National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering and Computer Science

Faculty Member: Dr. Rehan Ahmed Dated: 26/04/2023

Semester: 6th Section: BEE 12C

EE-421: Digital System Design

Lab 9: Memory Blocks

Group Members

		PLO4-CLO3		PLO5 - CLO4	PLO8 - CLO5	PLO9 - CLO6
Name	**Reg. No**	**Viva / Quiz / Lab Performance**	**Analysis of data in Lab Report**	**Modern Tool Usage**	**Ethics and Safety**	**Individual and Teamwork**
		5 Marks	**5 Marks**	**5 Marks**	**5 Marks**	**5 Marks**
Danial Ahmad	331388					
Muhammad Umer	345834					
Tariq Umar	334943					

1 Table of Contents

2 Finite State Machines

2.1 Objectives

In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology, it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. In this exercise we will examine the general issues involved in implementing such memory.

2.2 Introduction

In computer systems, it is necessary to provide a substantial amount of memory. This memory can be used to store data, instructions, and other information that is needed by the computer. If a system is implemented using FPGA technology, it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device.

In this exercise, we will examine the general issues involved in implementing such memory. We will start by discussing the different types of memory that can be implemented in an FPGA. We will then discuss the factors that need to be considered when choosing a memory type for a particular application. Finally, we will discuss the steps involved in implementing memory in an FPGA.

2.3 Software

Quartus Prime is a comprehensive design software developed by Intel Corporation for designing digital circuits using Field-Programmable Gate Arrays (FPGAs). It is a leading software platform in the field of digital design, offering a range of advanced tools and features that enable users to easily create, debug, and verify complex digital circuits. With Quartus Prime, users can benefit from a streamlined design flow that facilitates the creation of digital circuits from concept to implementation. It provides an intuitive graphical user interface that allows users to easily design, test, and debug their circuits. Additionally, Quartus Prime supports a variety of popular programming languages, making it a versatile platform for digital designers of all levels.

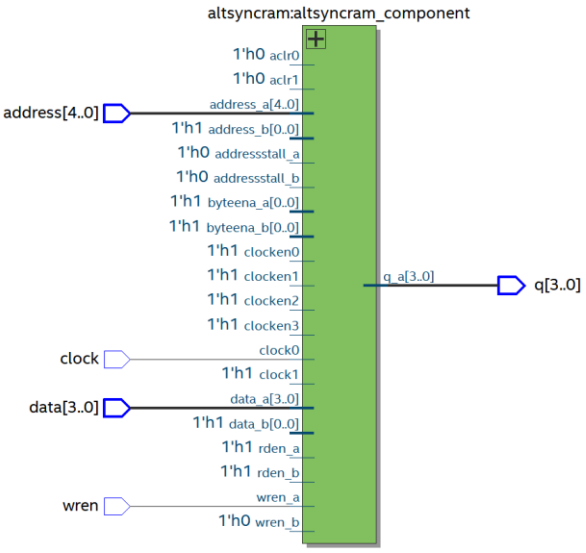
3 Lab Procedure

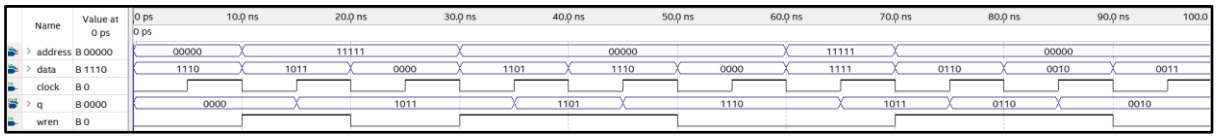
3.1 Part I

1. Create a new Quartus project to implement the memory module.
2. To open the IP Catalog in the Quartus software click on Tools > IP Catalog. In the IP Catalog window choose the RAM: 1-PORT module, which is found under the Basic Functions > On Chip Memory category. Select Verilog HDL as the type of output file to create, give the file the name ram32x4.v, and click OK. As shown in Figure 2 specify a memory size of 32 four-bit words. Select M9K if your DE-series board has a MAX 10 or Cyclone IV FPGA, otherwise select M10K.

```
module ram32x4 (input [4:0] address, input clock, input [3:0] data, input wren, output
                                                    [3:0] q);
```

3. Instantiate this subcircuit in a top-level Verilog file that includes appropriate input and output signals for the memory ports given in Figure 1b. Compile the circuit. Observe in the Compilation Report that the Quartus Compiler uses 128 bits in one of the FPGA memory blocks to implement the RAM circuit.
4. Simulate the behavior of your circuit and ensure that you can read and write data in the memory.



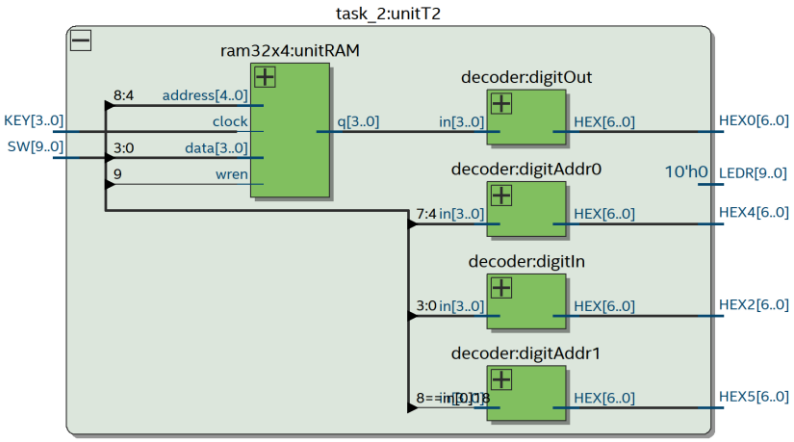


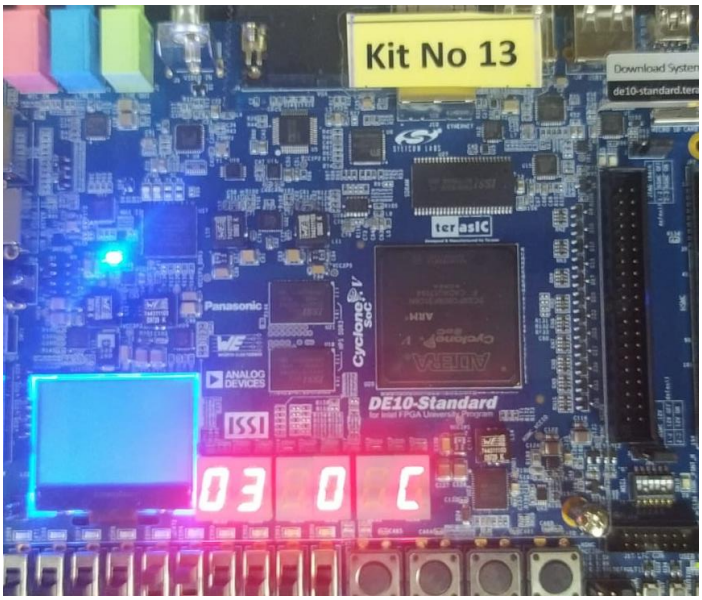
3.2 Part II

Now, we want to realize the memory circuit in the FPGA on your DE-series board, and use slide switches to load some data into the created memory. We also want to display the contents of the RAM on the 7-segment displays.

1. Make a new Quartus project which will be used to implement the desired circuit on your DE-series board.
2. Create another Verilog file that instantiates the ram32x4 module and that includes the required input and output pins on your DE-series board. Use slide switches SW3−0 to provide input data for the RAM, and use switches switches SW8−4 to specify the address. Use SW9 as the Write signal and use KEY0 as the Clock input. Show the address value on the 7-segment displays HEX5 − 4, show the data being input to the memory on HEX2, and show the data read out of the memory on HEX0.
3. Test your circuit and make sure that data can be stored into the memory at various locations.

```
module task_2 (
    input [3:0] KEY,
    input [9:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX2,
    output [6:0] HEX4,
    output [6:0] HEX5
);

    wire [3:0] out;

    ram32x4 unitRAM (
        .address(SW[8:4]),
        .clock(KEY[0]),
        .data(SW[3:0]),
        .wren(SW[9]),
        .q(out)
    );

    decoder digitOut(.in(out), .HEX(HEX0));
    decoder digitIn(.in(SW[3:0]), .HEX(HEX2));
    decoder digitAddr0(.in(SW[7:4]), .HEX(HEX4));
    decoder digitAddr1(.in(SW[8]), .HEX(HEX5));

endmodule
```





3.3 Part III

Instead of creating a memory module subcircuit by using the IP Catalog, we can implement the required memory by specifying its structure in Verilog code. In a Verilog-specified design it is possible to define the memory as a multidimensional array. A 32 x 4 array, which has 32 words with 4 bits per word, can be declared by the statement:

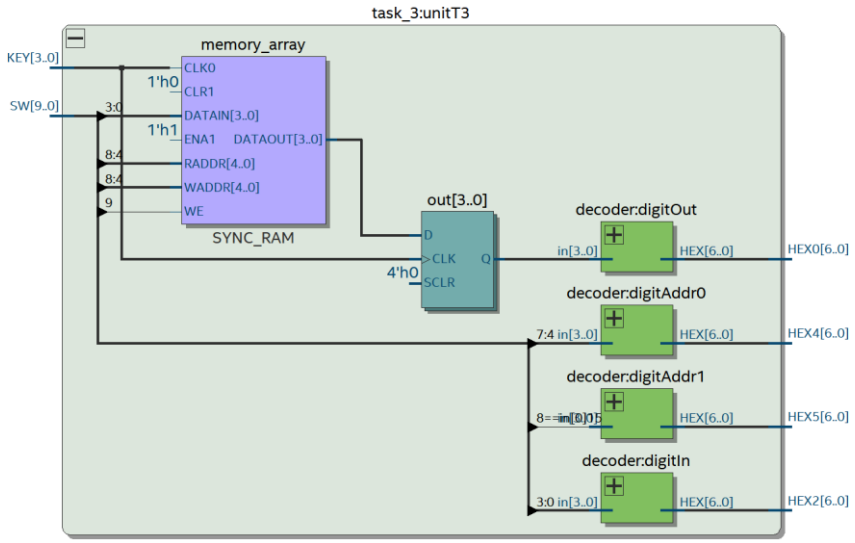
```
reg [3:0] memory_array [31:0];
```

Perform the following steps:

1. Create a new project which will be used to implement the desired circuit on your DE-series board.
2. Write a Verilog file that provides the necessary functionality, including the ability to load the RAM and read its contents as was done in Part II.
3. Assign the pins on the FPGA to connect to the switches and the 7-segment displays.
4. Compile the circuit and download it into the FPGA chip.
5. Test the functionality of your design by applying some inputs and observing the output.

```
module task_3 (
    input [3:0] KEY,
    input [9:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX2,
    output [6:0] HEX4,
    output [6:0] HEX5
);

    wire wren = SW[9];
    wire clk = KEY[0];
    wire [3:0] data = SW[3:0];
    wire [4:0] addr = SW[8:4];
```

```
    reg [3:0] out;
    reg [3:0] memory_array[31:0];

    always @(posedge clk) begin
        if (wren)
            memory_array[addr] = data;
        out = memory_array[addr];
    end

    decoder digitOut(.in(out), .HEX(HEX0));
    decoder digitIn(.in(SW[3:0]), .HEX(HEX2));
    decoder digitAddr0(.in(SW[7:4]), .HEX(HEX4));
    decoder digitAddr1(.in(SW[8]), .HEX(HEX5));

endmodule
```



3.4 Part IV

Create a new Quartus project for your circuit. To generate the desired memory module open the IP Catalog and select the RAM: 2-PORT module in the Basic Functions > On Chip Memory category. As shown in Figure 5, choose With one read port and one write port in the category called How will you be using the dual port ram?

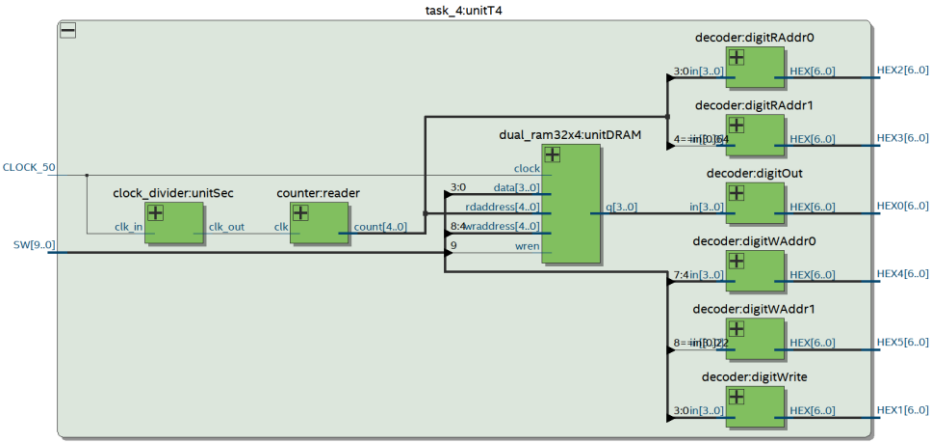
```
DEPTH = 32;
WIDTH = 4;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

0 : 0000;
1 : 0001;
2 : 0010;
3 : 0011;
. . . (some lines not shown)
1E : 1110;
1F : 1111;

END;
```

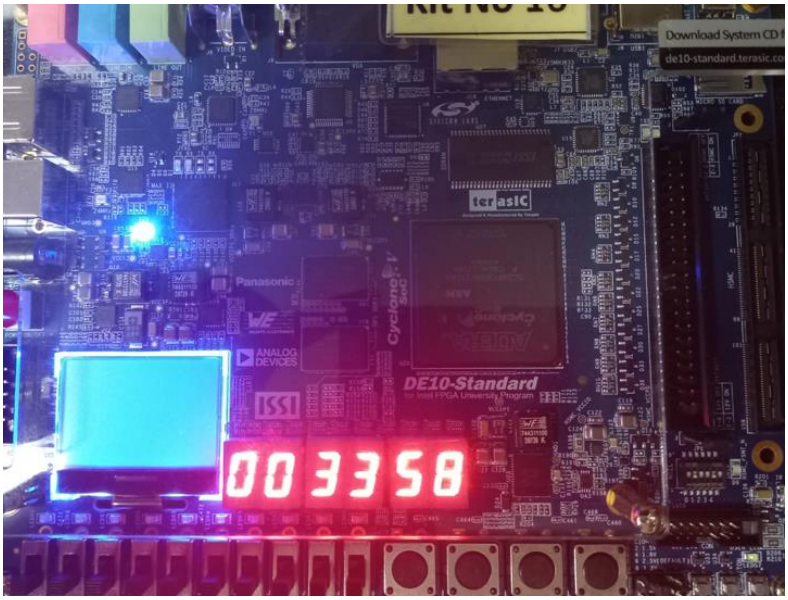
1. Write a Verilog file that instantiates your dual-port memory. To see the RAM contents, add to your design a capability to display the content of each four-bit word (in hexadecimal format) on the 7-segment display HEX0. Use a counter as a read address, and scroll through the memory locations by displaying each word for about one second. As each word is being displayed, show its address (in hex format) on the 7-segment displays HEX3−2. Use the 50 MHz clock, CLOCK_50, and use KEY0 as a reset input. For the write address and corresponding data use switches SW8−4 and SW3−0. Show the write address on HEX5−4 and show the write data on HEX1. Make sure that you properly synchronize the slide switch inputs to the 50 MHz clock signal.
2. Test your circuit and verify that the initial contents of the memory match your ram32x4.mif file. Make sure that you can independently write data to any address by using the slide switches.

```
module task_4 (
    input CLOCK_50,
    input [3:0] KEY,
    input [9:0] SW,
    output [6:0] HEX0,
    output [6:0] HEX1,
    output [6:0] HEX2,
    output [6:0] HEX3,
    output [6:0] HEX4,
    output [6:0] HEX5
);

    wire clk;
    clock_divider unitSec (
        .clk_in (CLOCK_50),
        .clk_out(clk)
    );

    wire [4:0] raddr;
    counter reader (.clk(clk), .count(raddr));

    wire wren = SW[9];
    wire [3:0] data = SW[3:0];
```

```
    wire [4:0] waddr = SW[8:4];
    wire [3:0] out;

    dual_ram32x4 unitDRAM (
        .clock(CLOCK_50),
        .data(data),
        .rdaddress(raddr),
        .wraddress(waddr),
        .wren(wren),
        .q(out)
    );

    decoder digitOut (
        .in (out),
        .HEX(HEX0)
    );
    decoder digitWrite (
        .in (data),
        .HEX(HEX1)
    );
    decoder digitRAddr0 (
        .in (raddr[3:0]),
        .HEX(HEX2)
    );
    decoder digitRAddr1 (
        .in (raddr[4]),
        .HEX(HEX3)
    );

    decoder digitWAddr0 (
        .in (SW[7:4]),
        .HEX(HEX4)
    );
    decoder digitWAddr1 (
        .in (SW[8]),
        .HEX(HEX5)
    );

endmodule

module clock_divider (
    input clk_in,  // 50 MHz input clock
    output reg clk_out
);

    reg [25:0] count = 0;  // Initialize count to zero

    always @(posedge clk_in) begin
        count <= count + 1;
```

```
        if (count == 26'd24_999_999) begin
            count   <= 0;
            clk_out <= ~clk_out;  // Invert the output clock
        end
    end

endmodule

module counter (
    input clk,
    output reg [4:0] count
);

    always @(posedge clk) begin
        if (count > 31) count <= 0;
        else count <= count + 1;
    end

endmodule
```





4 Conclusion

In this exercise, we have examined the general issues involved in implementing memory in an FPGA. We have discussed the different types of memory that can be implemented, the factors that need to be considered when choosing a memory type, and the steps involved in implementing memory.

We have learned that there are many different types of memory that can be implemented in an FPGA. The type of memory that is best for a particular application depends on the specific requirements of that application. The factors that need to be considered when choosing a memory type include the amount of memory that is needed, the speed of the memory, and the cost of the memory.