Toronto Metropolitan University

Faculty of Engineering and Architectural Science (FEAS)

Department of Electrical and Computer Engineering

Course Number	**COE758**
Course Title	Digital Systems Engineering
Semester/Year	Fall/2023

Instructor	Dr. Vadim Geurkov

ASSIGNMENT No.	**1**

Assignment Title	Project #1 – Memory Hierarchy: Cache Controller

Submission Date	Sunday, November 5, 2023
Due Date	Sunday, November 5, 2023 11:59 PM

Student Name	**Dexter Ryan Floreza**	**Adam Szava**
Student ID	500946679	501046722
Signature*	Dexter Ryan Floreza	Adam Szava

**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: www.ryerson.ca/senate/current/pol60.pdf.*

Table of Contents

Abstract..... 2
Objective..... 2
Approach..... 2
Results..... 2
Introduction..... 2
Motivation..... 2
Objective..... 3
Theory..... 3
Device Design Description..... 7
Symbols..... 7
Block Diagram (Complete)..... 8
State Diagram..... 9
Functional Results..... 9
Timing Results..... 12
Conclusion..... 14
References..... 15
Appendix with VHDL Code..... 16

Abstract

Objective

This design project had several objectives. The first objective was to learn the functionality of a cache controller and its interaction with block-memory (SRAM based) and SDRAM-controllers. The second objective was to get practical experience in the design and implementation of custom logic controllers and interfacing to SRAM memory units and other logic devices. The final objective was to learn VHDL-coding technique within the Xilinx ISE CAD environment of the system and a hardware evaluation platform based on the Xilinx Spartan-3E FPGA.

Approach

The Cache Controller was implemented in VHDL. The VHDL implementation for the CPU was already given, while everything else was designed from scratch.

Prior to VHDL implementation, the schematic diagram, block diagram, and a conceptual finite state machine were drawn by hand. This resulted in the creation of three reference diagrams to help understand the nature of the project. After that, component symbols were created and VHDL implementations of all the components were created. Several debugging variables and testbenches were created within each file to test functionality and correctness. Using Xilinx ISE, the software simulated several timing diagrams for analysis.

Results

The results of this design project are summarized in Table 1. Most of the timing parameters were measured using the Xilinx software. The only timing parameters calculated were the Miss Penalties for different values of D-bits.

Introduction

Motivation

Memory hierarchies in digital systems play a pivotal role in bridging the gap between high-speed processing capabilities of CPUs and the slower main memory component. In modern computer architecture technology, the concept of a memory hierarchy remains a fundamental concept in the advancement of this field. Significant technological improvements in system performance and energy efficiency are all a result of optimal memory hierarchy designs. Furthermore, as technology continues to grow, it becomes increasingly clear that there exists a need for efficient memory

management and faster data communications. On a fundamental level, it is crucial for students to grasp the workings of memory hierarchies in managing delays related to slower yet high-capacity memory components, all the while considering the interplay between memory capacity, memory access speed, and their proximity to the processor.

Objective

The main objectives of this lab were to:

- To learn the functionality of a cache controller and its interaction with block-memory (SRAM based) and SDRAM-controllers.
- To get practical experience in the design and implementation of custom logic controllers and interfacing to SRAM memory units and other logic devices.
- To learn VHDL-coding technique within the Xilinx ISE CAD environment of the system and a hardware evaluation platform based on the Xilinx Spartan-3E FPGA.

These objectives provide a structured framework for the experiment, guiding the investigation of cache controller functionality, memory interactions, custom logic design, and VHDL coding, while also emphasizing practical experience with the Xilinx Spartan-3E FPGA platform.

Theory

Computer systems are designed with a memory hierarchy to efficiently manage memory. This memory hierarchy consists of several levels, with the Cache Memory being the closest to the CPU. The Cache Memory stores small amounts of data recently accessed in the form of blocks, with each containing a number of words. The Cache Controller stores identifiers for the blocks currently stored in the Cache, which include several key bits such as: the index, tag, valid, and dirty bits, associated with a whole block of data. The aim of this project was to design and implement a Cache Controller for a hypothetical computer system that stored a total of 256 bytes [1].

The different behavioural cases involved with the CPU and Cache Controller are as follows:

1. Write a word to cache [hit]

When the CPU sends a write request to the cache, and the requested data is found in the cache (a cache hit), the cache updates the local SRAM memory by using the index and offset portions of the CPU's address as the write address for the new data.

Additionally, the dirty and valid bits for the specific data block are set to 1, and the data is written to the local SRAM memory.

2. Read a word from cache [hit]

When the CPU sends a read request to the cache, and the requested data is found in the cache (a cache hit), the cache forwards the index and offset portions of the CPU's address to the local SRAM and retrieves the requested data, sending it back to the CPU.

3. Read/Write from/to cache [miss] and dirty bit = 0

When a read or write request is received and the associated block is not found in the cache, a block replacement action in the Cache is required. First, we check the dirty bit for the corresponding CPU address. If the dirty bit is 0, use the entire CPU address with the offset portion set to “00000” as the base address for the block to be read from memory (SDRAM). Read the full block (32 bytes) from the SDRAM memory controller and write it to the local Cache SRAM. Replace the tag value in the cache’s tag register with the tag from the CPU address and set the valid bit to 1. Finally, perform the requested read or write operation following the appropriate procedure as if it were a cache hit [1].

4. Read/Write from/to cache [miss] and dirty bit = 1

When a cache miss occurs, if the dirty bit is set to 1, the current block is written back to main memory through the SDRAM memory controller at the address [Tag & Index & 00000]. The requested block is then fetched from main memory at the same address [Tag & Index & 00000], and the cache is updated with the new data and tag before executing the CPU's original request.

As stated above, the cache controller interacts with the CPU in several behavioural cases. The CPU itself consists of a strobe CS, a read/write indicator WR/RD, a 16-bit address ADD, 8-bit data input and output ports DIN and DOUT, and a ready indicator input RDY [1].

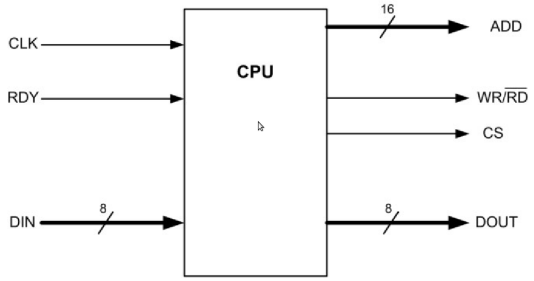


Figure 1: CPU Interface

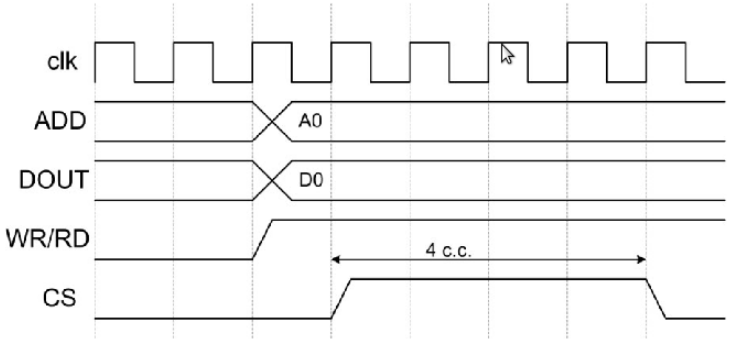


Figure 2: CPU Transaction Example

As mentioned previously, the Cache Controller, shown in **Figure 3**, is situated between the CPU and the SDRAM Controller. The SDRAM Controller is synchronized with the Cache Controller and responds to block read and write requests from it. It consists of a 16-bit address ADD, a read/write indicator WR/RD, a strobe MEMSTRB, and data input and output ports DIN and DOUT [1].

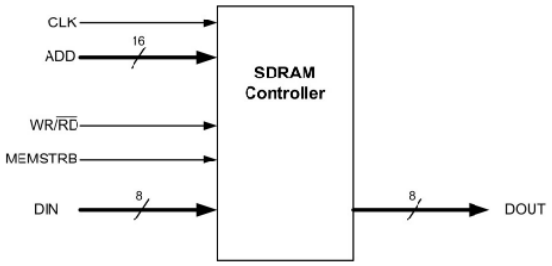


Figure 3: SDRAM Controller Interface

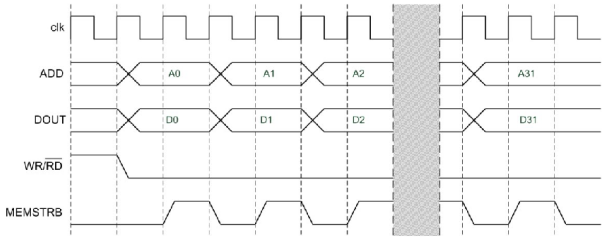


Figure 4: SDRAM Block Read

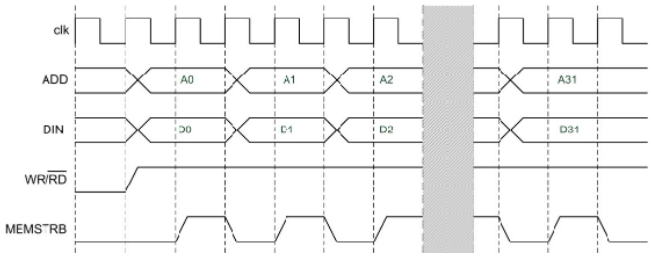


Figure 5: SDRAM Block Write

In **Figures 4 and 5**, an example of an SDRAM Block Read and Block Write operation is shown. Graphically, the difference between the two operations is that for Block Reads, WR/RD is held low, with no data being needed in the DIN port, whereas for Block Writes, the WR/RD signal is held high with the MEMSTRB being asserted for one clock cycle, while also repeating the process 32 times to write a full block to memory.

Finally, the BlockRAM, as shown below in **Figure 6**. is used to implement the local memory for the cache. It is synchronized with the Cache Controller, sharing the same clock, and consists of an 8-bit address ADD, 8-bit data input and output DIN and DOUT, and a write enable signal WEN [1].

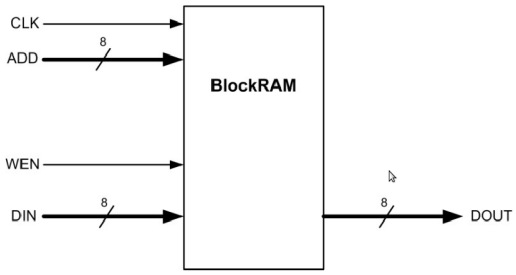


Figure 6: BlockRAM Interface

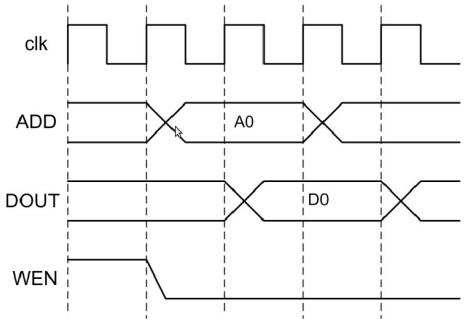


Figure 7: BlockRAM Read Example

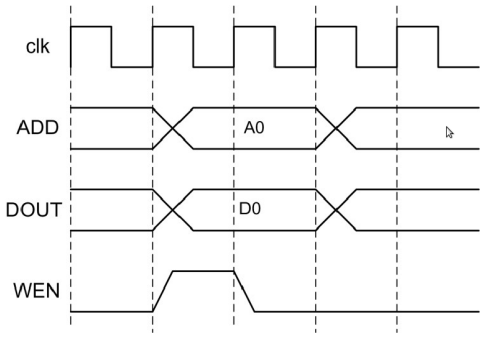


Figure 8: BlockRAM Write Example

In **Figure 7** and **Figure 8,** BlockRAM Read and Write operations are shown. Graphically, the Read operations are done by first setting the appropriate address on the address bus, then propagating the addressed data to the output DOUT after the next rising edge of the clock, whereas Write operations are performed by setting a specific address and data on the appropriate ports, and then asserting the write enable signal WEN, with the data being written to the specified address on the next rising edge [1].

Device Design Description

Symbols

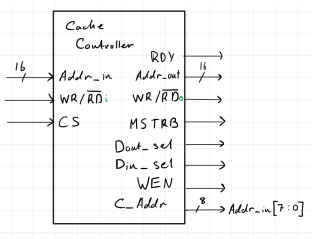


Figure 9: Cache Controller Symbol.

In **Figure 9**, shown above is the symbol for the Cache Controller. The inputs consist of a 16-bit address input, a WR/RD input, and a strobe CS. The output ports to the SDRAM controller are: a 16-bit Address Output, a WR/RD output, and a Memory Strobe (MSTRB). The outputs headed towards the Cache SRAM are: an 8-bit C_Addr and a Write Enable indicator. The other output includes a Dout_sel towards a Demultiplexer.

Block Diagram (Complete)

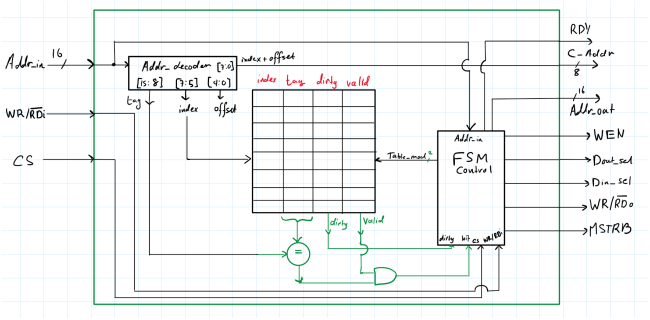


Figure 10: Block Diagram of Cache Controller [4]

The block diagram of the Cache Controller shown in **Figure 10** shows the component's internal behaviour. The 16-bit Input Address is connected to both the Address Decoder and the FSM control. The Address Decoder processes the request by dividing the request address into three fields: the tag field, the index field, and the offset field. The tag field is used as a unique identifier for a group of data. The index field is then sent to a table, which is used to locate the cache line within the cache memory that might hold the requested data. The offset simply corresponds to a particular location within a cache line. The table also includes two columns labelled "dirty" and "valid." A valid bit specifies whether there exists something stored in the cache entry, while a dirty bit specifies whether the byte was written on since it was brought into memory. The value of the valid bit is sent to a logical AND gate with the result of the address and the tag compared. The result of the AND operation is then sent into the FSM control's "bit" input port. From there, the FSM control will output the following signals: RDY, Addr_out, WEN, Din_sel, Dout_sel, WR/RDo, and MSTRB.

State Diagram

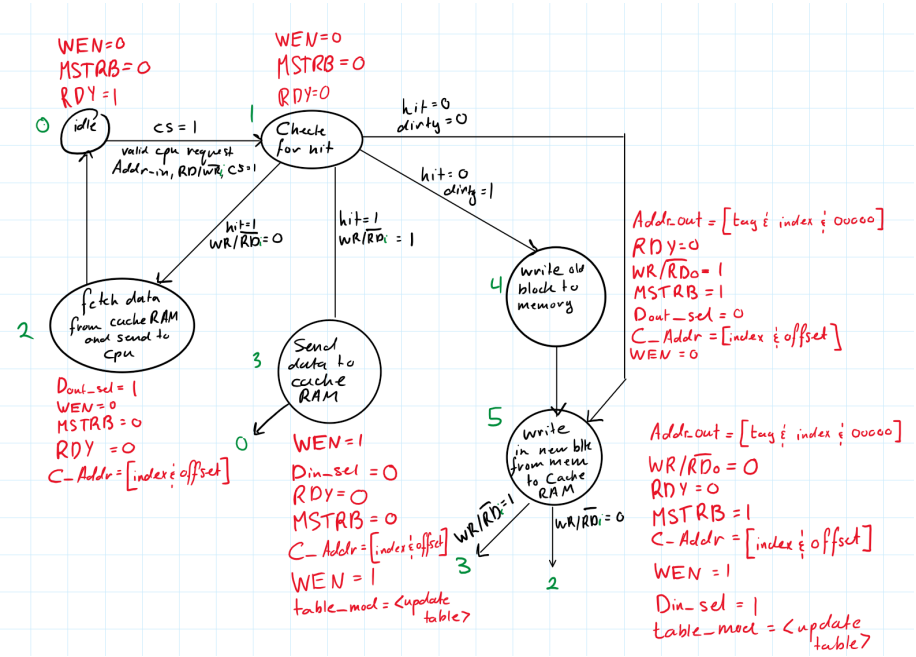


Figure 11: Finite State Machine (FSM) for Cache Controller

The Finite State Machine (FSM) shown in Figure 11 illustrates the various states and transitions involved in the caching process. In this diagram, there are six states, labeled from 0 through 5, each with corresponding conditions for each transition and control signals (written in red) associated with each state. The FSM largely corresponds to the Behavioural Cases discussed in the Theory section of this lab report. However, it also accounts for an initial “idle” state not explicitly mentioned, as well as table updates, denoted as “table_mod = <update table>”.

Functional Results

Example 1

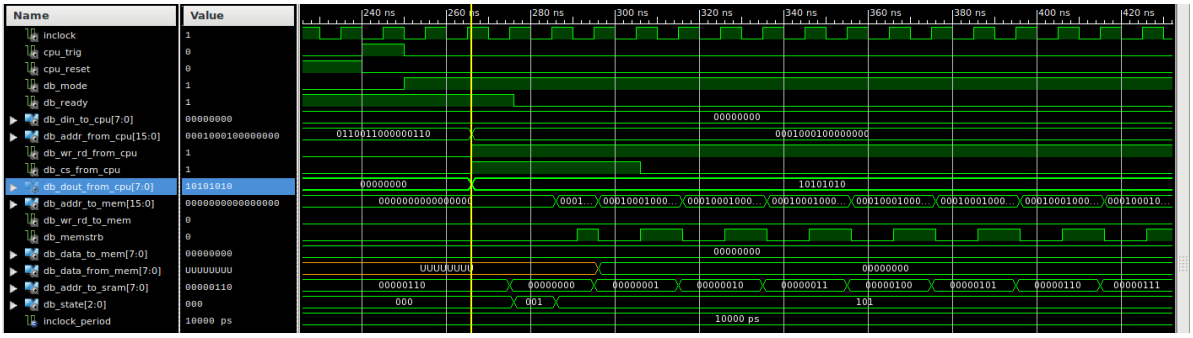


Figure 12: Functional Simulation for the process: State 0 to State 1, State 1 to State 5.

The above figure shows the result of the following instruction:

Tag	Index	Offset	R/W
0001 0001	000	00000	W

The cache begins in state 0 (idle) from which it goes into state 1 (hit/miss determination). Since the cache is fresh, it's a miss (dirty bit = 0), and so the cache goes into state 5. This triggers 32 memory transfers from memory into the cache SRAM. Once this process is done, the simulation diagram continues as:

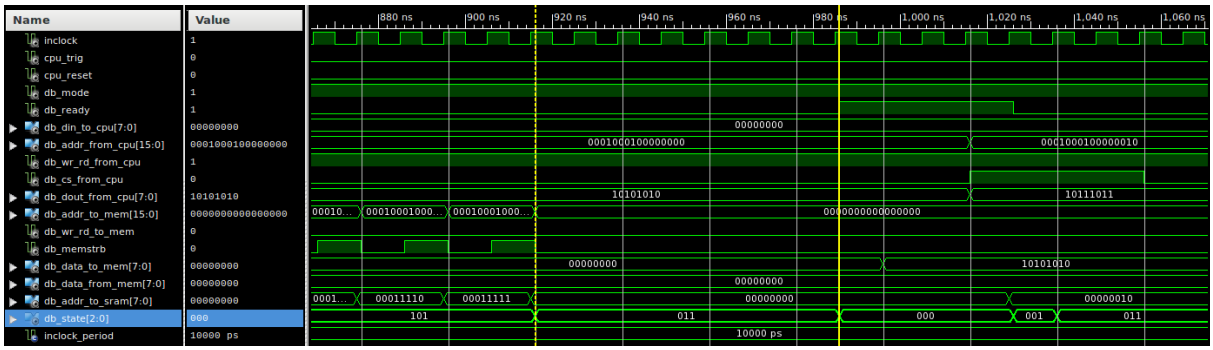


Figure 13: Functional Simulation for State 5 to State 3, then State 3 to State 0.

Once the memory block transfer is complete, the write needs to be serviced, which is done in state 2. Finally, the cache returns to state 0.

Example 2

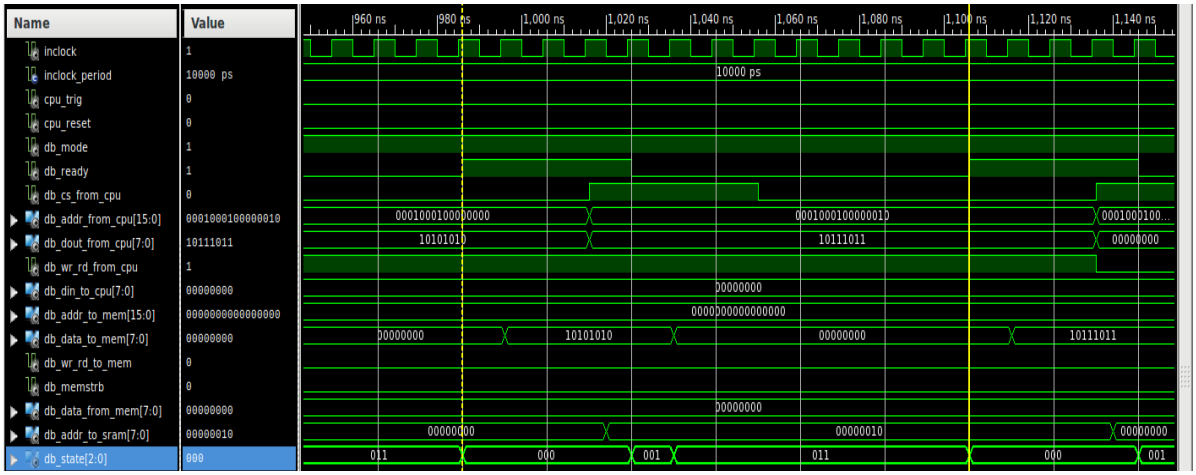


Figure 14: Functional Simulation for State 1 to State 3.

The above figure shows the result of the following instruction:

Tag	Index	Offset	R/W
0001 0001	000	00010	W

The cache begins in state 0 (idle) from which it goes into state 1 (hit/miss determination). Due to the results of **example 1**, this is a hit, and so the cache enters state 3 (write to cache).

Example 3

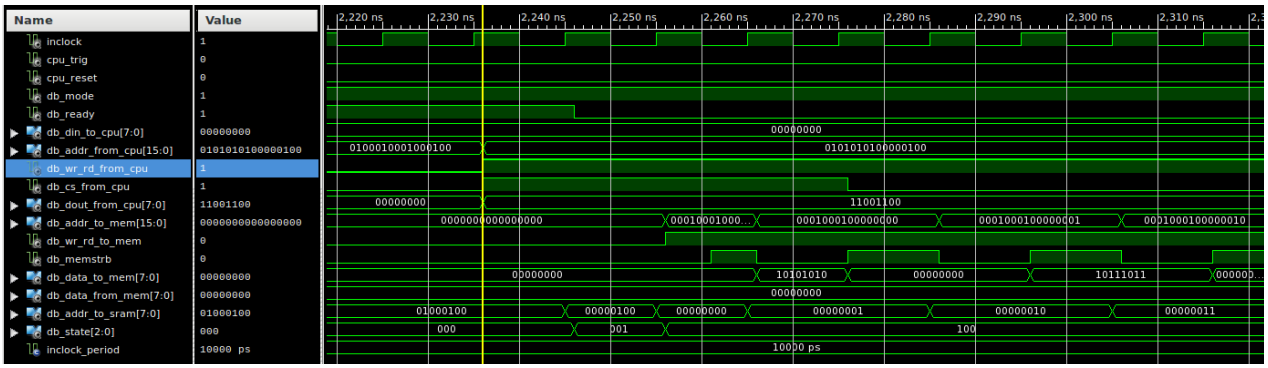


Figure 15: Functional Simulation for State 0 to State 1, State 1 to State 4, State 4 to State 5 (not shown).

The above figure shows the result of the following instruction:

Tag	Index	Offset	R/W
0101 0101	000	00100	W

The cache begins in state 0 (idle) from which it goes into state 1 (hit/miss determination). Due to the results of **example 1 & 2**, this is a miss with dirty bit =1, and so the cache enters state 4 (transfer block to memory). This triggers 32 transfers to occur, starting at the base address of the old block. Once this is complete the cache enters state 5 (transfer block from memory) which again triggers 32 transfers, starting at the base address of the new block. The tail end of these 64 transfers, as well as the caches service of the write (state 3), and return to idle (state 0) can be seen below:

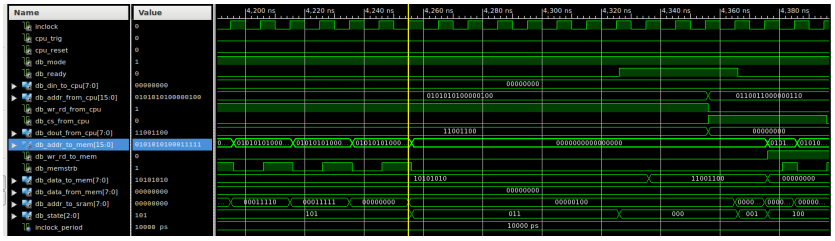


Figure 16: Functional Results for State 5 to State 3 and then State 3 to State 0.

Timing Results

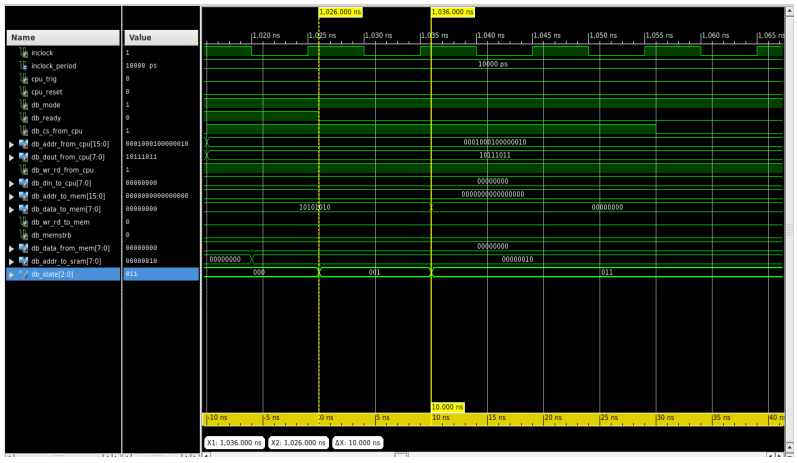


Figure 17: Timing Diagram showing Hit Determination Time

Figure 17 shows the Hit Determination Time. This is equivalent to the time spent in State 1 of the FSM.

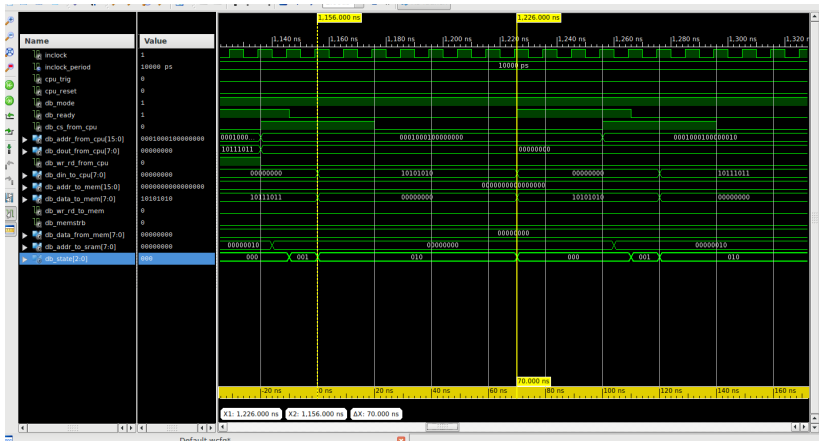


Figure 18: Timing Diagram showing Data Access Time.

Figure 18 shows the Data Access Time. This is equivalent to the time spent in State 2 of the FSM.

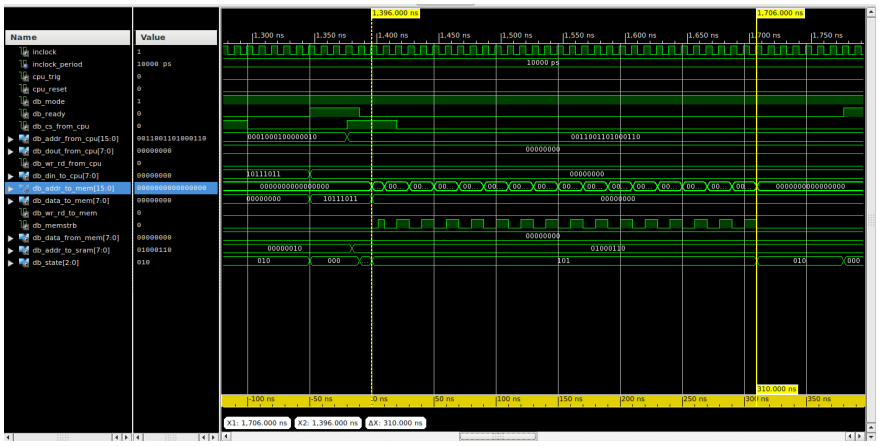


Figure 19: Timing Diagram showing Block Replacement Time.

Figure 19 shows the Block Replacement Time. This is equivalent to the time spent in State 4 or State 5 of the FSM.

To determine the miss penalties for cases 3 and 4, we use:

- Miss penalty (D-bit 0) = Hit/Miss determination time + Block replacement time + hit time = 746 ns
- Miss penalty (D-bit 1) = Hit/Miss determination time + 2*Block replacement time + hit time = 1412 ns

Table 1: Cache Performance Parameters

N	Cache Performance Parameter	Time in nS
1	Hit/Miss Determination Time	10
2	Data Access Time	70
3	Block Replacement Time	666
4	Hit Time (Case 1 and 2)	70
5	Miss penalty for Case 3 (when D-bit = 0)	746
6	Miss penalty for Case 4 (when D-bit = 1)	1412

Conclusion

In this design project, the behaviour of a cache controller was investigated by implementing a cache controller in VHDL using the Xilinx ISE CAD environment of the system. The Cache Performance Parameters were found through both measurement and calculation, as shown in Table 1, demonstrating various behaviours and characteristics of the Cache Controller. These findings prove the theoretical ideas behind Cache Controllers and how they function in relation to other components.

Throughout the design project, several challenges were encountered, with many of them stemming from simulation errors attributed to human error and the difficulty of aligning theoretical concepts with practical design implementation in VHDL. Various syntax errors and typographical errors throughout the programming process were found. These errors mostly came about from attempts to use a high-level programming language construction in the development of a project that uses a low-level programming language. To ameliorate this, VHDL documentation was consulted for solutions. Moreover, we encountered several technical errors during the process. For instance, at one point, the Xilinx ISE software generated a simulation file that exceeded 2GB in size, depleting the entire allocated disk space per user. Throughout the design project, we encountered various challenges, including simulation errors, syntax issues, and technical hiccups, all of which tested our abilities in implementing VHDL-based cache controller design, ultimately enriching our practical experience and problem-solving skills.

References

[1] *COE758 Digital Systems Engineering Project #1 - Memory Hierarchy: Cache Controller*, TMU, Toronto, ON, Canada, 2023. Accessed: Nov. 5, 2023. [Online]. https://courses.torontomu.ca/d2l/le/content/792271/viewContent/5322256/View

[2] C. Terman, "L14: The Memory Hierarchy," Computation Structures, https://computationstructures.org/lectures/caches/caches.html#39 Accessed Nov. 5, 2023. [Online].

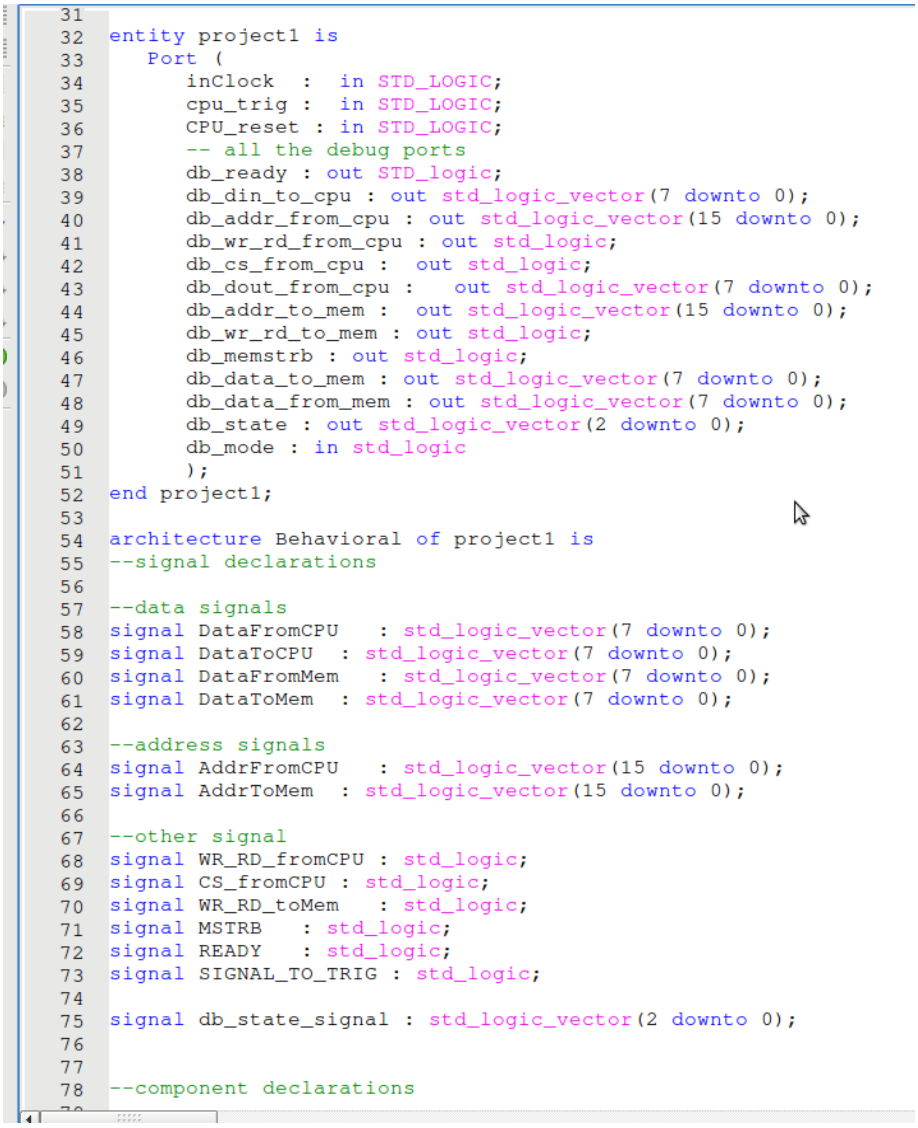
[3] *Project 1 Cache Controller: Interface Specifications*, TMU, Toronto, ON, Canada, 2023. Accessed: Nov. 5 2023. [Online].

[4] V. Guerkov, "Lec3 slides," in COE758 Digital Systems Engineering, 2023. [Online].

Appendix with VHDL Code

All VHDL files are included in the submission on D2L.

The following VHDL Code screenshots are select images used to show the general functionality of the project.



```
31
32  entity project1 is
33     Port (
34        inClock   :  in STD_LOGIC;
35        cpu_trig  :  in STD_LOGIC;
36        CPU_reset : in STD_LOGIC;
37        -- all the debug ports
38        db_ready : out STD_logic;
39        db_din_to_cpu : out std_logic_vector(7 downto 0);
40        db_addr_from_cpu : out std_logic_vector(15 downto 0);
41        db_wr_rd_from_cpu : out std_logic;
42        db_cs_from_cpu :  out std_logic;
43        db_dout_from_cpu :   out std_logic_vector(7 downto 0);
44        db_addr_to_mem :  out std_logic_vector(15 downto 0);
45        db_wr_rd_to_mem : out std_logic;
46        db_memstrb : out std_logic;
47        db_data_to_mem : out std_logic_vector(7 downto 0);
48        db_data_from_mem : out std_logic_vector(7 downto 0);
49        db_state : out std_logic_vector(2 downto 0);
50        db_mode : in std_logic
51        );
52  end project1;
53
54  architecture Behavioral of project1 is
55  --signal declarations
56
57  --data signals
58  signal DataFromCPU   : std_logic_vector(7 downto 0);
59  signal DataToCPU  : std_logic_vector(7 downto 0);
60  signal DataFromMem   : std_logic_vector(7 downto 0);
61  signal DataToMem  : std_logic_vector(7 downto 0);
62
63  --address signals
64  signal AddrFromCPU   : std_logic_vector(15 downto 0);
65  signal AddrToMem  : std_logic_vector(15 downto 0);
66
67  --other signal
68  signal WR_RD_fromCPU : std_logic;
69  signal CS_fromCPU : std_logic;
70  signal WR_RD_toMem   : std_logic;
71  signal MSTRB   : std_logic;
72  signal READY   : std_logic;
73  signal SIGNAL_TO_TRIG : std_logic;
74
75  signal db_state_signal : std_logic_vector(2 downto 0);
76
77
78  --component declarations
```

```
77
78  --component declarations
79
80
81     COMPONENT cache
82     PORT(
83        ADDR_in : IN std_logic_vector(15 downto 0);
84        WR_RD_in : IN std_logic;
85        CS : IN std_logic;
86        Din_0 : IN std_logic_vector(7 downto 0);
87        Din_1 : IN std_logic_vector(7 downto 0);
88        clk : IN std_logic;
89        ADDR_out : OUT std_logic_vector(15 downto 0);
90        WR_RD_out : OUT std_logic;
91        MSTRB : OUT std_logic;
92        Dout_0 : OUT std_logic_vector(7 downto 0);
93        Dout_1 : OUT std_logic_vector(7 downto 0);
94        RDY : OUT std_logic;
95        debug_state : out STD_logic_vector(2 downto 0)
96        );
97     END COMPONENT;
98     COMPONENT CPU_gen
99     PORT(
100       clk : IN std_logic;
101       rst : IN std_logic;
102       trig : IN std_logic;
103       Address : OUT std_logic_vector(15 downto 0);
104       wr_rd : OUT std_logic;
105       cs : OUT std_logic;
106       DOut : OUT std_logic_vector(7 downto 0)
107       );
108    END COMPONENT;
109 COMPONENT SDRAM_controller
110    PORT(
111       ADDR : IN std_logic_vector(15 downto 0);
112       WR_RD : IN std_logic;
113       MSTRB : IN std_logic;
114       Din : IN std_logic_vector(7 downto 0);
115       clk : IN std_logic;
116       Dout : OUT std_logic_vector(7 downto 0)
117       );
118    END COMPONENT;
119
```

```
121
122  begin
123
124  the_cache : cache PORT MAP(
125        ADDR_in => AddrFromCPU,
126        WR_RD_in => WR_RD_fromCPU,
127        CS => CS_fromCPU,
128        ADDR_out => AddrToMem,
129        WR_RD_out => WR_RD_toMem,
130        MSTRB => MSTRB,
131        Dout_0 => DataToMem,
132        Dout_1 => DataToCPU,
133        Din_0 => DataFromCPU,
134        Din_1 => DataFromMem,
135        RDY => READY,
136        clk => inClock,
137        debug_state => db_state_signal
138  );
139     CPU:  CPU_gen PORT MAP(
140        clk => inClock,
141        rst => CPU_reset,
142        trig => signal_to_trig,
143        Address => AddrFromCPU,
144        wr_rd => WR_RD_fromCPU,
145        cs => CS_fromCPU,
146        DOut => DataFromCPU
147     );
148
149     Inst_SDRAM_controller: SDRAM_controller PORT MAP(
150        ADDR => AddrToMem,
151        WR_RD => WR_RD_toMem,
152        MSTRB => MSTRB,
153        Din => DataToMem,
154        Dout => DataFromMem,
155        clk => inClock
156     );
157
158  process
159  begin
160     db_ready <= READY;
161     db_din_to_cpu <= DataToCPU;
162     db_addr_from_cpu <= AddrFromCPU;
163     db_wr_rd_from_cpu <= WR_RD_fromCPU;
164     db_cs_from_cpu <= CS_fromCPU;
165     db_dout_from_cpu <= DataFromCPU;
166     db_addr_to_mem <= AddrToMem;
167     db_wr_rd_to_mem <= WR_RD_toMem;
168     db_memstrb <= MSTRB;
169     db_data_to_mem <= DataToMem;
170     db_data_from_mem <= DataFromMem;
171     db_state <= db_state_signal;
172
173     if (db_mode = '0') then
174        SIGNAL_TO_TRIG <= cpu_trig;
175     else
176        SIGNAL_TO_TRIG <= READY;
177     end if;
178
179     wait for 1 ns;
180
181     wait for 1 ns;
182
183  end process;
184
185  end Behavioral;
186
```

Figure A1: Project VHDL Behavioural Code

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

entity cache is
    Port ( ADDR_in : in  STD_LOGIC_VECTOR (15 downto 0);
           WR_RD_in : in  STD_LOGIC;
           CS : in  STD_LOGIC;
           ADDR_out : out  STD_LOGIC_VECTOR (15 downto 0);
           WR_RD_out : out  STD_LOGIC;
           MSTRB : out  STD_LOGIC;
           Dout_0 : out  STD_LOGIC_VECTOR (7 downto 0);
           Dout_1 : out  STD_LOGIC_VECTOR (7 downto 0);
           Din_0 : in  STD_LOGIC_VECTOR (7 downto 0);
           Din_1 : in  STD_LOGIC_VECTOR (7 downto 0);
           RDY :  out   STD_LOGIC;
           clk :  in STD_LOGIC;
           debug_state : OUT std_logic_vector(2 downto 0);
           debug_table_match :   out STD_logic;
           debug_hit : out std_logic
          );
end cache;

architecture Behavioral of cache is

--signal declaration
--general signals
signal Dout_sel : STD_LOGIC;
signal Din_sel : STD_LOGIC;
signal WEN  : STD_LOGIC_VECTOR(0 DOWNTO 0);
signal C_addr  : STD_LOGIC_VECTOR (7 DOWNTO 0);

--cache memory signals
signal DataInSignal : STD_LOGIC_VECTOR(7 downto 0);
signal DataOutSignal : STD_LOGIC_VECTOR(7 downto 0);

--component declaration
--general declaration
COMPONENT mux2to1
   PORT(
      d0 : IN std_logic_vector(7 downto 0);
      d1 : IN std_logic_vector(7 downto 0);
      sel : IN std_logic;
      d_out : OUT std_logic_vector(7 downto 0)
      );
END COMPONENT;

COMPONENT demux2to1
   PORT(
      d_in : IN std_logic_vector(7 downto 0);
      sel : IN std_logic;
      d0 : OUT std_logic_vector(7 downto 0);
      d1 : OUT std_logic_vector(7 downto 0)
      );
END COMPONENT;

--cache memory
   COMPONENT Cache_SRAM
   PORT(
      ADDR : IN std_logic_vector(7 downto 0);
      WEN : IN std_logic_vector (0 downto 0);
      Din : IN std_logic_vector(7 downto 0);
      clk : IN std_logic;
      Dout : OUT std_logic_vector(7 downto 0)
      );
   END COMPONENT;

```

```
--cache controller
COMPONENT cacheController
   PORT(
      ADDR_in : IN std_logic_vector(15 downto 0);
      WR_RD_in : IN std_logic;
      CS : IN std_logic;
      clk : IN std_logic;
      ADDR_out : OUT std_logic_vector(15 downto 0);
      RDY : OUT std_logic;
      WR_RD_out : OUT std_logic;
      MSTRB : OUT std_logic;
      Dout_sel : OUT std_logic;
      Din_sel : OUT std_logic;
      WEN : OUT std_logic_vector(0 downto 0);
      C_addr : OUT std_logic_vector(7 downto 0);
      debug_state : out std_logic_vector(2 downto 0);
      debug_hit : out std_logic;
      debug_table_match : out std_logic
      );
END COMPONENT;

--COMPONENT bram
--  PORT (
--    clka : IN STD_LOGIC;
--    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
--    addra : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
--    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
--    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
--  );
--END COMPONENT;

begin

--cache memory

   Inst_Cache_SRAM: Cache_SRAM PORT MAP(
      ADDR => C_addr,
      WEN => WEN,
      Din => DataInSignal,
      Dout => DataOutSignal,
      clk => clk
   );

controller : cacheController PORT MAP(
      ADDR_in => ADDR_in,
      WR_RD_in => WR_RD_in,
      CS => CS,
      clk => clk,
      ADDR_out => ADDR_out,
      RDY => RDY,
      WR_RD_out => WR_RD_out,
      MSTRB => MSTRB,
      Dout_sel => Dout_sel,
      Din_sel => Din_sel,
      WEN => WEN,
      C_addr => C_addr,
      debug_state => debug_state,
      debug_hit => debug_hit,
      debug_table_match => debug_table_match
   );

multiplexer: mux2to1 PORT MAP(
      d0 => Din_0,
      d1 => Din_0,
      sel => Din_sel,
      d_out => DataInSignal
   );

demux: demux2to1 PORT MAP(
      d0 => Dout_0,
      d1 => Dout_1,
      d_in => DataOutSignal,
      sel => Dout_sel
   );

end Behavioral;
```

Figure A2: VHDL Implementation of Cache

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.NUMERIC_STD.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity Cache_SRAM is
34      Port ( ADDR : in  STD_LOGIC_VECTOR (7 downto 0);
35             WEN : in  STD_LOGIC_VECTOR(0 downto 0);
36             Din : in  STD_LOGIC_VECTOR (7 downto 0);
37             Dout : out  STD_LOGIC_VECTOR (7 downto 0);
38             clk : in  STD_LOGIC);
39  end Cache_SRAM;
40
41  architecture Behavioral of Cache_SRAM is
42
43  type mem_block is array (31 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
44  type memory is array (7 downto 0) of mem_block;
45  signal sram_mem: memory := ((others=> (others=> (others=>'0'))));
46
47  signal index : STD_LOGIC_VECTOR(2 downto 0);
48  signal offset : STD_LOGIC_VECTOR(4 downto 0);
49
50  begin
51
52  process(clk, WEN)
53  begin
54     index <= ADDR(7 downto 5);
55     offset <= ADDR(4 downto 0);
56
57     if (clk'event and clk = '1') then
58        if (WEN(0) = '1') then
59           sram_mem(to_integer(unsigned(index)))(to_integer(unsigned(offset))) <= Din;
60        elsif (WEN(0) = '0') then
61           Dout <= sram_mem(to_integer(unsigned(index)))(to_integer(unsigned(offset)));
62        end if;
63     end if;
64  end process;
65
66  end Behavioral;
67
```

Figure A3: VHDL Implementation of Cache SRAM

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity cacheController is
    Port ( ADDR_in : in  STD_LOGIC_VECTOR (15 downto 0);
           WR_RD_in : in  STD_LOGIC;
           clk :  in STD_LOGIC;
           CS : in  STD_LOGIC;
           ADDR_out : out  STD_LOGIC_VECTOR (15 downto 0);
           RDY : out  STD_LOGIC;
           WR_RD_out : out  STD_LOGIC;
           MSTRB : out  STD_LOGIC;
           Dout_sel : out  STD_LOGIC;
           Din_sel : out  STD_LOGIC;
           WEN : out  STD_LOGIC_vector (0 downto 0);
           C_addr : out  STD_LOGIC_VECTOR (7 downto 0);
           debug_state : out std_logic_vector(2 downto 0);
           debug_hit : out std_logic;
           debug_table_match : out std_logic
           );
end cacheController;

architecture Behavioral of cacheController is
signal tag  :  STD_LOGIC_VECTOR(7 downto 0) := ADDR_in(15 downto 8);
signal index   :  STD_LOGIC_VECTOR(2 downto 0)  := ADDR_in(7 downto 5);
signal offset   :  STD_LOGIC_VECTOR(4 downto 0)  := ADDR_in(4 downto 0);

signal table_WEN  : std_logic;
signal table_REN  : std_logic;
signal table_dataIn  : std_logic_vector(9 downto 0);
signal table_dataOut : std_logic_vector(9 downto 0);
signal table_match   : std_logic;
signal hit  :  std_logic;

signal debug_state_signal : std_logic_vector(2 downto 0);
```

```
COMPONENT controller_table
   PORT(
      clk : IN std_logic;
      write_enable : IN std_logic;
      address : IN std_logic_vector(2 downto 0);
      data_in : IN std_logic_vector(9 downto 0);
      data_out : OUT std_logic_vector(9 downto 0)
      );
END COMPONENT;

   COMPONENT ctrl_fsm
   PORT(
      tag : IN std_logic_vector(7 downto 0);
      old_tag : IN std_logic_vector(7 downto 0);
      index : IN std_logic_vector(2 downto 0);
      offset : IN std_logic_vector(4 downto 0);
      clk : IN std_logic;
      dirty : IN std_logic;
      hit : IN std_logic;
      cs : IN std_logic;
      WR_RDi : IN std_logic;
      MSTRB : OUT std_logic;
      RDY : OUT std_logic;
      Addr_out : OUT std_logic_vector(15 downto 0);
      WEN : OUT std_logic_vector(0 downto 0);
      Dout_sel : OUT std_logic;
      Din_sel : OUT std_logic;
      WR_RDo : OUT std_logic;
      table_WEN : OUT std_logic;
      data_to_table : OUT std_logic_vector(9 downto 0);
      debug_state : OUT std_logic_vector(2 downto 0)
      );
   END COMPONENT;

begin

   Inst_controller_table: controller_table PORT MAP(
      clk => clk,
      write_enable => table_WEN,
      address => index,
      data_in => table_dataIn,
      data_out => table_dataOut
   );
```

```
109
110     Inst_ctrl_fsm: ctrl_fsm PORT MAP(
111        tag => tag,
112        old_tag => table_dataOut(9 downto 2),
113        index => index,
114        offset => offset,
115        clk => clk,
116        MSTRB => MSTRB,
117        dirty => table_dataOut(1),
118        hit => hit,
119        cs => cs,
120        WR_RDi => WR_RD_in,
121        RDY => rdy,
122        Addr_out => Addr_out,
123        WEN => WEN,
124        Dout_sel => Dout_sel,
125        Din_sel => Din_sel,
126        WR_RDo => WR_RD_out,
127        table_WEN => table_WEN,
128        data_to_table => table_dataIn,
129        debug_state => debug_state
130     );
131
132     process(clk)
133     begin
134        tag <= ADDR_in(15 downto 8);
135        index <= ADDR_in(7 downto 5);
136        offset <= ADDR_in(4 downto 0);
137
138        C_addr <= ADDR_in(7 downto 0);
139        table_REN <= '1';
140        if (table_dataOut(9 downto 2) = tag) then
141           table_match <= '1';
142        else
143           table_match <= '0';
144        end if;
145
146        hit <= table_match AND table_dataOut(0);
147
148        debug_table_match <= table_match;
149        debug_hit <= table_match AND table_dataOut(0);
150
151     end process;
152
153  end Behavioral;
154
```

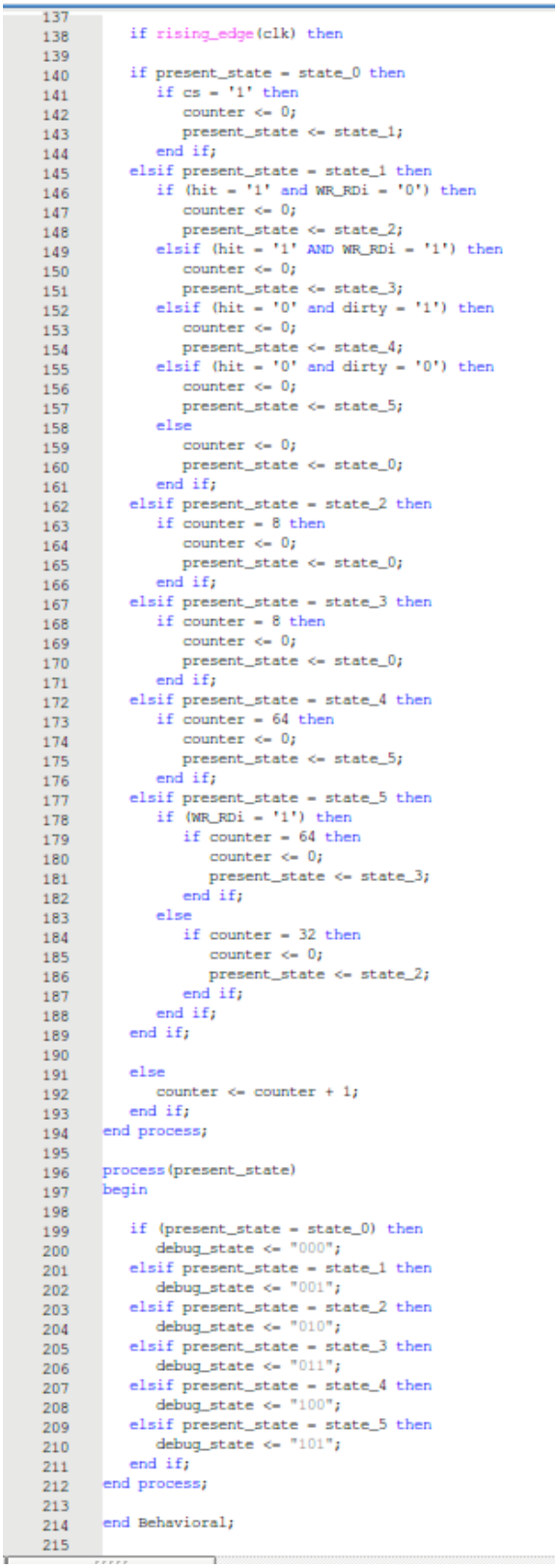
Figure A4: VHDL Implementation of Cache Controller Behavioural.

```
19
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_UNSIGNED.ALL;
23  use IEEE.NUMERIC_STD.ALL;
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity controller_table is
34      Port ( clk : in  STD_LOGIC;
35             write_enable : in  STD_LOGIC;
36             address : in  STD_LOGIC_VECTOR (2 downto 0);
37             data_in : in  STD_LOGIC_VECTOR (9 downto 0);
38             data_out : out  STD_LOGIC_VECTOR (9 downto 0));
39  end controller_table;
40
41  architecture Behavioral of controller_table is
42     type RegArray is array(0 to 7) of std_logic_vector(9 downto 0);
43     signal registers  :  RegArray := (others => (others => ('0')));
44
45  begin
46     process(clk)
47
48     begin
49        if rising_edge(clk) then
50           if write_enable = '1' then
51              registers(to_integer(unsigned(address))) <= data_in;
52           end if;
53        data_out <= registers(to_integer(unsigned(address)));
54        end if;
55     end process;
56
57  end Behavioral;
58
```

Figure A5: VHDL Implementation of FSM Control Unit

```
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ctrl_fsm is
    Port ( tag : in  STD_LOGIC_VECTOR (7 downto 0);
           old_tag : in  STD_LOGIC_VECTOR (7 downto 0);
           index : in  STD_LOGIC_VECTOR (2 downto 0);
           offset : in  STD_LOGIC_VECTOR (4 downto 0);
           clk : in std_logic;
           MSTRB : out  STD_LOGIC;
           dirty : in  STD_LOGIC;
           hit : in  STD_LOGIC;
           cs : in  STD_LOGIC;
           WR_RDi : in  STD_LOGIC;
           RDY : out  STD_LOGIC;
           Addr_out : out  STD_LOGIC_VECTOR (15 downto 0);
           WEN : out  STD_LOGIC_vector(0 downto 0);
           Dout_sel : out  STD_LOGIC;
           Din_sel : out  STD_LOGIC;
           WR_RDo : out  STD_LOGIC;
           --table control ports
           table_WEN :  out STD_LOGIC;
           data_to_table   :  out STD_logic_vector (9 downto 0);
           debug_state : out std_logic_vector(2 downto 0)
           );
end ctrl_fsm;

architecture Behavioral of ctrl_fsm is

TYPE STATETYPE IS (state_0, state_1, state_2, state_3, state_4, state_5);
SIGNAL present_state: STATETYPE := state_0;

SIGNAL counter : integer := 0;

begin
-- outputs depending on state
process(clk, present_state)
begin
   if present_state = state_0 then
      RDY <= '1';
```

```
58
59  TYPE STATETYPE IS (state_0, state_1, state_2, state_3, state_4, state_5);
60  SIGNAL present_state: STATETYPE := state_0;
61
62  SIGNAL counter : integer := 0;
63
64  begin
65  -- outputs depending on state
66  process(clk, present_state)
67  begin
68     if present_state = state_0 then
69        RDY <= '1';
70        Addr_out <= "00000000000000000";
71        WEN <= "0";
72        Dout_sel <= '0';
73        Din_sel <= '0';
74        WR_RDo <= '0';
75        MSTRB <= '0';
76        table_WEN <= '0';
77        data_to_table <= "0000000000";
78     elsif present_state = state_1 then
79        RDY <= '0';
80        Addr_out <= "00000000000000000";
81        WEN <= "0";
82        Dout_sel <= '0';
83        Din_sel <= '0';
84        WR_RDo <= '0';
85        MSTRB <= '0';
86        table_WEN <= '0';
87        data_to_table <= "0000000000";
88     elsif present_state = state_2 then
89        RDY <= '0';
90        Addr_out <= "00000000000000000";
91        WEN <= "0";
92        Dout_sel <= '1';
93        Din_sel <= '0';
94        WR_RDo <= '0';
95        MSTRB <= '0';
96        table_WEN <= '0';
97        data_to_table <= "0000000000";
98     elsif present_state = state_3 then
99        RDY <= '0';
100       Addr_out <= "00000000000000000";
101       WEN <= "1";
102       Dout_sel <= '0';
103       Din_sel <= '0';
104       WR_RDo <= '0';
105       MSTRB <= '0';
106       table_WEN <= '1';
107       data_to_table <= tag & "11";
108    elsif present_state = state_4 then
109       RDY <= '0';
110       Addr_out <= old_tag & index & std_logic_vector(to_unsigned(counter/2, 5));
111       WEN <= "0";
112       Dout_sel <= '0';
113       Din_sel <= '0';
114       WR_RDo <= '1';
115       if counter mod 2 = 0 then
116         MSTRB <= '0';
117       else
118         MSTRB <= '1';
119       end if;
120       table_WEN <= '0';
121       data_to_table <= "0000000000";
122    elsif present_state = state_5 then
123       RDY <= '0';
124       Addr_out <= tag & index & std_logic_vector(to_unsigned(counter/2, 5));
125       WEN <= "1";
126       Dout_sel <= '0';
127       Din_sel <= '1';
128       WR_RDo <= '0';
129       if counter mod 2 = 0 then
130         MSTRB <= '0';
131       else
132         MSTRB <= '1';
133       end if;
134       table_WEN <= '1';
135       data_to_table <= tag & "01";
136    end if;
```



```
137
138      if rising_edge(clk) then
139
140      if present_state = state_0 then
141         if cs = '1' then
142            counter <= 0;
143            present_state <= state_1;
144         end if;
145      elsif present_state = state_1 then
146         if (hit = '1' and WR_RDi = '0') then
147            counter <= 0;
148            present_state <= state_2;
149         elsif (hit = '1' AND WR_RDi = '1') then
150            counter <= 0;
151            present_state <= state_3;
152         elsif (hit = '0' and dirty = '1') then
153            counter <= 0;
154            present_state <= state_4;
155         elsif (hit = '0' and dirty = '0') then
156            counter <= 0;
157            present_state <= state_5;
158         else
159            counter <= 0;
160            present_state <= state_0;
161         end if;
162      elsif present_state = state_2 then
163         if counter = 8 then
164            counter <= 0;
165            present_state <= state_0;
166         end if;
167      elsif present_state = state_3 then
168         if counter = 8 then
169            counter <= 0;
170            present_state <= state_0;
171         end if;
172      elsif present_state = state_4 then
173         if counter = 64 then
174            counter <= 0;
175            present_state <= state_5;
176         end if;
177      elsif present_state = state_5 then
178         if (WR_RDi = '1') then
179            if counter = 64 then
180               counter <= 0;
181               present_state <= state_3;
182            end if;
183         else
184            if counter = 32 then
185               counter <= 0;
186               present_state <= state_2;
187            end if;
188         end if;
189      end if;
190
191      else
192         counter <= counter + 1;
193      end if;
194   end process;
195
196   process(present_state)
197   begin
198
199      if (present_state = state_0) then
200         debug_state <= "000";
201      elsif present_state = state_1 then
202         debug_state <= "001";
203      elsif present_state = state_2 then
204         debug_state <= "010";
205      elsif present_state = state_3 then
206         debug_state <= "011";
207      elsif present_state = state_4 then
208         debug_state <= "100";
209      elsif present_state = state_5 then
210         debug_state <= "101";
211      end if;
212   end process;
213
214   end Behavioral;
215
```

Figure A6: VHDL Implementation of FSM Control Unit

```
16  ----------------------------------------------------------------------------------
17  library IEEE;
18  use IEEE.STD_LOGIC_1164.ALL;
19  use IEEE.STD_LOGIC_ARITH.ALL;
20  use IEEE.STD_LOGIC_UNSIGNED.ALL;
21
22  ---- Uncomment the following library declaration if instantiating
23  ---- any Xilinx primitives in this code.
24  --library UNISIM;
25  --use UNISIM.VComponents.all;
26
27  entity CPU_gen is
28     Port (
29        clk      : in  STD_LOGIC;
30        rst      : in  STD_LOGIC;
31        trig     : in  STD_LOGIC;
32        -- Interface to the Cache Controller.
33        Address  : out  STD_LOGIC_VECTOR (15 downto 0);
34        wr_rd    : out  STD_LOGIC;
35        cs       : out  STD_LOGIC;
36        DOut     : out  STD_LOGIC_VECTOR (7 downto 0)
37     );
38  end CPU_gen;
39
40  architecture Behavioral of CPU_gen is
41
42     -- Pattern storage and control.
43     signal patOut : std_logic_vector(24 downto 0);
44     signal patCtrl : std_logic_vector(2 downto 0) := "111";
45     signal updPat : std_logic;
46
47     -- Main control.
48     signal st1 : std_logic_vector(2 downto 0) := "000";
49     signal st1N : std_logic_vector(2 downto 0);
50
51     -- Rising edge detection.
52     signal rReg1, rReg2 : std_logic;
53     signal trig_r : std_logic;
54
55  begin
56
```

```
60
61    -- State storage.
62    process(clk, rst, st1N)
63    begin
64       if(rst = '1')then
65          st1 <= "000";
66       else
67          if(clk'event and clk = '1')then
68             st1 <= st1N;
69          end if;
70       end if;
71    end process;
72
73    -- Next state generation.
74    process(st1, trig_r)
75    begin
76       if(st1 = "000")then
77          if(trig_r = '1')then
78             st1N <= "001";
79          else
80             st1N <= "000";
81          end if;
82       elsif(st1 = "001")then
83          st1N <= "010";
84       elsif(st1 = "010")then
85          st1N <= "011";
86       elsif(st1 = "011")then
87          st1N <= "100";
88       elsif(st1 = "100")then
89          st1N <= "101";
90       elsif(st1 = "101")then
91          st1N <= "000";
92       else
93          st1N <= "000";
94       end if;
95    end process;
96
97    -- Output generation.
98    process(st1)
99    begin
100      if(st1 = "000")then
101         updPat <= '0';
102         cs <= '0';
103      elsif(st1 = "001")then
104         updPat <= '1';
105         cs <= '0';
106      elsif(st1 = "010")then
107         updPat <= '0';
108         cs <= '1';
109      elsif(st1 = "011")then
110         updPat <= '0';
111         cs <= '1';
112      elsif(st1 = "100")then
113         updPat <= '0';
114         cs <= '1';
115      elsif(st1 = "101")then
116         updPat <= '0';
117         cs <= '1';
118      else
119      end if;
120   end process;
121
```

```
122    ----------------------------------------------------
123    -- Pattern generator and control circuit.
124    ----------------------------------------------------
125
126    -- Generator control circuit.
127    process(clk, rst, updPat, patCtrl)
128    begin
129       if(rst = '1')then
130          patCtrl <= "111";
131       else
132          if(clk'event and clk = '1')then
133             if(updPat = '1')then
134                patCtrl <= patCtrl + "001";
135             else
136                patCtrl <= patCtrl;
137             end if;
138          end if;
139       end if;
140    end process;
141
142    -- Pattern storage.
143    process(patCtrl)
144    begin
145       if(patCtrl = "000")then
146          patOut <= "000100010000000010101010101";
147       elsif(PatCtrl = "001")then
148          patOut <= "000100010000000101011101111";
149       elsif(PatCtrl = "010")then
150          patOut <= "000100010000000000000000000";
151       elsif(PatCtrl = "011")then
152          patOut <= "000100010000001000000000000";
153       elsif(PatCtrl = "100")then
154          patOut <= "001100110100011000000000000";
155       elsif(PatCtrl = "101")then
156          patOut <= "010001000100010000000000000";
157       elsif(PatCtrl = "110")then
158          patOut <= "010101010000010011001100l";
159       else
160          patOut <= "011001100000011000000000000";
161       end if;
162    end process;
```

```
164    ----------------------------------------------------
165    -- Rising edge detector.
166    ----------------------------------------------------
167
168    -- Register 1
169    process(clk, trig)
170    begin
171       if(clk'event and clk = '1')then
172          rReg1 <= trig;
173       end if;
174    end process;
175
176    -- Register 2
177    process(clk, rReg1)
178    begin
179       if(clk'event and clk = '1')then
180          rReg2 <= rReg1;
181       end if;
182    end process;
183
184    trig_r <= rReg1 and (not rReg2);
185
186    ----------------------------------------------------
187    -- Output connections.
188    ----------------------------------------------------
189
190    -- Output mapping:
191    -- Address [24 .. 9]
192    -- Data [8 .. 1]
193    -- Wr/Rd [0]
194
195    Address(15 downto 0) <= patOut(24 downto 9);
196    DOut(7 downto 0) <= patOut(8 downto 1);
197    wr_rd <= patOut(0);
198
199
200 end Behavioral;
```

Figure A7: VHDL Implementation of CPU Generator

```
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SDRAM_controller is
    Port ( ADDR : in  STD_LOGIC_VECTOR (15 downto 0);
           WR_RD : in  STD_LOGIC;
           MSTRB : in  STD_LOGIC;
           Din : in  STD_LOGIC_VECTOR (7 downto 0);
           Dout : out  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC);
end SDRAM_controller;

architecture Behavioral of SDRAM_controller is

type memory is array (65535 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
signal sdram_mem: memory := ((others=> (others=>'0')));

begin
   process(clk, MSTRB)
   begin
      if (clk'event AND clk = '1' AND MSTRB = '1') then
         if (WR_RD = '0') then
            Dout <= sdram_mem(to_integer(unsigned(ADDR)));
         elsif (WR_RD = '1') then
            sdram_mem(to_integer(unsigned(ADDR))) <= Din;
         end if;
      end if;
   end process;
end Behavioral;
```

Figure A8: VHDL Implementation of SDRAM Controller