RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering

Course Number	COE 758
Course Title	Digital Systems Engineering
Semester/Year	F2023

Instructor	Dr. Vadim Guerkov

Project No.	1

Project Title	Memory Hierarchy: Cache Controller

Submission Date	October 29th, 2023
Due Date	October 30th, 2023

Student Name	Abdulrehman Khan	Kai Xu
Student ID	500968727	501041980
Signature*	A.K.	K.X.

**By signing above you attest that you have contributed to this written lab report and confirm that all work you have swung the lab contributed to this lab report is your own work.*

Table of Contents

Abstract

The primary objectives of this project were to develop a cache controller within an FPGA and establish its communication protocols with the CPU, SRAM, and SDRAM controller. The project aimed to unravel the inner workings of the cache controller, utilizing VHDL to construct both the SRAM controller and cache controller. The SRAM component was constructed using the Xilinx ISE software, with the Xilinx Spartan 3E FPGA chosen as the hardware platform.

To complete the task, a comprehensive schematic block diagram was necessary, encompassing all major components and their interconnections. The cache memory stored data in 256-byte blocks, each consisting of 32 words. The cache controller served as the intermediary for the CPU's communication with the cache memory, managing data read and write operations. Moreover, the cache controller had access to the main memory, enabling it to interact with it as needed. Once all schematic components were interconnected, the VHDL components were successfully constructed and compiled.

The cache controller is responsible for processing CPU instructions, whether they involve reading or writing data. The cache's memory block utilizes a tag to determine if a request results in a cache hit or a cache miss. The project requires testing in four scenarios:

1. Reading a word from the cache memory block.
2. Writing a word to the cache memory block.
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 computer systems feature a central processing unit (CPU) that communicates with multiple forms of memory. Among these memory components, the CPU's built-in cache memory, one of several caches, holds the most critical data. The cache memory, positioned closest to the CPU, serves as a repository for data that needs to be rapidly and frequently accessed. This necessity for caching arises from the fact that other types of memory are often located at a considerable distance from the CPU, resulting in slower data retrieval.

To facilitate the interaction between the CPU and the cache SRAM, a cache controller is essential. This intermediary component connects the CPU and the cache memory while also providing access to the main memory. In essence, this concept, known as the *principle of locality*, underpins the efficient utilization of diverse memory systems and contributes to overall performance enhancements. The diagram below illustrates the hierarchical structure of the CPU and memory in a typical setup:

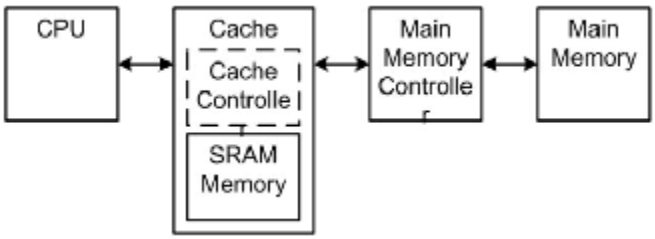

Figure 1: Hierarchical structure of CPU interfacing with Cache and other Memory blocks

In this project, we successfully incorporated the cache controller into an FPGA, utilizing Xilinx ISE and creating VHDL modules. After the VHDL modules were finalized, we rigorously tested them through simulations before seamlessly integrating them using a schematic file. The visual representation provided in the following figure illustrates the anticipated outcome of this implementation:

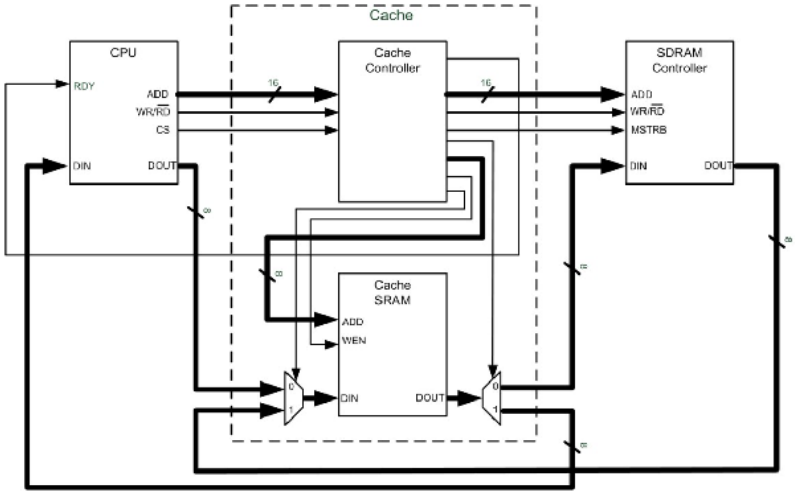

Figure 2: Structure of CPU interfacing with Cache and other modules

We were already supplied with the VHDL code for the CPU, but we needed to develop the Cache controller, SRAM, and SDRAM modules. To create the SRAM (Block RAM) module, we utilized the Xilinx memory generator software. Figure 2 illustrates the process whereby the CPU sends instructions to the cache controller, and subsequently, the cache controller interacts with the SRAM memory for read or write operations. Furthermore, the SDRAM controller transmits data to the CPU.

Specification

The project aims to design a logical circuit capable of effectively responding to read and write requests from the CPU. The cache controller has two primary options: retrieving complete data blocks from the main memory or conducting read/write operations within the cache memory.

The cache memory in this project is organized into eight blocks, each consisting of 32 words. Address words, which are 16 bits in length, are segmented into three components: tag (8-15), offset (0-3), and index (4-7). The tag field serves as a distinct identifier, containing essential address information for determining whether the corresponding block in the hierarchy matches the requested word. Meanwhile, the index and block offset fields play a crucial role in pinpointing the precise block and word within the cache memory that the CPU requires. Figure 3 provides a detailed breakdown of the cache memory structure:

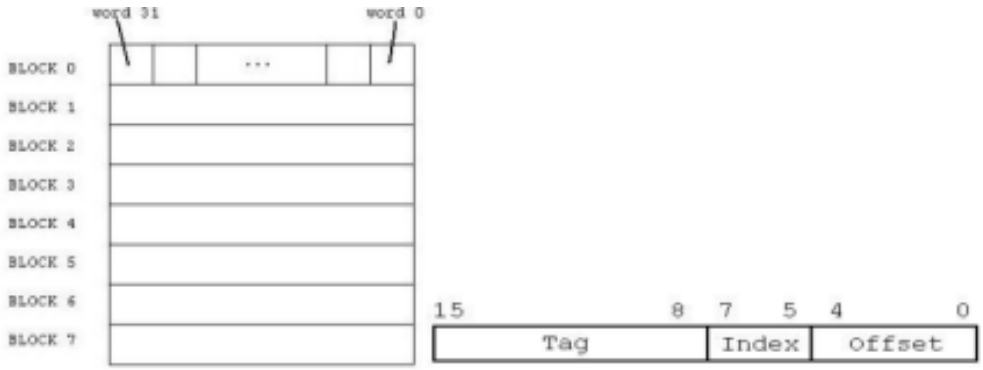

Figure 3: Organization of Cache memory, & Address Word Register fields

The controller determines its course of action by referencing its internal indicators, which provide insight into the status of each cache memory block. Specifically, the controller needs to assess the state of the dirty and valid bits associated with the target block and compare the address's tag component with the recorded tags for all eight blocks.

In order to facilitate data transfers between the main memory and the cache memory, we need to examine four distinct operational scenarios:

Case 1: Reading from Cache (Hit): In this case, when the CPU requests a word from the cache memory (a hit), it sends a read signal to the cache controller. The cache controller reads the cache memory, forwards the information to the SRAM controller, and the data is then transmitted to the CPU.

Case 2: Writing to Cache (Hit): When the CPU initiates a write operation to the cache, the cache controller receives a write signal. Data is written to the cache memory, resulting in a cache hit. The 16-bit address's index and offset components determine the location within the cache memory where the data will be stored. As data is written to the cache, the dirty bit is set to 1 to signify that it's been modified. Additionally, the valid bit is set to 1 to validate the data.

Case 3: Read/Write to Cache (Miss) if Dirty Bit = 0: In this scenario, the cache lacks the requested data despite receiving read/write signals from the CPU. When the dirty bit is 0, the SDRAM controller receives an offset of "0000," indicating the starting base address of the block with its specific index. The SDRAM controller reads and writes the block to the cache SRAM memory. The

tag address of the following register is replaced with the tag portion of the address, with the valid bit set to 1.

Case 4: Read/Write to Cache (Miss) if Dirty Bit = 1: Here, the CPU attempts to read/write to the cache controller, but the data is not present in the cache memory, resulting in a miss. The dirty bit is set to 1, indicating that the data block in the SRAM has been recently modified and needs to be written back into the main memory using the SDRAM memory controller. The memory block is written to the base address with tag+index+'0000', where the tag and index correspond to the 32-bit address, along with offset '0000'. The new block, based on the address requested by the CPU, is then copied to the cache memory block, and the SDRAM controller retrieves this data using the tag+index+'0000' address. The tag corresponds to the Address Word Register requested by the CPU in this case. Once this process is complete, the CPU's original request can be fulfilled.

Device Design

Symbols

Below are the simplified symbols representing the necessary components for constructing the cache controller, complete with their corresponding input and output pins:

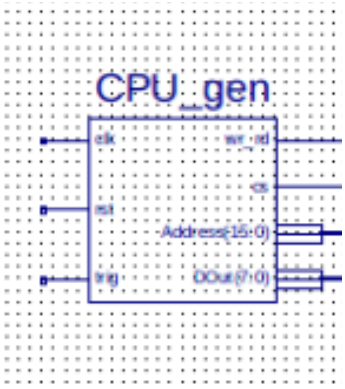

Figure 4: CPU interface

Figure 5: Cache Controller interface

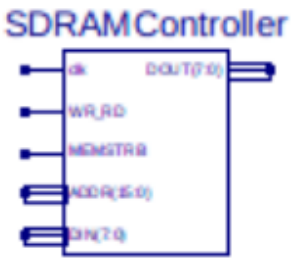

Figure 6: SDRAM Controller interface

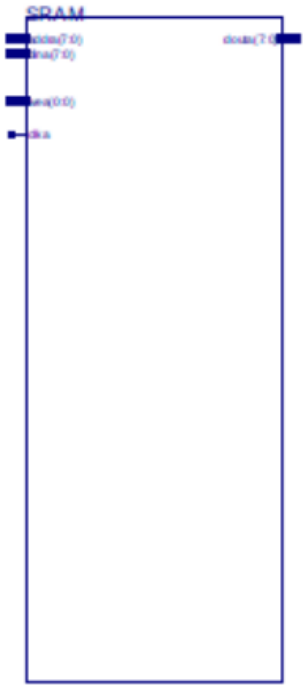

Figure 7: Block RAM (SRAM) interface

Block Diagram

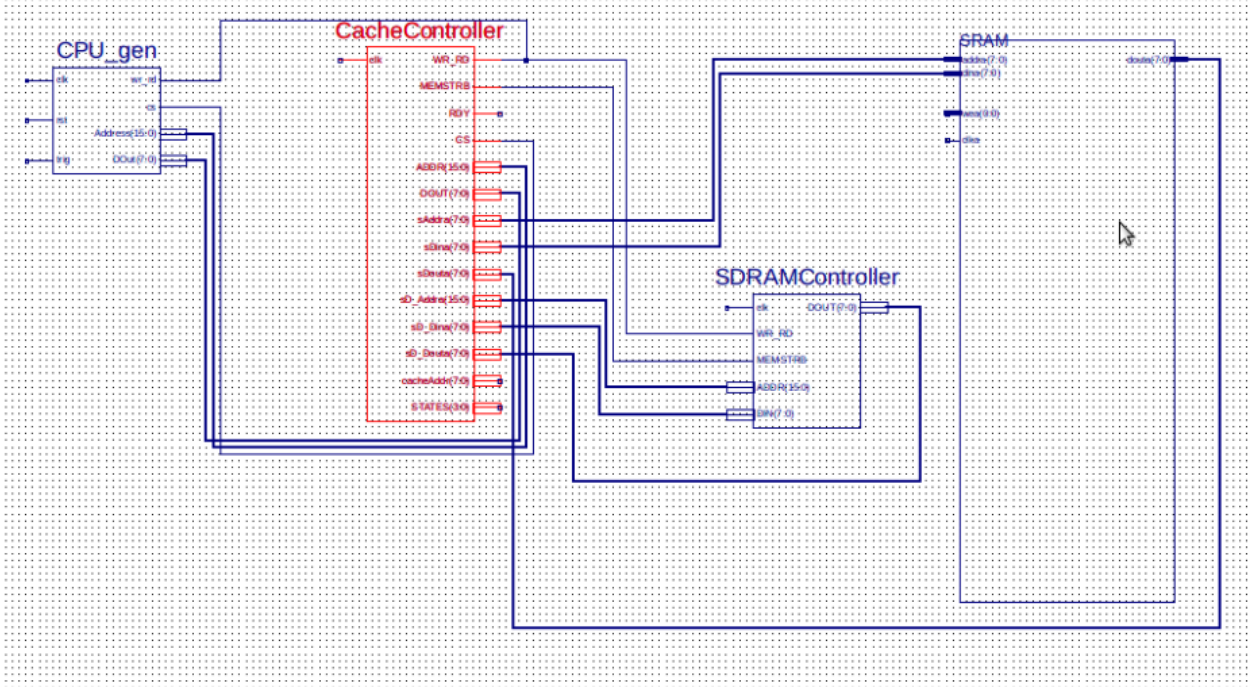

Figure 8: A schematic diagram that illustrates the interconnections between each component, forming the foundation of the project.

State Diagram

The following is the state diagram of the cache controller:

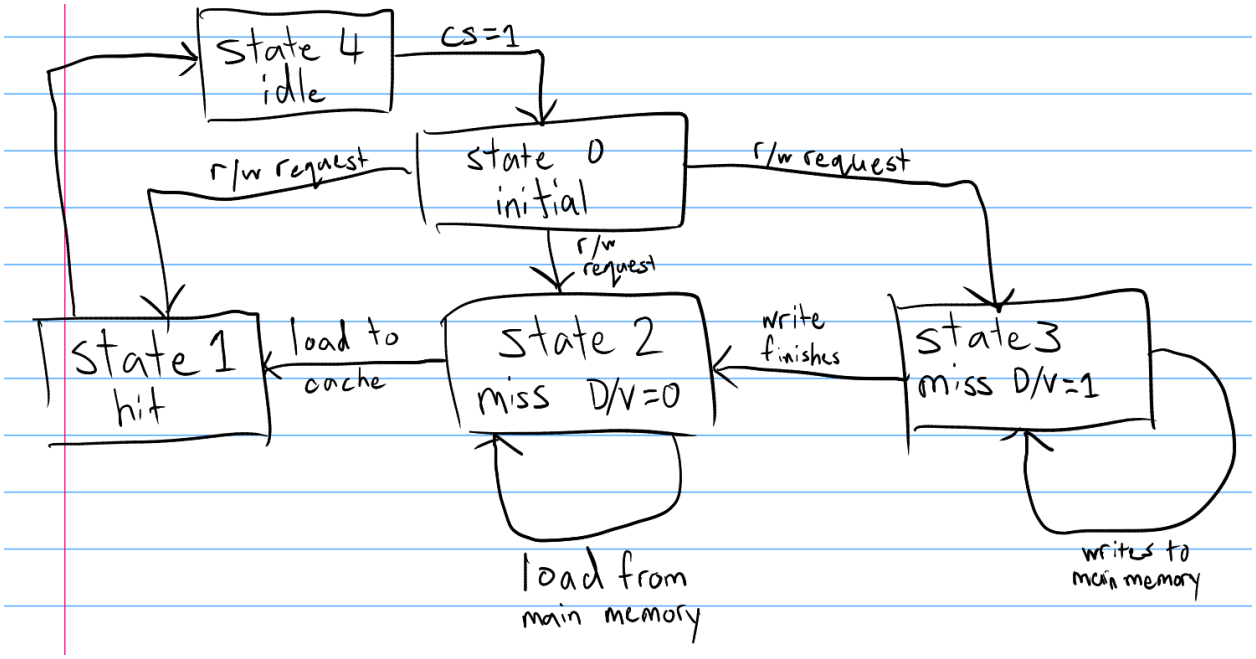

Figure 9: A state diagram depicting the Cache Controller's operational states and its interactions with the Main Memory and CPU.

Process Diagram

The following is the process diagram of the cache controller:

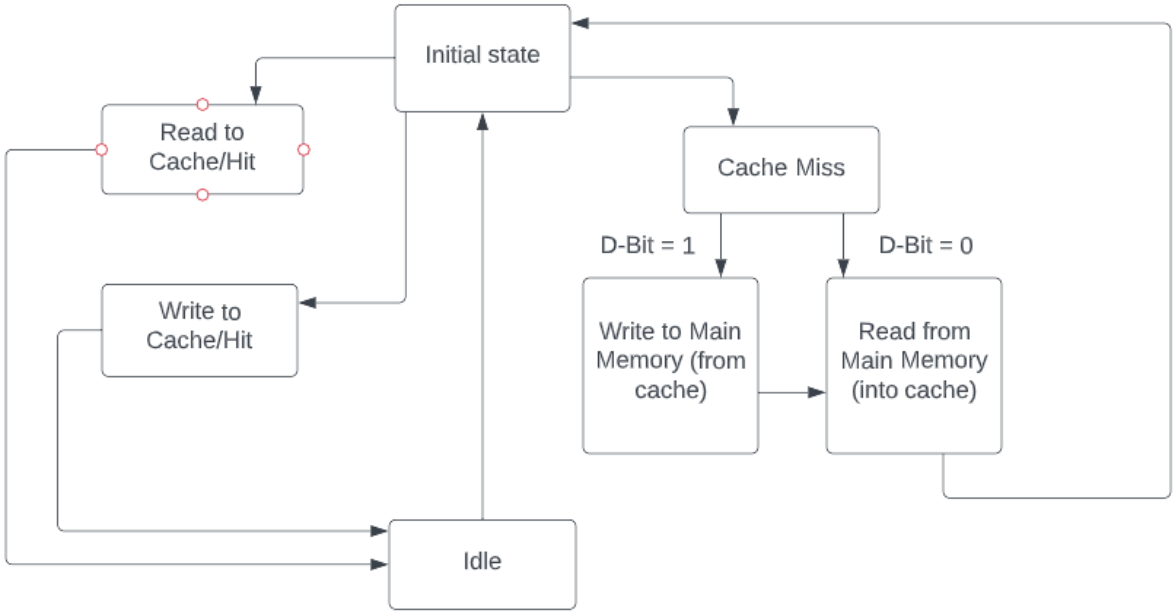

Figure 10: A process diagram illustrating the Cache Controller's functions and its interactions with both the Main Memory and CPU

Results

Functional/Timing Diagrams

We can gain insight into the interactions between the cache controller, cache memory, CPU, and SDRAM by conducting functional simulations. This implementation encompasses four distinct states.

State 0 serves as the initial state, where the Cache controller receives an address from the CPU and employs its comparative logic to ascertain whether it's a cache hit or miss. State 1 corresponds to the state following a cache hit. States 2 and 3 represent miss scenarios, with the former signifying the condition where the dirty/valid bit equals 1, prompting the loading of the required block from the main memory. The latter, State 3, signifies the scenario where the dirty/valid bit equals 1, leading to the writing of data back into the main memory. Lastly, State 4 represents the idle state, during which the cache controller awaits instructions from the CPU.

Below are the functional simulation diagrams corresponding to each of the behavioral scenarios outlined in the specification section of the report for the cache controller:

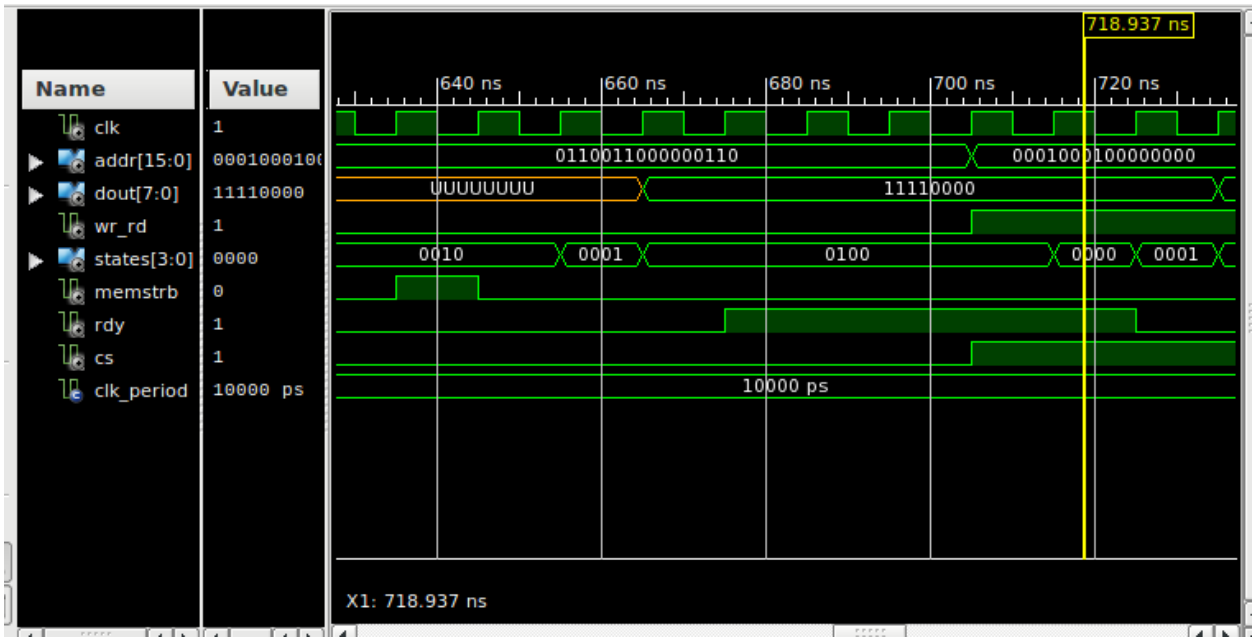

Figure 10: Going from state 2 (miss w/ V bit/ D bit =0) to state 1 (hit) to state 4(idle)

In Figure 10 above, a read operation is indicated by a low wr_rd signal. Initially, no data is present in the cache, leading to a cache miss (with V bit/D bit = 0). This triggers behavioral case 4, where the SDRAM writes the necessary block into the SRAM memory. Upon successful completion, the state transitions to state 1, indicating a cache hit (case 1), and the data is transmitted to the CPU. Subsequently, the cache enters an idle state, ready to receive the next instruction from the CPU.

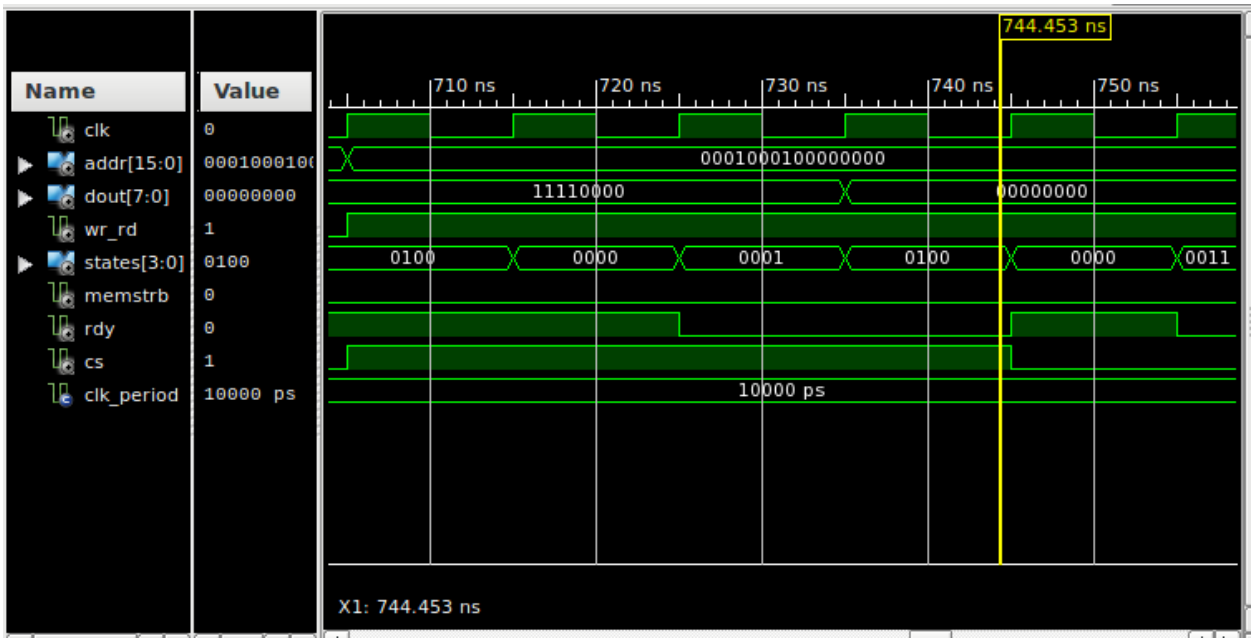

Figure 11: Going from state 4 (idle) to state 0 (initial) then going from state 1 (hit) to state 4(idle) to state 0 (initial)

In Figure 11 above, the cache controller initiates in an idle state, anticipating the next instruction from the CPU. Upon receiving this instruction, it transitions to state 0 and executes its comparative logic. In this instance, a write request is detected (indicated by a high wr_rd signal). As the data is already present in the cache, the state shifts to state 1, signifying a cache hit (case 1), and the data is forwarded to the CPU. Following this operation, the cache returns to an idle state, ready to receive the subsequent CPU instruction.

In the following Figure 12, the situation where the particular word is not present in the cache and the block itself has been altered (resulting in d/v bit = 1) triggers case 4. The sequence commences in an idle state (state 4), anticipating the next CPU instruction. Upon receipt of the instruction, it transitions to state 0, initiating the comparative logic. Given the absence of the data and its prior modification within the cache, a 'write-back' process occurs, leading the state to switch to state 3. Once that specific block has been written back into the main memory, the required block can now be retrieved, which entails setting v/d bit = 0 and executing case 3. Consequently, the state reverts to state 2. The data is loaded into the cache, resulting in a cache hit, and the data is dispatched to the CPU. Following this operation, the cache returns to an idle state, ready to accept the next CPU instruction once more.

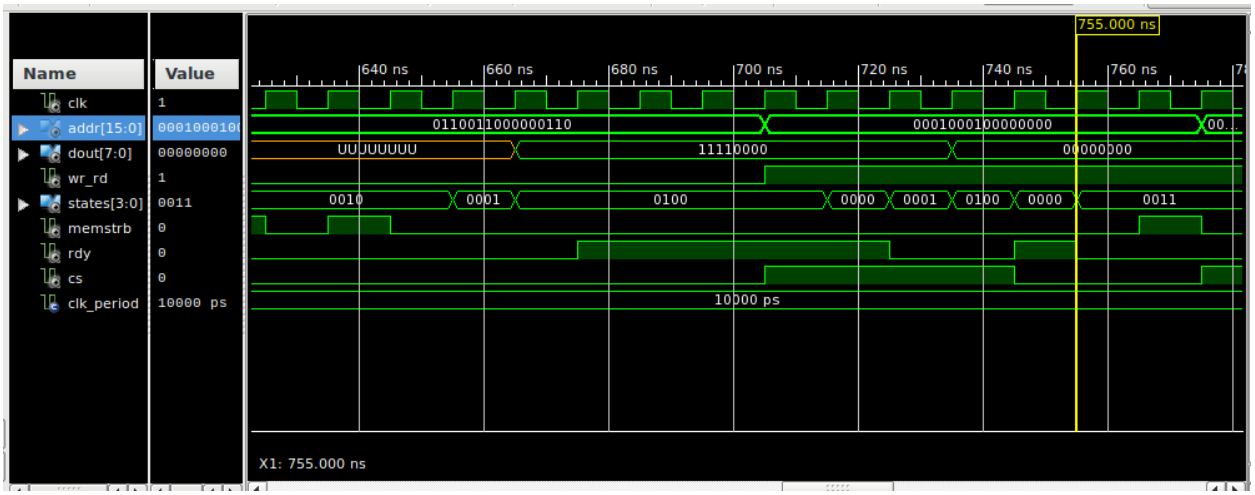

Figure 12 (a): Going from state 3 (miss w/ V-bit+Dbit=1) to state 2 (miss w/ V bit/ D bit =0) to state 1 (hit) to state 4(idle) to state 0 (initial)

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 1 to state 4 , 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

In conclusion, the project has been successfully accomplished. We constructed a block diagram using VHDL code, which was compiled successfully for each constituent component. Subsequently, we generated simulated waveforms based on the block diagram, revealing all potential interactions between the CPU and the Cache controller.

The VHDL code for each component is accessible below. These waveforms were observed in action by the TA, and they appeared to function as intended. The project significantly enhanced our understanding of the connections and engagements among the cache memory, CPU, and SDRAM controller. In sum, the project met expectations and objectives effectively.

References

1. Toronto Metropolitan University. 2023. Course Content: Design project 1. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System. https://courses.torontomu.ca/d2l/le/content/792271/Home

2. Toronto Metropolitan University. 2023. Course Content: COE758_Digital_Design_Tutorial. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System. https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5325205/View

3. Toronto Metropolitan University. 2023. Course Content: Cache Project[12-09-10]. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System. https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5322256/View

4. Toronto Metropolitan University. 2023. Course Content: P1_interfaces_doc_2011_09_15. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System. https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5323145/View

5. Toronto Metropolitan University. 2023. Course Content: Project 1 requirements. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's

Learning Management System.
https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5322257/View

6. Toronto Metropolitan University. 2023. Course Content: Project1_report_outline. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System.
https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5322259/View

7. Toronto Metropolitan University. 2023. Course Content: CPU_gen. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System.
https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5323148/View

Appendix

7. Appendix (code)

```
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);
                           sAddra    : 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);
                           STATES          : out STD_LOGIC_VECTOR(3 downto 0);
                           WR_RD, MEMSTRB, RDY ,CS         : out  STD_LOGIC);
end CacheController;

architecture Behavioral of CacheController is

--States
TYPE state_value IS (state0, state1, state2, state3, state4);
```

```
signal current_state            : state_value;
signal state                            : STD_LOGIC_VECTOR (3 downto 0);

--CPU_gen signals
signal CPU_RDY                  :STD_LOGIC;
signal CPU_ADD                  :STD_LOGIC_VECTOR (15 downto 0);
signal CPU_RW                   :STD_LOGIC;
signal CPU_CS                   :STD_LOGIC;
signal CPU_DOUT                 :STD_LOGIC_VECTOR (7 downto 0);

--SDRAM_Controller signals
signal SDRAM_ADD                :STD_LOGIC_VECTOR (15 downto 0);
signal SDRAM_RW                 :STD_LOGIC;
signal SDRAM_MEMSTRB            :STD_LOGIC;
signal SDRAM_DIN                :STD_LOGIC_VECTOR (7 downto 0);
signal SDRAM_DOUT                       :STD_LOGIC_VECTOR (7 downto 0);

--BlockRAM signals
signal SRAM_wen                 :STD_LOGIC_VECTOR(0 DOWNTO 0);
signal SRAM_ADD                 :STD_LOGIC_VECTOR(7 DOWNTO 0);
signal SRAM_DIN                 :STD_LOGIC_VECTOR(7 DOWNTO 0);
signal SRAM_DOUT                :STD_LOGIC_VECTOR(7 DOWNTO 0);

signal CPU_TAG                  :STD_LOGIC_VECTOR(7 downto 0);
signal CPU_DIN                  :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 DBIT                     :STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal VBIT                     :STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal counter                  :integer := 0;
signal sdoffset                 :integer := 0;
signal tagwen                   :STD_LOGIC := '0';

--ICON and 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);

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

COMPONENT CPU_gen
        Port (
```

```
                clk             : in  STD_LOGIC;
    rst         : in  STD_LOGIC;
    trig                : in  STD_LOGIC;
                -- Interface to the Cache Controller.
    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 BlockRAM
  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 SDRAM_Controller
    Port ( CLK : in  STD_LOGIC;
           ADD : 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 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
```

```
CPU: CPU_gen Port Map (
                clk => clk,
    rst => '0',
    trig => CPU_RDY,
    Address => CPU_ADD,
    wr_rd => CPU_RW,
    cs => CPU_CS,
    DOut => CPU_DOUT
        );

SDRAM: SDRAM_Controller Port Map (
                CLK => clk,
                ADD => SDRAM_ADD,
                WR_RD => SDRAM_RW,
                MEMSTRB => SDRAM_MEMSTRB,
                DIN => SDRAM_DIN,
                DOUT => SDRAM_DOUT
        );

SRAM : BlockRAM
  PORT MAP (
    clka => clk,
    wea => SRAM_WEN,
    addra => SRAM_ADD,
    dina => SRAM_DIN,
    douta => SRAM_DOUT
  );

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
                        --initial state
                        if (current_state = state0) 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);
                                SRAM_ADD(7 downto 0) <= CPU_ADD(7 downto 0);
                                SRAM_WEN <= "0";
```

```
                        if(VBIT(to_integer(unsigned(index))) = '1' AND
memtag(to_integer(unsigned(index))) = CPU_TAG) then
                                TAGWEN <= '1';
                                current_state <= state1;
                                state <= "0001";
                        else
                                if(VBIT(to_integer(unsigned(index))) = '1' AND
DBIT(to_integer(unsigned(index))) = '1') then
                                        current_state <= state3;
                                        state <= "0011";
                                else
                                        current_state <= state2;
                                        state <= "0010";
                                end if;
                        end if;
                --hit state
                elsif (current_state = state1) then
                        if (CPU_RW = '1') then
                                SRAM_WEN <= "1";
                                DBIT(to_integer(unsigned(index))) <= '1';
                                VBIT(to_integer(unsigned(index))) <= '1';
                                SRAM_DIN <= CPU_DOUT;
                                CPU_DIN <= "00000000";
                        else
                                CPU_DIN <= SRAM_DOUT;
                        end if;
                        current_state <= state4;
                        state <= "0100";
                --miss with dbit = 0
                elsif (current_state = state2) then
                        if (counter = 64) then
                                counter <= 0;
                                VBIT(to_integer(unsigned(index))) <= '1';
                                memtag(to_integer(unsigned(index))) <= cpu_tag;
                                sdoffset <= 0;
                                current_state <= state1;
                                state <= "0001";
                        else
                                if (counter mod 2 = 1) then
                                        SDRAM_MEMSTRB <= '0';
                                else
                                        SDRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
                                        SDRAM_RW <= '0';
```

```
                                        SDRAM_MEMSTRB <= '1';
                                        SRAM_ADD(7 downto 5) <= index;
                                        SRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
                                        SRAM_DIN <= SDRAM_DOUT;
                                        SRAM_WEN <= "1";
                                        sdoffset <= sdoffset + 1;
                                end if;
                                counter <= counter + 1;
                        end if;
                --miss with dbit = 1
                elsif (current_state = state3) then
                        if (counter = 64) then
                                counter <= 0;
                                DBIT(to_integer(unsigned(index))) <= '0';
                                sdoffset <= 0;
                                current_state <= state2;
                                state <= "0010";
                        else
                                if (counter mod 2 = 1) then
                                        SDRAM_MEMSTRB <= '0';
                                else
                                        SDRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
                                        SDRAM_RW <= '1';
                                        SRAM_ADD(7 downto 5) <= index;
                                        SRAM_ADD(4 downto 0) <=
STD_LOGIC_VECTOR(to_unsigned(sdoffset, offset'length));
                                        SRAM_WEN <= "0";
                                        SDRAM_DIN <= SRAM_DOUT;
                                        SDRAM_MEMSTRB <= '1';
                                        sdoffset <= sdoffset + 1;
                                end if;
                                counter <= counter + 1;
                        end if;
                --idle state
                elsif (current_state = state4) then
                        CPU_RDY <= '1';
                        if (CPU_CS = '1') then
                                current_state <= state0;
                                state <= "0000";
                        end if;
                end if;
        end if;
```

```
end process;

STATES <= state;
MEMSTRB <= SDRAM_MEMSTRB;
ADDR <= CPU_ADD;
WR_RD <= CPU_RW;
DOUT <= CPU_DIN;
RDY <= CPU_RDY;
CS <= CPU_CS;

sAddra <= SRAM_ADD;
sDina <= SRAM_DIN;
sDouta <= SRAM_DOUT;

sD_Addra <= SDRAM_ADD;
sD_Dina <= SDRAM_DIN;
sD_Douta <= SDRAM_DOUT;

cacheAddr <= CPU_ADD(15 downto 8);

ila_data(15 downto 0) <= CPU_ADD;
ila_data(16) <= CPU_RW;
ila_data(17) <= CPU_RDY;
ila_data(18) <= SDRAM_MEMSTRB;
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) <= SRAM_ADD;
ila_data(50 downto 43) <= SRAM_DIN;
ila_data(58 downto 51) <= SRAM_DOUT;
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;
```