instanţiază memoria care se doreşte a fi testată şi se descriu task-urile care implementează operaţiile unei memorii:

- Iniţializarea memoriei se realizează cu comanda:
  $readmemb (nume_fisier, nume_memorie)
  Formatul fişierului de intrare este prezentat în continuare:

```
0000000000000000  //conţinutul de la locaţia 0
0000000000000001  //conţinutul de la locaţia 1
0000000000000010  //conţinutul de la locaţia 2
0000000000000011  //conţinutul de la locaţia 3
0000000000000100  //conţinutul de la locaţia 4
0000000000000101  //conţinutul de la locaţia 5
```

- task-ul de citire este prezentat în continuare:

```
task Read;
input [7:0] addr_task;
begin
$display("time is %d, operatie de citire", $time);
// Adresa de unde se face citirea
addr <= addr_task;

// Semnalele de control setate pentru operaţia de citire
we <= 1'b0;
oe <= 1'b1;
ce <= 1'b1;
end
endtask
```

  Apelul task-ului de citire este de forma:
  **Read ( <adresă de citire>)**

- task-ul de scriere este prezentat în continuare:

```
task Write;
input [7:0] addr_task;
input [15:0] di_task;
begin
$display("time is %d, operatie de scriere", $time);
// Adresa unde se face scrierea
addr <= addr_task;

// Valoarea care se înscrie la locaţia pointată de adresă
di <= di_task;

// Semnalele de control setate pentru operaţia de scriere
we <= 1'b1;
oe <= 1'b1;
ce <= 1'b1;
```

```
end
endtask
```

Apelul task-ului de scriere este de forma:
**Write ( <adresă de scriere> , <data de intrare>)**

- citirea continutului memoriei

```
task Dump;
input [0:100] file_name;
integer k;
integer file_id;
begin
  file_id = $fopen(file_name);
  $display("time is %d, Citire conţinut memorie", $time);
  for (k=0; k<256; k = k + 1) begin
    $fdisplay (file_id,"Continutul de la adresa %d este : %b",
               k, DUT.mem[k]);
    $display  ("Continutul de la adresa %d este : %b",
               k, DUT.mem[k]);
  end
  $fclose(file_id);
end
endtask
```

- În urma apelului task-ului Dump rezultă un fişier care conţine o copie a memoriei. Formatul fişierului de ieşire este prezentat în continuare:

```
Continutul de la adresa           0 este : 1111111111110000
Continutul de la adresa           1 este : 1111111100001111
Continutul de la adresa           2 este : 1111000011111111
Continutul de la adresa           3 este : 0000111111111111
```

- Un model Verilog care foloseşte task-urile prezentate mai sus este prezentat în continuare:

```
module test ();

reg clk;
reg [7:0] addr;
reg [15:0] di;
reg  ce;
reg we;
reg oe;
wire  [15:0]  dout;

//descrierea task-urilor
task Read;
...
endtask
task Write;
...
```

```
endtask
task Dump;
...
endtask

initial begin

//iniţializarea memoriei
$readmemb ("mem_content.dat", DUT.mem);
// valoarea iniţială a ceasului
clk <= 0;
end

// generator de ceas
always #5 clk <= ~clk;

initial begin
@(posedge clk)
Read (8'h00);
@(posedge clk)
Read (8'h01);
@(posedge clk)
Read (8'h02);
@(posedge clk)
Read (8'h03);
@(posedge clk)
Write(8'h00,16'hFFF0);
@(posedge clk)
Write(8'h01,16'hFF0F);
@(posedge clk)
Write(8'h02,16'hF0FF);
@(posedge clk)
Write(8'h03,16'h0FFF);
@(posedge clk)
@(posedge clk)
Dump("new_mem_content.dat");

#1 $stop;
end

// instanţa de memorie testată
single_port_mem DUT ( .clk(clk), .addr(addr), .di(di),
                      .ce(ce), .we(we), .oe(oe),
                      .dout(dout));

endmodule
```

Sinteza unei memorii. În mod uzual memoriile nu se sintetizează. În cazul în care se sintetizează ca un bloc logic obişnuit, memoria va fi implementată pe bistabile ceea ce nu este

convenabil datorită ariei ocupate. Astfel pentru simularea memoriei se va folosi modelul comportamental descris în Verilog iar la sinteză se va folosi un model de memorie dintr-o bibliotecă de tehnologie care are informaţii de timpi de propagare şi arie ocuptă. Modelul comportamental, este însoţit de directive de sintetizare prin care se ignoră descrirea comportamentală păstrâdu-se interfaţa. Un astfel de exemplu de directive este prezentat în continuare:

```
module single_port_mem (clk, addr, di, ce, we, oe, dout );
// declaraţia porturilor
input               clk;  // ceasul memoriei
input  [7:0]        addr; // bus de adrese
input  [15:0]       di;   // bus date de intrare
input               ce;   // semnal de selecţie a memoriei
input               we;   // semnal de validarea scrierii
input               oe;   // semnal de activarea ieşirii
output [15:0]       dout; // bus date de ieşire
reg [15:0]       dout;

//synopsys translate_off

<modelul comportamental ignorat la sintezã pastrându-se interfaţa >

//synopsys translate_on

endmodule
```

## 7.3 Modelul unei memorii cu două porturi

Memoria cu două porturi prezintă două intrări distincte pentru adrese. O adresă este adresa de citire şi alta este adresa de scriere. Structura şi semnificaţia porturilor unei astfel de memorii este prezentată în tabelul 7.2.

| *Port* | *Dir* | *Dimensiune* | *Semnificaţie* |
|---|---|---|---|
| clk | in | 1 | Ceasul memoriei activ pe frontul pozitiv |
| rd_addr | in | 8 | Adresa de citire |
| wr_addr | in | 8 | Adresa de scriere |
| di | in | 16 | Date de intrare în memorie |
| ce | in | 1 | Semnal care selectează memoria |
| we | in | 1 | Semnal de validarea scrierii |
| oe | in | 1 | Semnal de activarea ieşirilor |
| dout | out | 16 | Datele de ieşire din memorie |

Tabelul 7.2: Semnificaţiile porturilor unei memorii cu două porturi.

Un model de memorie cu două porturi de adrese şi citire asincronă este prezentat în continuare:

```
module dual_port_mem  (clk, rd_addr, wr_addr, di, ce, we, oe, dout );
// declaraţia porturilor
input               clk;        // ceasul memoriei
input  [7:0]        rd_addr;    // bus de adrese de citire
input  [7:0]        wr_addr;    // bus de adrese de scriere
input  [15:0]       di;         // bus date de intrare
input               ce;         // semnal de selecţie a chipului de memorie
input               we;         // semnal de validarea scrierii
input               oe;         // semnal de activarea ieşirii
output [15:0]       dout;       // bus date de ieşire
reg [15:0]       dout;

// declaraţia de memorie
reg [15:0] mem [0:255];         // memorie de 256 de locaţii a câte 16 biţi

// declaraţia de memorie
reg [15:0] mem [0:255];
reg [15:0]       dout;

always @(posedge clk) begin
  if(ce)
    if(we)
      mem[wr_addr] <= di;        // scriere în memorie
end
// citire asincronă a memoriei
assign dout = (oe) ? mem[rd_addr] : 16'bz ;

endmodule
```

Testarea şi sinteza este asemănătoare cu cea de la o memorie cu un singur port. Task-urile sunt acelaşi cu modificări minore având în vedere noua interfaţă.

## 7.4 Modelarea unei memorii FIFO

În sistemele digitale avansate memoriile de tip FIFO sunt des întâlnite. În continuare este prezentat un model de stivă FIFO cu un singur ceas, în care toate operatiile sunt sincrone pe ceas. Modulul are un semnal de reset asincron.

```
module fifo (clk, reset, di,push, pop, dout, full, empty);

//declaraţia porturilor
input clk;
input reset;
input [15:0] di;
input push;
input pop;
// ieşirile din fifo
output [15:0] dout;
reg  [15:0] dout;
output full;
reg full;
output empty;
reg empty;

// declaraţia de memorie
reg [15:0] mem [0:7]; // 8 locaţii pe 8 biţi
// număratoare fifo
reg [2:0] wr_addr; // adresa de scriere
reg [2:0] rd_addr; // adresa de citire
reg [3:0] count_item;

// reset asincron
always @(posedge clk or negedge reset)
  if(!reset)
    begin
      wr_addr  <= 0;
      rd_addr  <= 0;
      count_item <= 0;
      dout  <= 0;
    end
  else
    begin
      if(push & !full) //operaţia de push
        begin
          mem[wr_addr]  <= di;
          wr_addr  <= wr_addr + 1;
          count_item  <= count_item + 1;
        end
      else
      if(pop & !empty) //operaţia de pop
        begin
          dout  <= mem[rd_addr];
          rd_addr <= rd_addr + 1;
          count_item <= count_item - 1;
        end
    end
```

```
//implementarea indicatorului full
always @(posedge clk or negedge reset)
begin
  if(!reset)
    full <= 0;
  else
    if (count_item  == 7 & push)
      full <= 1;
    else
      if (pop)
      full <= 0;
end

// implementarea indicatorului empty
always @(posedge clk or negedge reset)
begin
  if(!reset)
    empty <= 0;
  else
    if (count_item == 1 & pop)
      empty <= 1;
    else
      if (push)
      empty <= 0;
end
endmodule
```

Testarea unei stive este asemanătoare cu cea a unei memorii. În continuare sunt prezentate task-urile de push şi pop care sunt folosite la testarea unei astfel de stive:

```
//descrierea task-urilor
task Push;
input [15:0] data;
begin
$display("time is %d, operaţie de push", $time);
di <= data;
push <= 1;
pop <= 0;
end
endtask

task Pop;
begin
$display("time is %d, operaţie de pop", $time);
pop <= 1;
push  <= 0;
end
endtask
```

## 7.5 Teme de laborator

- Descrieţi în Verilog o memorie cu un singur port care realizează citirea sincronă pe frontul pozitiv de ceas, iar ieşirea să fie asincronă.
- Descrieţi în Verilog o memorie de 2kx32, având la dispoziţie blocuri de memorie de 2kx16.
- Descrieţi în Verilog o memorie de 2kx4 având la dispoziţie blocuri de memorie de 1kx4. Descrierea structurală este prezentată în figura 7.3

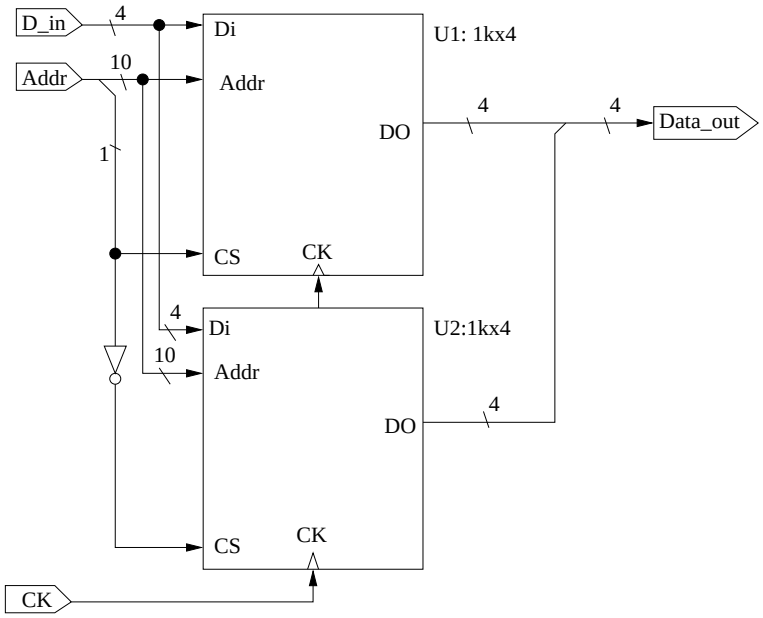


Figura 7.3: Memorie 2kx4 descriere structurală