Random Access Memory

Digital Systems

Beatriz Navas Aguilera

Random Access Memory

Beatriz Navas
UCFID: 5435667

Beatriz Navas

9/23/24

Equipment

In this experiment, a computer utilizing Xilinx's FPGA VIVADO HILx Edition software, along with a BASYS 3 development board, was used. The software, available at www.xilinx.com was used to program the BASYS 3 board and ultimately test the functionality of the various logic gates implemented in the experiment.

Apparatus List

- BASYS 3
- VERILOG

Objective

The objective of this experiment is to investigate how RAM (Random Access Memory) and ROM (Read-Only Memory) can be utilized as practical implementations for combinational logic circuits. The experiment will focus on configuring memory elements to store and retrieve specific data patterns that correspond to desired logic outputs, allowing for the realization of complex combinational functions. By understanding how to map logic functions into memory arrays, this approach demonstrates an alternative method for achieving combinational circuit design, emphasizing the flexibility, scalability, and efficiency of using memory devices over traditional gate-based implementations.

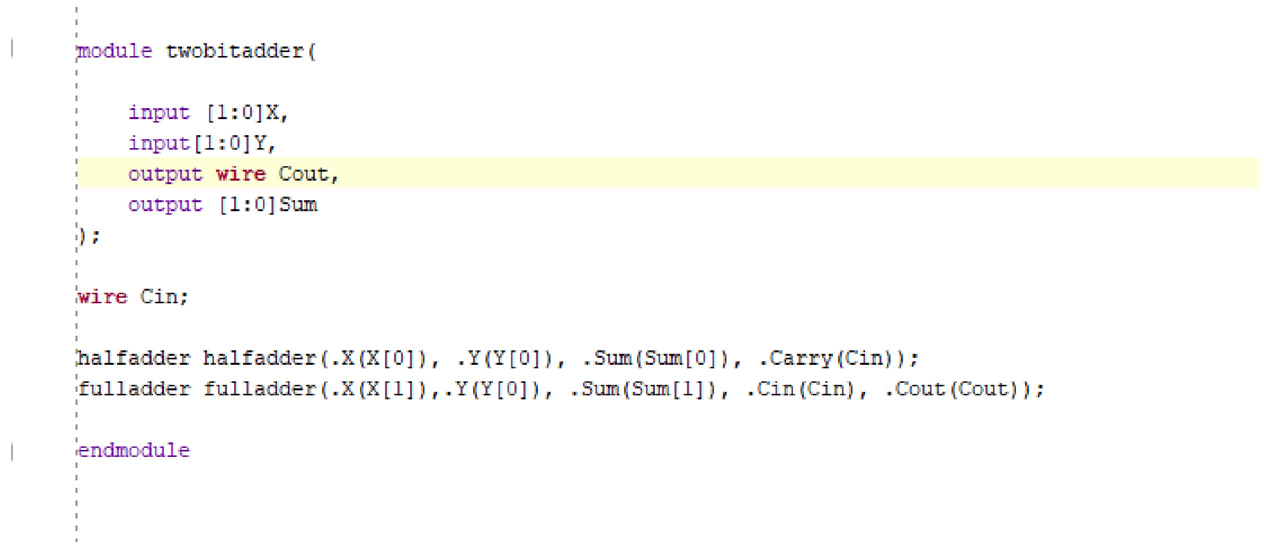
Design Plan & Procedure

This RAM module is configured to store 16 words, each 4 bits wide, with input data provided through data lines (D0–D3) and addresses specified through address lines (A0–A3). The design includes an active high Write Enable (WE) signal, which allows data to be written to a specific memory address when activated. In read mode (when WE is low), the stored data at the addressed location will be output on the lines O0–O3.

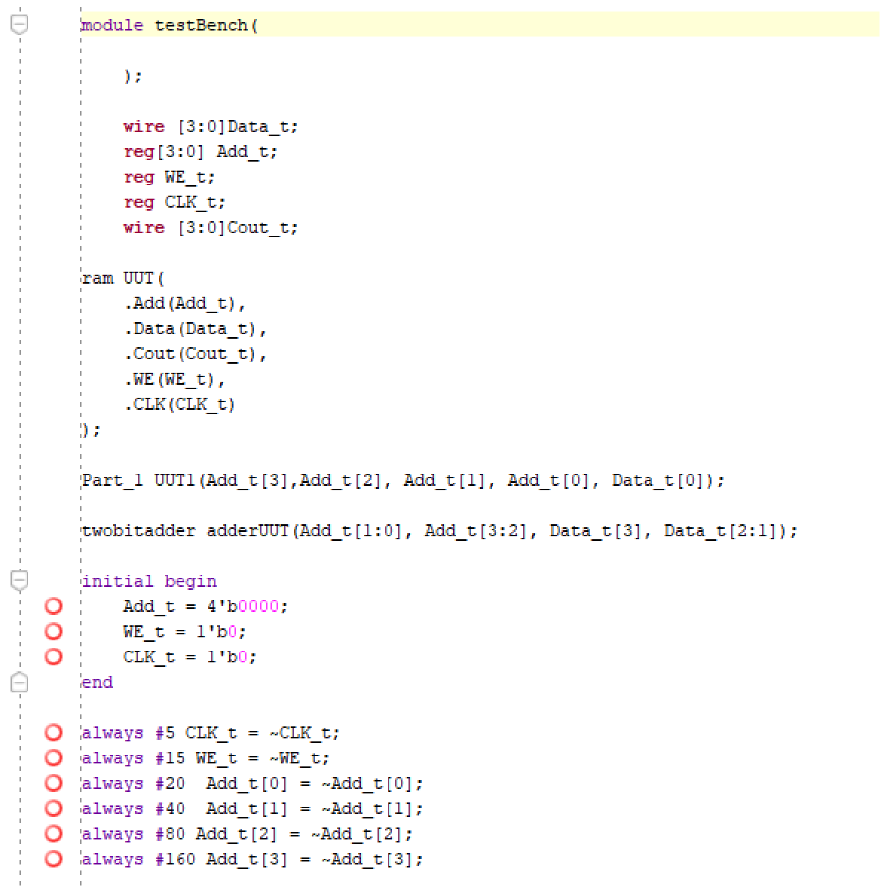
A3	A2	A1	A0	O3	O2	O1	O0

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

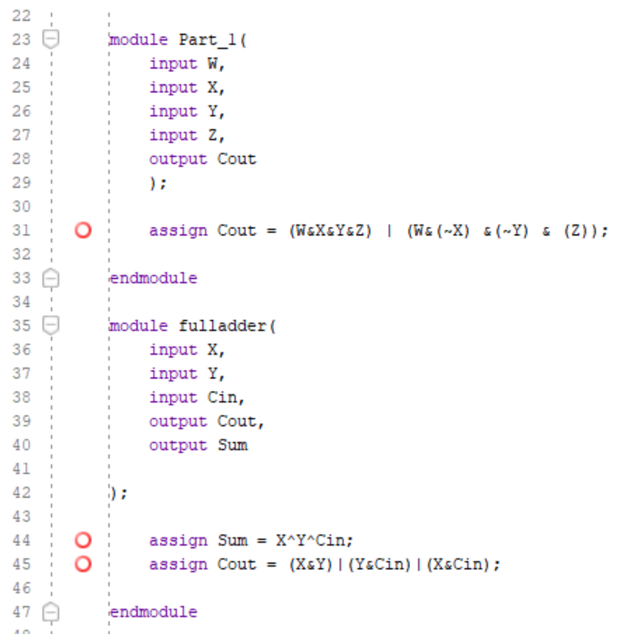
To demonstrate the functionality of the RAM module in performing combinational logic, two functions—F1 and F2—will be written into RAM. The first function, F1, uses three inputs (x, y, z) to generate an output pattern based on the lab-specified truth table. The second function, F2, is a two-bit adder with three outputs, two representing sum bits and one representing the carry-out bit. Data corresponding to the truth tables of these functions will be loaded into the RAM, allowing us to verify that each function is correctly implemented and accessible via address selection. The code was shown as below:



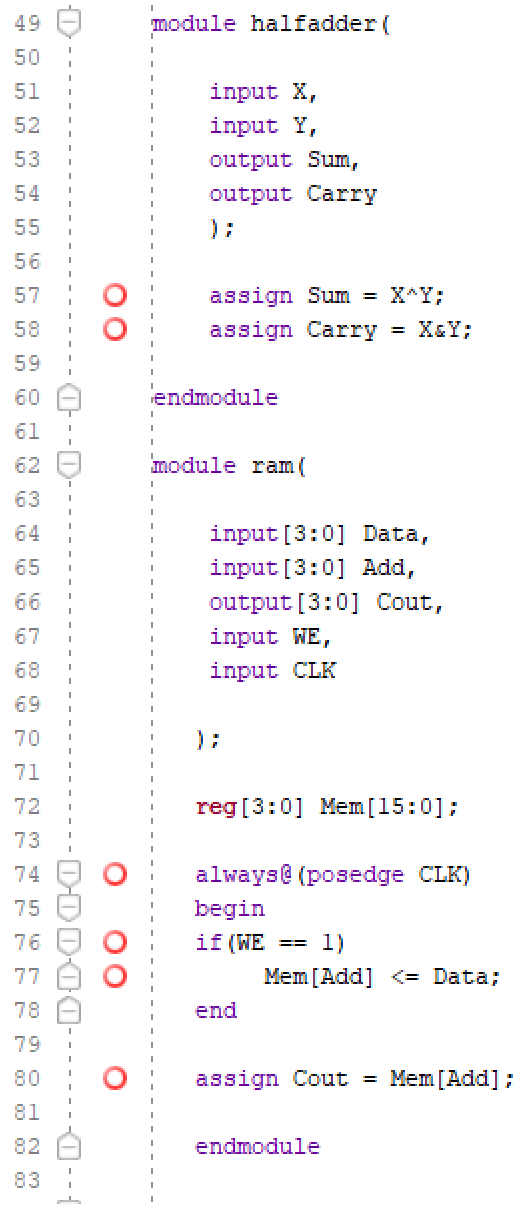
```
module twobitadder(

    input [1:0]X,
    input[1:0]Y,
    output wire Cout,
    output [1:0]Sum
);

wire Cin;

halfadder halfadder(.X(X[0]), .Y(Y[0]), .Sum(Sum[0]), .Carry(Cin));
fulladder fulladder(.X(X[1]),.Y(Y[0]), .Sum(Sum[1]), .Cin(Cin), .Cout(Cout));

endmodule
```



```
module testBench(

    );

    wire [3:0]Data_t;
    reg[3:0] Add_t;
    reg WE_t;
    reg CLK_t;
    wire [3:0]Cout_t;

ram UUT(
    .Add(Add_t),
    .Data(Data_t),
    .Cout(Cout_t),
    .WE(WE_t),
    .CLK(CLK_t)
);

Part_1 UUT1(Add_t[3],Add_t[2], Add_t[1], Add_t[0], Data_t[0]);

twobitadder adderUUT(Add_t[1:0], Add_t[3:2], Data_t[3], Data_t[2:1]);

initial begin
    Add_t = 4'b0000;
    WE_t = 1'b0;
    CLK_t = 1'b0;
end

always #5 CLK_t = ~CLK_t;
always #15 WE_t = ~WE_t;
always #20  Add_t[0] = ~Add_t[0];
always #40  Add_t[1] = ~Add_t[1];
always #80 Add_t[2] = ~Add_t[2];
always #160 Add_t[3] = ~Add_t[3];
```



```

module Part_1(
    input W,
    input X,
    input Y,
    input Z,
    output Cout
    );

    assign Cout = (W&X&Y&Z)  |  (W&(~X)  &(~Y)  &  (Z));

endmodule

module fulladder(
    input X,
    input Y,
    input Cin,
    output Cout,
    output Sum

);

    assign Sum = X^Y^Cin;
    assign Cout = (X&Y)|(Y&Cin)|(X&Cin);

endmodule
```

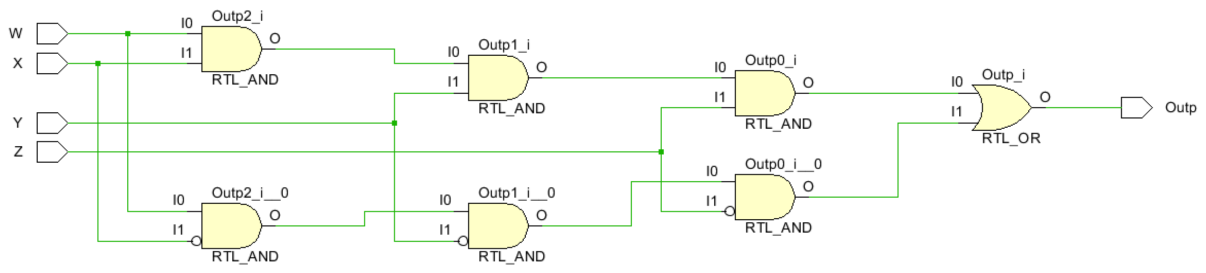


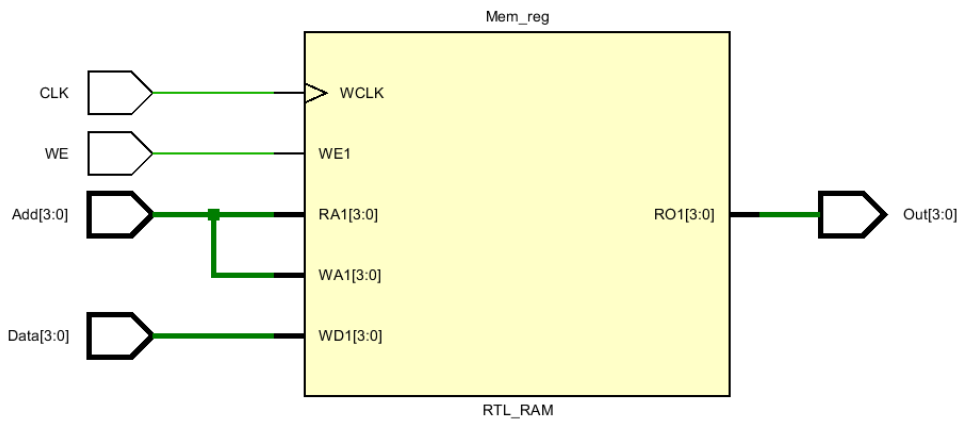
```
module halfadder(

    input X,
    input Y,
    output Sum,
    output Carry
    );

    assign Sum = X^Y;
    assign Carry = X&Y;

endmodule

module ram(

    input[3:0] Data,
    input[3:0] Add,
    output[3:0] Cout,
    input WE,
    input CLK

   );

   reg[3:0] Mem[15:0];

   always@(posedge CLK)
   begin
   if(WE == 1)
        Mem[Add] <= Data;
   end

   assign Cout = Mem[Add];

   endmodule

```

Testing the design involves loading data into the RAM by setting the WE signal high, specifying the address with A0–A3, and inputting data on D0–D3. After data is loaded, the RAM will be switched to read mode by setting WE low. Front 0000 to 1111, will be accessed to verify that the output matches the expected truth table for each function. The output values will be displayed via LEDs on the Basys 3 board, allowing a visual check against the defined truth tables for F1 and F2. We also performed RLT analysis as well as made a schematic.

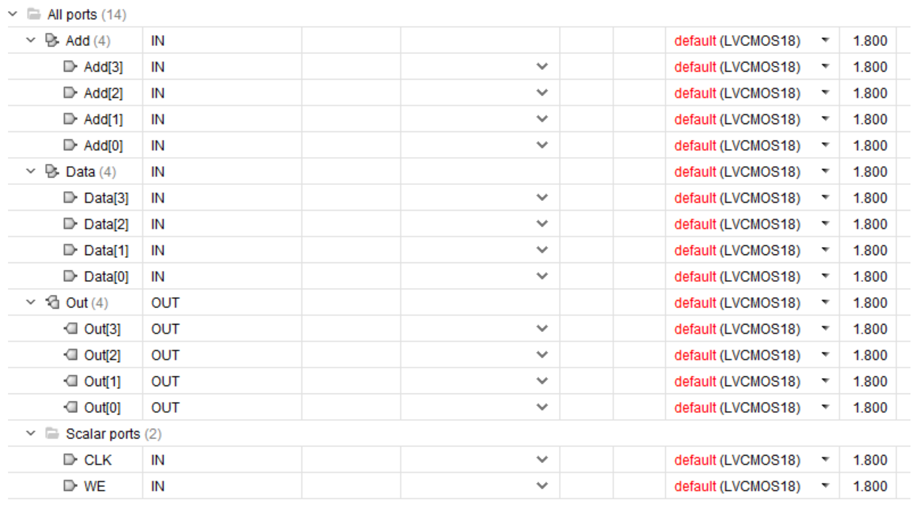
Beatriz Navas

9/23/24





Finally, I programmed the input and outputs for the board as follows:



All ports (14)			
Add (4)	IN	default (LVCMOS18)	1.800
Add[3]	IN	default (LVCMOS18)	1.800
Add[2]	IN	default (LVCMOS18)	1.800
Add[1]	IN	default (LVCMOS18)	1.800
Add[0]	IN	default (LVCMOS18)	1.800
Data (4)	IN	default (LVCMOS18)	1.800
Data[3]	IN	default (LVCMOS18)	1.800
Data[2]	IN	default (LVCMOS18)	1.800
Data[1]	IN	default (LVCMOS18)	1.800
Data[0]	IN	default (LVCMOS18)	1.800
Out (4)	OUT	default (LVCMOS18)	1.800
Out[3]	OUT	default (LVCMOS18)	1.800
Out[2]	OUT	default (LVCMOS18)	1.800
Out[1]	OUT	default (LVCMOS18)	1.800
Out[0]	OUT	default (LVCMOS18)	1.800
Scalar ports (2)			
CLK	IN	default (LVCMOS18)	1.800
WE	IN	default (LVCMOS18)	1.800

Beatriz Navas

9/23/24

Design Specification Plan

The design approach for the RAM experiment was structured to meet the functional requirements of implementing and verifying a 16-word by 4-bit RAM module, configured to perform as a combinational logic circuit on the Basys 3 FPGA board. The key components of the design are as follows:

Truth Table Generation

The truth tables for the functions to be stored in RAM, specifically with F1(x,y,z), a single output is generated for every unique combination of the 3 input bits. Which functions as a 2-bit adder, three outputs are generated: two bits representing the sum and one representing the carry-out. Each row in the truth table corresponds to a specific address in RAM, allowing us to map input combinations to output data directly.

RAM Configuration

The RAM was configured to store 16 memory words, each 4 bits wide, to cover all possible combinations of inputs for the functions. The address lines (A3, A2, A1, A0) were mapped to the input combinations for the two functions. The output lines (O3, O2, O1, O0) were then configured to hold the results for each function based on the address selected.

Verilog Implementation of RAM

The RAM module was implemented in Verilog with 4 address lines (A3–A0), 4 data input lines (D0–D3), and 4 output lines (O3–O0). A Write Enable (WE) signal controls whether the RAM is in read or write mode. In write mode (WE high), data is written to the specified address in RAM, while in read mode (WE low), the data stored at the selected address is output on O3–O0. This Verilog design was synthesized and implemented on the Basys 3 FPGA board to allow testing of both functions by setting specific input values and observing the outputs.

Testing and Verification

The RAM functionality was verified by first writing data into each address location according to the truth tables for F1 and F2. The RAM was then placed in read mode, and each address from 0000 to 1111 was accessed to confirm that the stored outputs matched the expected results.

The outputs were displayed using LEDs on the Basys 3 board, providing a visual verification of the correctness of each function. Additionally, test plans were established to validate the system's response to different input combinations and ensure that each address accurately reflected the intended output values as per the truth tables.

Modular Design and Scalability

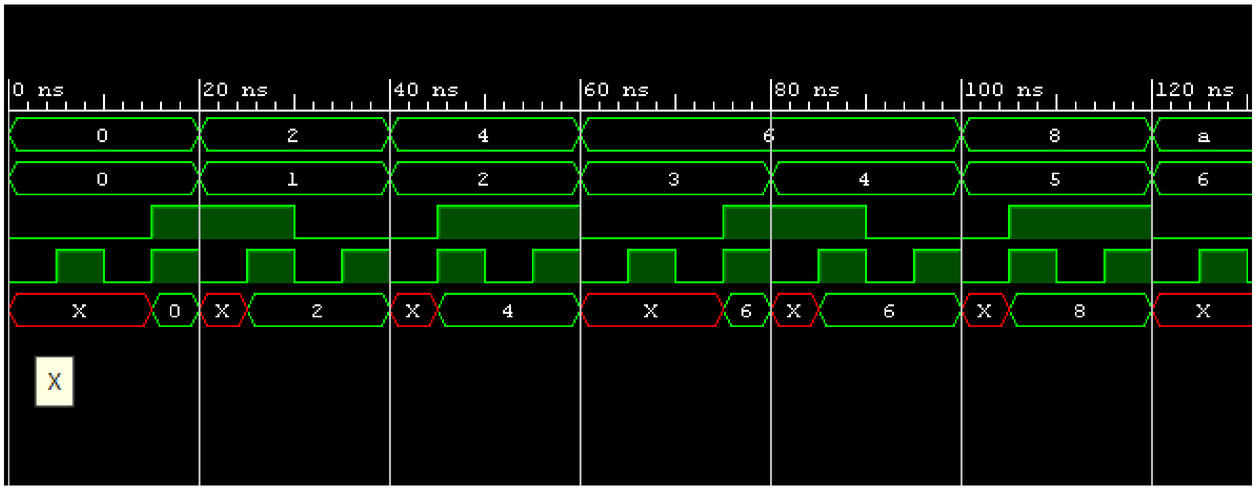
This RAM-based design provides a flexible and modular approach to implementing combinational logic functions, allowing easy modification of stored functions by simply adjusting the data stored at each address. The modular structure of the Verilog code enables potential scalability, allowing for more complex functions or larger address spaces to be implemented if needed.

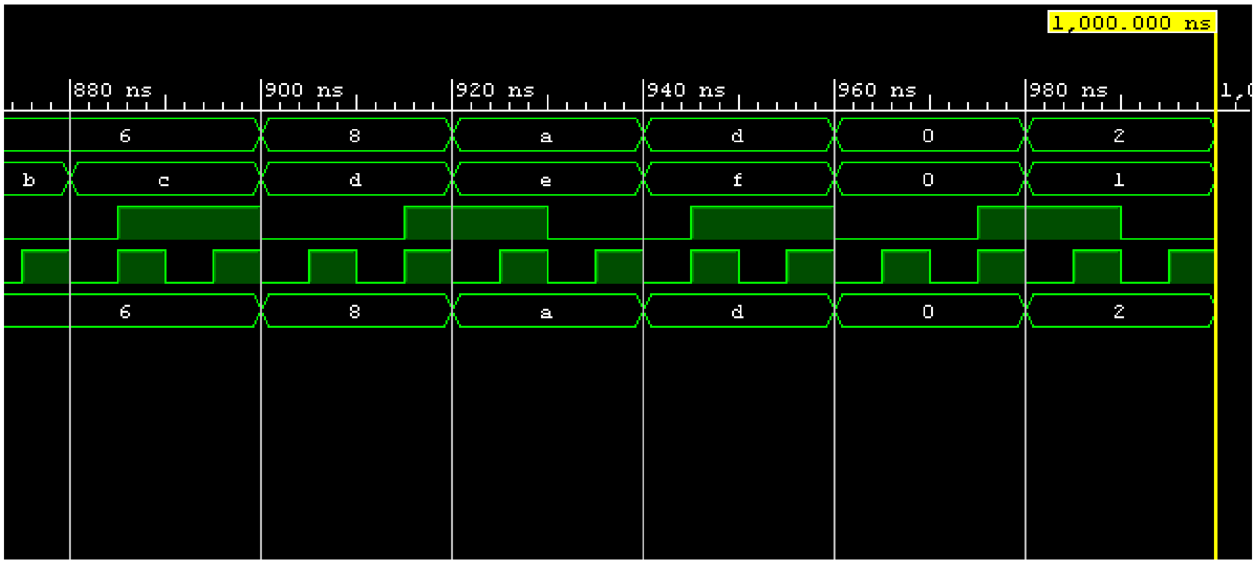
Test Plan

The test plan for the 16-word by 4-bit RAM module on the Basys 3 FPGA board aims to verify that the module correctly stores and retrieves data as defined by the truth tables for F1(x,y,z) and F2(x1,x0,y1,y0). First, the setup involves programming the FPGA with the RAM design, connecting input switches to address and data lines, and using LEDs for output verification. The Write Test checks data storage by setting WE high, entering each address (0000 to 1111), and confirming the output matches the expected values for F1 and F2 according to the truth tables. The Read Test, conducted with WE low, validates data retrieval by observing that each address outputs the correct stored data. Additionally, the WE Signal Test ensures that data can only be written when WE is high, verifying that data is stable on the output lines when WE is low, even if data inputs change. Together, these tests confirm that the RAM module functions accurately in storing and outputting combinational logic data as specified.

Results Statement

The results of the RAM experiment demonstrate that the 16-word by 4-bit RAM module implemented on the Basys 3 FPGA board successfully stores and retrieves combinational logic data as defined by the truth tables for F1(x,y,z) and F2(x1,x0,y1,y0). During the Write Test, each address was written with data corresponding to the truth table values while WE was set high. Observations showed that the output matched the expected values for each address, confirming accurate data storage. In the Read Test, with WE set low, retrieving data from each address produced outputs on the LEDs that aligned with the expected truth table results, validating correct data retrieval. The WE Signal Test verified that data could only be written when WE was high and that output data remained stable when WE was low, even if input values changed. Overall, the experiment confirmed that the RAM module functioned as designed, accurately implementing the combinational logic functions by storing and outputting the correct data for each input combination.





Conclusion

In conclusion, the RAM experiment successfully demonstrated the use of a 16-word by 4-bit RAM module to implement combinational logic functions on the Basys 3 FPGA board. By configuring the RAM to store specific outputs for each input combination, we effectively used memory as a means to realize logic functions, specifically for F1(x,y,z) and F2(x1,x0,y1,y0). The Write Test confirmed that data could be accurately stored in each address, while the Read Test verified reliable retrieval, with outputs matching the predefined truth tables. Additionally, the WE Signal Test showed that the module behaved correctly, only allowing data writing when WE was high and ensuring stable output in read mode. This approach highlights the flexibility of using RAM for combinational logic, offering a scalable alternative to traditional gate-based implementations. Overall, the experiment achieved its objectives, validating RAM as a viable option for implementing complex logic functions in digital systems.