LABORATOR 6

1. **Proiectarea memoriilor ROM**
2. **Proiectarea memoriilor RAM**

1. Considerații teoretice

Se poate spune că orice circuit secvențial conține o memorie de un anumit tip, aceasta datorită faptului că orice bistabil sau latch înmagazinează informație pe un bit. Totuşi când ne referim la o memorie ne gândim la o arie bi-dimensională, din care se accesează câte un rând de biți la un moment dat. În această secțiune a lucrării vor fi descrise câteva tipuri de memorii.

1.1 Memorii ROM

O memorie ROM (*Read Only Memory*) este un circuit combinațional cu *n* intrări şi *b* ieşiri, după cum se poate observa şi în figura 1. Intrările sunt numite *intrări de adresă* şi se notează de obicei A0,A1…An-1. Ieşirile se numesc *ieşiri de date* şi sunt notate D0, D1…Db-1. Se poate spune că o memorie ROM înmagazinează tabelul de adevăr al unei funcții logice cu n-intrări şi b-ieşiri.

Deoarece memoria ROM este un circuit combinațional este îndreptățită observația că nu mai poate fi încadrată în clasa memorii, specifică după cum am menționat circuitelor secvențiale. Totuşi memoria ROM poate fi gândită ca fiind un dispozitiv în care s-a înmagazinat informație la momentul în care a fost fabricată sau programată. Memoria ROM este o memorie *non-volatilă*, cea ce înseamnă că informația înmagazinată este păstrată chiar dacă memoria este deconectată de la sursa de energie.

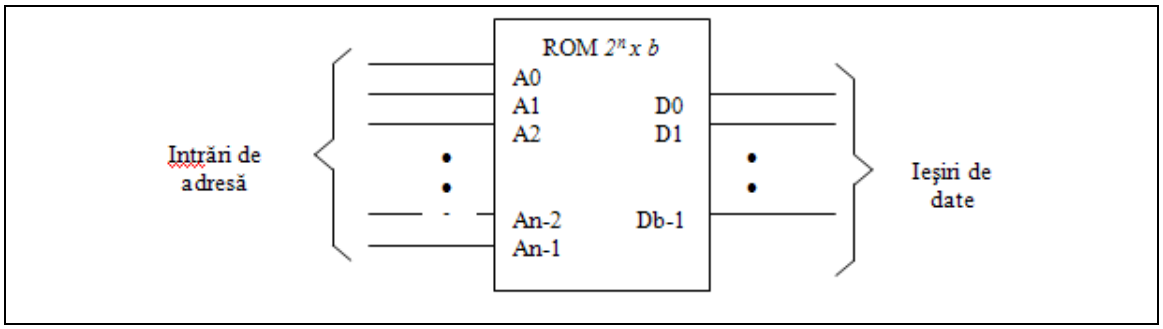

Figura 1 Structura de bază a unei memorii ROM $2^n xb$

1.2 Memorii RAM

Memoriile RAM fac parte din categoria *read/write memory* (RWM) cea ce înseamnă că informațiile vor putea fii atât citite din memorie cât şi scrise.

Memoria RAM (*random access memory*) permite citirea şi scrierea biţilor la adrese aleatoare. Aceste tipuri de memorii sunt memorii *volatile*, informaţia conţinută se pierde când memoria este deconectată de la sursa de energie. Memoriile RAM pot fi SRAM sau DRAM.

Într-o memorie SRAM (*static-RAM*), odată scris un cuvânt la o locaţie, va rămâne acolo atâta timp cât cipul este conectat la o sursă de energie cu condiţia ca la aceeaşi locaţie să nu fie scris un alt cuvânt. În memoriile DRAM (*dynamic-RAM*), datele stocate la fiecare locaţie trebuie reânprospătate periodic, prin citirea lor şi apoi rescriere, altfel se pierd.

Memorii SRAM

La fel ca şi memoriile ROM, RAM-urile au intrări de control, adresă şi ieşiri de date dar în plus au şi intrari de date. În figura 2 se pot vedea intrările şi ieşirile unui RAM static cu caracteristicile 2^n *x b-biţi* , astfel fiind definită capacitatea memoriei (numărul maxim de locaţii de memorie şi dimensiunea cuvintelor stocate).

Intrările de control sunt active pe nivel logic 0 şi au următoarele funcţii: - intrarea CS (*chip select*) activează cipul de memorie; - intrarea OE (*output enable*) permite transferul datelor de la o locaţie de memorie către ieşirea de date; - intrarea WE (*write enable*) selectează modul de lucru al memorie, datele sunt scrise în memorie (WE = 0) sau datele sunt citite din memorie (WE = 1).

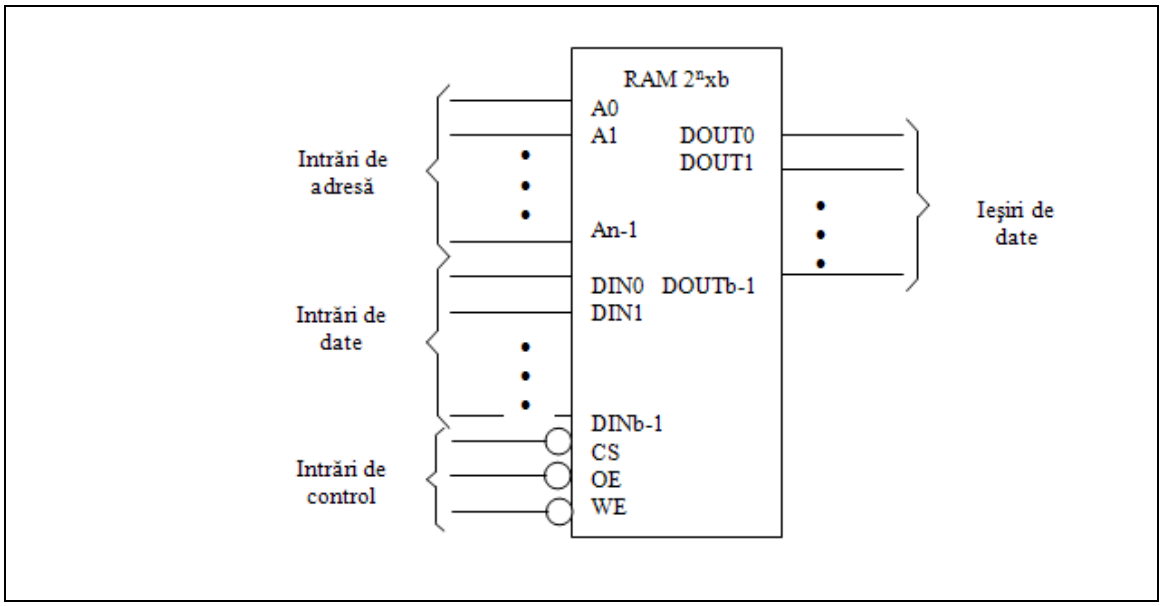

Figura 2 Structura de bază a unei memorii RAM $2^n xb$

Structura internă a memoriilor SRAM

Fiecare bit de memorie (sau celulă de memorie SRAM) are acelaşi comportament funcţional ca şi cel al circuitului din figura 3. Elementul de stocare al unei memorii SRAM este latch-ul de tip D.

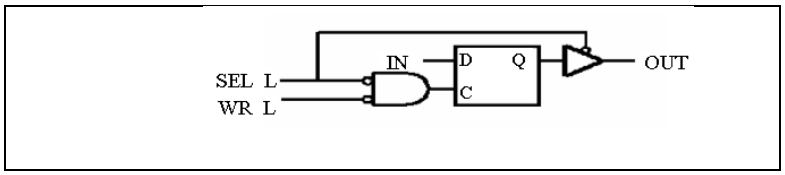

Figura 3 Structura celulei de memorie SRAM

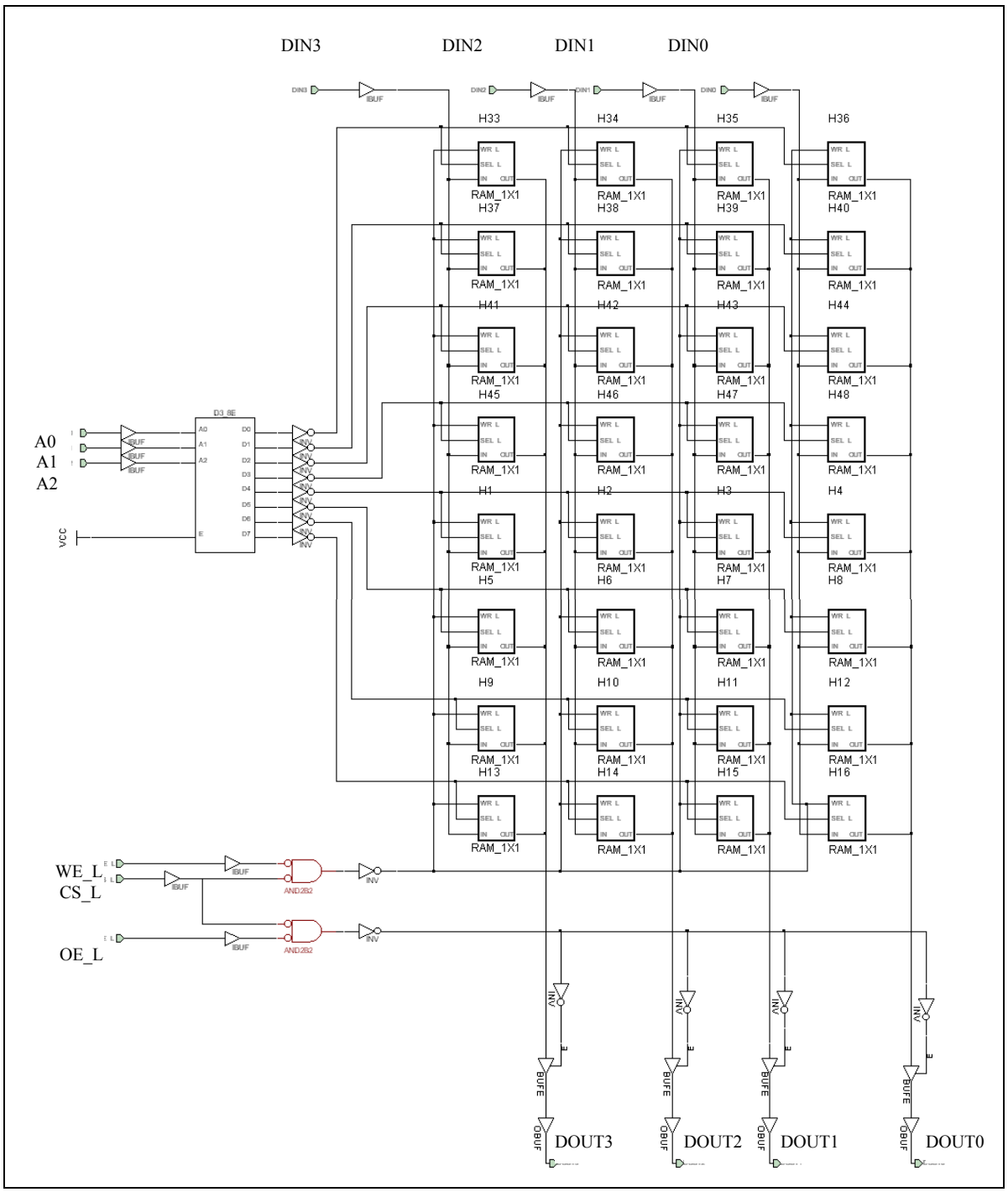

Figura 4 Structura internă a unei memorii RAM 8x4

Când intrarea SEL_L = 0, bitul stocat în latch este plasat la ieşirea celulei de memorie care este conectată la o linie de bit (vezi figura 4). Dacă SEL_L şi WR_L iau valoarea logică 0 simultan latch-ul va fi deschis permițându-se înscrierea unei noi valori. Mai multe astfel de celule de memorie, împreună cu logica de comandă corespunzătoare formează un RAM static. În figura 4 este prezentat o memorie SRAM 8x4. Un decodor conectat la liniile de adresă selectează un anumit rând de celule al memoriei în funcție de adresa indicată la intrarea decodorului.

2. Descrierea memoriilor în VHDL

2.1 Cod VHDL pentru memorie ROM

Deoarece conținutul unei memorii ROM se defineşte o singură dată pentru modelarea în VHDL a memoriei ROM putem definii o arie de valori constante corespunzătoare datelor stocate în memorie.

Prima versiune de memorie ROM - În codul din figura 5 apar definite două tipuri noi (*types*) de date, *integer* şi *array*. Tipul *integer* defineşte numerele întregi cuprinse între $-2^{31}+1$şi $+2^{31}-1$. Tipul *array* defineşte o arie de elemente de un anumit tip (în cazul nostru *std_logic_vector*). Fiecare element are un index în acest caz indexul fiecărui element al ariei este dat de numărul întreg corespunzător adresei de intrare. În linia de cod, corespunzătoare definirii ariei, cuvintele format bold sunt cuvintele cheie folosite la definirea acesteia.

```
library ieee;
use ieee.std_logic_1164.all;
entity rom16x8 is
  port (address : in std_logic_vector (3 downto 0);
        data : out std_logic_vector (7 downto 0));
end entity rom16x8;

architecture RTL of rom16x8 is
  type rom_array is array (0 to 15) of
std_logic_vector(7 downto 0);
  constant rom : rom_array := (
                        "00000000", --00
                        "00010001", --11
                        "00100010", --22
                        "01000011", --43
                        "10000100", --84
                        "01000101", --45
                        "00100110", --26
                        "00010111", --17
                       "00001000", --08
                       "00011001", --19
                       "00101010", --2A
                       "01001011", --4B
                       "10001100", --8C
                       "01001101", --4D
                       "00101110", --2E
```

```
                        "00011111");  --1F
begin
  data <= rom(conv_integer(address));
end architecture RTL;
```

Figura 5. Cod VHDL pentru memoria ROM 16x8

1. Sintetizaţi şi simulaţi codul din figura 5.
2. Definiţi o listă de constrângeri (fişier UCF) astfel încât intrările (4 de adrese) să fie asignate comutatoarelor, iar ieşirile de date (8 biţi) LED-urilor de pe placa DIO4.
3. Generaţi fişierul de implementare pentru placa Digilab2SB (circuit FPGA XC2S200E-200 din familia Spartan 2E) şi testaţi memoria RAM în hardware.

A doua versiune de memorie ROM - În Figura 6 este prezentat un alt exemplu de modelare a unei memorii de tip ROM în VHDL. De această dată definirea se face într-un pachet separat de arhitectura modulului ROM. Dispozitivul este definit ca un tablou de constante, numit ROM. Fiecare linie din tablou defineşte conţinutul unei adrese din ROM.

Numărul de locaţii de memorare şi numărul de biţi (dimensiunea) cuvântului pot fi schimbate uşor. Subtipul ROMRANGE specifică faptul că ROM conţine locaţiile de memorare 0,7. Constanta ROMWIDTH specifică dimensiunea cuvântului de 5 biţi. După ce a fost definit ROM, se pot citi valorile ori de câte ori dorim.

```
package ROMS is -- declarare un ROM 5 x 8  numit ROM constant
 ROM_WIDTH:  INTEGER := 5;
    subtype ROM_WORD is BIT_VECTOR  (1 to ROM_WIDTH);
    subtype ROM_RANGE is INTEGER range 0 to 7;
    type ROM_TABLE is array  (0 to 7)  of ROM_WORD;
  constant ROM: ROM_TABLE := ROM_TABLE'(      ROM_WORD'("10101"), --
continutul ROM
ROM_WORD' ("10000")
ROM_WORD'("11111")
ROM_WORD'("11111")
ROM_WORD' ("10000")
ROM_WORD' ("10101")
ROM_WORD'("11111")
ROM_WORD'("11111"));
end package;
--entitate care foloseşte ROM
use work.ROMS.all;
entity ROM_5x8 is
   port   (ADDR:  in ROM_RANGE;    DATA:
   out ROM_WORD);
end entity;
architecture BEHAVIOR of ROM_5x8 is
begin
    DATA <= ROM(ADDR);    -- Citire din ROM
end architecture;
```

Figura 6. Al doilea model VHDL pentru o memorie de tip ROM.

Generator de forme de undă modelat ca şi memorie ROM - În continuare, se va folosi dispozitivul ROM din aplicația anterioară pentru a implementa un generator de funcții (generator de forme de unda). Presupunem că se doreşte obținerea formelor de undă din Figura 7.

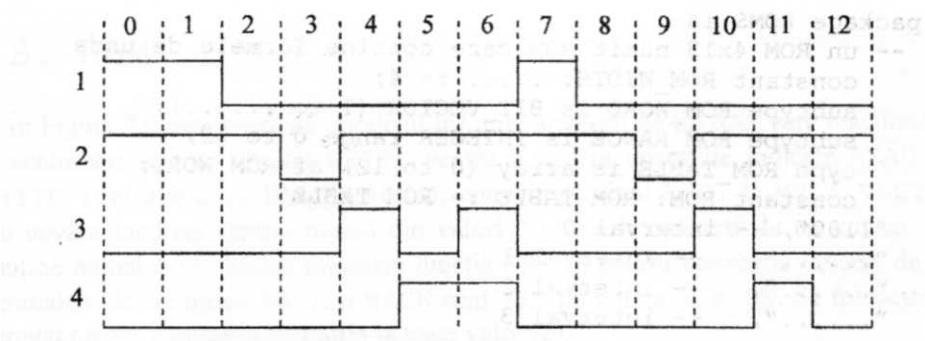

Figura. 7. Forme de undă generate de un generator de funcții

Pentru obținerea modelului avem în vedere următoarele:

1. Mai întâi se declară un ROM ale cărui cuvinte să aibă dimensiunea suficientă pentru a stoca semnalele de ieşire (4 biți) şi o capacitate suficient de mare pentru a reține toate momentele de timp (de la 0 la 12, aceasta înseamnă 13).
2. Se defineşte ROM astfel încât fiecare pas în timp este reprezentat de o intrare în ROM.
3. Se creează un numărător care ciclează paşii în timp (adresele ROM), generând formele de undă la fiecare pas.

În Figura 8 se prezintă modelul acestui generator de forme de undă.

```
package ROMS is -- un ROM 4x13 numit ROM care contine formele de unda
    constant ROM_WIDTH:   .. := 4;
    subtype ROM_WORD is BIT_VECTOR (1 to ......);
  subtype ROM_RANGE is INTEGER range 0 to 12;
  type ROM_TABLE is array  (0 to 12)  of ROM_WORD;
  constant ROM: ROM_TABLE := ROM_TABLE'
  ( "1100", -- interval 0
    "...",     -- interval 1
  ".....",     -- interval 2
               -- interval 3

  ".....",      -- interval 12
end package;

use work.ROMS.all;    -- entitate care foloseste ROM

entity WAVEFORM is
    port  (CLOCK:  in BIT;
           RESET:  in BOOLEAN;
           WAVES:  out ROM WORD);
end entity;

architecture BEHAVIOR of WAVEFOR is
    signal STEP:  ROM_RANGE;
```

```
    begin
    TIMESTEP_COUNTER: process
    begin
 wait until CLOCK'event and CLOCK = '1';
   if RESET then     -- detectare RESET
     STEP <= ROM_RANGE'low;
     elsif STEP = ROM_RANGE'high then;    -- Sfarsit?
        STEP <=   ; -- Retine ultima valoare
       --STEP <=  ;    unda continua
      else
      STEP <= STEP + 1;
   end process;
    WAVES <= .....;
end architecture;
```

Figura 8. Modelul VHDL al unui generator de forme de undă

1. Completați programul din Figura 8 compilați-1 şi simulați funcționarea circuitului.
2. Elaborați un raport scris care să conțină modelul VHDL complet (comentat) şi formele de undă care demonstrează funcționarea corectă a circuitului în toate cazurile posibile.

2.2 Cod VHDL pentru memorie RAM

O memorie RAM static poate fi modelată în mod asemănător memoriei ROM. Semnalul corespunzător intrării de date va fi declarat de mod *inout* deoarece datele trebuie să poată fi scrise în RAM (*in*) şi citite la ieşirea RAM-ului (*out*), deci în acest caz busul de date va fi comun pentru ambele operații.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RAM16x8 is
  port (Address: in std_logic_vector (3 downto 0);
          --   Address: in  INTEGER;
        Data: inout std_logic_vector(7 downto 0);
        CLK, CS, WE, OE: in std_ulogic);
end entity RAM16x8;
architecture behavioral of RAM16x8 is
signal data_In : std_logic_vector(7 downto 0);
signal data_out : std_logic_vector(7 downto 0);
begin
p0: process (CLk) is
      type ram_array is array (0 to 15) of
            std_logic_vector(7 downto 0);
```

```
      variable mem: ram_array;
      variable address_v : integer;

    begin

   if clk'event and clk ='1' then
      address_v := Conv_integer (address);
     if cs = '0' then
          if oe='0'and we = '1' then
          Data_out <= mem(address_v);
          elsif oe='1'and we = '0' then
          mem(address_v) := Data_in;
          else null;
          end if;
     end if;
   end if;
    end process p0;

p1:process (CS,OE, WE,  Data, data_in, data_out)
     begin
     if cs = '0' then
          if oe='0'and we = '1' then
               data <= data_out;
          elsif oe='1'and we = '0' then
               data <= (others => 'Z');
               data_in <= data;
          else
                    data <= (others => 'Z');
          end if;
     end if;
     end process p1;
end architecture behavioral;
```

Figura 9. Cod VHDL pentru memoria SRAM 16x8

În figura 9 este prezentat codul VHDL al unei memorii SRAM 16x8. Semnalele de control definite în acest exemplu sunt CS, OE, WE toate cele trei semnale sunt active pe ‘0’ logic. Ultimelor două nu li se poate atribui simultan valoarea logică ‘0’ , datele nu pot fi decât citite sau scrise în memorie. Dacă nici unul dintre cele două semnale nu este activ ieşirea va rămâne în starea de înaltă impedanţă (‘Z’).

Din blocul proces se observă că memoria RAM este modelată comportamental ca şi un tablou, dar spre deosebire de memoria ROM aceast tablou este definit ca variabilă în interiorul procesului. Valoarea implicită a unui semnal sau variabile este valoarea cea mai din stânga

corespunzătoare tipului semnalului sau variabilei. În cazul nostru valoarea cea mai din stânga corespunzătoare tipului *std_logic* este ‘U’, astfel că întreaga arie corespunzătoare memoriei va fi inițializată cu valoarea ‘U’.

4. Sintetizați și simulați codul din figura 9.
5. Definiți o listă de constrângeri (fișier UCF) să fie asignate comutatoarelor, butoanelor și LED-urilor de pe placa DIO4.
6. Generați fișierul de implementare pentru placa Digilab2SB (circuit FPGA XC2S200E-200 din familia Spartan 2E) și testați memoria RAM în hardware.