

3.Ce reprezinta Verilog HDL?

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

Limbajul Verilog descrie un sistem numeric ca un set de module. Fiecare dintre aceste module are o interfață cu alte module, pentru a specifica maniera în care sunt interconectate. Modulele operează concurențial.

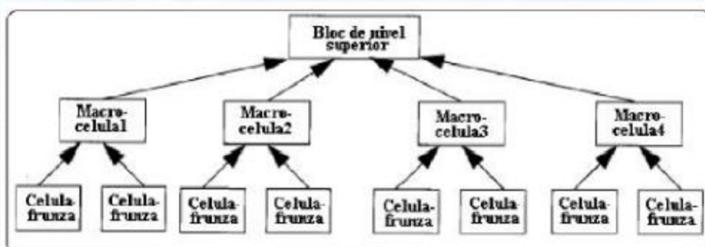
Modulele reprezintă parti hardware, care pot fi de la simple porti până la sisteme complete cum ar fi un microprocesor.

4.Descrieti stilul de proiectare bottom-up / top-down.

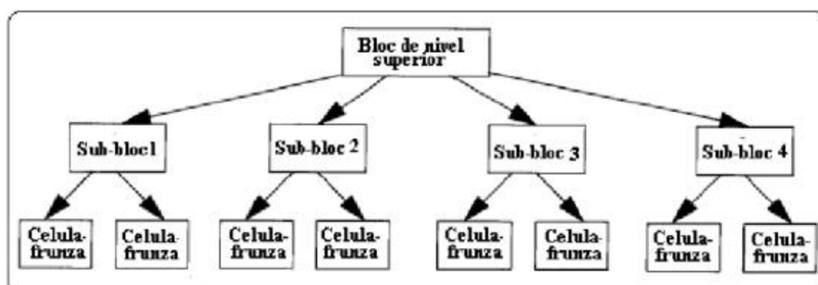
Verilog HDL – Stiluri de proiectare

Verilog permite ca la orice alt limbaj de descriere hardware, utilizarea ambelor metodologii: "bottom-up" și "top-down"

Bottom-up: metoda tradițională; proiectul este realizat la nivel de porti → dificultate în a reprezenta structuri de milioane de tranzistori → se grupează elementele la macronivel



Top-down: metoda actuală; se pleacă de la specificațiile de sistem și se detaliază în jos până la componentele de sistem. Există avantaje în ceea ce privește schimbarea tehnologiilor, proiectarea structurată.



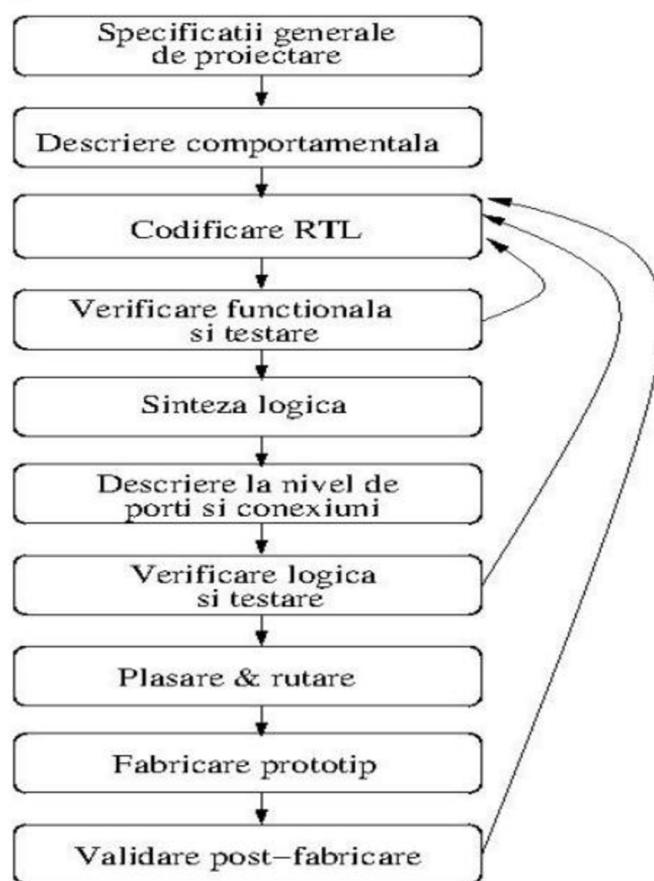
5. Indicati modalitatile de reprezentare hardware. + 6. Descrieti nivelurile ierarhice in modalitatea de reprezentare comportamentalala, structurala, fizica, abstracta

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

Descriere comportamentalala	Descriere structurala	Descriere la nivel de procese fizice	Descriere abstracta
<ul style="list-style-type: none">- Aplicatii software;- Sistem de operare;- Programe utilizator;- Functii, subrutine, instructiuni.	<ul style="list-style-type: none">- Procesor, placa de baza, placa video, memorie (RAM, HDD);- Sumatoare, porti, registrii;- Tranzistori, diode, condensatori etc.	<ul style="list-style-type: none">- Curenti de sarcina in circuite, dispozitive;- Functionarea elementelor de circuit: tranzistori, diode etc.	<ul style="list-style-type: none">- Arhitectura generica;- Algoritmi;- Functionalitatea modulelor;- Logica de comutatie;

7. Descrieti fluxul de proiectare Verilog HDL.

Diagrama fluxului de proiectare:



8. Ce presupune etapa de ...?

Verilog HDL : Fluxul de proiectare

Specificatii de proiectare:

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

Descriere comportamentală:

Este indicata descrierea comportamentală prin intermediul căreia este analizat sistemul din punct de vedere al funcționalității, performanței, al compliancei cu standardele și sunt considerate alte aspecte de nivel înalt. Astfel de descrieri pot fi implementate folosind limbaje de descriere hardware (Hardware Description Languages, HDLs).

Codificarea RTL:

Descrierea comportamentală este convertită în descriere RTL prin intermediul unui HDL. Este descris fluxul de date care implementează circuitul digital. Din acest punct, procesul de design este asistat de calculator (Computer-Aided design, CAD).

Sinteza logica:

Tool-urile de sinteză logică convertează descrierea RTL într-o descriere la nivel de porturi și conexiuni. Aceasta constituie un input pentru etapa următoare de placare și rutare.

Placarea și rutarea:

Este realizată dispunerea elementelor de circuit pe placă, în funcție de constrângeri (e.g. surse de tensiune, cablaj etc)

Validare postfabricare:

Se realizează un prototip care este testat în condiții reale. Această etapă precede lansarea produsului pe piață.

7. Verilog HDL

- indicati structura unui cod; semnificatia blocurilor *initial* si *always*;
- ce reprezinta un modul de test;
- semnificatia simbolului # ; functiile pot contine intarzieri?
- cand folosim registrii (reg) si cand folosim fire (wire) ?
- instructiuni blocante vs. instructiuni non-blocante

Structura unui program este urmatoarea:

- Modul main (de test);
- Modul sau module top-level;
- Submodul 1;

.....
- Submodul n;

Un modul de test este un modul de nivel inalt, care stabileste anumite seturi de date si care asigura monitorizarea variabilelor.

Structura unui modul este urmatoarea:

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

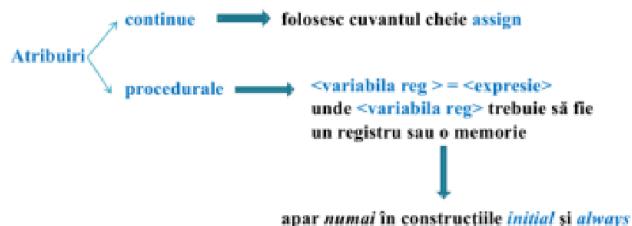
<nume_modul> reprezinta un identificator care, in mod unic, denumeste modulul.

<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 registre (reg), memorii si fir (wire), cat si constructiile procedurale ca function-s si task-s

<obiecte ale modulului> poate contine: constructii initial, constructii always, atribuirile continue sau aparitii/instante ale modulelor.

- 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.



- Variabilele reg stocheaza ultima valoare, care le-a fost atribuită procedural.
- Variabilele wire reprezintă conexiuni fizice între entități structurale cum ar fi porturile. 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:

Instrucțiunile din blocul construcției ***initial*** vor fi executate sevențial, dintre care unele vor fi întârziate de #1 cu o unitate de timp simulaț.

Construcția ***always*** se comportă în același mod ca și construcția ***initial*** cu excepția că ea ciclează la infinit (până la terminarea simulării).

- atribuirea **procedurală** modifică starea unui registru, adică a logicilor sevențiale
- atribuirea **continuă** este utilizată pentru a modela logica combinațională. Atribuirile continue comandă variabile de tip **wire**

Blocurile ***initial*** și ***always*** au aceeași construcție dar diferă prin comportare:

- ❖ blocurile ***initial*** sunt utilizate pentru inițializarea variabilelor, pentru efectuarea funcțiilor legate de aplicarea tensiunii de alimentare, pentru specificare stimulilor inițiali, monitorizare, generarea unor forme de undă;
- ❖ un bloc ***initial*** se execută o singură dată; după terminarea tuturor instrucțiunilor din blocul dat, fluxul ia sfârșit, fiind reluat odata cu simularea;
- ❖ blocurile ***always*** sunt utilizate pentru a descrie comportamentul sistemului;
- ❖ un bloc ***always*** este executat în mod repetat, într-o buclă infinită până la terminarea simulării specificată printr-o funcție sau task de system: \$finish, \$stop.
- ❖ este important ca blocul ***always*** să conțină cel puțin o instrucțiune cu întârziere sau controlată de un eveniment, în caz contrar blocul se va repeta la timpul zero, blocând simularea.

O funcție Verilog:

- asemanatoare cu un task, cu putine diferențe:
- funcția nu poate avea decât o singura ieșire;
- o funcție începe cu cuvântul cheie **function** și se termină cu **endfunction**;
- funcțiile sunt definite în modulul în care sunt utilizate; este posibil ca funcțiile să fie definite în fisiere separate și să se folosească directiva de compilare “**include**”
- funcțiile nu pot conține întârzieri și nu pot conține sincronizări de tip **posedge**, **negedge**, **#**, **@** sau **wait** → funcțiile se execută cu întârziere zero;
- simbolul **#** reprezintă un control al întârzierii, iar simbolul **@** reprezintă un eveniment de control (ex: **#10A = A + 1;** → specifică o întârziere de 10 unități de timp înainte de a executa instrucțiunea de asignare procedurală)

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 doua 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 $a = 7$.

10. Aplicatii ale teoriei informatiei. Definiti entropia lui Shanon.

- Teoria informatiei a fost dezvoltata de **Claude E. Shanon** si a avut 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 (ex: specificarea rezultatului la aruncarea unei monezi reprezinta infomatie mai putina decat specificarea rezultatului la aruncarea unui zar);
- In prezent, teoria informatiei este strans legata de discipline precum: sisteme adaptative, inteligenta artificiala, sisteme complexe, cibernetica, informatica etc;

Aplicatii:

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

- » Dacă un sistem se caracterizează prin **n** evenimente cu probabilități egale de apariție, în condițiile în care s-au obținut informații, care au redus cele **n** evenimente la **m** evenimente s-au obținut: $\log_2\left(\frac{n}{m}\right)$ biți de informație.

- » Entropia este cantitatea medie de informatie conținută într-un sir de date.

$$\text{entropia} = \sum_{i=1}^N \left(\frac{M_i}{N} \right) \cdot \log_2 \left(\frac{N}{M_i} \right)$$

unde:

- $\left(\frac{M_i}{N} \right)$ - reprezintă probabilitatea mesajului **i**
- $\log_2 \left(\frac{N}{M_i} \right)$ - constituie informația din mesajul **i**

11. Calculati informatia medie (in biti) avand in vedere evenimentele A1, A2,..., cu probabilitatile de aparitie si codurile asociate urmatoare (...).

- Se consideră un sistem, care reprezintă o mulțime formată din **n** obiecte, având proprietatea că fiecare obiect **i** posedă o probabilitate independentă **p_i** de apariție. Incertitudinea **H**, asociată acestui sistem, este definită ca:

$$H = - \sum_{i=1}^n p_i \times \log_2(p_i)$$

- Se presupune existența unei urne cu bile numerotate de la 1 la 8. Probabilitatea de a extrage o cifră dată, în urma unei trageri, este egală cu 1/8. Incertitudinea/informația medie asociată cu numărul selectat poate fi calculată cu ajutorul formulei de mai sus:

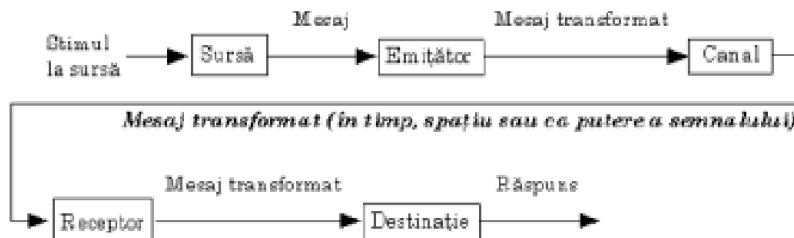
$$H = \sum_{i=1}^8 \left(\frac{1}{8} \times \log_2 \left(\frac{1}{8} \right) \right) = - \log_2 \left(\frac{1}{8} \right) = \log_2 8 = 3$$

- În orice schemă de codificare, care reprezintă o mulțime cu n elemente, cu probabilități egale de selectare, cel puțin unul din coduri trebuie să aibă o lungime egală sau mai mare decât măsura informației asociată mulțimii date, adică:

$$H = - \sum_{i=1}^n p_i \cdot \log_2(p_i)$$

Astfel, în cazul unui sistem fizic, care se poate afla în 13 stări distincte, codificarea stărilor se va realiza cu mesaje având lungimea de 4 biți. Mesajele de 4 biți lungime vor putea codifica 16 stări distincte.

În teoria comunicațiilor se consideră că mesajele recepționate, dar incomplet înțelese, conțin zgomot. Diagrama transmiterii semnalelor, după Shannon, este următoarea:



- Dacă un sistem se caracterizează prin n evenimente cu probabilități egale de apariție, în condițiile în care s-au obținut informații, care au redus cele n evenimente la m evenimente s-au obținut: $\log_2\left(\frac{n}{m}\right)$ biți de informație.

- Entropia este cantitatea medie de informație conținută într-un șir de date.

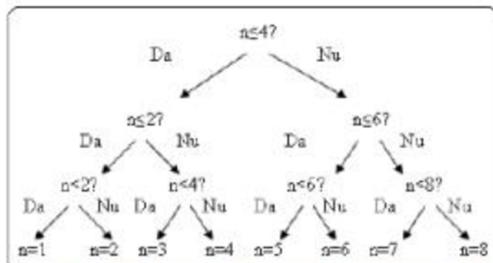
$$\text{entropie} = \sum_{i=1}^N \left(\frac{M_i}{N} \right) \cdot \log_2 \left(\frac{N}{M_i} \right)$$

unde:

$\left(\frac{M_i}{N} \right)$ – reprezintă probabilitatea mesajului i

$\log_2 \left(\frac{N}{M_i} \right)$ – constituie informația din mesajul i

12. Descrieti arborele lui Huffman pentru sevenanta de evenimente urmatoare (...).



Interogări și răspunsuri binare privind selectarea unei bile numerotate dintr-o urnă care conține 8 bile.

Numărul bilei	Schema 1 a mesajului	Schema 2 a mesajului
1	000	001
2	001	010
3	010	011
4	011	100
5	100	101
6	101	110
7	110	111
8	111	000

Schemele posibile de codificare pentru cele opt numere inscrise pe bilele din urnă

În situațiile când evenimentele au probabilități diferite de apariție, se obține mai multă informație atunci când se produce un eveniment cu probabilitate mică de apariție, decât în cazul producerii unui eveniment cu probabilitate mare de apariție.

❖ Informația furnizată de apariția evenimentului : $i = \log_2 \left(\frac{1}{p_i} \right)$ biți

❖ Entropia informației este : $\sum p_i \times \log_2 \left(\frac{1}{p_i} \right)$

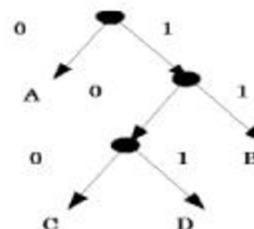
Exemplul 1:

Se consideră 4 evenimente A,B,C,D, cu probabilitățile de apariție și codurile asociate, conform tabelei de mai jos:

Eveniment	A	B	C	D
Prob. Apariție (p_i)	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$
Codificare	0	11	100	101

$$\text{Informația medie} = 0,5 \times 1 + 0,25 \times 2 + 2 \times 0,125 \times 3 = 1,750 \text{ biți}$$

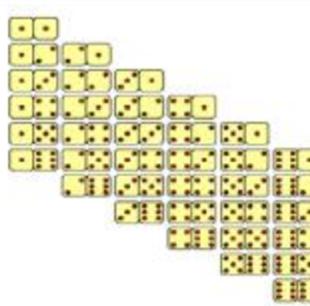
O schema de decodificare a unui mesaj ce conține codurile unei secvențe de evenimente A, C, D, A, B, C etc, se conformează următoarei structuri arborescente (arborele de decodificare Huffman):



Exemplul 2:

Se consideră suma a două zaruri, în sensul evaluării conținutului de informație existent în suma obținută la o aruncare.

Sumă	Possibilități
2	1+1
3	1+2, 2+1
4	1+3, 2+2, 3+1
5	1+4, 2+3, 3+2, 4+1
6	1+5, 2+4, 3+3, 4+2, 5+1
7	1+6, 2+5, 3+4, 4+3, 5+2, 6+1
8	2+6, 3+5, 4+4, 5+3, 6+2
9	3+6, 4+5, 5+4, 6+3
10	4+6, 5+5, 6+4
11	5+6, 6+5
12	6+6



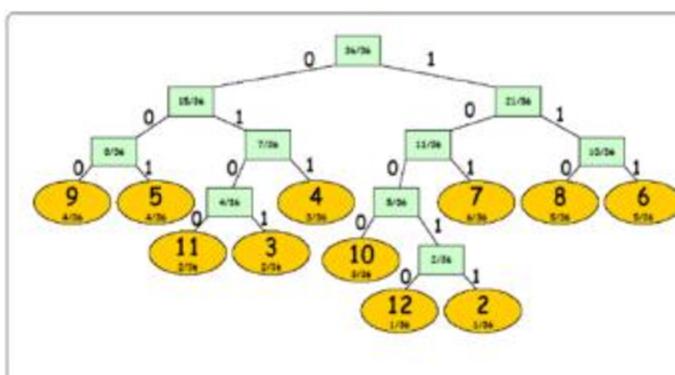
$$i_2 = \log_2 \frac{36}{1} = 5,170 \quad i_3 = \log_2 \frac{36}{2} = 4,170 \quad i_4 = \log_2 \frac{36}{3} = 3,585 \quad i_5 = \log_2 \frac{36}{4} = 3,170 \quad i_6 = \log_2 \frac{36}{5} = 2,848$$

$$i_7 = \log_2 \frac{36}{6} = 2,585 \quad i_8 = \log_2 \frac{36}{7} = 2,048 \quad i_9 = \log_2 \frac{36}{8} = 3,170 \quad i_{10} = \log_2 \frac{36}{9} = 3,585$$

$$i_{11} = \log_2 \frac{36}{10} = 4,170 \quad i_{12} = \log_2 \frac{36}{11} = 5,170$$

$$i_{med} = \sum_{j=1}^{12} \left(\frac{M_j}{N} \times \log_2 \left(\frac{N}{M_j} \right) \right) = \sum_{j=1}^{12} \left(p_j \times \log_2 \left(\frac{1}{p_j} \right) \right) = 3,275$$

Construirea arborelui binar



Arborele de decodificare Huffman : 2 - 10011; 3 – 0101; 4 – 011; 5 – 001; 6 – 111; 7 – 101; 8 – 110; 9 – 000; 10 – 1000; 11 – 0100; 12 - 10010.

13. Stabiliti dimensiunea medie a codului pentru secventa de evenimente urmatoare (...).

- ▶ Eficiența unei metode de codificare poate fi măsurată prin diferența între conținutul informational (entropia) al unui sir de simboluri și dimensiunea medie a codului.
- ▶ Dimensiunea medie a codului stabilit pentru sumele obținute la aruncarea a două zaruri se calculează astfel:

$$d_{med} = \frac{1}{36} \times 5 + \frac{2}{36} \times 4 + \frac{3}{36} \times 3 + \frac{4}{36} \times 3 + \frac{5}{36} \times 3 + \frac{5}{36} \times 3 + \frac{4}{36} \times 3 + \frac{3}{36} \times 4 + \frac{2}{36} \times 4 + \\ + \frac{1}{36} \times 5 = 3,306$$

- ▶ Rezultatul obținut se apropie destul de mult de informația medie (3,275)

14. Exemple de coduri cu capacitate de detectie a erorilor: repetition codes, parity bits, cyclic redundancy check, hash functions .

- Repetition codes:

Ex.: 1101 este replicat de 3 ori → 110111011101

Dacă se receptionează un cod cu un bit modificat într-o din secvențe, eroarea poate fi detectată și corectată.

- Parity bits:

Un bit "paritate" este adăugat mesajului binar; se pot detecta erori pe un număr 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 funcție 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 utilizează 5 biti și în care se folosesc fix 3 biti "0" și 2 biti "1". Există 10 variante prin care se pot reprezenta cifrele 0-9. Se pot detecta erorile de tip singlebit, sau nr. impar de bits și unele erori cu nr. par de biti (ex. Modificarea simultană a celor doi biti "1").

- Hamming codes:

Prin adăugarea mai multor biti de tip error-correcting se pot identifica pozițiile bitilor cu erori. Ex: pentru un cod de 7-bit, o secvență error-correcting pe 3 biti poate localiza și poziția erorilor de tip single-bit

```
for(i = 0; i < 10; i = i + 1)
begin
    Sdisplay("i= %0d", i);
end

i = 0;
while(i < 10)
begin
    Sdisplay("i= %0d", i);
    i = i + 1;
end

repeat (5)
begin
    Sdisplay("i= %0d", i);
    i = i + 1;
end
```

13. Cum se poate utiliza distanta hamming (minima) intre doua coduri valabile pentru a detecta si corecta erorile ?

Prin adaugarea bitului de paritate, distanta Hamming intre doua coduri (secvențe 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 marestea 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.

15. Operarea si organizarea unui sistem numeric. Principiile lui von Neumann.

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 lucru*, care efectueaza operatii aritmetice si logice, specificate in program;
- *mediu de ieșire*, 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.

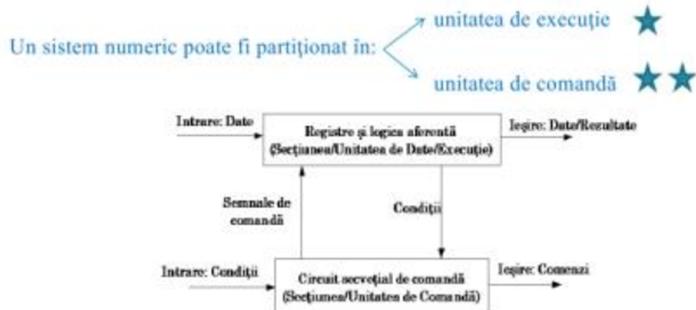
16.Ce este un algoritm? Descrieti principalele elemente ale unui algoritm.

Un algoritm reprezinta un set finit de reguli, care precizeaza o secventa de operatii, pentru solutarea unei clase de probleme.

Un algoritm posedă cinci elemente mai importante:

- ★ *caracter finit*: trebuie să se termine după un număr finit de pași;
- ★ *caracter determinist*: fiecare pas al unui algoritm trebuie definit în mod precis, acțiunile care se execută trebuie să fie specificate riguros, fără ambiguități, pentru fiecare caz; execuția algoritmului cu același set de date de intrare trebuie să conducă la același rezultat;
- ★ *intrare*: un algoritm are una sau mai multe intrări, reprezentând datele inițiale;
- ★ *ieșire*: un algoritm are una sau mai multe ieșiri, care reprezintă rezultatele, aflate într-o anumită relație cu intrările;
- ★ *eficacitate*: un algoritm trebuie să se execute exact și într-un interval finit de timp.

17. Descrierea unui sistem numeric prin partitionarea în secțiunea de date (unitate de execuție) și secțiunea de comandă (unitate de comandă).



★ asigură prelucrarea datelor, reprezentate sub formă unor vectori binari, în cadrul transferului acestora între registrele sursă și registrele destinație. Transferul se efectuează prin intermediul unor rețele/circuite logice combinaționale.

★ furnizează, pentru secțiunea de date, diverse semnale de comandă, sincrone cu ceasul sistemului.

18. Indicati diferența intre *logica combinationala* si *logica sequentiala*. Exemple.

Logica combinationala vs. sesequentiala

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

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

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

Exemplu de logica sesequentiala: telecomanda – butoanele “channel up” și “channel down”; canalul selectat depinde de input-urile anterioare care determină starea curentă; canalul selectat depinde de input-ul up/down și de stare.

26. Indicati restrictia pentru drumul cel mai lung sau calea cea mai lenta.

Restricția pentru drumul cel mai lung sau calea cea mai lentă:

$$T_{ciclu} \geq T_{CQmax} + T_{pmax} + T_{stabilire/setup}$$

27. Restrictia pentru drumul cel mai scurt sau calea cea mai rapida.

Restricția pentru drumul cel mai scurt sau calea cea mai rapidă:

$$T_{CQmin} + T_{pmin} \geq T_{mentinere/hold}$$

28. Ce reprezinta notiunea de clock skew / jitter.

Comparatie intre skew si jitter

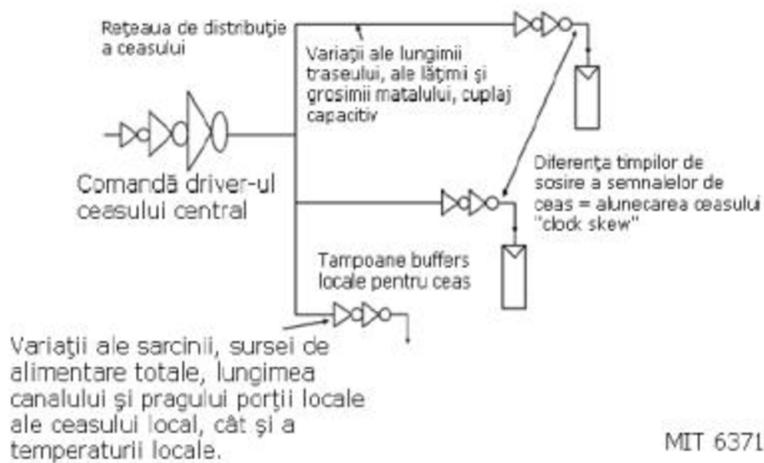
▪ **Skew-ul** reprezintă o variație spațială a timpilor de sosire a semnalelor de ceas: variația în ceea ce privește *același* front de ceas, văzut de către două sau mai multe bistabile *diferite*.

▪ **Jitter-ul** constituie variația temporală a timpilor de sosire: variația în ceea ce privește timpii de sosire a două fronturi succesive ale semnalului de ceas la *același* bistabil.

▪ Zgomotul sursei de alimentare reprezintă cauza principală a fenomenului jitter.

Furnizarea semnalului de ceas

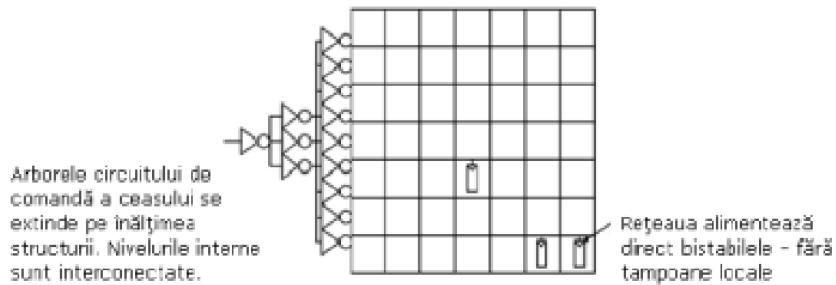
Semnalul de ceas nu poate fi distribuit în același moment la toate bistabilele din circuit



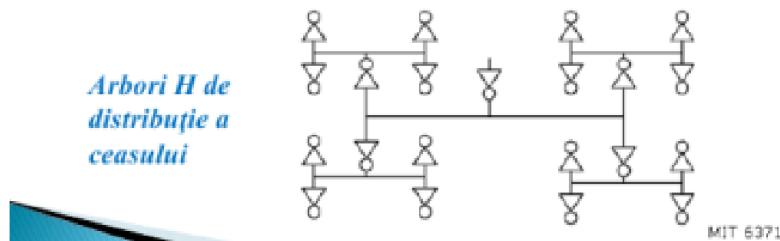
MIT 6371

29.Exemple de retele de distribuție a ceasului.

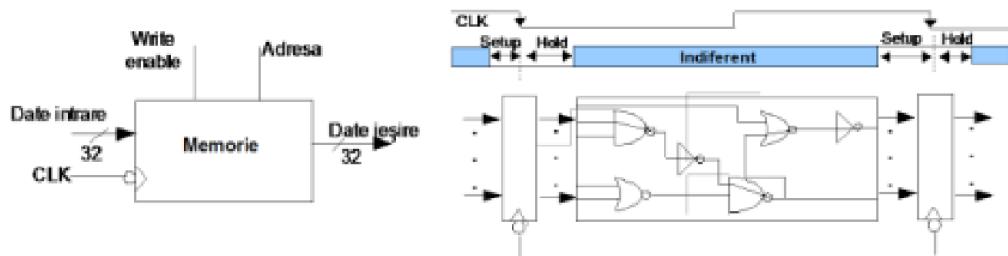
Rețea de distribuție a ceasului



MIT 6371



Memoria ideală



Metodologia de sincronizare :

- toate elementele de memorare sunt controlate pe același front al ceasului.
- durata ciclului este: $= CLK \rightarrow Q + \text{Întârzierea pe calea cea mai lungă} + \text{Timp de stabilire (Setup Time)} + \text{Alunecarea ceasului (Clock Skew)}$
- $(CLK \rightarrow Q + \text{Întârzierea pe calea cea mai scurtă} - \text{Alunecarea ceasului}) > \text{Timpul de menținere (Hold Time)}$

30.FPGA. Definitie. Avantaje. Structura.

Ariile de Porți Programabile (FPGA) reprezintă structuri bidimensionale formate din blocuri logice (CL) și de bistabile (FF), prevăzute cu facilitățile necesare configurației de către utilizator, atât a interconexiunilor între blocurile logice, cât și a funcției fiecărui bloc.

Comparație între soluțiile bazate pe ASIC, FPGA și MICRO pentru implementarea unui sistem numeric

performanță	NRE	Cost/ unitate	TTM
ASIC	ASIC	FPGA	ASIC
FPGA	FPGA	MICRO	FPGA
MICRO	MICRO	ASIC	MICRO

FPGArile au avantajul performanței, în raport cu microprocesoarele, deoarece circuitele pot fi adaptate ușor la aplicație. Microprocesoarele realizează funcțiile speciale în software, în condițiile operării în mai multe cicluri.

STRUCTURA ARIILOR DE PORȚI PROGRAMABILE

Componentele structurale de bază ale ariilor de porți, de tip FPGA, sunt:

- blocurile logice programabile,
- comutatoarele programabile
- traseele de interconectare (routing).

Blocurile Logice se pot realiza sub formă de:

- rețele de perechi de tranzistoare NMOS și PMOS, comutatoare de tip T-gates;
- rețele de porți logice combinaționale (NAND, XOR etc);
- multiplexoare;
- tabele asociative (lookup tables) cu n intrări;
- structuri SI-SAU cu multe intrări.

31. Prin ce difera familiile de FPGA-uri ?

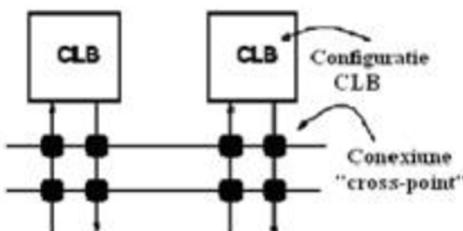
Familiile de FPGA-uri
diferă prin:

- mijloacele fizice de realizare a programării de către utilizator;
- organizarea traseelor de interconectare;
- funcțiile de bază ale blocurilor logice combinaționale (CLB).

Cele mai mari diferențe se regăsesc la tehniciile folosite pentru realizarea unor interconexiuni cât mai flexibile în cadrul blocurilor și între blocuri.

32. FPGA – tipuri de conexiuni, temporare și permanente.

Conexiuni de tip "puncte de intersecție" în cadrul unor trasee de tip "magistrale intersectate" sau "cross-bar"

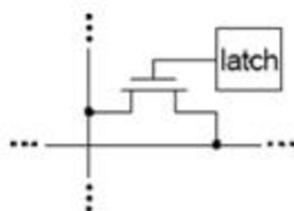


La intersecțiile între barele verticale și cele orizontale se pot stabili conexiuni permanente sau temporare, în funcție de tehnologia utilizată.

se utilizează elemente de tip "anti-fuse" cu contact permanent stabilit ca urmare a aplicării temporare a unei tensiuni ridicate



se utilizează în calitate de comutatoare tranzistoare NMOS, cu canal induș, sau tranzistoare cu poartă flotantă



• Avantaje: caracterul nevolatil, dimensiunile relativ mici, rezistența și capacitatea reduse.

• Dezavantaje: conținutul fix, imposibilitatea reprogramării.

• Avantaj principal: posibilitatea reconfigurării.

• Dezavantaje: caracterul volatil, dimensiunile relativ mari ale comutatoarelor.

33.FPGA – Caracteristicile arhitecturilor de interconectare (Xilinx, Actel, Altera).

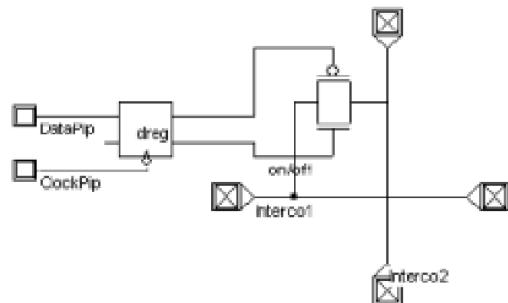
Structura de interconectare între BLC -uri , pe de o parte, și BLC-uril – BIE, pe de altă parte, constă în comutatoare programabile și fire/trasee de legătură.

- O conexiune directă, un *segment*, între două puncte, fără comutatoare între ele, va purta numele de *fir*.
- O înlățuire de segmente, care conțin și comutatoare prin care se transmite un semnal dat se numește *traseu*.

Tehnicile de interconectare sunt acelea care stabilesc raportul între ariile ocupate de către partea logică și partea de interconectare, din cadrul unui FPGA.

Comutatoarele sunt realizate cu ajutorul tranzistoarelor NMOS (tranzistoare de trecere/pass transistors), care asigură o bună transmisie a semnalului logic “0”, sau cu ajutorul cuplurilor de tranzistoare NMOS/PMOS (porți T/T-gates), care au avantajul unei bune transmisii a ambelor valori (“0”/“1”) ale semnalelor logice.

Structura unui Punct de Interconectare Programabil (PIP), care este format dintr-un bistabil de tip D și o poartă de tip T/Tgate.



Xilinx (fig. a) utilizează trasee de interconectare, care înconjoară fiecare bloc logic combinațional (CLB), la intersecțiile traseelor fiind plasate blocuri de comutatoare(BC), organizate sub formă de matrice, formate din tranzistoare de trecere, controlate pe grile cu ajutorul unor celule de memorie statică SRAM (fig. b).

ACTEL propune mai multe segmente de fire având orientare orizontală, față de cele care cu orientare verticală.

Altera propune o structură de interconectare bazată pe două niveluri ierarhice: local (a) și global (b).

40.Efectuati o scadere folosind complementul lui 2 (exemplu $13-2=11$).

$$13 - 2 = 11$$

$$13 = 01101$$

$$2 = 00010$$

$$C2(2) = 11101 + 00001 = 11110$$

$$13 + C2(2) = 01101 + 11110 = 101011 (= 11)$$

41.Efectuati o inmultire folosind algoritmul lui Booth.

$$m = 3 \quad ; \quad r = 4 \quad ; \quad -m = -3 \quad ; \quad p = 12$$

$$m = 0011 \quad -m = 1101 \quad ; \quad p = 1100 \\ r = 0100$$

$$A = 0011 \quad 0000 \quad 0$$

$$S = 1101 \quad 0000 \quad 0$$

$$P = 0000 \quad 0100 \quad 0$$

$$A \equiv m \quad 0000 \quad 0$$

$$S \equiv -m \quad 0000 \quad 0$$

$$P \equiv 0000 \quad r \quad 0$$

1. $P = 0000 \quad 0100 \quad 0$ $\rightarrow 00 \rightarrow P \text{ unchanged}$
 arithmetic shift right

2. $P = 0000 \quad 0010 \quad 0$ $\rightarrow 00 \rightarrow P \text{ unchanged}$
 a. s. r.

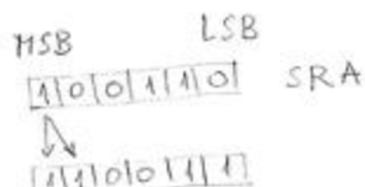
3. $P = 0000 \quad 0001 \quad 0$ $\rightarrow 10 \rightarrow P = P + S$
 $P = 1101 \quad 0001 \quad 0$

0000	0001	0
1101	0000	0
1101	0001	0

4. $P = 1110 \quad 1000 \quad 1$ $\rightarrow 01 \rightarrow P = P + A$
 $P = 0001 \quad 1000 \quad 1$

1110	1000	1
0011	0000	0
0001	1000	1

a. s. r.
 $\rightarrow P = \underbrace{0000 \quad 1100}_{} \quad 0$
 $= 12$



$$m = 14 \quad ; \quad r = -5 \quad ; \quad p = m \times r = -70$$

$$m = 01110 \quad ; \quad -m = 10010 \quad ; \quad s = 00101 \\ r = -5 = 11011$$

$$A = m \ 00000 \ 0 = 01110 \ 00000 \ 0$$

$$S = -m \ 00000 \ 0 = 10010 \ 00000 \ 0$$

$$P = 00000 \ r \ 0 = 00000 \ 11011 \ 0$$

1. $P = 00000 \ 11011 \ 0 \rightarrow 10 \rightarrow P = P + S$

$$\begin{array}{r} 10010 \ 00000 \ 0 \\ 10010 \ 11011 \ 0 \\ \hline 10010 \ 11011 \ 0 \end{array}$$

SRA
2. $P = 11001 \ 01101 \ 1 \rightarrow 11 \rightarrow P = \bar{P}$

SRA
3. $P = 11100 \ 10110 \ 1 \rightarrow 01 \rightarrow P = P + A$

$$\begin{array}{r} 01110 \ 00000 \ 0 \\ 01010 \ 10110 \ 1 \\ \hline 01010 \ 10110 \ 1 \end{array}$$

SRA
4. $P = 00101 \ 01011 \ 0 \rightarrow 10 \rightarrow P = P + S$

$$\begin{array}{r} 10010 \ 00000 \ 0 \\ 10111 \ 01011 \ 0 \\ \hline 10111 \ 01011 \ 0 \end{array}$$

SRA
5. $P = 11011 \ 10101 \ 1 \rightarrow 11 \rightarrow P = \bar{P}$

SRA
 $\rightarrow \underbrace{11101 \ 11010 \ 1}_{= -70}$

$$\begin{aligned} 70 &= 64 + 4 + 2 \\ 70 &= 0001000110 \\ -70 &= 1110111001 + 1 \\ &= 1110111010 \end{aligned}$$

42.Limbaje de asamblare: MIPS

- structura unei instructiuni;
- structura unui cod MIPS;
- ce reprezinta instructiunile: li, la, l.s, lw, sw etc;
- instructiuni de salt conditionat: ble, bge etc;

Limbajul MIPS este un limbaj de tip RISC (Reduced Instruction Set Computer), un concept care include mai multe concepte ce simplifică arhitectura internă și sporesc viteza de calcul. Fiind un limbaj de tip RISC, fiecare instructiune din MIPS are exact 32 de biți. O instructiune MIPS se translateaza la compilare intr-o singura instructiune masina si ei ii va corespunde o linie in fereastra "Text segment", care in (c) va contine un text MIPS identic sau echivalent cu linia din fisier.

Instructiunile sunt împărțite în trei tipuri: R, I și J. Fiecare instructiune începe cu un opcode pe 6 biți. În plus față de opcode, instructiunile de tip R specifică trei registri, un câmp de cantitate de schimbare și un câmp de funcții. Instructiunile de tip I specifică două registre și o valoare imediată de 16 biți; Instructiunile de tip J urmăresc opcodul cu o întârziere de salt de 26 de biți.

Structura unei instructiuni MIPS: add des src1, src2, unde:

add = functia MIPS pentru adunare (pot aparea si alte functii, precum sub, mul, div etc.)

des = registrul in care va fi stocata adunarea

src1 = primul regisztr care face parte din adunare

src2 = al doilea regisztr care face parte din adunare

Exemplu de instructiune MIPS: add \$t2, \$t0, \$t1 ($t2 = t0 + t1$)

Datele se declara astfel: nume: tip valori unde

- nume = este o eticheta, careia i se asociaza la compilare adresa pana la care s-a ajuns cu umplerea zonei de date a memoriei in acel moment
- tip = este un tip de date si poate fi ".byte", ".word", ".space", etc.

- valori = este o lista de valori compatibile cu "tip" care vor fi stocate in zona de date incepand de la adresa "nume", fiecare data ocupand o locatie corespunzatoare lui "tip"

ex: var: .word 3

Structura unui cod MIPS:

- ```
.data
declaratii date
.text
cod
main: # eticheta marcand punctul de start
cod
li $v0,10
syscall
o linie goala
```
- segmentul de date (.data) care afisaza zona de memorie ce contine datele statice ale programului, stiva si datele sistemului
  - segmentul de text (.text)
  - utilizarea unor etichete pentru variabilele din segmentul de date si pentru punctele de start
  - utilizare registrelor
  - apelul de system (syscall)

Exemplu de cod MIPS:

```
.data
val: .word 29
Adunare: .asciiz "Adunare = "
newLine: .asciiz "\n"
.text
.globl main
main:
 li $v0, 4
 la $a0, Adunare
 syscall

 li $t0, 25
 lw $t1, val
 add $t2, $t0, $t1
 li $v0, 1
 move $a0, $t2
 syscall

 li $v0, 4
 la $a0, newLine
 syscall

 li $v0, 10
 syscall
```

## Scheletul unui program

Un program conține 2 secțiuni:

### 1. .data - Secțiunea datelor

- Declararea datelor folosite în program:
  - Rezervarea unui spațiu în memorie;
  - Scrierea valorii în memorie.

### 2. .text - Secțiunea instrucțiunilor

- Scrierea instrucțiunilor;
- Construirea programului principal;
- O instrucțiune: 32 biți (4 bytes, 1 word).

li reg, val (load immediate) = incarca in registrul "reg" valoarea imediata "val"

la reg, var (load address) = incarca in registrul "reg" adresa variabilei cu numele "var"

lw reg, adr (load word) = incarca in registrul "reg" word-ul aflat la adresa "adr"

ls reg, adr = incarca in registrul "reg" variabila de tip float aflat la adresa "adr"

sw rt, adr = in memorie se scrie incepand de la adresa "adr" word-ul din registrul "rt"

sb rt, adr = in memorie se scrie incepand de la adresa "adr" byte-ul low din registrul "rt"

move rdest, rsrc = copiaza valoarea din registrul "rsrc" in registrul "rdest"

mov.s rdest, rsrc = copiaza valoarea de tip float din registrul "rsrc" in registrul "rdest"

Instructiuni de salt conditionat : beq/blt/ble/bgt/bge/bne rs, rt, eticheta

- testeaza continutul registrilor rs si rt
- verificand daca avem respectiv rs=rt(beq), rs<rt(blt), rs<=rt(ble), rs>rt(bgt), rs>=rt(bge), rs!=rt(bne)
- verificarea se face considerand continuturile ca intregi cu semn
- daca conditia este verificata, atunci executia ramifica la eticheta respectiva
- daca conditia nu este verificata, atunci executia trece la instructiunea urmatoare