Bialystok University of Technology

Faculty of Computer Science

EMBEDDED SYSTEMS FCS-00072

Report

Lab 2: Combinational circuits

Teacher: Adam Klimowicz

Student: Sevda Ghasemzadehnaghadehy

Task 1

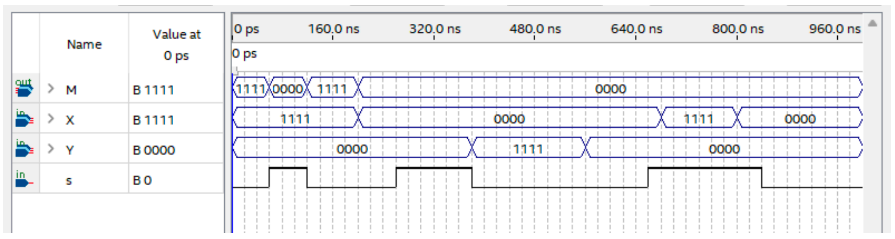
Objective:

1. Create a new Quartus project for your circuit.
2. Include your Verilog file for the four-bit wide 2-to-1 multiplexer in your project.
3. Compile and then simulate the project.

Verilog code:

```
module Lab2 (
    input [3:0] X,
    input [3:0] Y,
    input s,
    output [3:0] M
);

assign M[0] = (~s & X[0]) | (s & Y[0]);
assign M[1] = (~s & X[1]) | (s & Y[1]);
assign M[2] = (~s & X[2]) | (s & Y[2]);
assign M[3] = (~s & X[3]) | (s & Y[3]);

endmodule
```

Simulation



Conclusions:

In this task I created a circuit that chooses between two 4-bit inputs depending on a single control input. I wrote the code in Verilog and used Quartus Prime's simulation tool to test it. I tried different values for the inputs and changed the control signal to see if the output would follow the right one. Then I ran the simulation using the University Program method and looked at the waveform. The output changed as I expected, so I understood that my circuit worked correctly.

Task 2

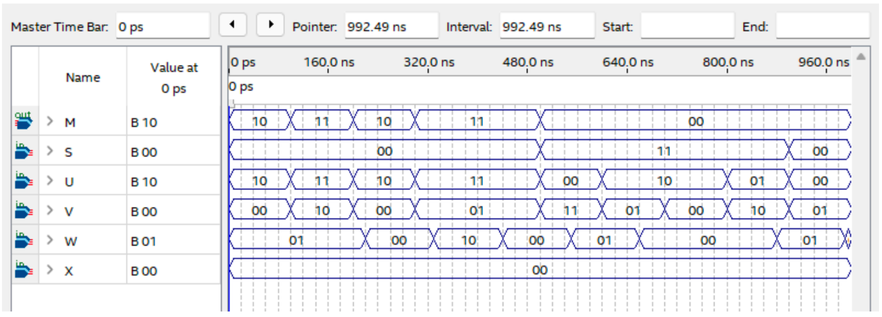
Objective:

Perform the following steps to implement the two-bit wide 4-to-1 multiplexer.
1. Create a new Quartus project for your circuit.
2. Create a Verilog module for the two-bit wide 4-to-1 multiplexer.
3. Compile project and test the functionality of the two-bit wide 4-to-1 multiplexer by simulating of toggling the inpots and observing the outputs.

Verilog code:

```
module Lab2_part2 (
    input [1:0] U,
    input [1:0] V,
    input [1:0] W,
    input [1:0] X,
    input [1:0] S,
    output reg [1:0] M
);

always @(*) begin
    case (S)
        2'b00: M = U;
        2'b01: M = V;
        2'b10: M = W;
        2'b11: M = X;
        default: M = 2'b00;
    endcase
end

endmodule
```

Simulation:



Master Time Bar: 0 ps   Pointer: 992.49 ns   Interval: 992.49 ns   Start: End:

Name	Value at 0 ps
M	B 10
S	B 00
U	B 10
V	B 00
W	B 01
X	B 00

Conclusions:

In the second part of the lab, I designed a 2-bit wide 4-to-1 multiplexer using Verilog and tested it with the University Program waveform simulation tool in Quartus Prime. The circuit chooses between four 2-bit inputs (U, V, W, X) depending on a 2-bit selector input (S). I applied different input values and changed S during the simulation to check if the output M matched the correct input. The waveform results showed that the output changed as expected in each case.

Task 3

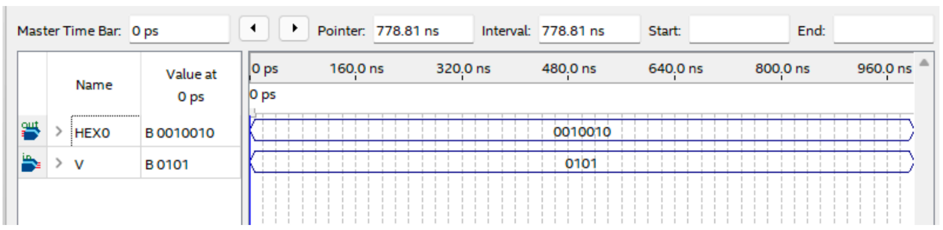
Objective:

1. Write a Verilog file that provides the necessary functionality. Include this file in your project and assign the pins on the FPGA to connect to the switches and 7-segment displays. Make sure to include the necessary pin assignments.
2. Compile the project and simulate using waveforms.

Verilog code:

```
module Lab2_part3 (
    input [3:0] V,
    output reg [6:0] HEX0
);

always @(*) begin
    case (V)
        4'd0: HEX0 = 7'b1000000;
        4'd1: HEX0 = 7'b1111001;
        4'd2: HEX0 = 7'b0100100;
        4'd3: HEX0 = 7'b0110000;
        4'd4: HEX0 = 7'b0011001;
        4'd5: HEX0 = 7'b0010010;
        4'd6: HEX0 = 7'b0000010;
        4'd7: HEX0 = 7'b1111000;
        4'd8: HEX0 = 7'b0000000;
        4'd9: HEX0 = 7'b0010000;
        default: HEX0 = 7'b1111111;   // Don't care
    endcase
end

endmodule
```

Simulation:



```
Master Time Bar: 0 ps    Pointer: 778.81 ns    Interval: 778.81 ns    Start:    End:
```

Name	Value at 0 ps	0 ps – 960.0 ns
HEX0	B 0010010	0010010
V	B 0101	0101

Conclusions:

In this part of the lab, I designed a binary to 7-segment display decoder in Verilog. The circuit takes a 4-bit input and displays numbers from 0 to 9 on a 7-segment display. I tested the functionality using waveform simulation in Quartus Prime. For example, when the input value was 0101 (which is decimal 5), the HEX0 output correctly turned on the segments that form the number 5 on a 7-segment display. The simulation showed that the circuit worked as expected and displayed the right numbers for all valid inputs between 0 and 9.

Task 4

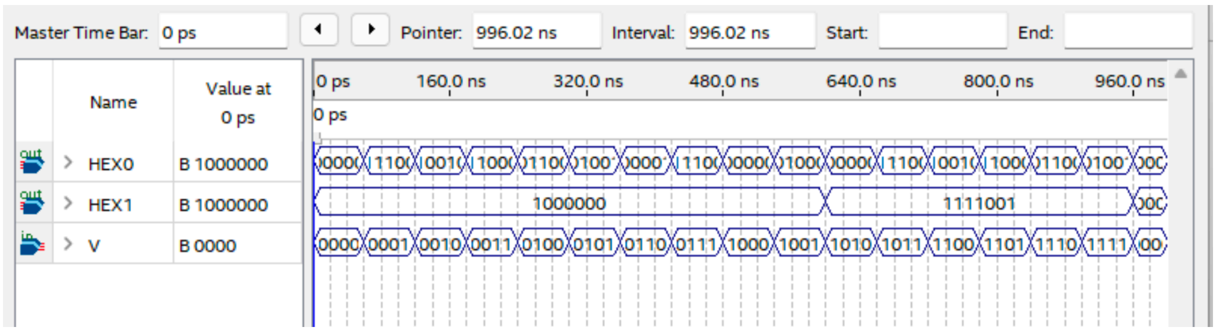
Objective:

1. Write Verilog code to implement your design. Make a Quartus project for your Verilog module.
2. Compile the circuit and use functional simulation to verify the correct operation of your comparator, multiplexers, and circuit A.
3. Test the circuit by trying all possible values of V and observing the output displays using simulator.

Verilog code:

```
module Lab2_Part4 (
    input [3:0] V,
    output reg [6:0] HEX0,
    output reg [6:0] HEX1
);

wire z;
wire [3:0] A;
wire [3:0] digit0;

assign z = (V > 4'd9);

assign A = V - 4'd10;

assign digit0 = (z == 1'b0) ? V : A;

always @(*) begin
    case (z)
        1'b0: HEX1 = 7'b1000000;
        1'b1: HEX1 = 7'b1111001;
    endcase
end

always @(*) begin
    case (digit0)
        4'd0: HEX0 = 7'b1000000;
        4'd1: HEX0 = 7'b1111001;
        4'd2: HEX0 = 7'b0100100;
        4'd3: HEX0 = 7'b0110000;
        4'd4: HEX0 = 7'b0011001;
        4'd5: HEX0 = 7'b0010010;
        4'd6: HEX0 = 7'b0000010;
        4'd7: HEX0 = 7'b1111000;
        4'd8: HEX0 = 7'b0000000;
        4'd9: HEX0 = 7'b0010000;
        default: HEX0 = 7'b1111111;
    endcase
end

endmodule
```

Simulation:



Master Time Bar: 0 ps | Pointer: 996.02 ns | Interval: 996.02 ns | Start: | End:

Name	Value at 0 ps
HEX0	B 1000000
HEX1	B 1000000
V	B 0000

```
0 ps   160.0 ns   320.0 ns   480.0 ns   640.0 ns   800.0 ns   960.0 ns
HEX0: 0000 1100 0010 1000 0110 0100 0000 1100 0000 0100 0000 1100 0010 1000 0110 0100 00C
HEX1: 1000000 1111001 00C
V:    0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 00
```

Conclusions:

In this part, I tested the designed circuit by giving all possible 4-bit binary values from 0 to 15 as input and observed the output on two 7-segment displays. The circuit successfully converted each binary input into its corresponding two-digit decimal form, showing the correct ones and tens digits separately. For values less than or equal to 9, the display correctly showed 0 on the tens display and the exact value on the ones display. For values greater than 9, the circuit displayed 1 on the tens display and the correct remainder (value minus 10) on the ones display. The waveform simulation results matched with the expected outputs, confirming that the circuit works as intended for all input combinations.