Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Digital Systems Engineering
Course Number:	COE758
Semester/Year (e.g.F2016)	Fall 2022

Instructor:	Lev Kirischian

Assignment/Lab Number:	1
Assignment/Lab Title:	Memory Heirarchy: Cache Controller

Submission Date:	November 11, 2022
Due Date:	November 11, 2022

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Iqbal	Hamza	500973673	032	H.I
Ayntabli	Krikor	500895492	032	K.A

Table of Contents

Abstract

Learning how to design a cache controller inside of an FPGA and how it should communicate with the CPU, SRAM, and SDRAM controller were the main goals of this project. This project's objective was to find out how the cache controller functions. For this project, the SRAM controller and cache controller have to be created using the VHDL language. The SRAM block was built using the Xilinx ISE programme. The Xilinx Spartan 3E was selected as the FPGS.

A schematic block diagram that incorporated every key component's block diagram and connected them all together to make a full schematic was required to complete this task. The 256 bytes of data that are kept in the cache memory are organized into blocks of 32 words. The CPU communicates with the cache memory through the cache controller. Data reading and writing to the cache memory is the function of the cache controller. The main memory is also accessible to the cache controller, allowing it to read from and write to that memory as well. Once the schematic and all the blocks were connected, all the VHDL components had been built. The VHDL code was successfully compiled.

The cache controller handles an instruction (read or write) after it has been sent by the CPU. Due to a tag that is present in the cache's memory block, the cache can output either a hit or a miss. We have to test the following four cases:

1. Read a word from the memory block of the cache.
2. Write a word to the memory block of a cache.
3. With the dirty bit set to 0, read from or write to the cache block.
4. Read from or write to the cache block with the dirty bit set to 0.

Introduction

The majority of computers are built with a CPU that communicates with several types of memory. The most crucial data is kept in the CPU's built-in cache memory, which is one of several caches. The CPU can store data that has to be retrieved as quickly & frequently as possible using the cache memory, which sits closest to the CPU. The requirement for the cache stems from the fact that other types of memory are frequently installed far from the CPU, making retrieving the data stored there time-consuming. A cache controller, which sits in between the CPU and the cache memory, must be constructed in order for the cache SRAM to interface with the CPU. It has access to the main memory as well. Overall, this concept, known as the *principle of locality*, has allowed for efficient use of various memory systems and overall performance improvements. The following figure showcases an example of the CPU & memory hierarchical structure:

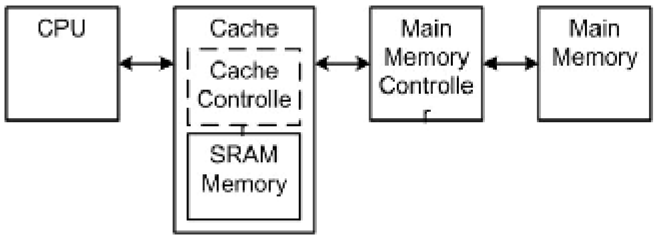


Figure 1: Hierarchical structure of CPU interfacing with Cache and other Memory blocks

With this project, the cache controller was implemented inside of an FGPA, using Xilinx ISE, via VHDL modules. Once the VHDL modules were completed, they were tested through simulations and then connected together using a schematic file. The following figure shows what the implementation would appear like once implemented:

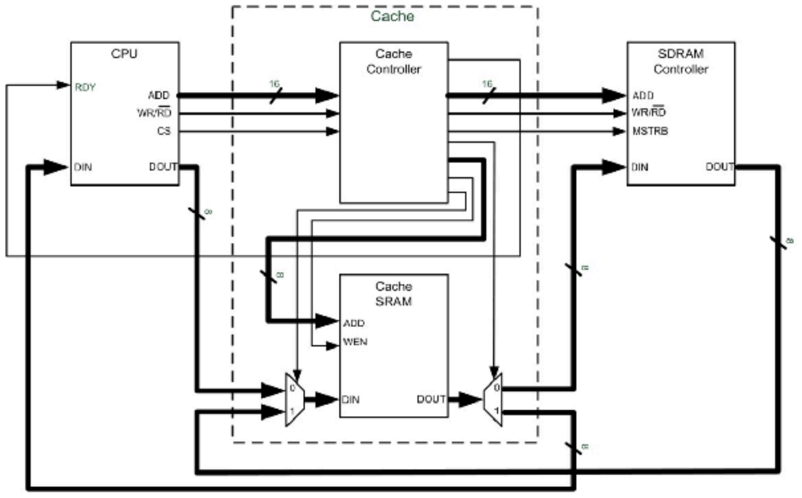


Figure 2: Structure of CPU interfacing with Cache and other modules

Already provided was the VHDL code file of the CPU. The Cache controller, SRAM, & SDRAM modules needed to be created. The Xilinx memory generator software was used to construct the SRAM (Block RAM) module. Figure 2 demonstrates that the CPU transmits instructions to the cache controller, which then the cache controller will either read from or write to the SRAM memory. Additionally, data is sent to the CPU by the SDRAM controller.

Specification

The project's objective is to create a logic circuit that reacts properly to read and write requests from the CPU. The options the cache controller has for action are retrieving complete data blocks from the main memory, or reading/writing to cache memory.

The project's cache memory is divided into 8 blocks, each of which is made up of 32 words. The 16-bit address words are divided into three components: tag (8-15), offset (0–3), and 4–7 (index). The tag field acts as a unique identifier, which contains the address information required to identify whether the associated block in the hierarchy corresponds to the requested word. The index and block offset field help in locating the specific block & word of memory within the cache that must be used by the CPU. Figure 3 showcases a breakdown of cache memory:

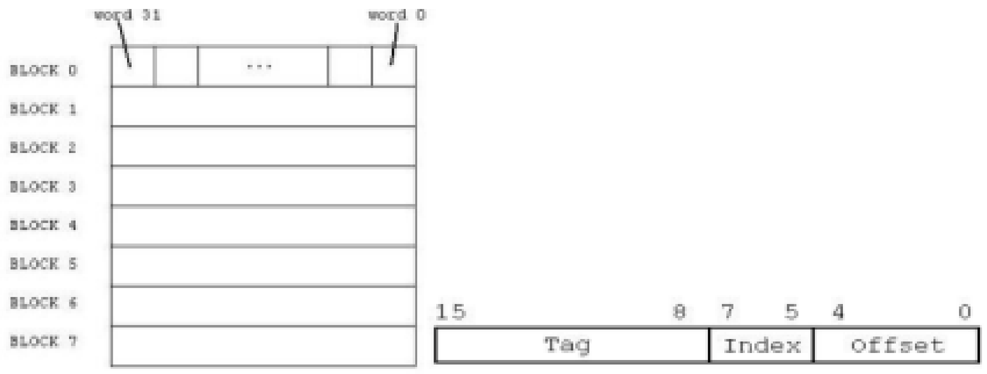


Figure 3: Organization of Cache memory, & Address Word Register fields

Based on the indicators it internally holds that define the status of each block in the cache memory, the controller chooses which of these actions to perform. In particular, the controller must check the status of the dirty and valid bits for the block being targeted and compare the tag component of the address with the tags recorded for each of the eight blocks.

To transfer data between main memory and cache memory or between main memory and cache memory. We must look into four distinct behavioral cases:

Case 1. Read word from Cache (Hit): When a word is read from cache memory (a hit), the CPU sends the cache controller a read signal.It reads the cache memory and sends the information to the SRAM controller, which transmits it to the CPU.

Case 2. Write word to the Cache (Hit): The cache controller receives a write signal from the CPU, and writes the data to the cache memory. A cache hit is the outcome of this. The 16-bit address's index and offset components decide where the data will be stored within the cache memory. Since data was written to the cache, the dirty bit now reads 1. To validate the data, the valid bit is also set to 1.

Case 3. If Dirty bit = 0, read from/write to the cache (Miss): The cache does not contain the data despite the write/read signal being sent by the CPU to the cache controller. If the dirty bit is 0, the SDRAM controller receives the offset “0000” (indicating the beginning, base address of the block with its specific index) and must read and write the block to the cache SRAM memory. The following register's tag address is replaced with the tag portion of the address, whose valid bit is set to 1.

Case 4. If Dirty bit = 1, read from/write to the cache (Miss): The CPU attempts to read/write to the cache controller, but the specific data is not present in the cache memory, resulting in a miss. The dirty bit is 1 in the case, indicating that the block of data in the SRAM has been used and modified recently and must be written back into main memory, using the SDRAM memory controller. The memory block is written to the bass address with tag+index+'0000', where the tag and index are of the 32-bit address, along with offset ‘0000’. The new block (which is based on the address requested by the CPU) is then copied to the cache memory block, where the tag+index+'0000' address is read by the SDRAM controller to retrieve this data. The tag is that of the Address Word Register requested by the CPU in this case. Once complete, the request originally sent by the CPU can be completed.

Device Design

Symbols

The following are the idealized symbols of the components required to implement the cache controller, along with its input and output pins:

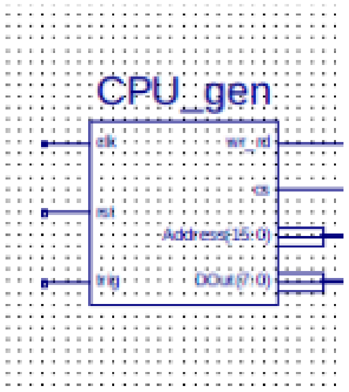


Figure 4: CPU interface

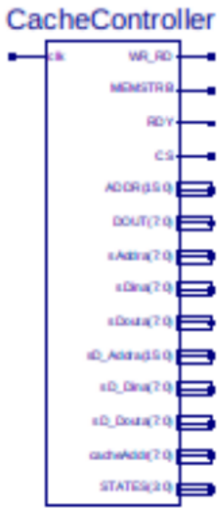


Figure 5: Cache Controller interface

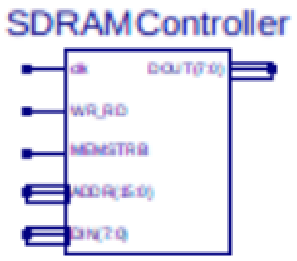


Figure 6: SDRAM Controller interface

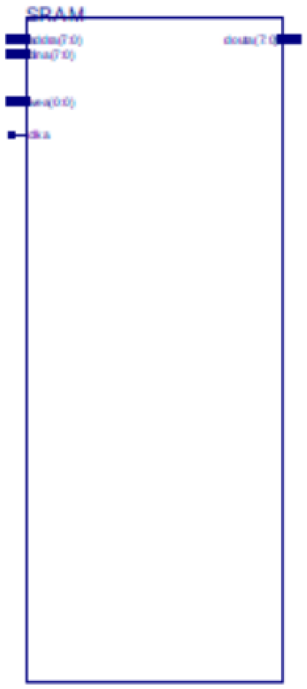


Figure 7: Block RAM (SRAM) interface

Block Diagram

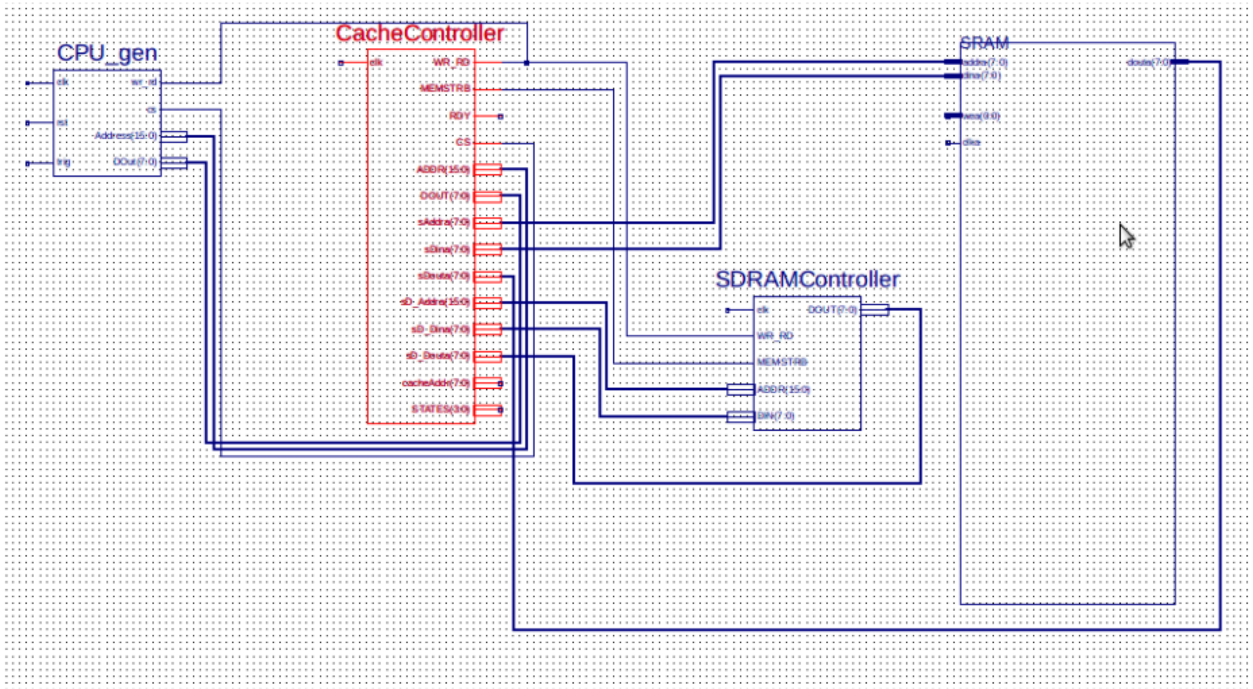


Figure 8: Schematic diagram, showcasing connections of each component with one another to create the project

State Diagram

The following is the state diagram of the cache controller:

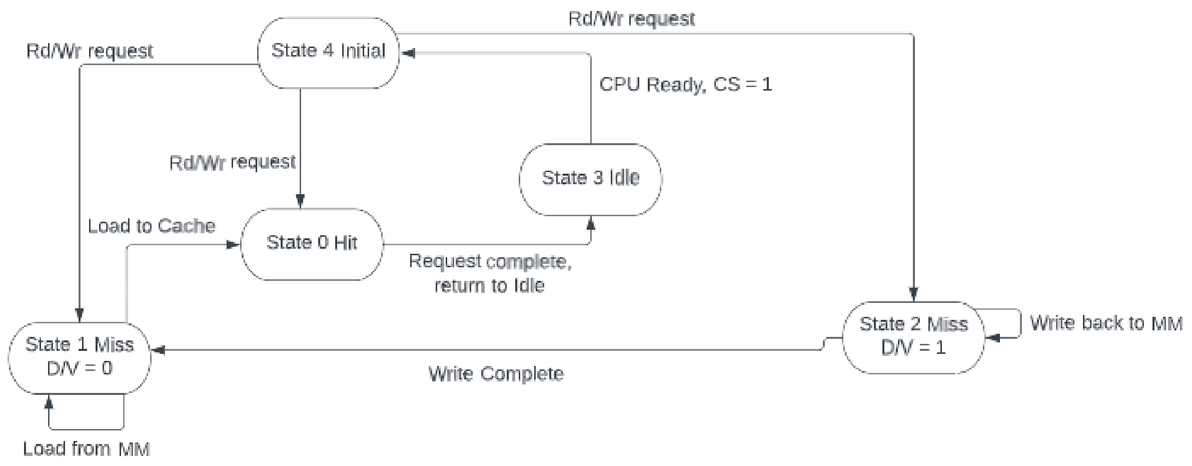


Figure 9: State diagram of the Cache Controller and its relationship with Main Memory & CPU

Process Diagram

The following is the process diagram of the cache controller:

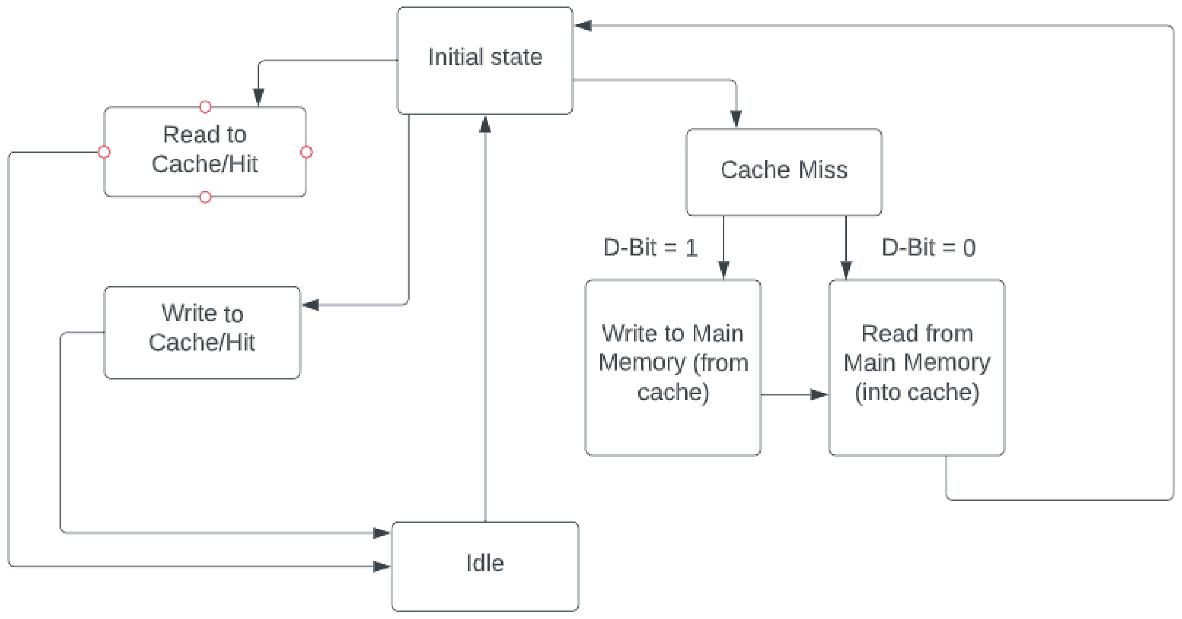


Figure 10: Process diagram of the Cache Controller and its relationship with Main Memory & CPU

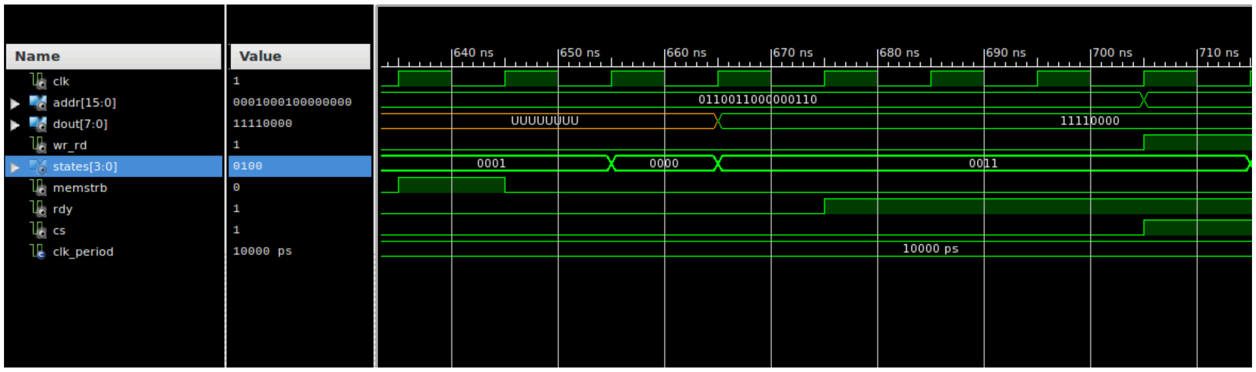
Results

Functional/Timing Diagrams

We can get a sense of how the cache controller interacts with the cache memory, the CPU, and the SDRAM through functional simulations. In this implementation, there are 4 states which are present.

State 4 represents the initial state, where the Cache controller will receive an address from the CPU and run its comparative logic to determine whether it is a hit or miss. State 0 will represent the state at which a hit within the cache has occurred. State 1 & 2 will represent the miss cases, with the former representing the case where the dirty/ valid bit = 0, to which the necessary block will simply be loaded in from main memory. The latter will represent the case where the dirty/ valid bit = 1, to which data will be written back into main memory. Lastly, state 3 will represent the idle state, where the cache controller will wait until it receives an instruction from the CPU.

The following are the functional simulation diagrams for each of the behavioral cases (as described within the specification section of the report) of the cache controller:



Name	Value
clk	1
addr[15:0]	0001000100000000
dout[7:0]	11110000
wr_rd	1
states[3:0]	0100
memstrb	0
rdy	1
cs	1
clk_period	10000 ps

Figure 10: Going from state 1 (miss w/ V bit/ D bit =0) to state 0 (hit) to state 3(idle)

In the figure 10 above, a read operation is to be performed, as the wr_rd is low. Initially, no data is found within the cache, resulting in a miss (w/ V bit/ D bit =0). This will cause the behavioral case 3, that is , the SDRAM will write the required block into the SRAM memory. Upon completion, The state will now be in state 0, that is, it will be a hit (case 1), and the data will be sent to the cpu. From there, the cache will be in an idle state, awaiting the next instruction from the CPU.

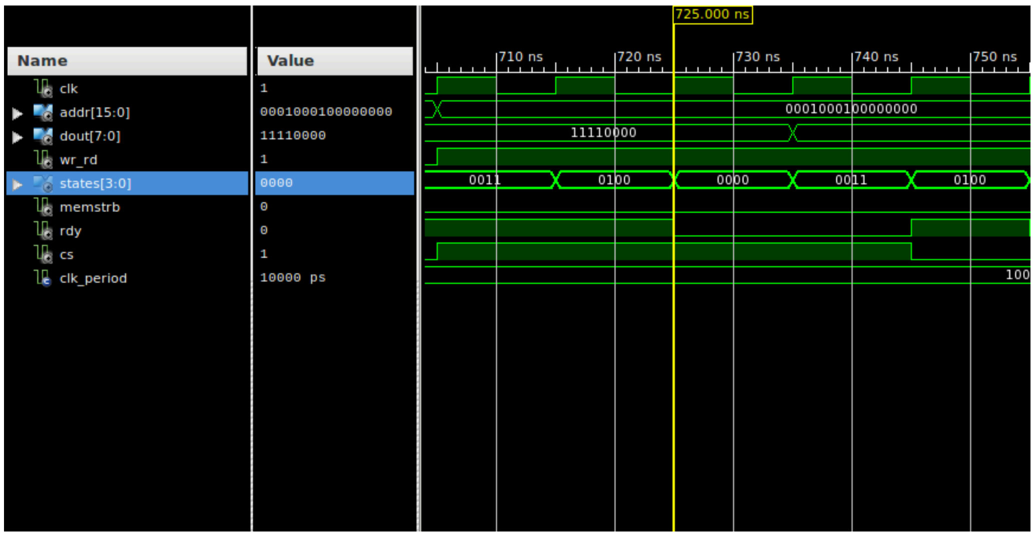


Figure 11: Going from state 3 (idle) to state 4 (initial) then going from state 0 (hit) to state 3(idle) to state 4 (initial)

In figure 11 above, the cache controller starts off in an idle state, waiting for the next instruction from the CPU. Upon retrieving it, it will switch the state 4, and run its comparative logic. A write request is received (as the wr_rd is high). Since the data already exists within the cache, The state will now be in state 0, that is, it will be a hit (case 1), and the data will be sent to the cpu. From there, the cache will be in an idle state, awaiting the next instruction from the CPU once again.

In the following figure (12 (a),(b), (c)) below, the case where the specific word is not within the cache, and the block itself has been modified (resulting in d/v bit=1), results in case 4 occuring. It starts off in an idle state (3), waiting for the next instruction from the CPU. Upon retrieving it, it will switch the state 4, and run its comparative logic. Since the data doesn't exist and has been modified within the cache, 'write-back' will occur, hence the state will now be in state 2. Once that specific block has been written into main memory, the required block can now be retrieved, that is, it will set v/d bit=0, and perform case 3, hence why the state goes back to state 1. The data will be loaded into the cache, a hit will then occur and the data will be sent to the cpu. From there, the cache will be return to an idle state, awaiting the next instruction from the CPU once again.

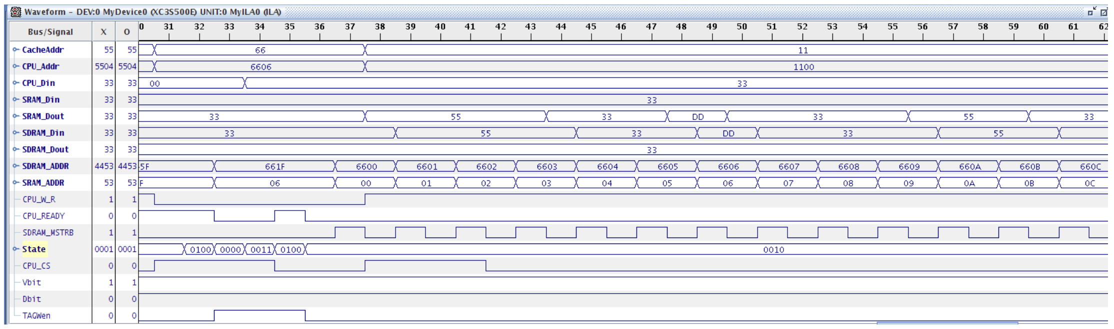


Figure 12 (a): Going from state 2 (miss w/ V-bit+Dbit=1) to state 1 (miss w/ V bit/ D bit =0) to state 0 (hit) to state 3(idle) to state 4 (initial)

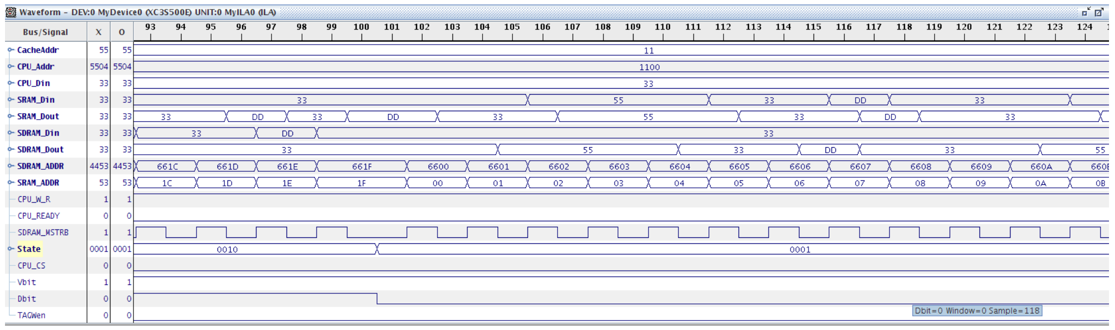


Figure 12 (b): Going from state 2 (miss w/ V-bit+Dbit=1) to state 1 (miss w/ V bit/ D bit =0) to state 0 (hit) to state 3(idle) to state 4 (initial)

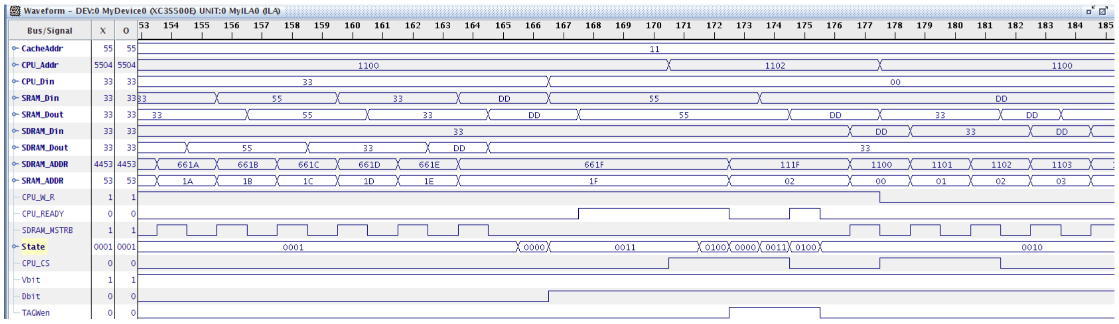


Figure 12 (c): Going from state 2 (miss w/ V-bit+Dbit=1) to state 1 (miss w/ V bit/ D bit =0) to state 0 (hit) to state 3(idle) to state 4 (initial)

Cache Performance Analysis

The following calculations and analysis were based on figures 12 (a),(b),(c):

Hit/Miss Time: Since the time must be less than 20ns (the CPU cycle), it must be approximately 15ns.
Data access time: To do this, we must subtract the time between the idle state and writing from cache to SDRAM, which results in 40 ns.
Block Replacement Time: Estimate the time to be approximately 1280 ns based on the memory strobe signal.
Hit time: Based on going from state 0 to state 3 , this would be around 10 ns for reading or writing.
Miss penalty for case 3: would result in a 710 ns penalty.
Miss penalty for case 4: would be around 1370 ns based on the diagram, due to excess time spent writing back into main memory.

Hit/miss time <20 ns (ns)	15
Data access time (ns)	40
Block replacement time (ns)	1280
Hit time (ns)	10
Miss penalty for case 3 (ns)	710
Miss penalty for case 4 (ns)	1370

Table 1: Cache Performance Results

Conclusion

Overall, the project was completed. The block diagram was made using the VHDL code, which was successfully compiled for each component. Then, using the block diagram, simulated waveforms were created to display all possible behaviors between the CPU and the Cache controller.

The VHDL code for each component is provided below. The TA saw the waveforms in action, and they appeared to be working. The project provided greater insight about the relationship & interactions between the cache memory, the CPU, and the SDRAM controller. The project worked as expected overall.

References

1. Cache Controller - an overview | ScienceDirect Topics. (n.d.). ScienceDirect Topics. Retrieved November 7, 2022, from https://www.sciencedirect.com/topics/computer-science/cachecontroller

2. L.K. (n.d.-b). Ryerson University - Lev Kirischian - COE758 Project. COE758 Project 1. Retrieved November 7, 2022, from https://www.ee.ryerson.ca/%7Elkirisch/coe758/labs.htm

3. L.K. (n.d.). Ryerson University - Lev Kirischian - COE758. COE758. Retrieved November 3, 2022, from https://www.ee.ryerson.ca/%7Elkirisch/coe758/coe758.htm

4. L.K. (n.d.). Ryerson University - Lev Kirischian - COE758 Project 1 lab manual - https://www.ee.ryerson.ca/~lkirisch/ele758/labs/Cache%20Project%5B11-09-15%5D.pd

5. Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design - The Hardware/Software Interface* (5th ed.). Morgan Kaufmann Publishers.

Appendix

7. Appendix (code screenshots)

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

entity CacheController 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);
                              sDout    : out  STD_LOGIC_VECTOR(7 downto 0);
                              sDin     : out  STD_LOGIC_VECTOR(7 downto 0);
                              sD_Addr  : out  STD_LOGIC_VECTOR(15 downto 0);
                              sD_Din   : out  STD_LOGIC_VECTOR(7 downto 0);
                              sD_Dout  : out  STD_LOGIC_VECTOR(7 downto 0);
                              cacheAddr : out  STD_LOGIC_VECTOR(7 downto 0);
                              STATES : out STD_LOGIC_VECTOR(3 downto 0);
           WR_RD, MEMSTRB, RDY ,CS        : out  STD_LOGIC);
end CacheController;

architecture Behavioral of CacheController is
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);
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 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 DOWNTO 0);
signal TAGWen                           : STD_LOGIC := '0';

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 sdoffset                               : integer := 0;

type cachememory is array (7 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
                  signal memtag: cachememory := ((others=> (others=>'0')));
```

```
        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);

        --Hit/Miss                        --0000 : state0
        --Load from Main Memory           --0001 : state1
        --Write back to Main Memory --0010 : state2
        --IDLE                            --0011 : state3
        --READY                           --0100 : state4

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


        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;

        COMPONENT SRAM
        PORT (
    clka                                        : IN STD_LOGIC;
    wea                                         : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra                                       : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    dina                                        : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta                                       : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
        END 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;
        COMPONENT icon
        PORT (
                CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
        END 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;

BEGIN
myCPU_gen: CPU_gen Port Map (clk,'0',CPU_RDY,CPU_ADD,CPU_W_R,CPU_CS,CPU_out);
SDRAM : SDRAMController Port Map
(clk,SDRAM_ADD,SDRAM_W_R,SDRAM_MSTRB,SDRAM_Din,SDRAM_Dout);
mySRAM              : SRAM Port Map (clk,sWen,sADD, sDin, sDout);
myIcon              : icon   Port Map (CONTROL0);
myILA               : ila    Port Map (CONTROL0,CLK,ila_data, TRIG0);

process(clk, CPU_CS)
        begin
                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";
                                if(Vbit(to_integer(unsigned(index))) = '1'
                                        AND memtag(to_integer(unsigned(index))) = cpu_tag) then
                                        TAGWen <= '1';
                                        state_current  <= state0;
                                        state          <= "0000";
                                else
                                        TAGWen <= '0';
                                        if (Dbit(to_integer(unsigned(index))) = '1'
                                                AND Vbit(to_integer(unsigned(index))) = '1') then
                                                state_current  <= state2;
                                                state          <= "0010";
                                        else
                                                state_current   <= state1;
                                                state           <= "0001";
                                        end if;
                                end if;

                        elsif(state_current = state0) then
                                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;
                                state <= "0011";
                        elsif(state_current = state1) then
                        if (counter = 64) then
                                        counter <= 0;
                                        Vbit(to_integer(unsigned(index))) <= '1';
                                        memtag(to_integer(unsigned(index))) <= cpu_tag;
                                        sdoffset <= 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:
```

```
                        elsif(state_current = state2) then
                        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;2
                        elsif(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;

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

        sAddr <= sADD;
        sDin <= sDin;
        sDout <= sDout;

        sD_Addr <= SDRAM_ADD;
        sD_Din <= SDRAM_Din;
        sD_Dout <= SDRAM_Dout;

        cacheAddr <= CPU_ADD(15 downto 8);

        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;
```