REALIZAREA CIRCUITELOR LOGICE FOLOSIND LIMBAJUL VERILOG

Scopul lucrării

- înțelegerea diferitelor tipuri de modelare ale circuitelor logice în HDL;
- realizarea unor circuite logice combinaționale și secvențiale folosind limbaluj Verilog;
- simularea circuitelor logice și verificarea funcționalității acestora folosind programele VPP și GTKWAVE.

Materiale necesare

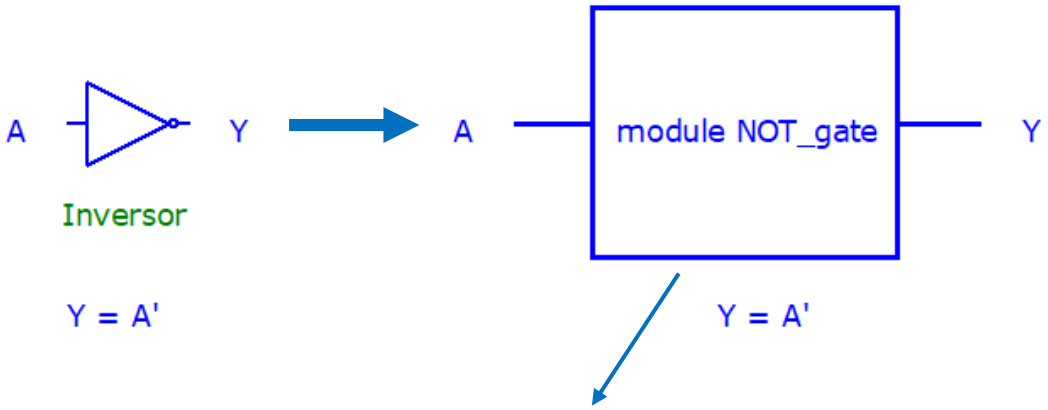
- un editor text (Visual Studio Code, Emacs etc.);
- pachetul Icarus Verilog (conține programele iverilog, vpp și gtkwave);
- (facultativ) pentru vizualizarea schemelor RTL: programul Intel Quartus Prime Lite Edition împreună cu un suport pentru dispozitiv (de exemplu Aria II), ambele disponibile pe bază de cont gratuit la adresa https://fpgasoftware.intel.com/?edition=lite.

Metodologia efectuării lucrării

1. Implementarea porților logice elementare în Verilog

1a. Implementarea unui inversor (NOT) folosind modelarea la nivel de poartă logică (gate level modeling)

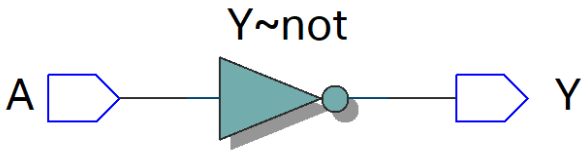
Un inversor (sau o poartă NOT) are o intrare (*A*), respectiv o ieșire (*Y*) și realizează operația logică *Y* = *A*'. În limbajul verilog putem crea un dispozitiv (modul) care conține o intrare, o ieșire și un inversor. Putem observa că în descrierea textuală a dispozitivului nostru am declarat un modul care are *Y* ca ieșire, *A* ca intrare și conține o poartă NOT ce are pinul *A* al dispozitivului legat la intrare, respectiv pinul *Y* legat la ieșire.



NOT_gate.v

```
1   module NOT_gate(output Y, input A);
2
3       not (Y, A);
4
5   endmodule
```

Folosind aplicația RTL Viewer din programul Intel Quartus Prime putem vizualiza schema RTL a modulului. Putem observa că modulul construit de noi are o intrare (*A*), o ieșire (*Y*) și conține un inversor.



Pentru a simula comportamentul dispozitivului va trebui să construim un "testbench" (banc de test) prin care să putem aplica valori la intrări și să monitorizăm variabilele de ieșire, un proces asemănător cu scrierea tabelului de adevăr al modulului. Fișierul testbench, NOT_gate_tb.v, conține modulul construit de noi (linia 1), NOT_gate.v, împreună cu un alt modul (linia 3), NOT_gate_tb, care conține un set de instrucțiuni pentru simulare:

- la început se declară variabila *A* ca fiind de tip *reg* (registru), iar variabila *Y* se declară ca fiind de tip *wire*; (liniile 5-6)
- în pasul următor se inițializează modulul NOT_gate împreună cu variabilele de intrare/ieșire; (linia 8)
- se definește o secvență prin care se specifică valorile variabilei de intrare *A* pentru 3 momente de timp (0s, 1s, respectiv 2s; #1 reprezintă un increment cu o unitate a timpului, în cazul nostru 1s); (liniile 10-14)
- la sfârșit se definește o secvență prin care se afișează pe ecran valorile timpului și ale variabilelor *A* și *Y* ($monitor()) împreună cu fișierul de date în care se vor salva rezultatele simulării (comenzile $dumpfile "NOT_gate_tb.vcd" și $dumpvars()); (liniile 16-21)

NOT_gate_tb.v

```
1   `include "NOT_gate.v"
2
3   module NOT_gate_tb;
4
5   reg A;
6   wire Y;
7
8   NOT_gate Instance0 (Y, A);
9
10  initial begin
11      A = 0;
12    #1 A = 1;
13    #1 A = 0;
14   end
15
16   initial begin
17      $monitor ("%t | A = %d | Y = %d", $time, A, Y);
18
19      $dumpfile("NOT_gate_tb.vcd");
20      $dumpvars();
21  end
22
23  endmodule
```

Simularea circuitului se face astfel:

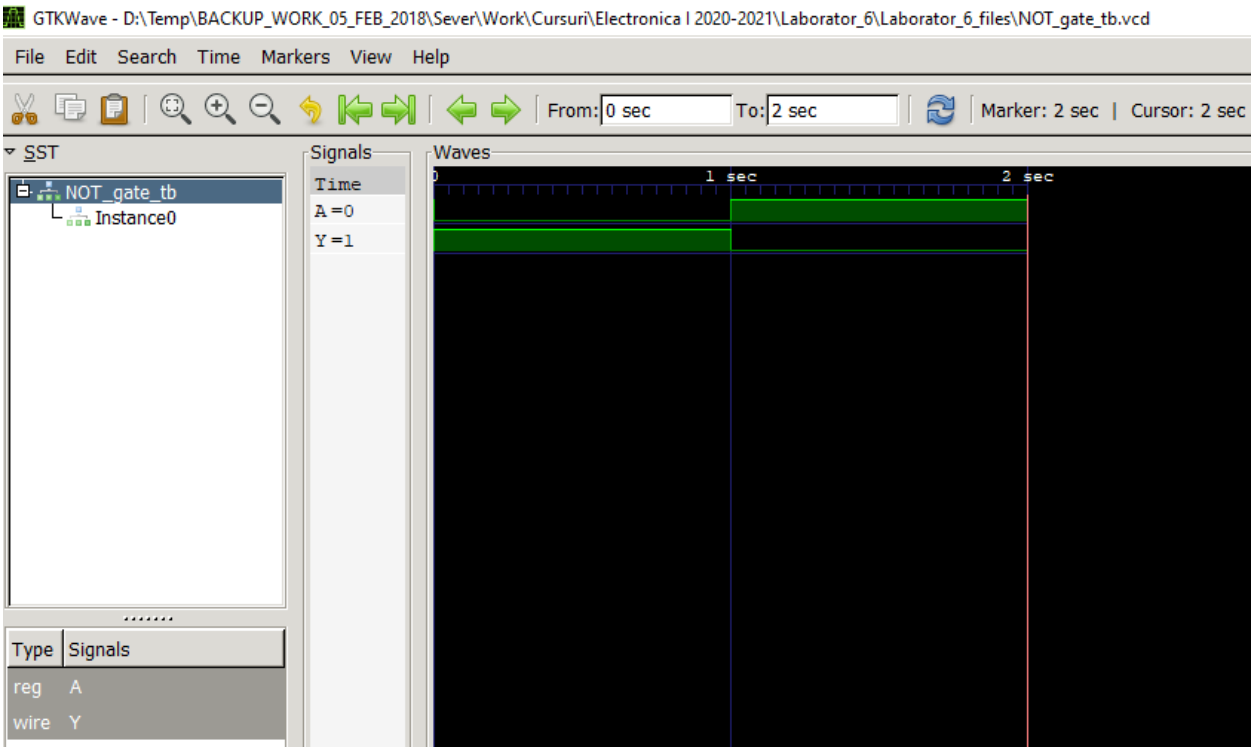
- pasul 1: compilarea fișierului testbench prin comanda de tip iverilog -o fisier_output.vvp fisier_input.v

```
iverilog -o NOT_gate_tb.vvp NOT_gate_tb.v
```

- pasul 2: execuția simulării prin comanda de tip vvp fisier_output.vvp

```
vvp NOT_gate_tb.vvp
VCD info: dumpfile NOT_gate_tb.vcd opened for output.
                    0 | A = 0 | Y = 1
                    1 | A = 1 | Y = 0
                    2 | A = 0 | Y = 1
```

- pasul 3: vizualizarea rezultatelor prin execuția programului *gtkwave* și încărcarea fișierului .vcd ce conține rezultatele simulării. Din analiza formelor de undă se poate observa că *Y = A'*.
- pasul 4: (facultativ) vizualizarea schemei RTL a dispozitivului folosind aplicația RTL Viewer;



1b. Implementarea unui inversor (NOT) folosind modelarea la nivel de funcție logică (data flow modeling)

Modelarea la nivel de poartă logică este fezabilă doar atunci când numărul de porți din circuit este mic. Dacă numărul de porți din circuit crește, atunci trebuie să trecem la un nivel de abstractizare superior. În această secțiune vom implementa poarta NOT pe baza funcției pe care aceasta o realizează. Vom crea un modul NOT_data cu *A* ca și intrare, respectiv *Y* ca și ieșire. Putem observa că în loc de inversor, în acest modul se specifică doar dependența lui *Y* de *A* (sau funcția logică realizată de acesta) folosind declarația assign. Un exemplu de cod folosind modelarea la nivel de funcție logică este dat mai jos.

NOT_data.v
```
1   module NOT_data(output Y, input A);
2
3       assign Y = ~A;
4
5   endmodule
```

- simulați modulul NOT_data folosind pașii descriși mai sus (un exemplu de testbench pentru modulul de față este prezentat mai jos);
- vizualizați semnalele A și Y folosind programul *gtkwave*;
- ce diferențe observați între cele două tipuri de modelare?

NOT_data_tb.v
```
1   `include "NOT_data.v"
2
3   module NOT_data_tb;
4
5   reg A;
6   wire Y;
7
8   NOT_data Instance0 (Y, A);
9
10  initial begin
11      A = 0;
12    #1 A = 1;
13    #1 A = 0;
14  end
15
16  initial begin
17      $monitor ("%t | A = %d | Y = %d", $time, A, Y);
18
19      $dumpfile("NOT_data_tb.vcd");
20      $dumpvars();
21  end
22
23  endmodule
```

1c. Implementarea unui inversor (NOT) folosind modelarea la nivel comportamental (behavioral modeling)

În cazul circuitelor complexe nu mai este fezabil să gândim în termeni de porți sau funcții logice. Circuitele pot fi construite sub formă de algoritm sau, altfel spus, într-un mod prin care putem descrie comportamentul acestora. Comportamentul porții NOT este cunoscut: valoarea ieșirii *Y* este 1 dacă *A* este 0, respectiv *Y* este 0 dacă *A* este 1. Acest comportament poate fi modelat textual printr-o declarație de tip `always @ (A)` ce conține o listă de expresii evaluate secvențial. (*A*) reprezintă lista de sensitivități (sensitivity / trigger list). Un exemplu de cod folosind modelarea la nivel comportamental este dat mai jos. Se poate observa că în modulul `NOT_behave` variabila de ieșire *Y* este declarată ca o variabila de tip *reg* datorită blocului `always @ (A)`.

NOT_behave.v

```
1   module NOT_behave(output reg Y, input A);
2
3   always @ (A) begin
4
5       if (A == 1) begin
6           Y = 0;
7       end
8       else if (A == 0) begin
9           Y = 1;
10      end
11  end
12
13  endmodule
```

- simulați modulul `NOT_data` folosind pașii descriși mai sus;
- vizualizați semnalele A și Y folosind programul *gtkwave*;
- ce diferențe observați între acest tip de moldelare și cele precedente?

1d. Implementarea unei porți SAU (OR)

- construiți module care să realizeze operația SAU folosind modelarea la nivel de poartă logică, modelarea la nivel de funcție logică și modelarea la nivel comportamental; exemplele de cod pentru dispozitive modelate la nivel de poartă logică respectiv la nivel comportamental sunt date mai jos;
- simulați comportamentul celor 3 dispozitive folosind pașii descriși anterior; un exemplu de testbench pentru un modul cu două intrări și o ieșire este prezentat mai jos;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

OR_gate.v

```
1   //poarta SAU - gate level
2
3   module OR_gate(output Y, input A, input B);
4
5   or(Y, A, B);
6
7   endmodule
```

OR_behave.v

```
1   //poarta SAU - behavioral level
2
3   module OR_behave(output reg Y, input A, B);
4
5   always @ (A or B) begin
6
7       if (A == 0 & B == 0) begin
8           Y = 0;
9       end
10      else
11          Y = 1;
12
13  end
14
15  endmodule
```

OR_behave_tb.v

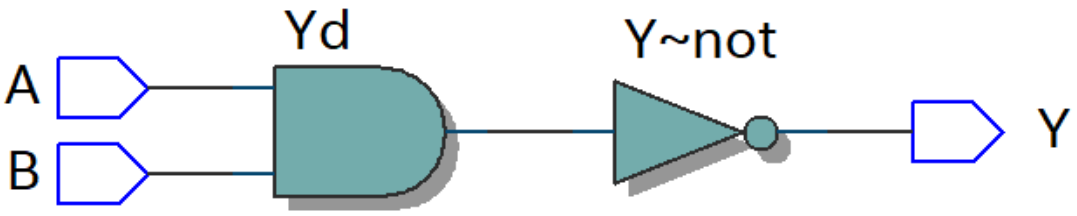
```
1   //testbench poarta SAU
2
3   `include "OR_behave.v"
4
5   module OR_behave_tb;
6
7   reg A;
8   reg B;
9   wire Y;
10
11  OR_behave Instance0 (Y, A, B);
12
13  initial begin
14      A = 0; B = 0;
15    #1 A = 1; B = 0;
16    #1 A = 0; B = 1;
17    #1 A = 1; B = 1;
18   end
19
20   initial begin
21      $monitor ("%t | A = %d | B = %d | Y = %d", $time, A, B, Y);
22
23      $dumpfile("OR_behave_tb.vcd");
24      $dumpvars();
25  end
26
27  endmodule
```

1e. Implementarea unei porți ȘI-NU (NAND)

- construiți module care să realizeze operația ȘI-NU folosind modelarea la nivel de poartă logică, modelarea la nivel de funcție logică și modelarea la nivel comportamental; un exemplu de cod pentru o poartă ȘI-NU modelată la nivel de poartă este dat mai jos; se poate observa că în cazul modelării la nivel de poartă logică vom folosi un fir *Yd* care conectează ieșirea porții ȘI de intrarea inversorului;
- simulați comportamentul celor 3 dispozitive folosind pașii descriși în secțiunile anterioare;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

NAND_gate.v

```
1   //poarta ȘI-NU - gate level
2
3   module NAND_gate(output Y, input A, input B);
4
5   wire Yd;
6
7   and (Yd, A, B);
8   not (Y, Yd);
9
10  endmodule
```



1e. Realizarea unor porți logice elementare folosind limbajul Verilog

- construiți module care să realizeze operațiile ȘI (AND), SAU-NU (NOR) respectiv SAU-exclusiv (XOR) folosind oricare dintre tipurile de modelare discutate mai sus;
- simulați comportamentul celor 3 dispozitive folosind pașii descriși anterior;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

2. Construcția unui multiplexor 2x1 folosind limbajul Verilog

- având la dispoziție tabelul de adevăr de mai jos, realizați un multiplexor de tip 2x1 folosind limbajul Verilog; tipul de modelare rămne la alegerea dumneavoastră;
- simulați comportamentul dispozitivului folosind pașii descriși anterior;
- (facultativ) vizualizați schema RTL a circuitului;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

S	B	A	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

3. Construcția unui bistabil JK folosind folosind limbajul Verilog

- descrieți bistabilul JK textual și salvați-l într-un fișier Verilog, de exemplu `JK_FF.v`; un exemplu de cod ce foloseste modelarea comportamentală este dat mai jos;
- expresiile din blocul `always @ (posedge CLK)` vor fi evaluate în paralel la fiecare front crescător al semnalului CLK; trebuie notat faptul că în acest caz vom folosi operatorul <= în loc de = pentru atribuirea valorilor;
- un exemplu de testbench este prezentat mai jos; expresia `forever #50 CLK = ~CLK` va nega valoarea lui *CLK* la fiecare 50s; simularea se va opri folosind comanda `$finish` ;
- simulați comportamentul bistabilului JK folosind un testbench;
- (facultativ) vizualizați schema RTL a circuitului;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

JK_FF.v

```
1   //JK flip-flop
2
3   module JK_FF(Q, QBAR, CLK, J, K);
4
5   input CLK, J, K;
6   output reg Q, QBAR;
7
8   always @ (posedge CLK)
9   begin
10    if (J == 1 & K == 0)
11     begin
12      Q <= 1;
13      QBAR <= 0;
14     end
15   if (K == 1 & J == 0)
16     begin
17      Q <= 0;
18      QBAR <= 1;
19     end
20   if (J == 0 & K == 0)
21     begin
22      Q <= Q;
23      QBAR <= QBAR;
24     end
25  if (J == 1 & K == 1)
26     begin
27      Q <= ~Q;
28      QBAR <= ~QBAR;
29     end
30  end
31
32  endmodule
```

JK_FF_tb.v

```
1   //testbench JK flip-flop
2
3   `include "JK_FF.v"
4
5   module JK_FF_tb;
6
7   reg J, K, CLK;
8   wire Q, QBAR;
9
10  JK_FF Instance0 (Q, QBAR, CLK, J, K);
11
12  initial begin
13    CLK=0;
14      forever begin #50 CLK = ~CLK; end
15
16  end
17   initial begin
18  #100 J = 1; K = 0;
19  #100 J = 0; K = 1;
20  #100 J = 1; K = 1;
21  #100 J = 0; K = 0;
22  #200 $finish;
23   end
24
25   initial begin
26      $monitor("time = %g, CLK = %b, J = %b, K = %b, Q = %b, QBAR = %b", $time, CLK, J, K, Q, QBAR);
27
28      $dumpfile("JK_FF_tb.vcd");
29      $dumpvars();
30  end
31
32  endmodule
```

4. Construcția unui bistabil T folosind folosind limbajul Verilog

- descrieți textual un bistabil T și salvați-l într-un fișier Verilog;
- simulați comportamentul dispozitivului construit folosind un testbench;
- (facultativ) vizualizați schema RTL a circuitului;
- vizualizați semnalele de intrare/ieșire folosind programul *gtkwave*;
- notați observațiile făcute și trageți concluzii.

Temă bonus (facultativă): Realizați un numărător MOD-16 folosind limbajul Verilog și demonstrați funcționalitatea acestuia folosind simulatorul *vpp* și programul *gtkwave*.