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 ۰ الی ۱۵ را در خود ذخیره کند.

کد زیر برنامه موجودیت ROM است.

```vhdl
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 ۱ می‌گذاریم و هر ۵ نانو ثانیه ان را not می‌کنیم تا ROM ما سنکروم شود.

بعد از این مرحله سیگنال reset را ۱ می‌گذاریم تا برنامه از حالت خاصی شروع نشود. مقدار اولیه سیگنال address را ۰ می‌گذاریم زیرا می‌خواهیم از خانه ۰ الی ۱۵ را بخوانیم.

بعد از ۲۰ نانو ثانیه تاخیر سیگنال reset را ۰ می‌کنیم و سیگنال read را هم ۱ می‌کنیم تا امکان خواندن داده از ROM فراهم شود.

بعد در یک loop مقدار address را هر ۲۰ نانو ثانیه ۱ واحد ببیشتر می‌کنیم تا روی خانه های حافظه ROM پیمایش داشته باشیم.

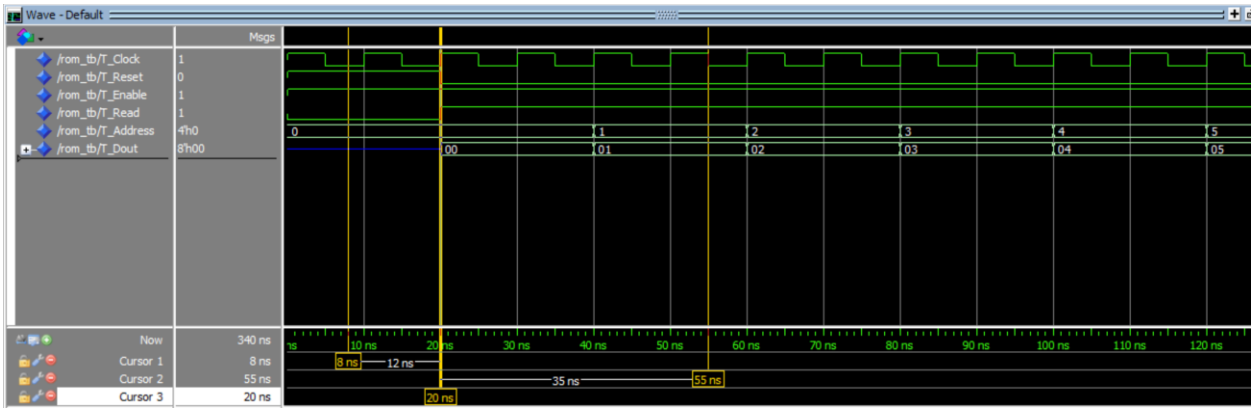
کد تست بنچ به صورت زیر است:

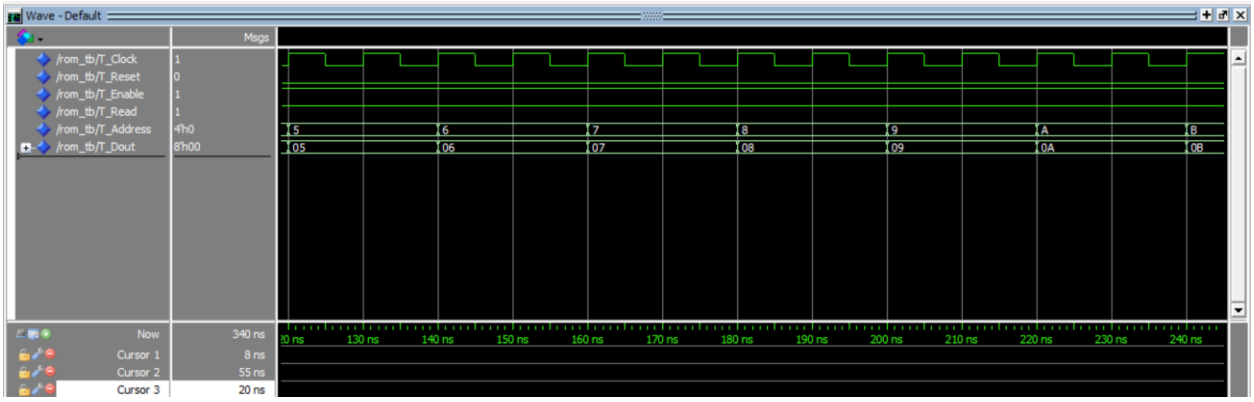
```
----------------------------------------------------------------
-- Test Bench for 16*8 ROM module
----------------------------------------------------------------

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;
```

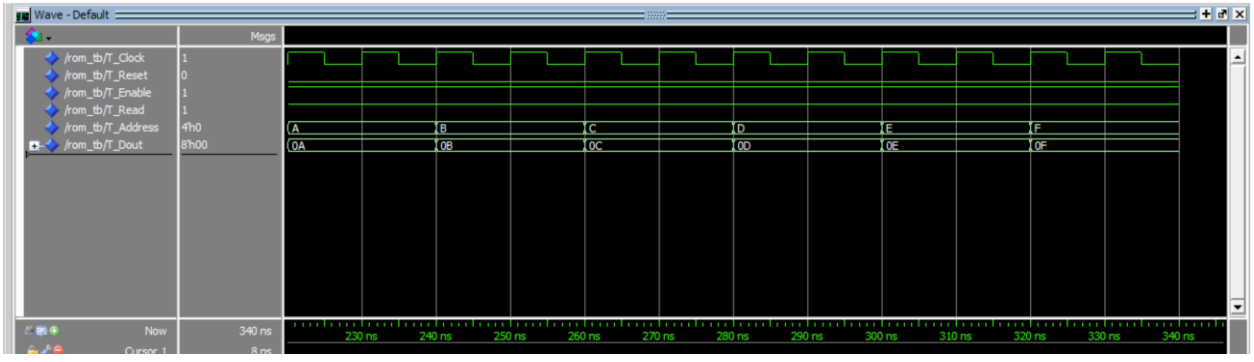
```
    process

    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:



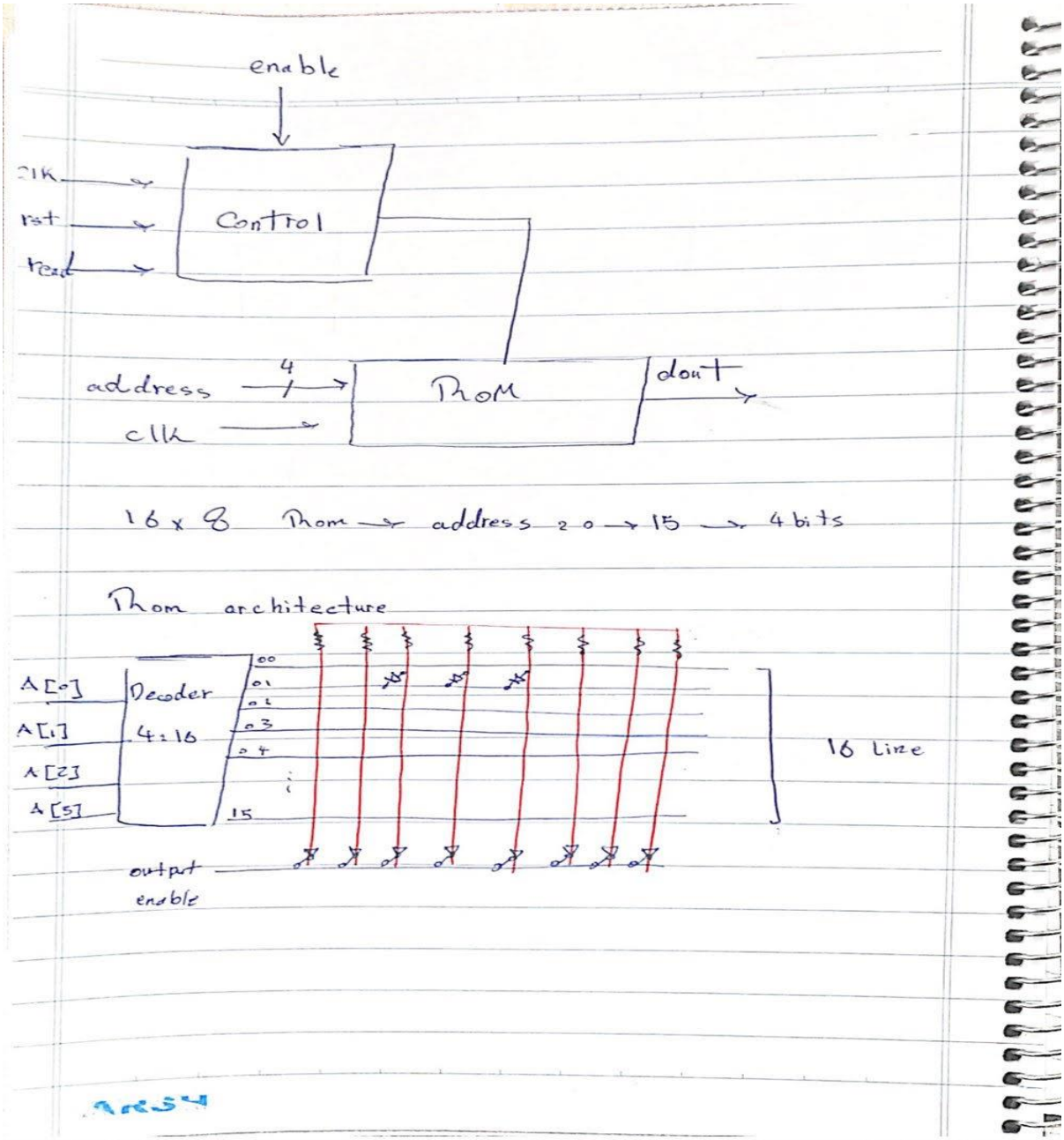




همانطور که در Wave خروجی مشاهده می‌کنید اعداد خانده شده از روی ROM از ۰ الی ۱۵ است و برنامه ما خروجی درست نشان داده است.

شماتیک ROM

همانطور که مشاهده می‌کنید واحد کنترلی اجازه خروج داده را از ROM می‌دهد و خود ROM از یک دیکودر ۴ به ۱۶ استفاده می‌کند چون 16x8 هست و ادرس ها ۴ 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:

```vhdl
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 ۱ می‌گذاریم و هر ۵ 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 ۰ 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 0Aو 0Fو 14و 19 در RAM داده های

05و 0Aو 0Fو 14و 19 را می‌نویسیم.

در حلقه دوم که ۵ بار اجرا می‌شود، به ترتیب از آدرس های 05و 0Aو 0Fو 14و 19 در RAM داده های

05و 0Aو 0Fو 14و 19 را می‌خوانیم.

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

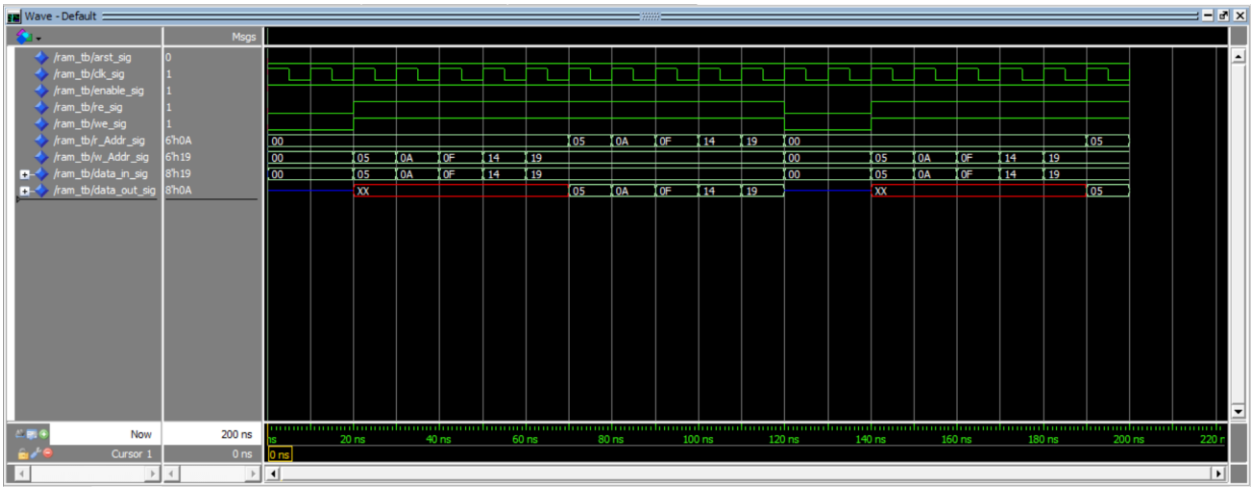
در ادامه کد تست بنچ و خروجی شبیه سازی را مشاهده می‌کنید.

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 dow
nto 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
    );
```

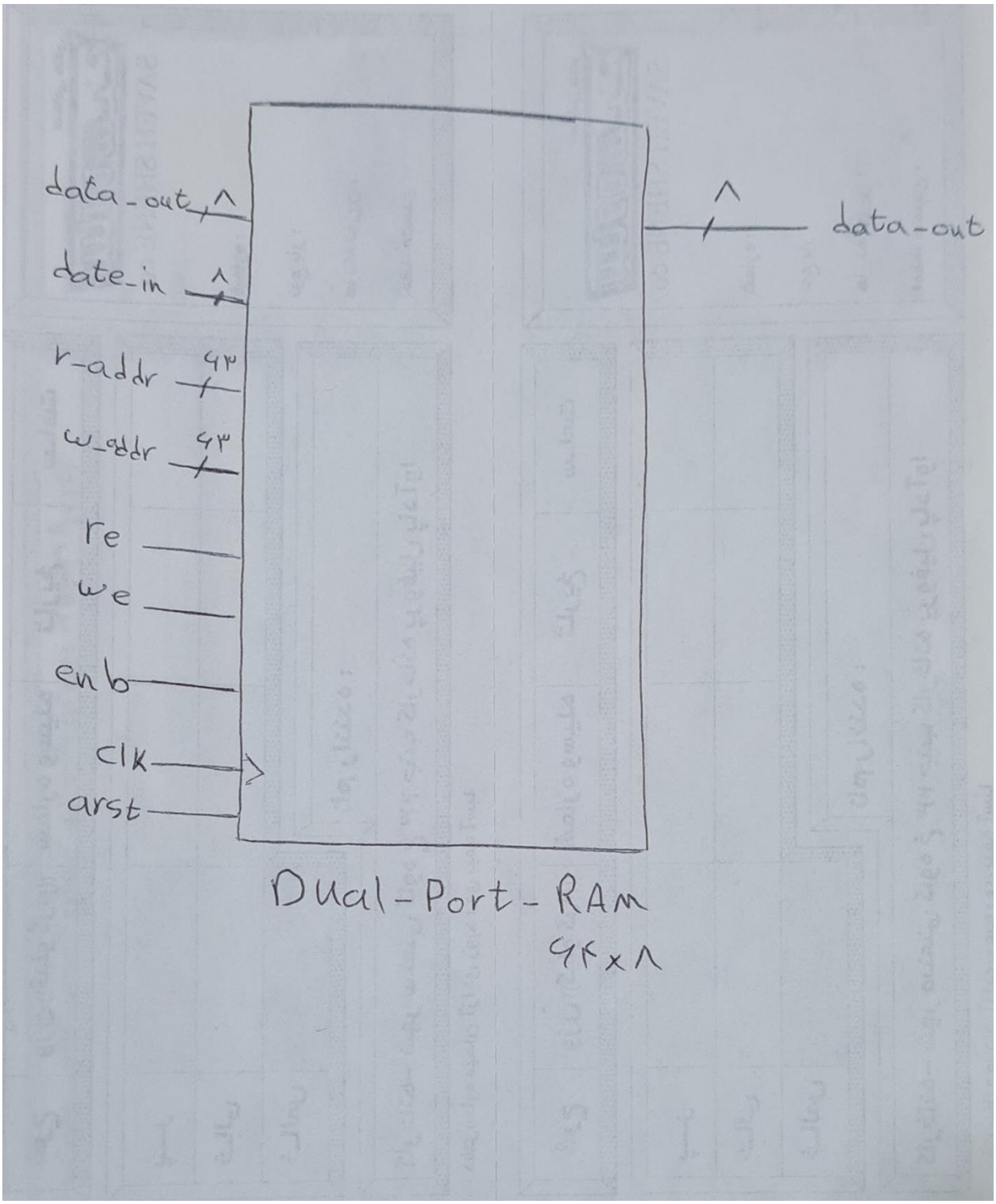
```
    clk_proccess : process
    begin
        clk_sig <= '1'; -- clock cycle 10 ns
        wait for 5 ns;
        clk_sig <= '0';
        wait for 5 ns;
    end process;

    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


data_out
date_in
r_addr
w_addr
re
we
enb
clk
arst
data_out
Dual-Port-RAM
۶۴x۸