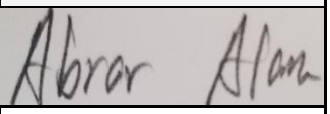
Ryerson University

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

Course Title:	Computer Organization and Architecture
Course Number:	COE 608
Semester/Year (e.g.F2016)	Winter 2021
Instructor:	Professor Nagi Mekhiel

Assignment/Lab Number:	4a
Assignment/Lab Title:	Data Memory Module

Submission Date:	February 27, 2021
Due Date:	February 27, 2021

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
ALAM	ABRAR	500725366	03	

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

Table of Contents

Objective

In this lab, a simple data-memory module has been implemented, which can write a 32-bit data to a specified 8-bit address and also read this written data from that address. This size of this data memory is 1K bytes (since we can store 256 4-byte or 32-bit numbers, and 256×4 = 1024 bytes = 1K byte). The block diagram of the designed data-memory module is presented in **figure 1**.

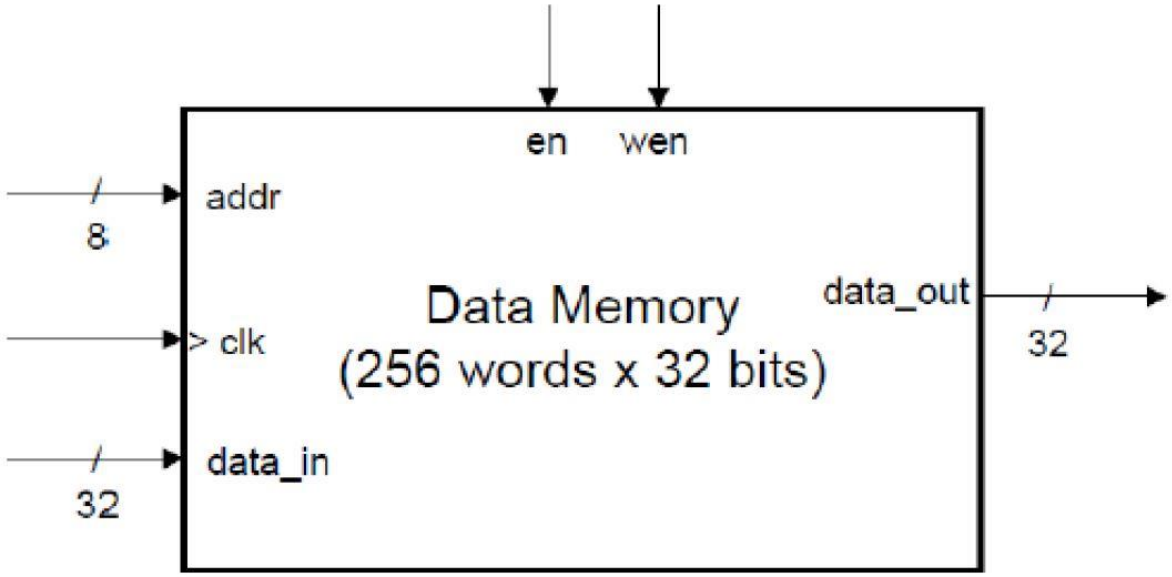


Figure 1: Block diagram of the data-memory module implemented.

Aside from the 32-bit input/output, and 8-bit address input, this memory module relies on an external clock **clk**, control signals **en**, and **wen**.

Design and implementation

The data-memory module has been implemented using a 2-D array of registers (256 words of 32-bit data), and VHDL **array** data type was used. Please note that this data-memory module has been designed to perform read/write operations upon the falling edge of the clock signal input. Besides, when the enabler, **en** is 0, the data-memory module is turned off. However, **en** = 1, and **wen** = 0 means a read request, and the memory module will output the content of the 8-bit address input, **addr** upon the falling edge of the **clk** signal, after some possible propagation delay. Due to single cycle access latency, however, if a read is requested exactly at the instant of falling edge of the clock, the memory will be able output the read value at the falling edge of following clock cycle. Besides a successful write operation (**wen** = 1, and **en** = 1) will require the stability of the **wen**, **en**, **addr**, and **data_in** signals (see **figure 1**).

Table 1 summarizes the control signals' (**wen**, and **en**) functionalities present within the memory-module

en	wen	Function	data_out
0	X	N/A	0
1	0	Read	M[addr]
1	1	Write M[addr] <= data_in	0

Table 1: Data-memory module unit functions

Results and Conclusions

The implementation VHDL code has been presented in the **Appendix** section. The correctness of the implemented memory-module has been verified by the functional simulation diagram presented in **figure 2**.

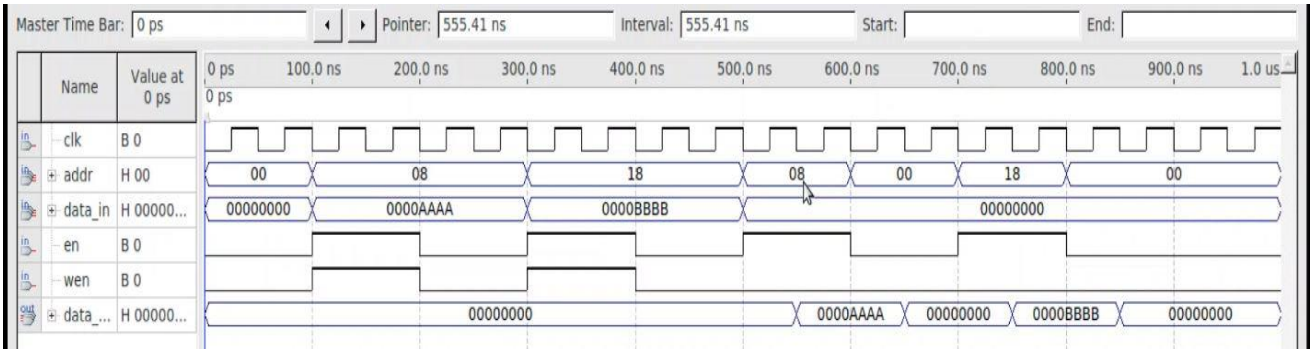


Figure 2: Timing simulation waveform of the implemented data-memory module

Although **figure 2** depicts correctness of our implementation, inevitable gate-level delays associated with our implementation was examined and has been presented in **figure 3**. In **figure 3,** we observe that our data-memory module took approx. **11.154 ns** to initialize at the beginning, then, during the reading operations, showed approx. **15 ns** delay (with respect to the falling edge of the clock, **clk**). Although such delays were present, they did not affect the correctness of the implementation.

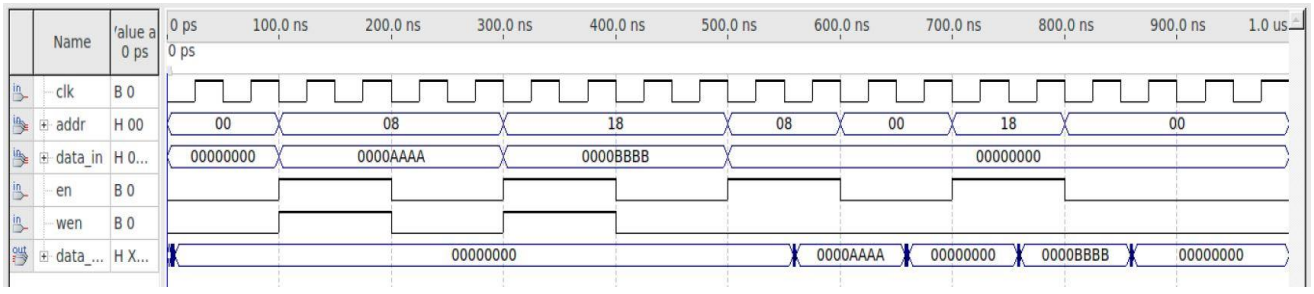


Figure 3: Timing waveform showing associated delays present in the data-memory module

Now, we pay attention to the worst-case delays associated with read and write operations. To achieve this, **Quartus** software's **TimeQuest Analyzer** option was used. In **figure 4**, the worst-case delays associated with reading (data_out port) has been presented. According to **figure 4**, the worst-case delay to set any single bit of the **data_out** port upon the falling edge of the clock signal, was **10.625 ns**. While the worst case-delay to clear a single bit of the **data_out** port upon the falling edge of the clock, was **10.538 ns**. To get the average delay associated with the **data_out**, we take the average of these two values, and get **10.5815 ns**.

Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	data_out[*]	clk	10.625	10.538	Fall	clk
1	data_out[0]	clk	8.968	8.979	Fall	clk
2	data_out[1]	clk	9.078	9.110	Fall	clk
3	data_out[2]	clk	9.717	9.662	Fall	clk
4	data_out[3]	clk	8.019	8.028	Fall	clk
5	data_out[4]	clk	8.924	8.962	Fall	clk
6	data_out[5]	clk	8.511	8.505	Fall	clk
7	data_out[6]	clk	8.516	8.525	Fall	clk
8	data_out[7]	clk	8.098	8.118	Fall	clk
9	data_out[8]	clk	9.084	9.047	Fall	clk
10	data_out[9]	clk	8.612	8.672	Fall	clk
11	data_out[10]	clk	9.377	9.319	Fall	clk
12	data_out[11]	clk	10.295	10.217	Fall	clk
13	data_out[12]	clk	8.884	8.878	Fall	clk
14	data_out[13]	clk	8.288	8.261	Fall	clk
15	data_out[14]	clk	8.599	8.612	Fall	clk
16	data_out[15]	clk	8.251	8.281	Fall	clk
17	data_out[16]	clk	8.081	8.092	Fall	clk
18	data_out[17]	clk	8.486	8.453	Fall	clk
19	data_out[18]	clk	8.831	8.814	Fall	clk
20	data_out[19]	clk	8.555	8.581	Fall	clk
21	data_out[20]	clk	8.233	8.273	Fall	clk

Figure 4: The top row showing the worst-case delays associated with data_out port (reading)

The delays associated with reading stored data (refer to **figure 3**) is therefore caused by the clock to output times presented in **figure 4**.

As for, the worst-case delays associated with writing data to the memory module, we examine the setup times generated by **TimeQuest Analyzer** option in **Quartus** software. The setup-times associated with various input ports are presented in **figure 5**.

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	addr[*]	clk	13.038	13.377	Fall	clk
2	data_in[*]	clk	8.436	8.668	Fall	clk
3	en	clk	9.097	9.535	Fall	clk
4	wen	clk	10.066	10.439	Fall	clk

Figure 5: Setup-times that can contribute to the delays associated with writing data to the memory module.

The initialization delay of about **11.154 ns** (refer to **figure 3**) is then likely to be caused by the setup-times associated with various input ports (see **figure 5**).

Reference

1. *COE 608 Lab 4a – Data Memory Module*. D2L.
2. *COE 608 Lab 4a Tutorial – Data Memory Module*. D2L.
3. “VHDL Convert to Integer?.”, https://hardwarecoder.com/qa/111/vhdl-convert-to-integer
4. “Arrays-VHDL Example”, https://www.nandland.com/vhdl/examples/example-array-type-vhdl.html
5. “Setup and hold time definition”, https://www.idconline.com/technical_references/pdfs/electronic_engineering/Setup_and_hold_time_definition.pdf

Appendix

This section presents VHDL codes of all the components designed in this lab.

Data-memory module:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity data_mem is
port(
    clk : in std_logic;
    addr    :   in unsigned(7 downto 0);
    data_in :   in std_logic_vector(31 downto 0);
    wen : in std_logic;
    en  : in std_logic;
    data_out    : out std_logic_vector(31 downto 0)
);
end data_mem;

architecture Behavior of data_mem is
    type RAM is array(0 to 255) of std_logic_vector(31 downto 0); --
we created a type called RAM,
                                                                   --
which is an array
    signal DATAMEM : RAM; --signal of type RAM
begin
    process(clk, en, wen)
    begin
        if(clk'event and clk='0') then --we are interested in falling
edge only
            if(en = '0') then
```

```
                data_out <= (others => '0');
            else
                if (wen = '0') then
                    data_out <= DATAMEM(to_integer(addr));
                end if;
                if (wen = '1') then
                    DATAMEM(to_integer(addr)) <= data_in;
                    data_out <= (others => '0');
                end if;
            end if;
        end if;
end process;
end Behavior;
```