding values. It compares the tag inside the memory of the CPU toward the tag of the request sent and the v-bit, if they are both equal to 1 then it sets the hit tag to equal to one and it sets the state to state 0. Or else, it is a miss, so it needs to figure out which state to set it to. If the D-bit and the V-bit equals to 1, set the state to zero, i.e. write back state; else, it goes to state 1 which is the read state.

Local SRAM

The block memory was generated with the wizard. It has a width of 8 for 8 caches block. It has the length of 256 which is obtained from 32 word* 8 bit which equals 256 bits. It means each cache block has 256 bits. There is a WEN which tells the block if it is a read or a write. It takes the memory one clock cycle before the transaction is finished.

SDRAM Controller

The SDRAM controller moves the data from the cache to the CPU or to the input of the cache. It has its own memory which keeps a copy of the cache memory data. The module only sets data when the Mem_Strobe =1 because that is when the memory from data in is finished. When the write = 1, the memory writes the din into the Ram_Memory. When write =0, it sends the Ram_Memory towards data out.

4.0 - Device Description

The CPU issues a read or write request periodically with respect to clock cycles. The CPU contains input ports of a strobe CS, a read/write indicator WR_RD, a 16-bit address, 8-bit data in, ready state indicator RDY, and output ports DIN and DOUT^[1]. When the CPU issues a request, it will set the corresponding word address and the read/write indicator where '0' for read and '1' for write. Once the request is been confirmed by the Cache Controller, the CPU_RDY will be set to high indicating that it is read to make the transaction; and once the transaction begins, the RDY will be set to low again indicating that it is not accepting any new transactions since it is processing the current transaction.

The SDRAM Controller interacts with the Cache Controller dealing with the read and write requests. It contains input ports of a 16-bit address, a read/write indicator WR_RD, a strobe MEMSTRB, 8-bit data in, and output ports of DIN and DOUT^[1]. When a write instruction is received, the WR_RD will be set to high and set to low when a read instruction is received. After confirming the request and all the signals are stable, the MEMSTRB strobe will be asserted for one clock cycle. To complete write the whole data block into the memory, the strobe will go through 32 clock cycles.

The local SRAM, or so called the BlockRAM, is used to store the data blocks that are recently accessed for future accesses. It contains an 8-bit address, 8-bit data input and output ports DIN and DOUT^[1]. For write operations, the BlockRAM sets a specific address on the specific port and enables WEN signal; for read operations, it sets the address on the address bus then send to the output signal DOUT after the next rising edge of clock.

4.1 - Symbols

Figures 1 to 3 are generated in the *Xilinx ISE CAD* based on the VHDL codes written. These symbols show the input and output pins for the devices. The VHDL code for the CPU was given in the lab instructions.

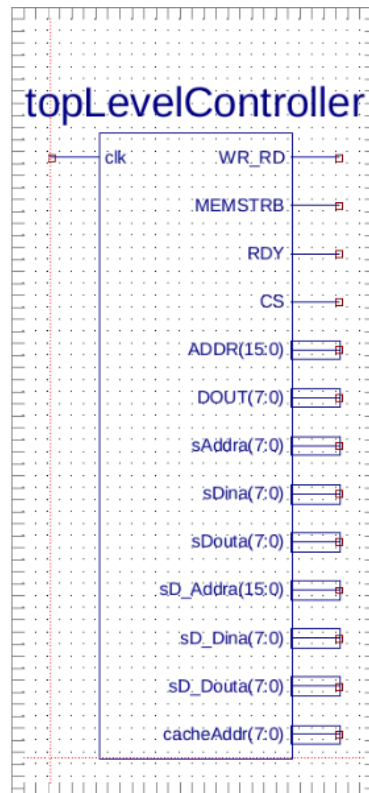


Figure 1 – Cache Controller Symbol

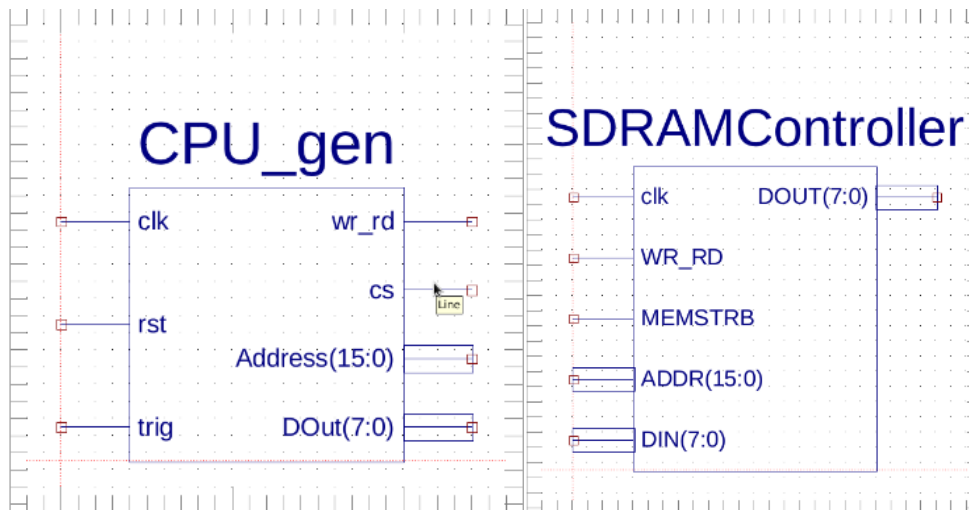


Figure 2 – CPU symbol

Figure 3 – SDRAM-controller symbol

4.2 - Block Diagram

The complete cache system diagram is shown in **Figure 4** [1]. This figure shows how each device are connected and mapped to perform a cache system. The Cache SRAM is the local BlockRAM associated with the Cache Controller and it is generated by the *Xilinx ISE*.

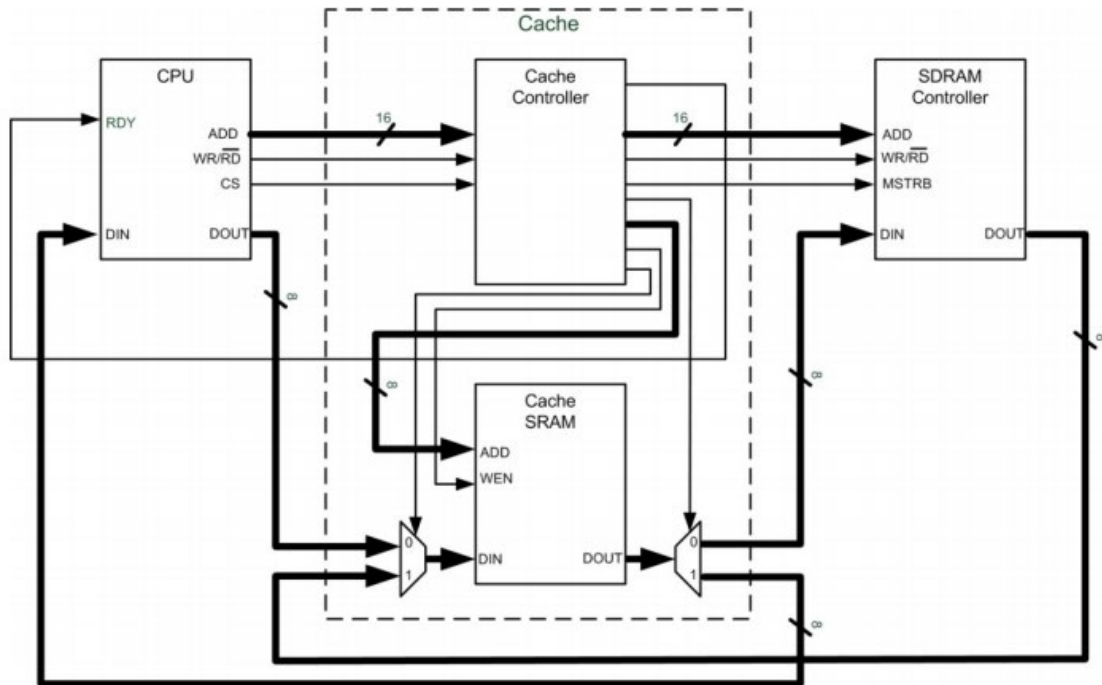


Figure 4 [1] – Complete Cache System

4.3 - State Diagram

There are five states that the cache controller will go through:

- State 0: update signals to CPU
- State 1: load from memory
- State 2: write back
- State 3: IDLE state
- State 4: compare tag

Figure 5 shows the interactions between each state and the conditions to switch states.



Figure 5 – State diagram of the Cache Controller during the read/write process

4.4 - Process Diagram

Based on the waveform generated by *Chipscope Pro Analyzer* in the *Xilinx ISE CAD*, the following diagram (**Figure 6**) shows the states of the cache controller went through during the process when the CPU issues a read or write request. **Figures 7 & 8** shows the waveform generated by the *Chipscope Pro Analyzer* for write and read process respectively.

Read Process

```
CPU_addr = '1102'
CPU_DIN = 'CC'
V-bit = '1'
D-bit = '1'
```

```
CPU_addr = '1102'
CPU_DIN = 'CC'
V-bit = '1'
D-bit = '1'
```

CPU_RW = '1'

CPU_RW = '0'

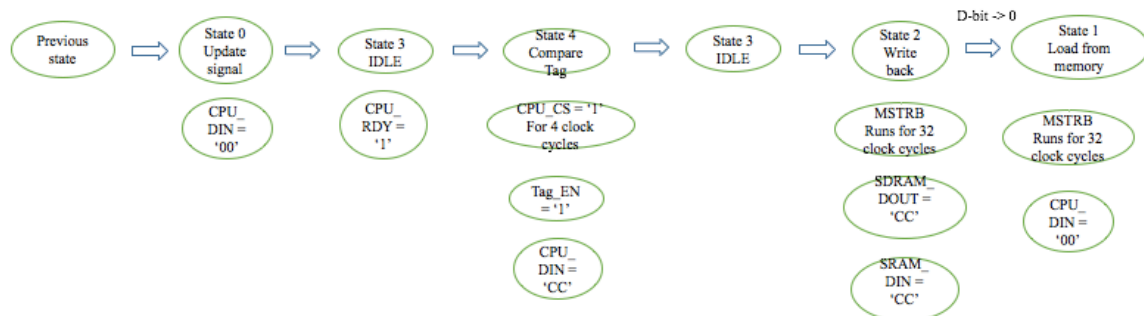


Figure 6 – Process diagram for read/write process

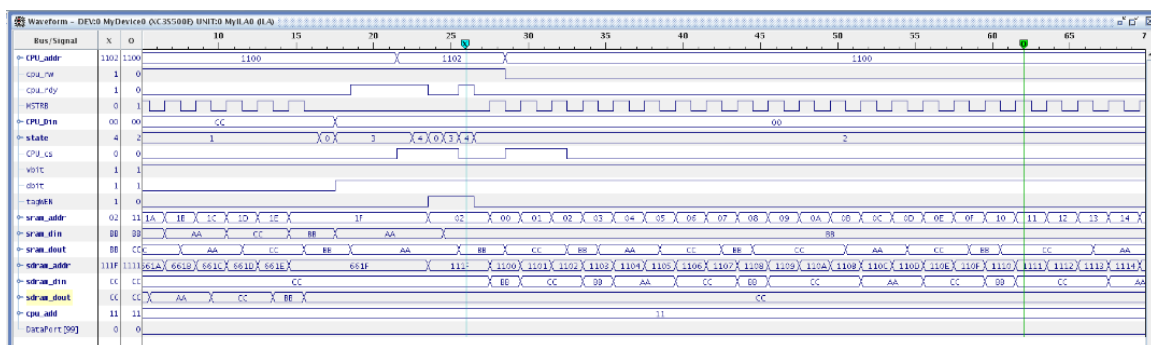


Figure 7 – Waveform generated by *Chipscope Pro Analyzer* for write process

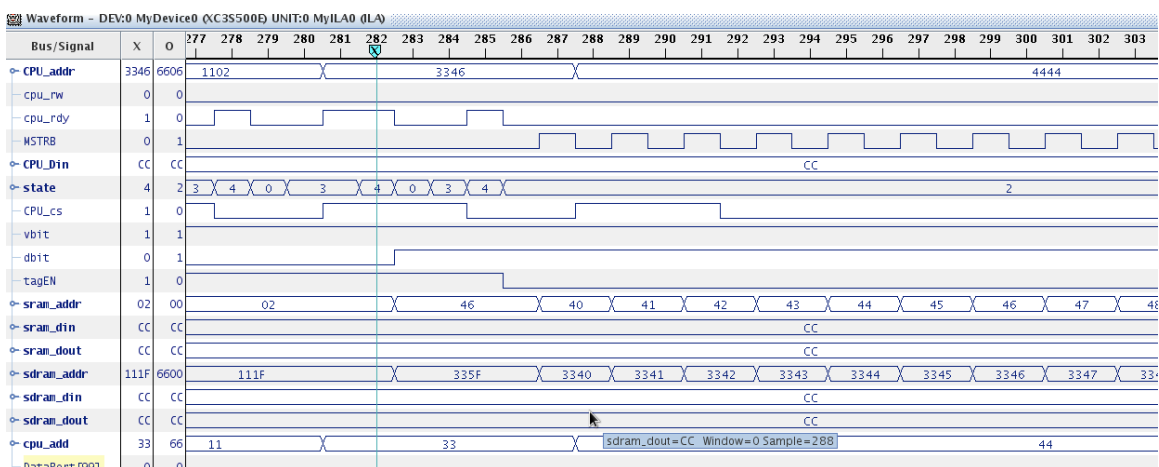


Figure 8 – Waveform generated by *Chipscope Pro Analyzer* for read process

5.0 – Results

The following parameters indicated on **Table 1** are determined based on the waveform generated by the *Chipscope Pro Analyzer*.

Table 1 – Cache performance parameters determined based on the timing scale on the waveform

N	Cache performance parameter	Time in nS
1	Hit/Miss determination time	1
2	Data access time	1
3	Block replacement time	64
4	Hit time (Case 1 and 2)	10
5	Miss penalty for Case 3 (when D-bit = 0)	64
6	Miss penalty for Case 4 (when D-bit = 1)	128

6.0 – References

[1] COE-758. (n.d.). Lev Kirischian. Cache Project. Retrieved from <https://www.ee.ryerson.ca/~lkirisch/coe758/labs.htm>.

7.0 – Appendix

Cache Controller VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity topLevelController is
    Port ( clk          : in STD_LOGIC;
          ADDR         : out STD_LOGIC_VECTOR(15 downto 0);
          DOUT         : out STD_LOGIC_VECTOR(7 downto 0);
          sAddr        : out STD_LOGIC_VECTOR(7 downto 0);
          sDina        : out STD_LOGIC_VECTOR(7 downto 0);
          sDouta       : out STD_LOGIC_VECTOR(7 downto 0);
          sD_Addra     : out STD_LOGIC_VECTOR(15 downto 0);
          sD_Dina      : out STD_LOGIC_VECTOR(7 downto 0);
          sD_Douta     : out STD_LOGIC_VECTOR(7 downto 0);
          cacheAddr    : out STD_LOGIC_VECTOR(7 downto 0);
          WR_RD, MEMSTRB, RDY ,CS      : out STD_LOGIC);
end topLevelController;

architecture Behavioral of topLevelController is

    -----
    --SDRAM controller component
    -----

    COMPONENT SDRAMController
    Port (
        clk                                : in
        STD_LOGIC;
        ADDR                                : in
        STD_LOGIC_VECTOR (15 downto 0);
        WR_RD                                : in STD_LOGIC;
        MEMSTRB                             : in STD_LOGIC;
        DIN                                 : in STD_LOGIC_VECTOR (7 downto 0);
        DOUT                                 : out STD_LOGIC_VECTOR (7 downto
0));
        END COMPONENT;
        signal SDRAM_Din,SDRAM_Dout        : STD_LOGIC_VECTOR(7 downto 0);
        signal SDRAM_ADD                    : STD_LOGIC_VECTOR(15
downto 0);
        signal SDRAM_MSTRB,SDRAM_W_R      : STD_LOGIC;
        signal counter                      : integer := 0;
```

```

        signal soffset                                     : integer := 0;
-----
--SRAM component
-----
        COMPONENT SRAM
        PORT (
            clka                                           : IN STD_LOGIC;
            wea                                           : IN STD_LOGIC_VECTOR(0 DOWNT0
0);
            addra                                         : IN STD_LOGIC_VECTOR(7 DOWNT0 0);
            dina                                           : IN STD_LOGIC_VECTOR(7 DOWNT0
0);
            douta                                         : OUT STD_LOGIC_VECTOR(7 DOWNT0 0));
        END COMPONENT;
        signal Dbit                                       : STD_LOGIC_VECTOR(7 downto 0):=
"00000000";
        signal Vbit                                       : STD_LOGIC_VECTOR(7 downto 0):=
"00000000";
        signal sADD, sDin, sDout                       : STD_LOGIC_VECTOR(7 downto 0);
        signal sWen                                       : STD_LOGIC_VECTOR(0 DOWNT0 0);
        signal TAGWen                                    : STD_LOGIC := '0';
        type cachememory is array (7 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
        signal memtag: cachememory := ((others=> (others=>'0')));
-----
--CPU component
-----
        COMPONENT CPU_gen
        Port (
            clk                                           : in
STD_LOGIC;
            rst                                           : in STD_LOGIC;
            trig                                           : in STD_LOGIC;
            Address                                       : out STD_LOGIC_VECTOR (15 downto
0);
            wr_rd                                         : out STD_LOGIC;
            cs                                           : out STD_LOGIC;
            Dout                                          : out STD_LOGIC_VECTOR (7 downto
0));
        END COMPONENT;
        signal CPU_Dout, CPU_Din                       : STD_LOGIC_VECTOR(7 downto 0);
        signal CPU_ADD                                   : STD_LOGIC_VECTOR (15 downto 0);
        signal CPU_W_R,CPU_CS                           : STD_LOGIC;
        signal CPU_RDY                                   : STD_LOGIC;
        signal cpu_tag                                   : STD_LOGIC_VECTOR(7 downto 0);
        signal index                                     : STD_LOGIC_VECTOR(2 downto 0);
        signal offset                                    : STD_LOGIC_VECTOR(4 downto 0);
        signal Tag_index                                : STD_LOGIC_VECTOR(10 downto 0);
-----
--ICON component
-----
        COMPONENT icon
        PORT (

```

```

CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
END COMPONENT;

-----
--ILA component
-----

COMPONENT ila
PORT (
CONTROL : INOUT
STD_LOGIC_VECTOR(35 DOWNTO 0);
CLK : IN STD_LOGIC;
DATA : IN
STD_LOGIC_VECTOR(99 DOWNTO 0);
TRIG0 : IN
STD_LOGIC_VECTOR(0 TO 0));
END COMPONENT;

-----
--Cache controller state
-----

TYPE state_value IS (state4, state0, state1, state2, state3);
signal state_current : state_value ;
signal state : STD_LOGIC_VECTOR(3 downto 0);

-----
--ICON & VIO & ILA Signals
-----

signal control0 : STD_LOGIC_VECTOR(35 downto 0);
signal ila_data : std_logic_vector(99 downto 0);
signal trig0 : std_logic_vector(0 TO 0);

BEGIN

-----
--Port mapping
-----

cpu : CPU_gen Port Map (clk,'0',CPU_RDY,CPU_ADD,CPU_W_R,CPU_CS,CPU_Dout);
sdcontroller : SDRAMController Port Map
(clk,SDRAM_ADD,SDRAM_W_R,SDRAM_MSTRB,SDRAM_Din,SDRAM_Dout);
csram : SRAM Port Map (clk,sWen,sADD, sDin, sDout);
sysIcon : icon Port Map (CONTROL0);
sysILA : ila Port Map (CONTROL0,CLK,ila_data, TRIG0);

process(clk, CPU_CS)
begin

-- state 0: update signals
-- state 1: load from memory
-- state 2: write back
-- state 3: IDLE
-- state 4: compare tag

if (clk'event AND clk = '1') then
if (state_current = state4) then
CPU_RDY <= '0';

```

```

cpu_tag      <= CPU_ADD(15 downto 8);
index        <= CPU_ADD(7  downto 5);
offset       <= CPU_ADD(4  downto 0);
SDRAM_ADD(15 downto 5)    <= CPU_ADD(15 downto 5);
sADD(7 downto 0)          <= CPU_ADD(7 downto
0);

sWen <= "0";
--Tag compare
if(Vbit(to_integer(unsigned(index))) = '1'
hit      AND memtag(to_integer(unsigned(index))) = cpu_tag) then --

    TAGWen <= '1';--hit or miss
    state_current <= state0;
    state <= "0000";
else --miss
    TAGWen <= '0';
    if (Dbit(to_integer(unsigned(index))) = '1'
        AND Vbit(to_integer(unsigned(index))) = '1') then
        --write back and then load from sdram
        state_current <= state2; --write back state
        state <= "0010";
    else --no write back needed as valid or dirty bit is 0

        state_current <= state1;
        state <= "0001";
    end if;
end if;

elseif(state_current = state0) then
--update signals
    if (CPU_W_R = '1') then
        sWen <= "1";
        Dbit(to_integer(unsigned(index))) <= '1';
        Vbit(to_integer(unsigned(index))) <= '1';
        sDin <= CPU_Dout;
        CPU_Din <= "00000000";

    else
        CPU_Din <= sDout;
    end if;

    state_current <= state3; --Switching to idle state as request is completed
    state <= "0011";

elseif(state_current = state1) then
--loading from main memory
    if (counter = 64) then
        counter <= 0;
        Vbit(to_integer(unsigned(index))) <= '1';
        memtag(to_integer(unsigned(index))) <= cpu_tag;
        sdooffset <= 0;
        state_current <= state0;

```

```

        state <= "0000";
    else
        if (counter mod 2 = 1) then
            SDRAM_MSTRB <= '0';
        else
            SDRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
            SDRAM_W_R <= '0';
            SDRAM_MSTRB <= '1';
            sADD(7 downto 5) <= index;
            sADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
            sDin <= SDRAM_Dout;
            sWen <= "1";
            sdoffset <= sdoffset + 1;
        end if;
        counter <= counter + 1;
    end if;

elseif(state_current = state2) then
    --writing back to main memory as valid or dirty bit is 1
    if (counter = 64) then
        counter <= 0;
        Dbit(to_integer(unsigned(index))) <= '0';
        sdoffset <= 0;
        state_current <= state1;
        state <= "0001";
    else
        if (counter mod 2 = 1) then
            SDRAM_MSTRB <= '0';
        else
            SDRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
            SDRAM_W_R <= '1';
            sADD(7 downto 5) <= index;
            sADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
            sWen <= "0";
            SDRAM_Din <= sDout;
            SDRAM_MSTRB <= '1';
            sdoffset <= sdoffset + 1;
        end if;
        counter <= counter + 1;
    end if;

elseif(state_current = state3) then
    CPU_RDY <= '1';
    if (CPU_CS = '1') then
        state_current <= state4;
        state <= "0100";
    end if;
end if;

```

```

        end if;
end process;

MEMSTRB <= SDRAM_MSTRB;
ADDR   <= CPU_ADD;
WR_RD  <= CPU_W_R;
DOUT   <= CPU_Din;
RDY    <= CPU_RDY;
CS     <= CPU_CS;

sAddra <= sADD;
sDina  <= sDin;
sDouta <= sDout;

sD_Addra <= SDRAM_ADD;
sD_Dina  <= SDRAM_Din;
sD_Douta <= SDRAM_Dout;

cacheAddr <= CPU_ADD(15 downto 8);
-----
--mapping ILA
-----
ila_data(15 downto 0) <= CPU_ADD;
ila_data(16) <= CPU_W_R;
ila_data(17) <= CPU_RDY;
ila_data(18) <= SDRAM_MSTRB;
ila_data(26 downto 19) <= CPU_Din;
ila_data(30 downto 27) <= state;
ila_data(31) <= CPU_CS;
ila_data(32) <= Vbit(to_integer(unsigned(index)));
ila_data(33) <= Dbit(to_integer(unsigned(index)));
ila_data(34) <= TAGWen;
ila_data(42 downto 35) <= sADD;
ila_data(50 downto 43) <= sDin;
ila_data(58 downto 51) <= sDout;
ila_data(74 downto 59) <= SDRAM_ADD;
ila_data(82 downto 75) <= SDRAM_Din;
ila_data(90 downto 83) <= SDRAM_Dout;
ila_data(98 downto 91) <= CPU_ADD(15 downto 8);

end Behavioral;

```


SDRAM Controller VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

entity SDRAMController is

```
    Port (
        clk                : in STD_LOGIC;
        ADDR                : in STD_LOGIC_VECTOR (15 downto 0);
        WR_RD               : in STD_LOGIC;
        MEMSTRB              : in STD_LOGIC;
        DIN                  : in STD_LOGIC_VECTOR (7 downto 0);
        DOUT                 : out STD_LOGIC_VECTOR (7 downto 0));
```

end SDRAMController;

architecture Behavioral of SDRAMController is

```
    type ramemory is array (7 downto 0, 31 downto 0) of std_logic_vector(7 downto 0);
    signal RAM_SIG: ramemory;
```

```
    signal counter : integer := 0;
```

begin

process (CLK)

begin

if CLK'event and CLK = '1' then

if counter = 0 then

for I in 0 to 7 loop

for J in 0 to 31 loop

RAM_SIG(i,j) <= "11110000";

end loop;

end loop;

counter <= 1;

end if;

if MEMSTRB = '1' then

if WR_RD = '1' then

RAM_SIG(to_integer(unsigned(ADDR(7 downto 5))),to_integer(unsigned(ADDR(4 downto 0)))) <= DIN;

ELSE

DOUT <= RAM_SIG(to_integer(unsigned(ADDR(7 downto 5))),to_integer(unsigned(ADDR(4 downto 0))));

end if;

end if;

end if;

end process;

end Behavioral;