Computer Architecture Lab Session 3

# ROM & RAM

# Bardia Ardakanian 9831072

# Ali Asad 9831004

## طراحی یک Read-only memory با زبان VHDL

برای نوشتن یک Read-only memory ابتدا موجودیت Rom را تعریف می‌کنیم، برای سنکروم بودن ROM برایش clock و reset می‌گذاریم. در این برنامه ما سعی در طراحی یک ROM با اندازه 16x8 بوده‌ایم. به حالتی که اعداد 0 الی 15 را در خود ذخیره کند.

کد زیر برنامه موجودیت ROM است.

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all; use ieee.std\_logic\_arith.all; use ieee.std\_logic\_unsigned.all;  --ROM with clock and reset --each word is 1 bytes (or 8 bits) --this architecture is 16 x 8 entity ROM is port(  clk : in std\_logic;  rst : in std\_logic;  enable : in std\_logic;  read : in std\_logic;  address : in integer range 0 to 15;  dout : out std\_ulogic\_vector(7 downto 0) ); end ROM;  architecture behave of ROM is  type ROM\_Array is array (0 to 15) of std\_ulogic\_vector (7 downto 0);   constant content: ROM\_Array := (  "00000000", "00000001", "00000010", "00000011",  "00000100", "00000101", "00000110", "00000111",  "00001000", "00001001", "00001010", "00001011",  "00001100", "00001101", "00001110", "00001111"  );  begin      process(clk, rst, enable, read, address)     begin         if(rst = '1') then              dout <= "ZZZZZZZZ";         elsif(clk'event and clk = '1') then             if (enable = '1') then                 if (read = '1') then                      dout <= content(address);                 else                      dout <= "ZZZZZZZZ";                 end if;             end if;         end if;     end process; end behave; |

## Test Bench برای ROM

برای طراحی Test Bench ابتدا باید موجودیت ROM\_TB را بسازیم، حال سیگنال های ورودی را تعریف می‌کنیم. در ابتدا سیگنال clock را 1 می‌گذاریم و هر 5 نانو ثانیه ان را not می‌کنیم تا ROM ما سنکروم شود.

بعد از این مرحله سیگنال reset را 1 می‌گذاریم تا برنامه از حالت خاصی شروع نشود. مقدار اولیه سیگنال address را 0 می‌گذاریم زیرا می‌خواهیم از خانه 0 الی 15 را بخوانیم.

بعد از 20 نانو ثانیه تاخیر سیگنال reset را 0 می‌کنیم و سیگنال read را هم 1 می‌کنیم تا امکان خواندن داده از ROM فراهم شود.

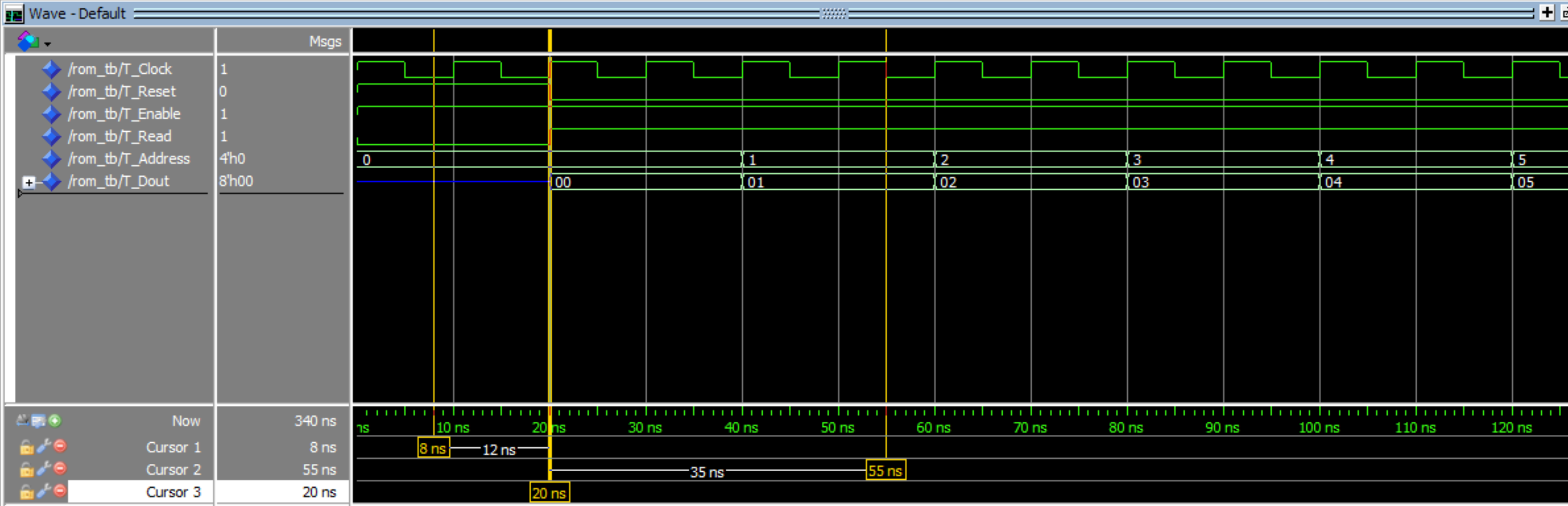
بعد در یک loop مقدار address را هر 20 نانو ثانیه 1 واحد ببیشتر می‌کنیم تا روی خانه های حافظه ROM پیمایش داشته باشیم.

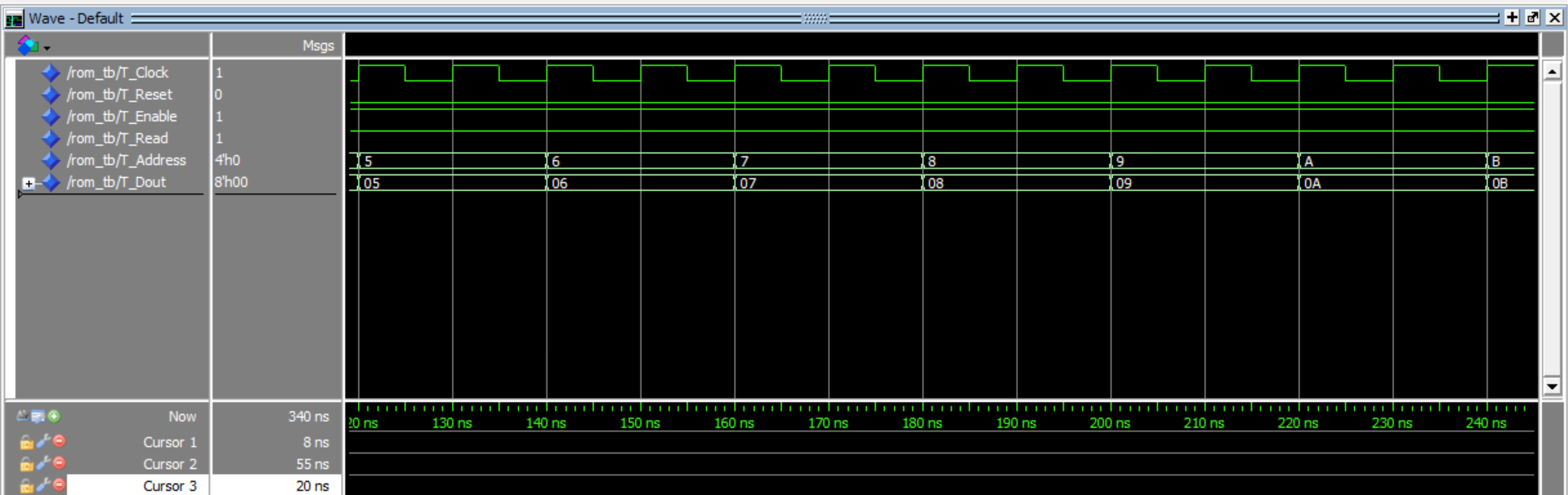
کد تست بنچ به صورت زیر است:

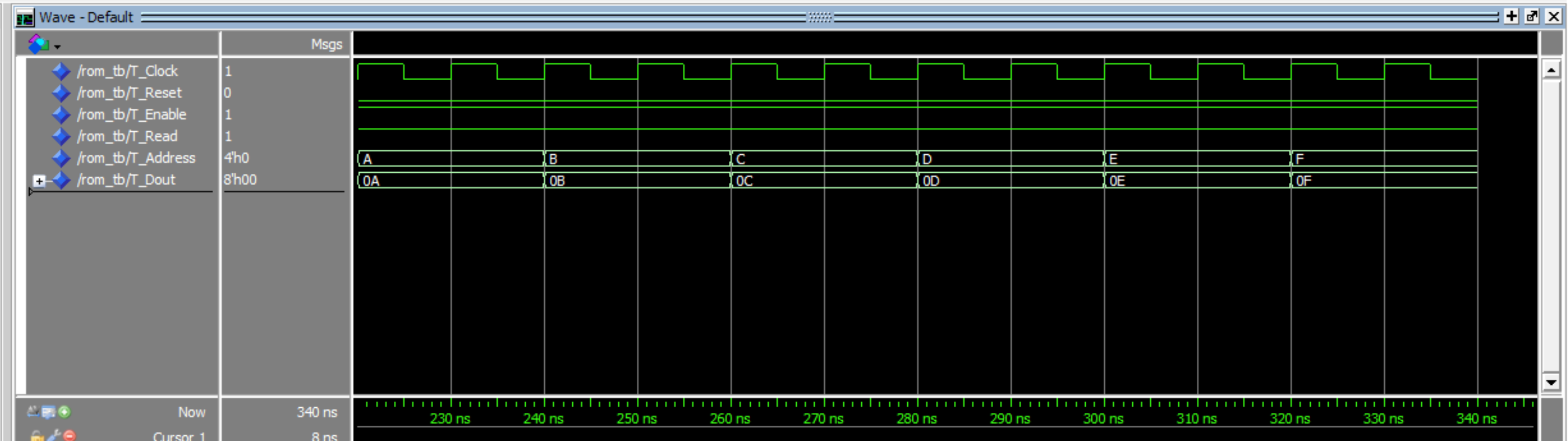
|  |
| --- |
| ------------------------------------------------------------- -- Test Bench for 16\*8 ROM module  -------------------------------------------------------------   library IEEE; use IEEE.std\_logic\_1164.all; use IEEE.std\_logic\_arith.all; use ieee.std\_logic\_unsigned.all;   entity ROM\_TB is end ROM\_TB;    architecture TB of ROM\_TB is  component ROM is port(  clk : in std\_logic;  rst : in std\_logic;  enable : in std\_logic;  read : in std\_logic;  address : in integer range 0 to 15;  dout : out std\_ulogic\_vector(7 downto 0) ); end component;  --init signals signal T\_Clock, T\_Reset, T\_Enable, T\_Read : std\_logic; signal T\_Address : integer range 0 to 15; signal T\_Dout : std\_ulogic\_vector(7 downto 0);  begin --port mapping      U\_ROM: ROM port map (  T\_Clock,   T\_Reset,   T\_Enable,   T\_Read,   T\_Address,   T\_Dout     );      clock\_process: process          --clock     begin         T\_Clock<='1';         wait for 5 ns;         T\_Clock<='0';         wait for 5 ns;     end process;       process      begin  --init reset as 1 so we are not in any specific state  T\_Enable <= '1';  T\_Read <= '0';  T\_Reset <= '1';  T\_Address <= 0;  wait for 20 ns;     --reset = 0 and read = 1 so we can read data from ROM  T\_Reset <= '0';  T\_Read <= '1';    --loop on ROM array addresses  for i in 0 to 31 loop      wait for 20 ns;      T\_Address <= T\_Address + 1;   end loop;   wait;     end process;  end TB; |

## Wave خروجی برنامه

Wave خروجی Test Becnh برنامه ما به صورت زیر است:



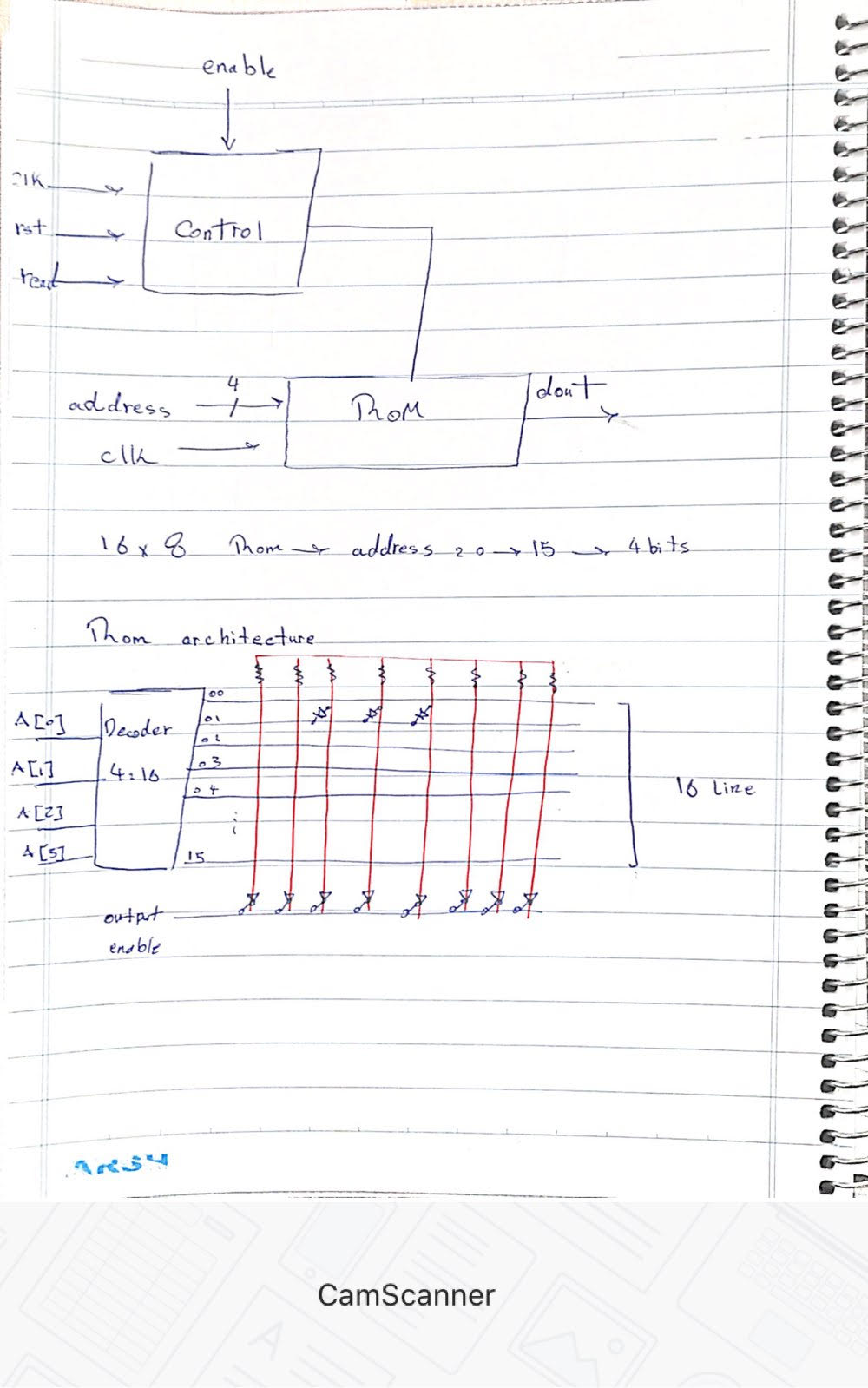




همانطور که در Wave خروجی مشاهده می‌کنید اعداد خانده شده از روی ROM از 0 الی 15 است و برنامه ما خروجی درست نشان داده است.

## شماتیک ROM

همانطور که مشاهده می‌کنید واحد کنترلی اجازه خروج داده را از ROM می‌دهد و خود ROM از یک دیکودر 4 به 16 استفاده می‌کند چون 16x8 هست و ادرس ها 4 بیتی هستند.



## RAM

میخواهیم یک Random Access Memory با دو ورودی (Dual Port) طراحی کنیم.

بدین صورت که همزمان بتواند از این حافظه بنویسیم یا بخوانیم.

برای طراحی این واحد حافظه در زبان VDHL ،‌ ابتدا موجودیت مورد نظر را تعریف کرده و سپس ورودی و خروجی های مربوطه را مشخص می‌کنیم.

سپس به طراحی معماری رفتاری این ماژول می‌پردازیم.

بدین صورت که ابتدا یک subtype ۸ بیتی تعریف میکنیم و سپس ۶۴ واحد از این زیرنوع را تعریف میکنیم. در واقع یک آرایه دوبعدی ساخته ایم که ۶۴\*۸ بیت میتواند ذخیره کند.

این آرایه همان RAM است .

حال در ادامه هر قسمت از عملیات های خواندن و نوشتن را در proccess جداگانه تعریف می‌کنیم و در صورت اجازه داشتن به نوشتن یا خواندن در هر کدام ،‌ به ترتیب در RAM در آدرس داده شده می‌نویسیم یا داده را از آدرس داده شده می‌خوانیم.

در ادامه کد RAM را به زبان VHDL مشاهده می‌کنید:

library ieee;

use ieee.std\_logic\_1164.all;

entity ram is

    port (

        arst     : in std\_logic;

        clk      : in std\_logic;

        enable   : in std\_logic;

        we       : in std\_logic;

        re       : in std\_logic;

        r\_addr   : in natural range 0 to 63;

        w\_addr   : in natural range 0 to 63;

        data\_in  : in std\_logic\_vector(7 downto 0);

        data\_out : out std\_logic\_vector(7 downto 0)

    );

end entity;

architecture behav of ram is

*-- Building a 2D-array for the RAM.*

*---------------------------------------------------*

*-- Defiening subtype word (8-bits).*

    subtype word\_t is std\_logic\_vector(7 downto 0);

*-- Defiening main type  (64 \* 8-bits).*

    type ram\_type is array(63 downto 0) of word\_t;

*---------------------------------------------------*

*-- Declaring the RAM signal.*

    signal ram\_sig : ram\_type;

begin

*-- Read Proccess Section*

    process (arst, clk, enable, re)

    begin

        if arst = '1' then

            data\_out <= (data\_out'range => 'Z');

        elsif (clk'event and clk = '1') then

            if enable = '1' then

                if re = '1' then

                    data\_out <= ram\_sig(r\_addr);

                else

                    data\_out <= (data\_out'range => 'Z');

                end if;

            end if;

        end if;

    end process;

*-- End of Read Proccess Section*

*-- Write Proccess Section*

    process (clk, enable, we)

    begin

        if (clk'event and clk = '1') then

            if enable = '1' then

                if we = '1' then

                    ram\_sig(w\_addr) <= data\_in;

                end if;

            end if;

        end if;

    end process;

*-- End of Write Proccess Section*

end behav;

## Test Bench برای RAM

برای اینکه به درستی ماژول RAM‌ی که طراحی کردیم پی ببریم تست بنچ برای آن نوشته و خروجی تست بنچ را با خروجی مورد نظر خود مقایسه می‌کنیم.

برای طراحی Test Bench ابتدا باید موجودیت ram\_tb را بسازیم، حال سیگنال های ورودی را تعریف می‌کنیم. در ابتدا سیگنال clock را 1 می‌گذاریم و هر 5 نانو ثانیه ان را not می‌کنیم تا RAM ما سنکروم شود.

بعد از این مرحله سیگنال reset را ۰ می‌گذاریم تا برنامه از حالت خاصی شروع نشود. مقدار اولیه سیگنال‌های w\_addr\_sig و r\_adde\_sig را 0 می‌گذاریم و re\_sig, we\_sig را غیر فعال می‌کنیم.

حال در دو حلقه به تست کد میپردازیم.

در حلقه اول که ۵ بار اجرا می‌شود، به ترتیب در آدرس های 05و 0Aو0F و 14و19 در RAM داده های

05و 0Aو0F و 14و19 را می‌نویسیم.

در حلقه دوم که ۵ بار اجرا می‌شود، به ترتیب از آدرس های 05و0A و0F و14 و19 در RAM داده های

05و 0Aو0F و 14و19 را می‌خوانیم.

با این حساب عملیات نوشتن و خواندن به درستی انجام می‌شود.

در ادامه کد تست بنچ و خروجی شبیه سازی را مشاهده می‌کنید.

کد تست بنچ به صورت زیر است:

*--------------------------------------------------------------------*

*-- Test Bench for memory module*

*-- use loop statement to test module completely*

*--------------------------------------------------------------------*

library ieee;

use ieee.std\_logic\_1164.all;

use IEEE.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity ram\_tb is *-- entity declaration*

end ram\_tb;

*--------------------------------------------------------------------*

architecture Testbench of ram\_tb is

    component ram is

        port (

            arst     : in std\_logic;

            clk      : in std\_logic;

            enable   : in std\_logic;

            we       : in std\_logic;

            re       : in std\_logic;

            r\_addr   : in natural range 0 to 63;

            w\_addr   : in natural range 0 to 63;

            data\_in  : in std\_logic\_vector(7 downto 0);

            data\_out : out std\_logic\_vector(7 downto 0)

        );

    end component;

    signal arst\_sig, clk\_sig, enable\_sig, re\_sig, we\_sig : std\_logic;

    signal r\_Addr\_sig, w\_Addr\_sig                        : natural range 0 to 63;

    signal data\_in\_sig, data\_out\_sig                     : std\_logic\_vector(7 downto 0);

begin

    ram\_compo : ram port map(

        arst\_sig, clk\_sig, enable\_sig, re\_sig, we\_sig,

        r\_Addr\_sig, w\_Addr\_sig,

        data\_in\_sig, data\_out\_sig

    );

    clk\_proccess : process

    begin

        clk\_sig <= '1'; *-- clock cycle 10 ns*

        wait for 5 ns;

        clk\_sig <= '0';

        wait for 5 ns;

    end process;

    process

    begin

        arst\_sig    <= '0';

        enable\_sig  <= '1';

        re\_sig      <= '0';

        we\_sig      <= '0';

        w\_Addr\_sig  <= 0;

        r\_Addr\_sig  <= 0;

        data\_in\_sig <= (data\_in\_sig'range => '0');

        wait for 20 ns;

*-- test write*

        for i in 0 to 4 loop

            w\_Addr\_sig  <= w\_Addr\_sig + 5;

            data\_in\_sig <= data\_in\_sig + "101";

            we\_sig      <= '1';

            re\_sig      <= '1';

            wait for 10 ns;

        end loop;

*-- test read*

        for i in 0 to 4 loop

            r\_Addr\_sig <= r\_Addr\_sig + 5;

            re\_sig     <= '1';

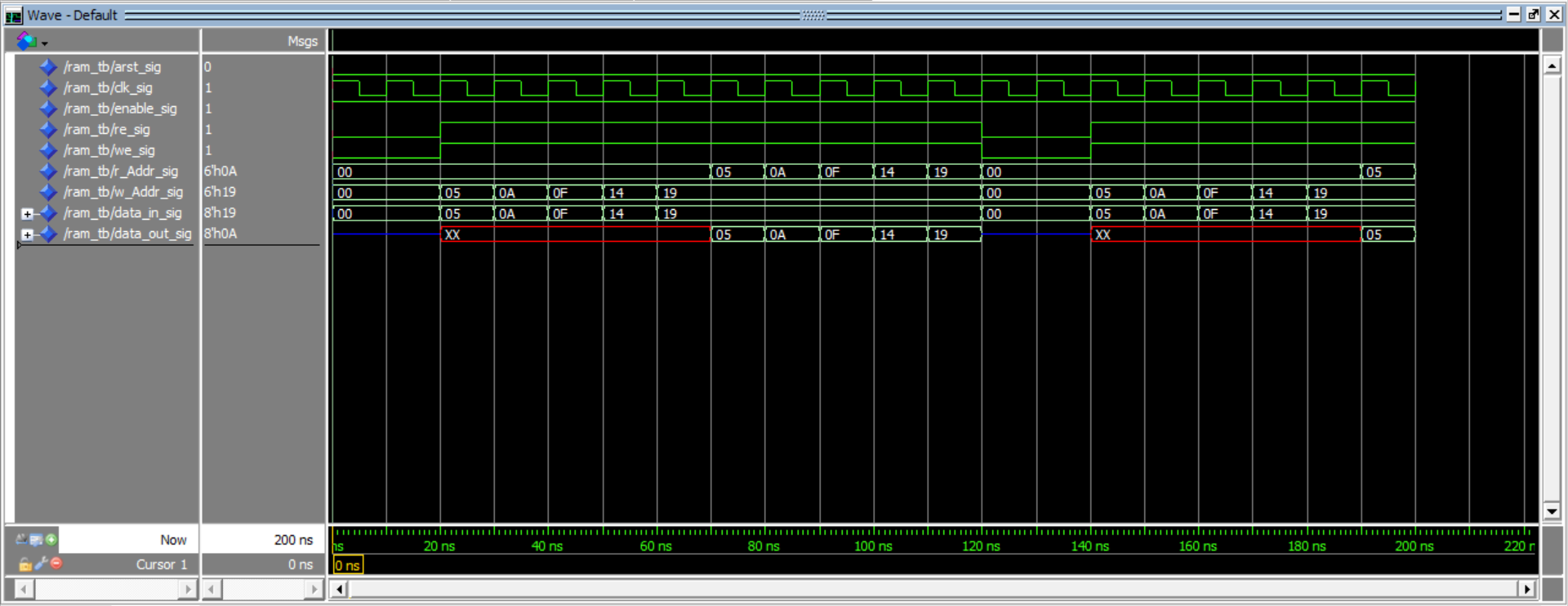
            wait for 10 ns;

        end loop;

    end process;

end Testbench;

## خروجی شبیه سازی



طبق توضیحات داده شده، خروجی درست است.

## شماتیک RAM

