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 13: Open Ended Lab

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 Open Ended Lab

2.1 Objectives

This is an exercise in using algorithmic state machine charts to implement algorithms as hardware circuits.

2.2 Introduction

Algorithmic state machine charts (ASM charts) are graphical notations for representing algorithms. ASM charts are like flowcharts, but they have several advantages for representing algorithms in hardware. First, ASM charts can be used to represent both sequential and combinational logic. Second, ASM charts can be used to represent complex algorithms that would be difficult to represent with a flowchart. Third, ASM charts can be used to generate VHDL code for the implemented circuits.

In this lab, you will learn how to use ASM charts to represent algorithms. You will then implement several algorithms as hardware circuits using ASM charts. Finally, you will analyze the performance of the implemented circuits

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

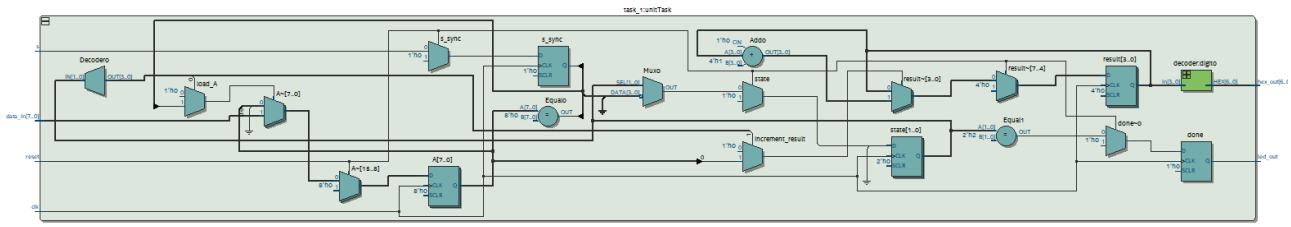
3.1 Task 1

Write Verilog code to implement the bit-counting circuit using the ASM chart shown in Figure 1 on a DE-series board. Include in your Verilog code the data path components needed and make an FSM for the control circuit. The inputs to your circuit should consist of an 8-bit input connected to slide switches SW7−0, a synchronous reset connected to KEY0, and a start signal (s) connected to switch SW9. Use the 50 MHz clock signal provided on the board as the clock input for your circuit. Be sure to synchronize the s signal to the clock. Display the number of 1s counted in the input data on the 7-segment display HEX0, and signal that the algorithm is finished by lighting up LEDR9.

```
module task_1 (
    input [7:0] data_in,
    input clk,
    input reset,
    input s,
    output [6:0] hex_out,
    output led_out
);

    // Define datapath components
    reg [7:0] A;
    reg [3:0] result;
    reg done;

    // Define control circuit states
    parameter S1 = 2'b00;
    parameter S2 = 2'b01;
    parameter S3 = 2'b10;
    reg [1:0] state;

    // Define control circuit outputs
    reg load_A;
    reg shift_A;
    reg increment_result;
    reg next_state;

    // Synchronize start signal to the clock
    reg s_sync;
    always @(posedge clk) begin
        if (reset) begin
            s_sync <= 1'b0;
        end else begin
            s_sync <= s;
        end
```

```
end

// Control circuit
always @(*) begin
    case (state)
        S1: begin
            // Load data into A and transition to S2 when s is asserted
            load_A = s_sync;
            shift_A = 1'b0;
            increment_result = 1'b0;
            next_state = s_sync ? S2 : S1;
        end

        S2: begin
            // Increment result and shift A to the right
            load_A = 1'b0;
            shift_A = 1'b1;
            increment_result = A[0];
            next_state = (A == 8'b0) ? S3 : S2;
        end

        S3: begin
            // Set done signal and wait for s to be deasserted
            load_A = 1'b0;
            shift_A = 1'b0;
            increment_result = 1'b0;
            next_state = (s_sync) ? S3 : S1;
        end

        default: begin
            // Invalid state, reset to S1
            load_A = 1'b0;
            shift_A = 1'b0;
            increment_result = 1'b0;
            next_state = S1;
        end
    endcase
end

// Datapath circuit
always @(posedge clk) begin
    if (reset) begin
        A <= 8'b0;
        result <= 4'b0;
        done <= 1'b0;
        state <= S1;
    end else begin
```

```
            // Load data into A on S1, shift A on S2
            A <= (load_A) ? data_in : {A[6:0], 1'b0};

            // Increment result on S2
            if (increment_result) begin
                result <= result + 1;
            end

            // Set done signal on S3
            done  <= (state == S3) ? 1'b1 : 1'b0;

            // Update control circuit state
            state <= next_state;
        end
    end

    decoder digit0(.in(result), .HEX(hex_out));

    assign led_out = (done) ? 1'b1 : 1'b0;

endmodule
```



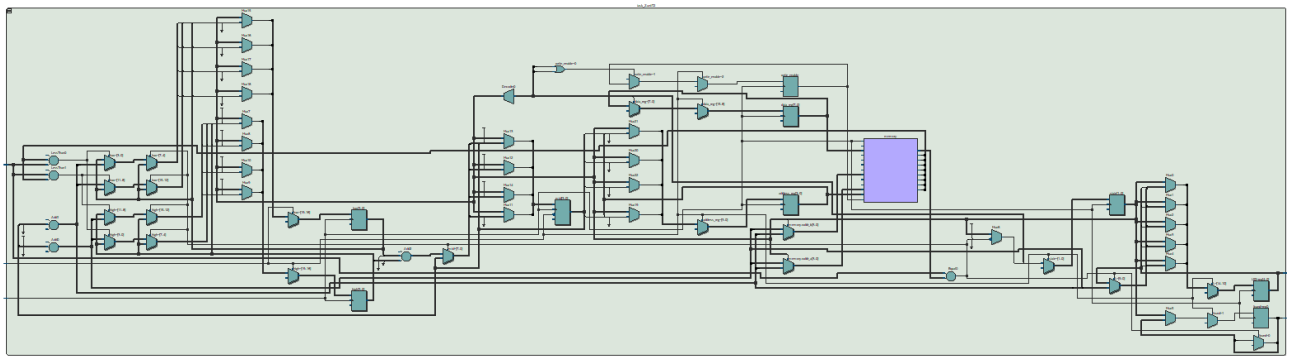
3.2 Task 2

1. Create an ASM chart for the binary search algorithm. Keep in mind that the memory has registers on its input ports. Assume that the array has a fixed size of 32 elements.
2. Implement the FSM and the datapath for your circuit.
3. Connect your FSM and datapath to the memory block as indicated in Figure 2.
4. Include in your project the necessary pin assignments to implement your circuit on your DE-series board. Use switch SW9 to drive the Start input, use SW7...0 to specify the value A, use KEY0 for Resetn, and use the board's 50 MHz clock signal as the Clock input (be sure to synchronize the Start input to the clock). Display the address of the data A, if found, on 7-segment displays HEX1 and HEX0, as a hexadecimal number. Finally, use LEDR9 for the Found signal.
5. Create a file called my_array.mif and fill it with an ordered set of 32 eight-bit integer numbers.
6. Compile your design, and then download and test it.

```
module task_2 (
    input wire clk,
    input wire reset,
    input wire [7:0] A,
```

```
    output reg [4:0] L,
    output reg found
);

    // Memory module for storing the sorted array
    reg [7:0] memory[0:15];
    reg [3:0] address_reg;
    reg [7:0] data_reg;
    reg write_enable;

    // FSM states
    parameter IDLE = 2'b00;
    parameter READ_MIDDLE = 2'b01;
    parameter SEARCH_LOWER = 2'b10;
    parameter SEARCH_UPPER = 2'b11;

    // FSM registers
    reg [1:0] state;
    reg [3:0] low;
    reg [3:0] high;
    reg [3:0] mid;

    always @(posedge clk) begin
        if (reset) begin
            // Initialize FSM and memory module
            state <= IDLE;
            low <= 0;
            high <= 15;
            found <= 0;
            write_enable <= 0;
            address_reg <= 0;
            data_reg <= 0;
            L <= 0;
        end else begin
            case (state)
                IDLE: begin
                    // Reset memory module
                    write_enable <= 0;
                    address_reg <= 0;
                    data_reg <= 0;

                    // Initialize search
                    low <= 0;
                    high <= 15;
                    mid <= 7;
                    state <= READ_MIDDLE;
                end

                READ_MIDDLE: begin
                    // Read middle element from memory
                    address_reg <= mid;
                    write_enable <= 0;
                    state <= state + 1;
                end

                SEARCH_LOWER: begin
                    // Check if A is in lower half of array
```

```
                    if (memory[mid-1] == A) begin
                        L <= mid - 1;
                        found <= 1;
                        state <= IDLE;
                    end else if (memory[mid-1] < A) begin
                        low   <= mid + 1;
                        mid   <= (low + high) / 2;
                        state <= READ_MIDDLE;
                    end else begin
                        high  <= mid - 1;
                        mid   <= (low + high) / 2;
                        state <= READ_MIDDLE;
                    end
                end

                SEARCH_UPPER: begin
                    // Check if A is in upper half of array
                    if (memory[mid+1] == A) begin
                        L <= mid + 1;
                        found <= 1;
                        state <= IDLE;
                    end else if (memory[mid+1] > A) begin
                        high  <= mid - 1;
                        mid   <= (low + high) / 2;
                        state <= READ_MIDDLE;
                    end else begin
                        low   <= mid + 1;
                        mid   <= (low + high) / 2;
                        state <= READ_MIDDLE;
                    end
                end
            endcase
        end
    end

    // Memory module for storing the sorted array
    always @(posedge clk) begin
        if (write_enable) begin
            memory[address_reg] <= data_reg;
        end
    end

endmodule
```



4 Conclusion

In conclusion, this lab report has introduced the concept of Algorithmic State Machine (ASM) charts and highlighted their advantages over flowcharts for representing algorithms in hardware. The report has emphasized the ability of ASM charts to represent both sequential and combinational logic, complex algorithms, and their ability to generate VHDL code for the implemented circuits. The lab exercises provided an opportunity to learn how to use ASM charts to represent algorithms and implement them as hardware circuits. The performance of the implemented circuits was also analyzed. By completing this lab, the participants have gained valuable experience and understanding of ASM charts, which can be applied to future hardware design projects.