Experiment 6

Random Access Memory

Digital Systems

EEE3342C-20FALL 0011

Spenser Tacinelli
PID: sp498810

Pre-Laboratory Assignment:

3. Given the Function F1(w, x, y, z) and F2(x1, x0, y1, y0), write the truth table for each function. F1(w, x, y, z) - Specified by the lab instructor F2(x1, x0, y1, y0) is a two bit adder. The function F2(x1, x0, y1, y0) has 3 outputs - 2 bits for the sum and 1 bit for the carry out Cout.

w	x	y	x	Out1
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

X1	X0	Y1	Y0	Out1	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1

0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

4. The variables w, x, y and z or x1, x0, y1 and y0 are inputs that are tied to the address lines of the RAM or ROM. Prepare a truth table with inputs A3, A2, A1 and A0 from 0000 to 1111 (16 combinations representing w, x, y and z or x1, x0, y1 and y0)and outputs O0, O1, O2 and O3. O0 represents the output of the function F1(w, x, y, z), O1 and O2 represents the two bits for the sum and O3 represents the carry out Cout bit. The truth table should have the following columns: Each of the input word 0000 to 1111 represents one memory location of the RAM. The output of each word represents the data to be written in the corresponding memory location. Since in this experiment you have 16 input words, you need 4 address lines A3 – A0.

a3	a2	A1	A0	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	1	0	1	0

1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	0	1	1
1	1	0	1	0	1	0	0
1	1	1	0	0	1	0	1
1	1	1	1	1	1	1	0

Objective:

To learn about random access memory and the implementation of this logic design.

Equipment:

The Xilinx's FPGA VIVADO HLx Editions design tools are available in the laboratory. These tools can also be downloaded from Xilinx's web site at www.xilinx.com. The WebPack version of this tool that we use for the laboratory experiments are located under the support download section at the related website (https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloa dNav/vivado-design-tools/2017-4.html). Please take note that the file download size exceeds 1 GB and also during the installation process updates may have to be installed. It can take you up to a few hours to download and install the software on your computer. The user does not need to have the BASYS development board interface to the computer to design and simulate an FPGA.

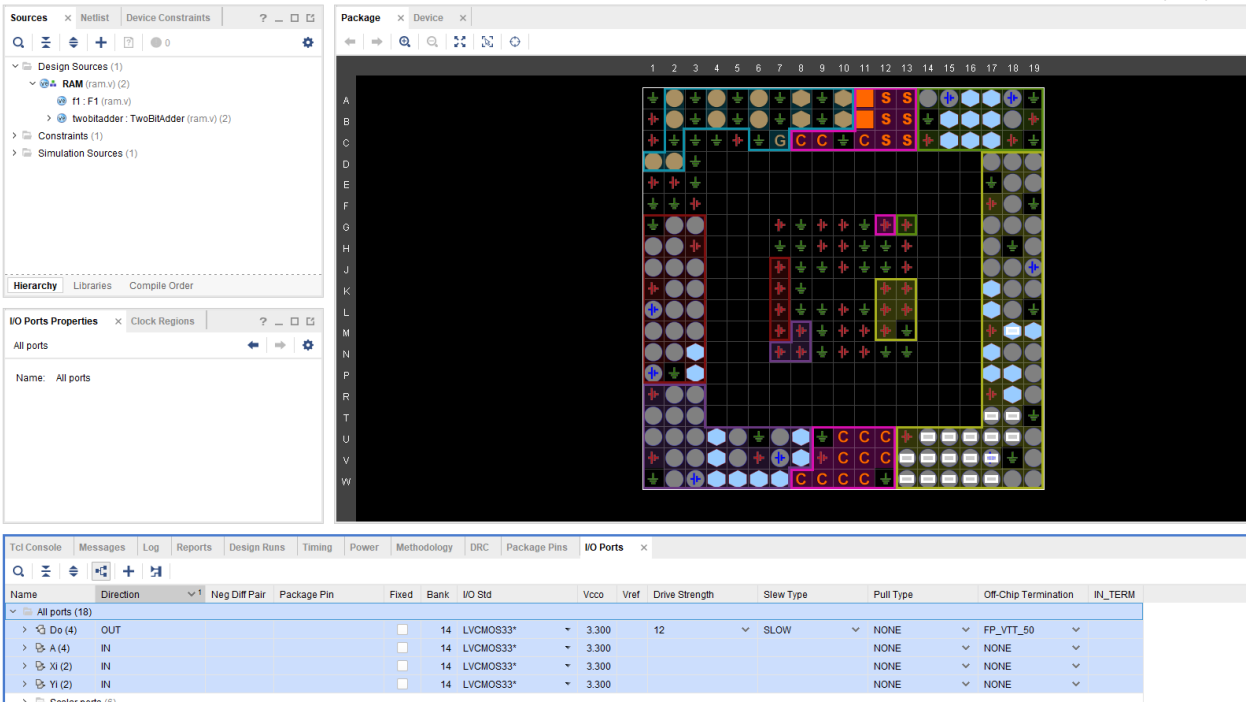
Design Steps:

I created the project (ram_proj) and added the eight inputs and the output register. I then programmed the logic for the half adder, full adder, and two bit adder, since these are all the building blocks of a RAM module.

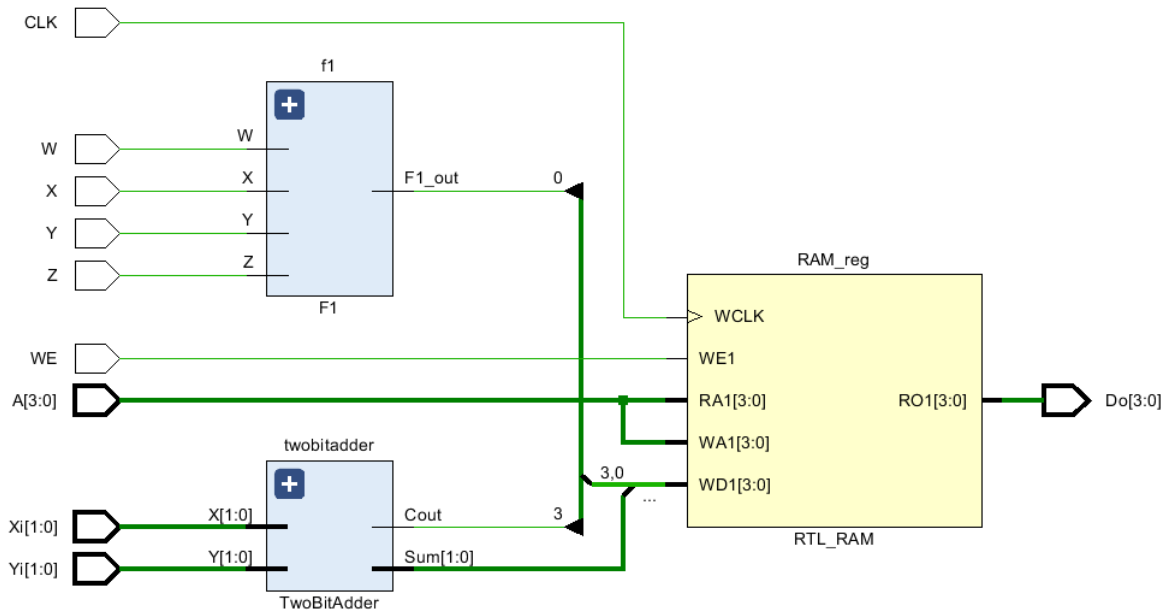
```
module HalfAdder(
    input X,
    input Y,
    output Sum,
    output Carry
    );
    assign Sum = X^Y;
    assign Carry = X&Y;
endmodule

module FullAdder(
    input X,
    input Y,
    input Cin,
    output Cout,
    output Sum
    );
    assign Sum = X^Y^Cin;
    assign Cout = (X&Y) | (Cin & (X+Y));
endmodule

module TwoBitAdder(
    input [1:0]X,
    input [1:0]Y,
    output wire Cout,
    output wire Cin,
    output [1:0] Sum
);

    wire h;
    HalfAdder halfadder(.X(X[0]), .Y(Y[0]), .Sum(Sum[0]), .Carry(h));
    FullAdder fulladder(.X(X[1]), .Y(Y[1]), .Cin(h), .Cout(Cout), .Sum(Sum[1]));
endmodule

module F1(
    input W,
    input X,
    input Y,
    input Z,
    output F1_out
);
    assign F1_out = (W&X&Y&Z) | (W&~X&~Y&Z);
endmodule
```

```
module RAM(
    input CLK,
    input WE,
    input [3:0] A,
    output [3:0] Do,

    input W,
    input X,
    input Y,
    input Z,
    input [1:0] Xi,
    input [1:0] Yi
    );

    wire [3:0] Di;
    reg [3:0] RAM [31:0];

    F1 f1(.W(W), .X(X), .Y(Y), .Z(Z), .F1_out(Di[0]));
    TwoBitAdder twobitadder(.X(Xi[1:0]), .Y(Yi[1:0]), .Sum(Di[2:1]), .Cout(Di[3]));

    always @(posedge CLK) begin
        if(WE)
            RAM[A] <= Di;
        end

    assign Do=RAM[A];
endmodule
```

I then ran synthesis and implementation, to implement I/O Planning and Schematics. I then changed the I/O std to LVCMOS33 in the I/O Planning.



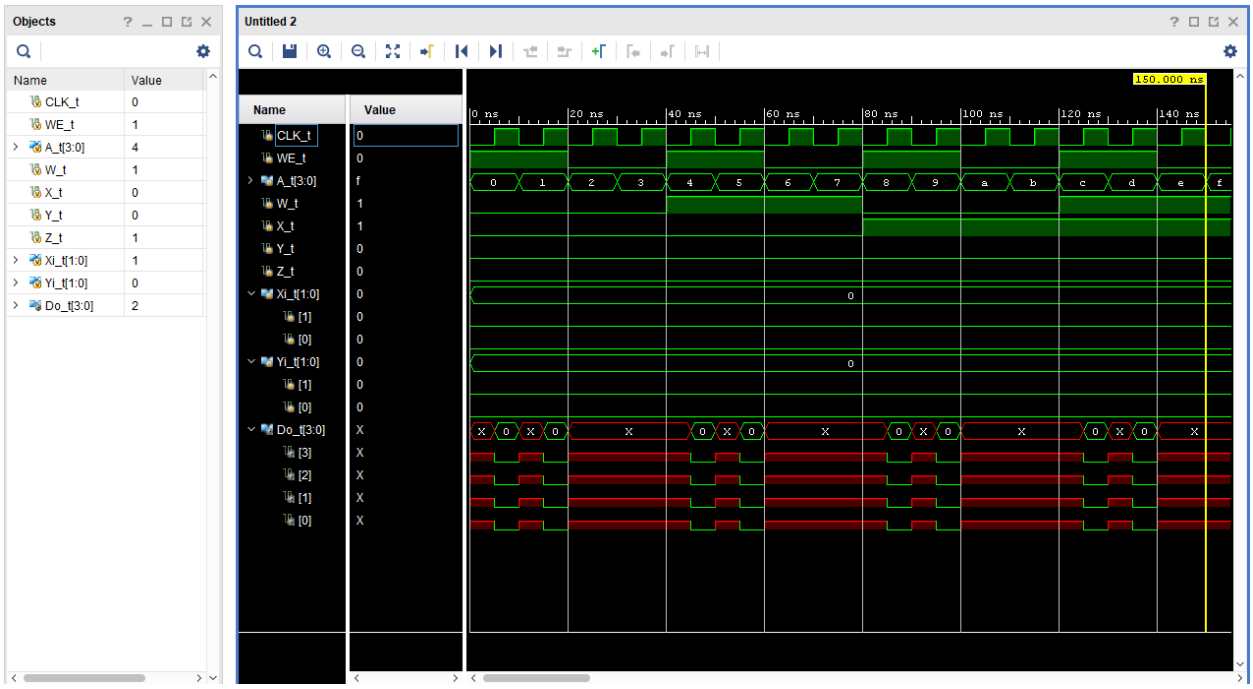
Then I generated the schematic of the RAM circuit.



I then programmed the logic for the simulation.

```
20  //////////////////////////////////
21
22
23  module ram_sim();
24
25  reg CLK_t;
26  reg WE_t;
27  reg [3:0] A_t;
28  reg W_t;
29  reg X_t;
30  reg Y_t;
31  reg Z_t;
32  reg [1:0] Xi_t;
33  reg [1:0] Yi_t;
34  wire [3:0] Do_t;
35
36  RAM UUT (
37      .CLK(CLK_t),
38      .WE(WE_t),
39      .A(A_t),
40      .W(W_t),
41      .X(X_t),
42      .Xi(Xi_t),
43      .Y(Y_t),
44      .Yi(Yi_t),
45      .Z(Z_t),
46      .Do(Do_t)
47
48  );
49
50  initial begin
51      CLK_t = 1'b0;
52      W_t = 1'b0;
53      X_t = 1'b0;
54      Y_t = 1'b0;
55      Z_t = 1'b0;
56      Xi_t = 2'b00;
57      Yi_t = 2'b00;
58      WE_t = 1'b1;
59      A_t = 4'b0000;
60  end
61
62  always #5 CLK_t = ~CLK_t;
63  always #10 A_t = A_t+1;
64  always #20 WE_t = ~WE_t;
65  always #40 W_t = ~W_t;
66  always #80 X_t = ~X_t;
67  always #160 Y_t = ~Y_t;
68  always #320 Z_t = ~Z_t;
69  always #640 Xi_t = Xi_t + 1;
70  always #1280 Yi_t = Yi_t + 1;
71
72  endmodule
73
```

I then ran a behavioral simulation.

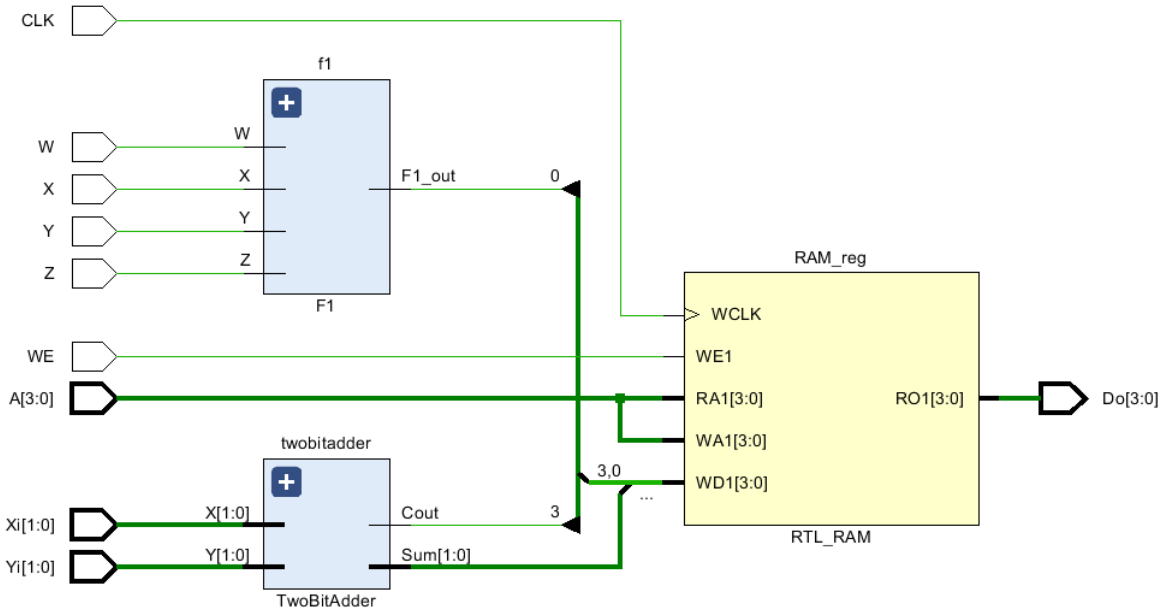


(behavioral)

RAM circuit Schematic

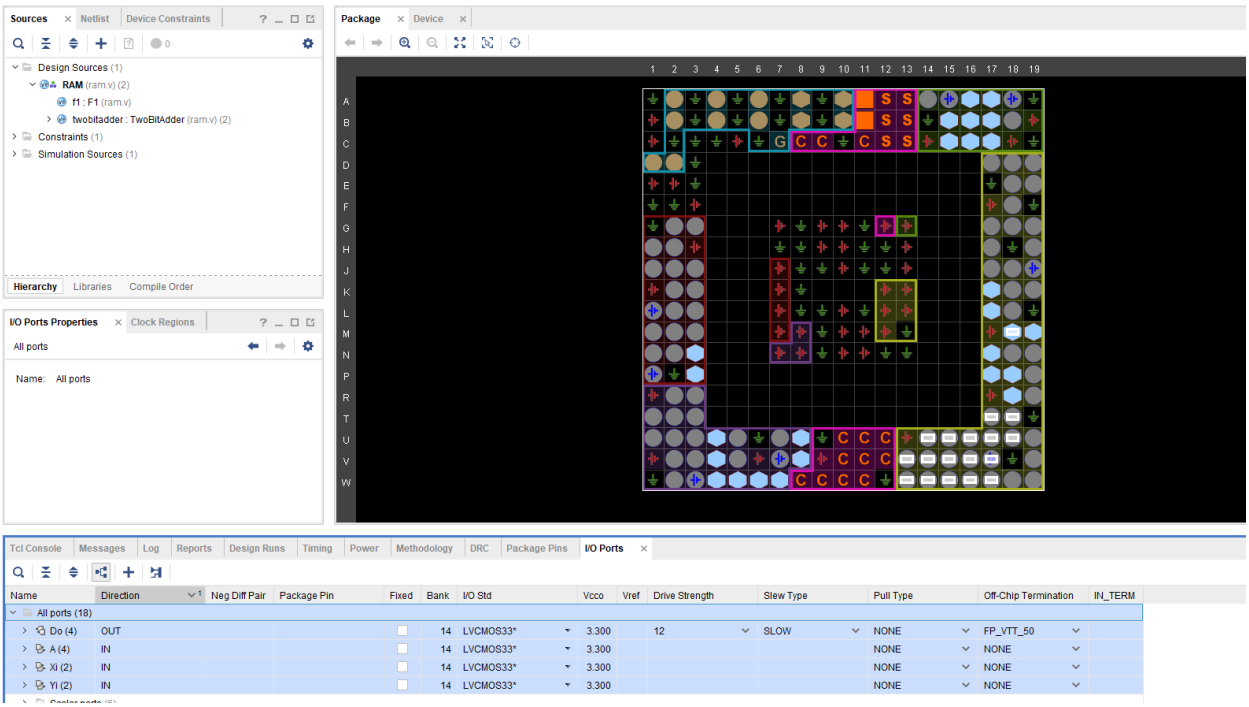
Logic Diagram:

The resulting schematic diagram for the multiplexer.



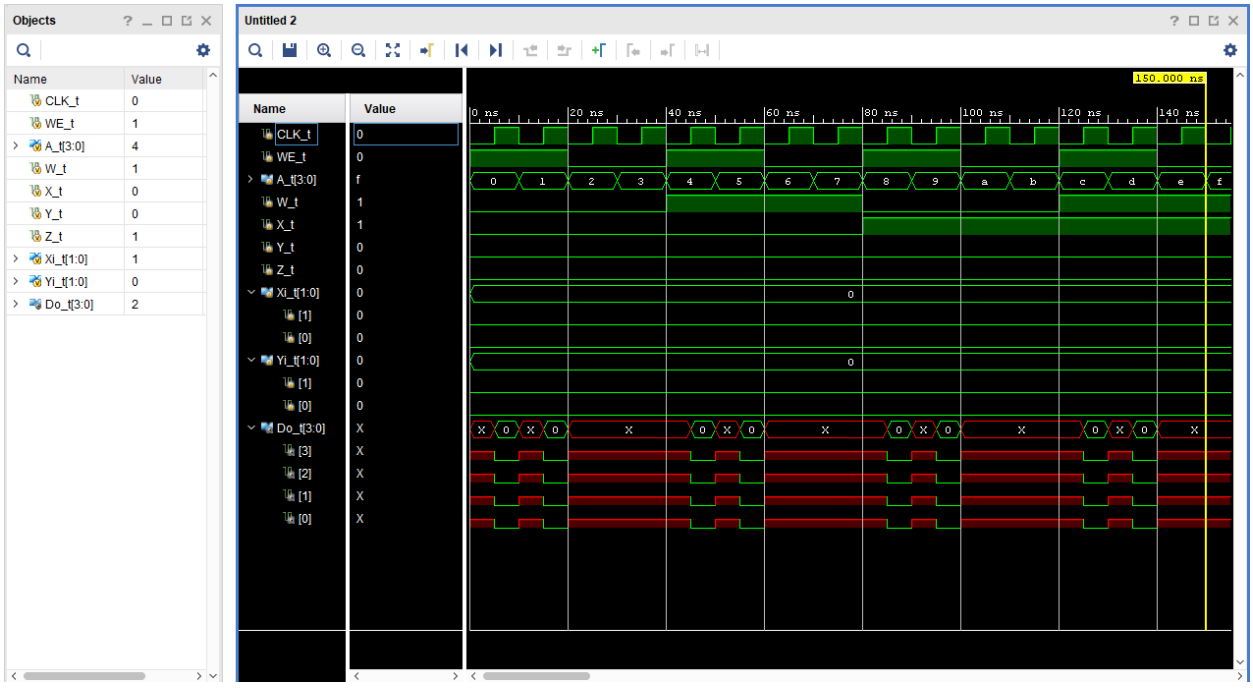
Design Specification Plan:

This is the design specification for a RAM module. It has eight inputs (CLK, WE, [3:0] A, [3:0] Do, W, X, Y, Z, [1:0] Xi, and [1:0] Yi)and one output ([3:0] Do).



Results Statement:

Here is the behavioral simulation results for the RAM module.



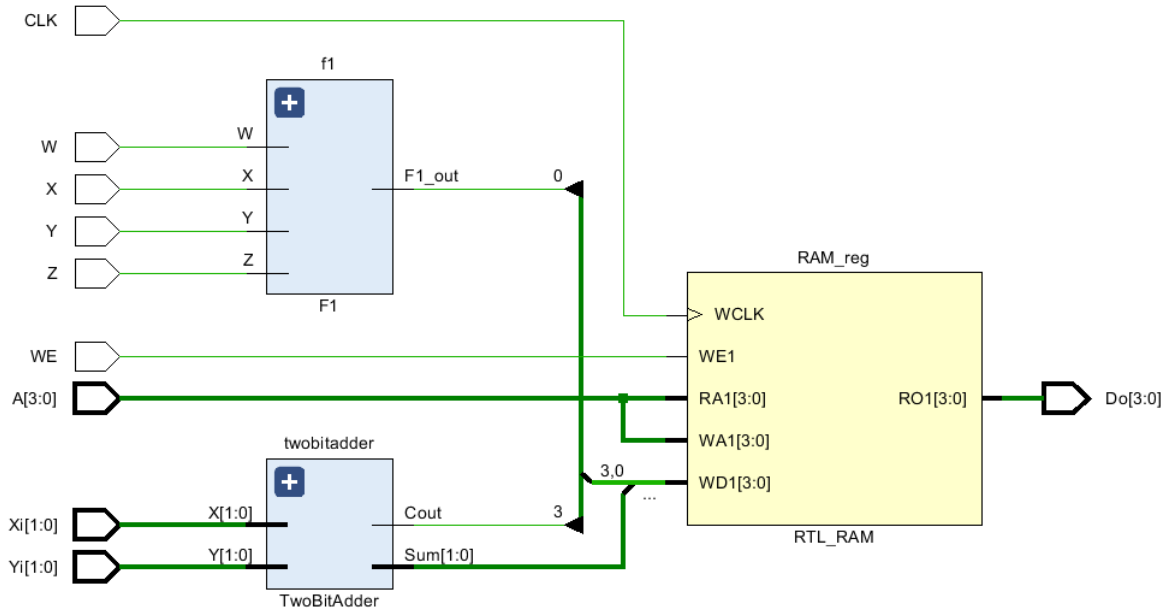
w	x	y	x	Out1	Out2	Out3	Out4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1

0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

RAM circuit Verilog

Logic Diagram

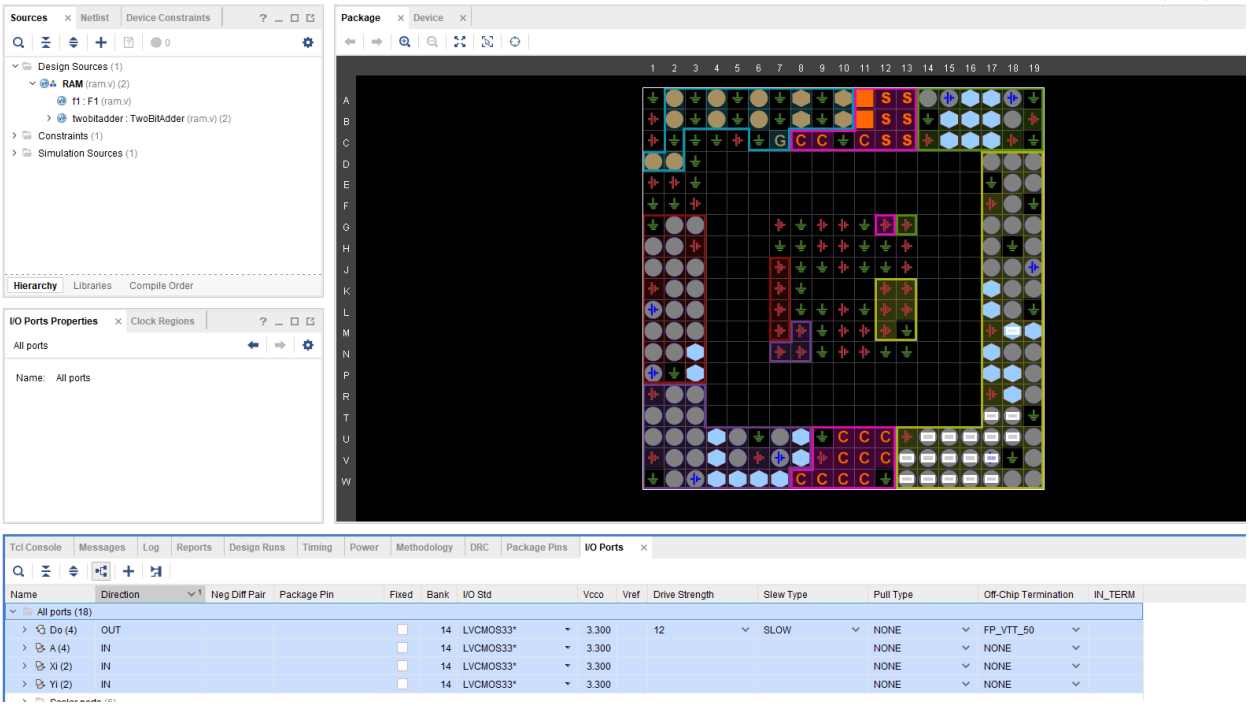
The results from the simulation testing for the simulation for multiplexor circuit.



Design Specification Plan

This is the design specification for a multiplexer. It has eight inputs (CLK, WE, [3:0] A, [3:0] Do, W, X, Y, Z, [1:0] Xi, and [1:0] Yi)and one output ([3:0] Do).

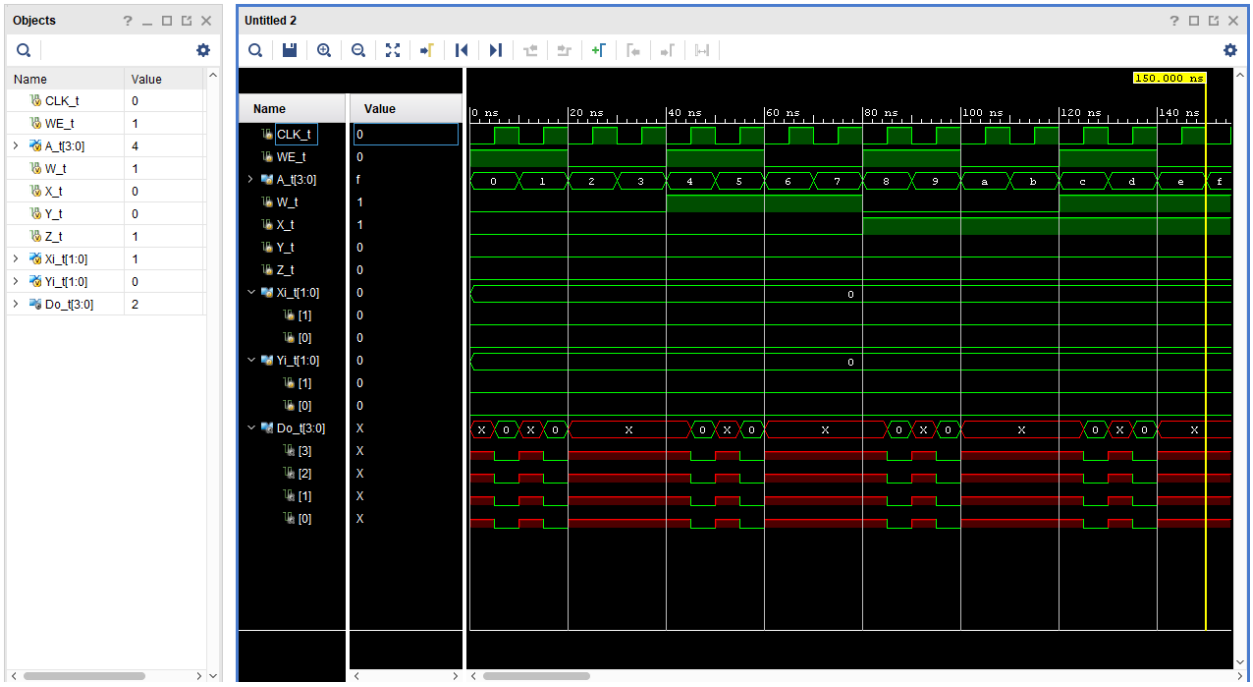
```
20  //////////////////////////////////
21
22
23  module ram_sim();
24
25  reg CLK_t;
26  reg WE_t;
27  reg [3:0] A_t;
28  reg W_t;
29  reg X_t;
30  reg Y_t;
31  reg Z_t;
32  reg [1:0] Xi_t;
33  reg [1:0] Yi_t;
34  wire [3:0] Do_t;
35
36  RAM UUT (
37      .CLK(CLK_t),
38      .WE(WE_t),
39      .A(A_t),
40      .W(W_t),
41      .X(X_t),
42      .Xi(Xi_t),
43      .Y(Y_t),
44      .Yi(Yi_t),
45      .Z(Z_t),
46      .Do(Do_t)
47
48  );
49
50  initial begin
51      CLK_t = 1'b0;
52      W_t = 1'b0;
53      X_t = 1'b0;
54      Y_t = 1'b0;
55      Z_t = 1'b0;
56      Xi_t = 2'b00;
57      Yi_t = 2'b00;
58      WE_t = 1'b1;
59      A_t = 4'b0000;
60  end
61
62  always #5 CLK_t = ~CLK_t;
63  always #10 A_t = A_t+1;
64  always #20 WE_t = ~WE_t;
65  always #40 W_t = ~W_t;
66  always #80 X_t = ~X_t;
67  always #160 Y_t = ~Y_t;
68  always #320 Z_t = ~Z_t;
69  always #640 Xi_t = Xi_t + 1;
70  always #1280 Yi_t = Yi_t + 1;
71
72  endmodule
73
```



Results Statement

Explanation of pictures and tables.

Here is the behavioral simulation results.



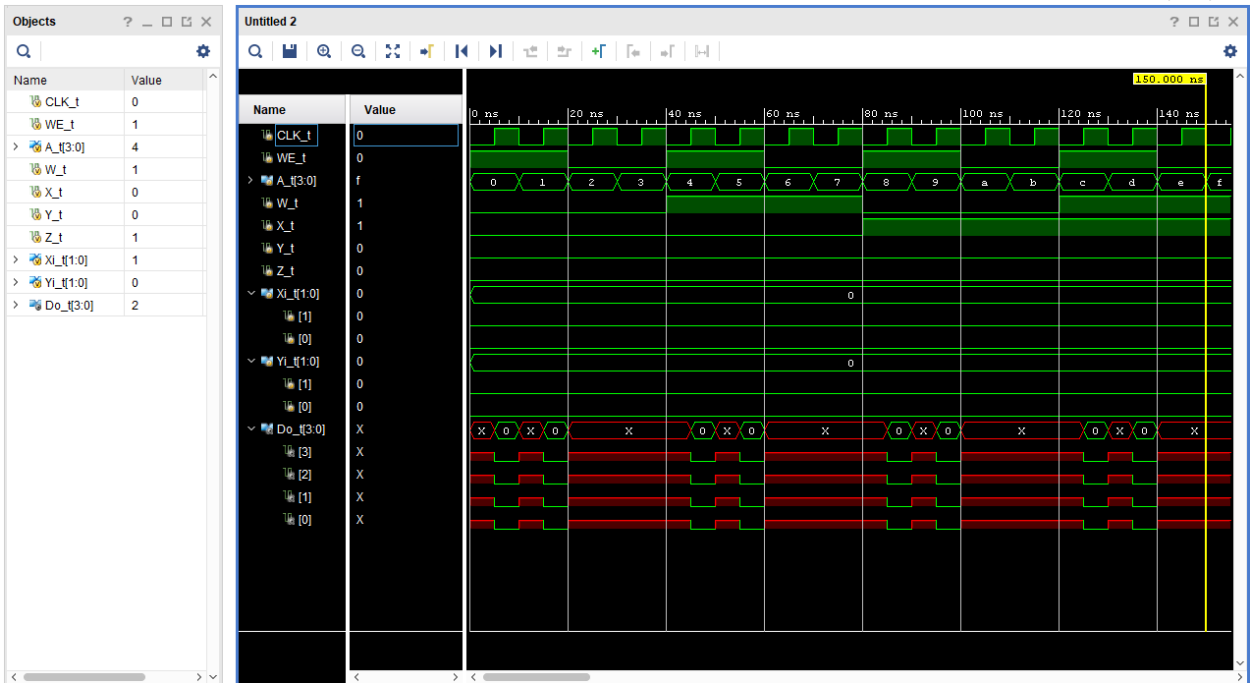
w	x	y	x	Out1	Out2	Out3	Out4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Test Plan

The design was tested through a behavioral simulation that looped though every combination of inputs, which would give every output.

Burglar alarm controller circuit

Here is the behavioral simulation results.



Conclusion

The results came out as I expected, including the simulation. By utilizing the simulation and lookup table to do some research to further understand the RAM module, with some trial and error with the programming logic, it really helped with understanding how RAM works.

1. Discuss the advantages of using RAM for implementing combinational functions.

RAM (random access memory) is a memory module that can be used for combinational logic. Its primary use to store data that does not need to be stored long term (mathematical calculations for example). This means that the RAM circuit is a volatile module, it must be ON to function, and it will purge all data when it is turned off. This makes it quite efficient.

2. Which one is more practical, read/write memory or read only memory? Explain.

Read and write. Without the writing capability, information/data would never be able to be stored.

3. Some memory devices have only one set of data lines for both input and output. What is the advantage and disadvantage of having the input and output lines separately or shared?

The primary advantage of this the separation of the memory address and the input/output. The primary disadvantage is the due to the fact that a certain i/o set is required.

4. Have you met all the requirements of this lab (Design Specification Plan)?

yes

5. How should your design be tested (Test Plan)?

With a behavioral simulation to test all input and output combinations.

6. What is the advantage of using RAM or ROM memory to implement combinational logic?

Since random access memory is volatile, it allows for the stored information to be changed while it is working.

7. How can two 32x4s devices be combined together to form one 64 by 4 bit memory system?

You take the two 32x4 devices, connect them with a data out, and then connect their eight outputs to four OR gates, which will have their own output, giving four outputs.