Lab Report 4:

Logic Lab

July 3, 2019

Group #2

Nathan Phipps, Erich Wanzek

Abstract

In this lab we examined Karnaugh maps, selectors and basic logic design principles to create/implement Verilog code for multi bit multiplexers. In accordance with our lab rubric this lab report will outline our approach to solving the problems set forth within the rubric's guidelines. Thus, the body of our lab report will exist as follows:

1.) **Analysis:** Analysis of the results contained within Lab 4,

2.) **Group Member Contributions:** A brief discussion on group member contributions,

3.) **Components for Part 1:**
   A.) Karnaugh Maps,
   B.) Final Equations from Karnaugh Maps,
   C.) Block Diagram From Equations,
   D.) FPGA Trials: Pictures of our FPGA board trials (as requested in part 2 of the rubric),

4.) **Components for Part 2:**
   A.) Verilog implementation Code,
   B.) Top Level Entity,
   C.) Arbitrary Trial Photos,

5.) **Components for Part 3:**
   A.) Verilog implementation Code (Note: the rubric didn't request a top level code for this part)
   B.) Final Equations for Implementation,
   B.) Wave Diagram Simulation: Pictures of the wave diagram simulations (as requested in part 3 of the rubric).

Analysis

Analysis Part 1:

In part 1 of Lab 4, we examined a two bit multiplexer, specifically a 2 to 1 “MUX”. During this initial part of the lab, we utilized Karnaugh maps to create minimized expressions (final equations), used for implementation in a Verilog code for a 2 to 1, “2 bits,” “MUX.” Additionally, we were tasked to perform multiple trials in order to display whether our code worked, or not. From these trials, we were able to determine that our code, did, in fact, perform as expected, thus providing us with the correct results.

Analysis Part 2:

During this part of the lab, we wrote Verilog codes to create and implement a 4 to 1 single bit multiplexer. One of these codes included a module that handled the creation of the multiplexer, while the other code was set as the top level entity. Additionally we utilized code from a past project to implement functions pertaining to the utilization of the FPGA board’s seven segment display. Since the rubric didn’t request a simulation here, we instead opted to upload our code to our boards, while taking arbitrary trial photos of different settings, as they pertained to the multiplexer. These trials confirmed our data.

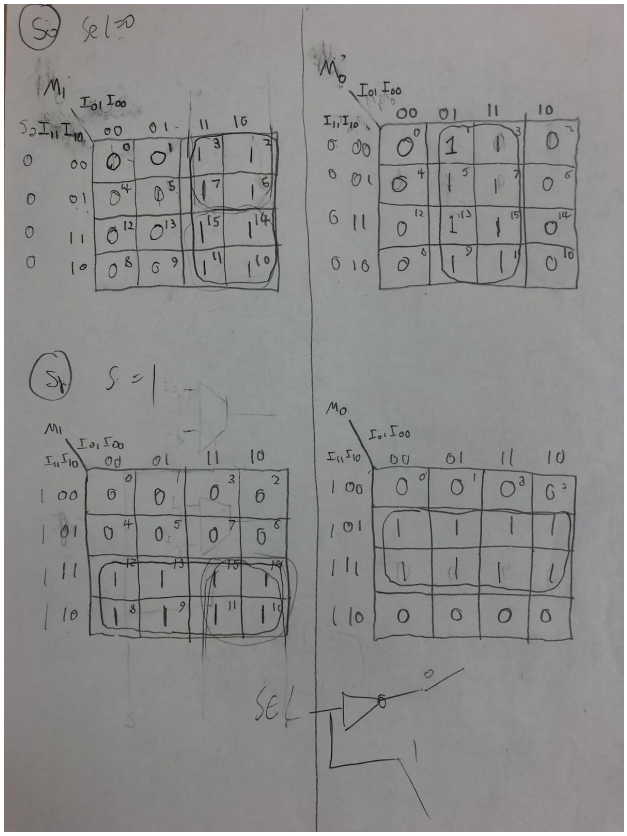
Analysis Part 3:

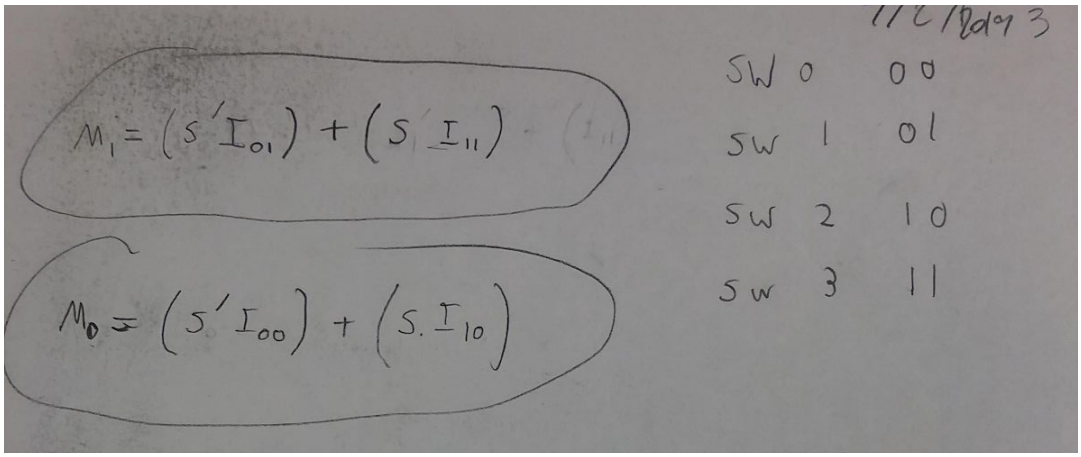
In part three of this lab we designed a 4 to 1 4-bit multiplexer implemented as a verilog code module. The 4 to 1 4-bit multiplexer could be implemented by extending out the equations of the two to one multiplexer, but we decided it was easier to instantiate the four to one 1_bit multiplexer module four times, making four parallel 4 to 1 bit multiplexers functioning as a whole as a 4 to 1 4-bit multiplexer. We ran a functional timing simulation of the 4 to 1 4-bit multiplexer. Upon a detailed inspection of the timing diagram for the 4 to 1 4-bit multiplexer, we agreed that the 4 to 1 4-bit multiplexer functioned properly.

Group Member Contributions

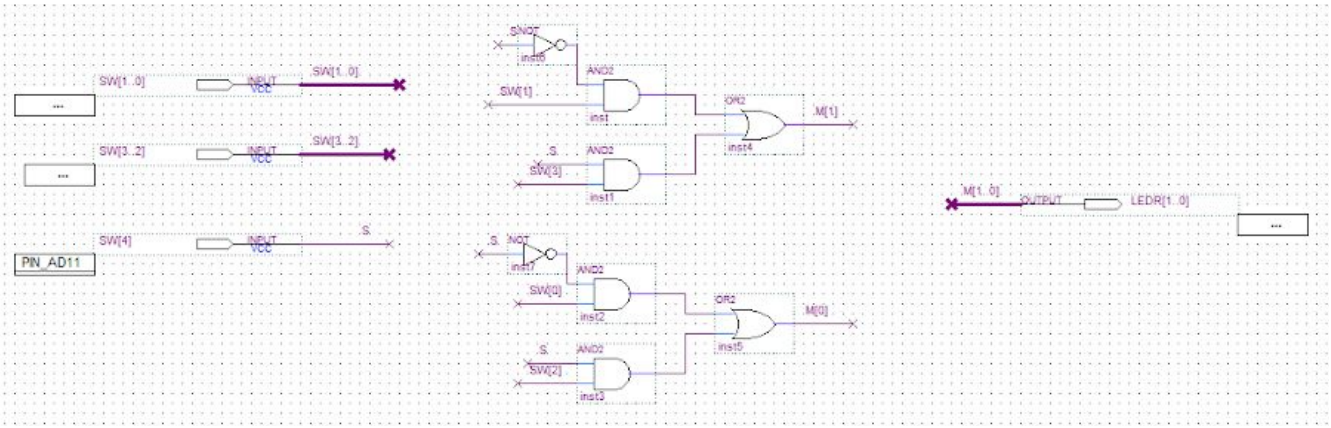
Throughout the entirety of this lab both group members, Nathan Phipps, and Erich Wanzek, performed equal duties, including the creation of Block diagrams, Karnaugh maps, solving for final equations, writing Verilog code, performing wave diagram simulations, compiling data, taking photos and typing this report.

COMPONENTS FOR PART 1

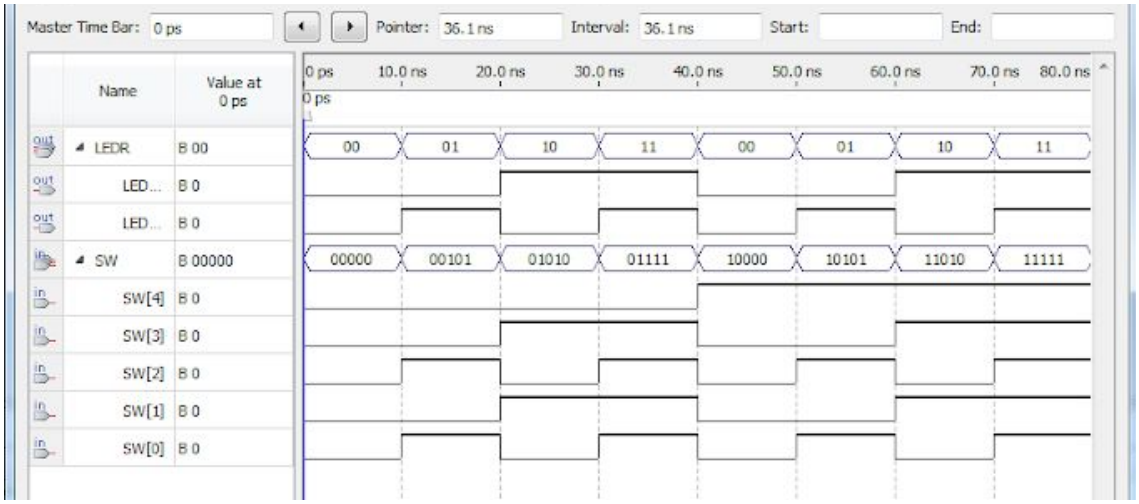




Block Diagram for Part 1 (2 to 1 With 2 Bits Multiplexer):

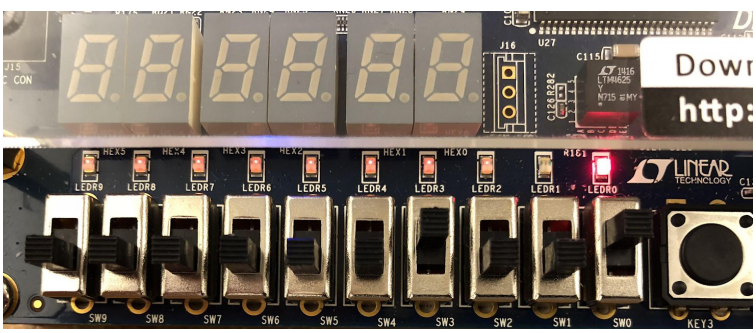


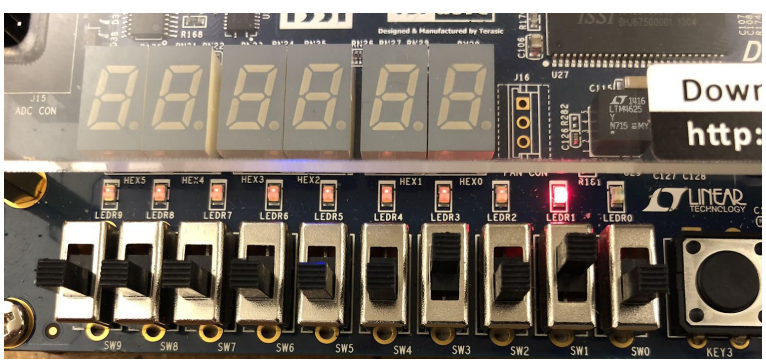
Wave Simulation Diagram for Part 1:

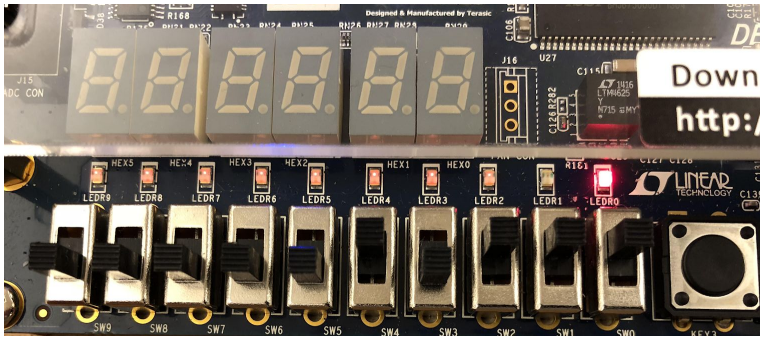


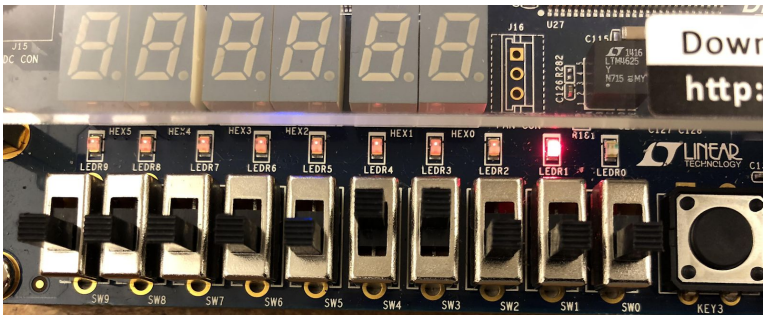
TEST CASES for PART 1

Inputs					Output	
Select	I_1		I_0		m	
0	1	0	0	1	0	1
0	1	0	1	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	0









COMPONENTS FOR PART 2

Final Equation, 4 to 1, 1 bit multiplexer:

(D[3] & S[0] & S[1]) | (D[2]& ~S[0]& S[1]) | (D[1] & S[0] & ~S[1]) | (D[0] & ~S[0] & ~S[1]);

Verilog Code for Part 2 (4 to 1, 1 Bit Multiplexer):

```
module Four_to_1_1_bit_mux(O,D,S); //declares module

        input wire [3:0]D; // four data inputs
        input wire [1:0]S; // two selectors
        output wire O;   //define input wire

        assign O = (D[3] & S[0] & S[1]) |  (D[2]& ~S[0]& S[1]) |  (D[1] & S[0] & ~S[1]) | (D[0] & ~S[0] & ~S[1]); //assigns logical expression to output for
        the 4 to 1 MUX

endmodule //ends module
```

Verilog Code for Part 2 (Seven Segment Display):

```
module hex_7seg_bitwise (X, segment); //module name 7 segment display, part 1e

        input wire [3:0]X; //defines a wire that's 4 bits wide
        output wire [6:0]segment; //defines an output wire that is 7 bits wide

        // Base for copying and pasting to make typing segments easier: A[3]&B[2]&C[1]&D[0] |

        // S0 = Sum(1,4,11,13)
        // Segment[0] = ~A[3]&~B[2]&~C[1]&D[0] | ~A[3]&B[2]&~C[1]&~D[0] | A[3]&B[2]&~C[1]&D[0] | A[3]&~B[2]&C[1]&D[0];
        assign segment[0] = ~X[3]&~X[2]&~X[1]&X[0] | ~X[3]&X[2]&~X[1]&~X[0] | X[3]&X[2]&~X[1]&X[0] | X[3]&~X[2]&X[1]&X[0];

        // S1 = Sum(5,6,11,12,14,15)
        // Segment[1] = A[3]&B[2]&~D[0] | ~A[3]&B[2]&~C[1]&D[0] | B[2]&C[1]&~D[0] | A[3]&C[1]&D[0];
        assign segment[1] = X[3]&X[2]&~X[0] | ~X[3]&X[2]&~X[1]&X[0] | X[2]&X[1]&~X[0] | X[3]&X[1]&X[0];

        // S2 = Sum(2,12,14,15)
        // Segment[2] = ~A[3]&~B[2]&C[1]&~D[0] | A[3]&B[2]&~D[0] | A[3]&B[2]&C[1];
        assign segment[2] = ~X[3]&~X[2]&X[1]&~X[0] | X[3]&X[2]&~X[0] | X[3]&X[2]&X[1];

        // S3 = Sum(1,4,7,9,10,15)
        // Segment[3] = ~A[3]&B[2]&~C[1]&~D[0] | ~B[2]&~C[1]&D[0] | B[2]&C[1]&D[0] | A[3]&~B[2]&C[1]&~D[0]
        assign segment[3] = ~X[3]&X[2]&~X[1]&~X[0] | ~X[2]&~X[1]&X[0] | X[2]&X[1]&X[0] | X[3]&~X[2]&X[1]&~X[0];

        // S4 = Sum(1,3,4,5,7,9)
        // Segment[4] = ~A[3]&B[2]&~C[1] | ~A[3]&D[0] | ~B[2]&~C[1]&D[0]
        assign segment[4] = ~X[3]&X[2]&~X[1] | ~X[3]&X[0] | ~X[2]&~X[1]&X[0];

        // S5 = Sum(1,2,3,7,13)
        // Segment[5] = ~A[3]&~B[2]&D[0] | ~A[3]&C[1]&D[0] | ~A[3]&~B[2]&C[1] | A[3]&B[2]&~C[1]&D[0]
        assign segment[5] = ~X[3]&~X[2]&X[0] | ~X[3]&X[1]&X[0] | ~X[3]&~X[2]&X[1] | X[3]&X[2]&~X[1]&X[0];

        // S6 = Sum(0,1,7,12)
        // Segment[6] = ~A[3]&~B[2]&~C[1] | ~A[3]&B[2]&C[1]&D[0] | A[3]&B[2]&~C[1]&~D[0]
        assign segment[6] = ~X[3]&~X[2]&~X[1] | ~X[3]&X[2]&X[1]&X[0] | X[3]&X[2]&~X[1]&~X[0];

endmodule //ends module
```

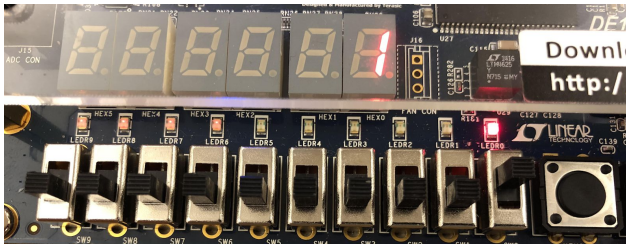
Verilog Code for Part 2 (Top Level Entity):

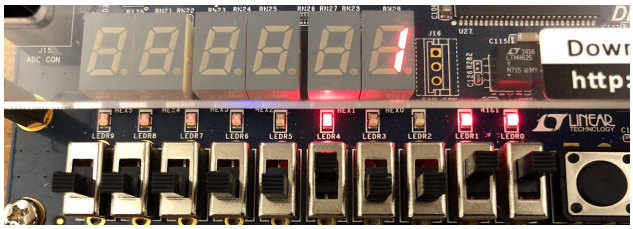
```
module top_level_4_1(HEX0, SW,LEDR); //declares module

        input wire [5:0]SW; //data and selection input

        output wire [5:0]LEDR; //outputs 6 wires for LEDR
        output wire [6:0]HEX0; //outputs 7 wires for our HEX display

        assign LEDR[5:0]    = SW[5:0]; //assigns LEDRs to switches

         wire m; //defines a wire m
        wire [3:0]X; //defines 4 wires X

        //assign {A, B, C, D} = {3'b000, m};

        Four_to_1_1_bit_mux (m,SW[3:0], SW[5:4]); //calls 4 to 1, 1 bit multiplexer, takes in m, and multiple switches

        assign X = {3'b000, m}; //concatenates the X wires

        hex_7seg_bitwise (X, HEX0); //calls hex_7seg_bitwise

endmodule //ends module
```

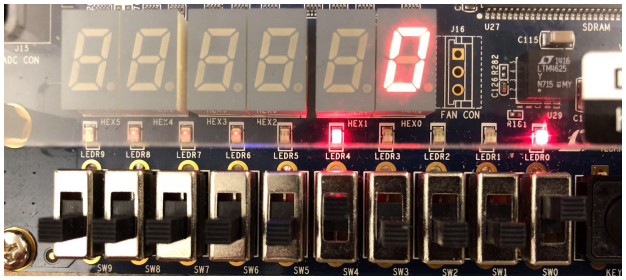
Verilog Code for Part 3 (4 to 1, 4 bits multiplexer):

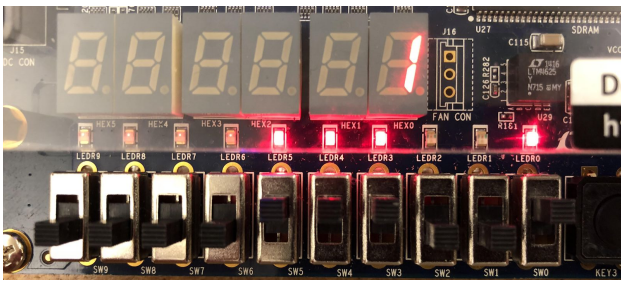
```
module Four_to_1_4_bit_mux(O,D,S); //declares module

        input wire [15:0]D; // four data inputs
        input wire [1:0]S; // two selector inputs
        output wire [3:0]O;   //define output data wire

        Four_to_1_1_bit_mux inst0(O[0],D[3:0],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst1(O[1],D[7:4],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst2(O[2],D[11:8],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst3(O[3],D[15:12],S); //calls the four to 1, 1 bit module from earlier.
        //using four of the four to 1, 1 bit functions from earlier, we can create a 4 to 1, 4 bits multiplexer.

endmodule //ends module
```

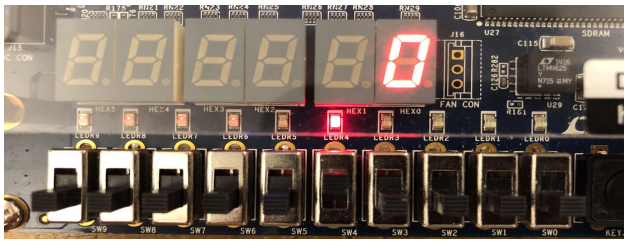
Arbitrary FPGA Trials From Part 2:

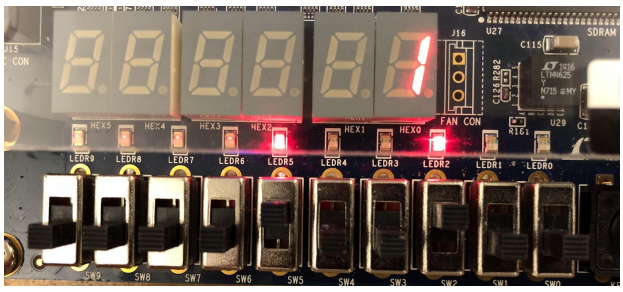


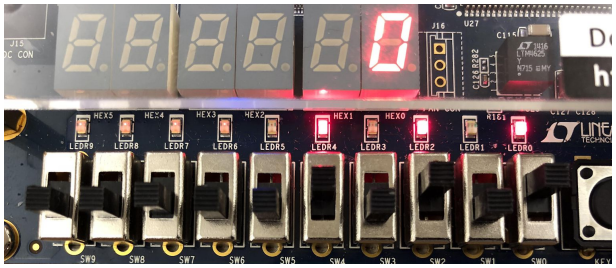










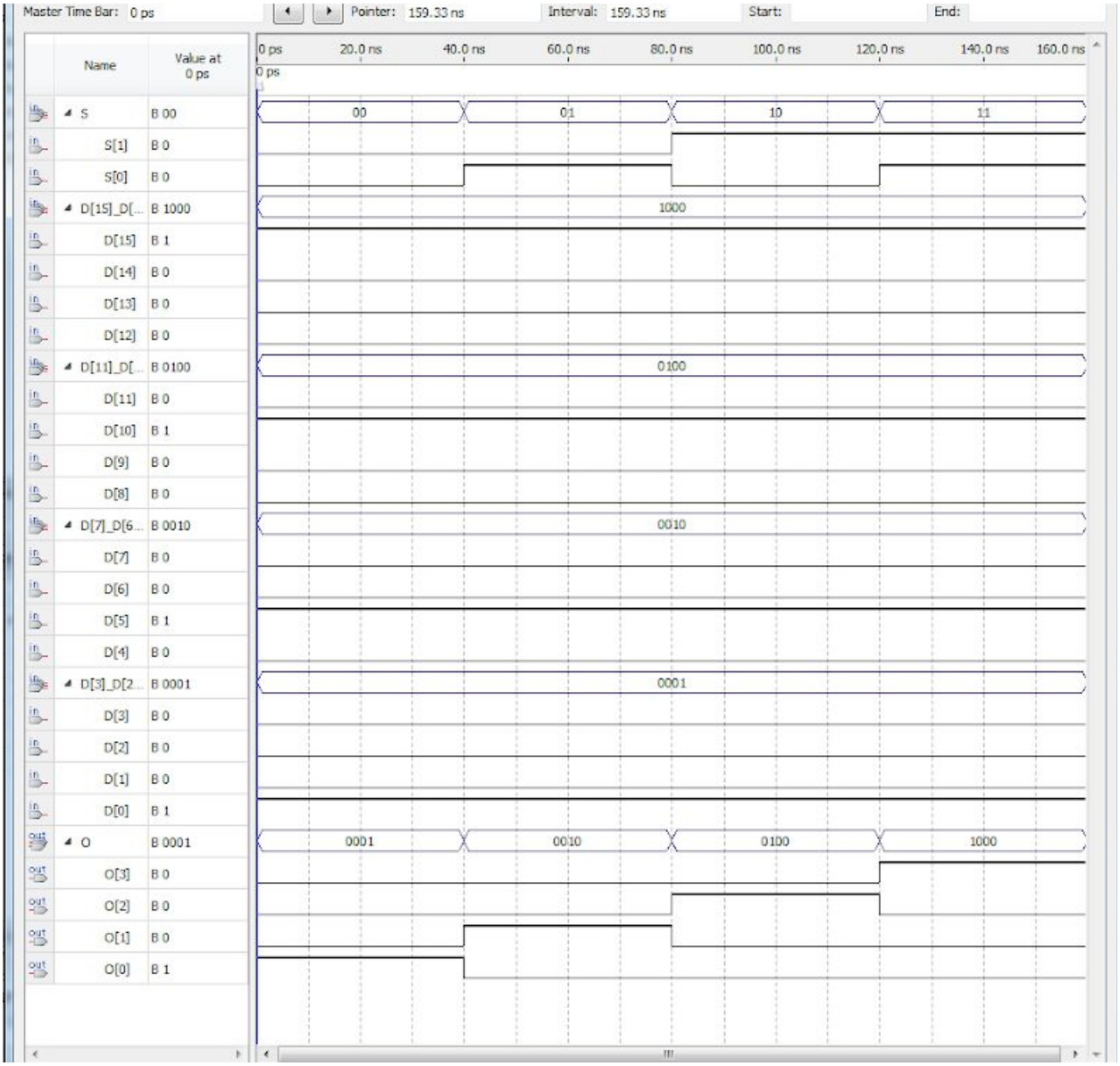


COMPONENTS FOR PART 3:

Verilog Code for Implementation:

```
module Four_to_1_4_bit_mux(O,D,S); //declares module

        input wire [15:0]D; // four data inputs
        input wire [1:0]S; // two selector inputs
        output wire [3:0]O;   //define output data wire

        Four_to_1_1_bit_mux inst0(O[0],D[3:0],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst1(O[1],D[7:4],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst2(O[2],D[11:8],S); //calls the four to 1, 1 bit module from earlier.
        Four_to_1_1_bit_mux inst3(O[3],D[15:12],S); //calls the four to 1, 1 bit module from earlier.
        //using four of the four to 1, 1 bit functions from earlier, we can create a 4 to 1, 4 bits multiplexer.

endmodule //ends module
```

Wave Diagram Simulation Part 3:



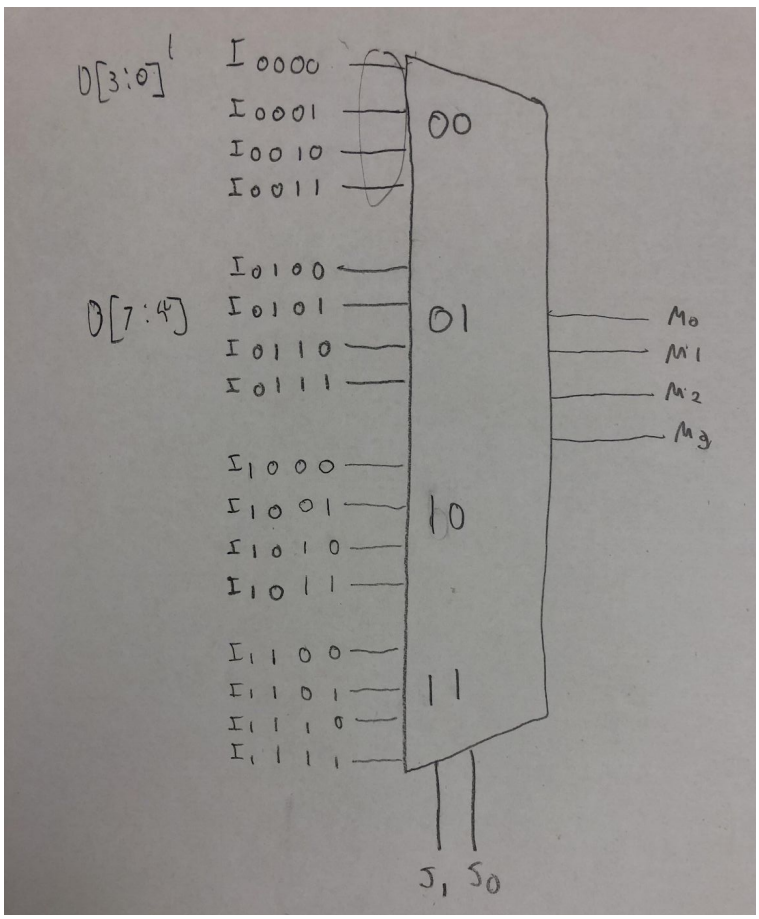
Final Equations for Inputs and Outputs:

$$M_0 = (S_1'S_0'I_{0000}) + (S_1'S_0 I_{0100}) + (S_1S_0' I_{1000}) + (S_1S_0 I_{1100})$$

$$M_1 = (S_1'S_0' I_{0001}) + (S_1'S_0 I_{0101}) + (S_1S_0' I_{1001}) + (S_1S_0 I_{1101})$$

$$M_2 = (S_1'S_0' I_{0010}) + (S_1'S_0 I_{0110}) + (S_1S_0' I_{1010}) + (S_1S_0 I_{1110})$$

$$M_3 = (S_1'S_0' I_{0011}) + (S_1'S_0 I_{0111}) + (S_1S_0' I_{1011}) + (S_1S_0 I_{1111})$$

00	01	10	11

Ancillary Work for Final Equations: