CALCULATOARE NUMERICE

Note de curs

G. A. Nemneș

CUPRINS:

I. Introducere

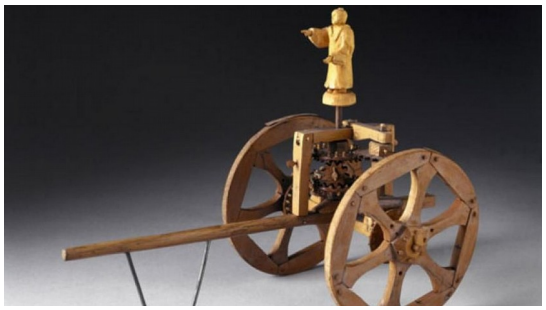
In prezent sistemele de calcul numeric reprezinta sisteme complexe care insumeaza tehnologii pentru procesare, transfer si stocare de date. Elementele constitutive sunt dispozitive nanoelectronice, nodul tehnologic actual ajungand la dimensiuni < 10nm, ceea ce reprezinta unui lungimea tranzistor.

Echipamentele de calcul au o istorie indelungata, primele exemple datand inca din antichitate. Odata cu dezvoltarea tehnologiei, acestea au evoluat de la instrumente de calcul, la sisteme mecanice de calcul, apoi la sisteme electro-mecanice de calcul, in prezent fiind sisteme electronice de calcul.

Cele patru etape de dezvoltare mentionate sunt detaliate in continuare.

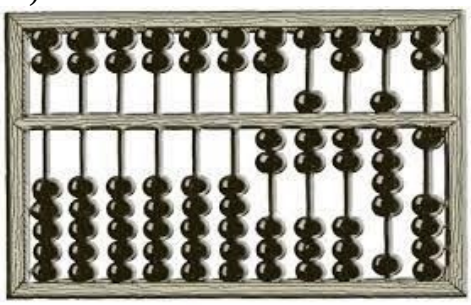
1. Etapa instrumentelor de calcul

- Carul care arata sudul – navigatie (1000 BC, China)

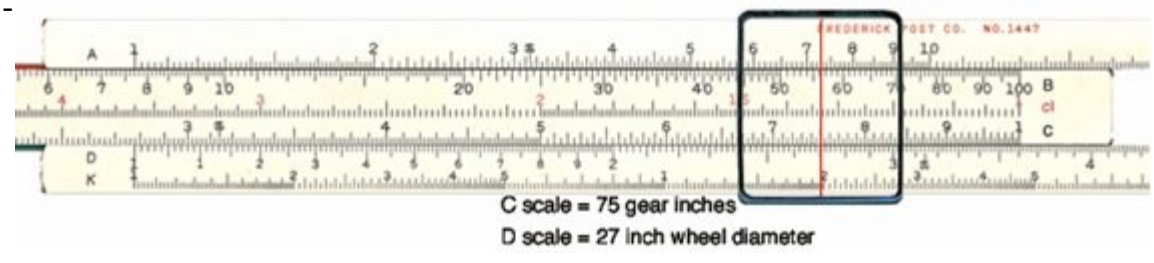


- Antikythera – calendar, primul calculator analogic (150-60 BC, Grecia)

- Abacul (Secolul XII, China)



-

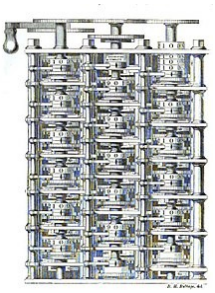


2. Etapa maşinilor mecanice de calcul

Se bazeaza pe sisteme actionate mecanic, cu roți dintate. In angrenaje, rotile dintate codifica cifrele zecimale.

- 1642 – Blaise Pascal: Masina de adunat Pascaline;

- 1694 – von Leibnitz: Masina de adunat si inmultit;
- 1823 – Charles Babbage: Masina diferentiala; este primul calculator cu executie automata a programului; im mod similar calculatoarelor moderne sistemul prezenta elemente principale: memorie, unitate de calcul, unitate de comanda si unitati de intrare/iesire;



- 1872 – E. Barbour: Prima masina de calcul cu imprimanta;

- 1892 – W. Burroughs: Masina de calcul de birou perfectionata;
- 1912 – F. Baldwin si J. Monroe: Masini de calcul cu patru operatii in productie de serie

3. Masini electromecanice de calcul

Spre deosebire de masinile mecanice, angrenajele cu roti dintate sunt actionate electric.

- 1930 – Masinile electromecanice sunt produse in masa; realizeaza adunare, scadere, inmultire, impartire, radacina patrata.

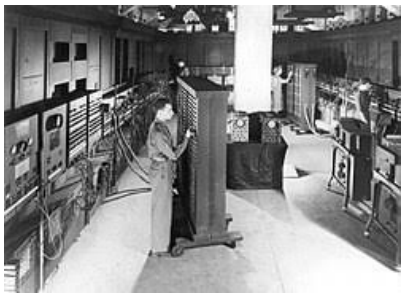
- 1937 - 1945 – Masini electromecanice cu relee electromagnetice avand functionalitatea unor elemente bi-stabile; efectueaza operatii binare; exemplu: Mark I.

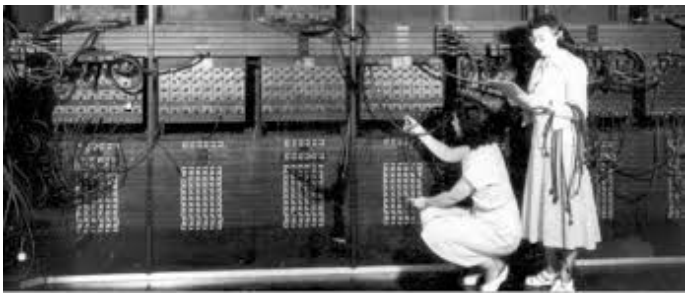
- 1937 : Howard Aiken (Harvard) demareaza constructia calculatorului Mark I; acesta este primul calculator cu secventa automata de comanda, urmarind principiile enuntate de C. Babbage;

4. Masinile electronice de calcul (calculatoarele moderne)

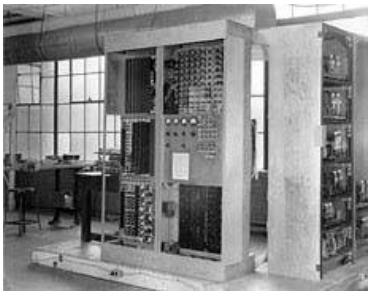
Constructia are la baza tuburi electronice, tranzistor si circuite integrate la scara
- mica (SSI: 20 tranzistori / pacheta de siliciu)
- medie (MSI: 20-1.000)
- larga (LSI: 1.000-50.000)
- foarte larga (VLSI: 50.000-1.000.000)
- ultra larga (ULSI: >1.000.000)

1943 – ENIAC (Electronic Numerical Integrator And Computer), la Universitatea din Pennsylvania, primul calculator cu tuburi elecronice; la proiect au participat J.P. Eckert, J.W. Mauchy si J. von Neumann; in memorie sunt stacate atat datele cat si programul → ceea ce permite modificarea cu usurinta a programului.





1945 – EDVAC (Electronic Discrete Variable Automatic Computer) utilizand pricipiile lui von Neumann;



Generatia I (1946 - 1956)
Hardware: Tuburi electronice, relee electromagnetice, tambur magnetic
Software: Programe cablate, cod masina.
Exemple: ENIAC, EDVAC, UNIVAC 1, IBM 650, CIFA 1-4, CIFA 101-102, MARICCA, MECIPT-1
Comunicatii: de tip telefon
Performante: Capacitate de memorie 2 KB, viteza de operare 10000 instructiuni/secunda.

CIFA -- Primul calculator din Romania! Calculatorul Institutului de Fizica Atomica.

Generatia II (1957 - 1963)
Hardware: Tranzistori, memorii cu ferite, cablaj imprimat, discuri magnetice
Software: Limbaje de nivel inalt (Algol, FORTRAN)
Exemple: NCR 501,IBM7094, CDC 6600, DACICC-1/2, CET 500/501, MECIPT-2, DACICC-200
Comunicatii: transmisiuni numerice, modulatii in coduri de impulsuri
Performante: Capacitate de memorie 32 KB, viteza de operare 2.000.000 instructiuni/secunda.

Generatia III (1964 - 1981)
Hardware: Circuite integrate, memorii cu semiconductori, cablaj imprimat multistrat, microprocesoare, discuri magnetice
Software: Limbaje de nivel foarte inalt, programare structurata, sisteme de operare oriectate pe limbaje de programare (Algol, FORTRAN), grafica, baze de date.
Exemple: IBM 360-370, PDP11/XX, Spectra 70, Honeywell 200, Cray-1, Illiac IV, Cyber205,RIAD 1-2, Felix C-256/512/1024, Independent 100/102F, Coral 4001/4030, Felix MC-8, Felix M18, M18-B, Felix M118, Felix M216
Comunicatii: prin satelit, microunde, fibre optice, tehnologii de comutare pe pachete
Performante: Capacitate de memorie 2 MB, viteza de operare 5.000.000 instructiuni/secunda.

ICE Felix HC-90 :

Generatia IV (1982 - 1989)
Hardware: VLSI, sisteme cu memorie distribuita, discuri optice, supercalculatoare
Software: Sisteme de operare evoluate, limbaje de programare cu nivel mare de abstractizare (ADA), pachete de programe de larga utilizare, limbaje orientate pe obiecte, baze de date relationale.

Exemple: IBM-43xx, VAX-11/7xx, IBM-308x, RIAD3, Coral 4021, Independent 106, Felix 5000, Coral 873, Felix PC
Comunicatii: retele digitale integrate
Performante: Capacitate de memorie 8 MB, viteza de operare 30.000.000 instructiuni/secunda.

Generatia V (1990 - prezent)
Hardware: ULSI, circuite integrate 3D, tehnologii Ga-As, jonctiuni Josephson, comonente optice, arhitecturi paralele, retele neuronale
Software: Sisteme de operare cu interfata evoluata, libaje concurente, programare functionala, prelucrare simbolica, realitate virtuala si augumentata
Exemple: statii de lucru, supercalculatoare, retele de supercalculatoare, GRID, HPC
Comunicatii: dezvoltare extensiva a sistemelor distribuite, retele din fibra optica de mare capacitate, frecvente de transmisie de ordinul GHz, telefonie digitala mobila, Internet
Performante: Capacitate de memorie >1 GB, viteza de operare 1 Ginstructiuni/secunda - 1 Tinstructiuni/secunda.

II. Verilog HDL

Verilog HDL reprezintă un limbaj utilizat pentru descrierea sistemelor numerice. Sistemele numerice pot fi calculatoare, componenete ale acestora sau alte structuri care manipuleaza informaţie numeric.

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.

Scurt istoric:

- Verilog : soft dezvoltat pentru a combina diferite niveluri de simulare;

- 1980 : existau simulatoare la nivel de comutare, porti sau functionale; majoritatea erau limbaje secventiale → dificultati in modelarea concurentei circuitelor digitale;

- 1984 : limbajul Verilog - inventat de Phil Morby de la Automated Integrated Design Systems (redenumita Gateway Design Automation-GDA)

- 1987 : extins si redenumit Verilog-XL; imprumuta elemente de la limbajele existente:
    - concurenta de la Modula si Simula;
    - sintaxa de la C;
    - nivel de abstractizare de la HiLo

- 1987 : GDA cumparata de Cadence → Verilog ajunge in domeniul public → alte companii dezvolta tool-uri alternative, ceea ce ajuta la popularizarea limbajului;
- In paralel, in 1981, Departamentul American al Apararii sponsorizeaza un workshop → 1983 se dezvolta VSIC HDL (Very High Speed Integrated Circuits HDL, VHDL) → 1987devine standard IEEE 1076

- 1991: Cadence organizeaza Open Verilog International (OVI)

- 1995: Verilog-HDL devine stnadard comercial IEEE-1364, referit ca Verilog-95

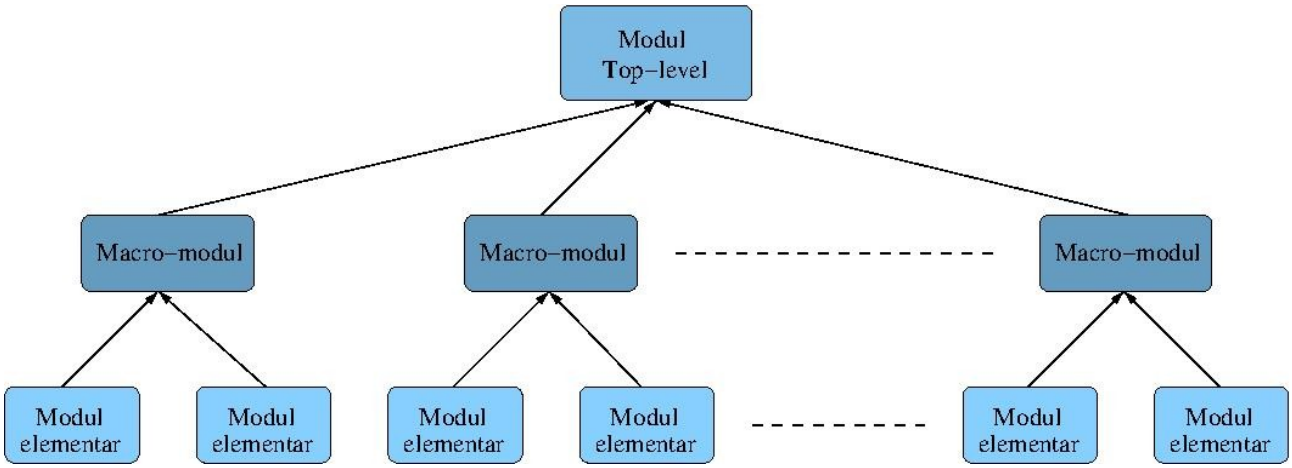
- 2001: apare Verilog-2001

- 2005: Verilog-2005, System Verilog; prin includerea suportului de modelare analogic si mixt devine Verilog AMS

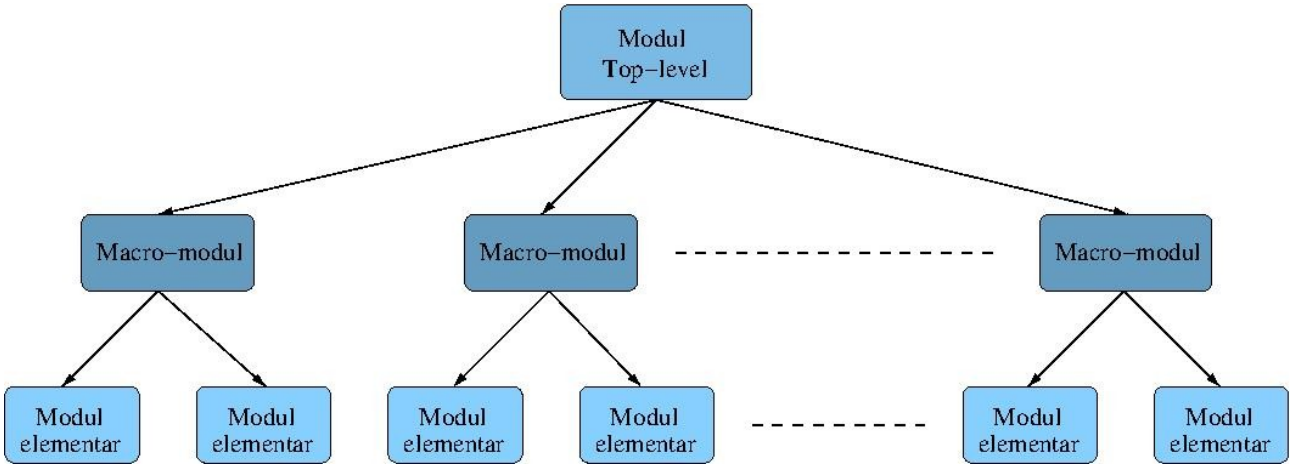
Verilog HDL : Stiluri de proiectare

Verilog permite ca la orice alt limbaj de descriere hardware, utilizarea ambelor metodologii: “bottom-up” si “top-down”.

Bottom-up: metoda traditionala; proiectul este realizat la nivel de porti → dificultate in a reprezenta structuri de milioane de tranzistori → se grupeaza elementele la macronivel



Top-down: metoda actuala; se pleaca de la specificatiile de sistem si se detaliaza in jos pana la componentele de sistem;
Exista avantaje in ceea ce priveste schimbarea tehnologiilor, proiectarea structurata.



Printr-o abordare ierarhica, un sistem hardware complex poate fi descris sub aspect comportamental, structural, fizic si abstract pe mai multe niveluri:

Descriere comportamentala	Descriere structurala	Descriere la nivel de procese fizice	Descriere abstracta
- Aplicatii software; - Sistem de operare; - Programe utilizator; - Functii, subrutine, instructiuni.	- Procesor, placa de baza, placa video, memorie (RAM, HDD); - Sumatoare, porti, registrii; - Tranzistori, diode, condensatori etc.	- Curenti de sarcina in circuite, dispozitive; - Functionarea elementelor de circuit: tranzistori, diode etc.	- Arhitectura generica; - Algoritmi; - Functionalitatea modulelor; - Logica de comutatie;

Verilog HDL : Fluxul de proiectare

Specificaţii de proiectare:
Este indicati parametrii de proiectare generici care definesc sistemul. Este definita functionalitatea sistemului, arhitectura, interfata cu medil extern. Sunt indicate prototipurile functiilor de sistem.

Descriere comportamentala:
Este indicata descrierea comportamentala prin intermediul careia este analizat sistemul din punct de vedere al functionalitatii, performantei, al compliantei cu standardele si sunt considerate alte aspecte de nivel inalt. Astfel de descrieri pot fi implementate folosind limbaje de descriere hardware (Hardware Description Languages, HDLs).

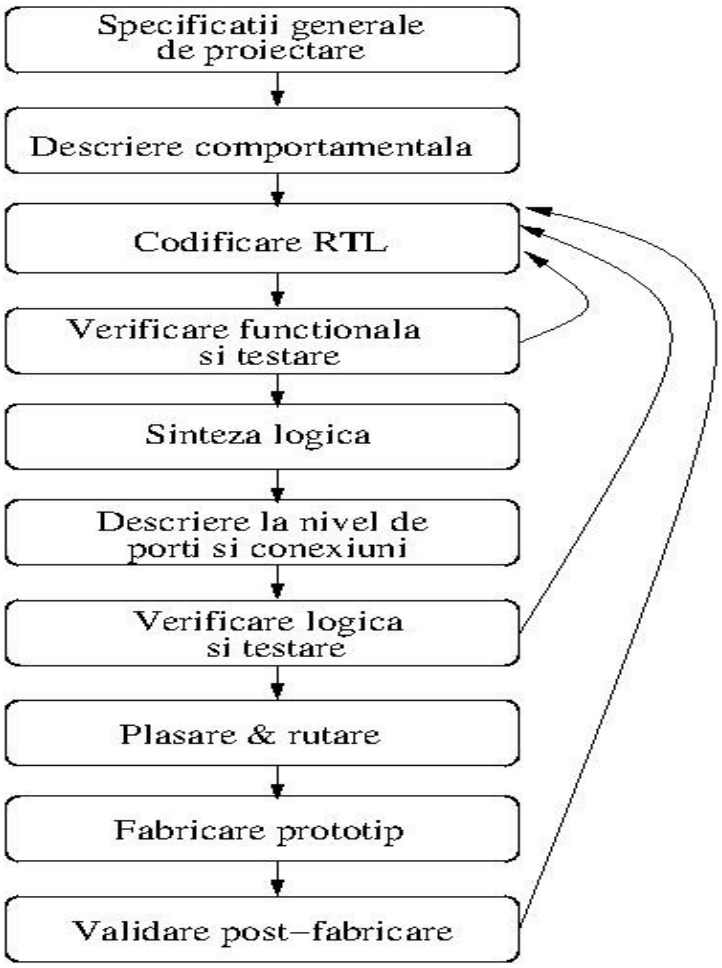
Codificarea RTL:
Descrierea comportamentala este convertita in descriere RTL prin intermediul unui HDL. Este descris fluxul de date care implementeaza circuitul digital. Din acest punct, procesul de design este asistat de calculator (Computer-Aided design, CAD).

Sinteza logica:
Tool-urile de sinteza logica convertesc descrierea RTL intr-o descriere la nivel de porti si conexiuni. Aceasta constituie un input pentru etapa urmatoare de plasare si rutare.

Plasarea şi rutarea:
Este realizata dispunerea elementelor de cicuit pe placa, in functie de constrangeri (e.g. surse de tensiune, cablaje etc)

Validare postfabricare:
Se realizeaza un prototip care este testat in conditii reale. Aceasta etapa precede lansarii produsului pe piata.

Diagrama fluxului de proiectare:



Sintaxa si conventii lexicale:

Conventiile lexicale sunt similare cu cele ale limbajului de programare C. Verilog este un limbaj “case sensitive” , astfel inacat toate cuvintele cheie sunt scrise cu litere mici.

Spațiile albe pot conține caractere precum: spații(/b– “blank”), caractere “tab” (/t), caractere linie noua (/n), caractere sfarsit de linie (“Carriage Return” <CR>), sfarsit de paragraf si sfarşit de fisier (EOF – “End Of File”).

Comentarii
Modalitati de a introduce comentarii:
- Comentarii pe o singura linie, utilizand //
- Comentarii pe mai multe linii, utilizand /* ... */

Identificatori
- Identificatorii sunt nume date pentru un obiect (registru, modul, functie etc). Caracterul de inceput este un caracter alfabetic sau caracterul “_”. Pot avea in componenta si caractere numerice si caracterul "$". Lungimea maxima a unui identificator este de 1024 caractere.

Structura unui program este urmatoarea:
- Modul main (de test);
- Modul sau module top-level;
- Submodul 1;
  ... ...
- Submodul n;

```
module hello_world;                 // declaratie modul

initial begin                       // incepe blocul initial
      $display( "Hello world!" );   // afiseaza mesaj
      #10 $finish;                  // exit
end                                 // se termina blocul initial

endmodule
```

Structura unui modul este urmatoarea:

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

<nume_modul> reprezinta un identificator care denumeste modulul in mod unic

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

<declaratii> specifica obiectele de tip date ca registrii (reg), memorii si fire (wire) cat si constructiile procedurale precum functii si task-uri.

<obiecte ale modulului> pot contine constructii initial, constructii always, atribuiri continue sau aparitii/instante ale modulelor.

EXEMPLU:

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

Porturile in1, in2 si out sunt etichete pe fire.

Instructiunea assign urmareste in permanenta eventualele modificari ale variabilelor din membrul drept, pentru reevaluarea expresiei si pentru propagarea rezultatului in membrul stang.

Instantierea unui modul:

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

<lista de parametri> contine valorile parametrilor care sunt transferati catre instanta

EXEMPLU:

```
// Construim o poarta AND din doua porti NAND
module AND( in1, in2, out);
        input in1, in2;
        output out;
        wire w1;
        NAND NAND1( in1, in2, w1);
        NAND NAND2( w1, w1, out);
endmodule
```

Observatii:
- Modulul AND are doua instante ale lui NAND conectate printr-un fir;
- Denumirea instantelor este arbitrara (e.g. NAND1 si NAND2);

In continuare este prezentat un exemplu in care se stabilesc datele de intrare si sunt monitorizate iesirile:

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

       // datele de test
       initial begin
          a = 0; b = 0;
          #1 a = 1;
          #1 b = 1;
          #1 a = 0;
       end

        // activarea monitorizarii
        initial begin
            $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** (registru) stocheaza ultima valoarea care le-a fost atribuita procedural.

Variabilele de tip **wire** (fire) nu au capacitate de memorare. Firele pot stabili conexiuni intre module, fiind comandate in mod continuu, e.g. utilizand intructiunea assign.

Atribuirile pot fi *continue* sau *procedurale*.

Atribuirile continue folosesc cuvantul cheie assign.

Atribuirile procedurale folosesc expresia

<variabila reg> = <expresie>

unde <variabila reg> este un registru sau o memorie. Atribuirile procedurale apar numai in constructiile initial si always.

Tipuri de date

Numere. Exista doua specificatii *sized* (cu dimensiune specificata) si *unsized* (fara dimensiune specificata).

Numerele sunt specificate prin

<marime>'<baza> <valoare>

- <marime> (opțional) este numarul de biți din intreg. Intregii fara semn sunt predefiniți la 32 biți
- <baza> (opțional) este baza de numerație (binara, octala, zecimala sau hexazecimala). Se utilizeaza numere decimale. Nu este case sensitive.
- <valoare> nu este case sensitive;

Numerele sunt cu semn sau fara semn. Intre marime, baza şi valoare sunt permise spații.

Baza	**Simbol**	**Valoare**
Binar	b sau B	0, 1, x, X, z, Z, ?, _
Decimal	d sau D	0 9, _
Octal	o sau O	0 7, x, X, z, Z, ?, _
Hexazecimal	h sau H	0 9, a f, A F, x, X, z, Z, ?, _

- Valori: cifre (0-9), hexazecimal (A-F), logice 0, 1, X (nedefinit), Z (impedanta ridicata).
- Numarul poate fi citit de la dreapta la stanga (in conventia LSB - *least significant bit*) sau de la stanga la dreapta (in conventia MSB - *most significant bit*).
- Daca <marime> < <valoare> atunci se trunchiaza rezultatul, eliminand biti cei mai semnificativi
- Caracterul "_" este ignorat; poate fi introdus oriunde in <valoare> pentru a usura citirea unui numar binar lung;
- Caracterul "?" este echivalent cu "Z"

Se pot defini numere reale:

- in format decimal: <valoare_1>.<valoare_2>
- in format stiintific: <valoare_mantisă>E<valoare_exponent>
unde E nu este case sensitive

Observatii:
- numerele reale se rotunjesc la cel mai apropiat intreg atunci cand sunt atribuite unui numar intreg;
- numerele reale nu pot contine valori de tip Z sau X;

EXEMPLE:

2871 // numar zecimal
5'b10011 // numar binar
3'o523 // numar octal
4'h9F4A // numar hexazecimal
4'b10x1 // numar binar cu un bit necunoscut
-4'b0001 // complement la 2 al 0001, adica 1111
4'd-2 // incorect

Valori logice.
0 – zero, nivel scazut sau fals
1 – unu, nivel inalt sau adevarat
Z sau z – inalta impedanta (valoare trei stari sau flotanta)
X sau x nedefinit, necunoscut sau neinițializat

Puteri logice.

0 - Impedanță înaltă - *high-z*
1 - Capacitate mică - *small*
2 - Capacitate medie - *medium*
3 - "Drive" slab cuplat - *weak*
4 - Capacitate mare - *large*
5 - "Drive" ținut („pull") - *pull*
6 - "Drive" puternic cuplat - *strong*
7 - "Drive" de alimentare - *supply*

Sunt utilizate pentru a rezolva conflicte pe fire. Astfel, daca doua semnale, strong1 si weak2, sosesc pe acelasi fir va prevala strong1. Daca semnalele au aceeasi tarie, e.g. strong1 si strong2, atunci rezultatul va fi "x" (nedeterminat). Este utilizat indeosebi pentru modelarea dispozitivelor MOS in regim dinamic.

Stringuri.

Reprezinta secvente de caractere incadrate de "...". Stringurile se definesc pe o singura linie.

EXEMPLU:
"Hello World! " // este un string
"a + b" // este un string

Observatii:

- Se pot stoca stringuri in registrii, fiecare caracter ocupand 8 biti.

- Daca dimensiunea registrului este mai mare --> se completeaza cu zero-uri

- Daca dimensiunea registrului este mai mica --> bitii din partea stanga a stringului sunt ignorati

Tipuri de date primare: registrii si fire.

Registrii reprezinta elemente de stocare de date, care pot retine o valoare pana cand o alta valoare le este atribuita. Registrul este asadar o variabila care poate retine o valoare si, spre deosebire de fire, nu necesita un *driver*. Sunt descrisi de cuvantul cheie **reg**. Un registru neintializat ia valoarea x. Registrii sunt utilizati in blocuri procedurale, folosind directive de tip *initial* sau *always*.

EXEMPLU:

```
reg r; // definim registrul pe 1 bit
initial begin
    r = 1'b1; // atribuim valoarea 1
    #10 r = 1'b0;  // dupa 10 pasi de intarziere valoarea este resetata
end
```

Tipuri de registrii: **reg**, **integer**, **time**, **real (realtime):**
- reg - poate defini o variabila pe un numar arbitrar de biti
- integer - variabile pe 32 biti
- time - variabile pe 64 biti
- real / realtime - variabile in virgula mobila cu dubla precizie

Firele reprezinta conexiuni intre elementele hardware. Ele nu pot stoca o valoare, ci doar preiau valoarea de iesire (output) de la un element de circuit si o conduc catre intrarea (input) unui alt element.

Tipuri de fire:

wire sau **tri**: Interconectare simplă de fire
wor sau **trior**: Ieşiri legate prin SAU cablat (model ECL)
wand sau **triand**: Ieşiri legate prin ŞI cablat (model ”open collector”)
tri0 : Legătura la masă cu trei stări (“Pulls down” cu „tri stated”)
tri1 : Legătura la sursă cu trei stări (“Pulls up” cu „tri stated”)
supply0 : Constantă logică 0 (“supply strength”)
supply1 : Constantă logică 1 (“supply strength”)
trireg : Reține ultima valoare când trece în trei stări (”capacitance strength”)

Firele transfera atat valori logice, cat si puteri logice.

EXEMPLU:

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

```
    NAND NAND2( w1, w1, out);
endmodule
```

Vectori.

Registrii si firele pot fi declarate ca vectori, lungimea acestora defining numarul de biti pe care stocheaza/conduce o valoare.

EXEMPLU:

```
reg a;  // registru pe 1 bit
reg [0:7] b; // registru pe 8 biti
wire w1; // fir pe 1 bit
wire [0:7] w2;  // fir pe 8 biti
```

Intr-o declaratie de tipul

```
reg [7:0] b;
```

este modificata ordinea bitilor in registru (LSB --> MSB).

Concatenarea valorilor.

O valoare concatenata se obtine {val1,val2}

```
A = 8'b11001101;
B = {{4{A[1]}}, A[4:7]};
```

Valoarea lui B este 8'b00001100.

Memorii.

Memoriile sunt vectori de registrii. Fiecare element de memorie este un cuvant (word). Fiecare cuvant poate avea o lungime de unul sau mai multi biti.

EXEMPLU:

O declaratie pentru o memorie de 1 kbits, avand lungimea cuvantului de 32 de biti arata astfel:

```
reg [31:0] memory[0:1023];
```

Memoria poate fi accesata ca orice array. Astfel, putem utiliza in cod valoarea elementului "511" sub forma memory[511].

Alte tipuri de date.

parameter - Constantă “run time” pentru memorarea întregilor, numerelor reale, timpului, întârzierilor sau şirurilor ASCII. Parametrii pot fi redefiniți pentru fiecare instanță a modulului, dar nu pot fi folositi ca variabile.

specparam - Specifică constanta bloc pentru memorarea întregilor, numerelor reale, timpului, întârzierilor sau şirurilor ASCII.

event - Un “flag” temporar fără valoare logică sau memorare date. Este utilizat adesea pentru sincronizarea activităților concurente dintr un modul.

OPERATORI

In general pot actiona asupra unui operand (in expresii unare) sau, mai uzual, folosind doi operanzi (expresii binare). Operanzii pot fi registrii sau fire sau vectori ai acestora.

a). Operatori aritmetici binari.
Exemple: +, - , * , / , % (modulo)

b). Operatori relationali (compara valori logice) si returneaza valoarea logica corespunzatoare.
Exemple: < , <= , > , >= , == , !=

c). Operatori logici: utilizati in instructiuni de tip *if* si *while*.
Exemple: "!" - negare ; "||" - sau logic ; "&&" - si logic.

d). Operatori la nivel de bit:

~a - Negatie la nivel de bit (inverseaza pe a) ;
a&b - SI la nivel de bit
a | b - SAU la nivel de bit
a ^ b - XOR la nivel de bit
a ~| b - NOR la nivel de bit
a ~^ b - NOT XOR = echivalenta la nivel de bit

e). Operatori de reducere:

&a - SI logic pe toti bitii
~&a - SI logic negat pe toi bitii
|a - SAU logic pe toti bitii
~|a - SAU logic negat pe toti bitii
^a - XOR pe toti bitii

f). Deplasarea pe biti:
a = a<<3 Deplasare la stanga cu doi biti. Bitii eliberati devin zero.
a = a >>3 Deplasare la dreapta cu doi biti, similar.

g). Alti operatori:
- Operator conditional ?:
a = b < c ? d+1 : d-1 // daca b<c atunci a = d+1, altfel a = d-1
- Concatenarea {,}
c = {a,b} // concateneaza pe a cu b rezultand un vector mai lung
- Multiplicarea {{,}}

c = {a{b}} // replica b de a ori

CONSTRUCTII DE CONTROL

Pentru realizarea elementelor de salt conditionat, cicluri se utilizeaza o serie de intructiuni similare cu cele din limbajul C.

1. Instructiunea *if ... else :*

EXEMPLU:

```
integer a, b;

if (a=1)
    begin
        b=2;
     end;
else
    begin
        b=3;
     end;
else
```

2. Instructiunea *case*:

```
case (<expresie>)
          <valoare1>: <instructiune1>
          <valoare2>: <instructiune2>
          <valoare3>: <instructiune3>
           default: <instructiune_default>
endcase
```

3. Instructiuni de ciclare (repetitie): *for*, *while, repeat* si *forever*.

a). Instructiunea *for*:

```
for ( i=0 ; i<8 ; i=i+1 )
       begin
            $display("i =%d", i);
       end
```

b). Instructiunea *while*:

```
i = 0;
while (i<8)
      begin
          $display("i =%d", i);
           i = i + 1;
      end
```

c). Instructiunea *repeat*:

```
i = 0;
repeat(8)
begin
    $display("i =%d", i);
    i = i + 1;
end
```

d). Instructiunea *forever*:

Executa continuu un grup de intructiuni.

```
forever
    begin
        <instructiuni>
    end
```

INSTRUCTIUNI BLOCANTE SI NON-BLOCANTE

Atribuirea se poate realiza in doua moduri: atribuire blocanta sau atribuire non-blocanta. In prima situatie, atribuirea se face dupa fiecare operatie. In a dou situatie, atribuirea se face la sfarsitul unitatii de timp.

Presupunem ca initial a = 4;
Atribuire blocanta "=":

```
always @ (negedge clk)
        begin
            a  = a + 2;   // a = 6
            a  = a + 3;   // a = 9
        end
```

Rezultatul final (la realizarea frontului negativ de ceas) este a = 9.

Atribuire non-blocanta "<=":

```
always @ (negedge clk)
        begin
            a  <= a + 2;   // a = 6
            a  <= a + 3;   // a = 7
        end
```

Rezultatul final (la realizarea frontului negativ de ceas) este a = 7.

FUNCTII SI TASK-URI

O functie Verilog este asemanatoare cu un task, cu unele diferente:

- Functia nu poate avea decat o singura iesire si cel putin o intrare;
- Functiile nu pot contine intarzieri, timing sau eveniment de control;
- O functie incepe cu cuvantul cheie **function** si se termina cu **endfunction**;
- Functiile sunt definite in modulul in care sunt utilizate; este posibil ca functiile sa fie definite in fisiere separate si sa se foloseasca directiva de compilare "*include*";

Sintaxa unei functii Verilog:

function [dimensiune/tip] nume_functie;
    - declaratii de input (registrii etc)
    - declaratii de variabile locale
    - constructii procedurale / instructiuni
endfunction

Caracteristicile functiilor Verilog:

- Functiile trebuie sa aibe cel putin o intrare si nu pot avea iesiri sau "inout"-uri;

- Nu pot contine sincronizari de tip tip posedge, negedge, #, @ sau wait → functiile se executa cu intarziere zero;

- Variabilele declarate intr-o functie sunt variabile locale;

- Functiile pot manipula si variabile globale : in acest caz, variabilele sunt definite inaintea functiei, in modulul in care se defineste functia;

- Functiile pot apela alte functii insa nu si task-uri;

- Functiile pot fi utilizate pentru a modela doar logica combinationala;

Exemplu de functie:

```
module modul_adunare;

function [7:0] functie_adunare;  // se defineste functia
input [7:0] a, b;
begin
    functie_adunare  = a + b;
end
endfunction

endmodule
```

Apelarea functiei se face in felul urmator:

```
module efectueaza_adunare ( a, b,  suma);

input [7:0] a, b;
output [7:0] suma;
wire [7:0] suma;
```

assign suma = adunare (a, b);

endmodule

Caracteristicile task-urilor:
- Pot contine intarzieri, timing sau operator eveniment;
- Procedura poate avea oricate iesiri (zero sau mai multe);

Exemplu: Se calculeaza paritatea unei secvente de biti utizand un task.

```
module exemplu_task;

task calculeaza_paritate;
      input [3:0] x;
      output z;
      z = ^ x;
endtask

initial
   begin: init1
      reg r;
      parity(4'b1101,r); // invocare task : 1011, 1111
      $display("p=%b",r);
   end

endmodule
```

Comparatie intre functii si task-uri:

Functii	Task-uri
Functiile au cel putin un argument de intrare. Nu pot avea argumente de tip output sau inout.	Task-urile pot avea zero sau mai multe argumente de intrare de tip input, output, inout
Functiile returneaza o singura valoare.	Task-urile pot avea mai multe iesiri (output sau inout)
Functiile nu pot contine intarzieri, timing sau operator eveniment.	Task-urile pot contine intarzieri, timing sau operator eveniment.
Functiile se executa in aceasi pas de timp.	Task-urile se pot executa pe parcursul mai mulor pasi de timp.
O functie poate chema o alta functie dar nu si un task.	Un task poate chema alte task-uri sau functii.

Controlul sincronizarii in Verilog HDL

In Verilog HDL avem la dispozitie trei modalitati de a controla sincronizarea:

1). Intarziere "#": detemina un interval de timp (in pasi de timp) intre aparitia instructiunii si momentul executiei;
2). Operator eveniment "@": conditioneaza executia intructiunii

3). Comanda "wait" : asteapta modificarea unui registru, valoare pe fir etc.

Asadar, intr-o simulare, instructiunile se succed cu diferite intarzieri datorate: unor intarzieri explicite date de "#" sau conditionate de operatorul eveniment "@" sau intructiunea "wait".

Exemple:

1). Intarzieri "#":

```
a = 1;
#3 a = 2;  // valoarea lui a se modifica dupa 3 pasi de intarziere
```

2). Operatorul eveniment "@":

```
@(posedge clock)   a = -a  //  se inverseaza semnul lui a pe frontul
                   //      pozitiv de ceas
```

3). Instructiunea "wait":

```
wait (a==2)     // astept pana cand a ia valoarea 2
begin
      b = b + 1
end
```

CIRCUITE COMBINATIONALE
Descriere structurala vs. comportamentala

Dorim sa asamblam un circuit care sa efectueze o anumita functie logica. Exista doua modalitati de abordare: (1) descriere structurala si (2) descriere comportamnetala.

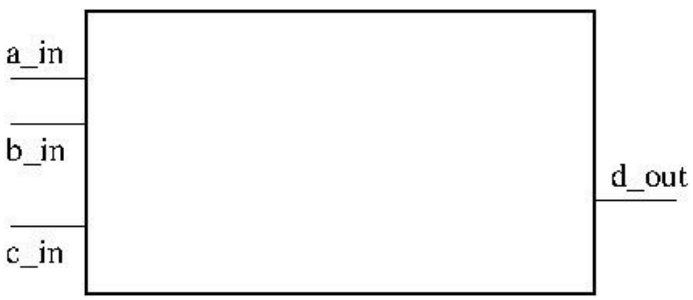
(1) Descriere structurala.

In aceasta abordare, circuitul este alcatuit din module (porti logice) care sunt interconectate. Sunt specificate porturile de intrare, porturile de iesire si fire de conexiune. Circuitul astfel construit este incadrat intr-un modul si urmeaza a fi testat intr-un modul de test (*testbench*). Descrierea structurala are avantajul ca indica concret blocurile logice in cele mai fine detalii care urmeaza sa fie integrate pe placa.

Parcurgem etapele de asamblare a circuitului logic:

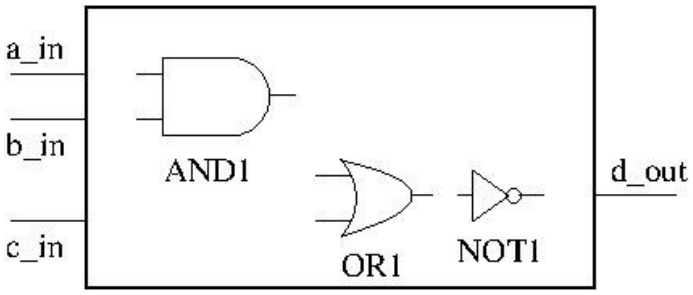
Etapa I. *Specificarea porturilor.*

```
// Circuit
module Circuit ( a_in, b_in, c_in, d_out);
      input a_in, b_in, c_in;
      output d_out;
      reg d_out;
endmodule
```



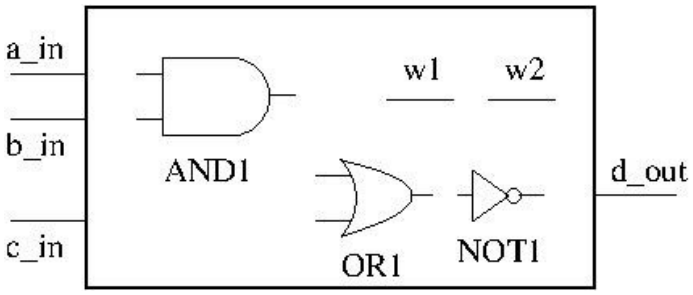
Etapa II. *Specificarea elementelor componente (e.g. porti logice).*

```
// Circuit
module Circuit ( a_in, b_in, c_in, d_out);
        input a_in, b_in, c_in;
        output d_out;
        reg d_out;

        AND  AND1();
        OR  OR1();
        NOT  NOT1();

endmodule
```



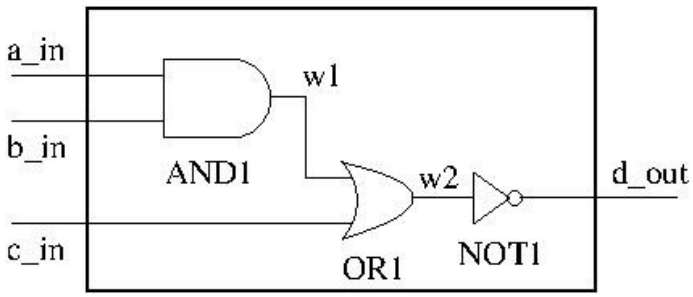
Etapa III. *Se indica firele de legatura.*

```
// Circuit
module Circuit ( a_in, b_in, c_in, d_out);
        input a_in, b_in, c_in;
        output d_out;
        reg d_out;

        wire w1, w2;

        OR  OR1();
        AND  AND1();
        NOT  NOT1();

endmodule
```



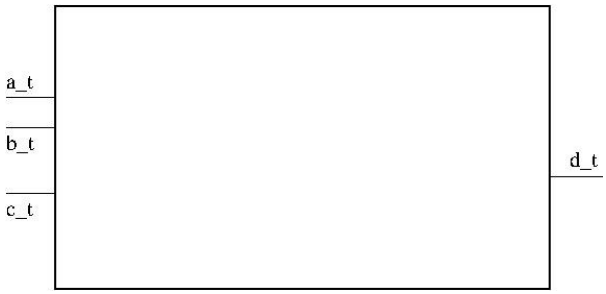
Etapa IV. *Se conecteaza firele la porturile intrare/iesire ale elementelor componente.*

```
// Circuit
module Circuit ( a_in, b_in, c_in, d_out);
      input a_in, b_in, c_in;
      output d_out;
      reg d_out;

      wire w1, w2;

      AND  AND1( a_in, b_in, w1) ;
      OR  OR1( w1, c_in,w2);
      NOT  NOT1( w2, d_out);

endmodule
```



Etapa V. *Se specifica porturile de intrare si iesire in modulul de test.*

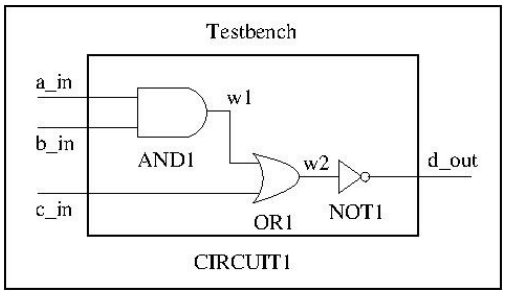
```
module Testbech;
     reg a_t, b_t, c_t;
     wire d_t;

endmodule
```



Etapa VI. *Se adauga circuitul in modulul de test.*

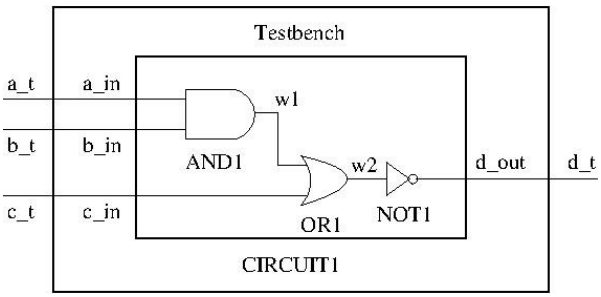
```
module Testbech;
```

```
    reg a_t, b_t, c_t;
    wire d_t;

    Circuit Circuit1( a_t, b_t, c_t, d_t);

endmodule
```



Etapa VII. *Se adauga valorile de test si se afiseaza rezultatul.*

```
module Testbench;
    reg a_t, b_t, c_t;
    wire d_t;

    Circuit Circuit1( a_t, b_t, c_t, d_t);

    initial begin
        // date de test
        a_t = 1;
        b_t = 0;
        c_t = 1;
        #1 $display ( "d_t = %b", d_t);
    end

endmodule
```



(2) Descriere comportamentala (BHV).

In aceasta descriere circuitul reproduce functionalitatea logica, fara a fi descompus pe componente fizice (module elementare, porti, dispozitive). Este o maniera mai directa de a obtine

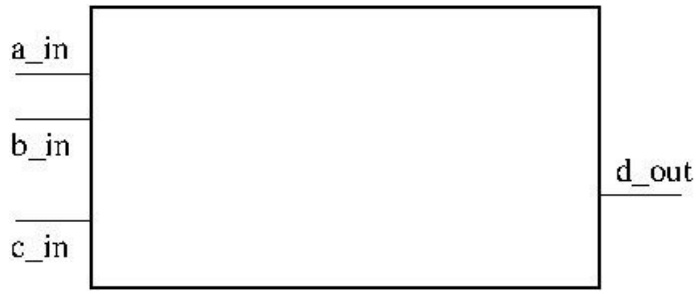
functionalitatea, insa totodata si un mod mai abstract, fara a putea lua in calcul optiuni pentru optimizarea localizarii elementelor pe placa.

Etapa I. *Se indica porturile de intrare in modulul circuitului.*

```
// Circuit 1
module Circuit ( a_in, b_in, c_in, d_out);

endmodule
```

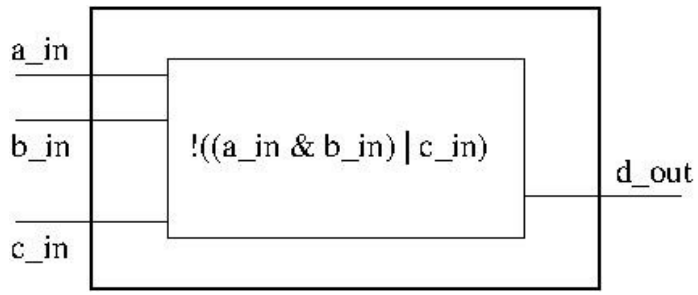
Etapa II. *Se indica porturile de intrare si iesire.*

```
// Circuit 1
module Circuit ( a_in, b_in, c_in, d_out);
    input a_in, b_in, c_in;
    output d_out;
    reg d_out;

endmodule
```



Etapa III. *Se adauga functia logica intr-un singur bloc.*

```
// Circuit 1
module Circuit ( a_in, b_in, c_in, d_out);
    input a_in, b_in, c_in;
    output d_out;
    reg d_out;

    always @ (a_in or b_in or c_in)
    begin
        d _out= ( a_in | b_in) & (~c_in);
    end

endmodule
```



Modulul de test se asambleaza intr-un mod similar ca si in cazul descrierii structurale.

Logica combinationala vs. secventiala

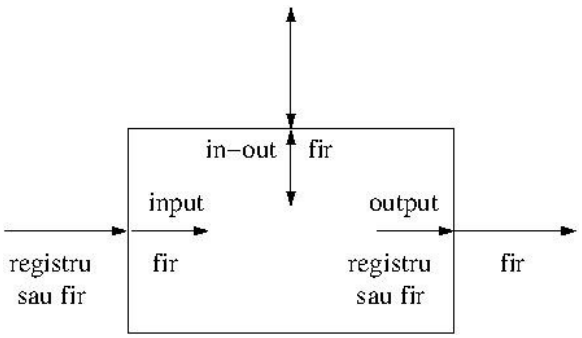
In **logica combinationala** output-ul este o functie pura de input-ul curent; se modeleaza prin circuite booleene.

In **logica secventiala** output-ul nu depinde numai de input-ul curent, ci si de cele anterioare, de istoria input-ului.

Asadar, logica secventiala are memorie, spre deosebire de logica combinationala.

Exemplu de logica secventiala: telecomanda – butoanele “channel up” si “channel down”; canalul selectat depinde de input-urile anterioare care determina starea curenta; canalul selectat depinde de input-ul up/down cat si de stare.

Reguli de conectare a porturilor : reg vs. wire



Firele (wires):

- Se folosesc pentru a conecta porturile de input si de output;
- Se folosec ca input/output in declaratiile unui modul;
- Trebuie conduse de un stimul initial;
- Nu se pot folosi in stanga operatorului de atribuire “=” sau “<=” intr-un bloc “always@” ;
- Se pot folosi in partea stanga intr-o instructiune de tip assign;
- Se folosesc pentru a modela logica combinationala;

Exemplu:

```
wire  w1, w2, w3;
reg  A;
```

```
assign  w1 = w2 & w3;    // utilizare cu assign

always @ ( w2 or w3 )    // utilizare la dreapta lui "always"
       begin
           A = w2 & w3;     // constructie procedurala
       end
```

Registrii:

- Similari cu firele, dar se pot folosi si pentru a stoca informatia (starea);
- Nu pot fi conectati la output-ul unei instantieri de modul;
- Pot fi folositi ca output in declaratiile unui modul;
- Nu pot fi folositi ca input in declaratiile unui modul;
- Este singura varianta permisa in stanga operatorului “=” sau “<=” in blocul always @;
- Este singura varianta permisa in stanga operatorului “=” intr-un bloc initial (in TestBench);
- Nu se poate folosi in stanga operatorului “=” intr-o intructiune de tip assign.

PROCEDURI STRUCTURATE: initial si always
- Exista doua proceduri structurate in Verilog: initial si always;
- Sunt folosite pentru modelarea comportamentala;
- Fiecare bloc initial sau always reprezinta o activitate separata in Verilog;
- Fiecare activitate (bloc) incepe la timpul t=0.

Blocul initial:
Grupul de instructiuni din bloc se efectueaza o singura data.

Exemplu: Se utilizeaza 4 blocuri initial. Urmarim succesiunea instructiunilor.

```
module initial_example;

reg A, B, C, D, E;

initial
     A = 1;

initial
    begin
      #2  B = 0;
      #8  C = 1;
    end

initial
    begin
      #5  D = 1;
      #12  E = 0;
    end

initial
     #25  $finish;
```

Succesiunea temporala este urmatoarea (in pasi de timp t):
t = 0 A = 1;
t = 2 B = 0;
t = 5 D = 1;
t = 10 C = 1;
t = 17 E = 0;
t = 25 exit

Blocul always:
Grupul de instructiuni se efectueaza in bucla.

Exemplu: Generator de ceas.

```
module generator_ceas;

reg ceas;

// se initializeaza ceasul
ceas  = 1'b0;

// valoarea 'ceas' se schimba intre 0 si 1 cu o
// perioada de 4 pasi de intarziere
always
      #2   ceas = ~ceas

initial
      #100   $finish    // exit

endmodule
```

III. Elemente de teoria informatiei

- A fost dezvoltata de **Claude E. Shanon** si avand ca scop aflarea limitelor fundamentale cu privire la **procesarea semnalelor, comprimarea, stocarea si comunicarea datelor;**

- O modalitate de masura a informatiei este **entropia**, prin care se evalueaza incertitudinea in prezicerea unei valori corespunzatoare unei variabile aleatoare;

Exemplu: specificarea rezultatului la aruncarea unei monezi reprezinta infomatie mai putina decat specificarea rezultatului la aruncarea unui zar;

- Aplicatii:
  - lossless data compression (e.g. zip)
  - lossy data compression (MP3, JPEG)
  - channel coding (noisy channel coding theorem)

- Conceptele de baza se regasesc in comunicarea umana;

1). Cuvintele mai des folosite (“un”, “o” , “the” etc) ar trebui sa fie mai scurte decat cuvintele care nu sunt atat de des folosite (“imputernicire”, “mecanism” etc); aceasta repartitie a lungimilor cuvintelor in functie de frecventa de aparitie sta la baza compresiei de date si reprezinta aspectul esential din **source coding**;

2). Al doilea aspect este legat de fpatul ca, in prezenta zgomotului, mesajul trebuie sa fie rectionat cu cat mai mare acuratete; includerea unui grad suficient de robustete in transmisia mesajelor este esential in comunicare → se realizeaza in **channel coding**;

Observatie:
- in teoria informatiei nu se discuta “importanta” sau “semnificatia mesajelor” mesajelor; spre exemplu “ma duc sa ma plimb” sau “cheama o ambulanta” sunt mesaje de marimi similare, insa cu importanta diferita.
- asadar, nu se urmareste “calitatea” mesajelor”, cat cantitatea si capacitatea de a intelege mesajele;

- Fondata de Shanon in 1948 prin lucrarea "A Mathematical Theory of Communication"

- Paradigma centrala: transferul informatiei printr-un canal cu zgomot

- Rezultatele fundamentale ale teoriei:
  - **source coding theorem** : numarul de biti necesar pentru a reprezenta rezultatul unui eveniment incert este dat de entropie;
  - **noisy-channel coding theorem** : comunicatie robusta este posibila pe canale cu zgomot, daca rata comunicarii se face sub un anumit prag, numit capacitate a canalului;

In prezent, teoria informatiei este strans legata si alte discipline:
- sisteme adaptative;
- inteligenta artificiala;
- sisteme complexe;
- cibernetica ;

- informatica etc;

Vom urmari in continuare cateva aspecte:
- Definirea şi masurarea informaţiei;
- Codificarea informaţiei: coduri de lungime fixa, coduri de lungime variabila si eficienţa codificarii;
- Detectarea şi corectarea erorilor.

Domeniul ingineriei consideră ca informaţia inlatura/elimina incertitudinea. Informaţia este, in mod natural, “ceea ce nu se poate prezice”.

In cadrul procesului de edificare a domeniului teoriei informatiei s-au dat si precizat o serie de definitii formale privind continutul informational al unui mesaj (Hartley - 1928, Kolmogorov - 1942, Wiener - 1948, Shannon - 1949). Sub forma cea mai generala, informatia este considerata ca o inlaturare/eliminare a incertitudinii.

Se considera un ansamblu de *n* evenimente, cu probabilitatea de aparitie p_i. Incertitudinea *I* asociata sistemului este:

$$I = -\sum_{i=1}^{N} p_i \times \log_2(p_i)$$

Baza logaritmului a fost aleasa "2" pentru a exprima informatia in unitati de bit.

Un exemplu elocvent se poate imagina prin alegerea unui numar de la 1 la 16, cu probabilitate egala de aparitie, i.e. 1/16. Informatia asociata cu numarul ales se poate calcula cu relatia de mai sus:

$$I = -\sum_{i=1}^{16} \frac{1}{16} \times \log_2\left(\frac{1}{16}\right) = 4$$

Valoarea I = 4 obtinuta reprezinta numarul de intrebari de tip DA/NU necesare pentru a afla cu certitudine numarul ales.

Observatie: Conform relatiei Shannon, daca evenimentele au frecvente de aparitie diferite, atunci aparitia unui eveniment mai putin probabil aduce mai multe informatie, data de $i = \log_2(1/p_i)$;

Codificarea informatiei:

Codificarea informatiei se refera la modul in care aceasta este reprezentata, e.g. intr-un cod binar. Sunt cateva aspecte care se urmaresc in legatura cu eficienta si robustetea codificarii:

a). Dispozitivele / echipamentele necesare pentru a codifica / stoca informatia;

b). Eficienta codificarii : este data de numarul de biti necesari pentru a codifica un mesaj;

c). Fiabilitatea codificarii : rezistenta la zgomot;

d). Securitatea codificarii : este in stransa legatura cu metoda de criptare.

Pentru o secventa de evenimente cu diferite probabilitati de aparitie, o metoda eficienta de a realiza o codificare este *arborele lui Huffman*.

In acest caz, dimensiunea medie a codului, $<L> = \sum_i p_i * l_i$, poate fi comparata cu entropia Shanon. Cu cat <L> se apropie mai mult de entropia Shannon, codificarea este mai eficienta.

Exemplu: Arborele lui Huffman.

Detectia si corectia erorilor:

(a) Detectia erorilor.

- Se realizeaza de obicei prin utilizarea functiilor de tip hash sau printr-un algoritm de checksum;

- Exista o varietate larga a functiilor de tip hash; cateva sunt foarte raspandite pe motivul simplitatii lor cat si pentru faptul ca sunt adecvate unor probleme specifice (e.g. cyclic-redundancy check pentru erori de tip burst)

- Random error-correcting codes pot fi o alternativa la functiile de tip hash cand se doreste ca detectia unui numar minim de erori sa fie garantata;

- Exemple de coduri cu capacitate de detectie a erorilor: repetition codes, parity bits, cyclic redundancy check, cryptographic hash functions, care sunt detaliate in continuare.

- **Repetition codes**:
Ex.: 1101 este replicat de 3 ori → 110111011101
Daca se receptioneaza un cod cu un bit modificat intr-una din secvente, eroarea poate fi detectata si corectata.

- **Parity bits**:
Un bit “paritate” este adaugat mesajului binar; se pot detecta erori pe un numar impar de biti (1,3,5, …); la un nr. par de erori bitul paritate va fi calculat corect.

- **Cyclic redundancy check:**
Este bazat pe o functie tip hash non-secure; se imparte un polinom fixat la polinoame generate de datele de input, iar restul devine rezultat;

- **Two-out-five :**
Este o schema de codare care utilizeaza 5 biti si in care se folosesc fix 3 biti “0” si 2 biti “1”. Exista 10 variante prin care se se pot reprezenta cifrele 0-9. Se pot detecta erorile de tip single-bit, sau nr. impar de bits si unele erori cu nr. pare de biti (ex. Modificarea simultana a celor doi biti “1”).

- **Hamming codes**:
Prin adaugarea mai multor biti de tip error-correcting se pot identifica pozitiile bitilor cu erori. Ex: pentru un cod de 7-biti, o secventa error-correcting pe 3 biti poate localiza si pozitia erorilor de tip single-bit

(b). Corectia erorilor

Prin adaugarea bitului de paritate, distanta Hamming intre doua coduri (secvente binare) valide devine 2. Exemplu: 00 si 11 sunt coduri valide, in timp ce 01 si 10 sunt coduri in care au aparut erori.

Daca se mareste distanta Hamming la d_H, atunci se poate detecta pana la (d_H-1) erori la nivel de bit.

Daca distanta Hamming intre doua coduri este egala cu 3, erorile pe 1 bit se pot corecta. Exemplu: 000 si 111 sunt coduri valide, atunci, spre exemplu, 100 va fi corectat in 000, iar 101 va fi corecta in 111. In general, daca d_H este distanta Hamming intre doua coduri, atunci se pot corecta (d_H-1)/2 biti eronati.

Daca utilizam 4 biti pentru 3 biti de informatie, se pot genera 8 coduri cu distanta Hamming egala cu 1. Sunt 2 coduri care sunt separate de distanta Hamming 4. Ca atare, se pot corecta erorile pe 1 bit si se pot detecta erori pe 2 biti.

Operarea si organizarea unui sistem numeric

Principiile lui von Neumann:

Un calculator poseda urmatoarele elemente:
- *mediu de intrare*, pentru instructiuni si date;
- *memorie*, in care se stocheaza datele (program, date de intrare, rezultate);
- *ansamblu de prelucrare*, care efectueaza operatii aritmetice si logice, specificate in program;
- *mediu de iesire*, pentru a transfera rezultatele intr-o forma accesibila pentru utilizator;
- *element de decizie*, prin care se selecteaza dintre optiunile posibile, avand ca input rezultate partiale.

Circuite bistabile : latches / flip-flops

- Circuit care are doua stari stabile pentru a stoca informatia;

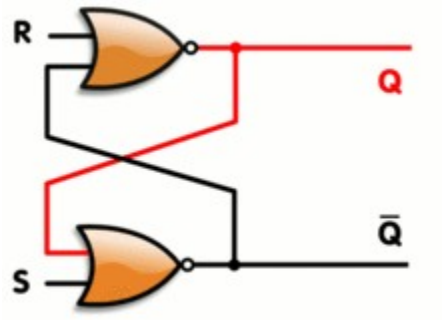
- Element de stocare de baza in logica secventiala;

- Schimbarea starii se face pe unul sau mai multe canale de intrare, avand una sau doua iesiri;

Terminologie:
- simple (transparente sau opace) → latches (zavoare)
  → level-sensitive
- clocked (synchronous / edge triggered) → flip-flops
  → edge sensitive

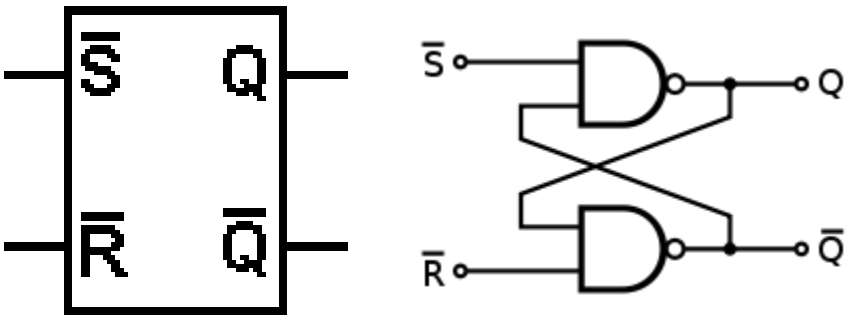
Exemple: SR NOR latch, SR NAND latch, JK latch

SR NOR latch:



Stare initiala: R = S = 0 ; Q = 1; !Q = 0
- reset → R = 1 → Q = 0; !Q = 1
- R = 0 → neschimbat, Q = 0 ; !Q = 1
- set → S = 1 → Q = 1; !Q = 0
- S = 0 → neschimbat, Q = 1; !Q = 0

SR NAND latch:



S negat	R negat	Rezultat
0	0	input invalid
0	1	Q = 1
1	0	Q = 0
1	1	fara schimbare

JK latch:

Este un latch SR, in care se inverseaza valoarea Q pentru input 11

J	K	Q	Rezultat
0	0	Q	fara schimbare
0	1	0	Reset
1	0	1	Set
1	1	Q negat	Toggle

Gated Latches. Transparenta conditionala.

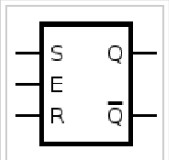
- Zavoarele (latches) sunt elemente transparente : o schimbare a semnalului la input cauzeaza imediat o modificare la output;

- Cand mai multe zavoare transparente se succed, semnalele sunt transmise prin toate in acelasi timp;

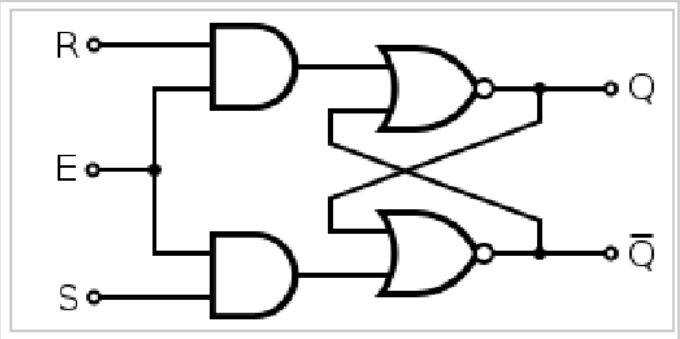
- Prin elemente de logica aditionala, se pot produce non-transparent (opaque) latches;

- Se pot asambla scheme mast
transparent-low; E = high – sen
→ transparent

E = low – latch closed (opaque)
→ ramane in starea curenta



Symbol for a gated SR latch

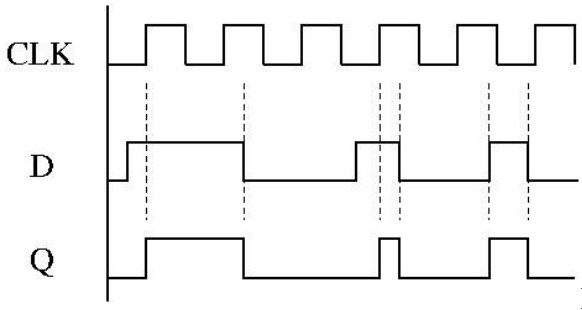


E	Rezultat
0	Fara schimbare
1	Similar cu un latch SR

Data latch (D-latch) : data, delay latch

Output-ul depinde de ceas:
- clock high : inputul trece in output
- clock low : se mentine output

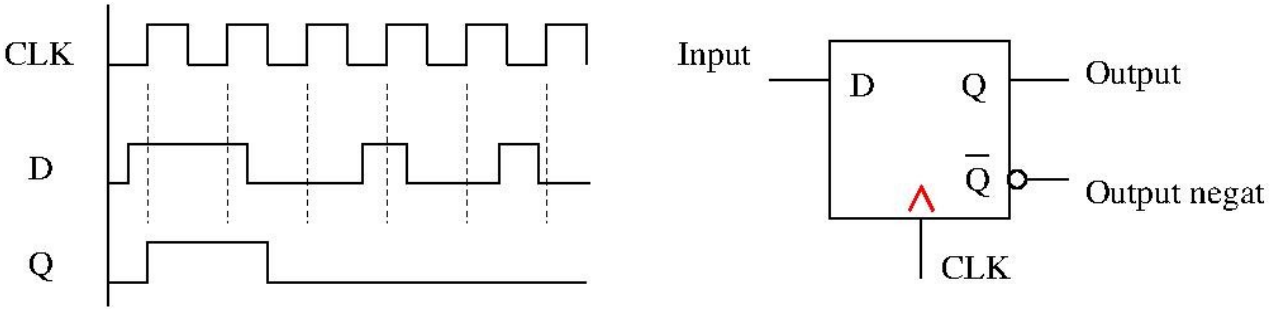
- Level sensitive and transparent



D flip-flop

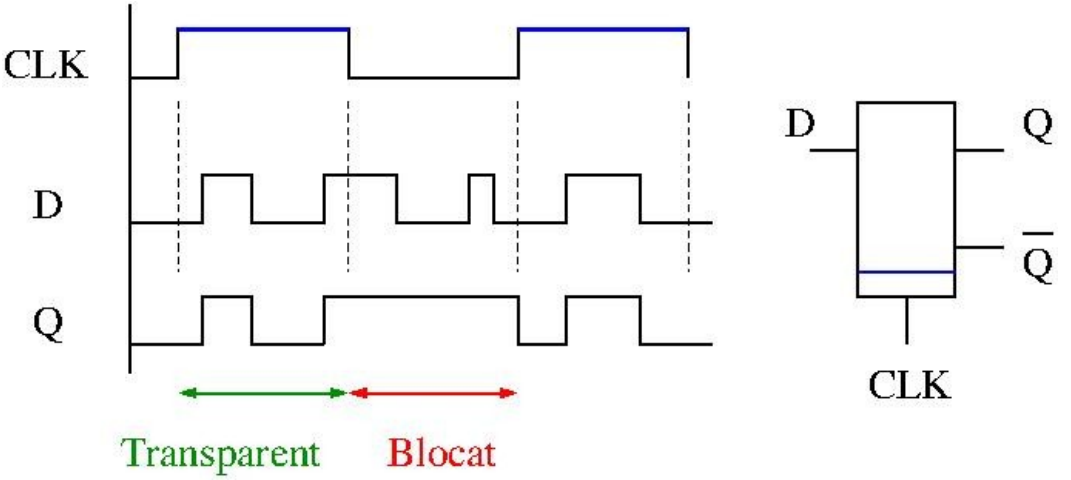
- Inputul depinde de schimbarea frontului de ceas
  - front crescator : input → output
  - altfel : isi pastreaza outputul

- Elementele de tip flip-flop functioneaza pe front crescator, descrescator sau in scheme de tip master-slave

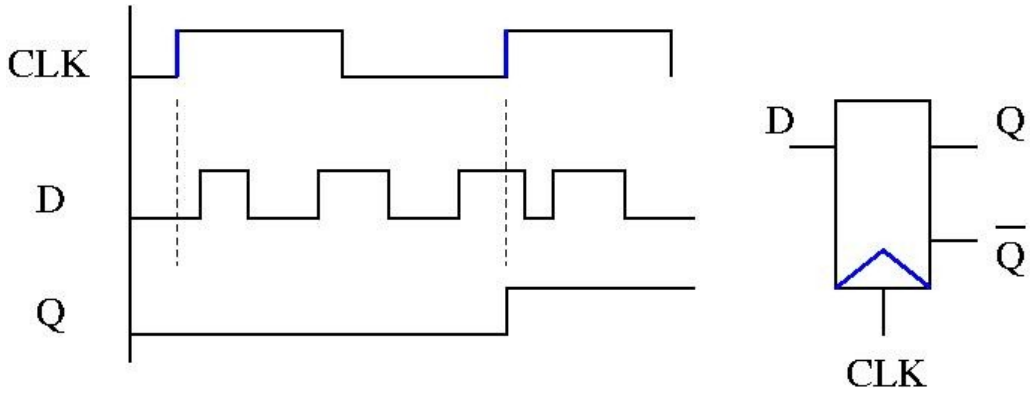


Exemple: D latches / flip-flops

Latches



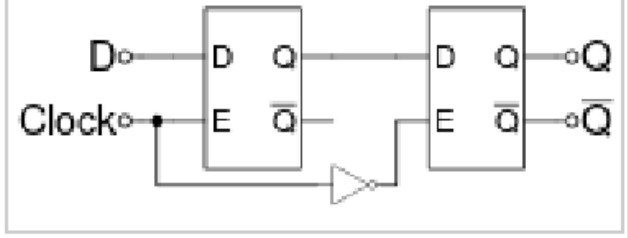
Flip-flops



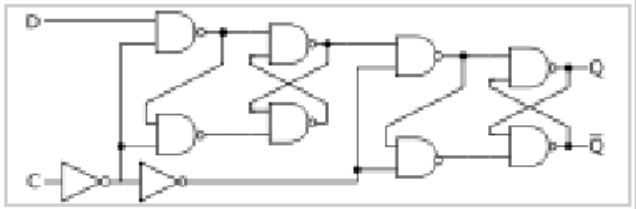
Master-slave edge triggered D flip-flop

- Realizat prin conectarea a doua elemente de tip gated D latch;

- al doilea latch in serie (slave) isi modifica outputul ca raspuns al primului latch (master)



A master–slave D flip-flop. It responds on the falling edge of the *enable* input (usually a clock)



An implementation of a master–slave D flip-flop that is triggered on the rising edge of the clock

T flip-flop operation[23]

Characteristic table				**Excitation table**			
T	Q	Q_{next}	**Comment**	Q	Q_{next}	T	**Comment**
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

- Este un flip-flop de tip SR (J=Set, K=Reset), la care se adauga conditia S=R=1, interpretata ca o comanda de tip toggle;

- J = 1, K = 0 set flip-flop;
- J = 0, K = 1 reset flip-flop;
- J = K = 1 toggle flip-flop;
- J = K = 0 mentine starea;
- K este complementul lui J → D flip-flop
- K = J → T flip-flop

→ JK flip-flop este universal (poate fi configurat in SR, D, T flip-flop)

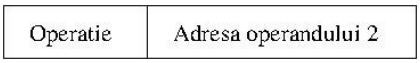
IV. Limbaje de asamblare (MIPS)

Limbajele de asamblare sunt limbaje de programare low-level in care instructiunile sunt operatii elementare, care includ transfer de date, control al programului, operaţii aritmetice şi logice, operaţii de deplasare pe biti, gestionare a întreruperilor. In acest context, instructiunile trebuie sa aibe o reprezentare unica si este de preferat ca acestea sa aibe o structura regulata.

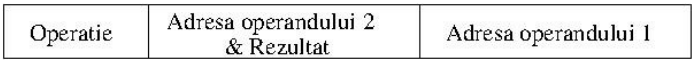
Formatul general al instructiunilor:

Instructiunile pot avea un format diferit, in functie de arhitectura sistemului numeric. Vom prezenta in continuare cateva tipuri de comenzi, mai des intalnite:

- Instructiunea cu un singur operand : Comanda contine doua campuri, i.e. codul operatiei si adresa operandului "2". Daca operatia este una binara (cu doi operanzi), primul operand este cunoscut, poate fi o constanta. Exemplu operatia aduna pe x la valorea constanta "5". Adresa rezultatului va fi inscrisa in campul al doilea.



- Instructiunea cu doi operanzi : De aceasta data, comanda contine trei campuri, i.e. codul operatiei, adresa operandului 1 (inainte de efectuarea operatiei) / rezultat (dupa efectuarea operatiei) si adresa operandului "2". Asfel, se pot implementa operatii cu doi operanzi. Dezavantajul consta in faptul ca valoarea primului operand se pierde in urma operatiei, campul fiind rescris cu adresa operandului "2".



- Instructiunea cu trei operanzi : Este cea mai uzuala, intrucat adesea folosim operatii cu doi operanzi. In acest caz, avem patru campuri, i.e. codul operatiei, adresa primului operand, adresa celui de al doilea operand si adresa rezultatului. In acest fel, se pastreaza informatia cu privire la primul operand, al doilea operand si rezultat.

In memoria calculatorului instructiunea se prezinta sub forma unui sir binar. Formatul unei instructiuni este specificat de arhitectura si ca atare, se pot identifica codurile binare asociate cu fiecare camp in parte.

Structura operatiei MIPS:

Sunt definite 6 campuri indicate mai jos:

OP – codul operatiei
RS – registrul sursa (primul operand)

RO – al doilea registru sursa (al doilea operand)
RD – registrul destinatie (rezultatul)
SHIFT – numarul de biti de deplasat intr-o instructiune de deplasare (shift)
FCT – selecteaza varianta operatiei definite in campul OP

Desi nu exista un format standard sau optim din toate punctele de vedere, in timp s-au cristalizat cateva principii de design:

a). Simplitatea şi regularitatea – de exemplu, in majoritatea operatiilor se gasesc 2 operanzi (E.g. operatii aritmetice)

b). Instructiunile mai scurte sunt mai rapide : Lugimea unui cuvand este de obicei 32 sau 64 biti.

c). Este de preferat ca instructiunile sa aibe aceeasi lungime. Atunci, instructiuni diferite pot avea formate diferite, realizand un compromis in stabilirea regularitatii.

Tipuri de date in MIPS

In limbajul de asamblare MIPS se utilizeaza urmatoarele tipuri de date : byte, halfword, word, float, double, space, a caror dimensiune este prezentata in tabelul de mai jos.

Tipul de date	Lungime
byte	intreg pe 8 biti
halfword	intreg pe 16 biti
word	intreg pe 32 biti
float	numar real (single precision) 32 biti
double	numar real (double precision) 64 biti
space	alocare spatiu in RAM

INSTRUCTIUNI MIPS:

1) Instructiuni aritmetice:

Instructiune	**Exemplu**	**Functionalitate**
adunare	add $t1, $t2, $t3	t1 = t2 + t3
scadere	sub $t1, $t2, $t3	t1 = t2 - t3
adunare la o constanta	addi $t1, $t2, 5	t1 = t2 + 5
adunare unsigned	addu $t1, $t2, $t3	t1 = t2 + t3
scadere unsigned	subu $t1, $t2, $t3	t1 = t2 - t3
adunare unsigned la o constanta	addiu $t1, $t2, 5	t1 = t2 + 5

2) Instructiuni logice:

Instructiune	Exemplu	Functionalitate
SI	and $t1, $t2, $t3	t1 = t2 & t3 (la nivel de bit)
SAU	or $t1, $t2, $t3	t1 = t2 | t3 (la nivel de bit)
SI IMEDIAT	andi $t1, $t2, 5	t1 = t2 & 5 (la nivel de bit)
SAU IMEDIAT	ori $t1, $t2, 5	t1 = t2 | 5 (la nivel de bit)
SHIFT LOGIC STANGA	sll $t1, $t2, 5	t1 = t2 << 5
SHIFT LOGIC DREAPTA	srl $t1, $t2, 5	t1 = t2 >> 5

3) Instructiuni prin care se transfera date din RAM in registrii si invers:

Instructiune	Exemplu	Functionalitate
incarca cuvant din RAM (load word)	lw $t1, 5($t2)	incarca in reg t1 valoare din RAM aflata la adresa $t2+5
incarca cuvant in RAM (store word)	sw $t1, 5($t2)	incarca in RAM aflata la adresa $t2+5 valoarea din reg $t1
incarca constanta in registru (load upper immediate)	lui $t1, 5	incarca constanta 5 in segmentul superior al lui $t1

4). Instructiuni de ramificare conditionata:

Instructiune	Exemplu	Functionalitate
branch on equal	beq $t1, $t2, 5	if ($t1==$t2) goto PC+4+5
branch on not equal	bne $t1, $t2, 5	if ($t1!=$t2) goto PC+4+5
set on less than	slt $t1, $t2, $t3	if ($t1<$t2) $t1=1; else $t1=0

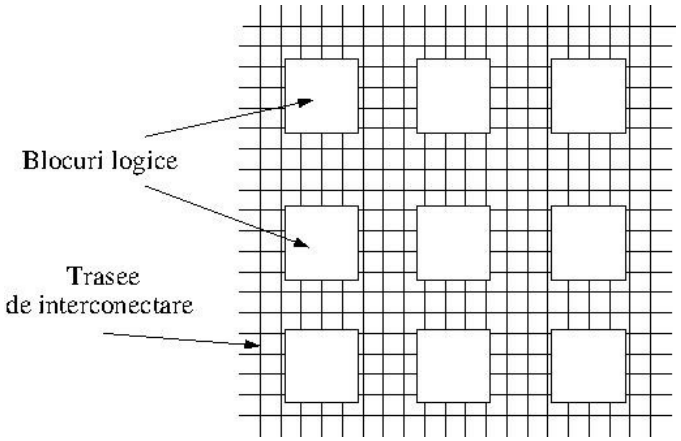
5). Instructiuni de salt neconditionat:

Instructiune	Exemplu	Functionalitate
jump	j 800	goto 800; salt la adresa 800
jump register	jr $31	goto $31; revine dintr-o functie (CR)
jump and link	jal 900	$31 = PC + 4; goto 800; apelare functie

V. Arii de porti programabile

(Field Programmable Gate Arrays - FPGAs)

Ariile de porti programabile sunt alcatuite din retele de blocuri logice si bistabile (flip-flops). Spre deosebire de un microprocesor, FPGA-urile pot fi programate hardware, modificand conexiunile dintre blocuri.



Este foarte relevanta o comparatie intre sisteme ASIC (Application Specific Integrated Circuits), Microprocesoare si FPGA, punctand avantaje si dezavantaje pentru fiecare arhitectura in parte.

Perfomanta	NRE	Cost	TTM
ASIC	ASIC	FPGA	ASIC
FPGA	FPGA	MICRO	FPGA
MICRO	MICRO	ASIC	MICRO

Astfel, perfomanta cea mai mare o au ASIC-urile, asa cum era de asteptat, intrucat ele sunt optimizate pentru o sarcina specifica. Prin posibilitatea de reconfigurare, FPGA-urile sunt mai performante de cat microprocesoarele, care emuleaza din soft diferiti algoritmi.

Pe de alta parte, NRE (Non Recurring Engineering) are valoarea cea mai mare la ASIC, deoarece o sarcina noua presupune multe modficari pe placa. Costurile sunt relativ mari petru FPGA, iar TTM (Time to Market) este unul mediu comparativ cu celelalte doua solutii.

O placa FPGA este alcatuita din urmatoarele elemente de baza:
(1) Blocuri logice programabile;
(2) Comutatoare programabile;
(3) Trasee de interconectare;

Sistemele FPGA se clasifica in functie de :
- modalitatea de programare;
- modul de organizarea al traseelor de interconectare;
- functiile blocurilor logice combinationale;

(1) Blocuri logice programabile.

Blocurile logice se pot realiza utilizand perechi de tranzistor NMOS si PMOS in comutatoare de tip T-gates, retele de porti logice (NAND etc), tabele asociative, multiplexoare etc.

Dimensiunea blocurilor logice poate varia. Intrucat blocurile mari implemenateaza mai multa logica, numarul acestora tinde sa scada pentru a implementa o functie data.

Aria ocupata de blocurile logice, raportata la aria totala este relativ mica, aproximativ 20 - 30 % din aria totala.

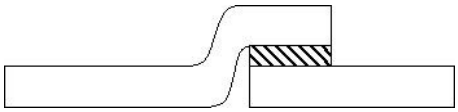
In FPGA-urile care contin tabele asociative, un numar de 4 intrari este o valoare optima, urmarind raportul arie utilizata pe logica implementata.

(2) Comutatoare programabile.

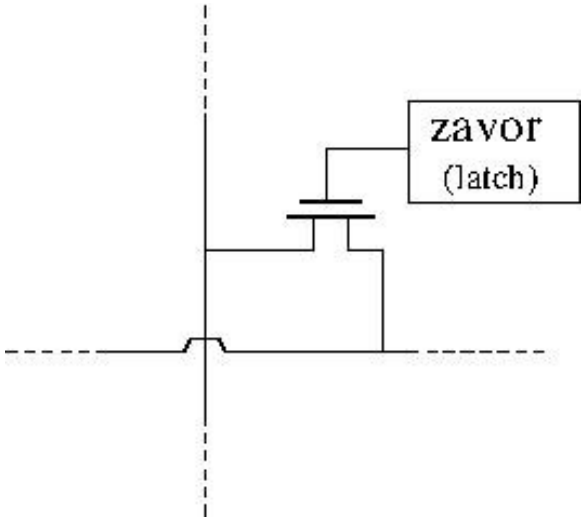
Conexiunile contin puncte de intersectie, fiind organizate in magistrale intersectate sau cross-bar.

Conexiunile pot fi permanente sau temporare.

Conexiunile permanente se realizarea prin aplicarea unei tensiuni mari care realizeaza un contact permanent. Aceste conexiuni nu se pot reprograma, insa au unele avantaje, cum ar fi: caracter nevolatil, rezistenta/capacitate mica, dimensiuni mici.



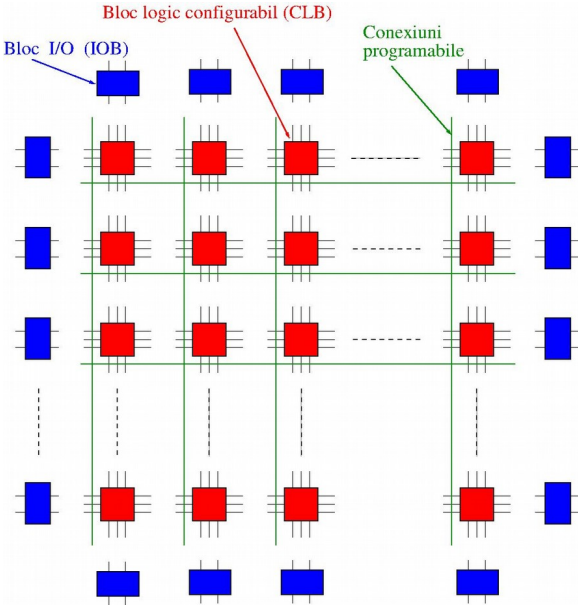
Conexiunile temporare se realizeaza cu ajutorul tranzistorilor. Avantajul principal consta in posibilitatea reconfigurarii, insa contrar conexiunilor permanente, dimensiunile conexiunilor sunt relativ mari si au caracter volatil.



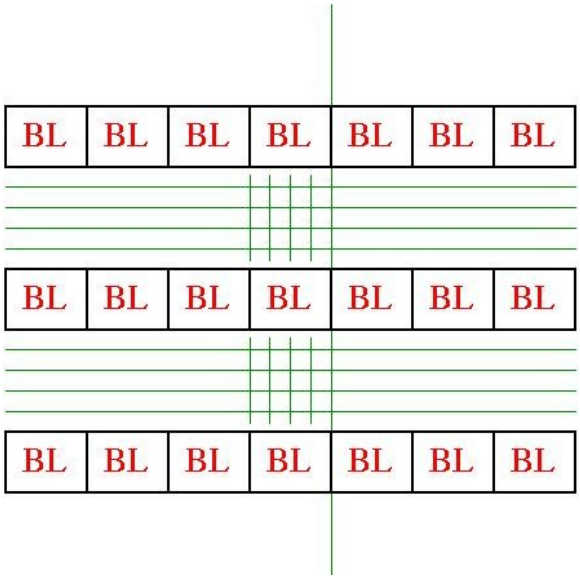
Cateva notiuni:
- *firele* reprezinta conexiuni directe, fara comutatoare intre doua puncte; se mai numesc si *segmente*;
- traseul este inlantuire de segmente, care contin si comutatoare.

(3) Trasee de interconectare / Tipuri de arhitecturi.

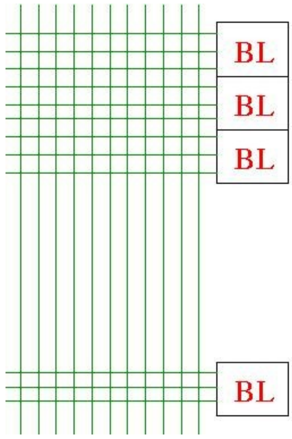
a). Arhitectura Xilinx: Traseele de interconectare inconjoara fiecare bloc logic combinational, la intersectiile traseelor fiind plasate blocuri de comutatoare. Acestea sunt organizate sub forma de matrici cu tranzistori de trecere controlate cu celule de memorie statica SRAM.

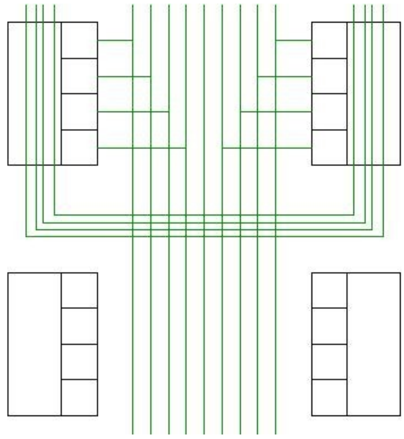


b). Arhitectura ACTEL : Firele sunt aranjate preponderent e orizontala in raport cu verticala. Blocurile logic-combinationale sunt organizate pe linii.



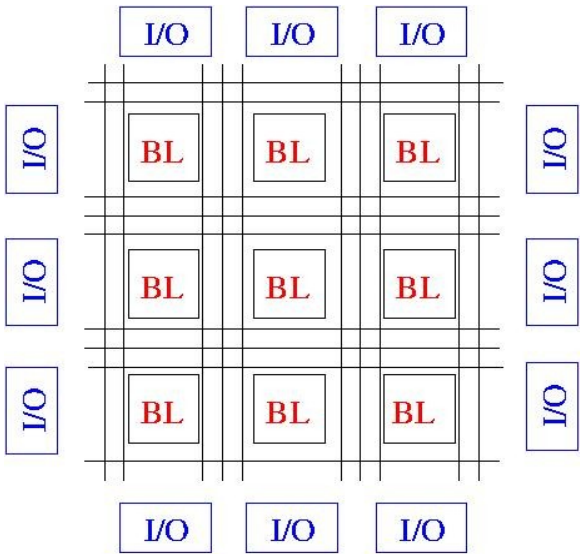
c). Arhitectura Altera : Interconectarea se face pe doua niveluri ierarhice, local si global.



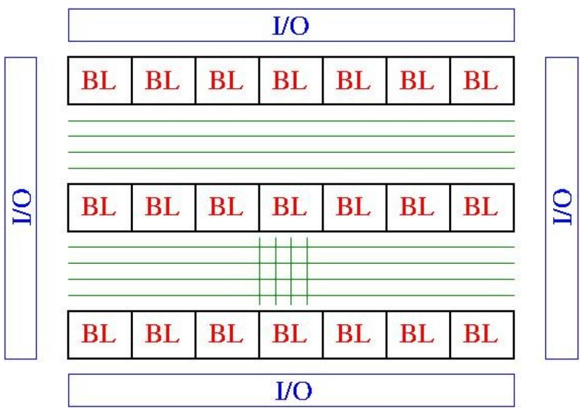


In functie de organizarea interna a blocurilor, sistemele FPGA se pot clasifica in:

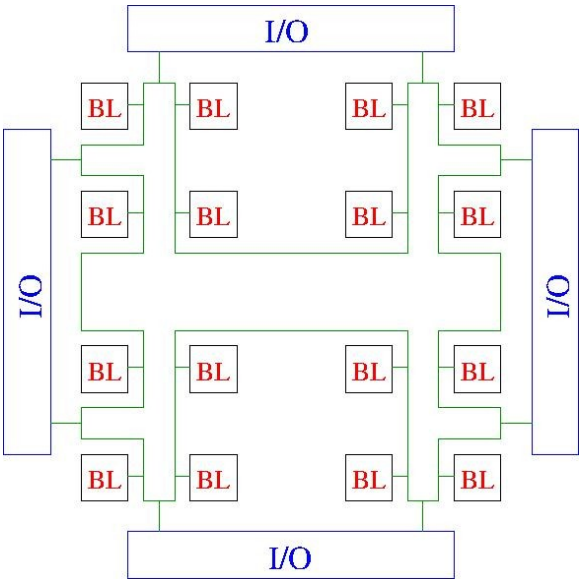
(a) Structuri simetrice : Blocurile logic combinationale sunt organizate matricial pe linii si coloane, avand intre ele blocuri de comutatoare programabile. Aceaste sunt inconjurate de conexiuni.



(b) Structuri organizate pe linii : Blocurile logic combinationale sunt organizate pe linii, alternand cu trasee orizontale de interconectare. Blocurile de input/output sunt plasate la margini (stanga/dreapta). Traseele orizontale se pot conecta prin fire verticale.



(c). Structuri ierarhice : Prezinta mai multe niveluri pentru blocurile logic combinationale, care sunt alcatuite din module logice, porti logice etc. Fiecare unitate are o unitate de memorie.



VII. Administrarea sistemelor de calcul numeric

1). Instalarea automatizata a sistemului de operare Linux

Odata cu dezvoltarea sistemelor de calcul distribuite, concomitent cu cresterea numarului de noduri de calcul a devenit imperios necesara instalarea automatizata a sistemelor de operare. Dat fiind faptul ca structura unui cluster de calcul poate fi heterogena (noduri cu proprietati diferite), nu este posibil intotdeauna ca sistemele de operare sa fie replicate printr-o simpla clonare a discurilor. O optiune mai buna, care asigura ulterior si mentenanta, este instalarea sistemelor de operare cu ajutorul unui server. In continuare sunt prezentate etapele configurarii unui server FAI (Fully Automated Install) pentru instalarea sistemului de operare Debian.

Configuratia sistemului de calcul numeric distribuit este urmatoarea: un nod central (server FAI) permite accesul intr-o subretea. Nodul central dispune de doua placi de retea: (1) pentru a realiza conexiunea cu mediul extern (retea privata, acces internet) si (2) pentru a realiza conexiunea cu masinile de calcul utilizand un switch. Nodurile de calcul sunt conectate la serverul FAI prin acest switch.

Pasul 1:
Pe nodul central este instalat sistemul de operare Debian. Pachetele necesare instalarii serverului FAI se gasesc in distributie. Aceastea se instaleaza, urmand ca apoi sa configuram serverul FAI. Acesta are asignate doua IP-uri: (1) 192.168.0.200 in reteaua privata din care face parte serverul si (2) 192.168.20.1 care reprezinta IP-ul de poarta pentru reteaua nodurilor de calcul avand IP-uri 192.168.20.x ($1< x < 255$).

Pasul 2:
Pe nodul central se instaleaza un server DHCP. Acesta este configurat sa seteze IP-urile nodurilor de calcul, asociind un IP stabilit pentru o adresa fizica (MAC).

Pasul 3:
Se stabileste politica de rutare pe nodul central. Acesta trebuie sa asigure accesul nodurilor de calcul catre reteaua externa (internet), fara a permite accesul dinspre exterior.

Pasul 4:
Se configureaza serverul pentru a instala automat sisteme de operare pe clienti (nodurile de calcul).

a). In directorul /srv/fai/config/class modificam fisierul *50-host-classes

E.g. clu47)
echo "FAIBASE DEBIAN";;

unde "clu47" reprezinta numele clientului (nodul de calcul 47).
FAIBASE reprezinta o selectie (minimala) de pachete.

b). Se modifica partitiile

/srv/fai/config/disk_config/FAIBASE

c). Se adauga clase noi pentru a instala alte pachete

/srv/fai/config/package_config/

e.g. UTILS, SCIENCE

care se adauga in scriptul 50-host-classes
echo "FAIBASE DEBIAN UTILS SCIENCE";;

EXEMPLU:
O posibila configurare a claselor UTILS si SCIENCE:

```
# system
nfs-common
ntp

# utils
vim
mc

# science
gsl-bin             # GSL
openmpi-bin         #
libblas3            #
libblas-dev         #
liblapack3          #
liblapack-dev
libatlas3-base      #
libatlas-dev
```

libblacs-openmpi1
libscalapack-openmpi1 #

compilers
gcc
g++
gfortran

graphical interface
gnome

X programs
grace
gnuplot
xcrysden

other
autodock
lammps

Pasul 5:
Pentru a instala sistemul de operare pe client se ruleaza pe server

fai-chboot -IFv -u nfs://192.168.10.1/srv/fai/config clu47

unde clu47 este hostname in *50-host-classes .

Se genereaza astfel fisierul e.g. "C0A80A65" in /srv/tftp/fai/pxelinux.cfg .

Serverul este acum pregatit pentru a demara instalarea pe client.
Acesta va fi necesar si pentru updates.

Pasul 6:
Clientul este setat in BIOS sa booteze prin retea. Se booteaza clientul si se sistemul de operare este instalat.

Atentie!
Instalarea sistemului de operare pe client se face automat conform fisierelor de configurare, fara a necesita alte validari. Prin urmare, este posibil sa modificam configurarea doar dupa ce pachetele sunt instalate. Daca masina are un sistem de operare sau partitii pre-existente acestea vor fi supra-scrise automat fara avertisment!

Cateva elemente pentru siguranta (post-install):
1). Se modifica ordinea de boot la client: mai intai HDD, ..., ultima optiune fiind LAN (care se poate dezactiva)
2). Serverul FAI se poate dezactiva sau fisierele de configurare de pe serverul fai
din /srv/tftp/fai/pxelinux.cfg se pot muta intr-un subdirector creat special (e.g. "OFFLINE"). Acestea trebuie pastrate pentru update-uri ulterioare.

Update-uri de software (softupdate):

Se instaleaza pe client "nfs-common" pentru a avea acces la partitia nfs de pe serverul FAI care contine fisierele de configurare.

Se adauga e.g. in clasa UTILS noile pachete

Se ruleaza pe client
fai -v -s nfs://192.168.10.1/srv/fai/config softupdate

Ghidul complet se gaseste la adresa web:

https://fai-project.org/fai-guide/

BIBLIOGRAFIE:

[1] *https://www.webopedia.com/DidYouKnow/Hardware_Software/FiveGenerations.asp*
https://turbofuture.com/computers/Classification-of-Computers-by-Generation
[2] *Verilog HDL, A guide to digital design and synthesis*, Samir Palnitkar, SunSoft Press, 1996.
[3] *Calculatoare numerice I, A. Petrescu, C.Popescu, N. Popescu, Ed. Printech.*
[4] *A Mathematical Theory of Communication*, C.E. Shannon, Reprinted with corrections from The Bell System Technical Journal, vol. 27, pp. 379–423, 623–656, July, October, 1948.
[5] *MIPS Assembly Language Programming using QtSpim*, Ed Jorgensen, 2014.
[6] *Architecture of Field Programmable Gate Arrays*, Jonathan Rose et al., Proceedings of the IEEE vol. 81, pag. 1013, 1993.
[7] *https://fai-project.org/fai-guide/*