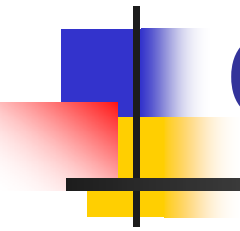
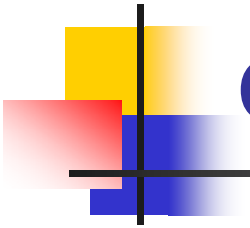


Circuite integrate digitale

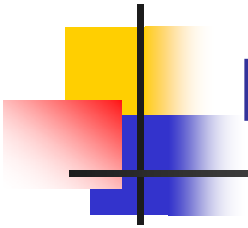
Curs 11



Cuprins

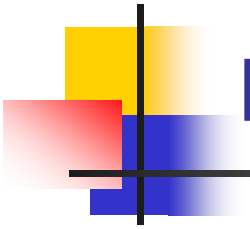
Exemple de automate simple

FPGA



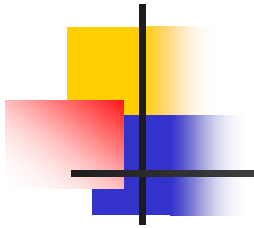
Exemplul 1 – circuit de arbitrare

- Să se descrie în Verilog un automat care realizează funcţia de arbitrare a două solicitări req_0 şi req_1, dintre care req_0 este prioritară. Implementaţi automatul în două variante:
  - folosind două procese always pentru calculul stării următoare, respectiv ieşirii
  - folosind un singur proces always.
- Temă: Scrieţi un modul de test şi testaţi funcţionarea automatului.

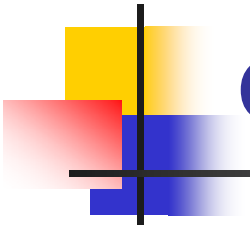


Rezolvare

- Etapele iniţiale ale rezolvării (înainte de a începe să scriem codul) presupun:
  - stabilirea intrărilor şi ieşirilor
  - desenarea grafului sau organigramei
  - codificarea stărilor.

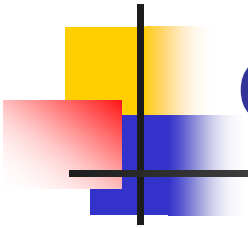


- intrările circuitului: clock, reset, req_0, req_1
- ieşirile: gnt_0, gnt_1
- funcţia realizată:
  - dacă req_0 este 1, se generează gnt_0 (este 1)
  - dacă req_1 este 1, se generează gnt_1 (este 1)
  - dacă sunt active simultan req_0 şi req_1, se generează gnt_0, adică req_0 este prioritar



Graful

- 3 stări şi tranziţiile dintre ele...
  - IDLE, starea implicită, în care ieşirile sunt inactive (0) şi se aşteaptă o solicitare – se rămâne aici câtă vreme intrările sunt inactive
  - GNT0: starea în care req_0 este activ şi ieşirea gnt_0 este activă (se ajunge aici din IDLE, dacă req_0 este activ şi se rămâne până când se dezactivează după care se revine în IDLE)
  - GNT1: starea în care req_1 este activ şi ieşirea gnt_1 este activă (se ajunge aici din IDLE, dacă req_0 este inactiv şi req_1 este activ şi se rămâne până când se dezactivează req_1 după care se revine în IDLE)



Codificarea stărilor

- one hot:
- IDLE = 3'b001,
- GNT0 = 3'b010,
- GNT1 = 3'b100

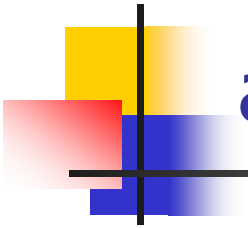
Varianta 1 - declaraţii

```
module fsm_arbiter (
clock       , // clock
reset       , // Active high, syn reset
req_0       , // Request 0
req_1       , // Request 1
gnt_0       , // Grant 0
gnt_1        // Grant 1
);
input   clock,reset,req_0,req_1;
output  gnt_0,gnt_1;
reg     gnt_0,gnt_1;
parameter IDLE  = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100 ;
reg   [2:0]          state        ;// starea curenta
reg   [2:0]          next_state   ;// starea urmatoare
```

calculul stării următoare

```
always @ (state or req_0 or req_1)
begin
 next_state = 3'b000;
 case(state)
   IDLE : if (req_0 == 1) begin
              next_state = GNT0;
            end else if (req_1 == 1)
   begin
              next_state= GNT1;
            end else begin
              next_state = IDLE;
            end
```

```
   GNT0 : if (req_0 == 1) begin
                next_state = GNT0;
              end else begin
                next_state = IDLE;
              end
   GNT1 : if (req_1 == 1) begin
                next_state = GNT1;
              end else begin
                next_state = IDLE;
              end
   default : next_state = IDLE;
  endcase
end
```



actualizarea stării

```
always @ (posedge clock)
begin
  if (reset == 1) begin
    state <= #1 IDLE;
  end else begin
    state <= #1 next_state;
  end
end
```

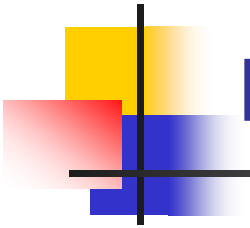
calculul ieşirii

```
always @ (posedge clock)
begin
if (reset == 1) begin
  gnt_0 <= #1 1'b0;
  gnt_1 <= #1 1'b0;
end
else begin
  case(state)
    IDLE : begin
                  gnt_0 <= #1 1'b0;
                  gnt_1 <= #1 1'b0;
                end
   GNT0 : begin
                   gnt_0 <= #1 1'b1;
                   gnt_1 <= #1 1'b0;
                 end
GNT1 : begin
                   gnt_0 <= #1 1'b0;
                   gnt_1 <= #1 1'b1;
                 end
   default : begin
                    gnt_0 <= #1 1'b0;
                    gnt_1 <= #1 1'b0;
                  end
  endcase
end
end
endmodule
```

Varianta 2 – cu aceleaşi declaraţii calculul stării următoare şi al ieşirii

```
always @ (posedge clock)
begin
if (reset == 1) begin
  state <= #1 IDLE;
  gnt_0 <= 0;
  gnt_1 <= 0;
end else
 case(state)
   IDLE : if (req_0 == 1) begin
              state <= #1 GNT0;
              gnt_0 <= 1;
            end else if (req_1 == 1) begin
              gnt_1 <= 1;
              state <= #1 GNT1;
            end else begin
              state <= #1 IDLE;
            end
   GNT0 : if (req_0 == 1) begin
              state <= #1 GNT0;
            end else begin
              gnt_0 <= 0;
              state <= #1 IDLE;
            end
   GNT1 : if (req_1 == 1) begin
              state <= #1 GNT1;
            end else begin
              gnt_1 <= 0;
              state <= #1 IDLE;
            end
   default : state <= #1 IDLE;
endcase
end
```

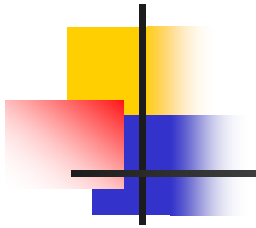
- Observaţie. Nu mai este necesar nextstate



Exemplul 2

- Descrieţi în Verilog un automat care recunoaşte o secvenţă numerică de patru cifre zecimale. Acest automat poate fi folosit pentru a comanda deschiderea încuietorii unui seif (varianta simplă cu care sunt dotate camerele de hotel). Automatul va avea două ieşiri, una pentru semnalarea erorii, iar alta pentru a comanda deschiderea seifului, numai atunci când secvenţa introdusă corespunde cifrului. Verificaţi funcţionarea corectă pentru diferite secvenţe numerice.

- *Indicaţie.* Pentru a nu oferi niciun indiciu, în cazul introducerii unui cod greşit se va semnala eroare numai după ce au fost introduse patru cifre.



- În rezolvarea propusă, automatul recunoaşte numărul 3729 (cifrul este definit ca parametru).

```
module automatrecunoastere(
    input [3:0] nr,
    input ck,
   input reset,
    output err,
    output open
    );
reg [3:0] state;
parameter N1 = 3;
parameter N2 = 7;
parameter N3 = 2;
parameter N4 = 9;
```

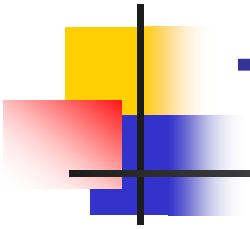
Stările... desenaţi graful!

```
localparam S0 = 4'd0  ;
localparam S1 = 4'd1  ;
localparam S2 = 4'd2  ;
localparam S3 = 4'd3  ;
localparam S4 = 4'd4  ;
localparam E1 = 4'd5  ;
localparam E2 = 4'd6  ;
localparam E3 = 4'd7  ;
localparam E4 = 4'd8  ;
```

```
always @ (posedge ck)
   if (reset)
        state <= S0;
   else
    case (state)
        S0: if (nr == N1) state <= S1;
                          else state <= E1; //ca sa nu ne prindem ce cifra
   gresim
        S1: if (nr == N2) state <= S2;
                          else state <= E2;
        S2: if (nr == N3) state <= S3;
                          else state <= E3;
        S3: if (nr == N4) state <= S4;
                          else state <= E4;
        S4: state <= S4;
        E1: state <= E2;
        E2: state <= E3;
        E3: state <= E4;
        E4: //if (nr == N1) state <= S1; else
                           state <= E4;
        default: state <= S0;
   endcase

assign open = (state      == S4)? 1:0;
assign err = (state       == E4)? 1:0;
endmodule
```

```
module testautomat;
	// Inputs
	reg [3:0] nr;
	reg ck;
	reg reset;
	// Outputs
	wire err;
	wire open;
	// Instantiate the Unit Under Test (UUT)
	automatrecunoastere uut (
		.nr(nr),
		.ck(ck),
		.reset(reset),
		.err(err),
		.open(open)
	);
```

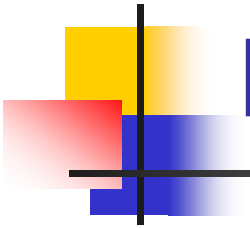
```
initial begin
        ck =0;
        forever #10 ck = ~ ck;
        end
        initial begin
                // Initialize Inputs
                nr = 0;
                reset = 0;
                #5 reset = 1;
                #20 reset = 0; nr = 3;
                #20 nr = 7;
                #20 nr = 2;
                #20 nr = 9;
                #100 reset = 1;
                #20 reset = 0;
                #100 reset = 1;
                #20 reset = 0;
                nr = 3;
                #20 nr = 7;
                #20 nr = 2;
                #20 nr = 3;
                $stop;
        end

endmodule
```



Tema 13

- scrieţi un modul de test pentru automatul arbitru din exemplul 1

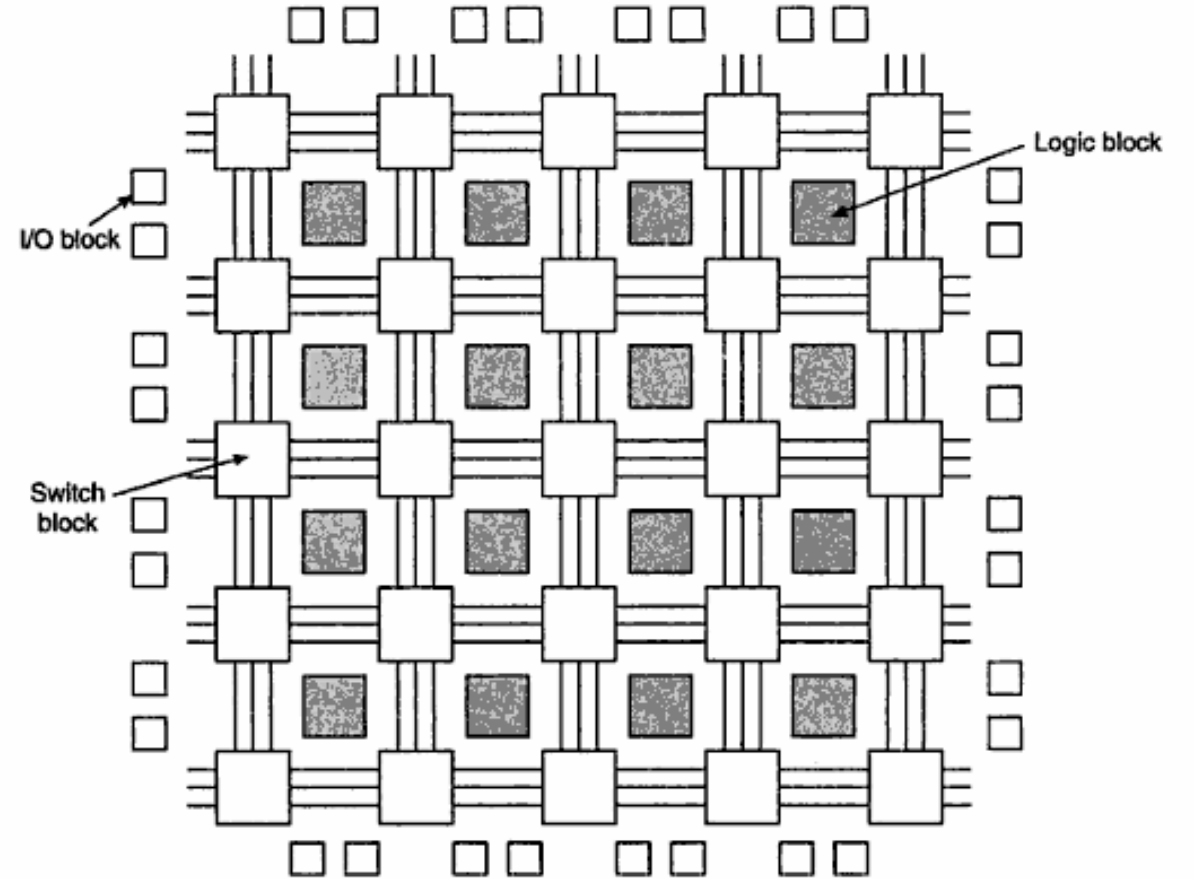

FPGA



Definiţii

- Un FPGA (Field Programmable Gate Array) este un circuit digital integrat care conţine
  - **blocuri de circuite logice** programabile (configurabile) şi
  - **interconexiuni** configurabile între aceste blocuri.
- Prin programarea (configurarea) acestora, FPGA-urile pot realiza o gamă foarte mare de funţii
- În funcţie de implementare (tehnologie), unele FPGA pot fi programate o singură dată, iar altele pot fi reprogramate din nou şi din nou.
  - OTP- *one time programmable*

Arhitectura FPGA



Definiţii (continuare)

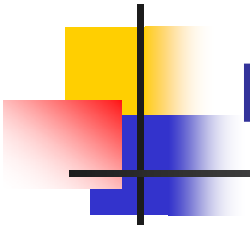
- Field programmable – se referă la faptul că pot fi programate "pe teren", adică funcţionalitatea lor nu este prestabilită prin hardware-ul realizat de fabricant. De exemplu, pot fi programate in laborator.
- Unele FPGA-uri pot fi reprogramate şi după ce au fost incluse deja într-un sistem digital (embedded system): ISP – *in-system programmable*

- Alte definiţii pentru acronimele utilizate:
  - PLD – *programmable logic devices* SPLD, CPLD
  - ASIC – *application-specific integrated circuits*
  - ASSP – *application-specific standard parts*

PLD – FPGA – ASIC

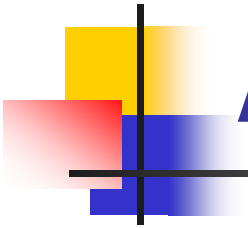
- **PLD**
  - arhitectura este predeterminată de fabricant
  - pot fi configurate ulterior de utilizator pentru a realiza diferite funcții
  - conțin un număr redus de porți logice în comparație cu FPGA şi pot realiza funcții mai simple şi limitate ca număr

- **ASIC şi ASSP**
  - conțin sute de milioane de porți logice
  - pot realiza funcții foarte complexe (sunt proiectate pentru o anumită aplicație)
  - sunt bazate pe acelaşi flux de proiectare şi proces tehnologic
  - ASSP – sunt utilizate de diferite companii
  - cele mai eficiente din pdv al numărului de tranzistoare, complexitate şi performanțe
  - costisitoare
  - nu pot fi modificate!



FPGA

- ocupă domeniul intermediar între PLD şi ASIC
  - funcţionalitatea poate fi modificată
  - pot conţine milioane de porţi logice
  - pot implementa sisteme de dimensiune şi complexitate mare, care anterior nu puteau fi realizate decât cu ASIC
- costuri şi proiectare:
  - costurile proiectelor sunt mult reduse în comparaţie cu ASIC
  - componentele finale în producţia pe scară largă sunt mult mai ieftine cu ASIC
  - implementarea modificărilor – avantajos cu FPGA
  - timpul de lansare pe piaţă – mult mai redus cu FPGA



Aplicaţii

- anii 80: *glue logic*, automate finite de complexitate medie, procesare de date de dimensiuni relativ limitate
- începutul anilor 90: telecomunicaţii şi reţele (procesare şi transfer de date)
- ulterior: auto, aplicaţii industriale, electronica de consum – creştere spectaculoasă a utilizării

- Iniţial FPGA-urile erau folosite şi pentru realizarea prototipurilor ASIC, pentru platforme experimentale de testare a noilor algoritmi de calcul – datorită costurilor reduse au fost ulterior regăsite tot mai mult în produsele finale
- FPGA-urile de înaltă performanţă conţin la ora actuală microprocesoare încorporate, dispozitive I/O de mare viteză etc. Pot fi folosite pentru a implemente aproape orice, inclusiv aplicaţii DSP şi SOC

Domenii de dezvoltare și aplicații

- **ASIC și implementare pe siliciu** – FPGA-urile se aplică în domenii exclusiv acoperite anterior de ASIC
- **DSP – *digital signal processing*** – FPGA-urile cele mai noi conțin multiplicatoare, circuite de rutare destinate aplicațiilor aritmetice, RAM. Paralelismul masiv al FPGA le face de 500 de ori mai rapide decât alte implementări DSP
- ***Embededed microcontrollers*** – înlocuiesc microcontrollerele simple, deoarece implementează în același timp microprocesorul și controllerele IO dorite

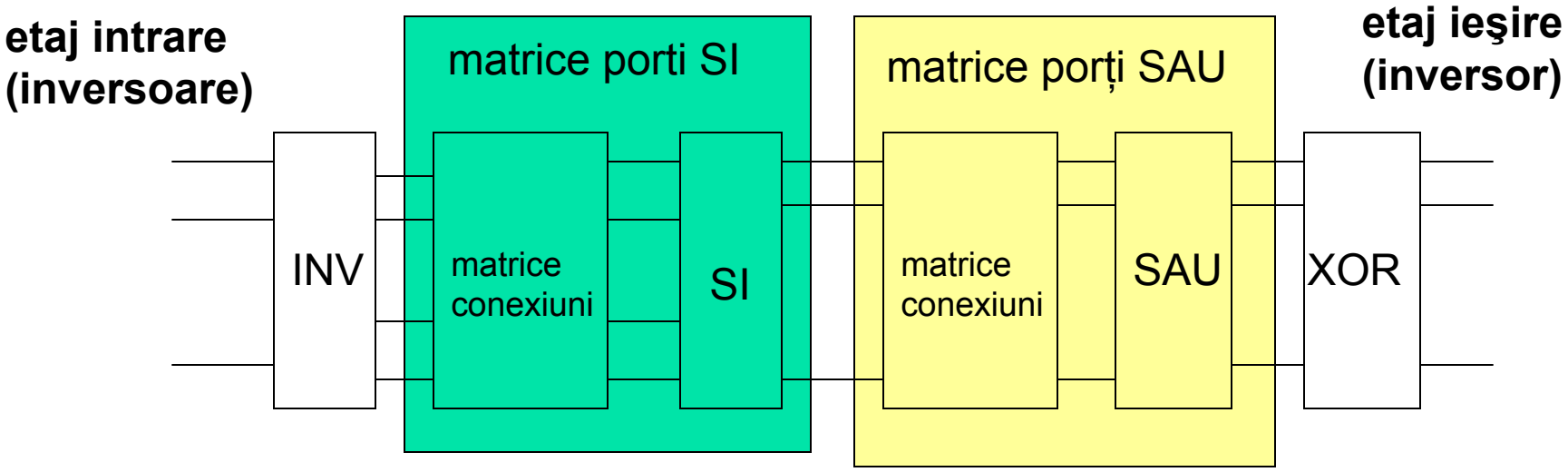
Domenii de dezvoltare și aplicații

- **comunicaţii (stratul fizic)** – inițial folosite pentru circuitelelogice care asigură interfața dintre „stratul fizic” al cipurilor de comunicații și protocoalele de interconectare de nivel înalt, acum FPGA încorporează transievere high speed – un singur chip pentru comunicații și interconectare

- ***reconfigurable computing*** – segment de piață creat de FPGA – folosirea paralelismului și a hardwareului configurabil pentru a accelera funcționarea algoritmilor software. Aplicații: simulare de hardware, analize criptografice

PLD

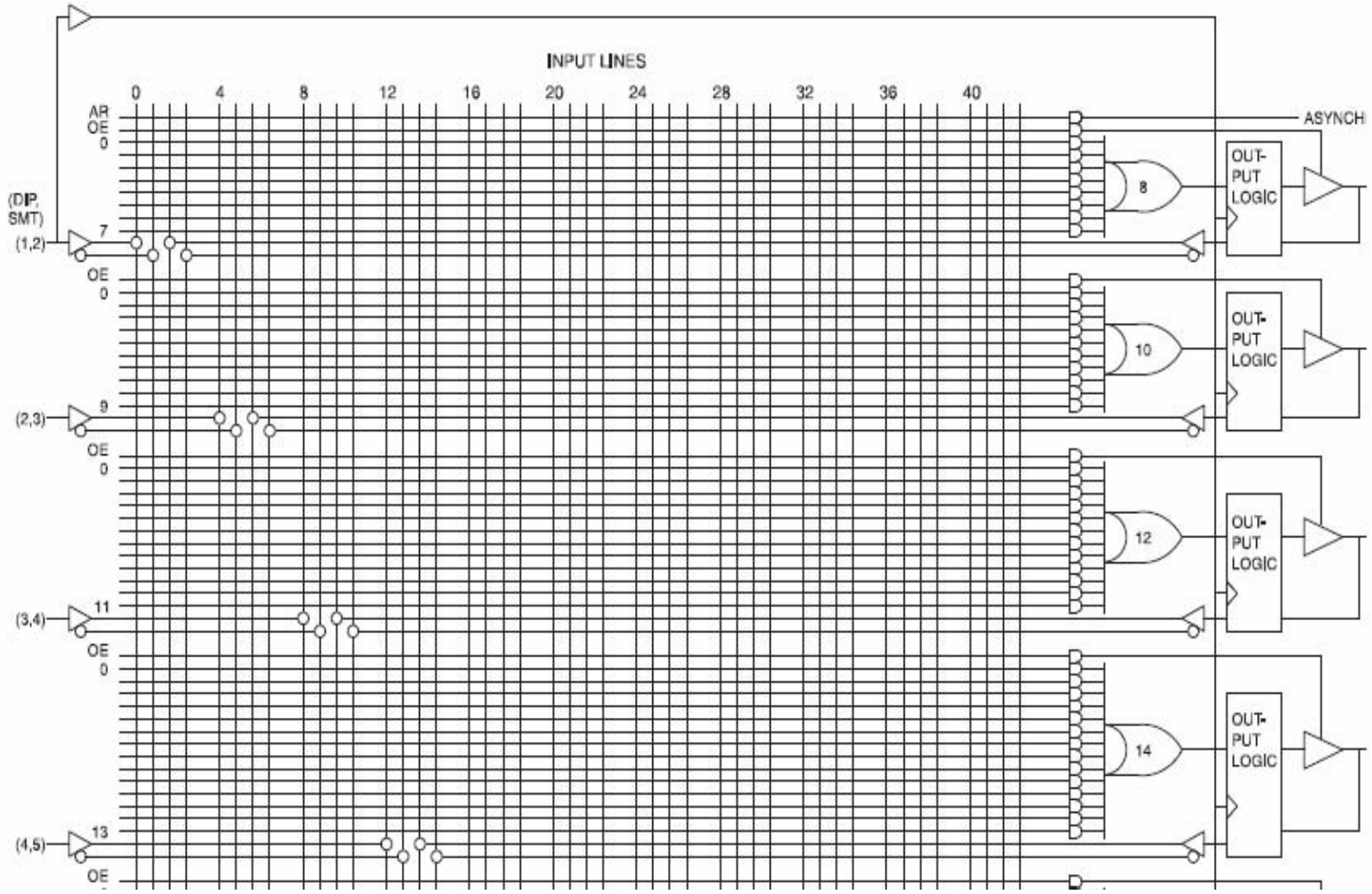
- dispozitive logice programabile (se "programează" conexiunile)

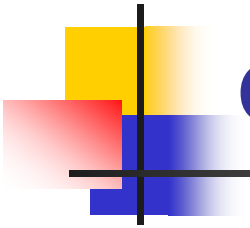


Tipuri de PLD-uri

	matrice SI	matrice SAU
ROM	fixă (DCD) maximă	programabilă
PLA	programabilă	programabilă
PAL	programabilă	fixă

11. Functional Logic Diagram

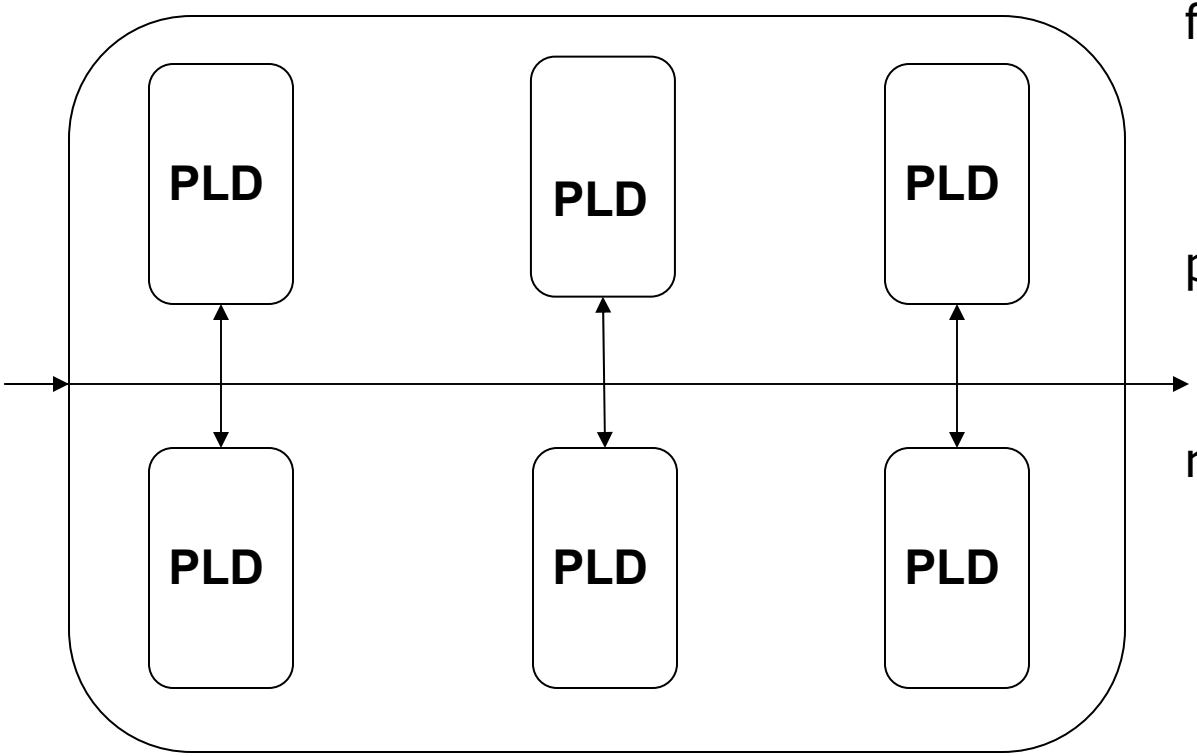




CPLD

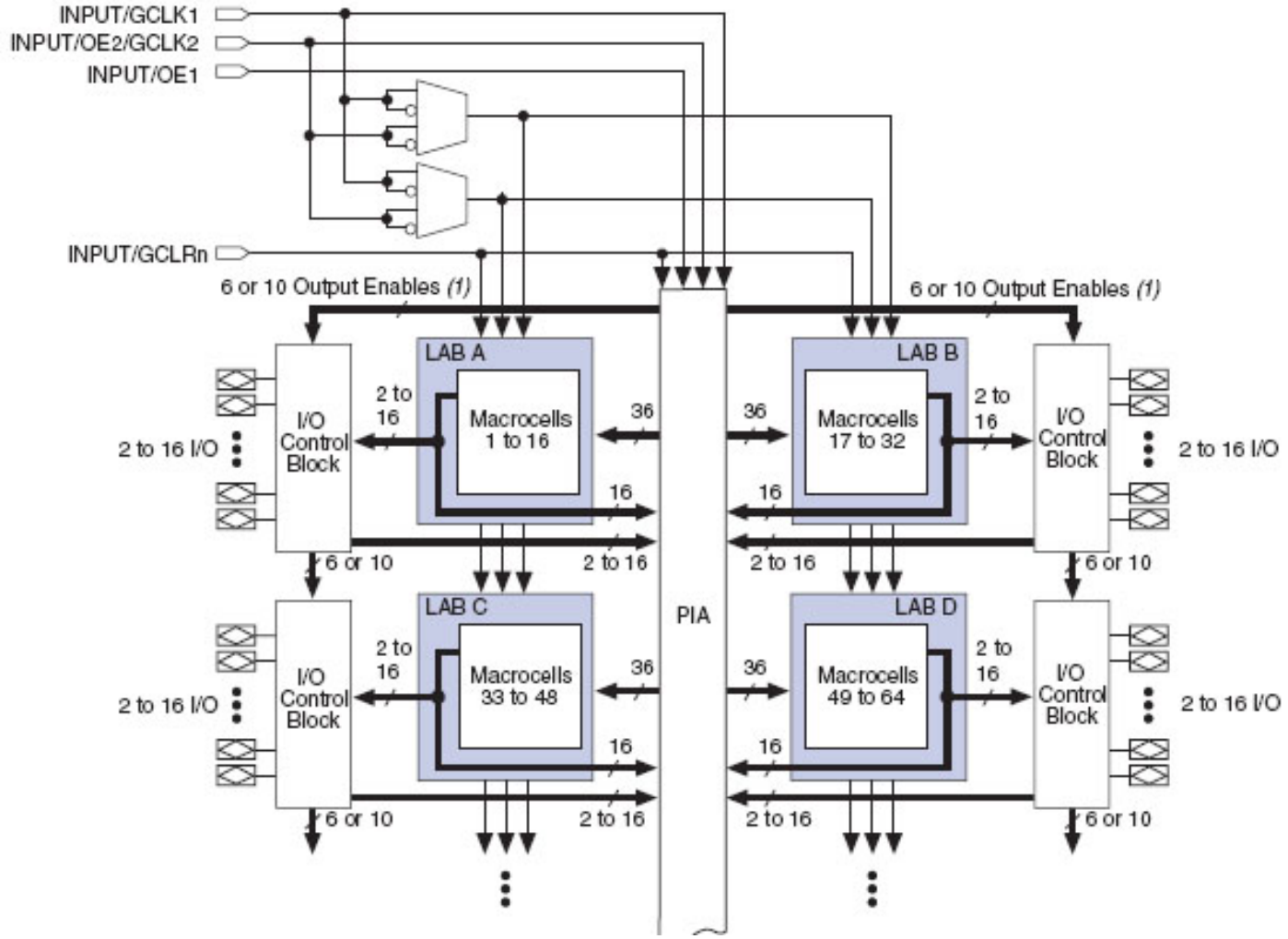
- Complex Programmable Logic Device
- dispozitiv logic programabil cu complexitate intermediară între PLD şi FPGA
  - PLD: programmable logic device
  - FPGA: field programmable gate array
- elemente de structură de la PLD şi FPGA

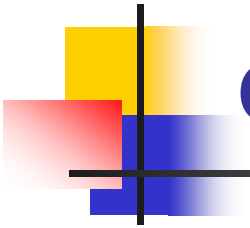
Arhitectura de principiu a unul CPLD



- fiecare PLD are o structura interna asemănătoare cu un PLA
- porţi AND cu fan-in mare
- reţea de interconectare programabilă

MAX 3000A Device Block Diagram

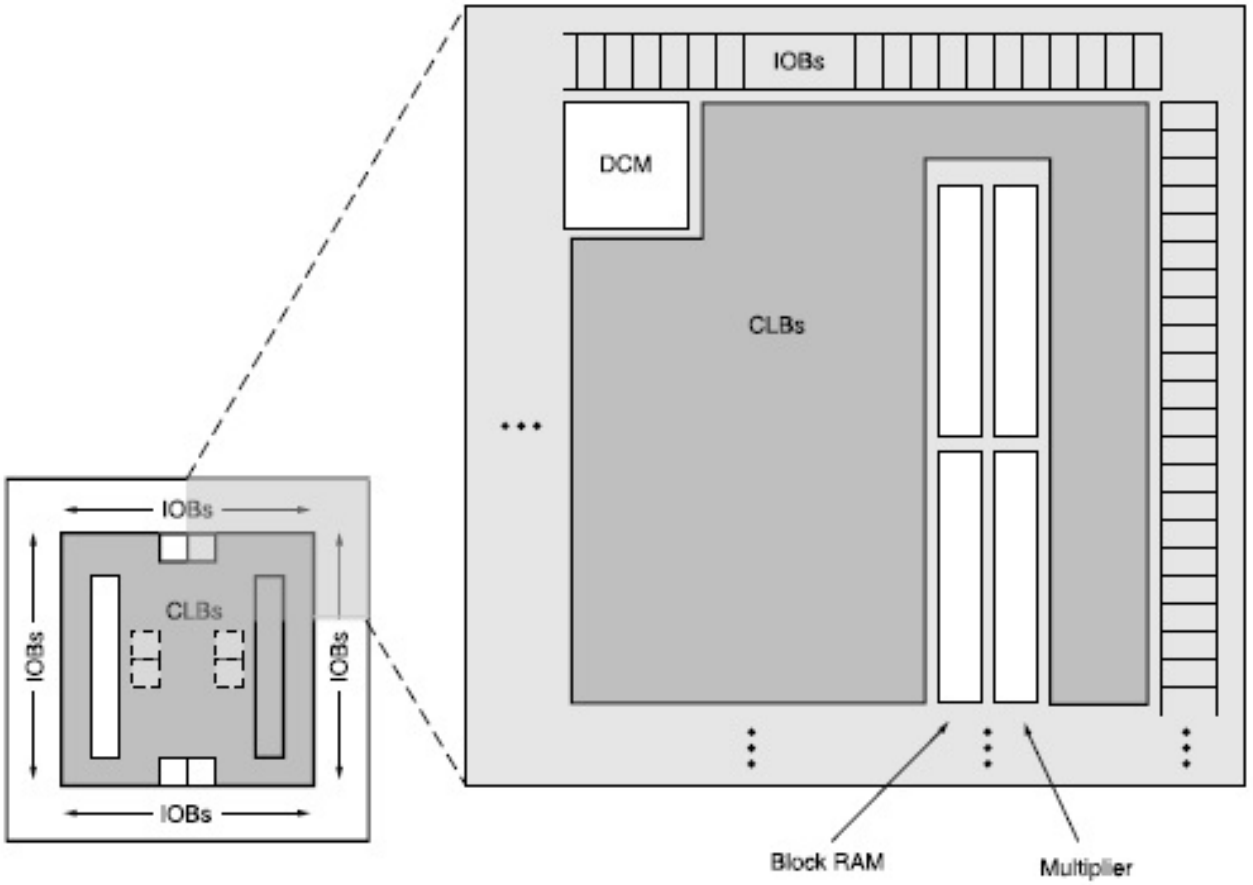


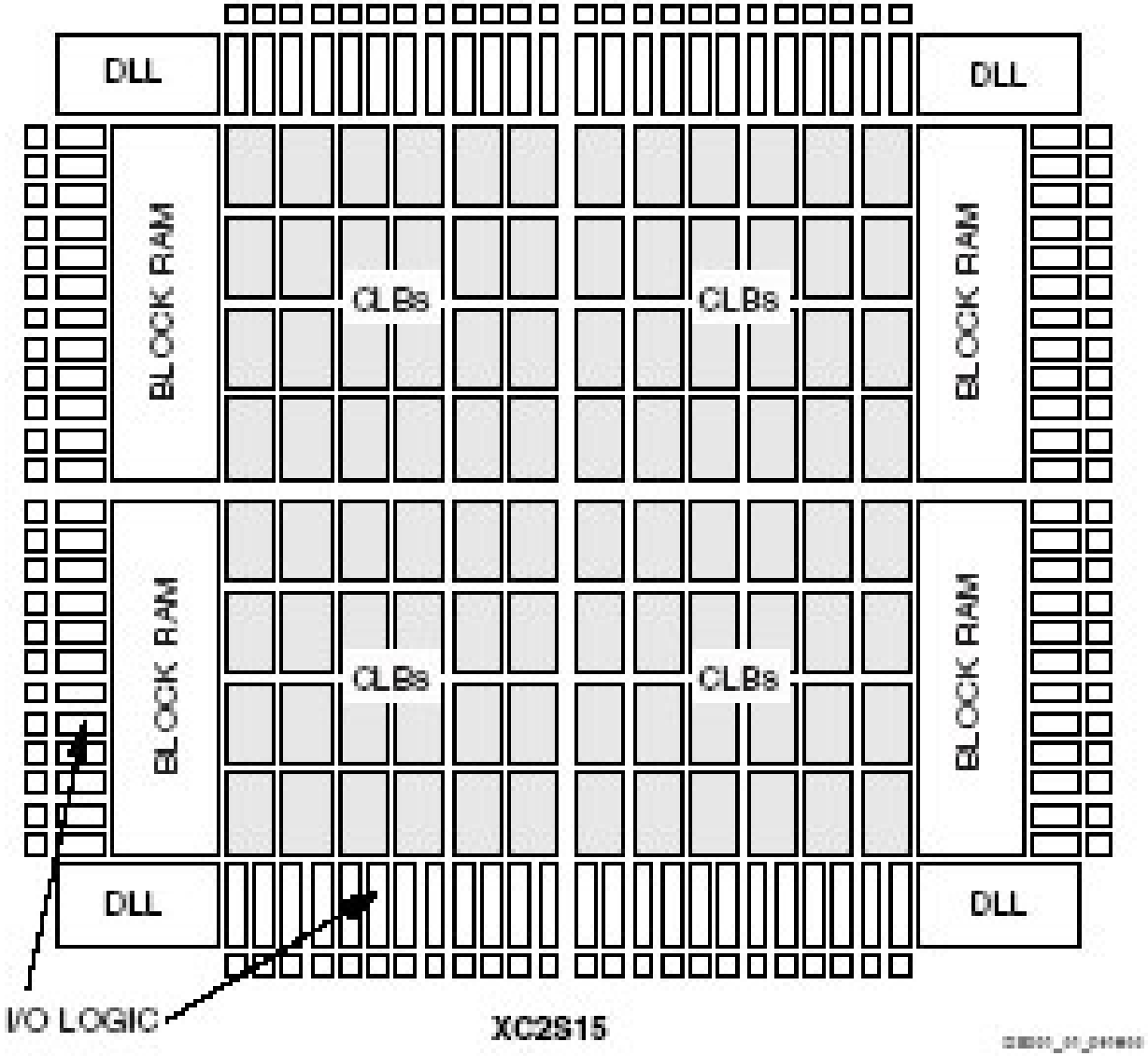


Componente principale CPLD

- Logic Array Blocks (LABs)
- Macrocells
- circuite pentru expandarea produselor
- Programmable Interconnect Array (PIA)
- blocuri de control I/O

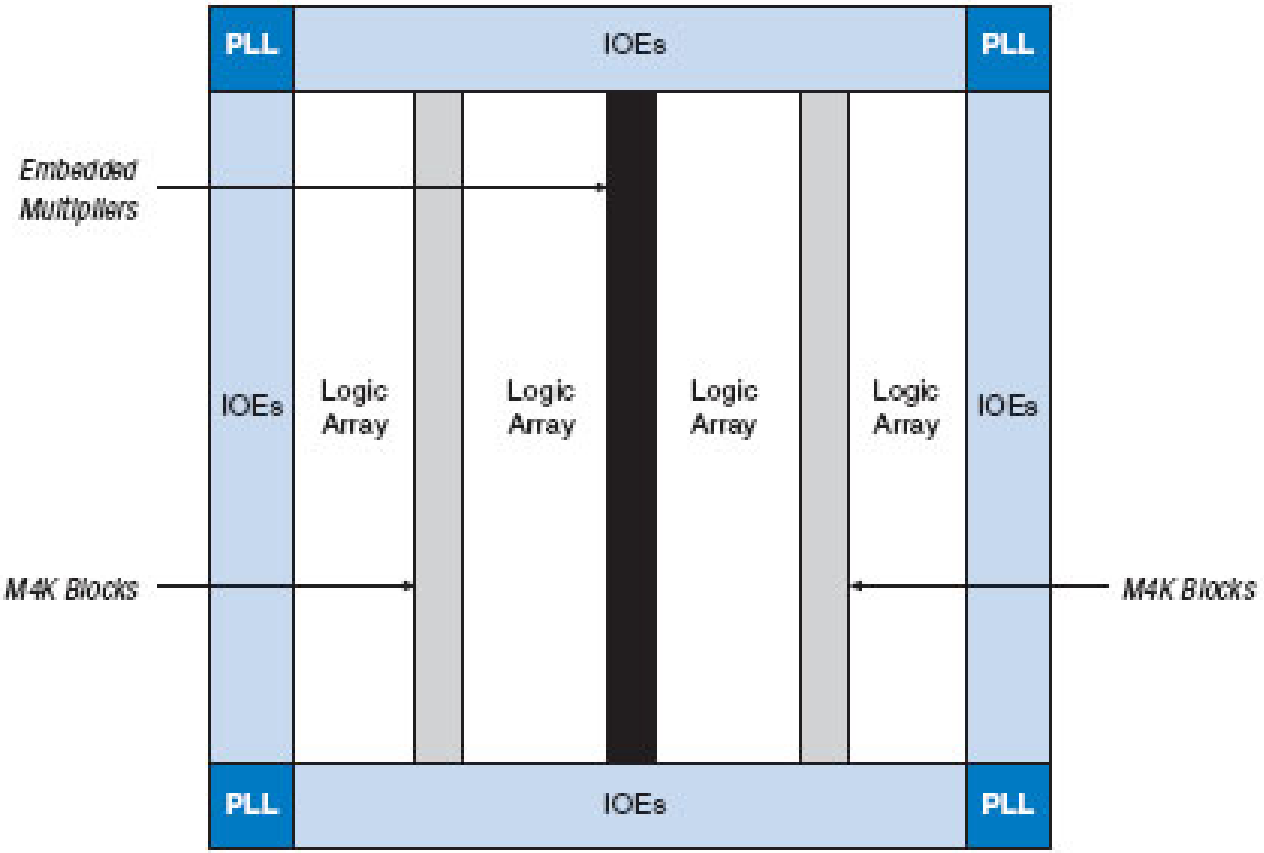
Arhitectura FPGA

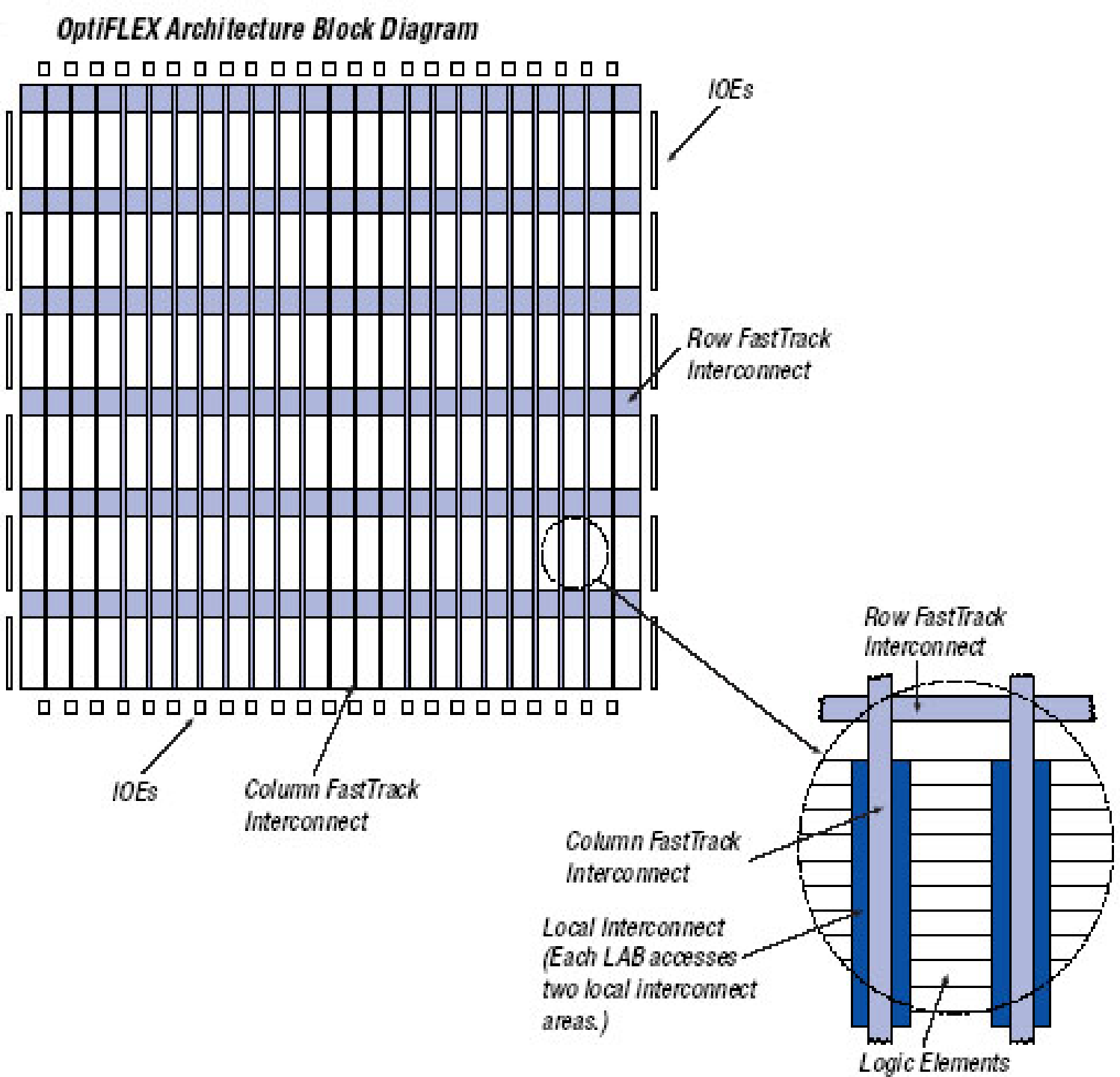




Basic Spartan-II Family FPGA Block Diagram

Cyclone II EP2C20 Device Block Diagram




OptiFLEX Architecture Block Diagram
IOEs
Row FastTrack Interconnect
IOEs
Column FastTrack Interconnect
Row FastTrack Interconnect
Column FastTrack Interconnect
Local Interconnect (Each LAB accesses two local interconnect areas.)
Logic Elements

Comparaţie CPLD-FPGA
1- logica

- CPLD-urile conţin o matrice de blocuri tip PAL care implementează funcţii de tip sumă de produse cu dimensiune mare a intrării
- porţi fan-in mare
- Implementarea sumelor de produse (SOP)

- La FPGA structura este mai complexă, în principal conţine mai multe registre, blocurile funcţionale sunt mai flexibile
- porţi fan-in redus
- implementarea tabelelor de adevăr (look-up table)

Comparaţie CPLD-FPGA
2- rutarea

- CPLD – model timing predictibil (nu depinde de rutarea semnalelor)
- reţea de tip crossbar
- performanţele nu depind consistent de dispunerea aplicaţiei pe matrice

- FPGA – model timing complicat (depinde de rutarea semnalelor)
- Reţea mai complexă şi flexibilă de interconectare
- performanţele depind semnificativ de rutare (poziţionarea circuitelor folosite)

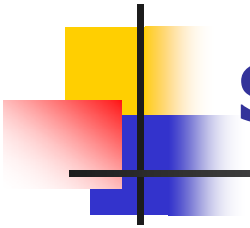
Comparaţie CPLD-FPGA
3 - Reconfigurarea

- CPLD – memorie EEPROM
- număr limitat reconfigurări

- FPGA – antifuse, SRAM, flash
- Antifuse: OTP, permanent
- SRAM: se şterge când nu mai este alimentat, număr nelimitat de reconfigurări
- Flash – similar EEPROM

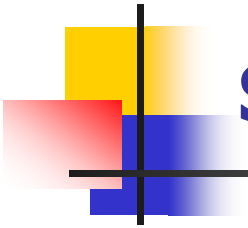
Comparaţie CPLD-FPGA
4 - aplicaţii

- aplicaţii de dimensiune redusă şi medie
- aplicaţii de dimensiune mare (nr de porţi, complexitate)



Specific FPGA

1. performanţa pentru orice aplicaţie depinde de rutare
2. funcţionalitatea este implementată prin multiplexare sau tabele de adevăr
3. în general sunt volatile



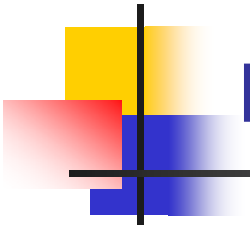
Structura unui FPGA

1. matrice de unităţi funcţionale programabile care implementează logica sevenţială şi combinaţională
2. structură de interconectare programabilă care stabileşte rutarea semnalelor
3. o memorie pentru configurare, care programează funcţionalitatea dispozitivului
4. resursele I/O

5. Suplimentar: blocuri de memorie RAM, circuite de calcul (MAC-uri pentru DSP), core-uri se microprocesor, transeivere etc.

Programarea unui FPGA

- un program (secvenţă binară complexă) încărcat într-o memorie CMOS statică – pe principiul registrului de deplasare.
- conţinutul acestui SRAM este aplicat apoi liniilor de control ale porţilor de transmisie şi celorlalte dispozitive
- se programează
  1. funcţionalitatea blocurilor funcţionale
  2. se configurează diverşi parametri şi caracteristici
  3. se stabileşte conectarea între diferitele blocuri funcţionale
  4. sunt configurate porturile bidirecţionale I/O

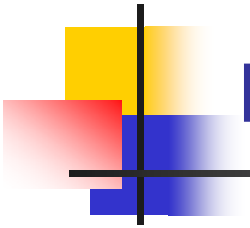


De ce avem nevoie de FPGA-uri?

- alternativă pentru ASIC (implementează pe un singur chip toată logica suplimentară aferentă unui sistem digital)
- posibilitatea reprogramării
- implementare relativ facilă (nu industrial)
- aplicaţie particulară în dezvoltarea prototipurilor ASIC

FPGA

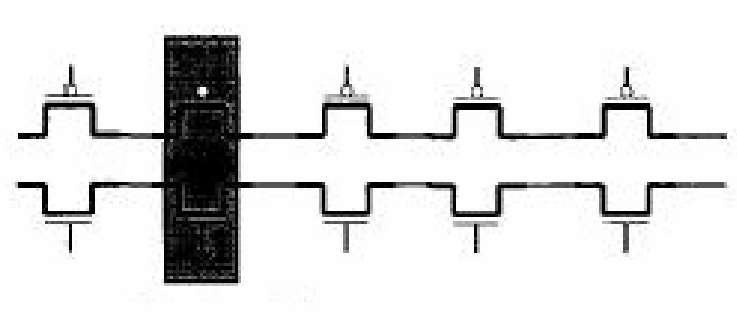
Blocuri logice



Principii

- crosspoint (tranzistoare)
- multiplexoare (Plessey, Actel)
- tabel de adevăr (look-up table, LUT) (Xilinx)
- celule PLD (Altera)

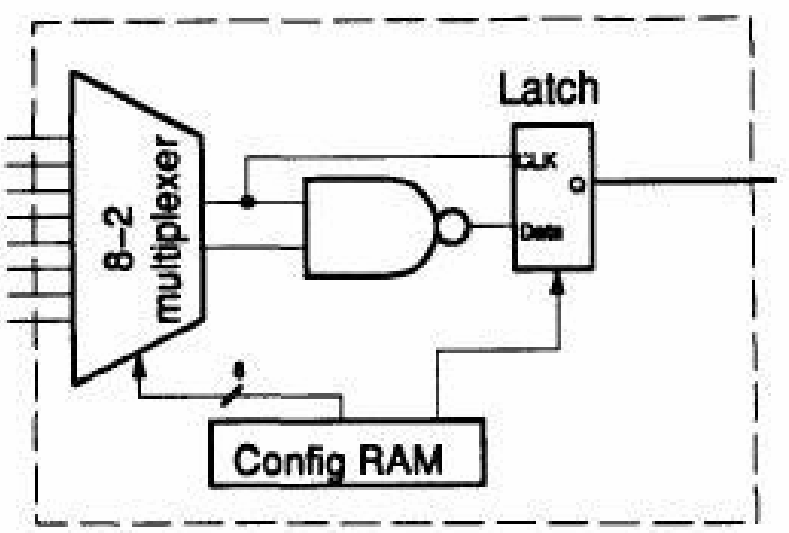
Crosspoint

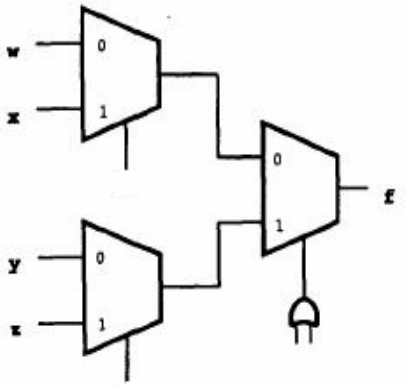


- lanţuri de perechi de tranzistoare
- blocuri RAM

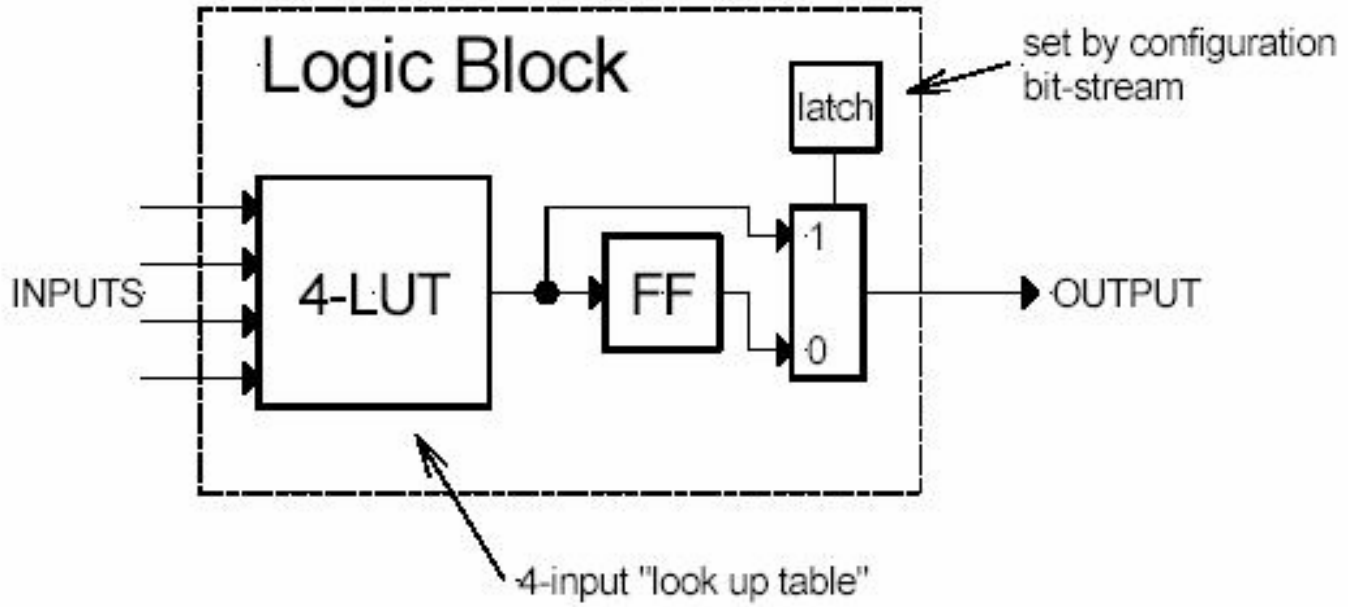
Multiplexoare

- Plessey (sus)
- Actel (jos)
- prin configurarea selecţiilor şi a unor intrări în MUX-uri se programează funcţia

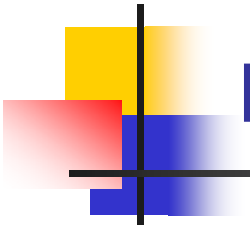




LUT (look-up table)



- implementarea funcţiilor cu ROM – se memorează tabelul de adevăr
- aici se memorează într-un RAM



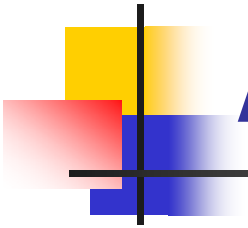
Dimensiune/performanţă

- un bloc logic de dimesiune mare implementează mai multe funcţii logice (o funcţionalitate mai complexă) – necesare mai puţine blocuri. Dar ocupă un spaţiu mai mare
- aria logică activă e mai mică decât aria logică totală datorită interconexiunilor programabile
- aria de rutare ocupă tipic mai mult spaţiu decât blocurile logice (70-90%)
- pentru tabele de adevăr: optim, 4 intrări
- granularitatea blocurilor logice influenţează performanţa unui FPGA (granularitate mare-întârzire redusă, dar fire mai lungi)

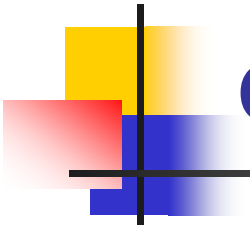
FPGA

tehnici de rutare

Arhitectura de rutare (conexiuni)



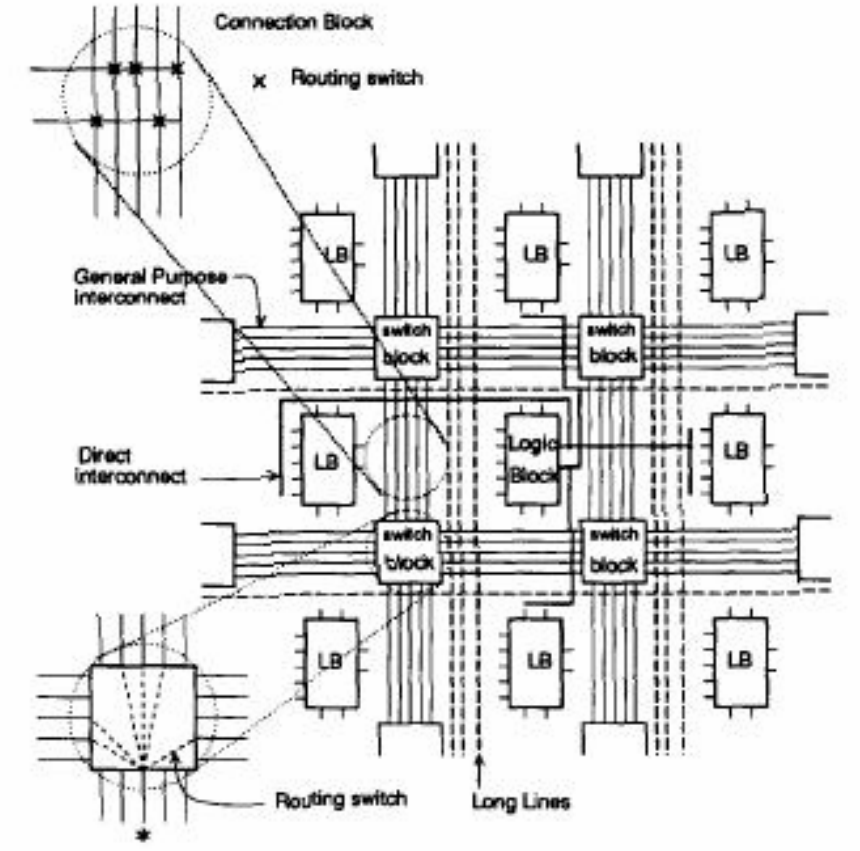
- “fire”
- comutatoare programabile
- între blocurile I/O şi blocurile logice, între diferite blocurilor logice
- tehnologia de rutare determină aria interconexiunilor şi densitatea blocurilor logice
- determină aria consumată de fire (segmente conexiuni), prin comparaţie cu aria blocurilor logice

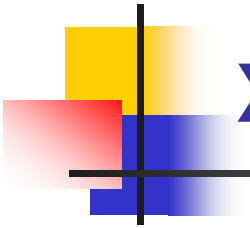


Clasificare după organizarea rutării

- Matrice (simetrică) – Xilinx
- Liniar (asimetrică) – Actel
- Ierarhică - Altera

Xilinx

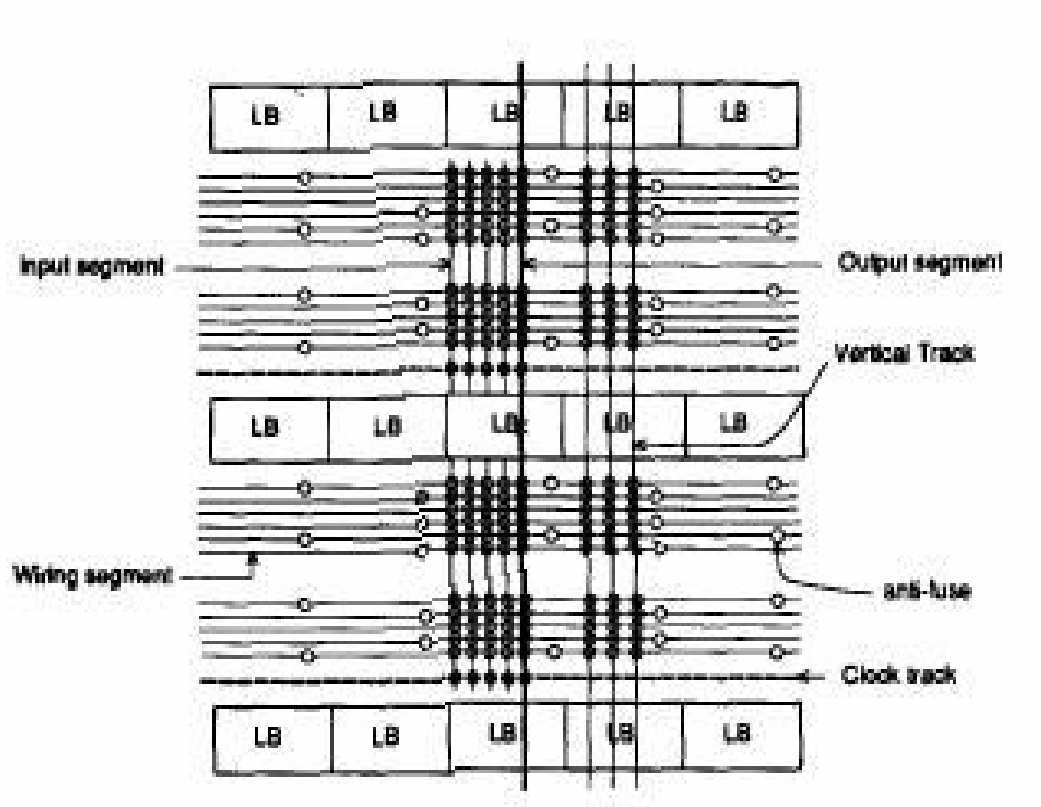

Connection Block
Routing switch
General Purpose interconnect
Direct interconnect
LB
switch block
Logic Block
Routing switch
Long Lines

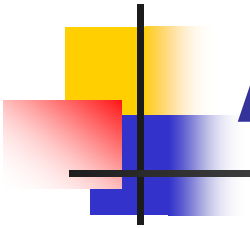


Xilinx

- fiecare bloc logic (LB) e conectat la un bloc de conexiuni (programabile) cu 4 secţiuni, se leagă pinii blocurilor funcţionale la segmentele de interconectare
  - tranzistoare de trecere pentru ieşiri,
  - multiplexoare pentru intrări
- fan-out mare (ieşirile se pot lega la oricâte conexiuni)
- 4 tipuri de segmente (fire, conexiuni)
  1. generale (cele care trec prin blocurile de switch-uri)
  2. interconexiuni directe (fiecare bloc logic e conectat la alte 4)
  3. linii lungi (fan-out mare, întârziere uniformă)
  4. linii de ceas (accesibile în orice punct al chipului)

Actel

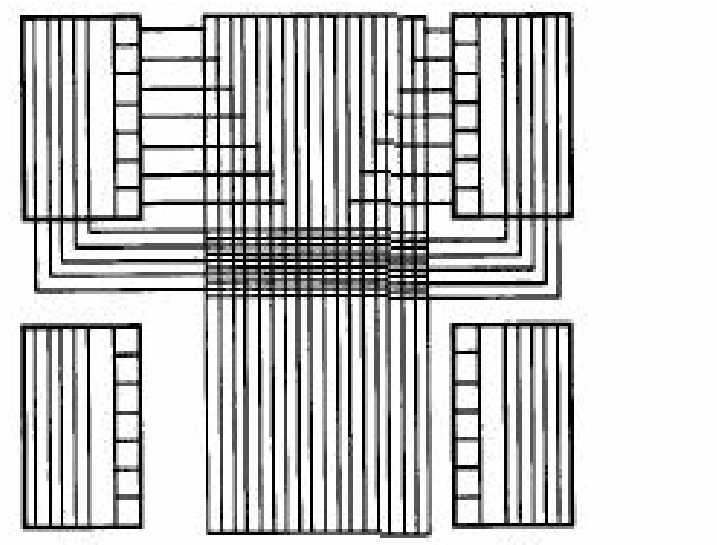

LB
LB
LB
LB
LB
Input segment
Output segment
Vertical Track
LB
LB
LB
LB
LB
Wiring segment
anti-fuse
Clock track
LB
LB
LB
LB
LB

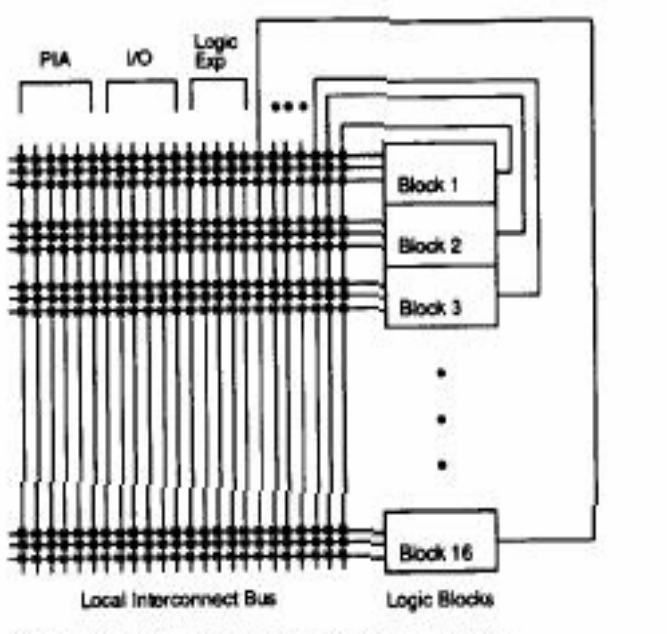


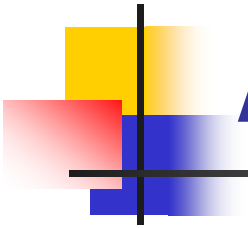
Actel

- mai multe conexiuni pe orizontală decât pe verticală
- pinii de intrare ai blocurilor logice sunt conectaţi la toate liniile orizontale de pe latura respectivă
- pinii de ieşire sunt proiectaţi fizic să intersecteze doar 4 linii (2 sus şi 2 jos) şi pot fii conectaţi doar la aceştia
- Blocurile de switch-uri sunt distribuite de-a lungul canalelor orizontale
- orice linie verticală poate fi conectată la orice linie orizontală

Altera (conexiuni globale şi locale)







Altera

- 2 nivele ierarhice de rutare
- primul nivel: 16 sau 32 de blocuri logice sunt grupate (Logic Array Block) cu structură asemănătoare a unui CPLD (bus vertical)
- 4 tipuri de conexiuni
  1. de la fiecare ieşire a tuturor blocurilor logice din LAB
  2. de la expandoarele logice
  3. de la ieşirile unor blocuri logice din alte LAB-uri
  4. conexiuni de la şi către padurile I/O

conexiunile între LAB-uri constau din canale (segmente), conexiunea generală asemănătoare cu PIA de la CPLD

avantaj: regularitate

dezavantaj: număr mare switchuri, încărcare capacitivă