

DESIGN OF RAM CIRCUIT USING VHDL

***A Project Report for Winter Industrial Training***

***Submitted By,***

***Dilip kumar panigrahi (ECE 3rd Year, GIET)***

***D.Gouri shankar(ECE 3rd Year,GIET )***

**BONAFIDE CERTIFICATE**

CERTIFIED THAT THIS PROJECT WORK WAS CARRIED OUT UNDER MY SUPERVISION

**“DESIGN OF RAM CIRCUIT USING VHDL”**IS THE BONAFIDE WORKOF

* **Dilip kumar panigrahi**
* **D.Gouri shankar**

**PROJECT MENTOR:**

**SIGNATURE**

**Ardent Original Seal**

**ACKNOWLEDGEMENT**

We take this opportunity to express our deep gratitude and sincerest thank to our project mentor, for giving most valuable suggestion, helpful guidance and encouragement in the execution of this project work.

Last but not the least I am grateful to all the faculty members of Ardent ComputechPvt. Ltd. for their support.

**INTRODUCTION**

RAM(Random-access memory) is a form of computer data storage. A random-access memory device allows data items to be accessed (read or written) in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement delays.

Today, random-access memory takes the form of integrated circuits . RAM is normally associated with volatile types of memory(such as DRAM memory modules),where stored information is lost if power is removed, although many efforts have been made to develop non-volatile RAM chips. Other types of non-volatile memory exist that allow random access for read operations, but either do not allow write operations or have limitations on them. These include most types of ROM and a type of flash memory called NOR-Flash.

Integrated-circuit RAM chips came into the market in the late 1960s, with the first commercially available DRAM chip, the intel1103, introduced in October 1970.

**OBJECTIVES**

The main purposes of this project are:

* To design a RAM circuit
* Each input to produce a required output based on the output selector line.
* The possible outputs to be OR, AND, NOR, NOT, XOR, NAND, XNOR, ADDER, INCREMENT, SUBTRACTOR, DECREMENT AND TRANFER SAME.
* Our task is also to design and implement the RAM circuit using the Xilinx Foundation tools.

**What is VHDL??**

**VHDL** ([VHSIC](http://en.wikipedia.org/wiki/VHSIC) (Very High Speed Integrated Circuits) [hardware description language](http://en.wikipedia.org/wiki/Hardware_description_language)) is commonly used as a design-entry language for [field-programmable gate arrays](http://en.wikipedia.org/wiki/Field-programmable_gate_array) . VHDL was originally developed at the behest of the [US](http://en.wikipedia.org/wiki/United_States) [Department of Defence](http://en.wikipedia.org/wiki/United_States_Department_of_Defense) in order to document the behaviour of the [ASICs](http://en.wikipedia.org/wiki/Application-specific_integrated_circuit) that supplier companies were including in equipment. That is to say, VHDL was developed as an alternative to huge, complex manuals which were subject to implementation-specific details.

The idea of being able to simulate this documentation was so obviously attractive that [logic simulators](http://en.wikipedia.org/wiki/Logic_simulation) were developed that could read the VHDL files. The next step was the development of [logic synthesis](http://en.wikipedia.org/wiki/Logic_synthesis) tools that read the VHDL, and output a definition of the physical implementation of the circuit. Modern synthesis tools can extract [RAM](http://en.wikipedia.org/wiki/Random_Access_Memory), [counter](http://en.wikipedia.org/wiki/Counter), and arithmetic blocks out of the code, and implement them according to what the user specifies.

VHDL borrows heavily from the [Ada programming language](http://en.wikipedia.org/wiki/Ada_(programming_language)) in both concepts (for example, the slice notation for indexing part of a one-dimensional array) and [syntax](http://en.wikipedia.org/wiki/Syntax). VHDL is not a case sensitive language. One can design hardware in a VHDL IDE (such as Xilinx or Quartus) to produce the [RTL](http://en.wikipedia.org/wiki/Register_transfer_level) schematic of the desired circuit. After that, the generated schematic can be verified using simulation software (such as ModelSim) which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. To generate an appropriate test bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

The key advantage of VHDL when used for systems design is that it allows the behaviour of the required system to be described (modelled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system (many parts, each with its own sub-behaviour, working together at the same time).

A final point is that when a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a [CPLD](http://en.wikipedia.org/wiki/CPLD) or [FPGA](http://en.wikipedia.org/wiki/FPGA), and then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip.

**USED TOOLS AND SOFTWARES**

* Xilinx
* DSCH

**DESIGN ,DESCRIPTION, EVALUATION OF EACH BLOCK OF RAM CIRCUIT WITH VHDL CODE, SIGNALING DIAGRAM**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1-BIT NOT GATE**   |  |  | | --- | --- | | TRUTH TABLE | | | X | Y | | 0 | 1 | | 1 | 0 |     **VHDL CODE**  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  ---- Uncomment the following library declaration if instantiating  ---- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity NOT1BIT is  Port ( X : in STD\_LOGIC;  Z : out STD\_LOGIC);  end NOT1BIT;  architecture Behavioral of NOT1BIT is  begin  Z<=NOT X;  end Behavioral;    TEST BENCH WAVE FROM  **1-BIT AND GATE**     |  |  |  | | --- | --- | --- | | TRUTH TABLE | | | | X | Y | Z | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 |   **VHDL CODE**  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  ---- Uncomment the following library declaration if instantiating  ---- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity AND1BIT is  Port ( A : in STD\_LOGIC;  B : in STD\_LOGIC;  C : out STD\_LOGIC);  end AND1BIT;  architecture Behavioral of AND1BIT is  begin  C <= A AND B;  end Behavioral;  **TEST BENCH WAVFORM**  **1-BIT NAND GATE**   |  |  |  | | --- | --- | --- | | TRUTH TABLE | | | | X | Y | Z | | 0 | 0 | 1 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 |     **VHDL CODE**  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  ---- Uncomment the following library declaration if instantiating  ---- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity NAND1BIT is  Port ( A : in STD\_LOGIC;  B : in STD\_LOGIC;  C : out STD\_LOGIC);  end NAND1BIT;  architecture Behavioral of NAND1BIT is  begin  C<=NOT(A AND B);  end Behavioral;  **TEST BENCH WAVEFORM**    **4-BIT NAND GATE**  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  ---- Uncomment the following library declaration if instantiating  ---- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity FBNAND is  Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0);  B : in STD\_LOGIC\_VECTOR (3 downto 0);  Z : out STD\_LOGIC\_VECTOR (3 downto 0));  end FBNAND;  architecture Behavioral of FBNAND is  begin  Z(0)<= A(0) NAND B(0);  Z(1)<= A(1) NAND B(1);  Z(2)<= A(2) NAND B(2);  Z(3)<= A(3) NAND B(3);  end Behavioral;  **TEST BENCH WAVEFORM**    **S-R FLIP FLOP**  **VHDL CODE**   |  |  |  |  | | --- | --- | --- | --- | | **S** | **R** | **Q+** | **Action** | | **0** | 0 | Q | **NoChange** | | 0 | 1 | 0 | Reset | | 1 | 0 | 1 | Set | | 1 | 1 | - | Illegal |   **Set-ResetTruth table**  Library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  ---- Uncomment the following library declaration if instantiating  ---- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity srf is  Port ( s : in STD\_LOGIC\_VECTOR (1 downto 0);  re : in STD\_LOGIC;  clk : in STD\_LOGIC;  i : in STD\_LOGIC;  n : in STD\_LOGIC;  q : out STD\_LOGIC);  end srf;  architecture Behavioral of srf is  begin  process (s,n,i,re,clk)  begin  if(re='1')then  q<='0';  elsif(clk='1' and clk 'event)then  case s is  when "00" =>q<=n;  when "01" =>q<='0';  when "10" =>q<='1';  when "11" =>q<=i;  when others =>null;  end case;  end if;  end process;  end Behavioral;  **TEST BENCH WAVEFORM** |

**VHDL CODE:**

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_std.all;

entity Ram\_TB is

end entity Ram\_TB;

architecture Bench of Ram\_TB is

  signal Address :Std\_logic\_vector(3 downto 0);

  signal DataIn, DataOut :Std\_logic\_vector(3 downto 0);

  signal WE : Std\_logic;

  signal clock : Std\_logic;

  signal StopClock : boolean := FALSE;

begin

  UUT: entity work.sync\_ram(RTL)

  port map (

    clock => clock,

    we => WE,

    address => Address,

    datain => DataIn,

    DataOut => DataOut

  );

  ClockGen: process is

  begin

    while not StopClock loop

      clock <= '0';

      wait for 5 ns;

      clock <= '1';

      wait for 5 ns;

    end loop;

    wait;

  end process ClockGen;

  Stim: process is

  begin

    wait until rising\_edge(clock); -- cycle 1

    datain <= "0000";

    address <= "0001";

    we <= '0';

    wait until rising\_edge(clock); -- cycle 2

    we <= '1';

    datain <= "0100";

    wait until rising\_edge(clock); -- cycle 3

    datain <= "0111";

    address <= "0010";

    wait until rising\_edge(clock); -- cycle 4

    we <= '0';

    wait until rising\_edge(clock); -- cycle 5

    address <= "0001";

    wait until rising\_edge(clock); -- cycle 6

    address <= "0010";

    wait until rising\_edge(clock); -- cycle 7

    wait until rising\_edge(clock); -- cycle 8

    StopClock <= true;

    wait;

  end process;

end architecture Bench;

architecture Bench2 of Ram\_TB is

  signal Address :Std\_logic\_vector(3 downto 0);

  signal DataIn, DataOut :Std\_logic\_vector(3 downto 0);

  signal WE : Std\_logic;

  signal clock : Std\_logic;

  signal StopClock : boolean := FALSE;

  signal ok: boolean := true;

begin

  UUT: entity work.sync\_ram(RTL)

  port map (

    clock => clock,

    we => WE,

    address => Address,

    datain => DataIn,

    DataOut => DataOut

  );

  ClockGen: process is

  begin

    while not StopClock loop

      clock <= '0';

      wait for 5 ns;

      clock <= '1';

      wait for 5 ns;

    end loop;

    wait;

  end process ClockGen;

  Stim: process is

  begin

    -- Initialise input signals

    address <= "0000";

    datain <= "0000";

    we <= '0';

    wait until rising\_edge(clock);

    -- write to all addresses

    while address /= "1111" loop

      we <= '1';

      wait until rising\_edge(clock);

      address <= std\_logic\_vector(unsigned(address) + 1);

      datain <= std\_logic\_vector(unsigned(datain) + 1);

    end loop;

    -- stop writing

    address <= "0000";

    datain <= "0000";

    we <= '0';

    wait until rising\_edge(clock);

    -- read from all addresses

    while address /= "1111" loop

      address <= std\_logic\_vector(unsigned(address) + 1);

      wait until rising\_edge(clock);

      -- ok should permanently go false on the first error

      ok <= ok and dataout = std\_logic\_vector(unsigned(address(3 downto 0)) - 1);

    end loop;

    StopClock <= true;

    wait;

  end process;

end architecture Bench2;

**VHDL Code for Ram Circuit:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.Numeric\_Std.all;

entity sync\_ram is

  port (

    clock : in std\_logic;

    we : in std\_logic;

    address : in std\_logic\_vector (3 downto 0);

    datain : in std\_logic\_vector (3 downto 0);

    dataout : out std\_logic\_vector (3 downto 0)

  );

end entity sync\_ram;

architecture RTL of sync\_ram is

   type ram\_type is array (0 to (2\*\*address'length)-1) of std\_logic\_vector(datain'range);

   signal ram : ram\_type;

   signal read\_address : std\_logic\_vector(address'range);

begin

  RamProc: process(clock) is

  begin

    if rising\_edge(clock) then

      if we = '1' then

        ram(to\_integer(unsigned(address))) <= datain;

      end if;

      read\_address <= address;

    end if;

  end process RamProc;

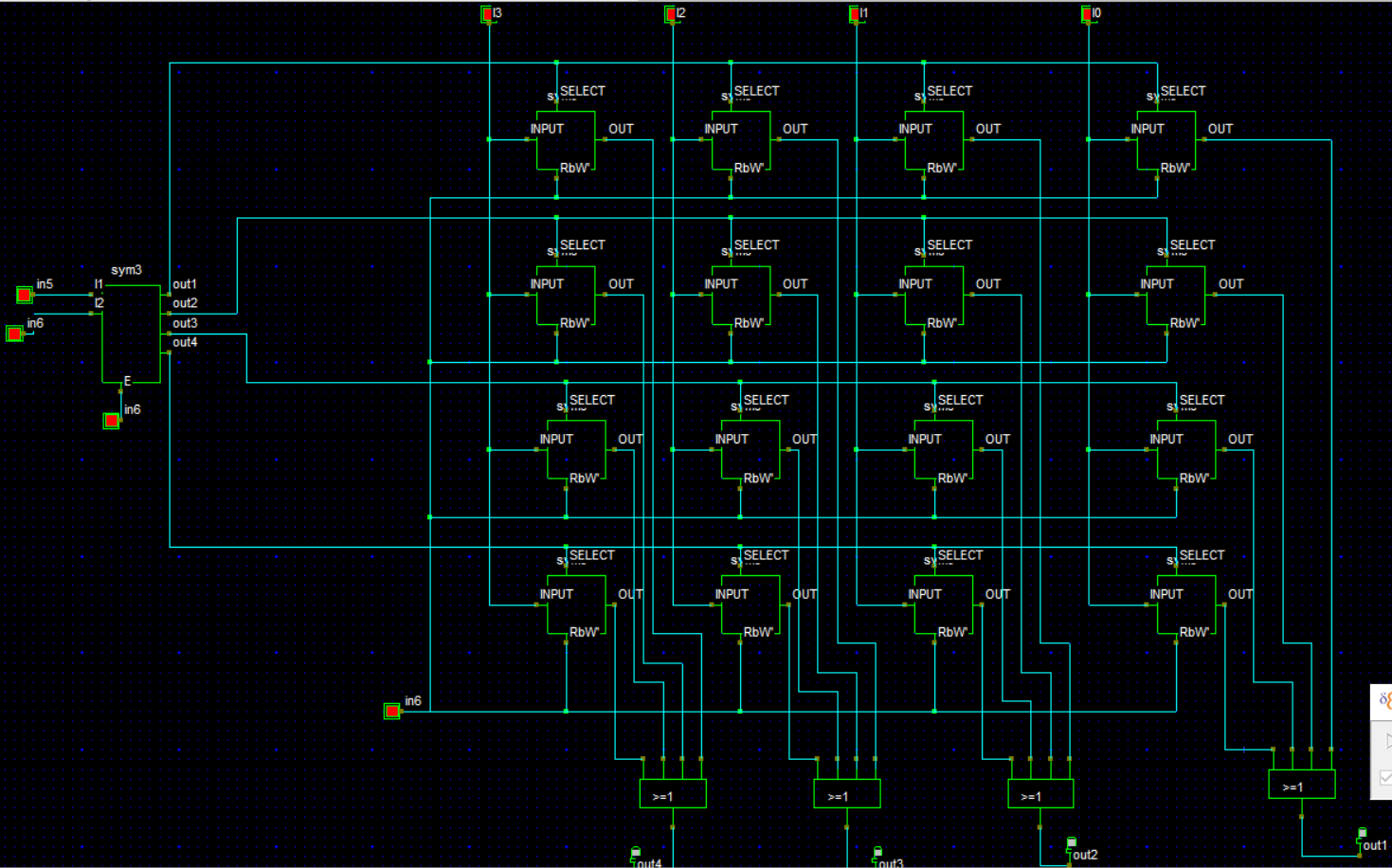
  dataout <= ram(to\_integer(unsigned(read\_address)));

end architecture RTL;

**Circuit Diagram:**

In this Circuit we use Binary Cell,2X4 Decoder,4 input nor gate.

For input we use switch & for output use Led.

****

**Conclusion:**

We have noticed clearly that all memory cells are separate from each other and independent in their work therefore been linked to these cells gates logical OR, a gateway linking every bit in a cell to another bit in another cell and so that is linked to all cells.The Memory RAM industry has a very great importance to the RAM memory of importance in the computer system so

take them in this paper and explain designed using Flip\_Flop gates because of its ease and simplicity in understanding and design instead of using the complex transistor in design mode, because when the memory is designed using transistor must to have a great knowledge of his way.But in general,this easy-to-Memory and industry must take into account the possibility of its industry locally.