A Project Report on,

MEMORY DESIGN OF RAM IN ARTIX-7 FPGA KIT

SUBMITTED BY,

Ganga Mythra H [BU22EECE0100469]
Laasya priya R [BU22EECE0100154]
M Lokvignesh [BU21EECE0100099]
Maria Punya [BU21EECE0100309]
Meghana B [BU21EECE0100559]

UNDER GUIDANCE OF,

Dr. Arun Kumar Manoharan
Associate Professor , Department of EECE , GST,
GITAM DEEMED TO BE UNIVERSITY, Bengaluru

Dr. Karthick S
Associate Professor , Department of EECE , GST,
GITAM DEEMED TO BE UNIVERSITY, Bengaluru

GITAM DEEMED TO BE UNIVERSITY, BENGALURU

Department of Electronics & Communication Engineering,

Summer Development Program-2024

INTRODUCTION OF THIS PROJECT:

RAM allows your computer to perform many everyday tasks, such as loading applications, browsing the internet, editing spreadsheets, or playing games. It also enables quick task switching, retaining the state of each task. RAM, or random-access memory, is a type of computer memory that can be read and written in any order, typically used for storing working data and machine code. Unlike other storage media like hard disks, CD-RWs, DVD-RWs, and older magnetic tapes, RAM allows data to be read or written almost instantaneously, regardless of its physical location, due to the absence of mechanical limitations.

RAM contains multiplexing and demultiplexing circuitry, which connects data lines to the addressed storage for reading or writing. Typically, multiple bits of storage are accessed by the same address, and RAM devices are often categorized by their data line width, such as "8-bit" or "16-bit" devices. This architecture enables efficient and fast data access, making RAM crucial for the smooth operation and performance of computing systems.

In this project, we designed a 16x8 RAM module on the Digilent Nexys 4 DDR Artix-7 FPGA board to provide quick and controllable temporary data storage. Efficient memory design is crucial for enhancing performance in DSP, specialized computing hardware, embedded systems, and educational applications. The RAM module features 16 words, each 8 bits wide, with minimal latency and a read/write time of one clock cycle. Using Verilog for its clarity and compatibility, we ensured robust simulation and verification. Implementing RAM on FPGA offers consistent, sub-millisecond response times, leveraging the FPGA's reconfigurability and programmable logic blocks for efficient memory access.

OVERVIEW OF THIS PROJECT:

• To design and implement a 16x8 RAM module on the Digilent Nexys 4 DDR Artix-7 FPGA board, fulfilling the requirement for rapid and manageable temporary data storage in various digital applications.

• To enhance the capabilities of custom computing hardware and educational tools by integrating an efficient RAM module that supports quick data access and manipulation.

• To utilize FPGA technology to create a synchronous single-port RAM module, addressing the need for streamlined and effective memory solutions in specialized computing applications.

METHODOLOGY:

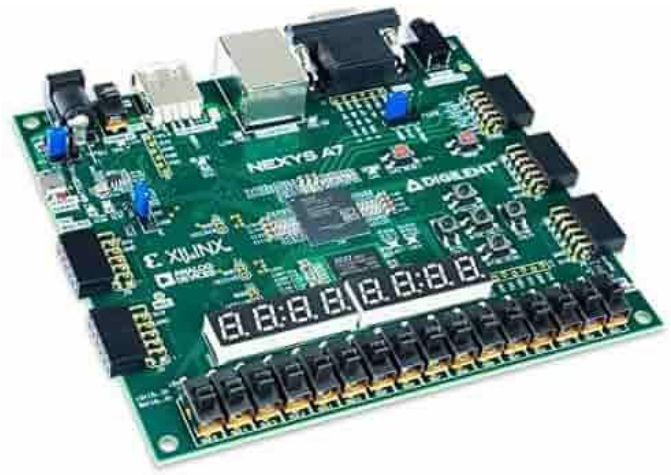
SOFTWARE USED IN THIS PROJECT:

• **Xilinx Vivado Software:** Vivado, developed by Xilinx, is an advanced software suite tailored for FPGA and SoC design. It supports design entry via schematics, Verilog, VHDL, and high-level synthesis, optimizing designs through efficient place-and-route. Vivado enables bitstream programming, provides robust debugging tools, supports IP integration, and facilitates hardware validation through simulation**.**

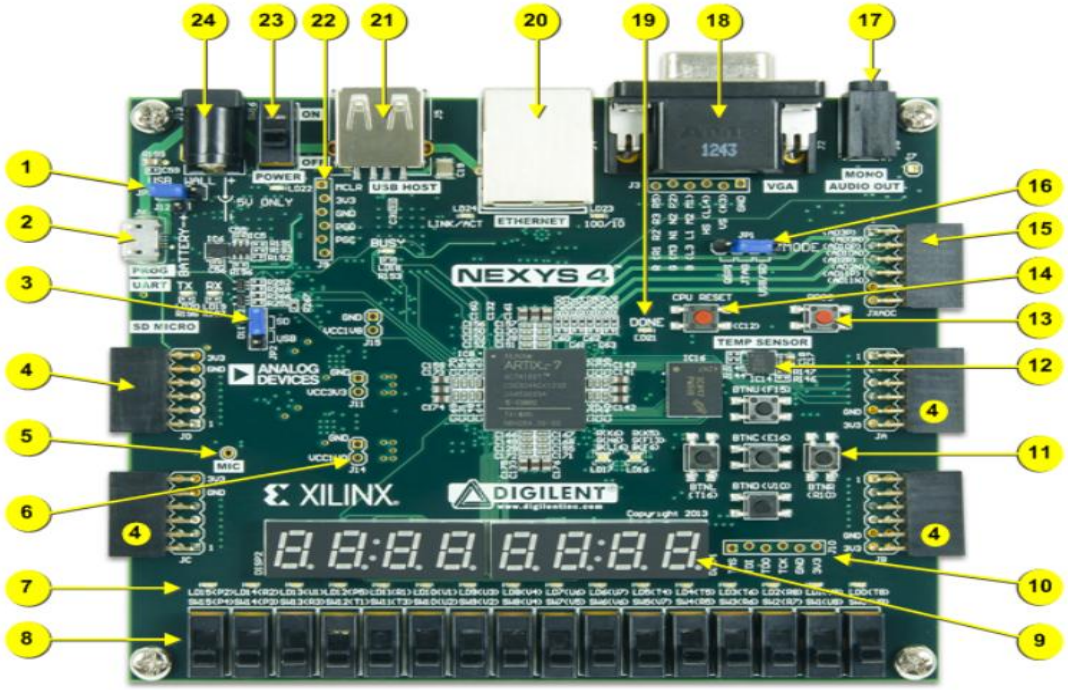
HARDWARE USED IN THIS PROJECT:

• **Digilent Nexys 4 DDR Artix-7 FPGA board:**

The Nexys 4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys 4 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and a lot of I/O devices allow the Nexys 4 to be used for a wide range of designs without needing any other components.



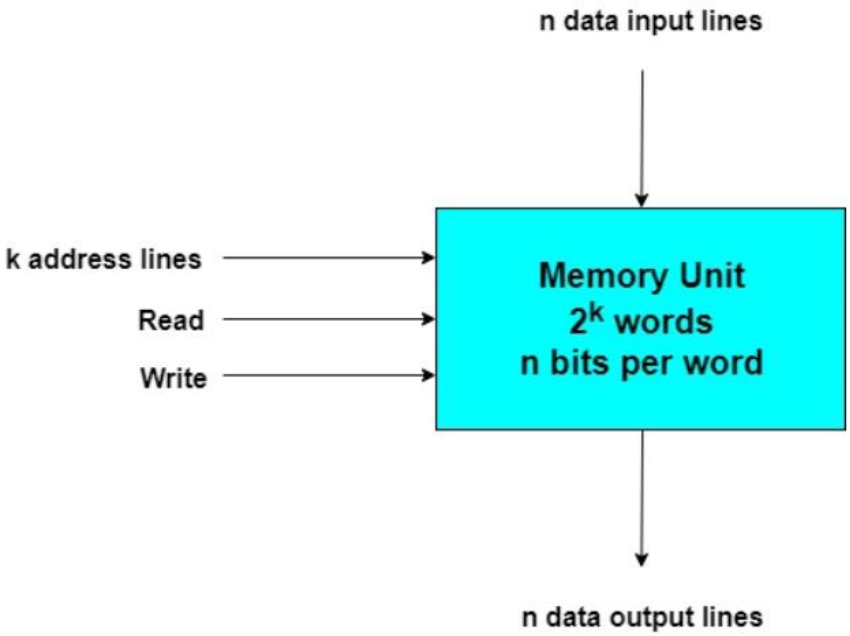
FPGA KIT BOARD FEATURES:



Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

BLOCK DIAGRAM:

A block diagram of a RAM unit is shown below:



Textual description of 16X8 RAM:

• **Address Lines (4 bits):** Input to the RAM block to select one of the 16 memory locations.

• **Data Input Lines (8 bits):** Input lines where data is provided to be written into the RAM.

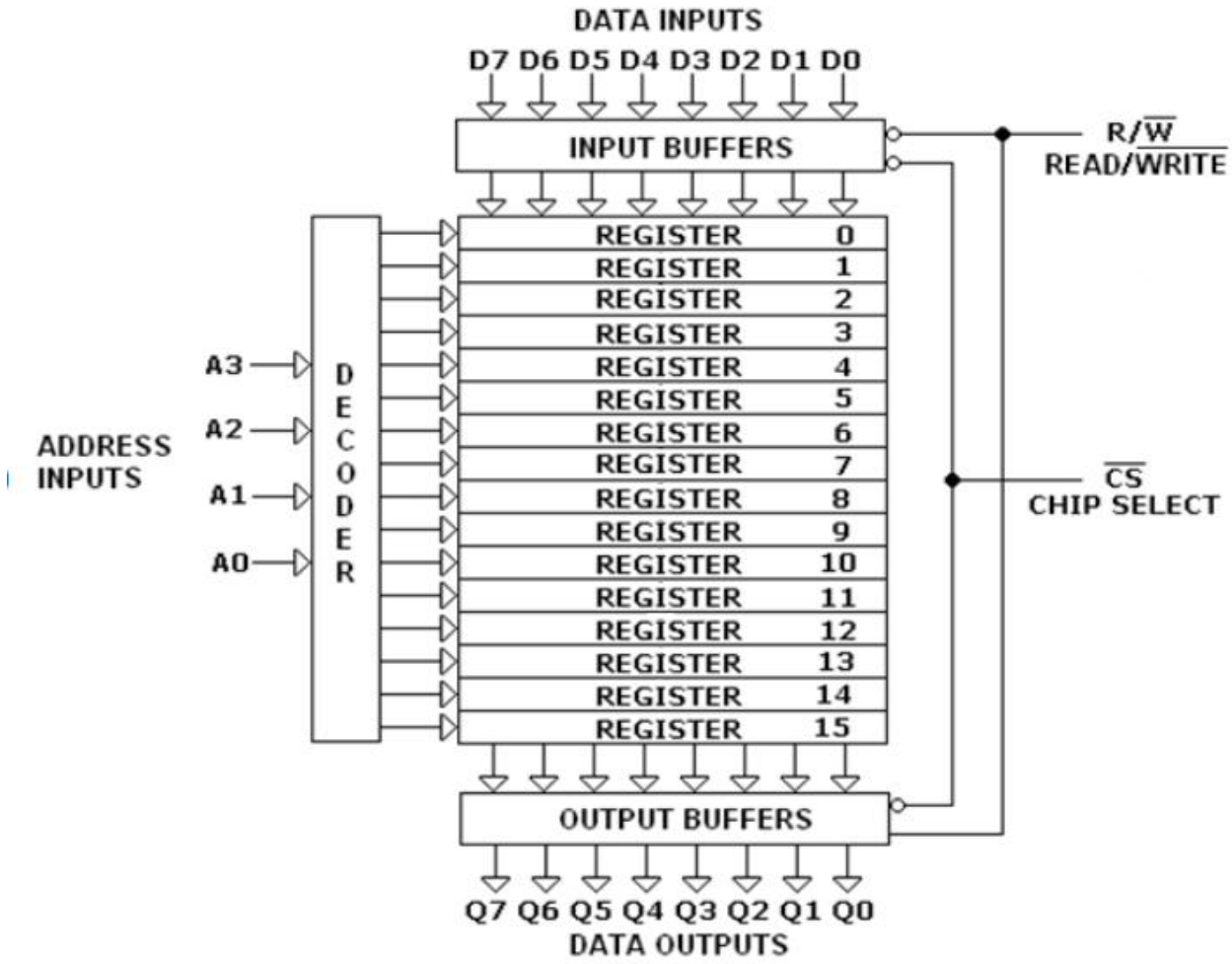
• **Data Output Lines (8 bits):** Output lines where data is read from the RAM.

• **Write Enable (WE):** Control signal to specify whether to write data to the memory.

• **Clock (CLK):** Clock signal for synchronization.

• **Memory Array:** 16x8 storage cells.

• **Multiplexer/Demultiplexer:** For routing the correct data lines based on the address.



1. The internal structure of a 16x8 RAM.

Table of Descriptions

Element	Description
Address Lines (4)	These lines (A0, A1, A2, A3) select one of the 16 locations in the RAM.
Data Input Lines (8)	Data to be written to the selected memory location is provided here.
Data Output Lines (8)	Data read from the selected memory location is output here.
Write Enable (WE)	This control signal determines whether a write operation should occur.
Clock (CLK)	The clock signal synchronizes all operations.
Address Decoder	Decodes the 4-bit address to access one of the 16 memory locations.
Data Input Mux/Write Logic	Manages data routing for write operations.
Data Output Mux	Manages data routing for read operations.
Write Enable Control	Ensures data is written to memory only when the WE signal is active.

PROCEDURE:

STEP 1: Project creation

• Open Vivado software > Create a new project > provide project name & select the project location > click on next > Select project type as “RTL Project” > In Add sources click on Create File > type in the file name “ram” > click on next.

• Select Product Category : **General Purpose >** Select Family : **Artix-7 >** Select Package: **csg324 >** Select Speed grade : **-1 >** Select Product Category : **General Purpose >** Select Temp grade : **All Remaining >** Select **xc7a100tcsg324-1 >** Click on next > Click on Finish.

STEP 2: Add the code , Testbench and Constraints

• Add code for ram.v and click on save

```
module ram (
  input wire clk,
  input wire write_en,
  input wire [3:0] address,
  input wire [7:0] data_in,
  output reg [7:0] data_out
);

  // Memory array
  reg [7:0] memory_array [0:15];

  // Write and read operations
  always @(posedge clk) begin
    if (write_en) begin
      memory_array[address] <= data_in; // Write operation
    end
    data_out <= memory_array[address]; // Read operation
  end

endmodule
```

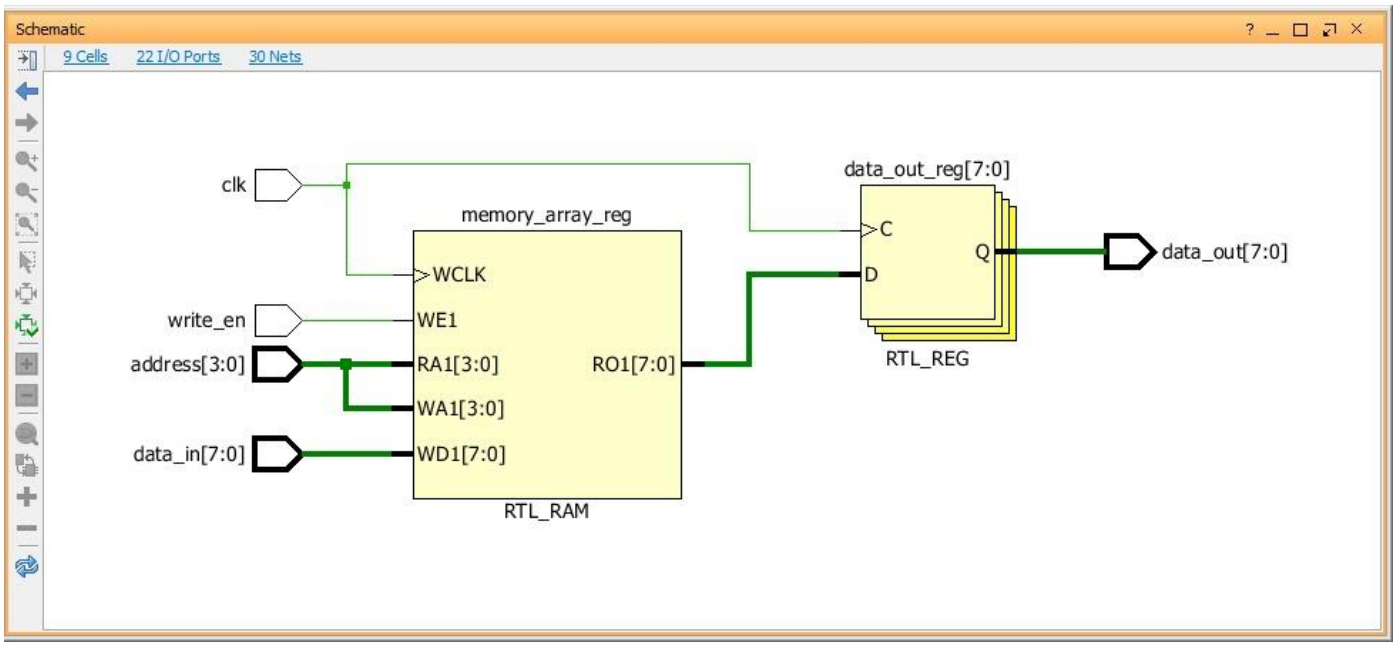
• Right click on Simulation Sources > Add sources > Select “Add or create simulation sources” > Click Next > Create file “ram_tb” > Click Finish > Add the code in here.

```
module ram_tb;

    reg clk;
    reg write_en;
    reg [9:0] address;
    reg [7:0] data_in;
    wire [7:0] data_out;

    ram uut (
        .clk(clk),
        .write_en(write_en),
        .address(address),
        .data_in(data_in),
        .data_out(data_out)
    );

    initial begin
        clk = 0;
        write_en = 0;
        address = 0;
        data_in = 8'h00;

        // Write operation
        #10;
        write_en = 1;
        address = 10;
        data_in = 8'hFF;
        #10;
        write_en = 0;

        // Read operation
        #10;
        address = 10;
        write_en = 0;

        // Add more test cases as needed

        $finish;
    end

    always #5 clk = ~clk;

endmodule
```

• Right click on Constraints > Add constraints > Select “Add or create constraints” > Click Next > Create file “ram_constraints” > Click Finish > Add the code in here.

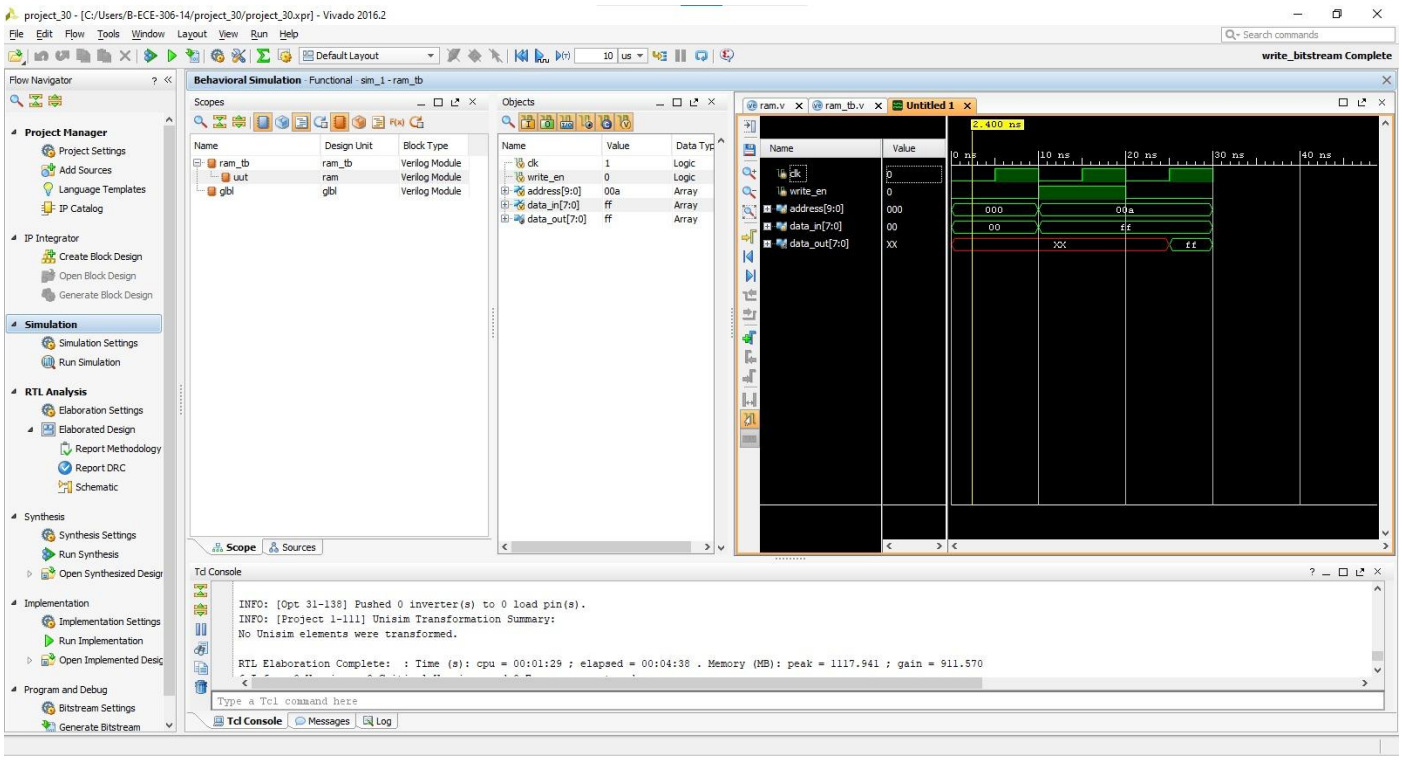
```
## Clock signal
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

## Write enable signal
set_property PACKAGE_PIN F4 [get_ports write_en]
set_property IOSTANDARD LVCMOS33 [get_ports write_en]

## Address signals
set_property PACKAGE_PIN J15 [get_ports {address[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {address[0]}]
set_property PACKAGE_PIN L16 [get_ports {address[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {address[1]}]
set_property PACKAGE_PIN M13 [get_ports {address[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {address[2]}]
set_property PACKAGE_PIN M14 [get_ports {address[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {address[3]}]

## Data input signals
set_property PACKAGE_PIN P4 [get_ports {data_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[0]}]
set_property PACKAGE_PIN P3 [get_ports {data_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[1]}]
set_property PACKAGE_PIN R3 [get_ports {data_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[2]}]
set_property PACKAGE_PIN T1 [get_ports {data_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[3]}]
set_property PACKAGE_PIN T3 [get_ports {data_in[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[4]}]
set_property PACKAGE_PIN U2 [get_ports {data_in[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[5]}]
set_property PACKAGE_PIN V2 [get_ports {data_in[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[6]}]
set_property PACKAGE_PIN U4 [get_ports {data_in[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[7]}]

## Data output signals
set_property PACKAGE_PIN P2 [get_ports {data_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[0]}]
set_property PACKAGE_PIN R2 [get_ports {data_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[1]}]
set_property PACKAGE_PIN U1 [get_ports {data_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[2]}]
set_property PACKAGE_PIN P5 [get_ports {data_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[3]}]
set_property PACKAGE_PIN R1 [get_ports {data_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[4]}]
set_property PACKAGE_PIN V1 [get_ports {data_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[5]}]
set_property PACKAGE_PIN U3 [get_ports {data_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[6]}]
set_property PACKAGE_PIN V4 [get_ports {data_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[7]}]
```

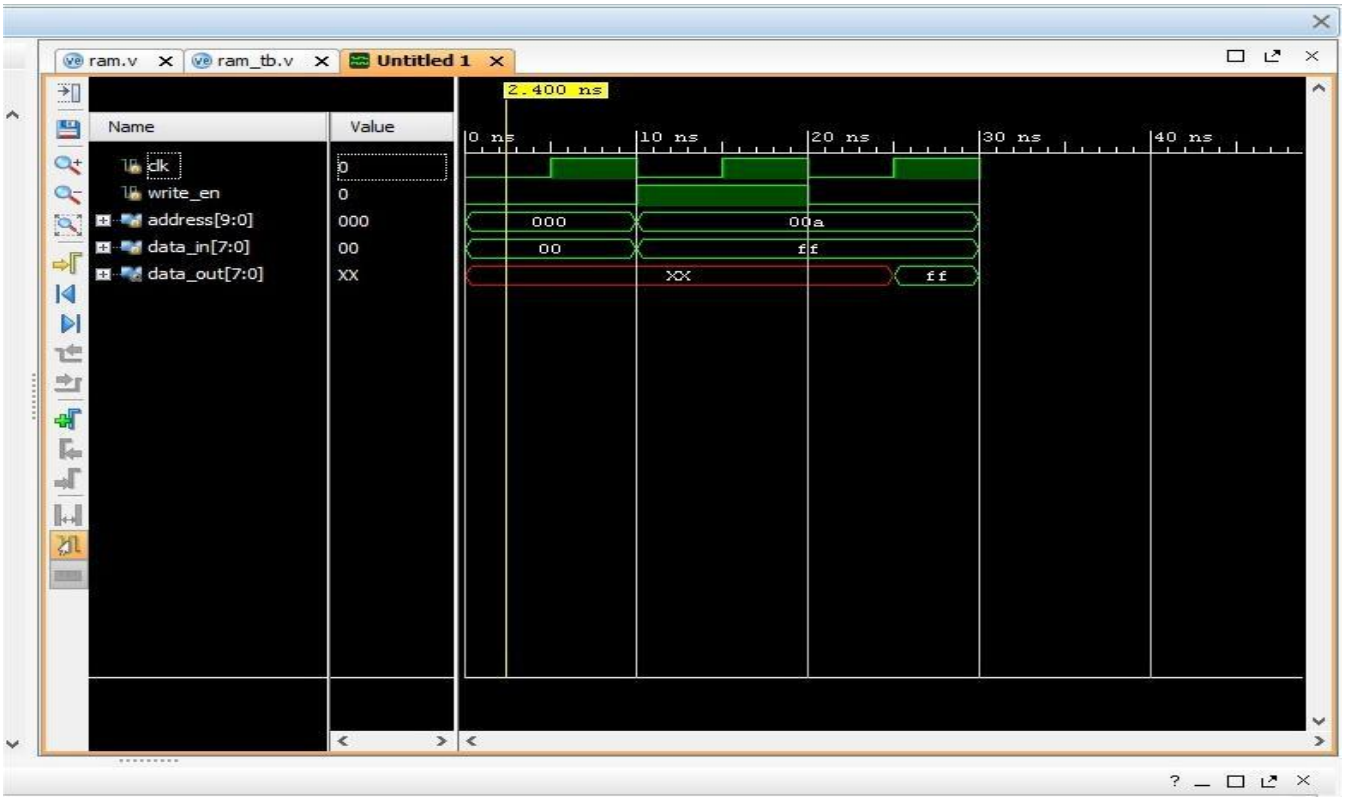
STEP 3 : Open Elaborate Design



STEP 4 : Simulation Results

- Click on Run Simulation



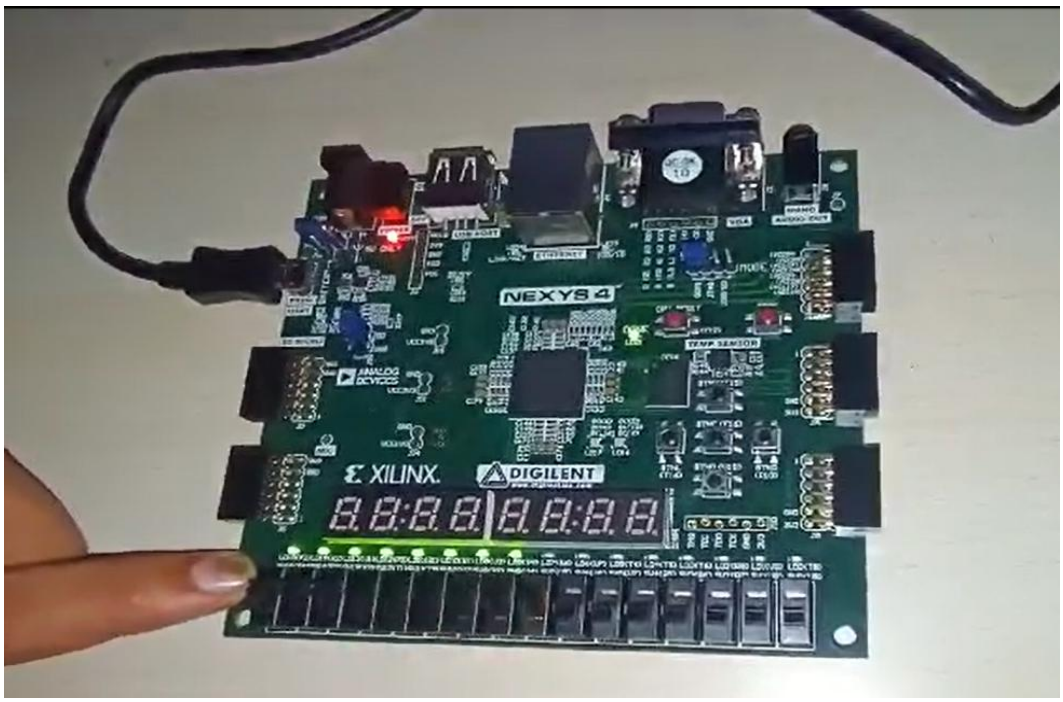


STEP 5: Synthesize & Implementation

• Connect the FPGA Kit to your computer.

• Click on Run synthesis > Click on Run Implementation > Open Hardware Manager > Select Open Target > Auto Connect > Click on program device. Check for any errors or warnings and ensure they are resolved.

• FPGA Results:



Explanation of Results obtained on FPGA Kit :

• The 16x8 RAM implies you have 16 address lines (address[3:0]) and 8 data lines (data_in[7:0] and data_out[7:0]).

• RAM is a memory device that stores data at specific addresses. Reading from RAM involves providing an address (address[3:0]) and retrieving the data (data_out) stored at that address.

• Read Operation typically involves setting the address (address[3:0]) to the desired memory location and then initiating a read cycle. The read cycle might be triggered by a clock edge.

• Data Output (data_out): data_out represents the data read from the RAM at the specified address (address[3:0]).

• LED turns on as the input (data_in) goes from 0 to 1 on your FPGA kit, it typically signifies that a read operation from the 16x8 RAM was initiated successfully, and the LEDs are showing the data (data_out) read from the RAM location specified by address[3:0]. This behavior confirms the functionality of your RAM read operation and the interaction between your Verilog code and the FPGA hardware.

CONCLUSION:

In conclusion, the design and implementation of a 16x8 RAM module on an FPGA kit represent a significant advancement in digital system performance and flexibility. By leveraging Verilog for hardware description and Xilinx Vivado for FPGA synthesis and implementation, the project achieves efficient memory management with minimal latency. Key components such as address decoding, data input/output multiplexing, and synchronous control logic ensure reliable operation within the FPGA architecture.

The RAM module's ability to store and retrieve data on command, coupled with its integration into larger digital systems, underscores its utility in applications ranging from DSP and embedded systems to educational environments. The use of synchronous single-port access optimizes memory performance, enhancing overall system responsiveness and data throughput. This project not only demonstrates the power of FPGA technology in custom hardware design but also highlights its versatility in meeting diverse computational needs effectively.

REFERENCES & LINKS:

- **Drive link for the video tutorial of this project:**
https://drive.google.com/drive/folders/1gDJ_6FV3sRVH6F76ynSujg_Q3SLYpg7n?usp=sharing

- **Getting start with Xilinx Vivado:**
https://digilent.com/reference/vivado/getting_started/start

- **Basics of RAM**:
https://www.geeksforgeeks.org/random-access-memory-ram-and-read-only-memory-rom/