Verilog HDL

– Curs 2 –

***Verilog HDL** reprezintă un limbaj utilizat pentru descrierea sistemelor numerice. Sistemele numerice pot fi calculatoare, componenete ale acestora sau alte structure care manipulează informaţie numerică.*

Verilog poate fi utilizat pentru descrierea sistemelor numerice din punct de vedere comportamental și structural:

- **Descrierea comportamentală este legată de modul în care operează sistemul şi utilizează construcţii ale limbajelor tradiționale de programare, de exemplu *if* sau *atribuiri*.**

- **Descrierea structurală exprimă modul în care entităţile/ componentele logice ce alcătuiesc sistemul sunt interconectate în vederea realizării comportamentului dorit.**

Structura unui Program

Limbajul Verilog descrie un sistem numeric *ca un set de module*. Fiecare dintre aceste module are o interfata cu alte module, pentru a specifica maniera in care sunt interconectate. Modulele opereaza concurent.

Modulele reprezinta parti hardware, care pot fi de la simple porti pana la sisteme complete cum ar fi un microprocesor.

O specificare comportamentala defineste comportarea unui sistem numeric (modul) folosind construcțiile limbajelor de programare tradiționale.

O specificare structurală exprimă comportarea unui sistem numeric (modul) ca o conectare ierarhica de submodule.

Structura unui Program

Structura unui modul este urmatoarea:

```
module <nume_modul> (<lista de porturi>);
<declaratii>
<obiecte ale modulului>
endmodule
```

<nume_modul> reprezinta un identificator care, in mod unic, denumește modulul.

<lista de porturi> constituie o lista de porturi de intrare (input), ieșire (output) sau intrare/ieșire (inout), care sunt folosite pentru conectarea cu alte module.

<declaratii> specifica obiectele de tip date ca registre (reg), memorii și fire (wire), cât și construcțiile procedurale ca function-s și task-s.

<obiecte ale modulului> poate conține: construcții *initial*, construcții *always*, atribuiri continue sau apariții/instanțe ale modulelor.

Exemplu

```
module NAND(in1, in2, out);
        input in1, in2;
        output out;
                    // instructiune de atribuire continuă
        assign out = ~(in1 & in2);
endmodule
```

Porturile in1, in2 şi out sunt etichete pe fire.

assign urmareşte în permanenţă eventualele modificari ale variabilelor din membrul drept, pentru reevaluarea expresiei şi pentru propagarea rezultatului în membrul stang (out).

Observaţie!!!!

Instrucţiunea de atribuire continuă este utilizată pentru a modela *circuitele combinaţionale* la care ieşirile se modifică ca urmare a modificărilor intrărilor.

Invocarea unei instanţe este următoarea:

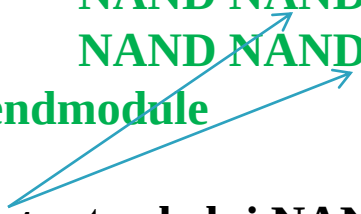
<nume_modul > <lista de parametri > <numele instantei> (<lista de porturi>);

<lista de parametri> are valorile parametrilor, care sunt transferate către instanţă (ex. întârzierea pe o poartă).

Exemplu:

// Modelul structural al unei porți AND formată din două porți NAND

```
module AND(in1, in2, out);
    input in1, in2;
    output out;
    wire w1;
    NAND NAND1(in1, in2, w1);
    NAND NAND2(w1, w1, out);
endmodule
```



Acest modul are doua instanţe ale lui NAND conectate printr-un fir intern w1.

Arhitectură TOP-DOWN !!!!

Exemplu de modul de nivel înalt, care stabileşte anumite seturi de date şi care asigură monitorizarea variabilelor.

```
module test_AND;
        reg a, b;
        wire out1, out2;

        initial begin // Datele de test
        a = 0; b = 0;
        #1 a = 1;
        #1 b = 1;
        #1 a = 0;
        end
        initial begin // Activarea monitorizarii
        $monitor("Time=%0d a=%b b=%b out1=%b out2=%b",
        $time, a, b, out1, out2);
        end
        // Instantele modulelor AND si NAND
        AND gate1(a, b, out2);
        NAND gate2(a, b, out1);
endmodule
```

- **Variabilele de tip reg stocheaza ultima valoare care le-a fost atribuita procedural.**
- **Firul, wire, nu are capacitatea de memorare. El poate fi comandat in mod continuu, de exemplu, prin instructiunea de atribuire continua assign sau prin iesirea unui modul.**

Atribuiri

- **continue** → **folosesc cuvantul cheie assign**
- **procedurale** → **<variabila reg > = <expresie>**
  unde <variabila reg> trebuie să fie un registru sau o memorie

apar *numai* în construcţiile *initial* şi *always*

Instrucţiunile din blocul construcţiei *initial* vor fi executate secvenţial, dintre care unele vor fi întârziate de #1 cu o unitate de timp simulat.

Construcţia *always* se comportă în acelaşi mod ca şi construcţia *initial* cu excepţia că ea ciclează la infinit (până la terminarea simulării).

- **atribuirea procedurală modifică starea unui registru, adica a logicii secvenţiale**
- **atribuirea continuă este utilizată pentru a modela logica combinaţională. Atribuirile continue comandă variabile de tip *wire***

Convenţii lexicale

- **Limbajul este case sensitive.**
- **Numerele sunt specificate:**

<dimensiune>< format baza><numar>

<dimensiune> specifica dimensiunea constantei ca număr de *biţi* (opţional)
<format baza> are un singur caracter ' urmat de unul dintre următoarele caractere b, d, o si h, care specifica baza de numeratie: binara, zecimala, octala si hexazecimala.
<numar> contine cifre, care corespund lui < format baza>.

Exemple:

```
549 // numar zecimal
'h 8FF // numar hexzecimal
'o765 // numar octal
4'b11 // numarul binar cu patru biti 0011
3'b10x // numar binar cu 3 biti, avand ultimul bit necunoscut
5'd3 // numar zecimal cu 5 ranguri
-4'b11 // complementul fata de 2, pe patru ranguri al numarului 0011 sau 1101
```

Tipuri de Date Fizice

- Variabilele **reg** stocheaza ultima valoare, care le-a fost atribuită procedural.

- Variabilele **wire** reprezintă conexiuni fizice între entităţi structurale cum ar fi porţile. Un fir (**wire**) nu stochează o valoare. O variabilă **wire** reprezintă numai o etichetă pe un fir.

Exemplu 1:

```
reg [0:7] A, B;
wire [0:3] Dataout;
reg [7:0] C;
```

Exemplu 2:

```
initial begin: int1
A = 8'b01011010;
B = {A[0:3] | A[4:7], 4'b0000};  // B este fortat la o valoare egala cu suma logica a primilor patru biti din A si a ultimilor patru biti din A, concatenata cu 0000. B are acum valoarea 11110000
end
```

Într-o expresie gama de referire pentru indici trebuie sa aibă expresii constante. Un singur bit poate fi referit ca o variabila.

Exemplu:

```
reg [0:7] A, B;
B = 3;
A[0: B] = 3'b111; // ILEGAL – indicii trebuie sa fie constanti!!
```

Un argument poate fi replicat prin specificarea numărului de repetiţii.

Exemple:

```
C = {2{4'b1011}};      //lui C i se asigneaza vectorul de biti: 8'b10111011
C = {{4{A[4]}}, A[4:7]};        // primii 4 biti reprezinta extensia lui A[4]
A[B] = 1'b1;                    // referirea la un singur bit este LEGALA
```

Tipuri de Date Abstracte

- **integer** reprezinta un intreg de 32 de biti cu semn
- **real** este fara semn
- **time** specifica cantitati, pe 64 de biti, care sunt folosite in conjunctie cu functia de sistem $time.

Operatori aritmetici binari: + ; - ; *; / ; %

Operatorii relationali compara doi operanzi si intorc o valoare logica, adica TRUE (1) sau FALSE (0) : <; >; <= ; >= ; == ; != .

Operatorii logici opereaza cu operanzi logici si intorc o valoare logica, adica TRUE (1) sau FALSE (0). Sunt utilizati in instructiunile *if* si *while*. A nu se confunda cu operatorii logici Booleeni la nivel de bit.

! Negatia logica
&& AND logic
|| OR logic

Operatori la nivel de bit:

~	**Negaţia la nivel de bit**
&	**AND la nivel de bit**
|	**OR la nivel de bit.**
^	**XOR la nivel de bit**
~&	**NAND la nivel de bit**
~|	**NOR la nivel de bit**

~^ sau ^~ Echivalenţa la nivel de bit NOT XOR

{ , } Concatenarea: {A[0], B[1:7]} concateneaza bitul zero din A cu bitii 1 pana la 7 din B.

<< Deplasare la stanga:

A = A << 2; // deplaseaza A cu doi biti la stanga si forţează zero în biţii eliberaţi.

>> Deplasare la dreapta

?: Condiţional:

A = C>D ? B+3 : B-2 //semnifică faptul că dacă
//C > D, atunci valoarea lui A este B+3, altfel B-2.

Construcţiile de control

sunt utilizate în secţiunile procedurale de cod,
adică în cadrul blocurilor **initial** şi **always**

1. Selecţia

instructiunea if

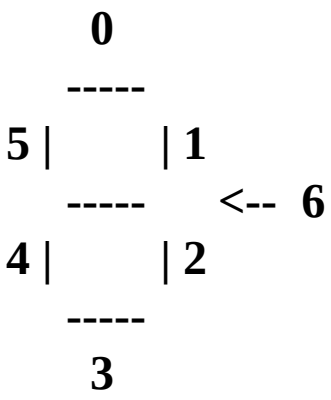
```
if (A == 4)
   begin
            B = 2;
   end
else
   begin
            B = 4;
   end
```

instrucţiunea case.

```
case (<expression>)
            <value1>: <instructiune>
            <value2>: <instructiune>
            default: <instructiune>
endcase
```

Exemplu de convertor binar-hexazecimal

7-segment encoding




```
module segment7(sin, sout);
        input [7:0] sin;
        output [6:0] sout;
        reg [6:0] sout;
  always @(sin)
    case (sin)
      8'b00000001 : sout = 7'b1111001;   // 1
      8'b00000010 : sout = 7'b0100100;   // 2
      8'b00000011 : sout = 7'b0110000;   // 3
      8'b00000100 : sout = 7'b0011001;   // 4
      8'b00000101 : sout = 7'b0010010;   // 5
      8'b00000110 : sout = 7'b0000010;   // 6
      8'b00000111 : sout = 7'b1111000;   // 7
      8'b00001000 : sout = 7'b0000000;   // 8
      8'b00001001 : sout = 7'b0010000;   // 9
      8'b00001010 : sout = 7'b0001000;   // A
      8'b00001011 : sout = 7'b0000011;   // b
      8'b00001100 : sout = 7'b1000110;   // C
      8'b00001101 : sout = 7'b0100001;   // d
      8'b00001110 : sout = 7'b0000110;   // E
      8'b00001111 : sout = 7'b0001110;   // F
      default : sout = 7'b1000000;   // 0
    endcase
endmodule
```