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: 14/05/2023

Semester: 6th Section: BEE 12C

EE-421: Digital System Design

Lab 12: Implementation of a FIFO Memory

Group Members

Name	Reg. No	PLO4-CLO3		PLO5 - CLO4	PLO8 - CLO5	PLO9 - CLO6
		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 Implementation of a FIFO Memory

2.1 Objectives

The purpose of this exercise is an implementation of a FIFO memory using LEDs, DIP Switches & Push Buttons. Each circuit will be described in Verilog and implemented on an Intel FPGA DE10-Lite, DE0-CV, DE1-SoC, or DE2- 115 board.

2.2 Introduction

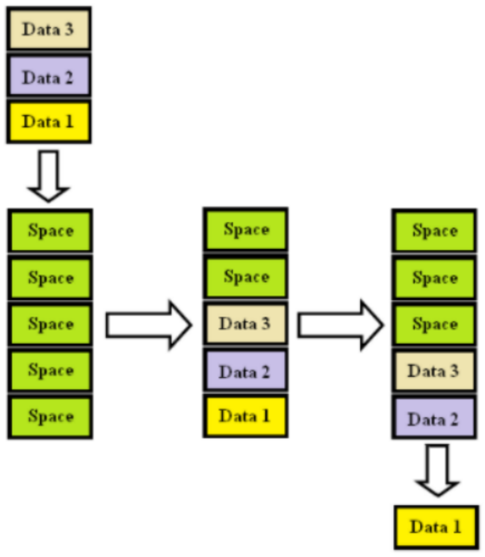
FIFO is an acronym for First In, First Out, which describes how data is managed relative to time or priority. In this case, the first data that arrives will also be the first data to leave from a group of data. A FIFO Buffer is a read/write memory array that automatically keeps track of the order in which data enters the module and reads the data out in the same order. In hardware FIFO buffer is used for synchronization purposes. It is often implemented as a circular queue, and has two pointers:

- Read Pointer/Read Address Register
- Write Pointer/Write Address Register

Read and write addresses are initially both at the first memory location and the FIFO queue is Empty. When the difference between the read address and write address of the FIFO buffer is equal to the size of the memory array then the FIFO queue is Full. FIFO can be classified as synchronous or asynchronous depending on whether the same clock (synchronous) or different clocks (asynchronous) control the read and write operations.

2.3 Synchronous FIFO

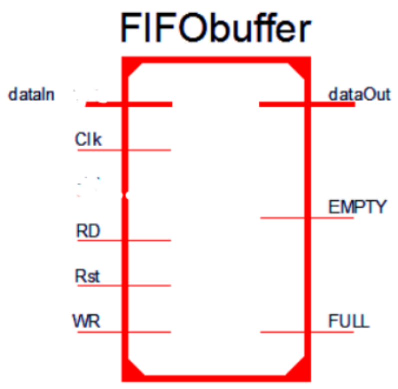
A synchronous FIFO refers to a FIFO design where data values are written sequentially into a memory array using a clock signal, and the data values are read out sequentially from the memory array using the same clock signal. Figure 1 shows the flow of the operation of a typical FIFO.



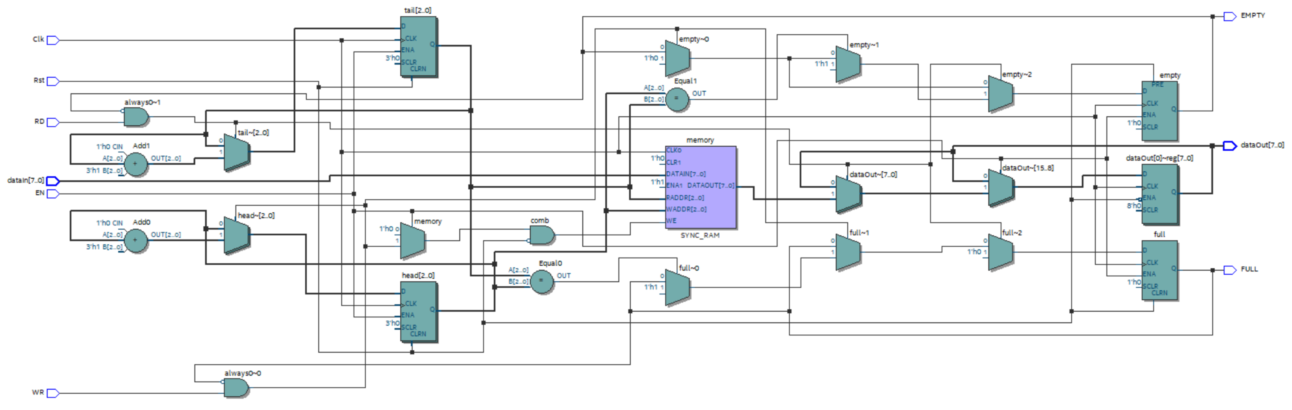
3 Lab Procedure

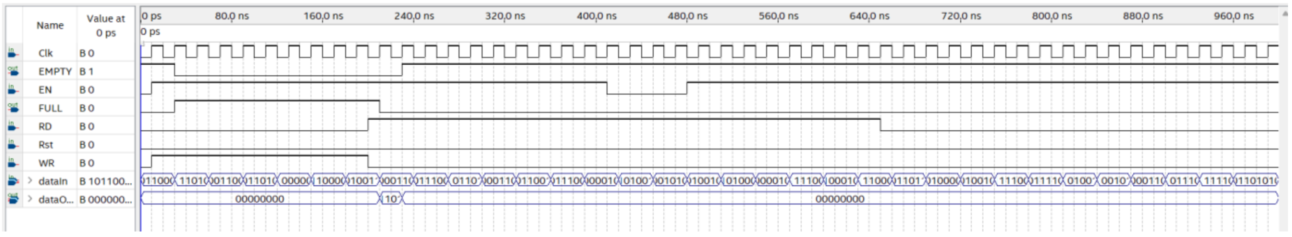
3.1 Part I

Implement 8x8 FIFO memory as shown in Figure 2. This FIFO Buffer can store eight 8-bit values. The FIFO Buffer module consists of a 8-bit data input line, dataIn and a 8-bit data output line, dataOut. The module is clocked using the 1-bit input clock line Clk. The module also has a 1-bit enable line, EN and a 1-bit active high reset line, Rst. The 1-bit RD line is used to signal a data read operation on the FIFO Buffer and the 1-bit WR line is used to signal a data write operation on the FIFO Buffer. Both the RD and WR lines are active high. The module also has two output lines FULL and EMPTY which are each 1-bit wide. The FULL line becomes high when the FIFO Buffer is or becomes full (internal counter becomes eight). The EMPTY line becomes high when the FIFO Buffer is or becomes empty (internal counter becomes zero).



```
module lab_12 (
    input Clk,
    input Rst,
    input EN,
    input RD,
    input WR,
    input [7:0] dataIn,
    output reg [7:0] dataOut,
    output FULL,
    output EMPTY
);
    reg [7:0] memory[0:7];
    reg [2:0] head;
    reg [2:0] tail;
    reg full;
    reg empty;

    always @(posedge Clk or posedge Rst) begin
        if (Rst) begin
            head  <= 3'b000;
            tail  <= 3'b000;
            full  <= 1'b0;
            empty <= 1'b1;
        end else if (EN) begin
            if (WR && !full) begin
                memory[head] <= dataIn;
                head <= head + 1;
                if (head == tail) full <= 1'b1;
                empty <= 1'b0;
            end

            if (RD && !empty) begin
                dataOut <= memory[tail];
                tail <= tail + 1;
                if (tail == head) empty <= 1'b1;
                full <= 1'b0;
            end
        end
    end

    assign FULL  = full;
    assign EMPTY = empty;

endmodule
```



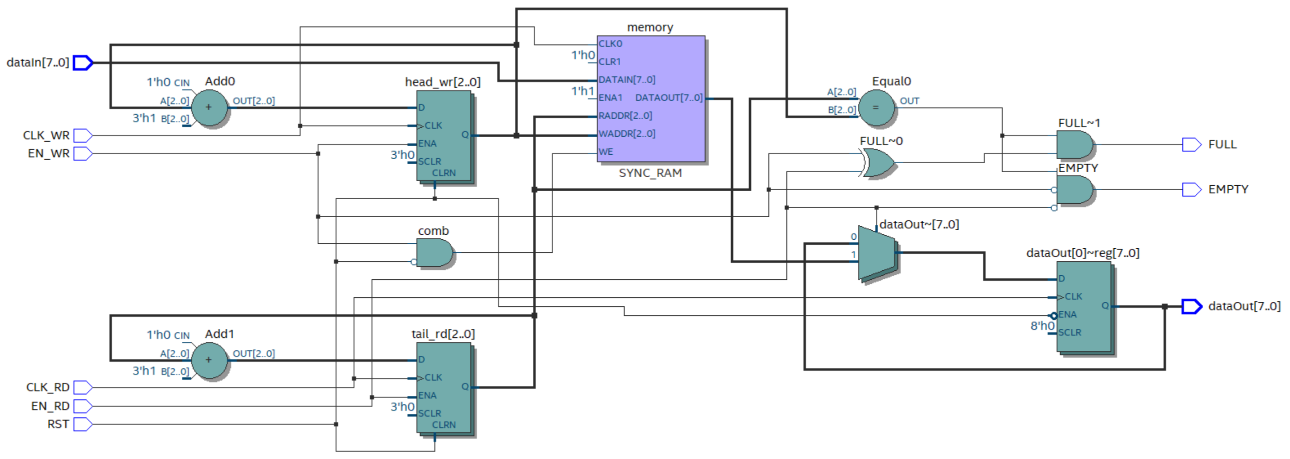


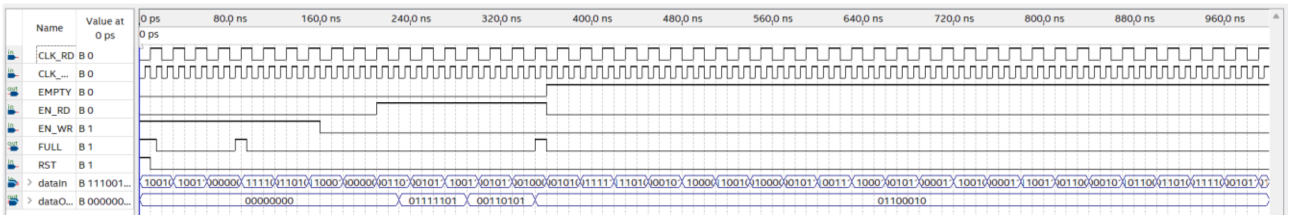
3.2 Part II

Implement asynchronous FIFO with the same specifications as mentioned above for synchronous FIFO. (Hint: Asynchronous FIFO is a FIFO design in which data is written from one clock domain to the FIFO buffer and read from the same FIFO buffer in another clock domain, which is asynchronous to each other.)

```
module lab_12 (
    input CLK_WR,
    input CLK_RD,
```

```verilog
    input RST,
    input EN_WR,
    input EN_RD,
    input [7:0] dataIn,
    output reg [7:0] dataOut,
    output FULL,
    output EMPTY
);
    reg [7:0] memory[0:7];
    reg [2:0] head_wr;
    reg [2:0] tail_rd;
    wire full;
    wire empty;

    always @(posedge CLK_WR or posedge RST) begin
        if (RST) begin
            head_wr <= 3'b000;
        end else if (EN_WR && !full) begin
            memory[head_wr] <= dataIn;
            head_wr <= head_wr + 1;
        end
    end

    always @(posedge CLK_RD or posedge RST) begin
        if (RST) begin
            tail_rd <= 3'b000;
        end else if (EN_RD && !empty) begin
            dataOut <= memory[tail_rd];
            tail_rd <= tail_rd + 1;
        end
    end

    assign FULL  = (head_wr == tail_rd) && (EN_WR != EN_RD);
    assign EMPTY = (head_wr == tail_rd) && (!EN_WR && !EN_RD);

endmodule
```





4 Conclusion

In this exercise, we explored the implementation of synchronous and asynchronous FIFO memory modules in Verilog and verified their performance on an Intel FPGA DE10-Standard. Through testing and analysis, we gained valuable insights into the efficient management of data flow between different clock domains. The synchronous FIFO demonstrated synchronized data transfer, while the asynchronous FIFO enabled flexible integration in systems with multiple clock domains. This exercise deepened our understanding of FIFO memory designs, their trade-offs, and their significance in digital systems for reliable data storage and retrieval.