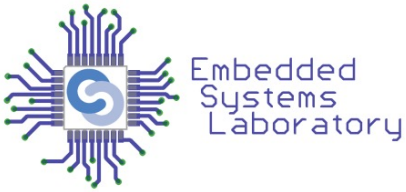
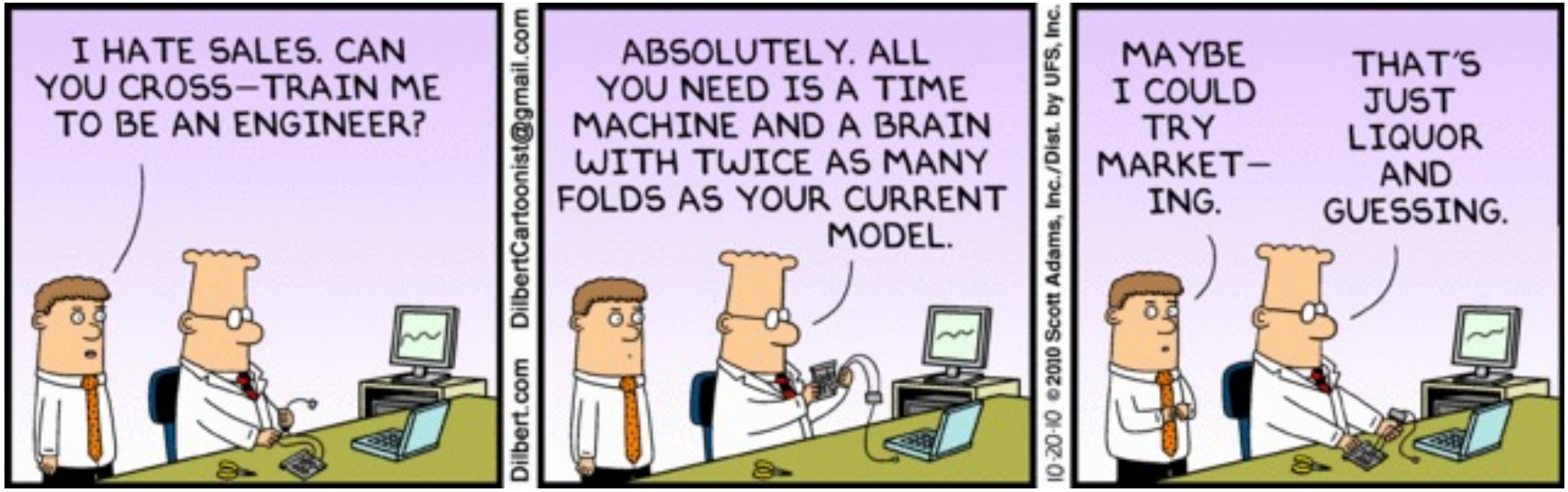


Sisteme Încorporate

Cursul 10

Sisteme Reconfigurabile

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

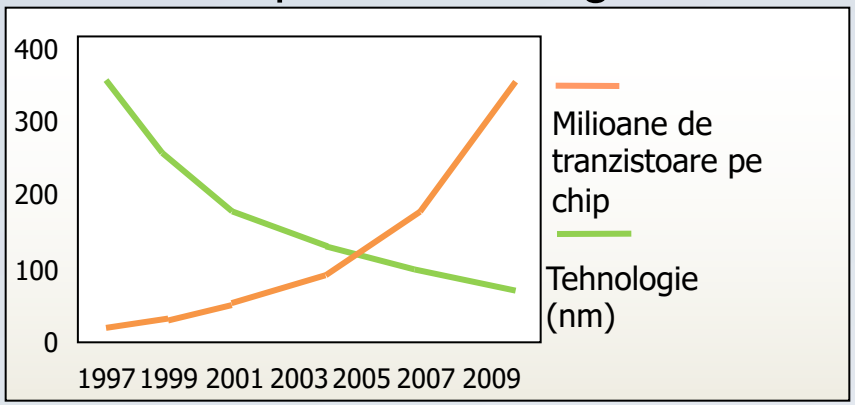


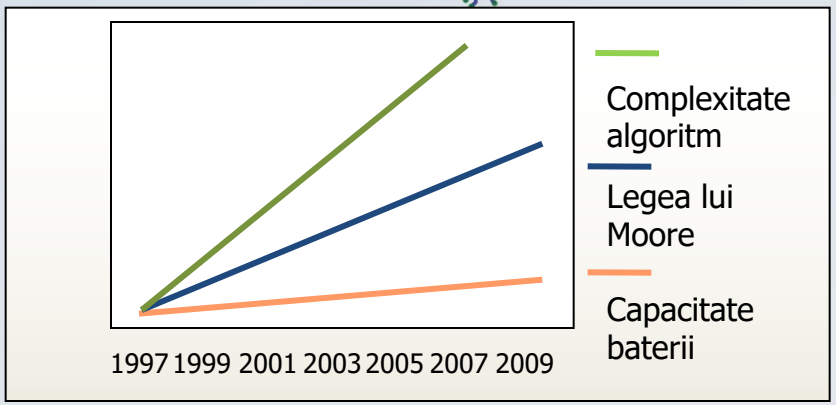
http://dilbert.com/strips/comic/2010-10-20/

De ce avem nevoie de reconfigurabilitate?

Embedded Systems Laboratory

- Densitatea foarte mare a tranzistoarelor în circuitele moderne
- Costuri sporite de integrare





- Complexitate crescută a algoritmilor
- Limitări puternice a capabilităților bateriilor

Se dorește
Și performantă
Și flexibilitate

Constrângeri severe pentru consumul de energie

Analiza algoritmilor pentru sisteme embedded

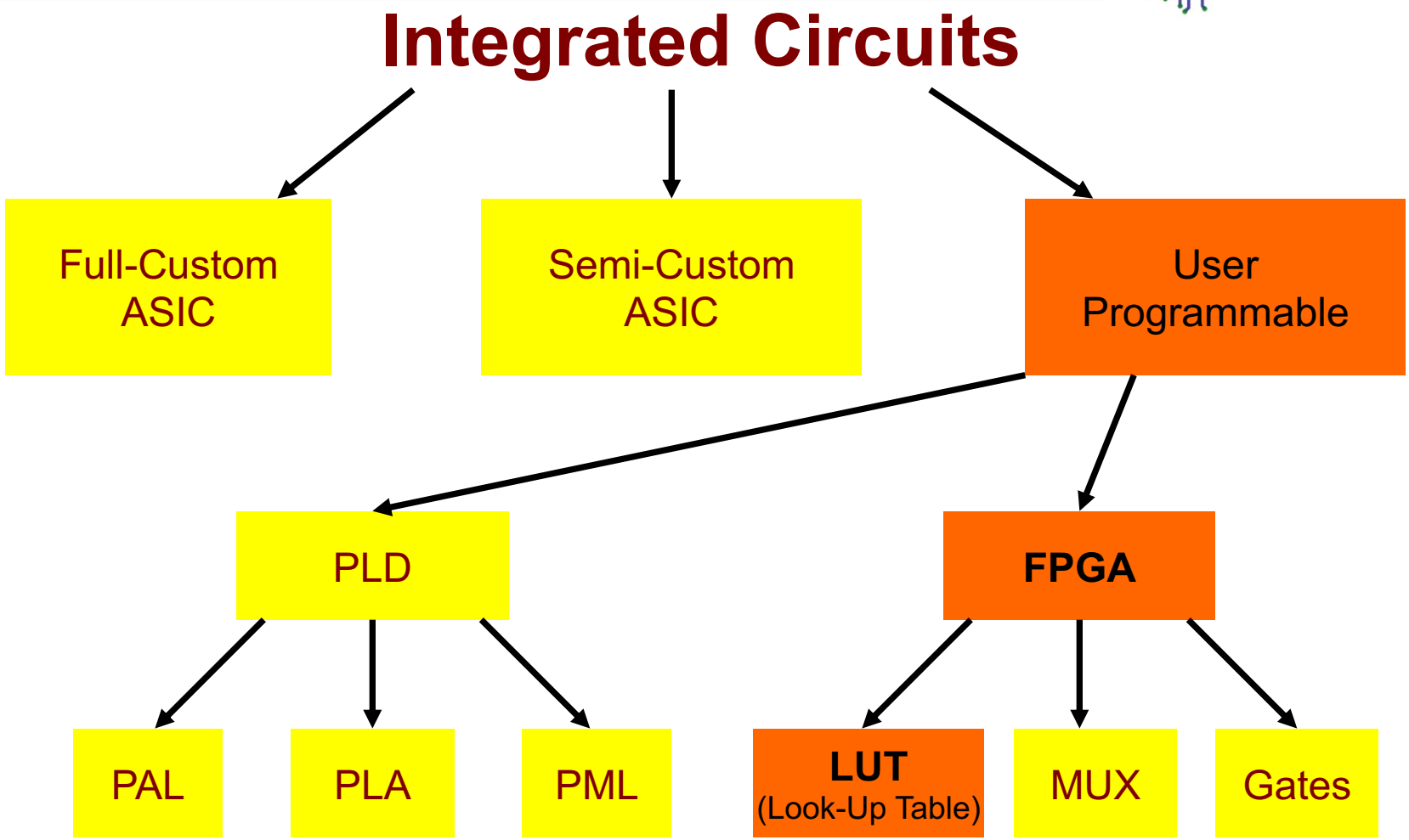
Embedded Systems Laboratory

- **90% din complexitatea unui algoritm este concentrată în porțiuni bine definite ce constituie o parte foarte mică din codul total**
- **Mulți algoritmi au porțiuni de cod care pot fi paralelizate**
  - ✓ Performanța este îmbunătățită prin folosirea căilor paralele de date
- **Granularitatea operanzilor este de obicei destul de diferită de 32 de biți**
  - ✓ Un UAL tradițional este ineficient (consumă prea multă energie)

Se pot obține îmbunătățiri semnificative dacă funcționalitatea procesoarelor embedded este extinsă cu funcții specifice aplicației

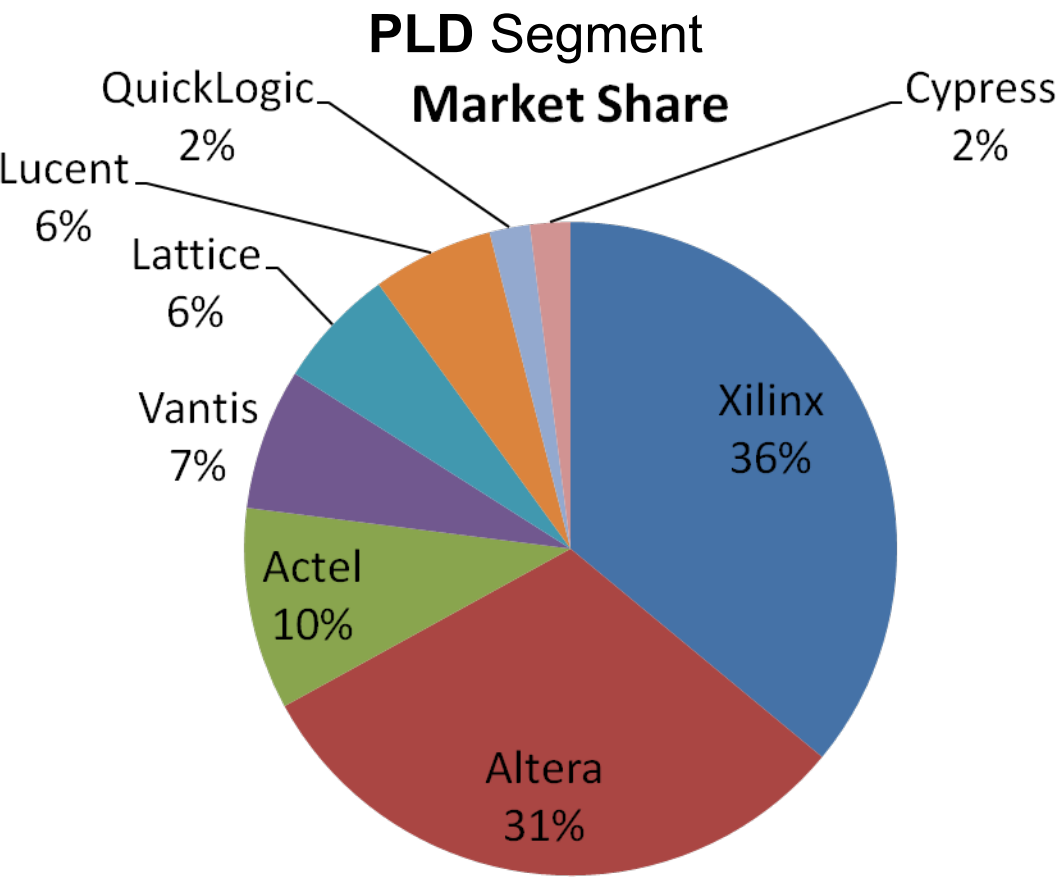
Sisteme de calcul reconfigurabile pentru un maxim de flexibilitate

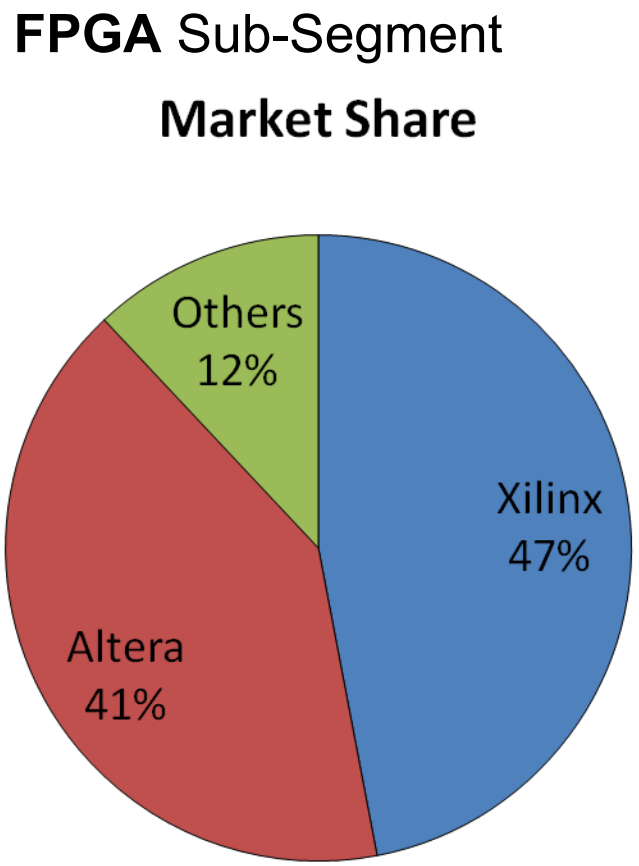
Tipuri de circuite integrate



Piața de circuite reconfigurabile

Calendar Year 2013





Doi producători majori – piață ajunsă la maturitate

http://sourcetech411.com/2013/04/top-fpga-companies-for-2013/

Două implementări concurente

ASIC
Application Specific Integrated Circuit

- un design trebuie trimis pentru fabricare într-un fab de semiconductoare. Procesul durează timp și este foarte costisitor.
- sunt proiectate complet, de la descrierea comportamentală până la layout-ul fizic

FPGA
Field Programmable Gate Array

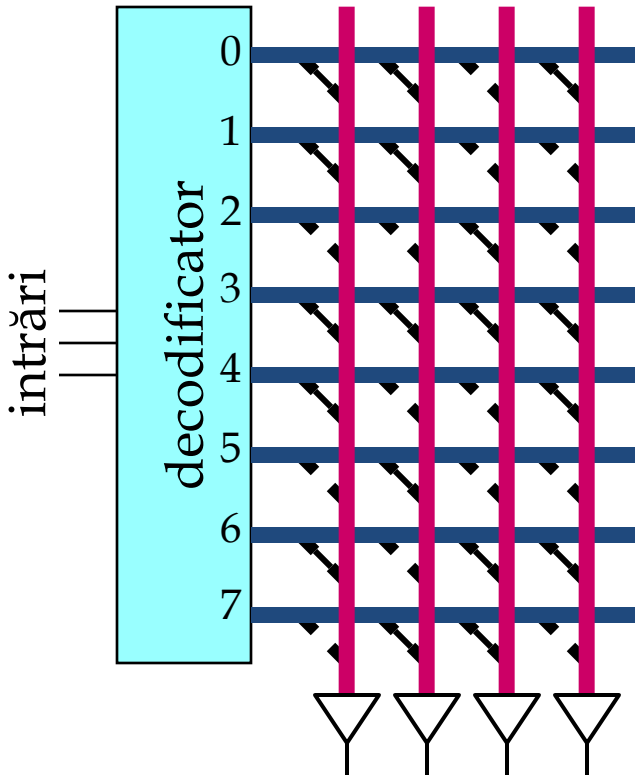
- reconfigurat de fiecare dată de către proiectanți
- nu se proiectează un layout fizic al componentelor. Proiectarea are ca rezultat un bitstream cu care se configurează dispozitivul.

Programmable Logic Device (PLD)

- Sunt matrici simple de circuite logice
  - Implementeaza funcții logice pe două niveluri (AND/OR)
  - Au o structură simplă de interconectare programabilă
  - Sunt de mai multe tipuri
    - Read Only Memory (ROM si PROM)
    - Programmable Logic Array (PLA)
    - Programmable Array Logic (PAL)
- Field Programmable Gate Arrays (FPGA)
  - Multe copii ale aceleiași structuri configurabile de bază
  - Fiecare bloc poate fi configurat pentru a îndeplini orice funcție logică și include de obicei un flip-flop și un generator de funcții cu 4 intrări
  - Interconectare programabilă
  - Blocuri de memorie SRAM
  - Un FPGA mare are de obicei 100k+ circuite flip-flop, 100k generatoare de funcții și 10Mb SRAM

Implementarea unei memorii ROM

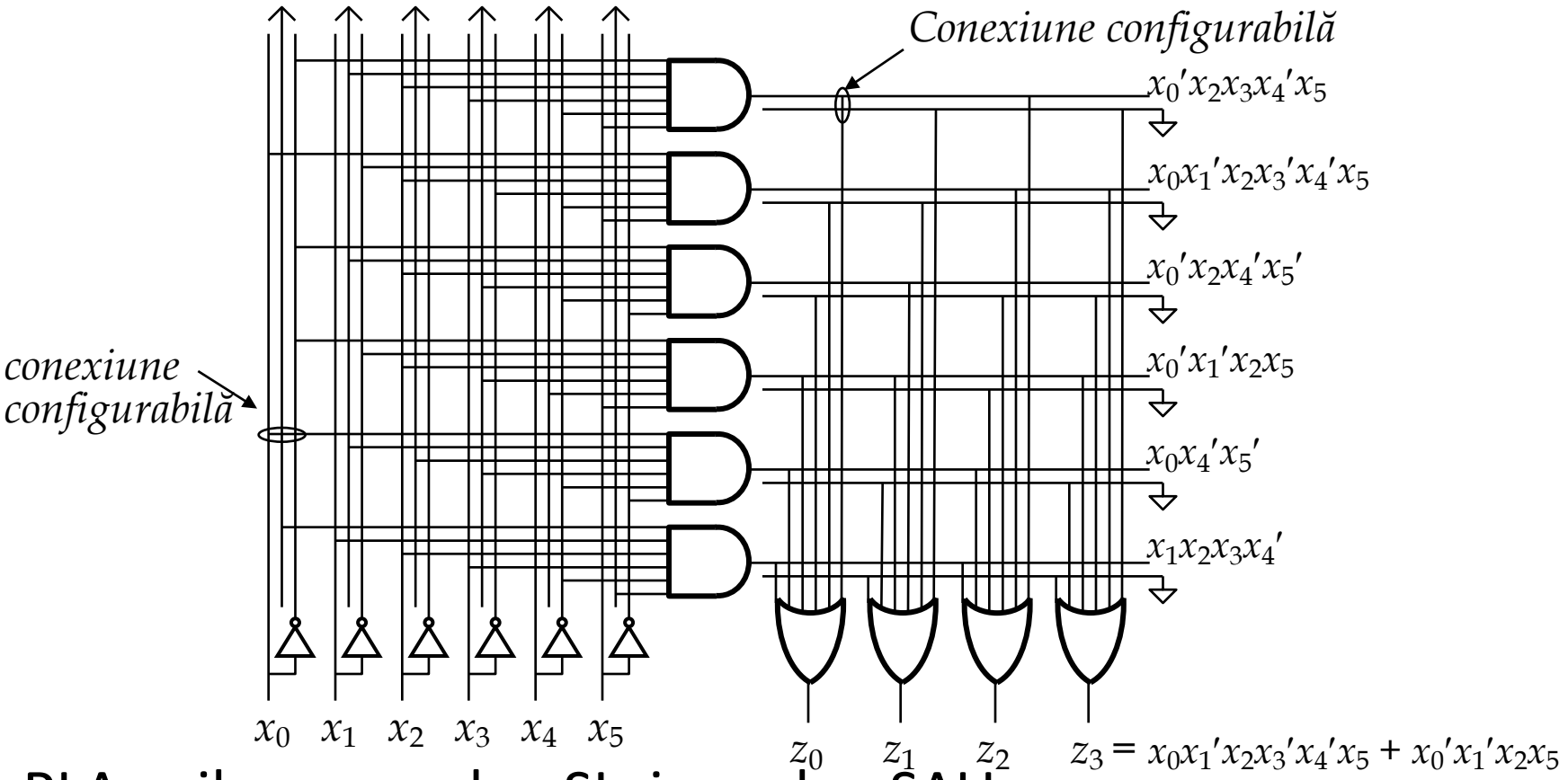
- ROM poate fi implementată printr-un aranjament ortogonal de conexiuni
  - Conexiune la fiecare intersecție = 1 logic
  - Decodificatorul pune 1 logic pe linia selectată iar dacă conexiunea este facută, la ieșire se poate citi un octet de date
- Unele PROM-uri sunt scrise prin eliminarea conexiunilor
  - Tensiune mare aplicata pe linia și coloana pe care se vrea marcarea unui 0 logic
  - Curentul mare aplicat legăturii linie-coloană determină “arderea” legăturii
- Alte memorii pot fi arse și reprogramate (EPROM, EEPROM)



Logica în RAM/ROM

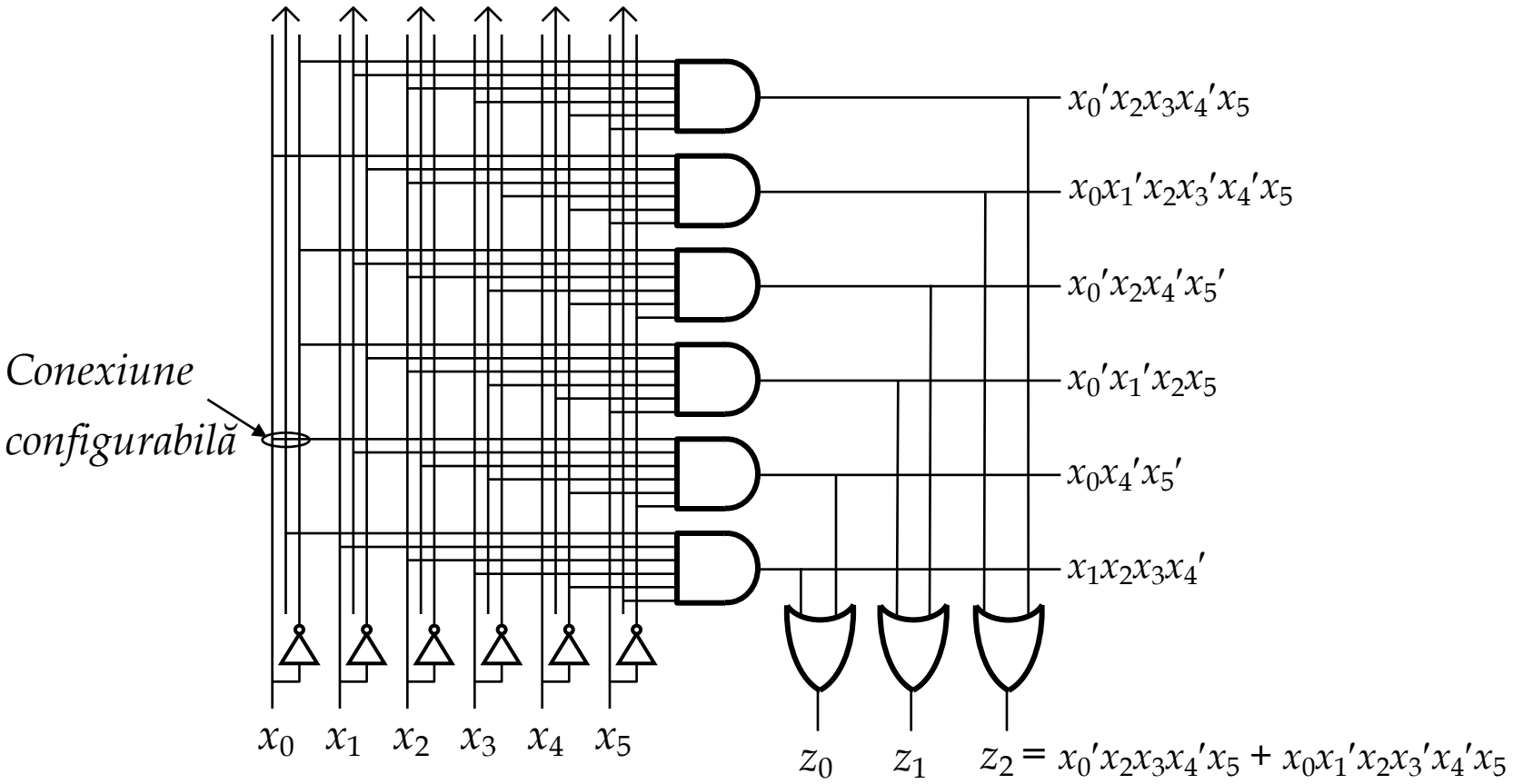
- Orice memorie RAM poate implementa funcții logice reconfigurabile
  - Stochează tabela de adevăr a funcției logice în memorie
  - Exemplu: folosesc RAM de 4 biți pentru a simula o poartă ȘI cu două intrări stocând 0 la adresele 00, 01 și 10 și 1 la adresa 11
  - cu 2^m cuvinte de 1 bit se poate implementa orice funcție de *m* intrări
  - O memorie cu 2^m cuvinte de *w* biți lățime poate implementa *w* funcții logice diferite cu *m* intrări
- Memoriile ROM sau EEPROM pot implementa aceleași funcții logice dar cu anumite avantaje:
  - Memoria este nevolatilă
  - Datele sunt stocate la programare și pot fi reconfigurate printr-un update de firmware
  - Densitate mai mare decât memoria RAM

Programmable Logic Array (PLA)



- PLA-urile au un plan ȘI și un plan SAU
- Pot să implementeze orice circuit de două niveluri (SAU/ȘI)
- Implementate în CMOS.

Programmable Array Logic (PAL)

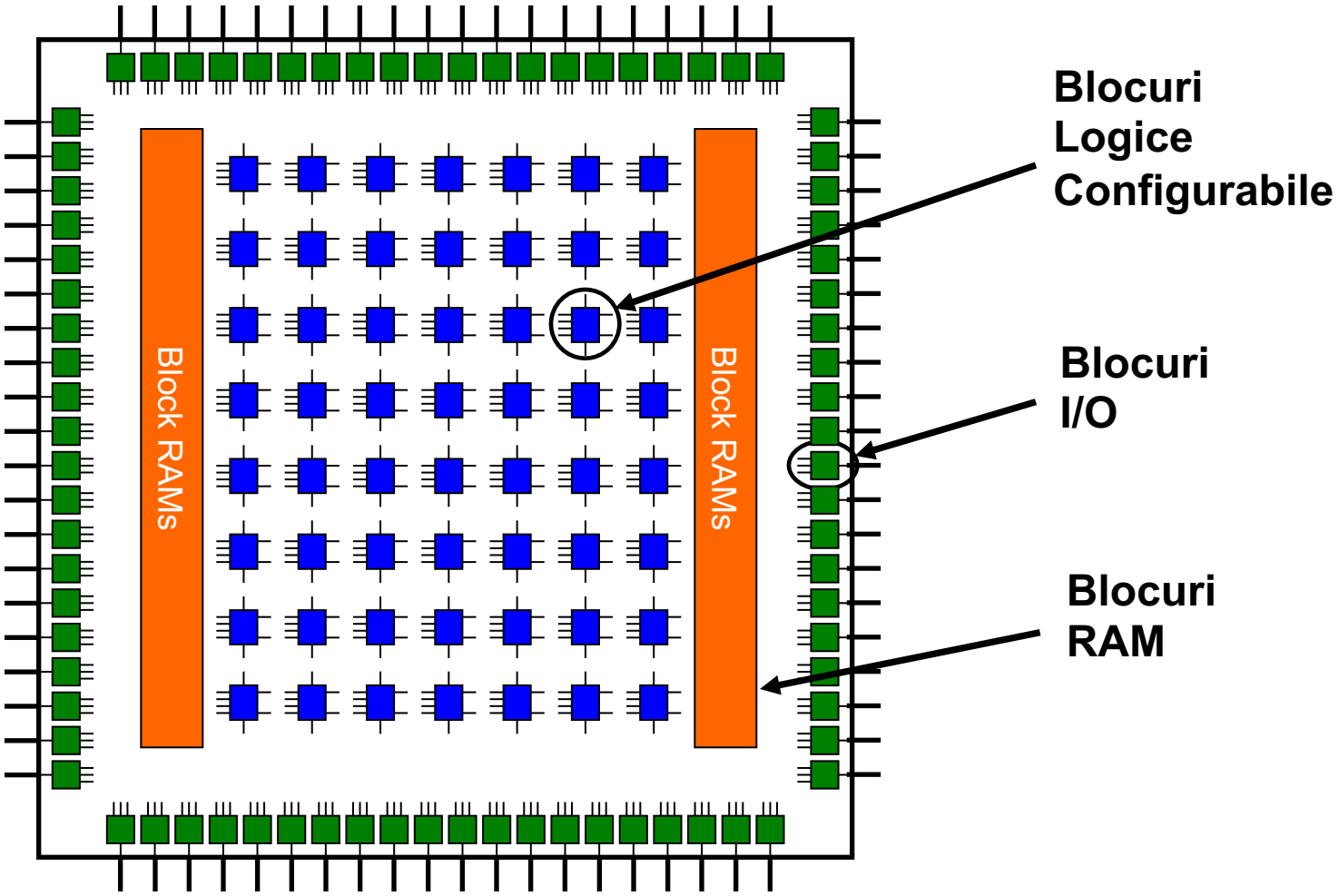


- PAL este similar cu PLA dar are un plan SAU fix.
- Mai ușor de programat (și mai ieftin de produs)
- Dezavantaj: număr limitat de termeni generați la ieșire

Field Programmable Gate Array (FPGA)

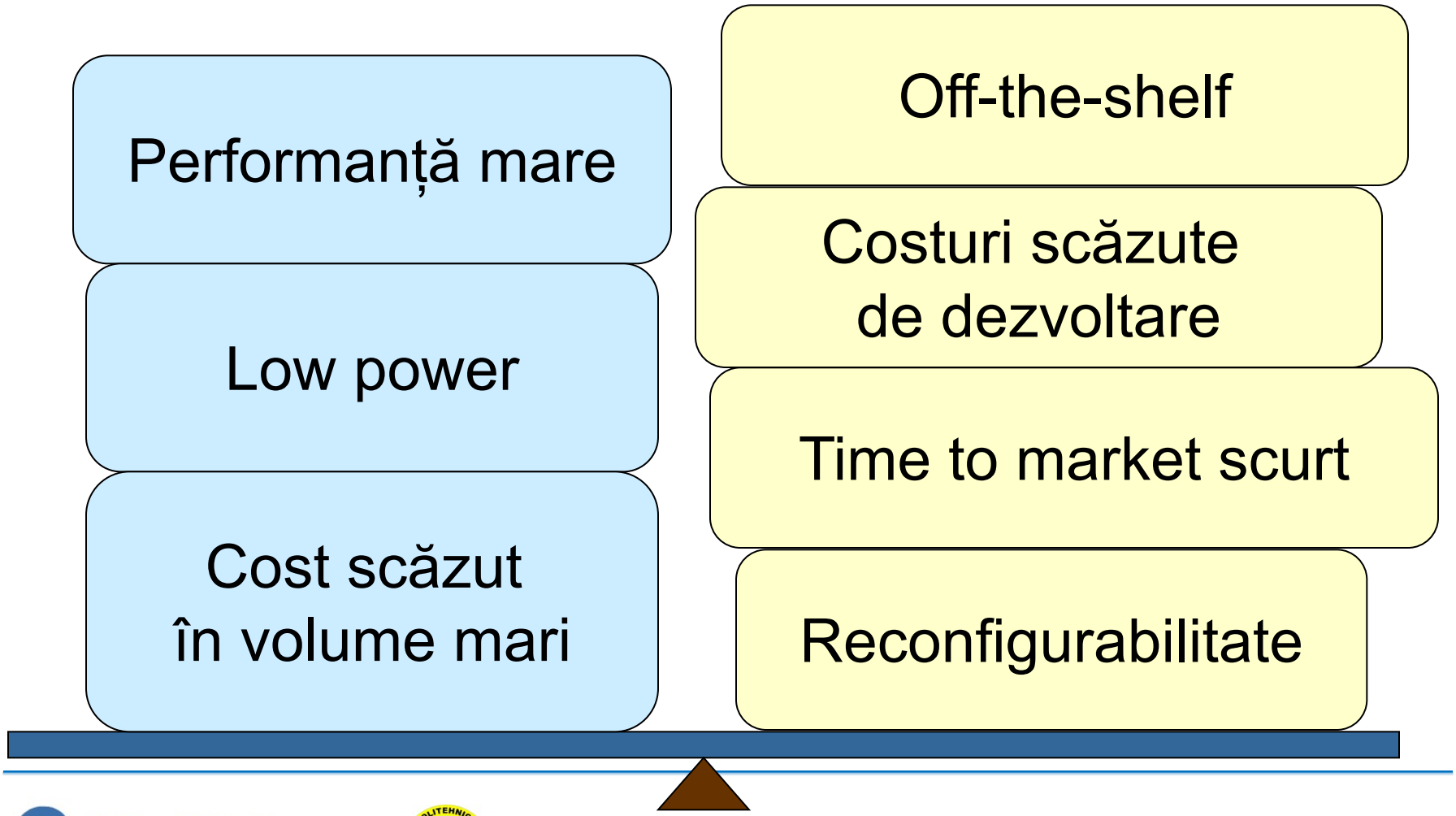
- Circuitele FPGA sunt folosite pentru generarea circuitelor logice complexe.
- Un chip conține un număr foarte mare (zeci de mii) de blocuri logice configurabile.
  - Programele de tip CAD mapează circuitele de nivel înalt peste matricea de blocuri de bază prin configurarea generatoarelor de funcții, interconexiunilor și altor elemente configurabile
- Blocurile logice sunt rutate folosind interconexiuni programabile
  - Segmentele sunt conectate la blocurile logice și la alte segmente învecinate prin switch-uri configurabile
  - Programele CAD determină configurația optimă pentru toate switch-urile folosite.

Ce este un FPGA?



Care sunt opțiunile?

Embedded Systems Laboratory



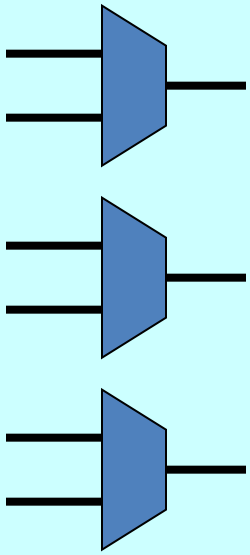
Alte avantaje FPGA

- Ciclul de fabricație al unui ASIC este foarte costisitor, lung și necesită proiectanți din mai multe domenii
  - Greșelile care nu sunt detectate la proiectare au un impact foarte mare asupra asupra costurilor și a timpului de dezvoltare
- FPGA-urile sunt perfecte pentru prototiparea rapidă a unui circuit digital
- Se pot face upgrade-uri foarte usor, ca și cum ar fi în cazul unui software
- Domenii unice de aplicații
  - Sisteme reconfigurabile

Lățimea de bandă computațională

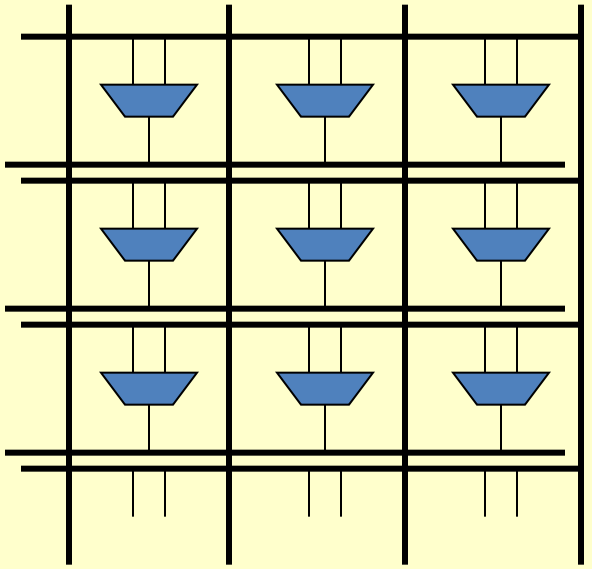
Embedded Systems Laboratory

Fixă



CPU

"Fără limite"



RC

Producătorii majori de FPGA

Embedded Systems Laboratory

FPGA SRAM

- **Xilinx, Inc.**
- **Altera Corp.**

} **Aproape 60% din piață**

- Atmel
- Lattice Semiconductor

FPGA Flash

- Actel Corp.
- Quick Logic Corp.

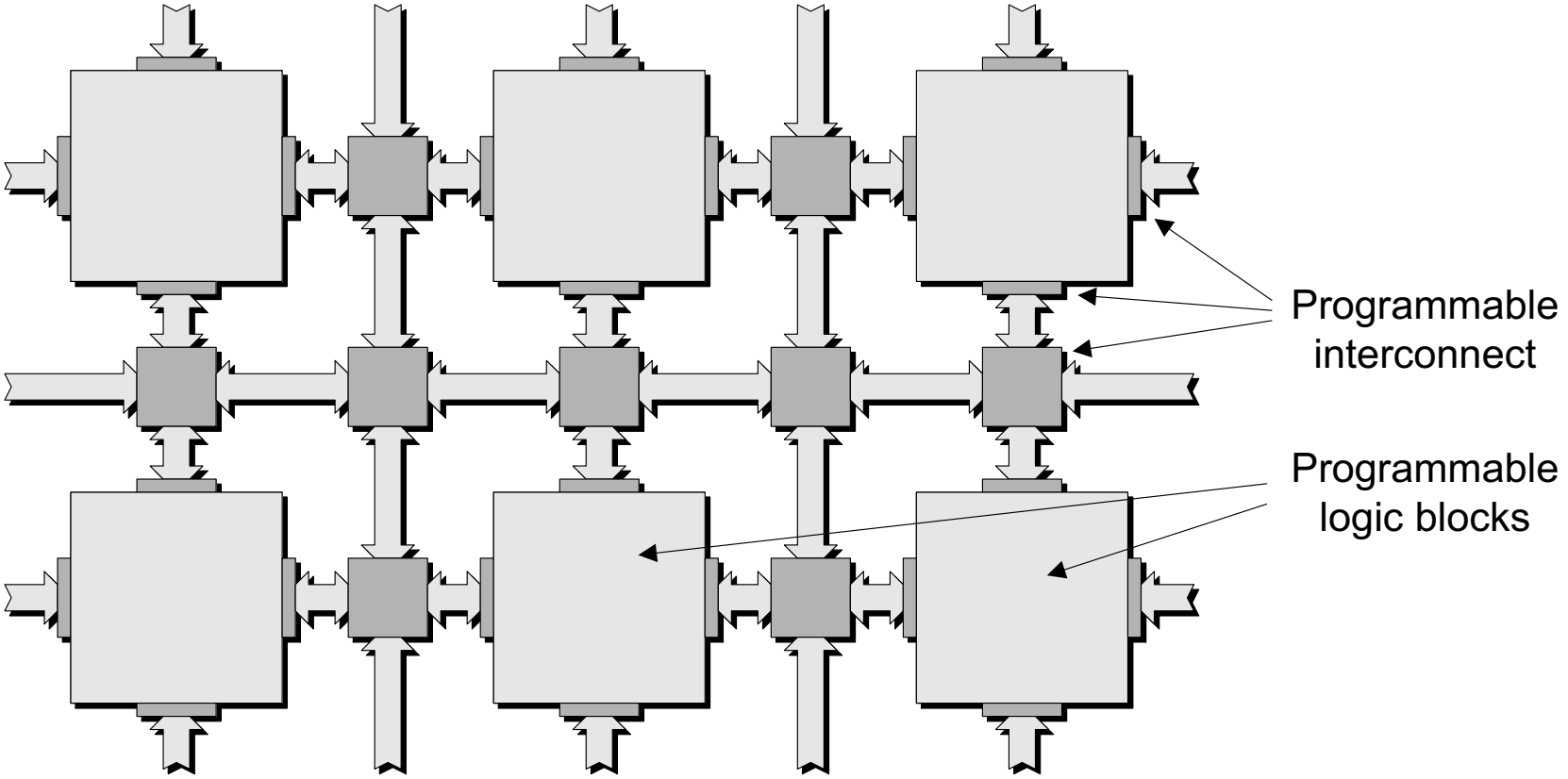
Familii de FPGA Xilinx

Embedded Systems Laboratory

- Familii vechi
  - XC3000, XC4000, XC5200
  - Tehnologie 0.5µm, 0.35µm si 0.25µm . Nerecomandate pentru produse noi.
- **Familii High-Performance**
  - Virtex (0.22µm)
  - Virtex-E, Virtex-EM (0.18µm)
  - Virtex-II, Virtex-II PRO (0.13µm)
  - Virtex-4 (0.09µm)
  - Virtex-5
- **Familii Low Cost**
  - Spartan/XL – derivat din XC4000
  - Spartan-II – derivat din Virtex
  - Spartan-IIE – derivat din Virtex-E
  - **Spartan-3**
  - **Spartan-3E**

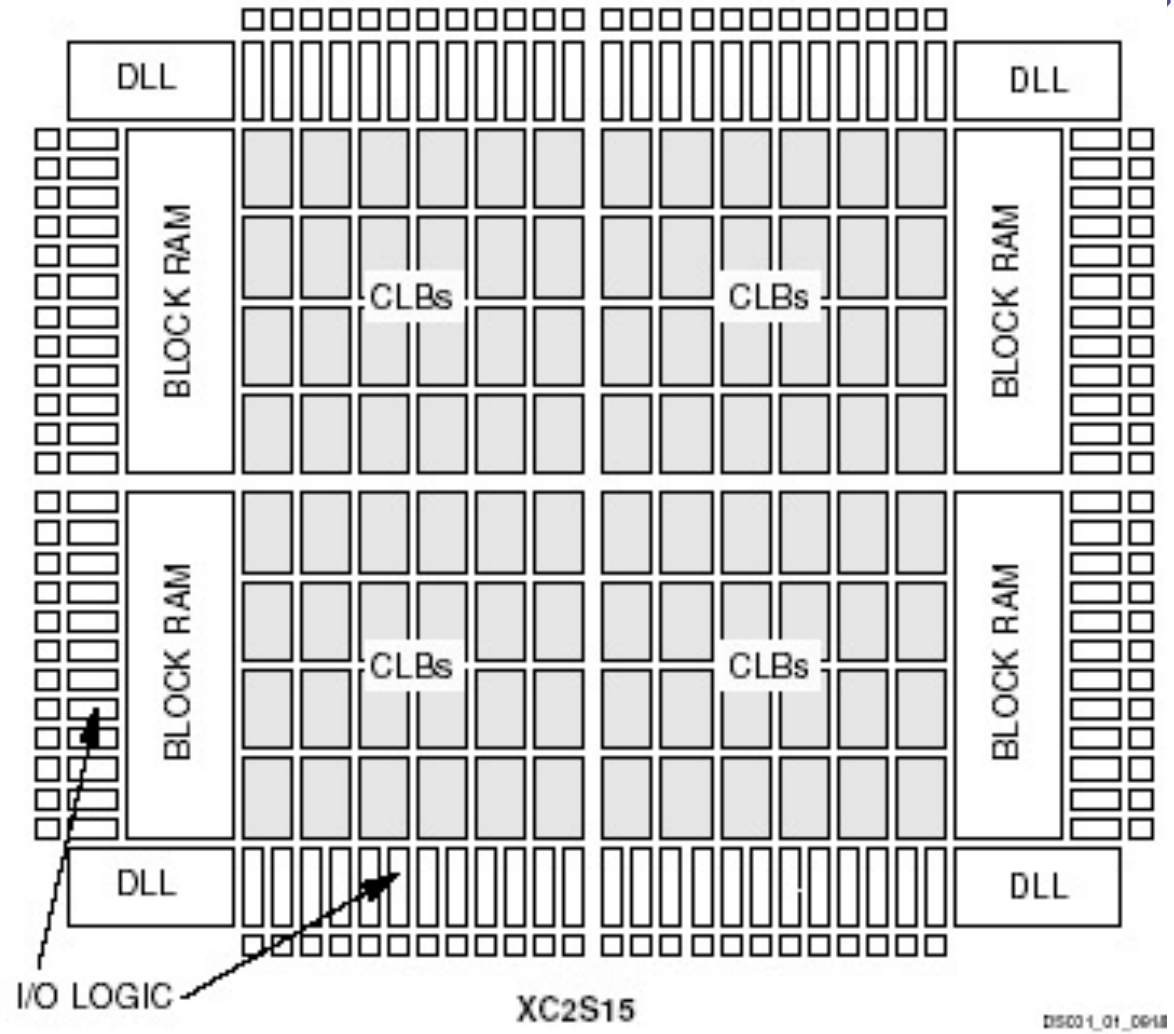
Prezentare

Structura generală a unui FPGA

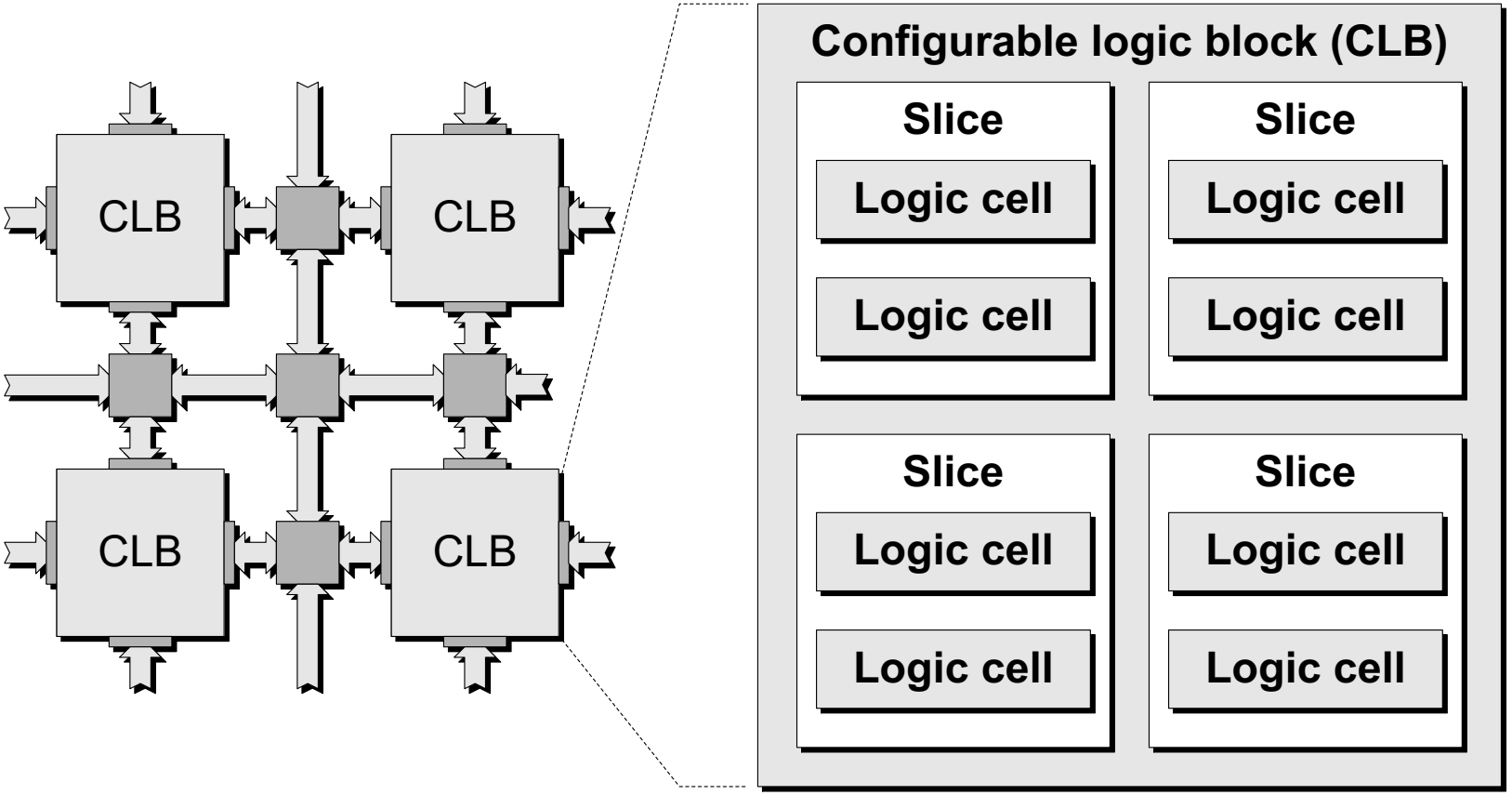


The Design Warrior’s Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Xilinx FPGA Block Diagram

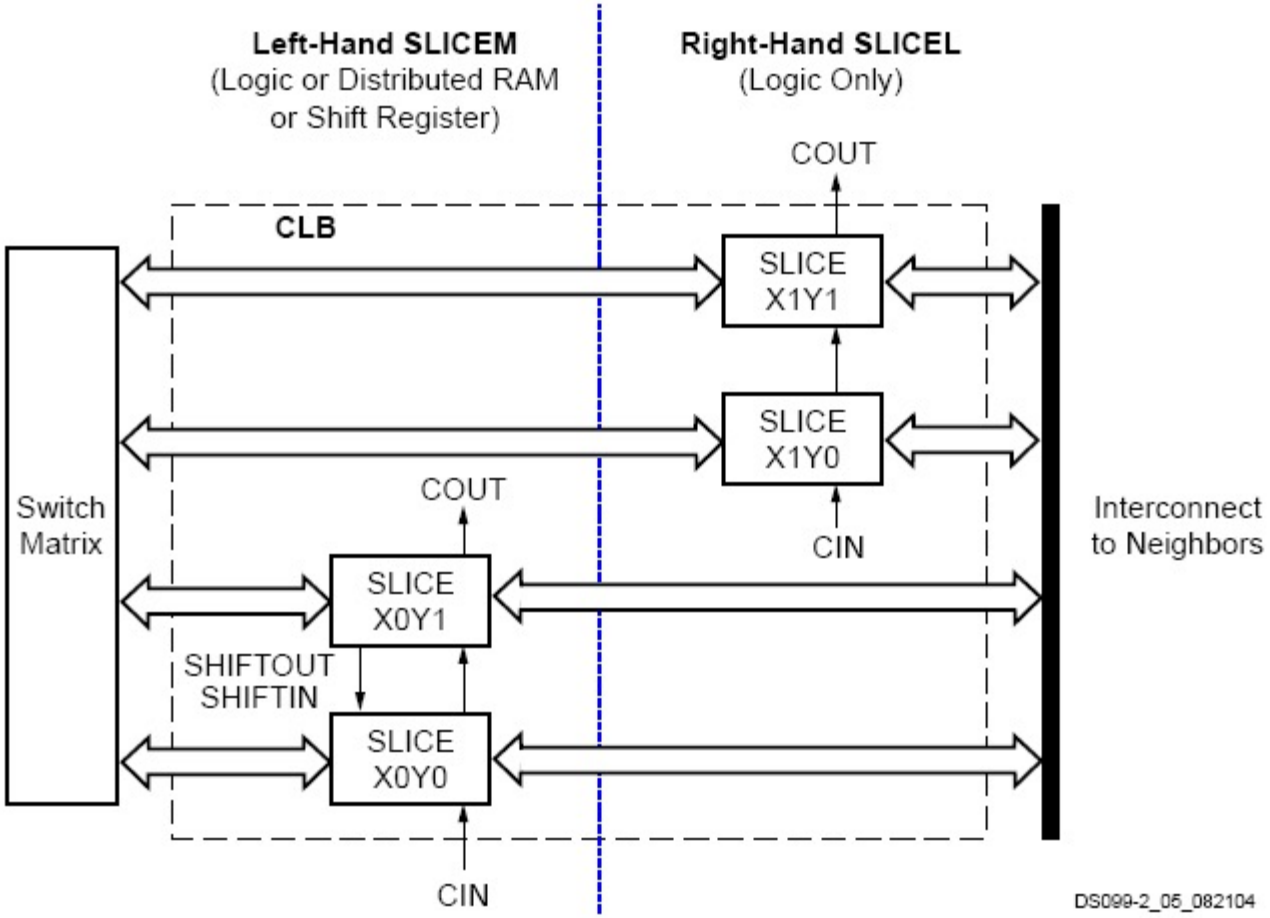


Xilinx CLB

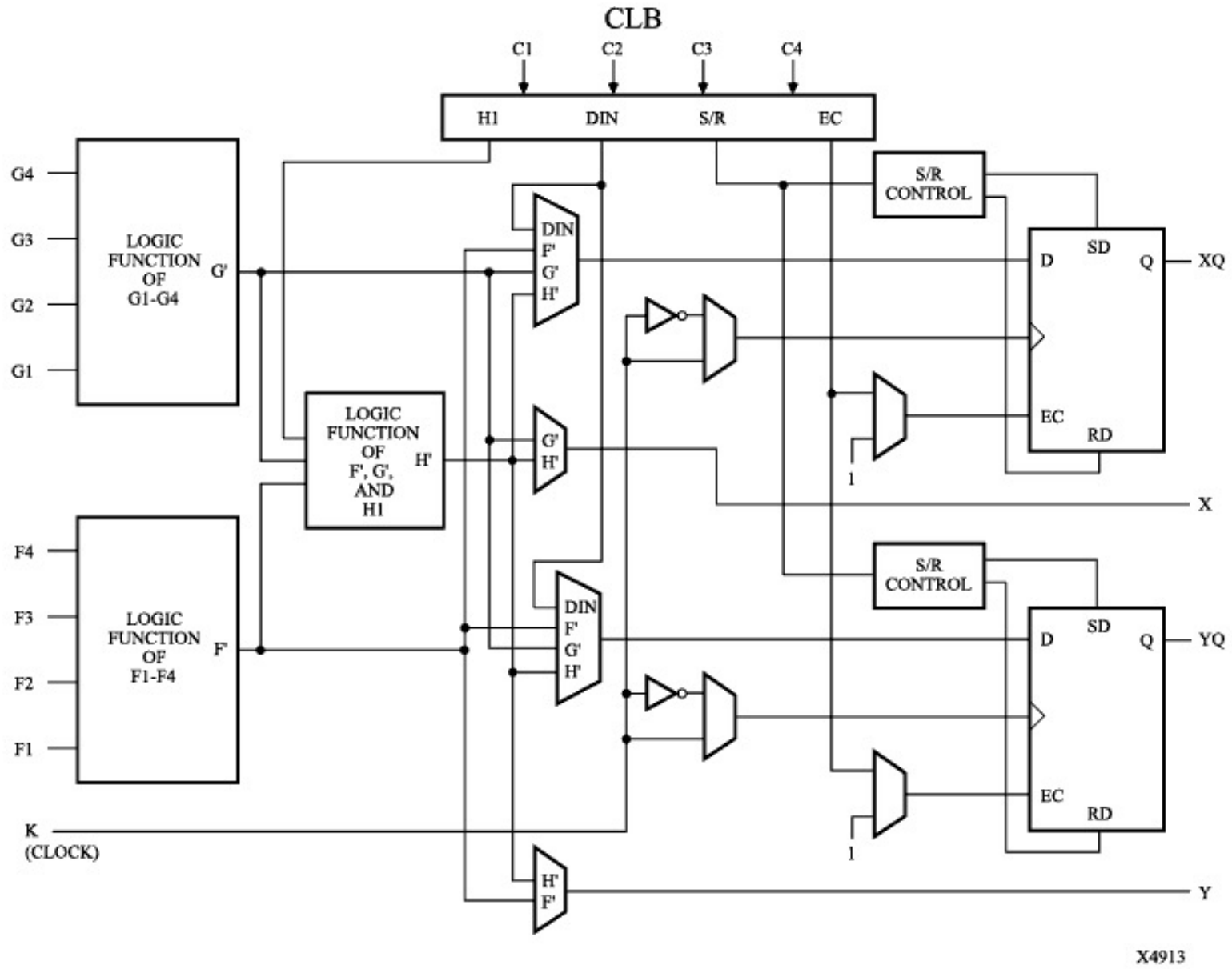


The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Configurable Logic Block CLB

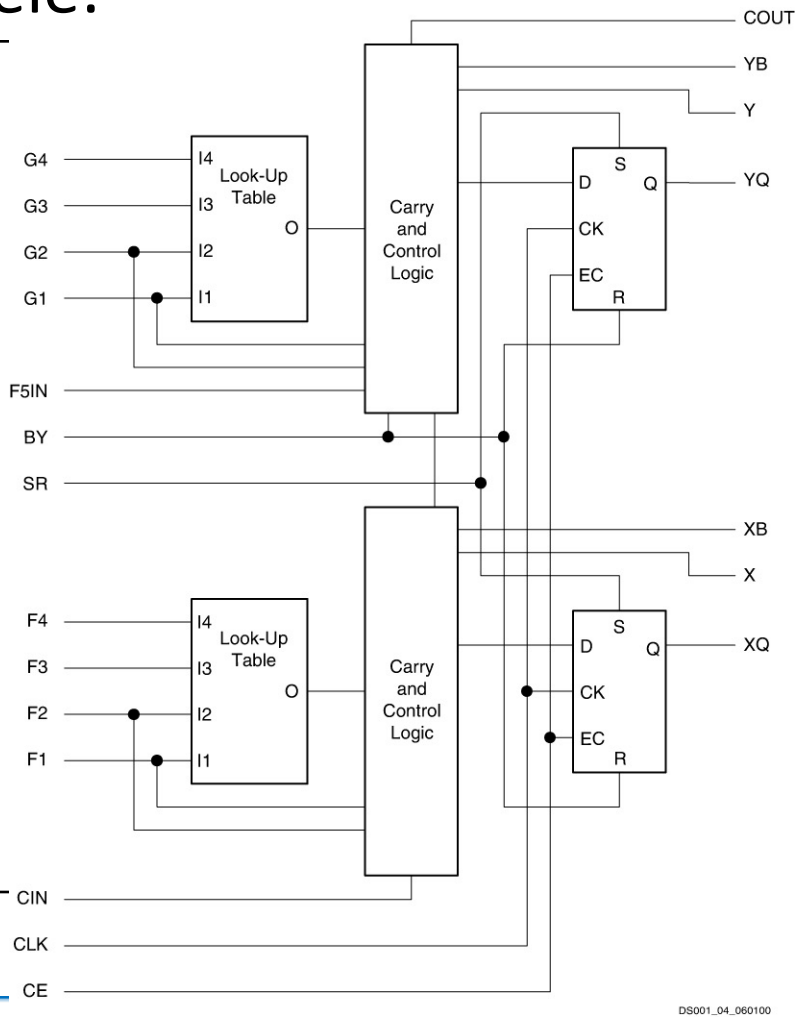


CLB Slice


CLB
C1
C2
C3
C4
H1
DIN
S/R
EC
G4
G3
G2
G1
LOGIC FUNCTION OF G1-G4
G'
LOGIC FUNCTION OF F', G', AND H1
H'
F4
F3
F2
F1
LOGIC FUNCTION OF F1-F4
F'
K (CLOCK)
DIN
F'
G'
H'
G'
H'
H'
F'
S/R CONTROL
S/R CONTROL
SD
D
Q
XQ
EC
RD
1
X
YQ
Y
X4913

Structura unui slice CLB

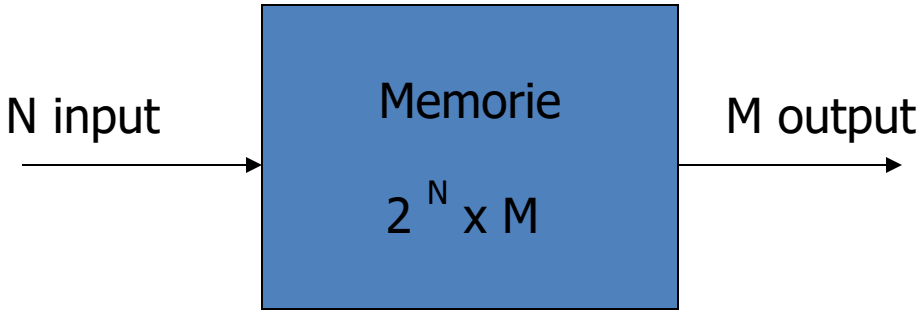
- Fiecare slice conține următoarele:
  - LUT cu patru intrări
    - Orice funcție logică cu 4 intrări,
    - sau RAM 16bit x 1
    - sau Registru Shift pe 16 biți
  - Carry & Control
    - Logică aritmetică rapidă
    - Logică pentru înmulțire
    - Logică de multiplexare
  - Elemente de stocare
    - Latch sau flip-flop
    - Set / reset
    - Ieșiri normale sau inversate
    - Control pentru funcționare sincron/asincron



Funcțiile Logice

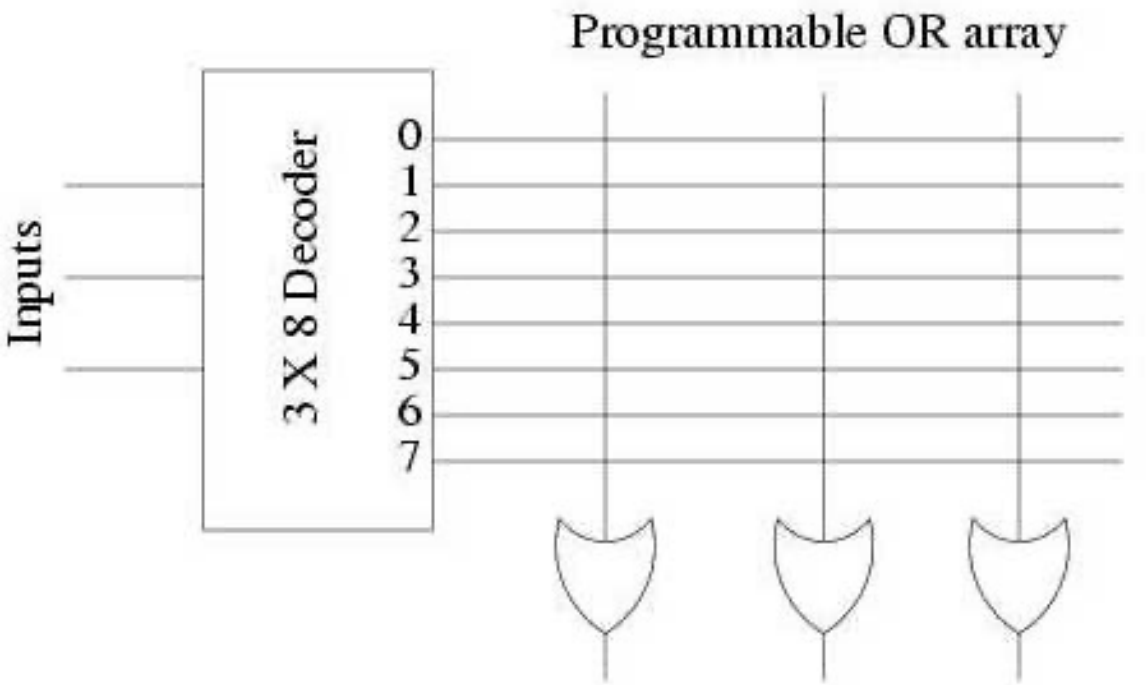
- Implementate folosind Look-up Table (LUT)
- Un LUT cu k intrări corespunde unei memorii de 2^K x 1 bit
- Un LUT cu k intrări poate implenta orice funcție logică cu k intrări și o singură ieșire

Lookup Table (LUT)



- Adrese: N biți; Ieșire: M biți
- Memoria conține 2^N cuvinte a câte M biți
- Valorile intrărilor decid cuvântul care va fi disponibil la ieșire la un moment dat

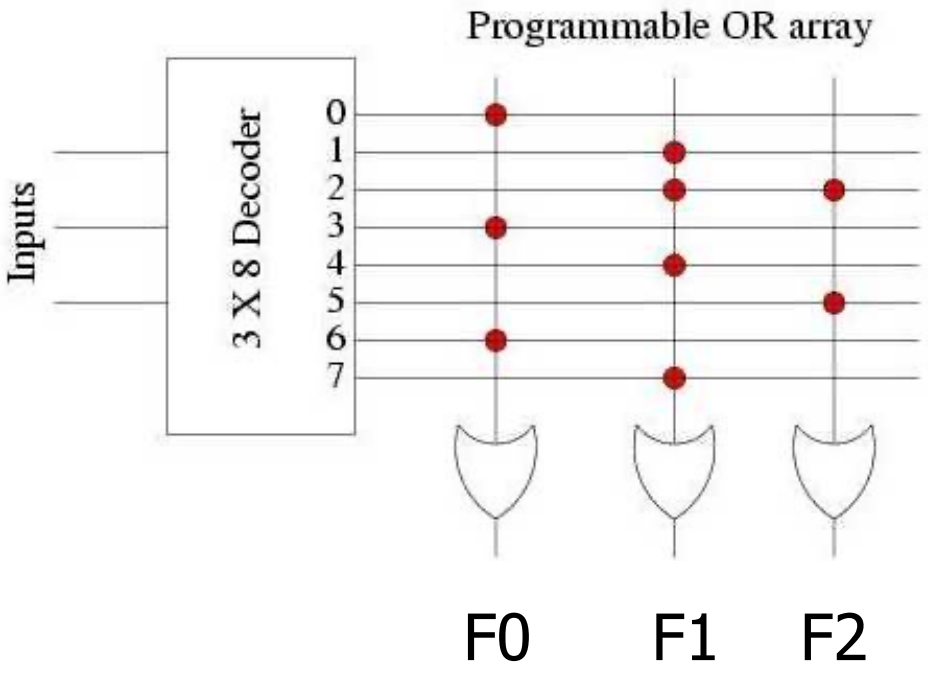
Diagrama logică a unui LUT 8x3



Sumă de mintermeni

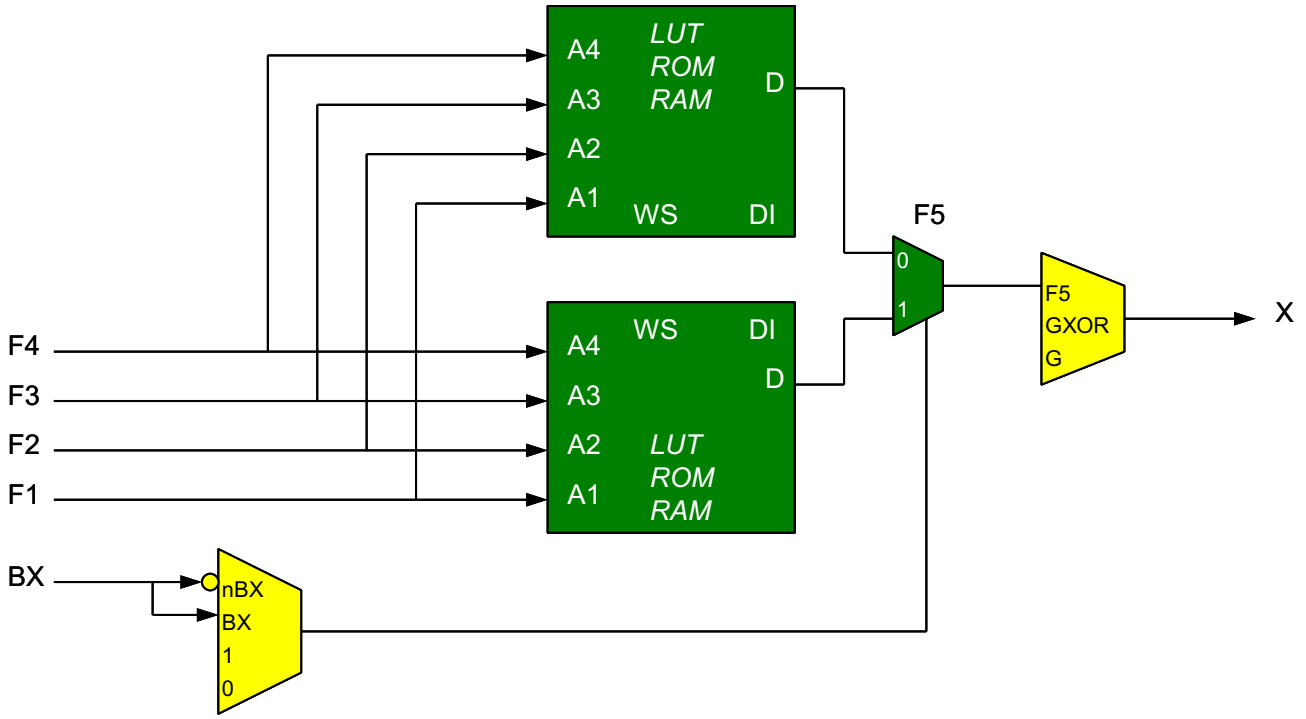
Implementarea unei funcții logice folosind LUT

I0	I1	I2	F0	F1	F2
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	1	0	0
1	1	1	0	1	0



Implementarea unei funcții logice cu 5 intrări folosind 2 LUT

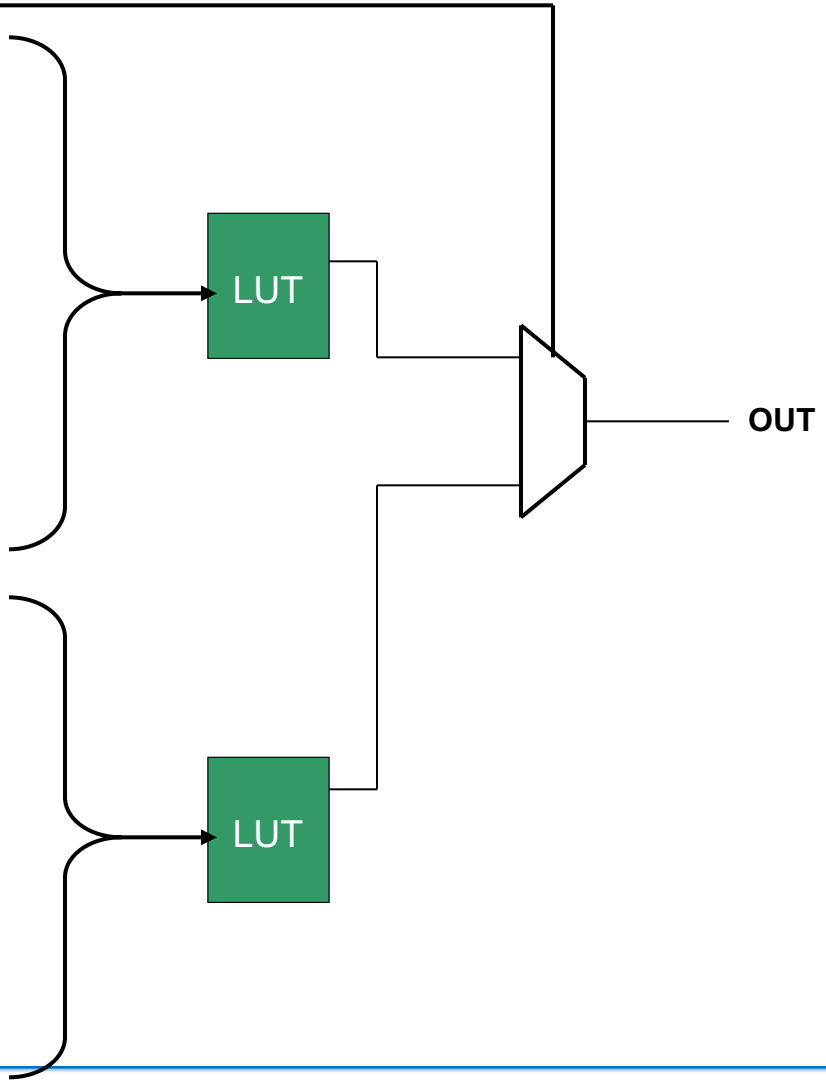
- Un slice CLB poate să implementeze și funcții cu mai mult de 4 intrări logice, folosind două LUT
- Funcția este partiționată între cele două LUT
- Multiplexorul final selectează ieșirea corectă



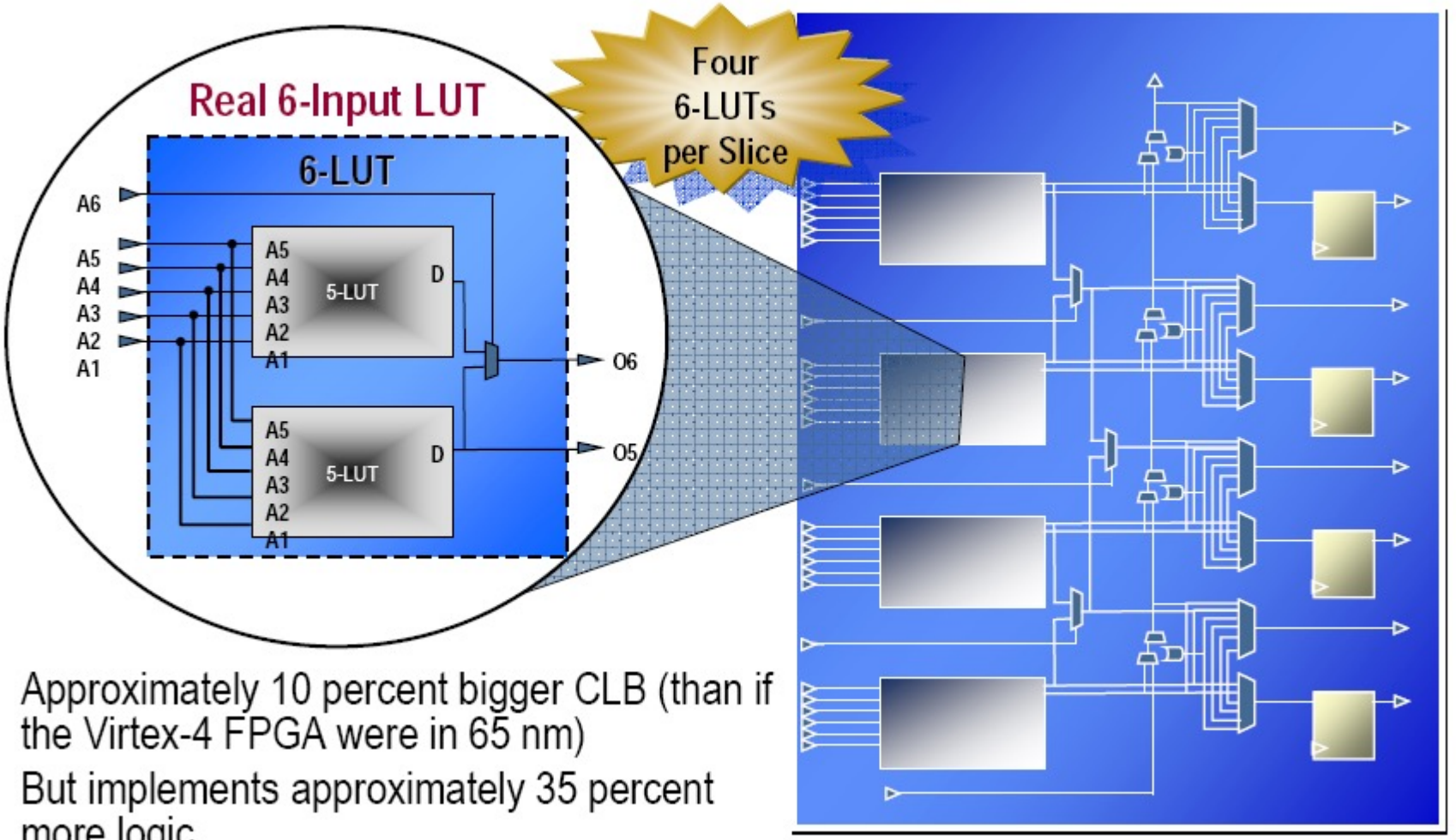
Implementarea unei funcții logice cu 5 intrări folosind 2 LUT

Embedded Systems Laboratory

X_5	X_4	X_3	X_2	X_1	Y
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0



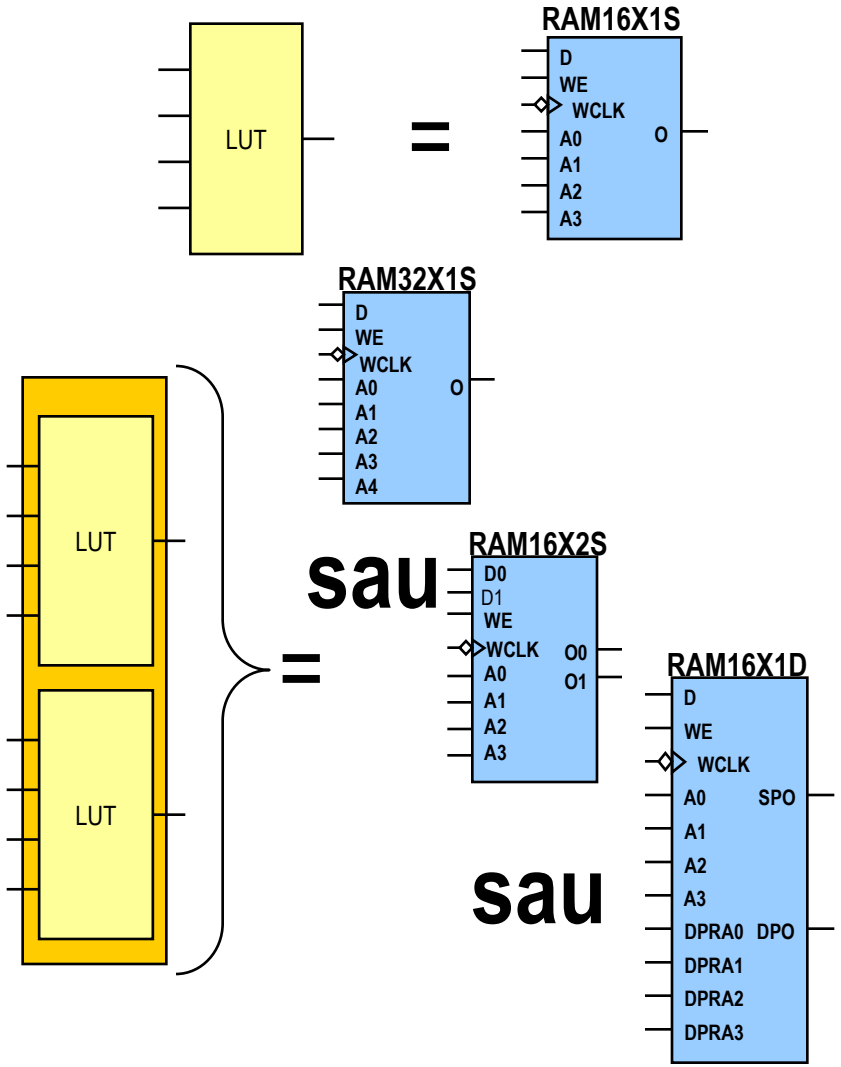
Exemplu Virtex



- Approximately 10 percent bigger CLB (than if the Virtex-4 FPGA were in 65 nm)
- But implements approximately 35 percent more logic

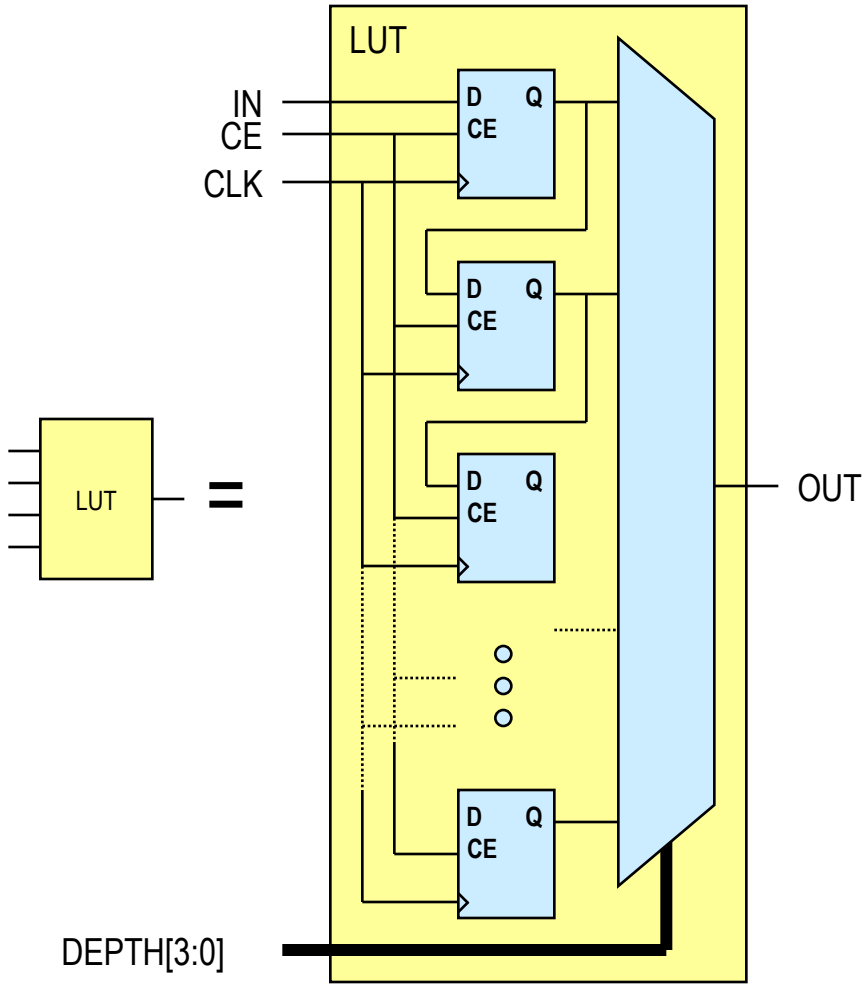
RAM Distribuit

- CLB LUT configurabil ca RAM
  - Un LUT egal 16x1 RAM
  - Memoria poate fi Single sau Dual-Port
  - Cascadarea LUT-urilor măreste lățimea memoriei
- Scriere Sincronă
- Citire Sincronă/Asincronă
  - Circuitele flip-flop pot fi folosite pentru implmentarea citirii sincrone

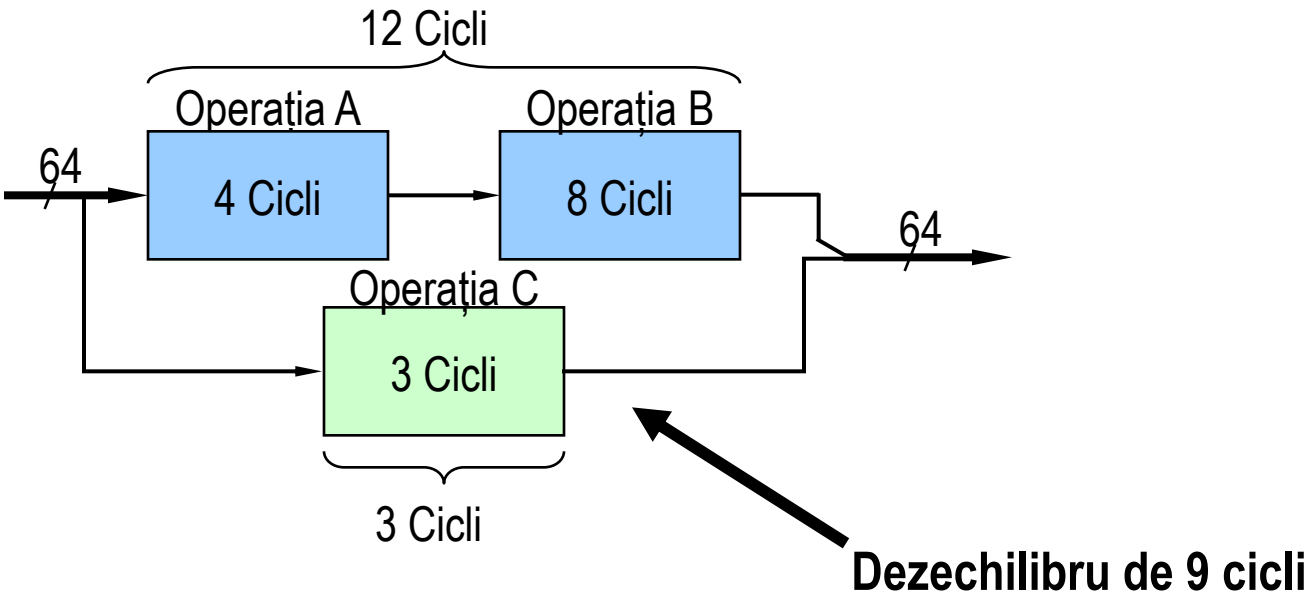


Registru Shift

- Fiecare LUT poate fi configurat ca un registru shift
  - Serial in, serial out
- Întârziere dinamică de până la 16 cicli
- Cascadarea mărește numărul ciclilor de întârziere



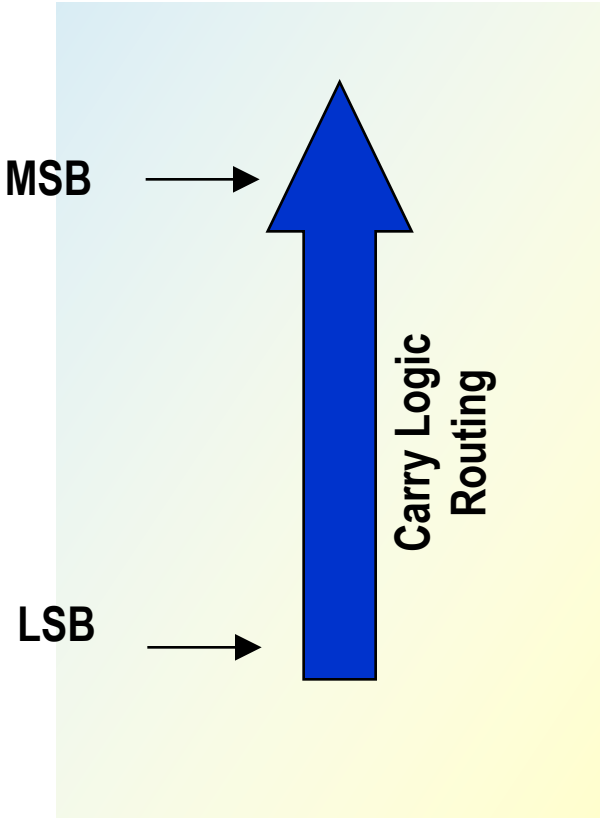
Registru Shift



- FPGA bogat în registre
  - Permite adăugarea de etaje pipeline pentru a mări productivitatea
- Căile de date trebuie să fie echilibrate pentru a păstra funcționalitatea sistemului

Fast Carry Logic

- Fiecare CLB conține logică separată pentru rutarea și generarea rapidă a semnalelor de sumă și carry
  - Mărește eficiența și performantele sumatoarelor , multiplicatoarelor, acumulatoarelor, comparatoarelor și numărătoarelor
- Logica de carry este independentă de logica normală și poate lucra în conjuncție cu LUT



Accesarea Carry Logic

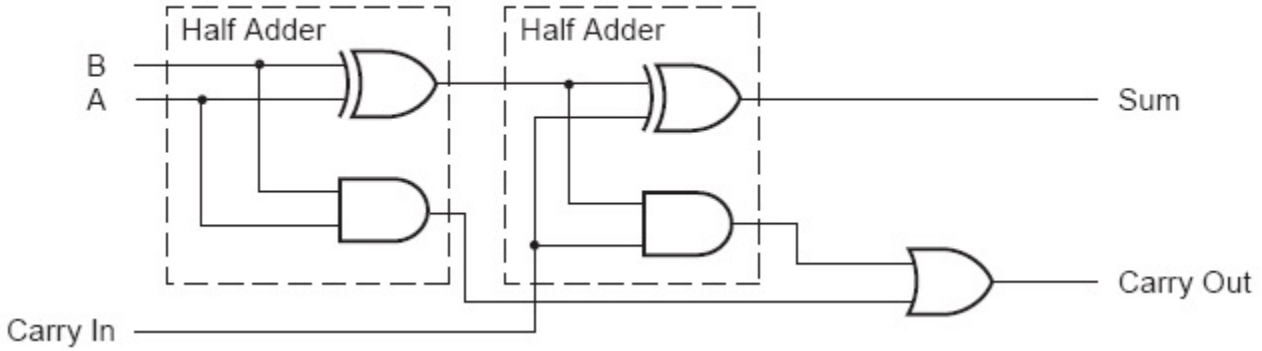
- Toate tool-urile majore de siteză pot invoca carry logic pentru funcțiile aritmetice
  - Adunare (SUM <= A + B)
  - Scădere (DIFF <= A - B)
  - Comparații (if A < B then…)
  - Numărătoare(count <= count +1)

Implementarea Carry Logic

Embedded Systems Laboratory

Abordarea clasică: Half Adder

A	B	Sum (A XOR B)	Carry Out (A AND B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Dezavantaje:

1. Propagare lenta a carry
2. Utilizeaza 2 LUT-uri

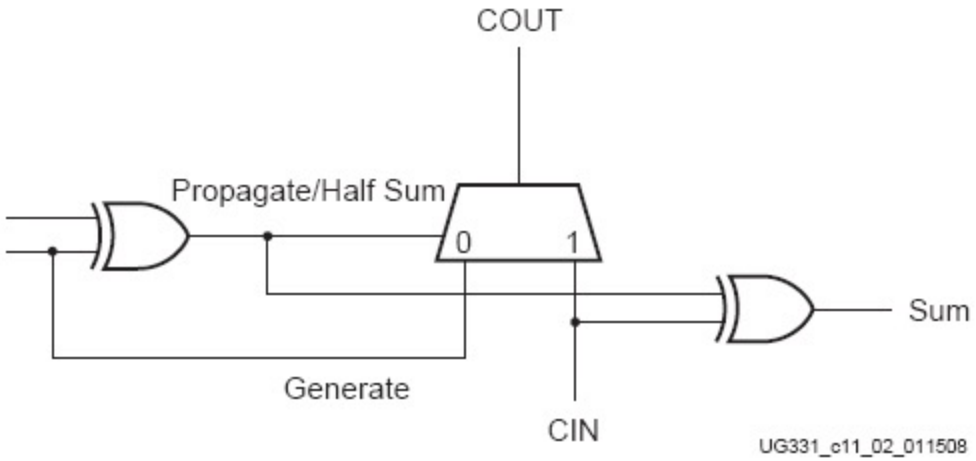
Implementarea Carry Logic

- Carry Look-Ahead

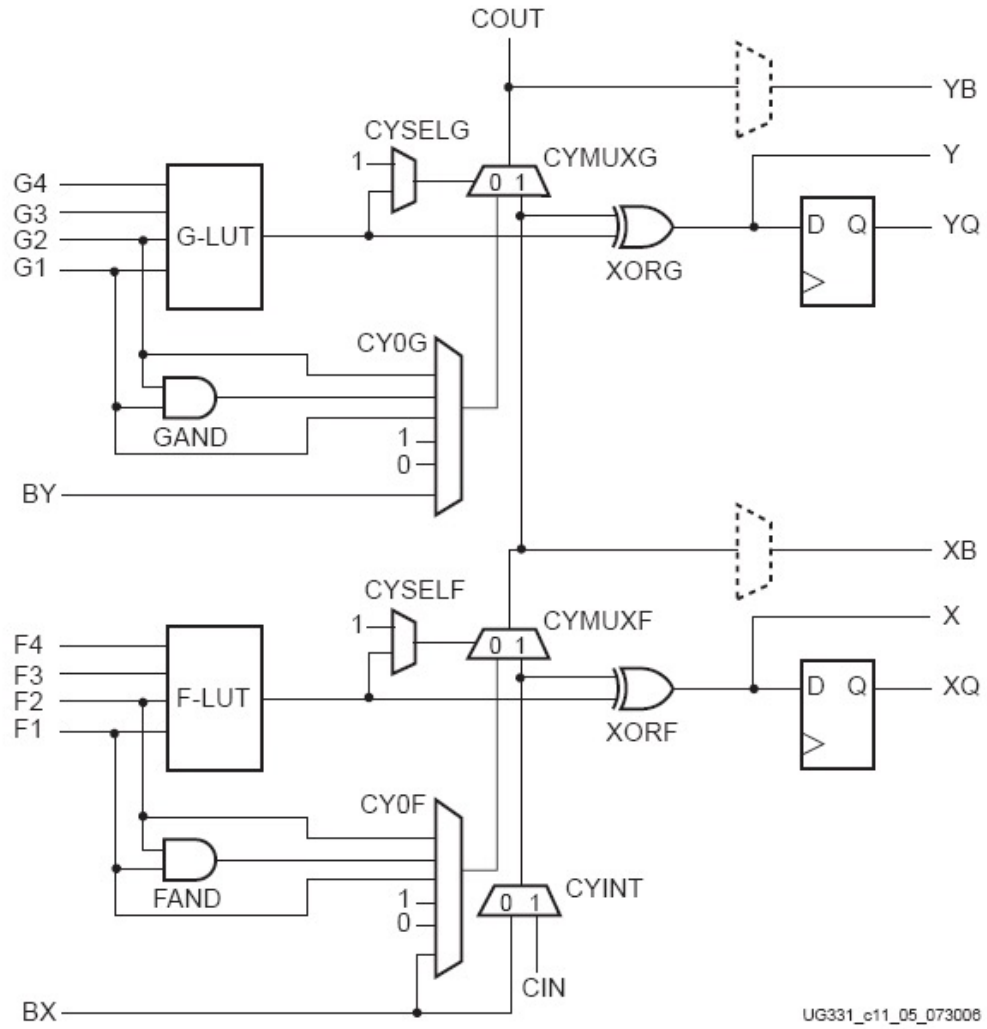
A	B	Propagate	Generate
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Avantaje:

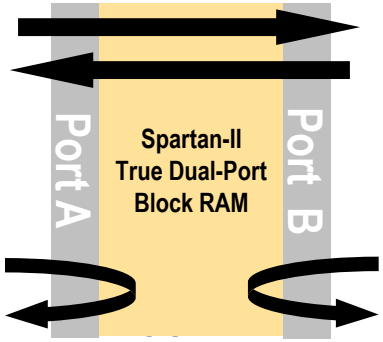
1. Folosește un singur LUT (trei intrări)
2. Carry se propagă rapid – un singur MUX



Structura unui slice cu logică adaugată

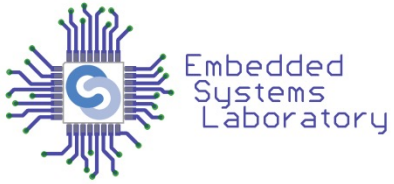


Block RAM



- Implementare eficientă a memoriei între CLB și porturile IO
  - Blocuri dedicate
- Toată logica de control este implementată în celula de RAM
  - De la 4 la 104 blocuri de memorie
    - 18 kbiți pe bloc
  - Blocurile se pot cascada pentru implementarea unei memorii mai mari
- Poate funcționa ca single sau dual-port RAM

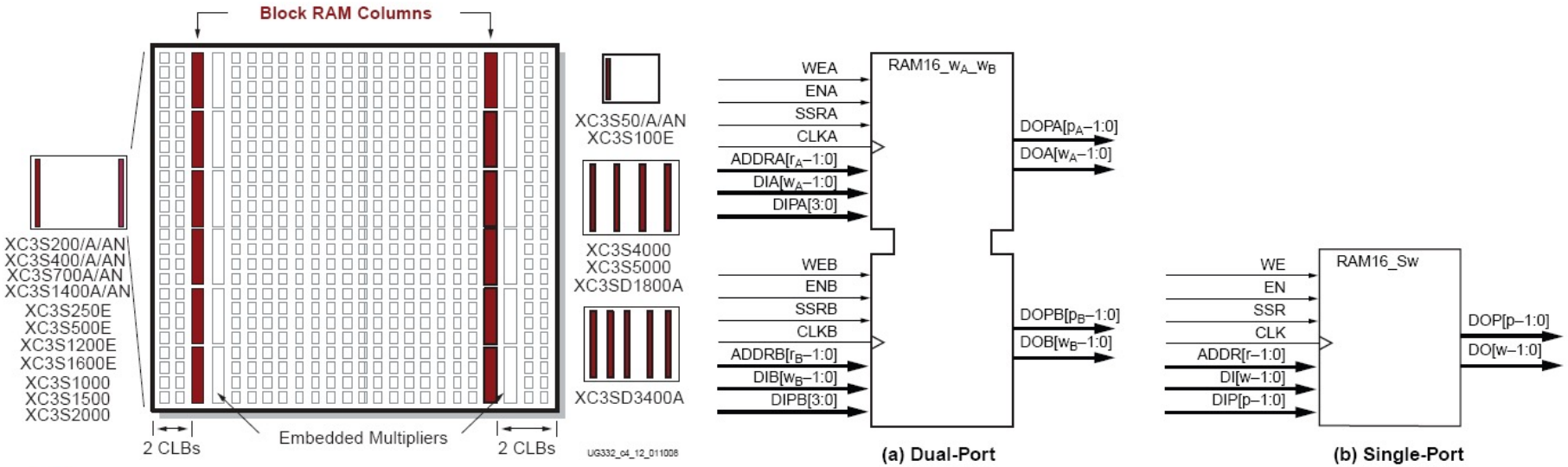
Cantitatea de RAM familia Spartan 3



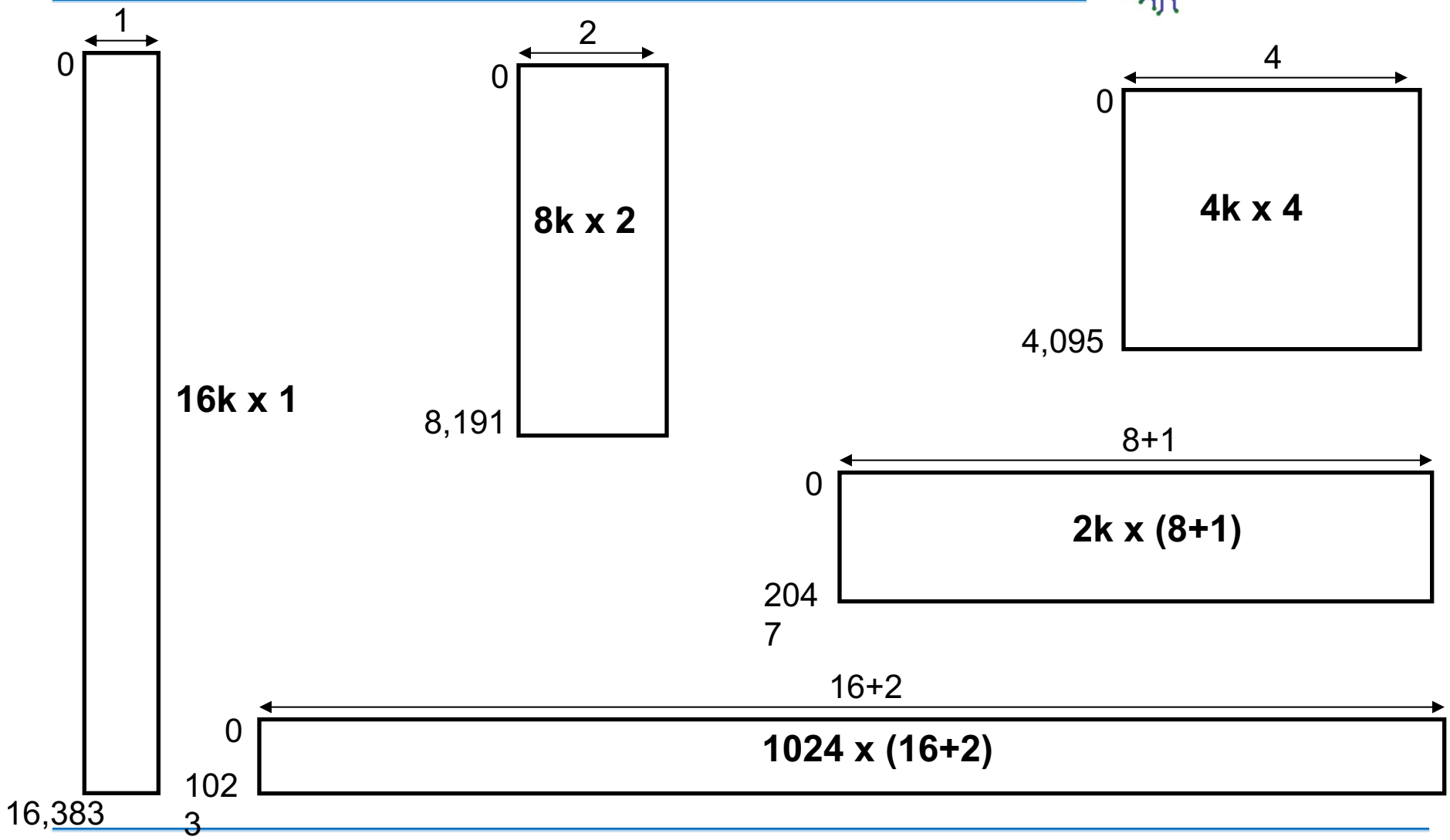
Device	Total Number of RAM Blocks	Total Addressable Locations (bits)	Number of Columns
XC3S50	4	73,728	1
XC3S200	12	221,184	2
XC3S400	16	294,912	2
XC3S1000	24	442,368	2
XC3S1500	32	589,824	2
XC3S2000	40	737,280	2
XC3S4000	96	1,769,472	4
XC3S5000	104	1,916,928	4

Block RAM Port Aspect Ratios

Total RAM bits, including parity	18,432 (16K data + 2K parity)
Memory Organizations	16Kx1 8Kx2 4Kx4 2Kx8 (no parity) 2Kx9 (x8 + parity) 1Kx16 (no parity) 1Kx18 (x16 + 2 parity) 512x32 (no parity) 512x36 (x32 + 4 parity) 256x72 (single-port only)

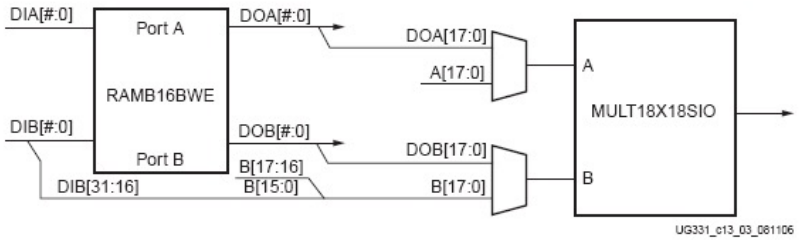


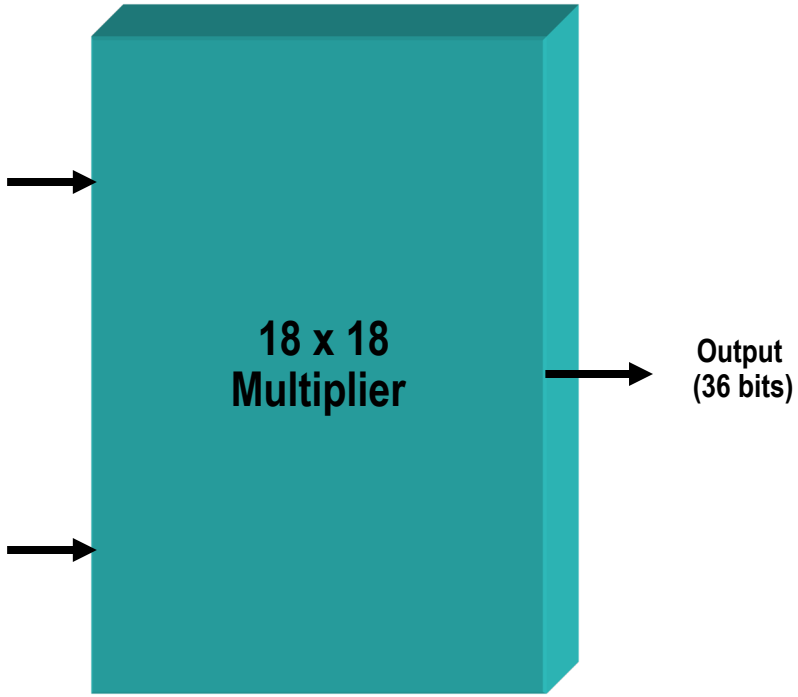
Block RAM Port Aspect Ratios



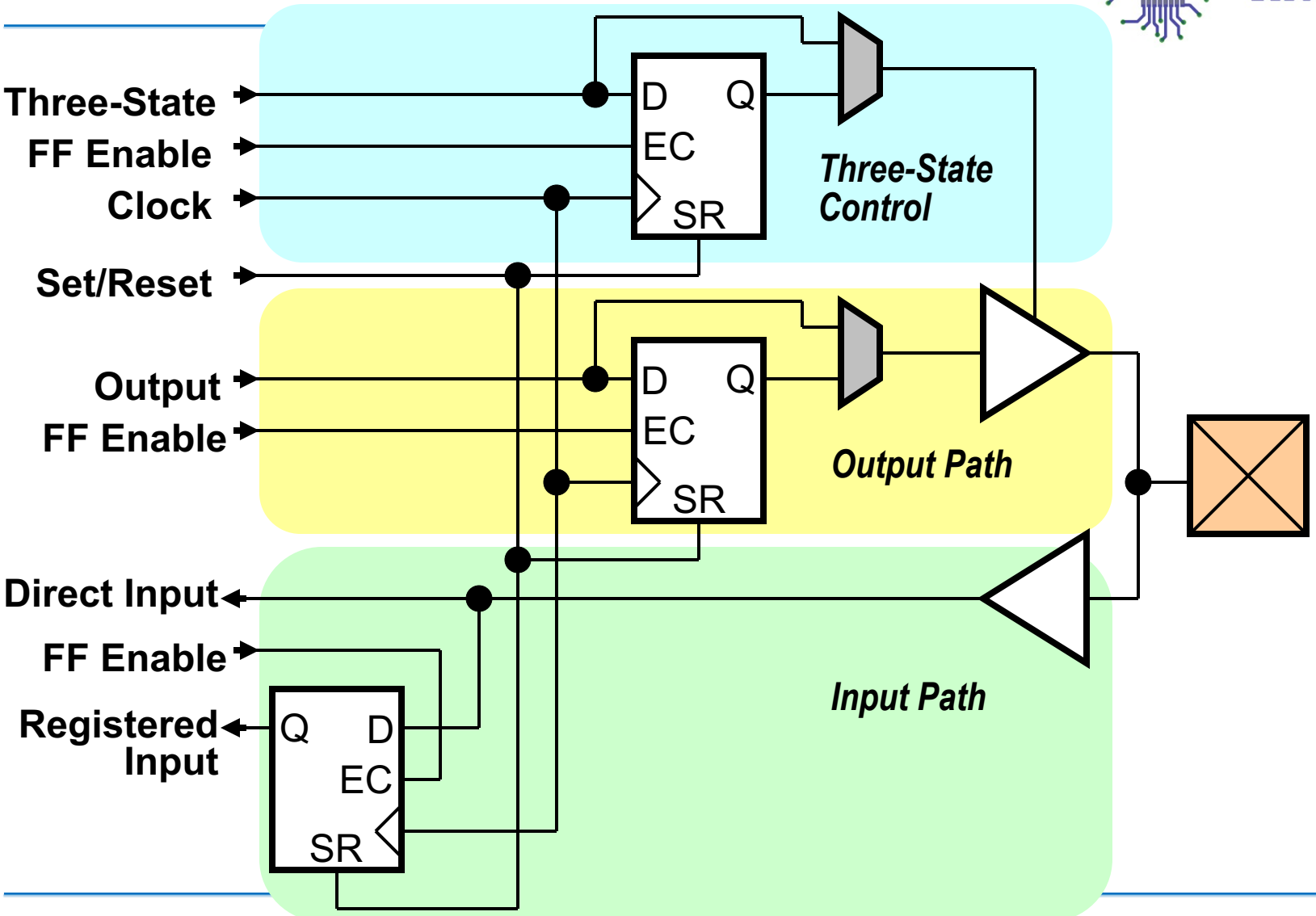
18 x 18 Embedded Multiplier

- Întărește funcționalitatea de DSP a circuitului
  - Optimizat pentru viteză și performanță maximă. Implementează module tip multiply/accumulate
  - Sunt organizate în coloane adiacente coloanelor de blocuri RAM
  - Fiecare multiplicator are doi operanzi de 18 biți lățime și este implementat în logică pur combinațională. Se conectează prin magistrală la blocul RAM adiacent





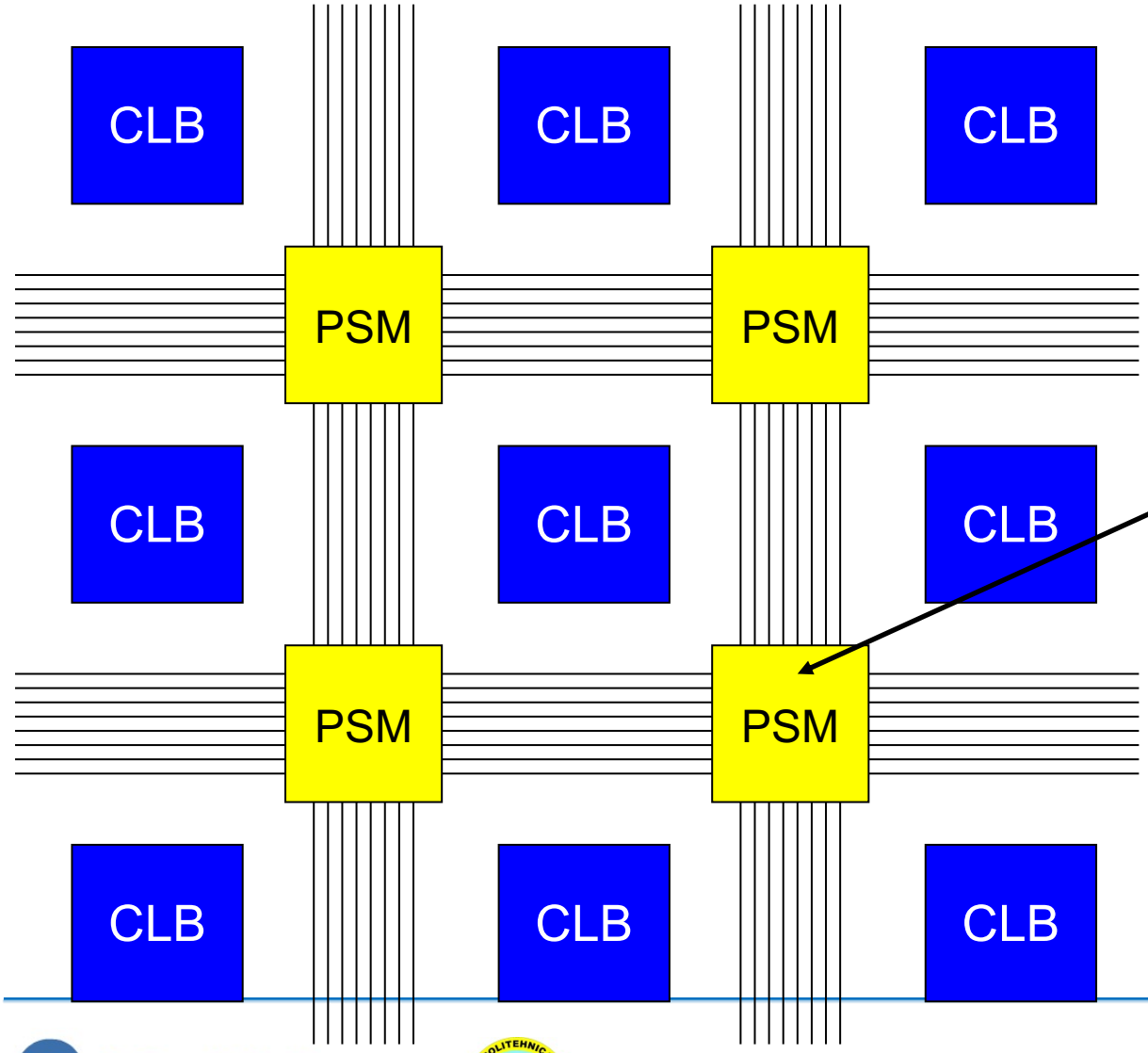
Structura de bază a unui bloc I/O



Funcționalitatea unui bloc I/O

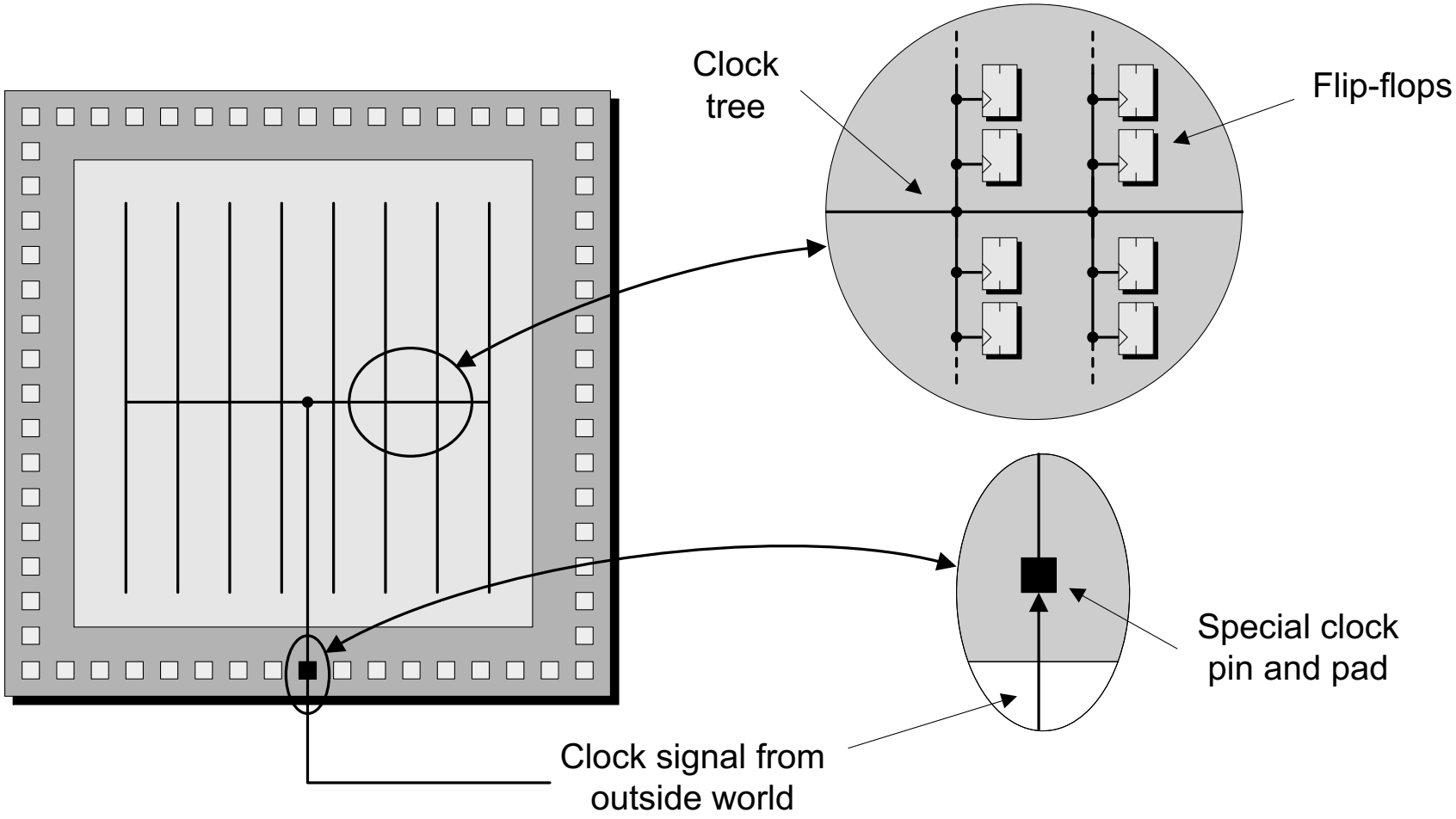
- Blocurile I/O (IOB) permit interconectarea pinilor circuitului la structura internă de blocuri CLB
- Fiecare IOB poate să funcționeze ca un port uni sau bidirecțional
- Ieșirile pot fi forțate în starea de impedanță mărită
- Intrările și ieșirile pot fi trecute printr-un buffer de tip registru
- Intrările pot fi amânate

Resurse de rutare



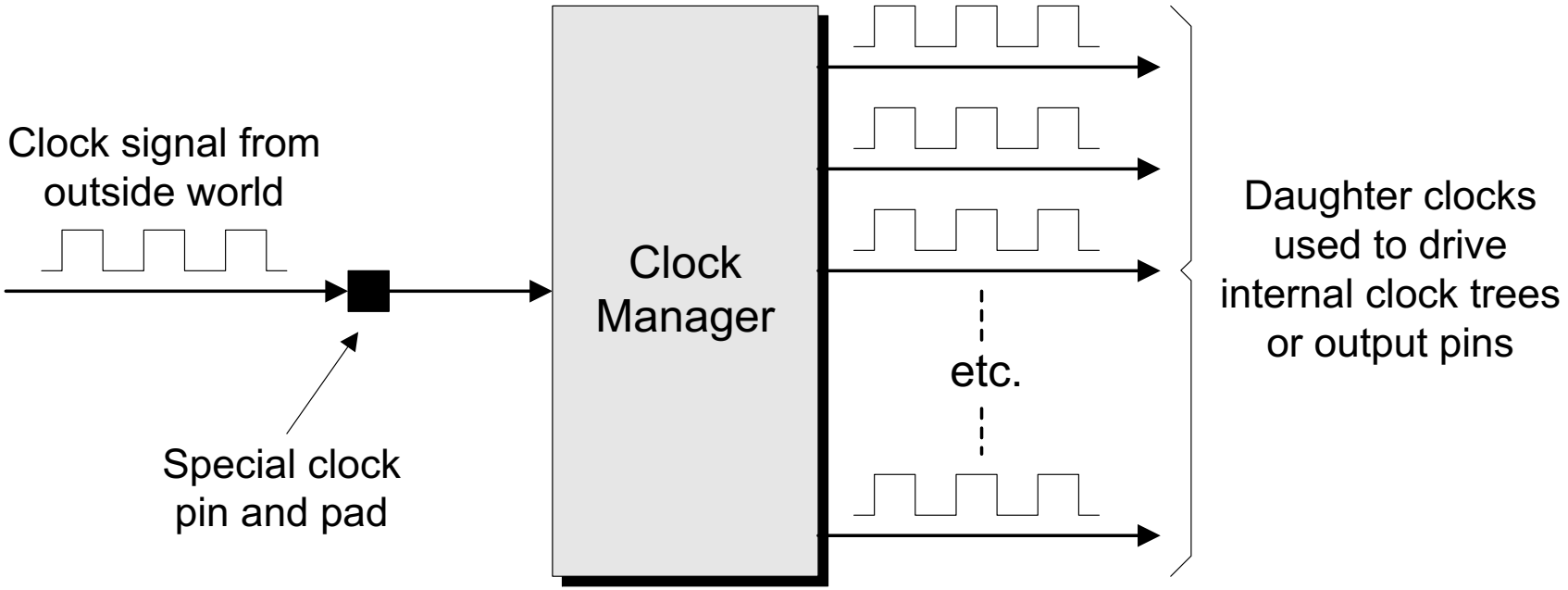
Programmable
Switch
Matrix

Clock Tree



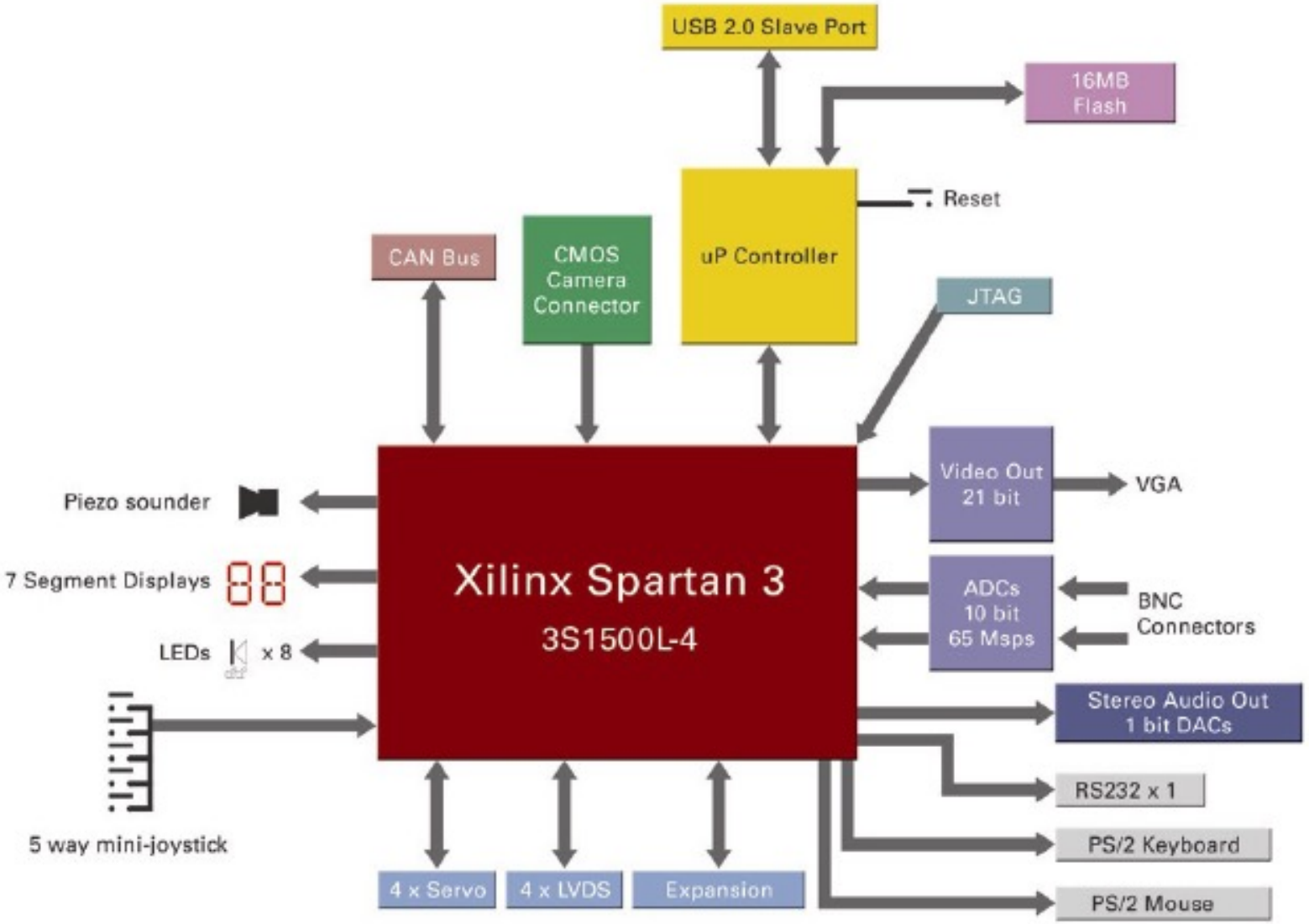
The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Digital Clock Manager (DCM)



The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

Exemplu de sistem cu FPGA



FPGA Design Flow

Design flow (1)

Embedded Systems Laboratory

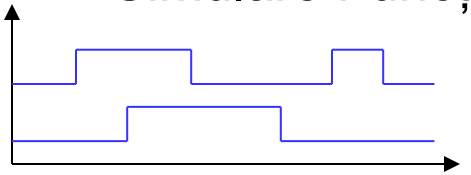
Design and implement a simple unit permitting to speed up encryption with RC5-similar cipher with fixed key set on 8031 microcontroller. Unlike in the experiment 5, this time your unit has to be able to perform an encryption algorithm by itself, executing 32 rounds…..

Specificații (descrierea functionalității)

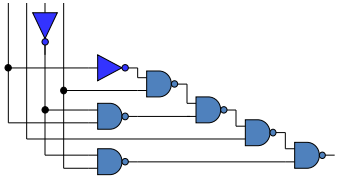
```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RC5_core is
        port(
            clock, reset, encr_decr: in std_logic;
            data_input: in std_logic_vector(31 downto 0);
            data_output: out std_logic_vector(31 downto 0);
            out_full: in std_logic;
            key_input: in std_logic_vector(31 downto 0);
            key_read: out std_logic;
        );
end AES_core;
```

Descriere VHDL (Fișiere sursă)

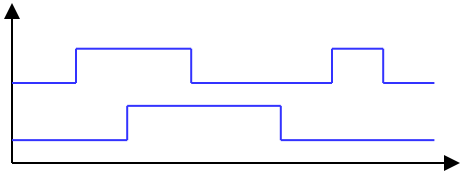
Simulare Funcțională



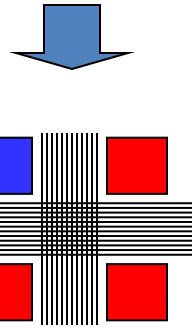
Sinteză



Simulare Post-synthesis

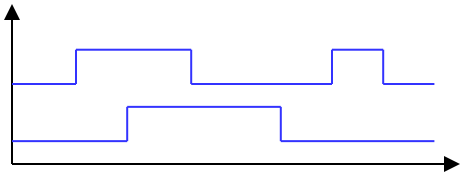


Design flow (2)



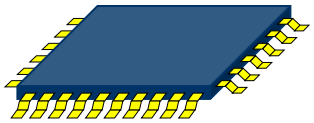
Implementare

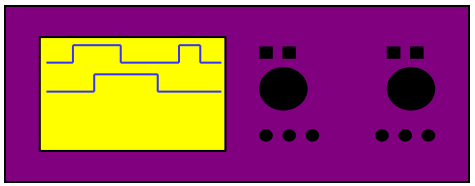
Simularea timpilor



Configurare

Testare On chip





Sinteza

Tool-uri de Sinteză

Synplify Pro

Xilinx XST

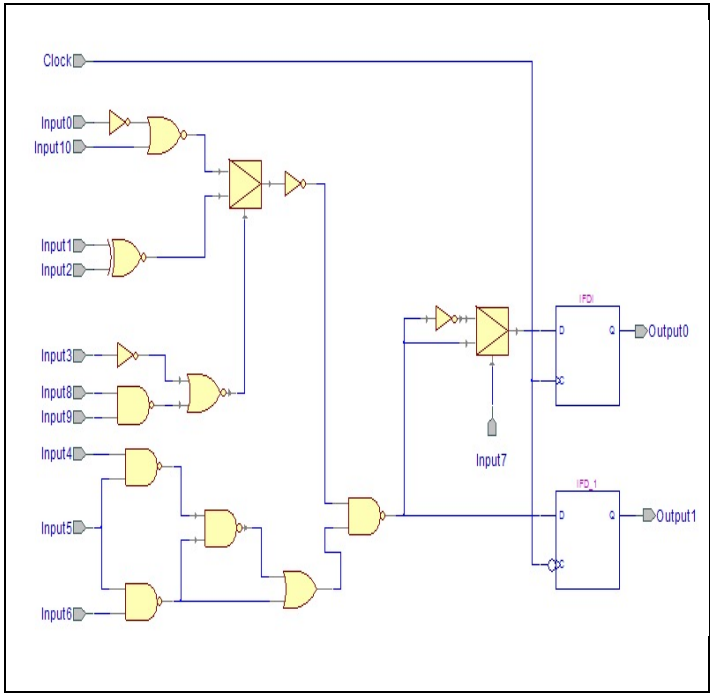
… și altele

Sinteza Logică

Descriere VHDL

```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
                A1<=A when (NEG_A='0') else
                                not A;
                B1<=B when (NEG_B='0') else
                                not B;
                Y<=Y1 when (NEG_Y='0') else
                                not Y1;

                MUX_0<=A1 and B1;
                MUX_1<=A1 or B1;
                MUX_2<=A1 xor B1;
                MUX_3<=A1 xnor B1;

                with (L1 & L0) select
                                Y1<=MUX_0 when "00",
                                                MUX_1 when "01",
                                                MUX_2 when "10",
                                                MUX_3 when others;

end MLU_DATAFLOW;
```

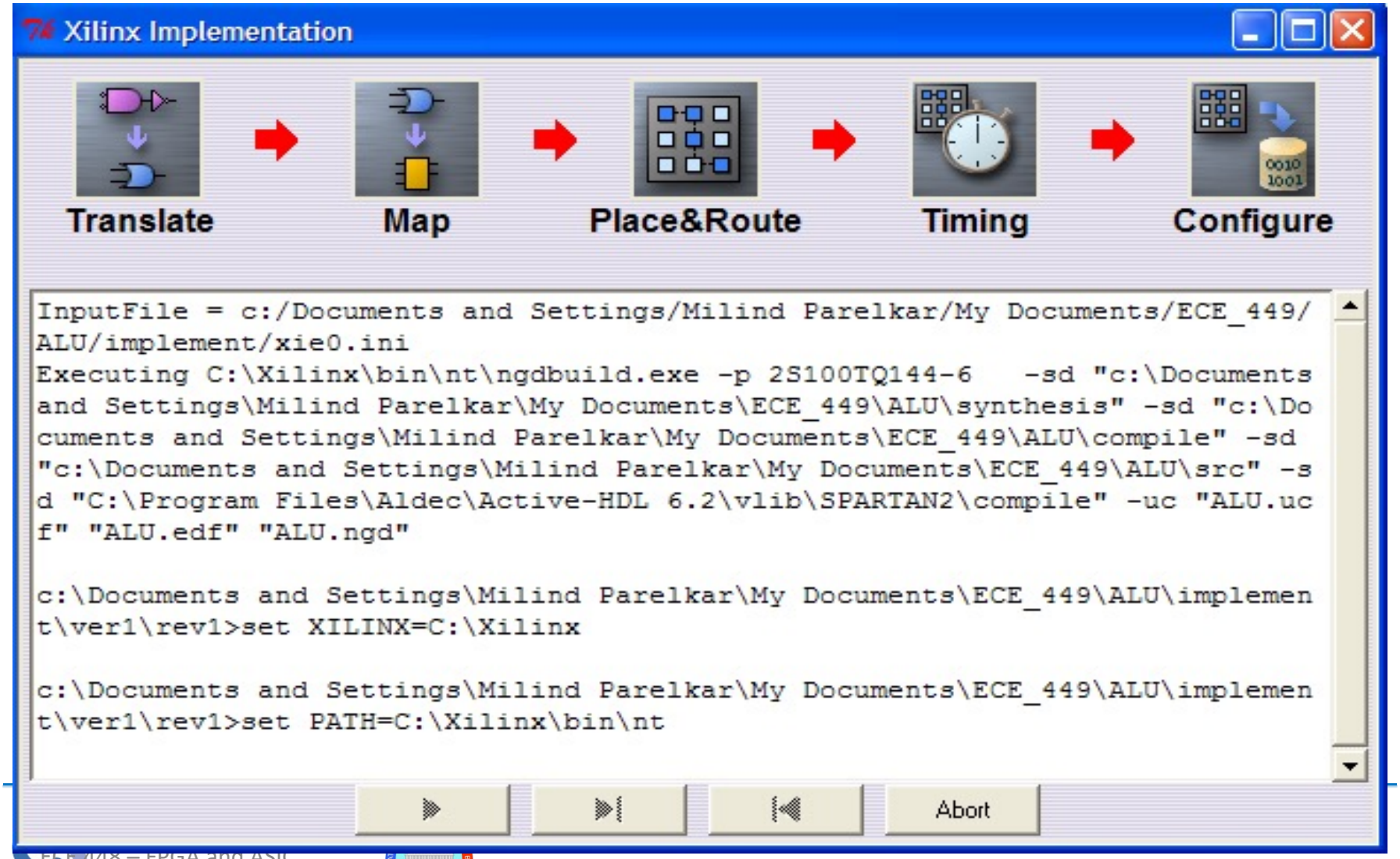
Netlist Circuit



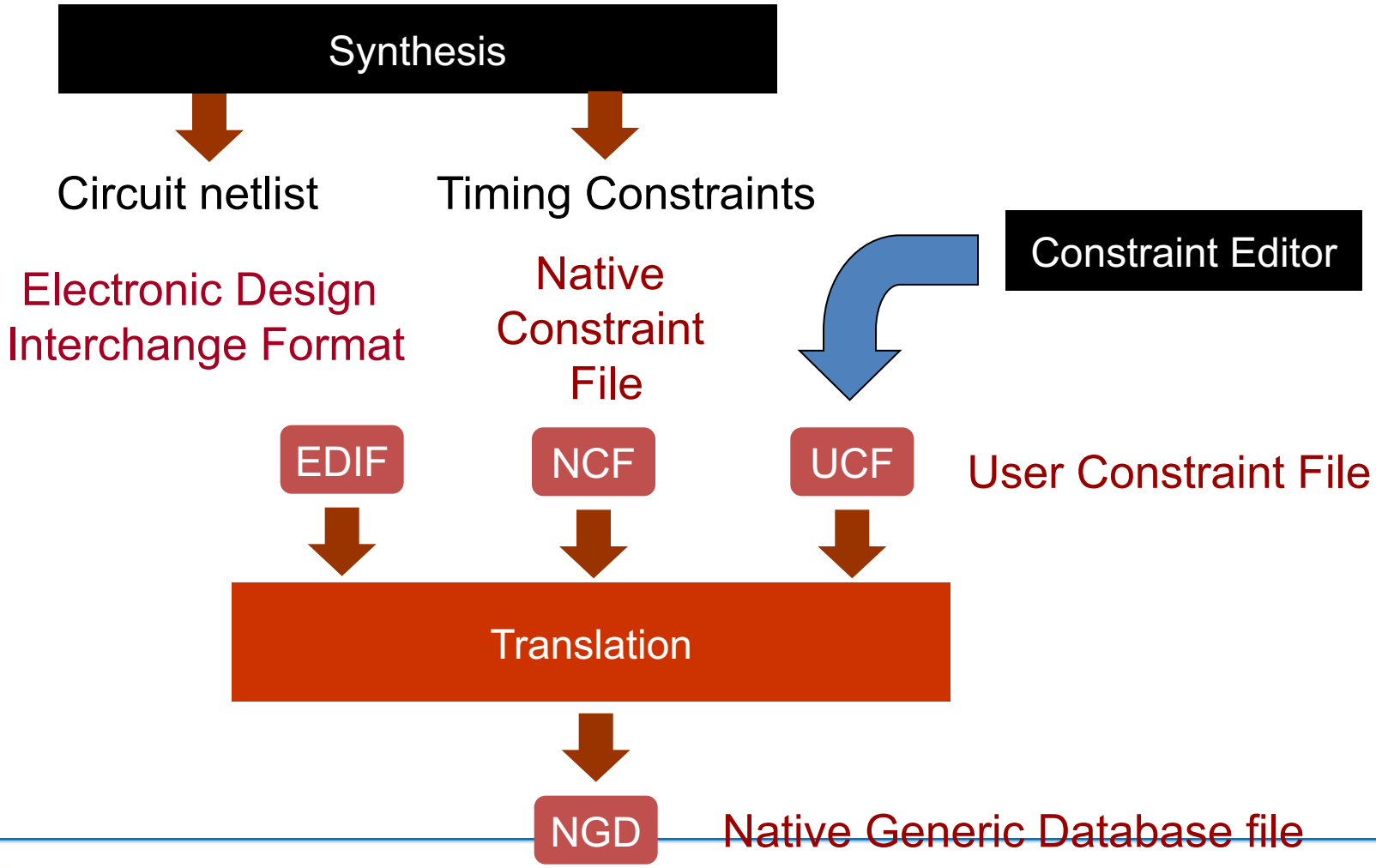
Implementare

Implementare

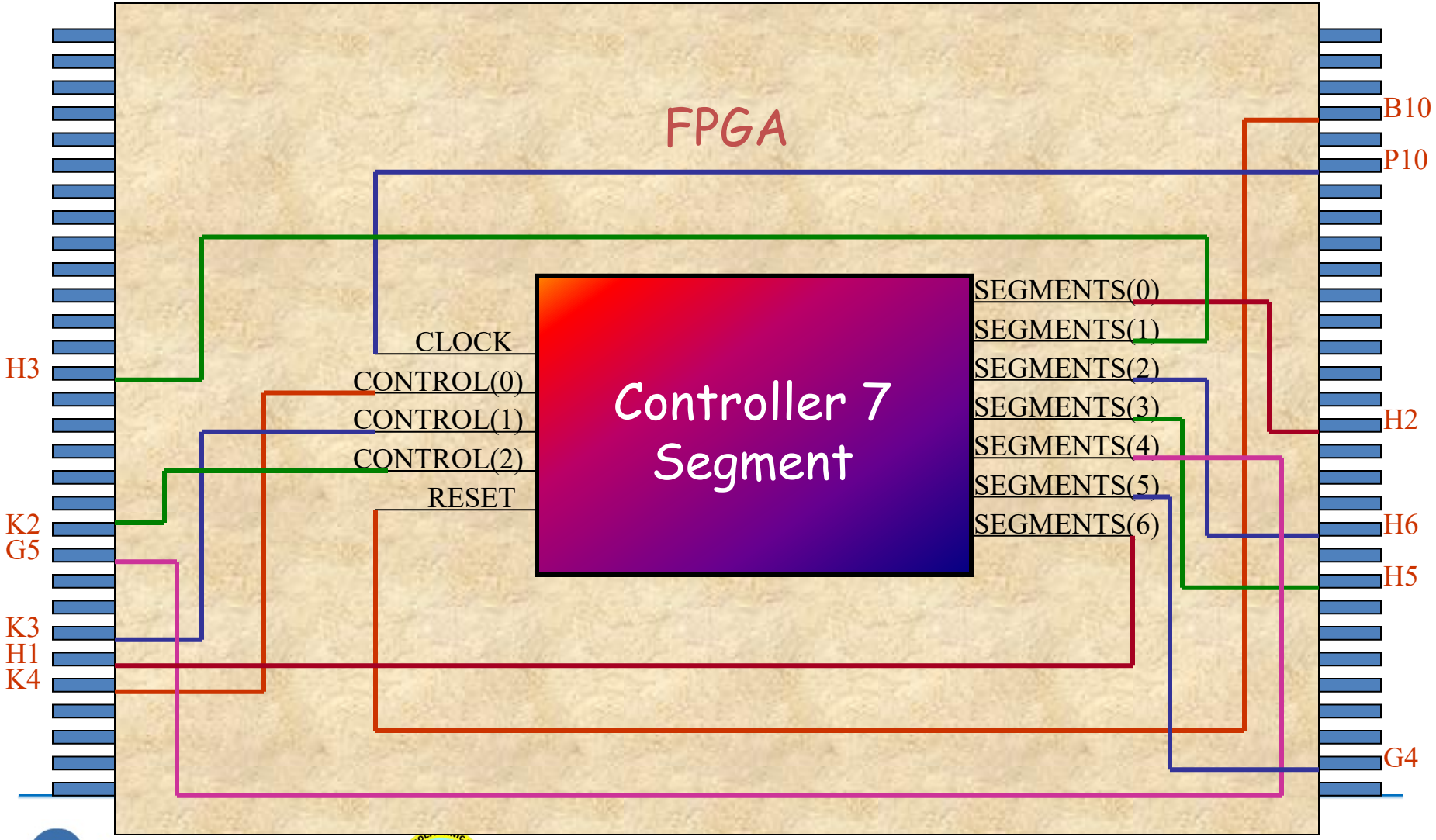
- Dupa siteza, întregul proces de implementare este realizat de tool-uri FPGA proprietare


Xilinx Implementation
Translate
Map
Place&Route
Timing
Configure
InputFile = c:/Documents and Settings/Milind Parelkar/My Documents/ECE_449/ALU/implement/xie0.ini
Executing C:\Xilinx\bin\nt\ngdbuild.exe -p 2S100TQ144-6 -sd "c:\Documents and Settings\Milind Parelkar\My Documents\ECE_449\ALU\synthesis" -sd "c:\Documents and Settings\Milind Parelkar\My Documents\ECE_449\ALU\compile" -sd "c:\Documents and Settings\Milind Parelkar\My Documents\ECE_449\ALU\src" -sd "C:\Program Files\Aldec\Active-HDL 6.2\vlib\SPARTAN2\compile" -uc "ALU.ucf" "ALU.edf" "ALU.ngd"
c:\Documents and Settings\Milind Parelkar\My Documents\ECE_449\ALU\implement\ver1\rev1>set XILINX=C:\Xilinx
c:\Documents and Settings\Milind Parelkar\My Documents\ECE_449\ALU\implement\ver1\rev1>set PATH=C:\Xilinx\bin\nt
Abort

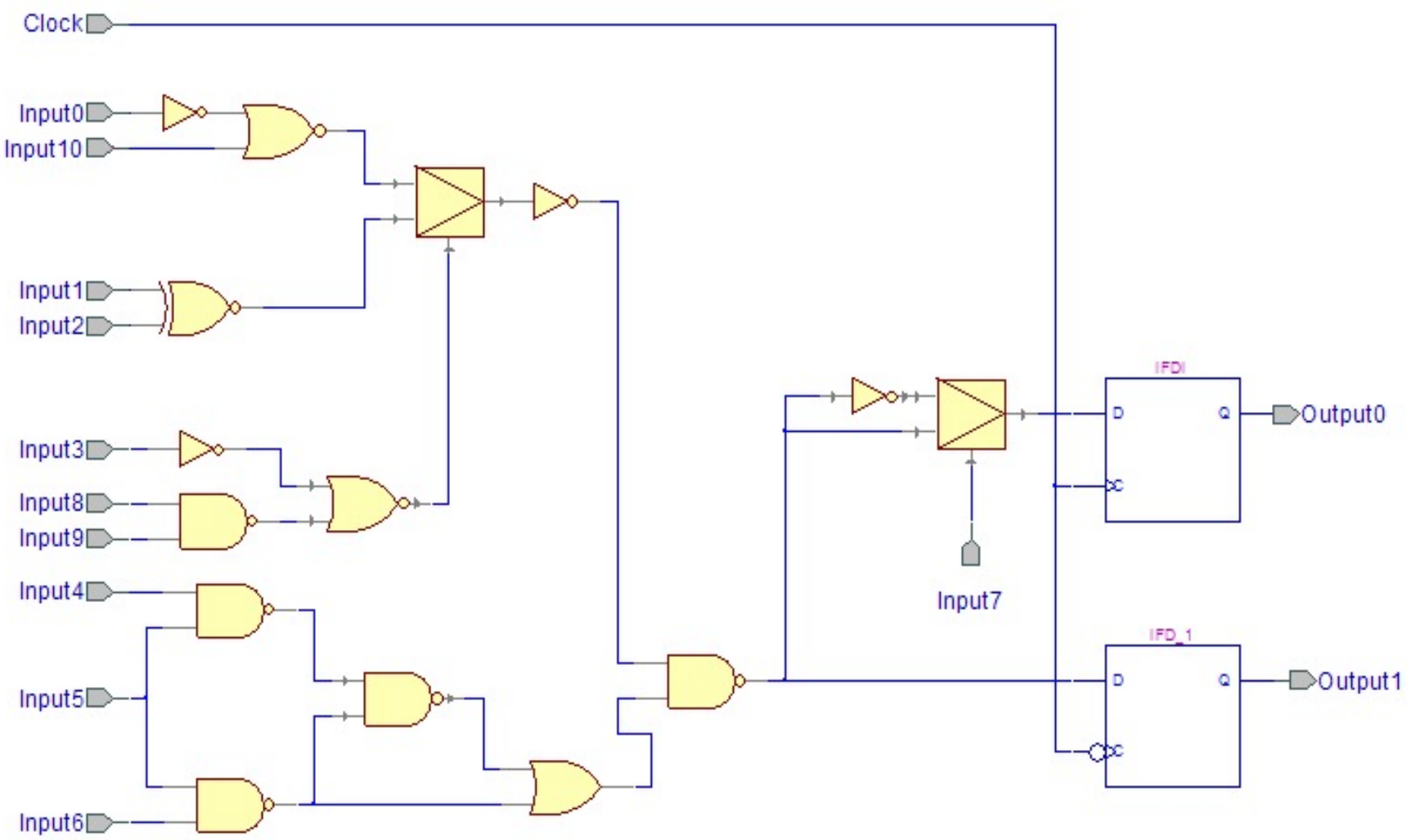
Translatie


Synthesis
Circuit netlist
Timing Constraints
Electronic Design Interchange Format
Native Constraint File
Constraint Editor
EDIF
NCF
UCF
User Constraint File
Translation
NGD
Native Generic Database file

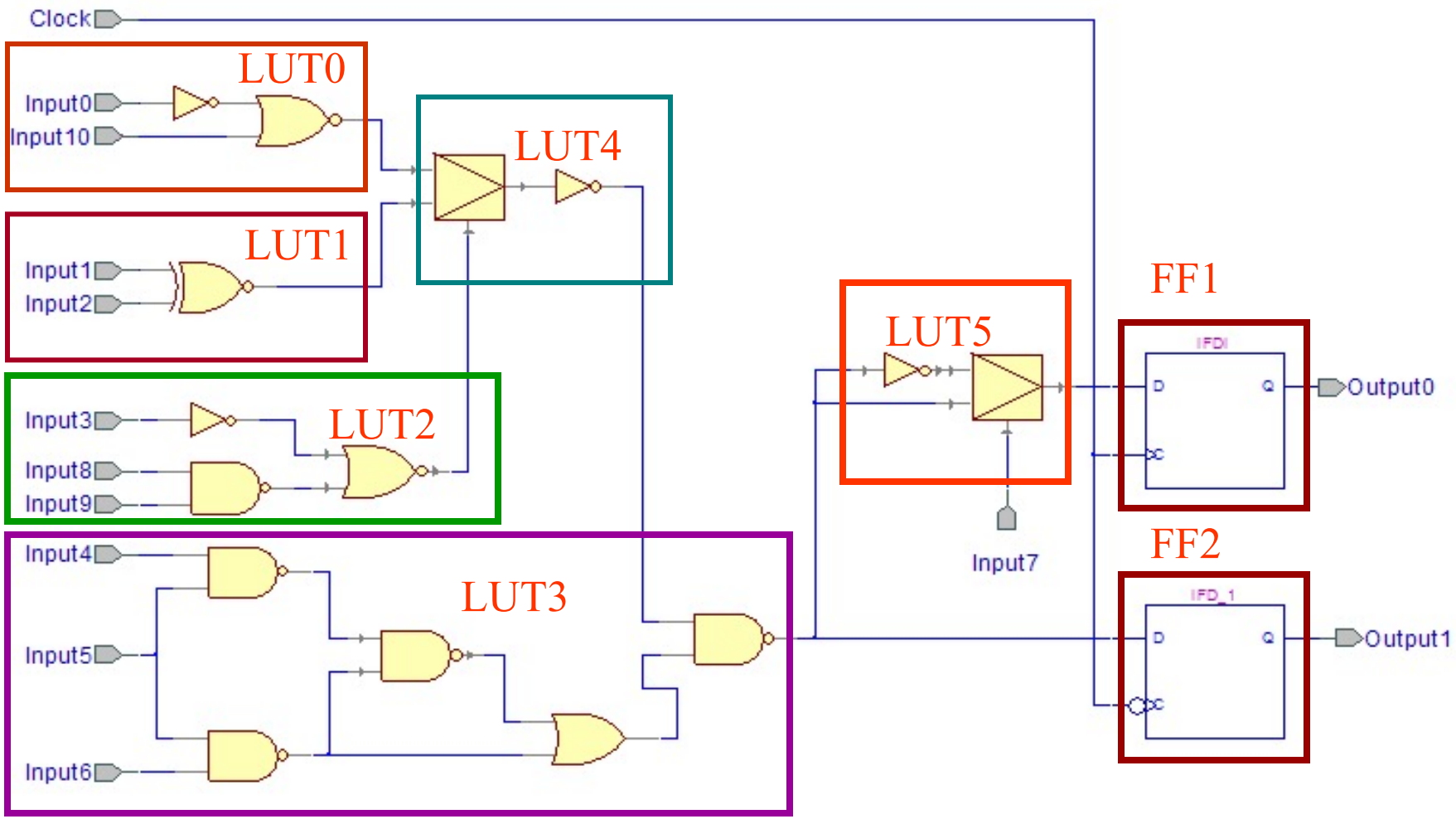
Asocierea Pinilor



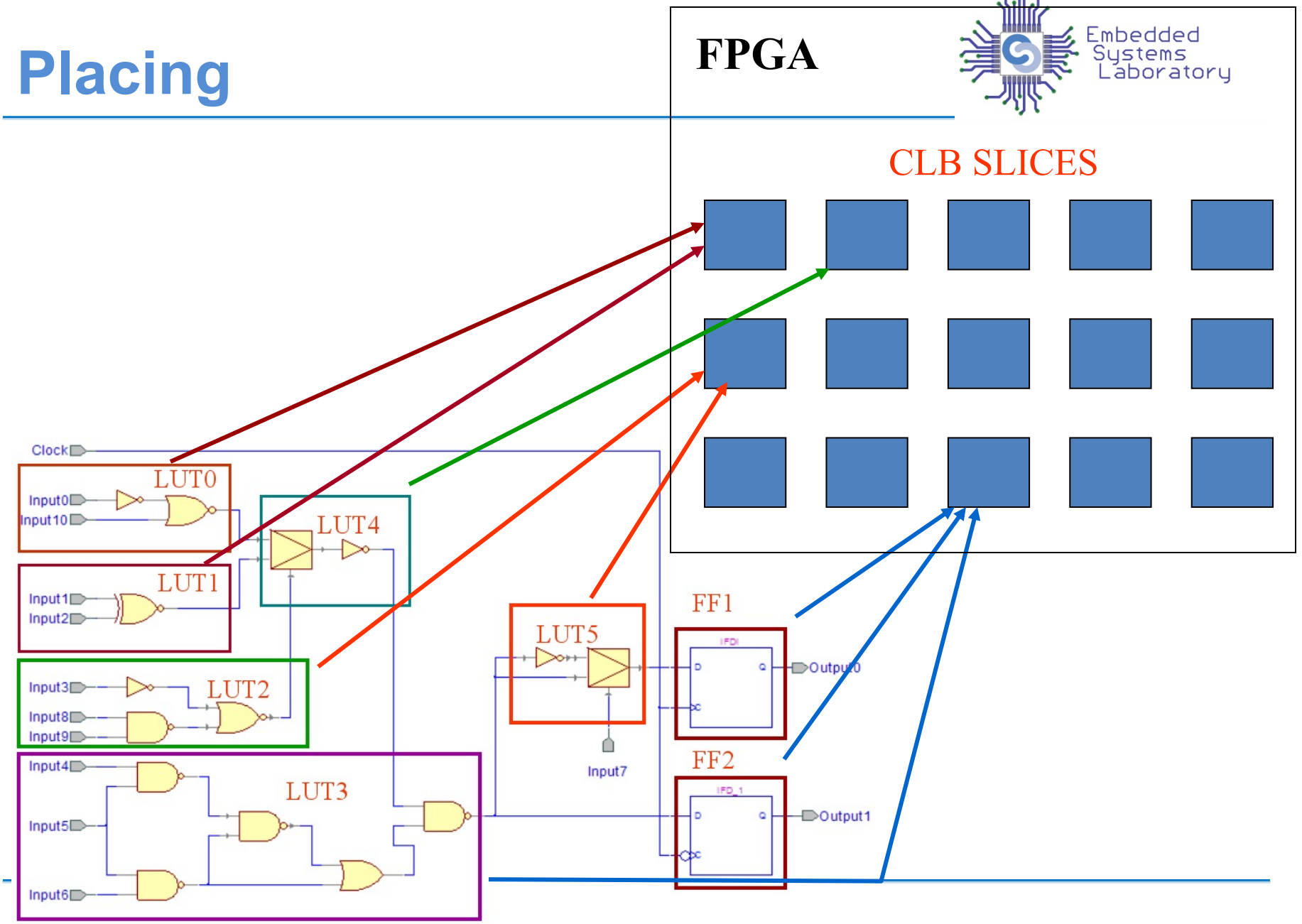
Netlist Circuit



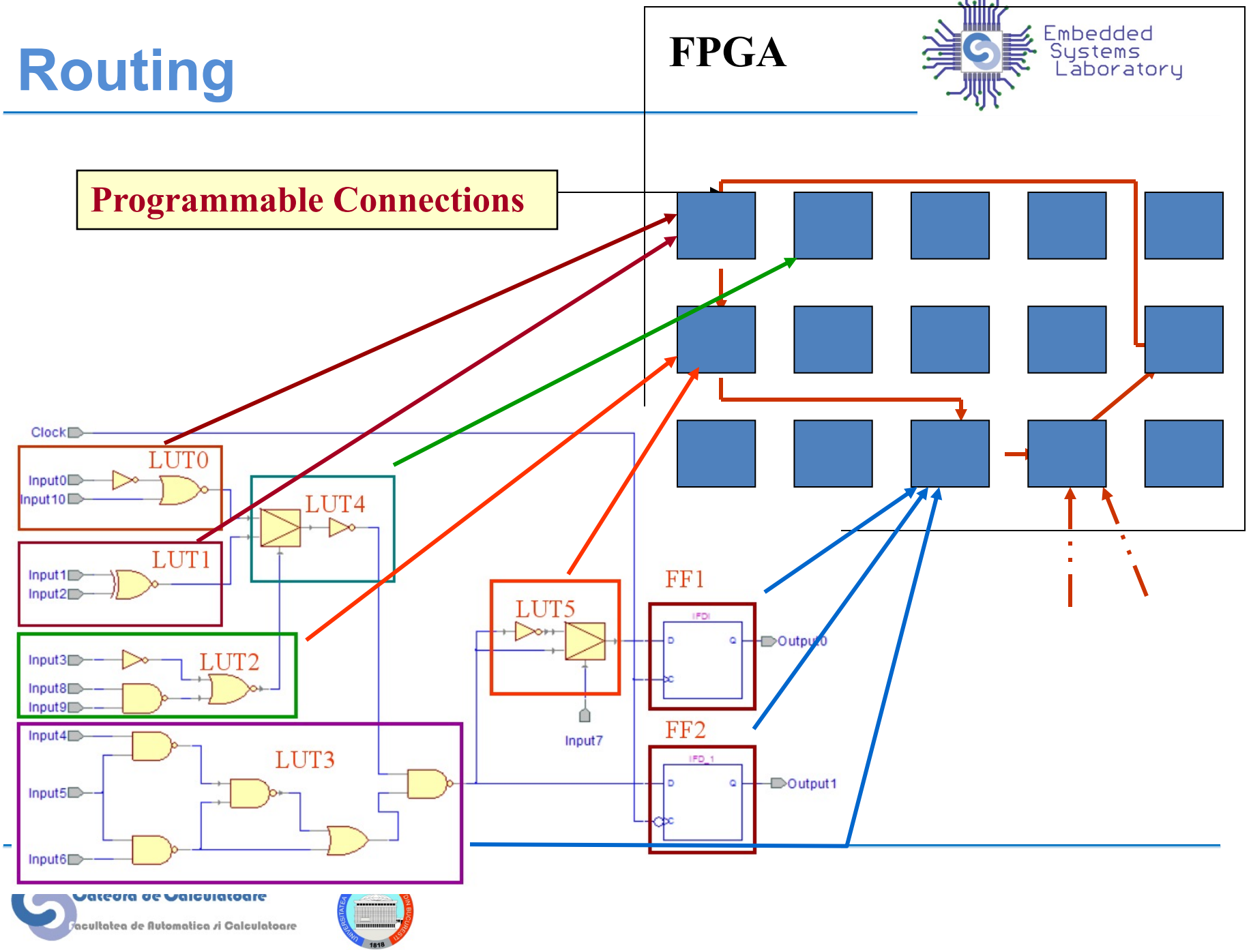
Mapare


Clock
LUT0
Input0
Input10
LUT4
LUT1
Input1
Input2
LUT5
FF1
IFDI
D
Q
Output0
Input3
LUT2
Input8
Input9
Input7
FF2
IFD_1
Input4
LUT3
Input5
Input6
Output1

Placing


FPGA
Embedded Systems Laboratory
CLB SLICES
Clock
LUT0
Input0
Input10
LUT1
Input1
Input2
LUT4
Input3
Input8
Input9
LUT2
Input4
Input5
Input6
LUT3
LUT5
Input7
FF1
FF2
Output0
Output1

Routing


FPGA
Embedded Systems Laboratory
Programmable Connections
Clock
LUT0
Input0
Input10
LUT1
Input1
Input2
LUT4
Input3
Input8
Input9
LUT2
Input4
Input5
Input6
LUT3
LUT5
Input7
FF1
FF2
Output0
Output1

Configurare

- Odată ce design-ul este implementat, trebuie creat un fișier pe care FPGA-ul poate să-l înțeleagă.
  - Acest fișier este numit bit stream: BIT file (extensia .bit)

- Fișierul BIT poate fi descărcat direct în FPGA sau poate fi transformat într-un fișier PROM care stochează informațiile despre programare.