

MEMORII

În cadrul acestui laborator vor fi discutate două tipuri importante de memorii:

- Memorie cu un singur port de I/O
- Memorie cu două porturi de I/O

Memoria cu un singur port de I/O

Conform figurii de mai jos, acest tip de memorie are un singur port de intrare pentru date și un singur port de ieșire a datelor. Aceste porturi au același număr de biți (4, 8, 16, 32, etc.). Operațiile permise pe aceste tipuri de memorii sunt: scriere și citire.

Semnificația restului de semnale din Figura 1 sunt:

- clk – semnal de ceas;
- we – semnal de activare operație de scriere (Write Enable);
- addr – semnal care specifică adresa de scriere/citire a unei valori;
- d_in – port prin care datele din exterior sunt scrise în memorie;
- d_out – port prin care datele memorate sunt oferite mediului extern.

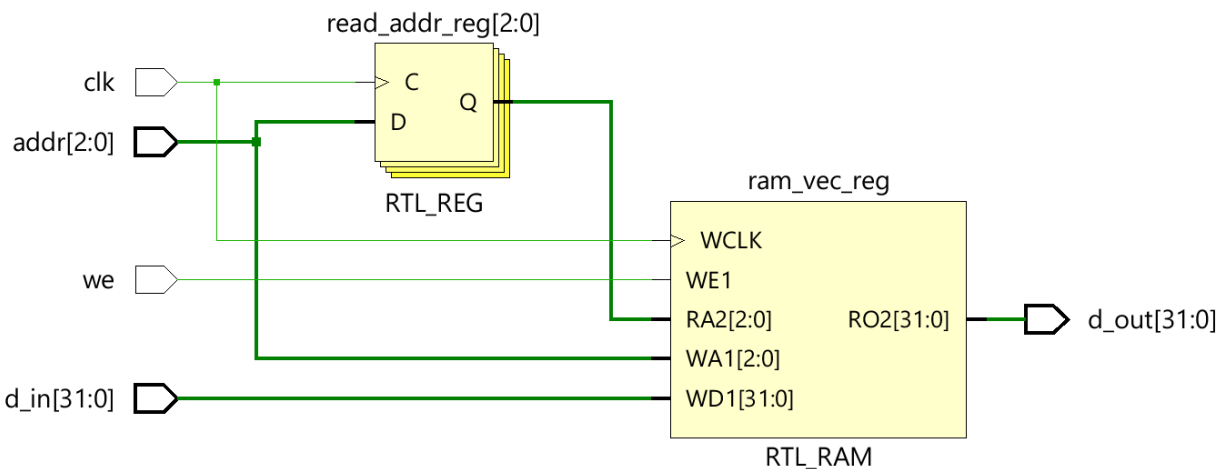


Figura 1. Memorie cu un singur port de I/O

Se observă că semnalul este pe 3 biți de unde rezultă că putem avea maximum 2^3 locații de memorie, adică în memorie pot fi memorate maximum 8 valori. Totodată, variabila **d_in** este reprezentată pe 32 de biți, ceea ce implică faptul că fiecare valoare memorată va fi reprezentată pe 32 de biți. Se observă că variabila **d_out** are exact același număr de biți ca și variabila **d_in**.

În concluzie, memoria prezentată în Figura 1, este practic un vector (sau o matrice cu o singură coloană) de dimensiune 32 x 8.

Din punct de vedere al funcționării, orice memorie suportă operațiile de scriere în memorie și respectiv citire din memorie. În cazul acestui tip de memorie este permisă doar o operație la un moment de timp dat. Nu putem efectua citire din memorie și scriere în memorie în același moment de timp deoarece avem doar un singur port de I/O.

Operația de scriere în memorie presupune realizarea următorilor pași:

- activare semnal **we** (**we = 1**)
- apariția frontului pozitiv de ceas
- memorarea valorii aflată pe portul **d_in** în memorie la adresa indicată de **addr**.

Operația de citire din memorie presupune realizarea următorilor pași:

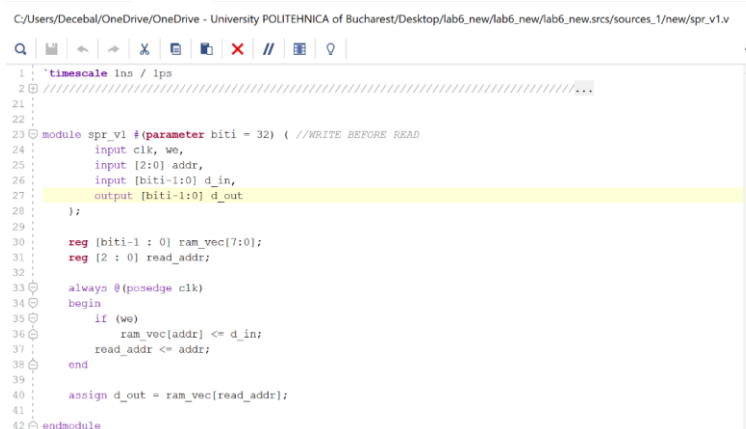
- activare semnal **we** (**we = 0**)
- apariția frontului pozitiv de ceas
- citirea din memorie a valorii aflată la adresa **addr** și transmiterea ei pe portul **d_out** al memoriei.

În funcție de implementarea aleasă când dorim să suprascriem o valoare de la o anumită adresă și apoi să afișăm valoarea de la respectiva adresă de memorie putem avea cazurile:

- **WRITE BEFORE READ** – dorim să suprascriem valoarea (presupunem ca valoarea este 10) aflată în memorie la locația 5 cu valoarea 200. Când vom dori să vizualizăm, valoarea aflată în memorie la adresa 5 vom avea valoarea 200.
- **READ BEFORE WRITE** - dorim să suprascriem valoarea (presupunem ca valoarea este 10) aflată în memorie la locația 5 cu valoarea 200. Când vom dori să vizualizăm, valoarea aflată în memorie la adresa 5 vom vizualiza întâi valoarea 10 și la următorul ciclu de ceas vom vizualiza valoarea 200.

Este foarte importantă modalitatea de implementare a unei memorii în FPGA deoarece vom dori ca în următoarele laboratoare să facem procesare de imagini.

În Figura 2 este prezentată implementarea în Verilog a unei memorii cu un singur port de I/O varianta **Write Before Read**.



```
1: timescale 1ns / 1ps
2: //////////////////////////////////////////////////...
21:
22:
23: module spr_v1 #(parameter biti = 32) ( //WRITE BEFORE READ
24:     input clk, we,
25:     input [2:0] addr,
26:     input [biti-1:0] d_in,
27:     output [biti-1:0] d_out
28: );
29:
30:     reg [biti-1 : 0] ram_vec[7:0];
31:     reg [2 : 0] read_addr;
32:
33:     always @(posedge clk)
34:     begin
35:         if (we)
36:             ram_vec[addr] <= d_in;
37:         read_addr <= addr;
38:     end
39:
40:     assign d_out = ram_vec[read_addr];
41:
42: endmodule
```

Figura 2. Implementarea în Verilog a unei memorii cu un singur port

În figura 3 se prezintă fișierul de simulare pentru memoria cu un singur port de I/O varianta **Write Before Read**.

```
module sim_spr_v1 #(parameter biti = 32); //WRITE BEFORE READ
    reg clk, we;
    reg [2:0] addr;
    reg [biti-1 : 0] d_in;
    wire [biti-1 : 0] d_out;
    integer i; // contor clk

    spr_v1 #(biti) inst1(clk, we, addr, d_in, d_out);

    always #10 clk = ~clk;

    initial begin
        clk = 0; we = 0;
        d_in = {biti{1'bx}}; //d_in = 32'bx
        addr = 3'bx;
        //Test de scriere
        for (i=0; i<8; i=i+1)
            begin
                @(negedge clk);
                #5 addr = i; we = 1; d_in = 15 - i;
            end
        //Test de citire
        @(negedge clk);
        for (i=0; i<8; i=i+1)
            begin
                @(negedge clk);
                #5 addr = i; we = 0;
            end
        //Scriere si citire la aceeași adresa în același ciclu de ceas
        @(negedge clk);
        #5 addr = 5; we = 1; d_in = 99;
        @(negedge clk);
        addr = 3'bx; we = 1'bx; d_in = {biti{1'bx}};
        @(negedge clk);
        $finish;
    end
endmodule
```

Figura 3. Fișierul Verilog pentru simularea unei memorii cu un singur port I/O varianta **Write Before Read**.

Rezultatele simulării pot fi observate în Figura 4.

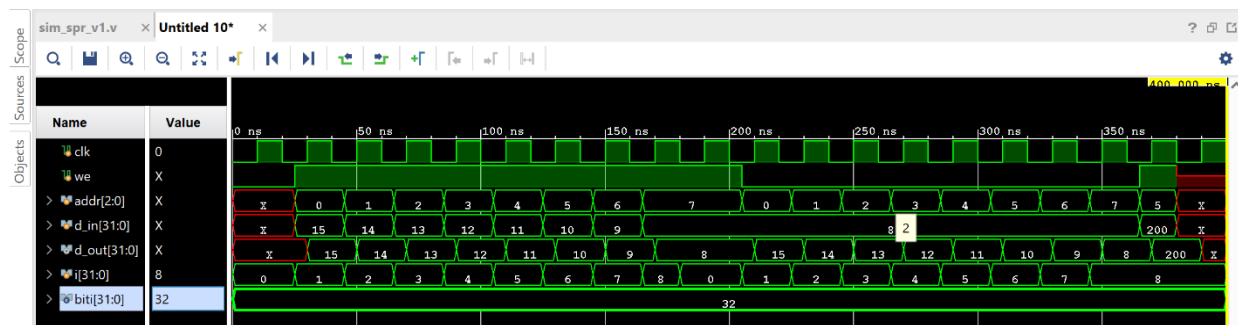


Figura 4. Rezultatele simulării. După cum se poate observa, se afișează valoarea 200 direct, fără a se mai afișa valoarea inițială din memorie aflată la adresa de memorie 5.

Având în vedere modificările minore care trebuie aduse codului din Figura 2 (modulul pentru simulare din Figura 3 poate rămâne nemodificat) pentru a obține o memorie cu un sigur port de I/O în variantă **Read Before Write**, el va fi realizat în cadrul laboratorului. De observat că la simulare înainte de apariția valorii 200, se va fișă valoarea existentă în memorie la locația de memorie 5.

Memoria cu două porturi de I/O

O astfel de memorie este prezentată în Figura 5. În fapt acest tip de memorie dublează I/O-urile și semnalele memoriei precedente (cu excepția semnalului de ceas). Această memorie va permite realizarea de operații I/O în paralel, dar evident că nu la sau de la aceeași locație de memorie. Implementarea unei astfel de memorii presupune în fapt doar realizarea a două cicluri **always** care funcționează în paralel, câte unul pentru fiecare port al memoriei.

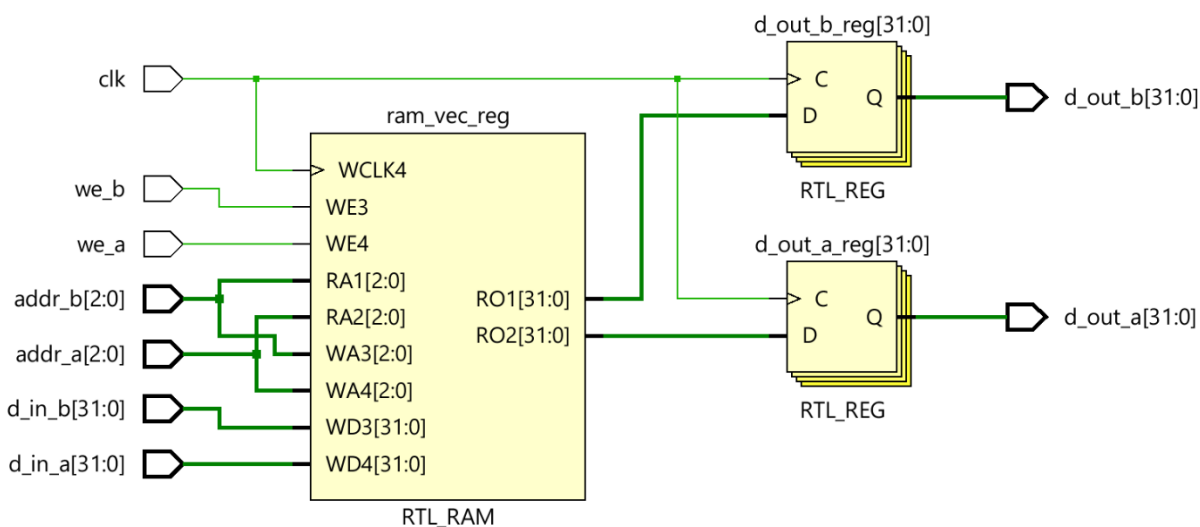


Figura 5. Realizarea unei memorii cu două porturi de I/O

Având în vedere faptul că implementarea Verilog a acestei memorii are un grad scăzut de dificultate, ea se va efectua ca și exercițiu în timpul orelor de laborator.

În cele ce urmează, se va prezenta codul Verilog necesar realizării simulării acestei memorii.

```
module sim_dpr #(parameter biti = 32);
    reg clk, we_a, we_b;
    reg [2:0] addr_a, addr_b;
    reg [biti-1:0] d_in_a, d_in_b;
    wire [biti-1:0] d_out_a, d_out_b;
    integer i;
    dpr #(biti) inst1(clk, we_a, we_b, addr_a, addr_b, d_in_a, d_in_b, d_out_a, d_out_b);
    always #10 clk = ~clk;
    initial begin
        clk=0; we_a=0; we_b=0; d_in_a = {biti{1'bx}}; d_in_b = {biti{1'bx}};
        addr_a = {3{1'bx}}; addr_b = {3{1'bx}};

        //Test scriere
        for(i=0; i<4; i=i+1)
            begin
                @(negedge clk);
                #5 addr_a=i; we_a=1; d_in_a=15-i; addr_b=i+4; we_b=1; d_in_b=15-i+4;
            end

        //Test citire
        @(negedge clk);
        for(i=0; i<4; i=i+1)
            begin
                @(negedge clk);
                #5 addr_a=i; addr_b=i+4; we_a=0; we_b=0;
                d_in_a={biti{1'bx}}; d_in_b={biti{1'bx}};
            end

        //Scriere și citire de la aceeași adresa la același moment de timp
        @(negedge clk);
        #5 addr_a=5; we_a=1; d_in_a=200;
```

```

//Terminare
@(negedge clk);

@(negedge clk);

$finish;

end

endmodule

```

Rezultatele simulării sunt prezentate în figura 6.

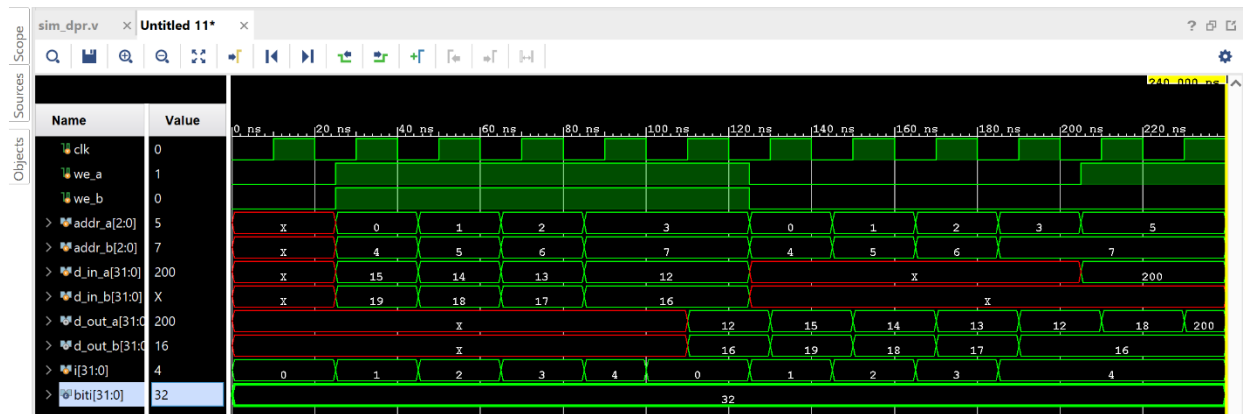


Figura 6. Rezultatele simulării unei memorii cu două porturi de I/O.