Introducere. FPGA. VHDL
Generalitati si caracteristici

S.l. Ing. Vlad-Cristian Miclea

Universitatea Tehnica din Cluj-Napoca

Departamentul Calculatoare

CUPRINS

1) Detalii curs si continut
2) Circuite Hardware
   - Recapitulare componente hardware
   - Circuite integrate
   - ASIC: CPU vs microcontrollere vs GPU
3) FPGA
   - Generalitati
   - Structura FPGA
   - Etapele proiectarii folosind FPGA
4) Introducere – VHDL
   - Obiective VHDL
   - Specificare si simulare
   - Descrierea componentelor
   - Structura unui program
5) Concluzii

INFORMATII CURS

Info generale

- Vlad.Miclea@cs.utcluj.ro;
- users.utcluj.ro/~vmiclea -> Teaching -> PSN/DSD
- Cursurile – Aula Instalatii
- Continuare ASDN/PL – Hardware 2.0 (complexitate++)
- Intrebari ORICAND!!!
- Probleme – discutate live + Vivado (incarcate pe site)

Evaluare

- Examen final: 60 puncte
  - Examen scris
  - Fara examen partial
- Testare laborator: 40 puncte
  - 20 puncte – test VHDL
  - 20 puncte – mini-proiect

PLAN CURS – posibil sa apara schimbari

- Partea 1 – FPGA si VHDL
  1. **FPGA**
  2. Limbajul VHDL – 1
  3. Limbajul VHDL – 2
- Partea 2 – Implementarea sistemelor numerice
  4. Microprogramare
  5. Unitate de comanda
  6. Unitate de executie
- Partea 3 – Automate
  7. Automate finite
  8. Stari
  9. Automate sincrone
  10. Automate asincrone
  11. Identificarea automatelor
  12. Automate fara pierderi
  13. Automate liniare
- Partea 4 – Probleme si discutii

INFORMATII LABORATOR

Info generale

- Prezenta obligatorie!
- Sala 204/211 - Obs
- Mici conspecte, cititi laboratorul inainte – important!!
  - Snippet-uri de cod VHDL
  - Mici diagrame

Evaluare (40 puncte)

- Partea 1 – VHDL
  - 6 laboratoare – se invata sa se scrie cod de descriere hardware
  - Se incheie cu un test laborator (20 puncte)
- Partea 2 – Mini-proiect
  - 5 laboratoare – un circuit mai complex, implementat in VHDL
  - Se incheie cu o prezentare (20 puncte)

Tipuri de proiecte

- Proiecte de 20p: codul ar trebui sintetizat + implementat pe FPGA
- Proiecte de 16p: doar scris cod in VHDL + simulare!

PLAN LABORATOR - deocamdata

Lucrari laborator

1. Introducere in VHDL
2. Unitati fundamentale de proiectare
3. Semnale. Parametri generici. Constante
4. Operatori. Tipuri de date
5. Atribute
6. Domeniul secvential. Procese
7. Instructiuni Secventiale
8. Domeniul concurrent
9. Instructiuni Concurente
10. Sub-programe
11. Module de simulare
12. Pachete standard si predefinite

Orar Laborator

1. Introducere lab + proiecte
2. L1 + L2
3. P1 – schema bloc
4. L3
5. P2 – scheme detaliate
6. L4 + L5
7. L6 + L7
8. L8 + L9
9. P3 – implementare proiecte
10. P4 – implementare proiecte
11. L10 + L11 + L12
12. Predare proiecte - 1
13. Test laborator
14. Predare proiecte - 2

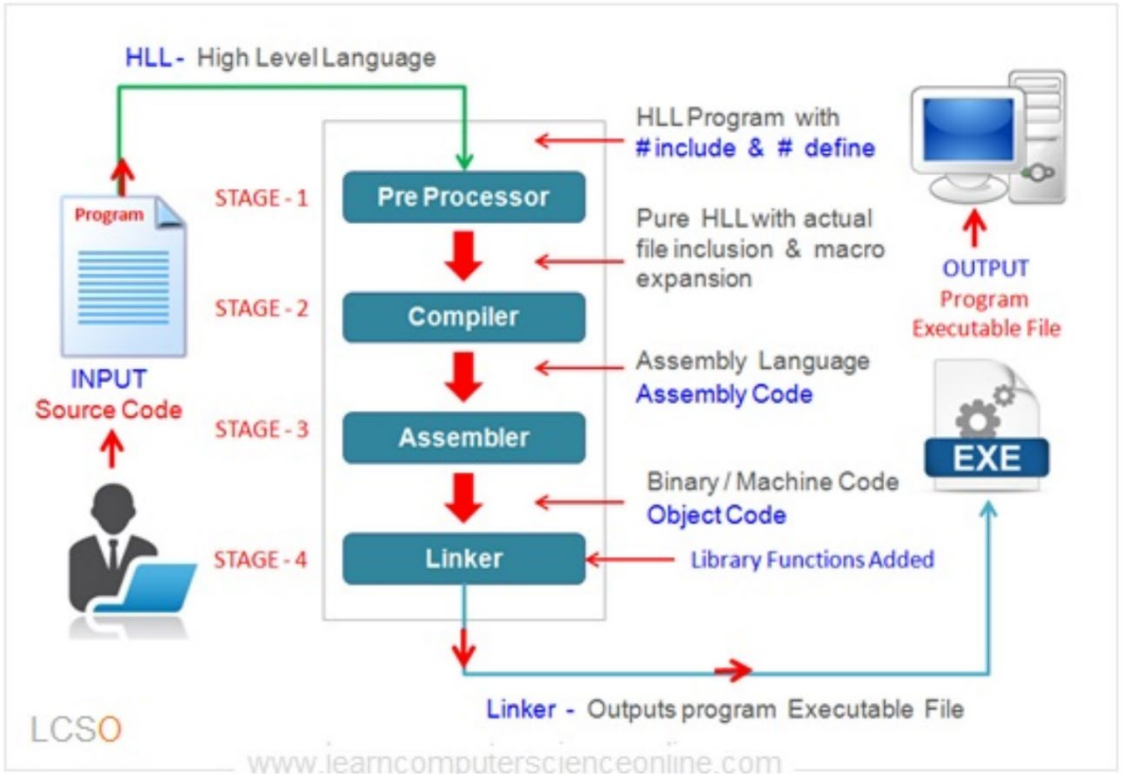
Limbaj de descriere hardware

Obiectivul principal al materiei: invatarea unui limbaj de descriere hardware

- Cum pot sa creez eu componente hardware?
  - Trebuie sa existe fizic?
  - Cum pot sa interactionez cu ele?
  - Pot sa le “sterg” dupa ce le-am creat?
- Cum ar trebui sa arate un limbaj de descriere?
- Care e legatura intre componente hardware (MUX-uri, numaratoare etc.) si instructiunile pe care le scriu?
- Pot sa scriu instructiuni ca si pt un program C?
  - Trebuie sa inteleg ce rol are fiecare instructiune in context diferite

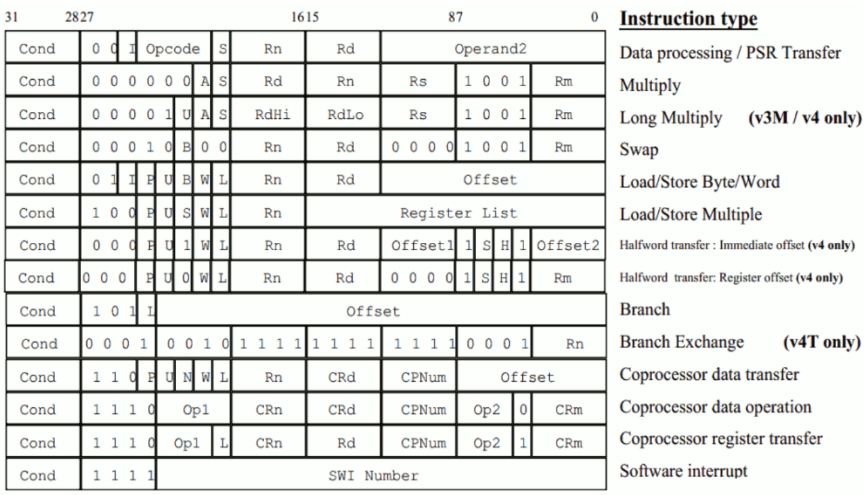
Compilarea folosind un procesor

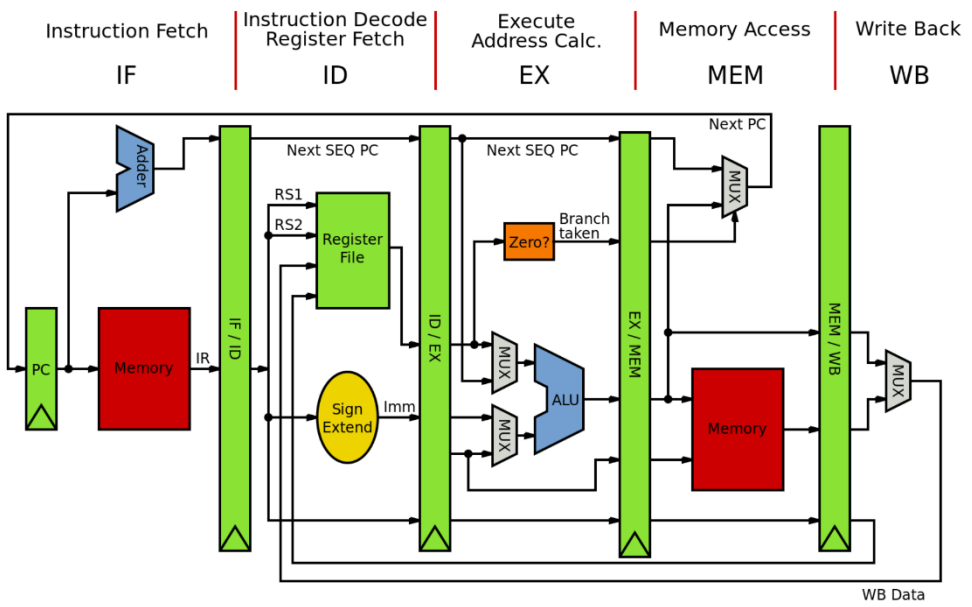
- Etapele compilarii codului (ex. C)
  - Preprocesare
  - Compilare
  - Asamblare
  - Linkeditare
- Primeste codul sursa
- Rezulta un **cod binar**
  - **Instructiuni fixe**
  - Se executa pe procesor
- Ce e “hardware” aici?



CPU (Central processing unit)

- Avem un set de instructiuni care trebuie executate
- Fiecare instructiune are un camp, care este transmis spre o componenta hardware (Memorie, Registru, MUX,)

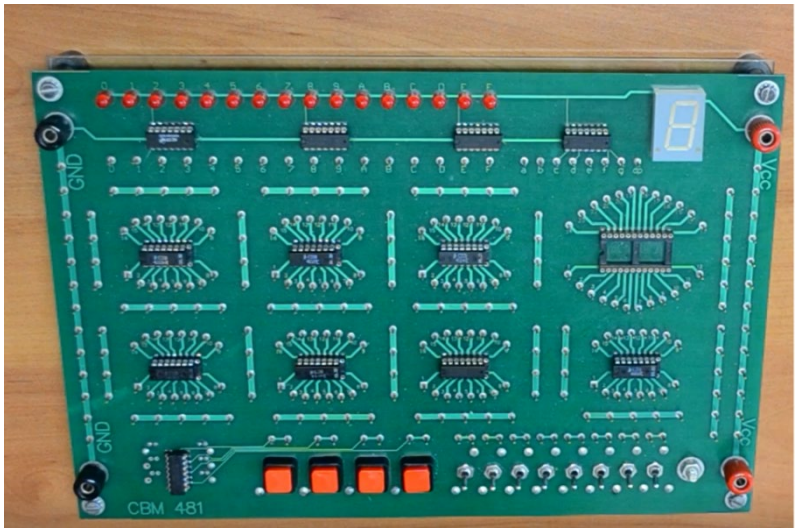




- CPU: Unitatea hardware care executa astfel de instructiuni
  - FIXA!!!
  - Permite accesarea/sau nu a unor componente hardware
  - Ex: Arhitectura MIPS
- Important: ARHITECTURA HARDWARE NU SE SCHIMBA!!!

ASIC

- Hardware in semestrul 1 (Proiectare Logica):
  - Am invatat cum sa construim circuite hardware
    - Porti logice
    - CLC
    - SLC
  - Am folosit circuite integrate
    - Create de altcineva
    - Am folosit fire pentru a le lega
    - Le-am legat pt generarea unor noi componente
  - “Proiectarea” = interconexiuni de circuite existente



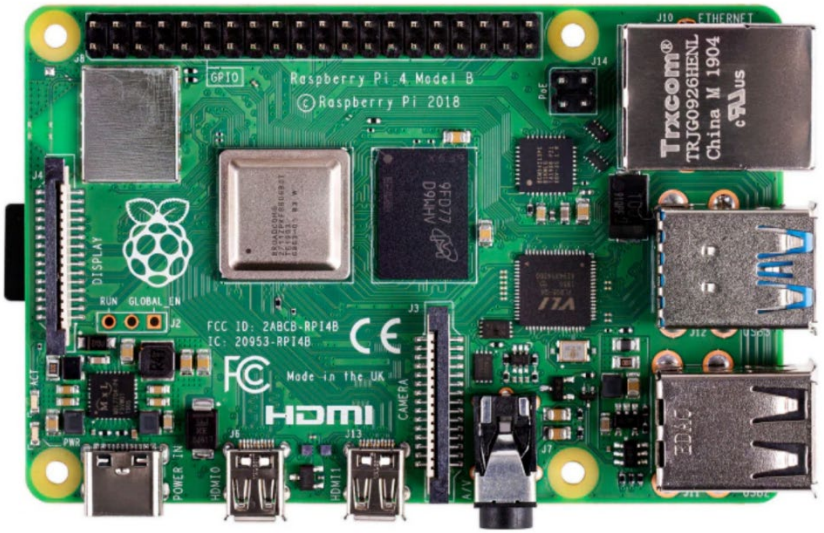
- ASIC
  - Application-specific Integrated Circuit
  - Circuite deja existente, cu legaturi fixe
  - Majoritatea circuitelor de care ati auzit
  - Procesoare: Intel, AMD, MIPS, ARM
  - Microprocesoare/mictocontrollere: Raspberry Pi, Arduino etc.
  - GPU: Nvidia

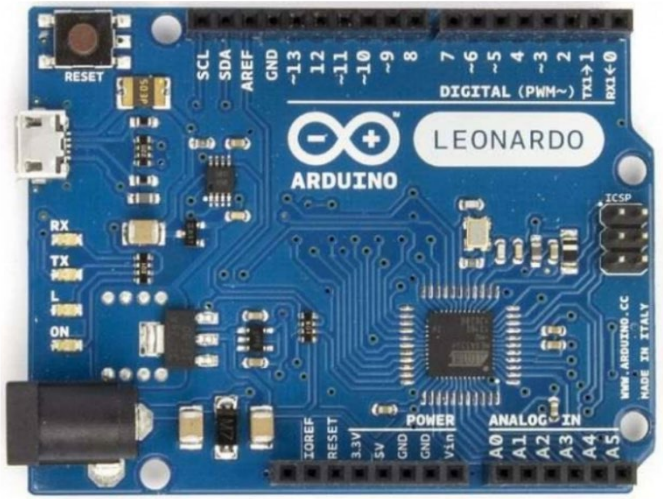
ASIC - CPU

- Obiectiv principal: prelucreaza si proceseaza informatia primita de la intrari si o trimite spre iesiri
  - Instructiuni aritmetice, logice: +,-,*,/,<<,>>,|| ...
  - Instructiuni de lucru cu memoria: load, store
  - Instructiuni de intrare/iesire:
- “General-purpose”
  - Nu sunt specializate, deci nu sunt f performante
  - Permit o variatate mare de aplicatii
- Mai multe tiputi de arhitecturi
  - Difera, in functie de setul de instructiuni
  - Majoritatea: 32/64b per instructiune
  - RISC/CISC
  - Difera in functie de sistemul de memorie
  - Pot utiliza mai multe nuclee, in paralel
- Mai multe informatii:
  - Arhitectura Calculatoarelor (An2 Sem 2)
  - Structura Sistemelor de Calcul (An 3 Sem 1)

ASIC - Microcontroller

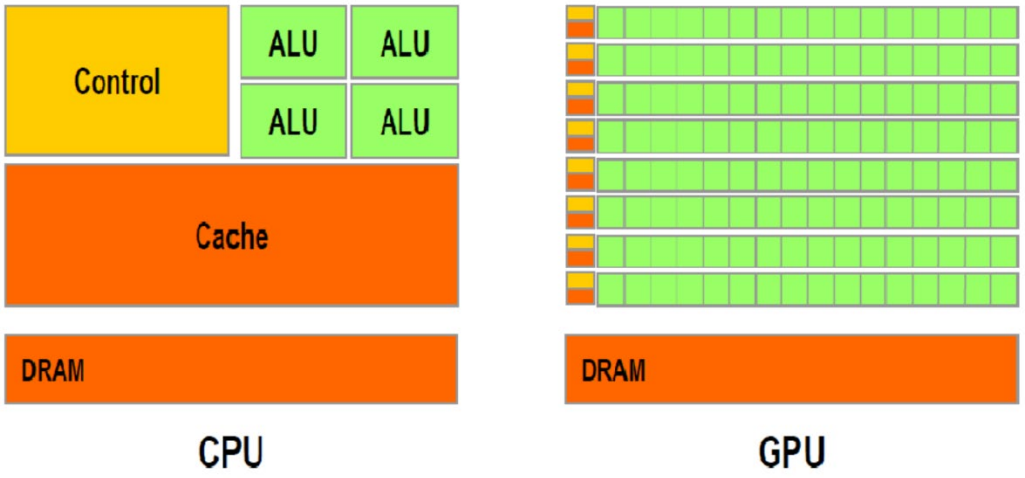
- ASIC cu resurse limitate, folosit pentru anumite aplicatii
- Un singur chip, care contine toate componentele necesare
- Contine:
  - Un CPU cu resurse limitate
  - Memorie limitata
  - Porturi seriale/paralele
  - Porturi pentru I/O
  - Controller pt intreruperi
- Application-specific
  - Optimizat pentru o singura aplicatie
  - Nu contine multe instructiuni – resurse putine
  - Nu e nevoie de viteza mare
  - Nu ofera flexibilitate
- Exemple: Arduino, Raspberry Pi
- Mai multe informatii:
  - Proiectare cu Microprocesoare (An 3, Sem 1)





ASIC - GPU

- Circuit specializat, utilizat pentru accelerarea anumitor calcule
- Original – utilizat pentru afisarea de imagini
- Recent – utilizate pentru aplicatii complexe (ex. Retele neuronale convolutionale)
- Au o structura paralela
  - Multe nuclee de procesare, fiecare cu bloc de memorie personal
  - Foarte utile pentru paralelism – operatii executate in acelasi timp
  - Au nevoie de un CPU, care sa ofere partea de control/comanda
- Au nevoie de limbaje de programare specializate (ex. Cuda)

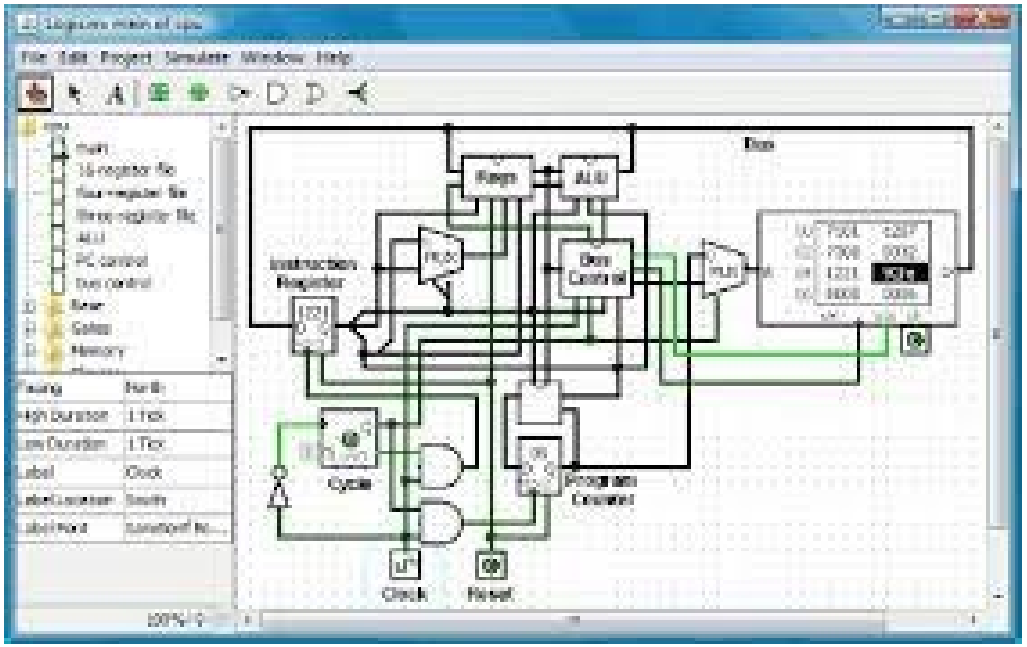


ASIC - GPU

- Importanta paralelismului: CPU vs GPU

Cum putem crea circuite noi?

- Semestrul 1:
  - Am utilizat circuite
    - Circuite integrate
    - 7400, 74192
  - Am descris noi circuite
    - Logisim/Xilinx ISE - diagrame
    - “Drag-and-drop”
    - Diagrame bloc – legaturi virtuale
    - Creare de circuite noi – noi bloc-uri
    - Simulare
  - Proiectare ierarhica

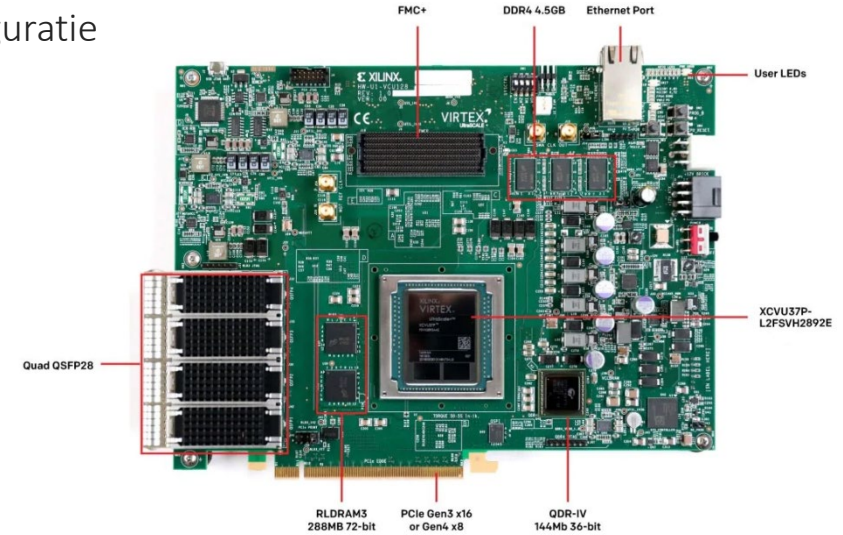


- Semestrul 2:
  - Vom invata cum sa realizam noi circuite, care sa functioneze fizic, dupa anumite specificatii
  - Vom scrie cod pentru generarea lor
  - Vom testa circuitele “inventate”/proiectate de noi

FPGA – Descriere

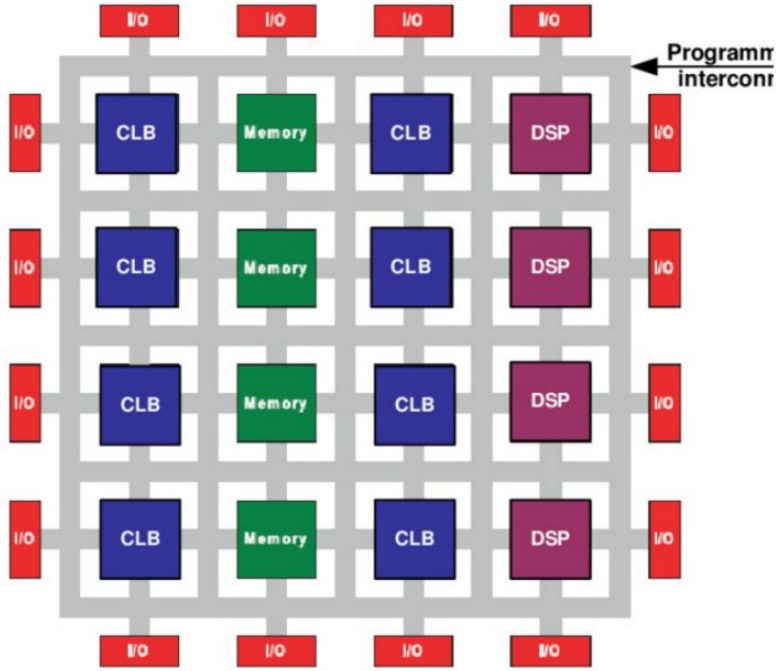
- Sa presupunem ca stim sa descriem circuitele
  - Cum putem sa le creem fizic? Cum le interconectam?
  - V1: Avem nevoie de pistol de lipit, cositor si tranzistoare
  - V2: Avem nevoie de niste circuite speciale...

- FPGA
  - Field Programmable Gate Array
  - Hardware reconfigurabil
  - Celule are pot fi “configurate”
  - O celula:
    - Poate avea rol de poarta logica SI pentru o configuratie
    - Poate avea rol de poarta logica SAU pentru o configuratie
  - Putem interactiona cu hardware-ul creat
    - Periferice – switch-uri, butoane, led-uri
    - Pot fi si ele configurate

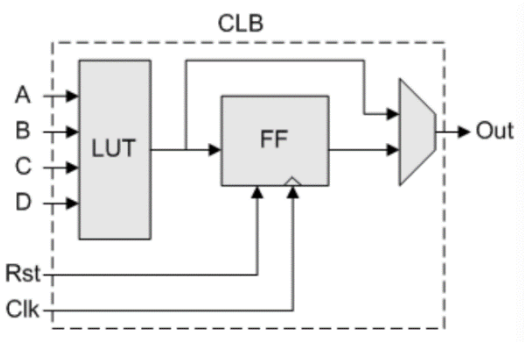


- Exemplu:
  - Basys3, Nexys4
  - Familii: Spartan, Basys, Nexys, Virtex, Artix
  - Difera, in functie de dimensiune, viteza, I/O etc.

Structura FPGA



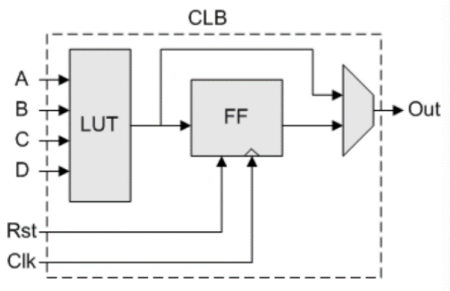
- **CLB** – Configurable Logic Block
- Celula logica programabila, care contine:
  - Parte combinationala – **LUT** – Look-up Table
    - Implementeaza functii logice
  - Unitati de memorie – **bistabile D**
    - Comuta pe frontal ceasului
    - Pot fi configurate si ca Latch-uri asincrone
- Matrice de interconexiuni
  - Stabileste legaturile intre CLB-uri si cu I/O
- Memorii suplimentare
  - Pot fi configurate pentru aplicatii care au nevoie
  - Si bistabilele din CLB pot fi utilizate cu rol de memorie



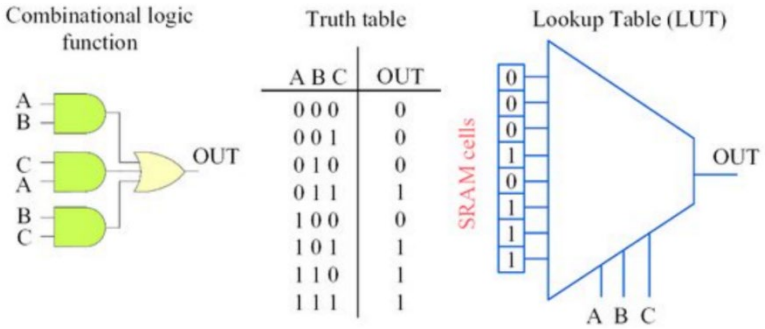
- DSP – digital signal processing units
  - unitati aritmetico-logice special, utilizate pentru calcule complexe
- I/O – block de intrar/iesiri – conexiuni cu switch-uri, butoane, led-uri, SSD etc.

Look-up Table

- Celula logica de baza, pentru functiile combinationale
- Similar cu un tabel de adevar pentru o functie/o parte din functie
- Utilizarea LUT se bazeaza pe principiul de “Functie universala”
- **Functii universale**: blocuri logice care pot fi configurate astfel încât să realizeze orice funcţie logică data de intrările blocului



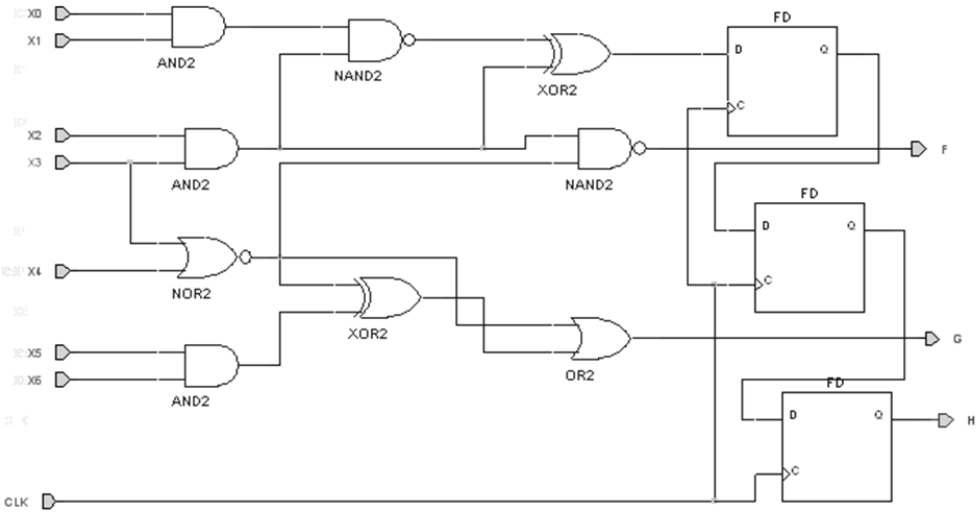
- Tipuri de astfel de blocuri logice:
  - Memorii
  - Multiplexoare
  - toate aceste blocuri logice pot realiza funcţii booleene formând tabelele lor de adevăr
- Exemplu:

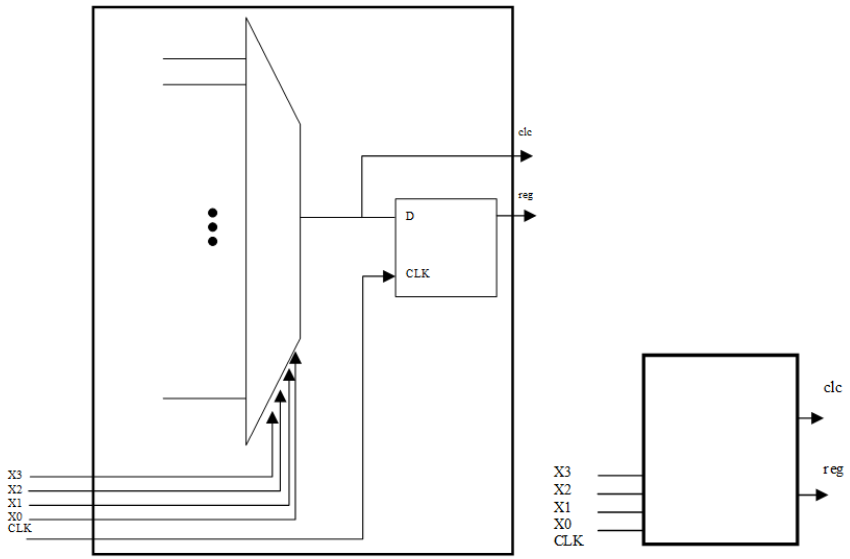


- Pentru circuite secventiale, se combina functia data de LUT cu un bistabil D
- Pentru functii de mai multe variabile, e nevoie de LUT-uri mai mari sau de mai multe LUT-uri combinate

Implementarea unui circuit cu LUT+FF

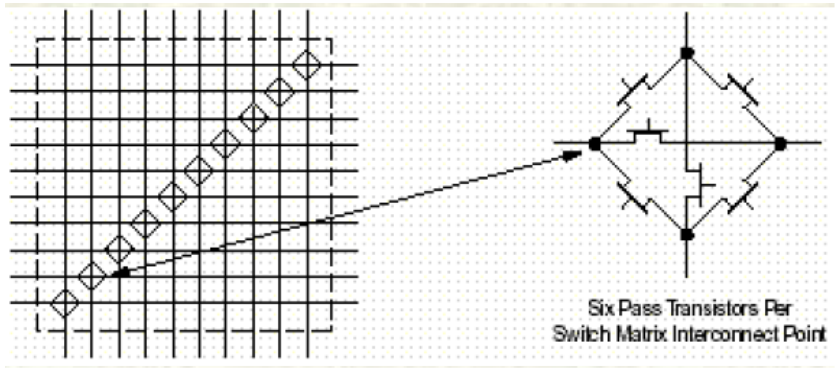
- Cum credeti ca s-ar “mapa” in CLB-uri, urmatorul circuit?
  - CLB are un LUT 4:1 si un bistabil D

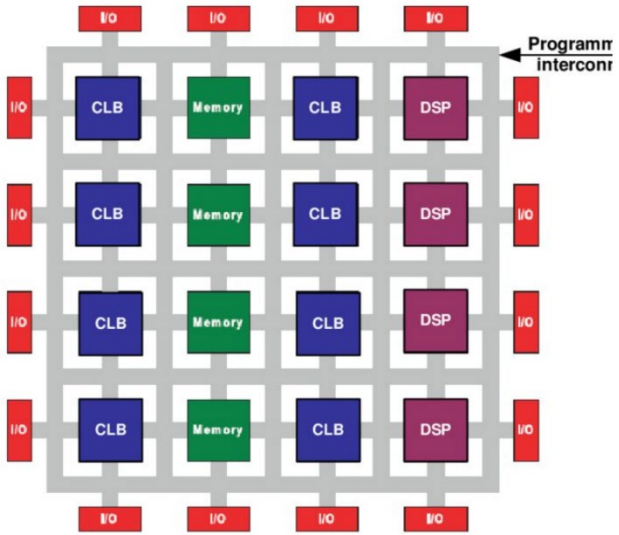




Arhitectura de interconectare

- Fire metalice cu puncte de conexiuni programabile
- Exista o matrice de conectare
- Memorie de interconectare SRAM - fiecare bit este dedicat controlului unui punct de conectare intern (PIP – Programmable Interconnection Point)
- Blocurile de I/O au un inel de rutare separat, in jurul matricei CLB-urilor





Generarea Circuitului in FPGA

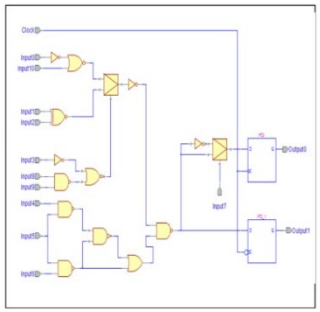
- Presupunem ca pornim cu descrierea circuitului intr-o forma compilata, verificata si simulata

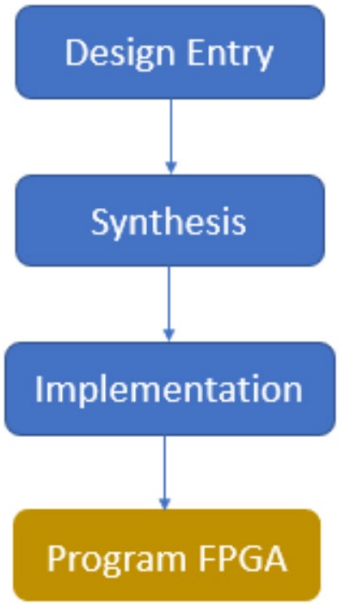
- Sintetizare
  - Transformare din diagrama bloc/cod in reprezentare fizica

VHDL description

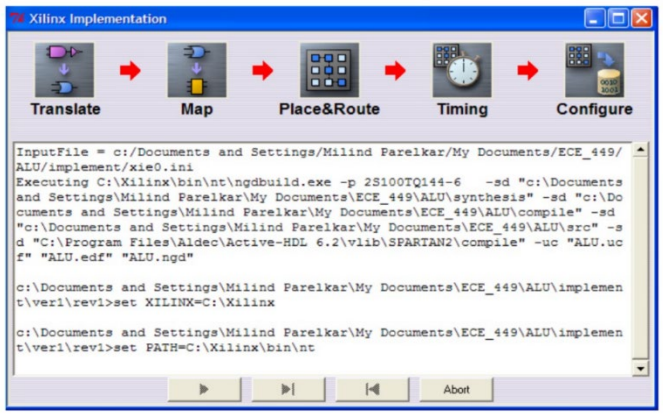
```
architecture MLU_DATAFLOW of MLU is

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;
```

Circuit netlist





- Implementare
  - Legaturi cu pinii de I/O (exemplu, conectare la switchuri/butoane)
  - Pozitionare optima in FPGA
    - Mapare
    - Plasare+rutare
    - Exemplu: Introducere de etaje de pipeline

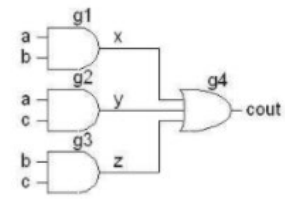


- Programare
  - Circuitul va fi descarcat pe FPGA

Sintetizare - Translatare

- Pasul 1 din workflow
- Obiectiv: translatarea functiilor dorite de proiectant in conexiuni intra si inter-CLB
- Contine 2 sub-etape:
  - Translatarea efectiva – converteste functiile in functii realizabile
  - Verificare – verifica daca translatarea este corecta
- Rezultatul etapei: **NETLIST**
  - **Lista de componente**
  - **Lista de conexiuni intre componente**

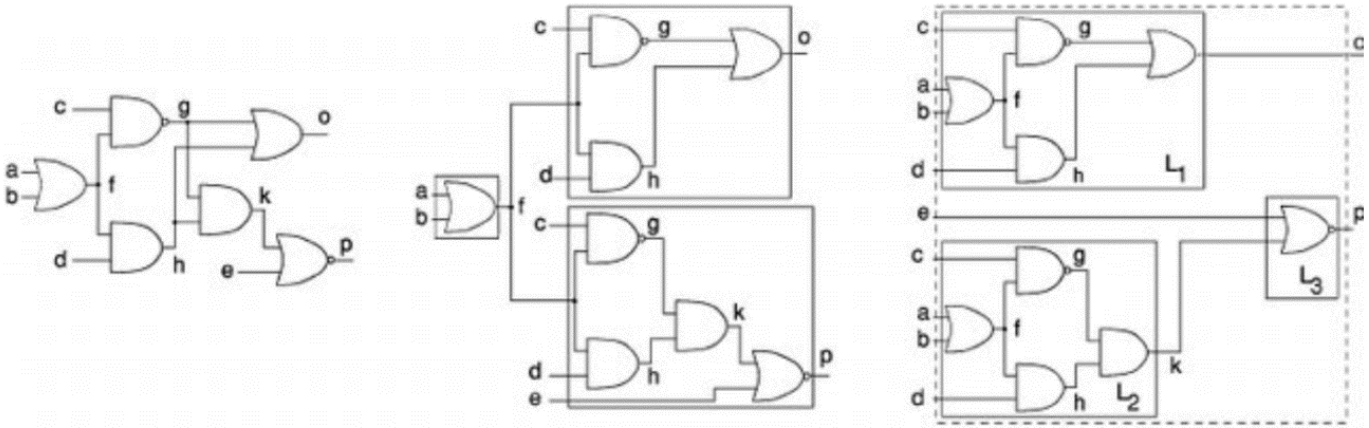
```
module carry(input  a, b, c,
             output cout)

   wire   x, y, z;

   and g1(x, a, b);
   and g2(y, a, c);
   and g3(z, b, c);
   or  g4(cout, x, y, z);
endmodule
```



- Pasii procesului de translatare:
  - se verifică dacă numărul pinilor de intrare / ieşire este mai mare decât numărul pinilor necesari în proiect
  - se verifică dacă există suficiente celule pentru a acoperi numărul de porţi logice din proiect
  - se caută în proiect grupuri de componente al căror număr de intrări şi de ieşiri este egal cu cel al celulei logice de bază
  - se aleg întâi grupurile cele mai mari care pot fi formate

Implementare – Mapare Tehnologica

- Evidentierea legaturilor dintre semnale de I/O si pinii de I/O
- Optimizarea Netist-ului
  - Se combina eventuale functii similar
  - Se minimizeaza functiile folosind teoremele algebrei booleene si diagrame Karnaugh
- Transformarea din functii logice in LUT – element de baza al CLB
  - In functie de resursele existente pe FPGA
  - LUT4/ LUT6 – depinde de tehnologia utilizata
  - Pot fi si alte tipuri de CLB (cu 2-3 LUT-uri si mai multe FF)
  - Exemplu de la slide 19 – mapare tehnologica
- Rezulta un nou netlist - optimizat
  - In care se pastreaza designul sub forma de interconexiune de LUT



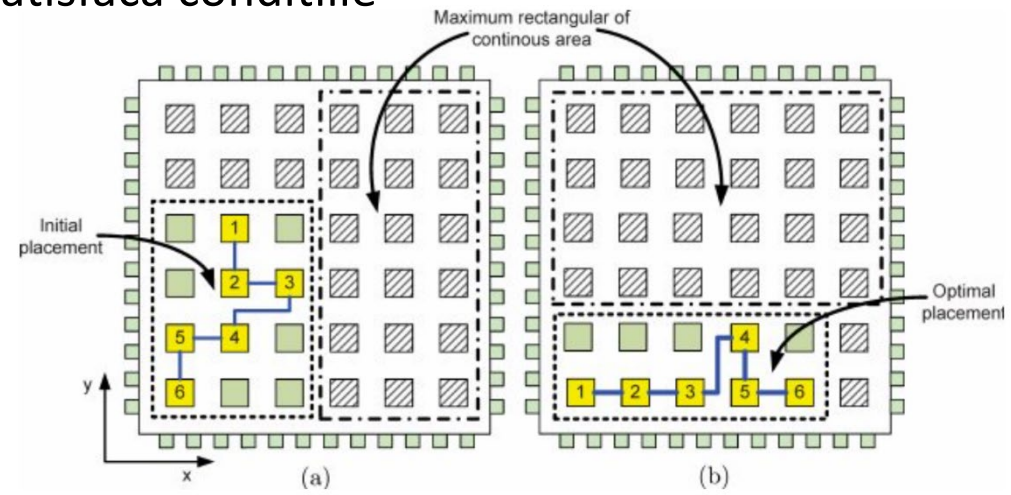
Implementare - Plasare

Plasare

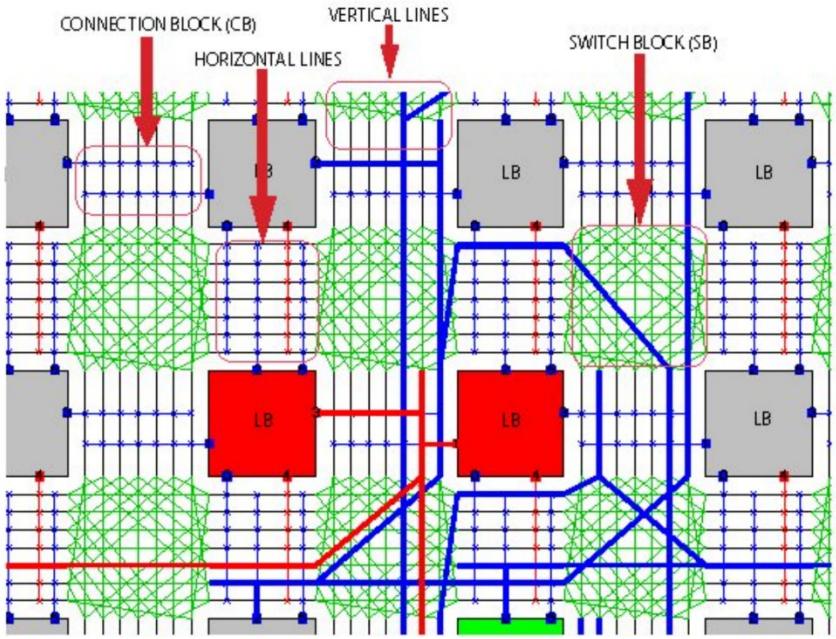
- Proces de **amplasare fizică** a componentelor logice în celulele logice din circuitele programabile
- Se urmăreşte amplasarea funcţiilor logice adiacente în celule alăturate
- Criteriul critic de plasare – realizarea legăturilor între cellule

Etape

- Se incearca o plasare initiala a circuitului pe FPGA
- Se verifica daca se indeplinesc conditiile impuse (timp, pini I/O etc.)
- Daca nu, se face o re-plasare care sa satisfaca conditiile
  - Pe baza unui alg de AI
  - Simulated Annealing
  - Functioneaza iterativ
  - Pana se gaseste o plasare optima



Implementare - Rutare



Interconectarea

- După plasarea adecvată
- Inspectarea netlistei şi a plasării curente
- **Linii de conectare "consumate"**
  - la un capăt au un semnal de ieşire
  - la celălalt capăt un semnal de intrare
- **Congestie** - nu mai sunt linii de conectare disponibile
- **Re-rutare** - reluarea procesului de plasare şi rutare
- Rezultat - fişier care descrie proiectul iniţial în termenii celulelor circuitelor programabile
- Fişierul descrie asignarea poziţiilor celulelor şi interconexiunile dintre celule
- Fişierul va fi translatat într-o **hartă de biţi** care va fi transmisă unui programator de circuite programabile ⇒ **configurarea**
- Principalul factor in rutare: **caile critice**

FPGA – Avantaje si dezavantaje

Avantaje

- Pot fi proiectate la nivel de poarta logica
  - Implementare rapida
  - **Nivel mare de paralelism!**
- Pot fi re-programate – flexibilitate mare, se poate pune orice circuit, oricand si apoi se poate sterge/reprograma
- Cost mic – nu necesita tool-uri de lipire
- Proces automatizat de mapare, plasare, rutare – nu necesita timp pierdut

Dezavantaje

- Necesita cunostinte de HDL – nu sunt asa simple ☺
- Nivel ridicat de consum electric – nu exista optimizare
- Trebuie sa se tina cont de resursele existente (LUT, bistabile, memorii)
- Viteza redusa pentru orice aplicatie obtinuta, comparativ cu un ASIC
  - Multe resurse aditionale, neutilizate

FPGA – de ce sa folosim?

- Se pot crea circuite customizabile, in functie de nevoia utilizatorului
- Se pot adauga usor periferice
  - intrari de la butoane, intrari externe
  - comunicare usoara cu alte dispozitive
- Se pot crea procesoare modificabile
  - Putem adauga orice instructiune avem nevoie, pentru o anumitta functionalitate
  - Ex: FMA - instructiune care face si inmultire, si adunare, intr-un sinur ciclu
- Se pot obtine viteze mari, pentru anumite calcule/aplicatii
  - Nivel mare de paralelizare
  - Putem avea mai multe circuite, care functioneaza simultan
- Foarte utile pentru prototipizare
  - Se poate crea un circuit
  - Se poate testa in conditii reale
  - Apoi se poate trimite in productie – rezulta un ASIC
- Foarte utile pentru **invatare**

FPGA – aplicatii interesante

Instrumente de proiectare - Lab

Etapa 1: Scris cod VHDL + implementare circuite simple

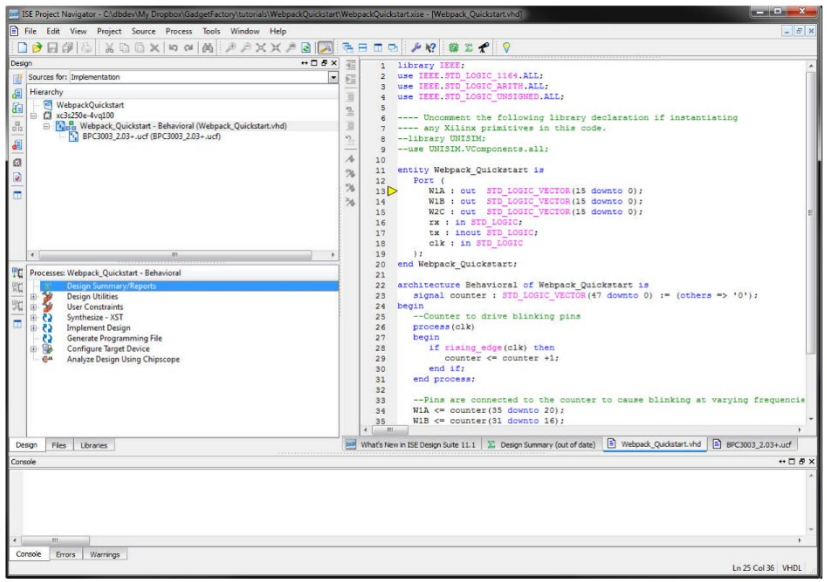
- Vivado – pus direct pe placa + vizualizare schematic

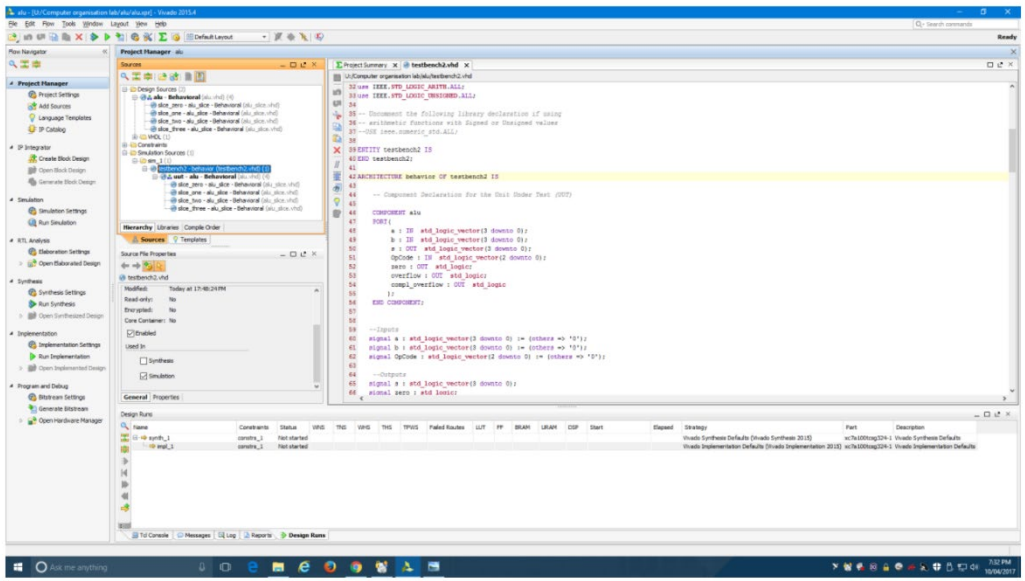
Etapa 2: Scris cod VHDL + module de simulare – circuite complicate - simulare

- Vivado/Xilinx ISE – doar cu modul de simulare

Etapa 3: Scris cod VHDL + implementare FPGA – circuite complicate - implementare

- Vivado/Xilinx ISE – translatare, mapare, plasare, rutare, programare FPGA





VHDL - INTRODUCERE

VHDL

- VHSIC - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit
- HDL - **H**ardware **D**escription **L**anguage
- Prima varianta - 1980; standard 1987; extins 1993; variantă 2004; acum 1076/2008

Cum am putea descrie “algoritmic” circuite hardware?

- Descriere folosind diagrame bloc (exemplu sem 1)
  - Probleme la design-uri mari (ex. Conexiuni)
  - Anumite structuri – repetitive, pot fi descrise mai usor “algoritmic”
  - Exemplu: MUX (simplu if); counter (incrementare)
- Limbaje de programare
  - Permit doar excutia secventiala a unor instructiuni
    - Can “se termina” executia lor, nu mai exista
    - De obicei se executa pe un circuit fizic/procesor
  - Avem nevoie de o descriere care sa “creeze” circuitul

INTRODUCERE VHDL

HDL: limbaj de descriere a sistemelor electronice hardware

- Contine:
  - structură de blocuri
  - relaţii
  - interconexiuni
- VHDL definit şi integrat în instrumentele CAD (Computer-Aided Design)
- toate instrumentele CAE (Computer-Aided Engineering) - produse cu intrări / ieşiri
- Usor de interpretat si “citit” pentru a fi implementat pt circuite fizice

Atentie: INTOTDEAUNA TREBUIE PRIVIT CA UN LIMBAJ DE DESCRIERE HARDWARE, NU UN LIMBAJ DE PROGRAMARE!!!

- Limbaj de programare – instructiuni – cod masina – ruleaza pe un processor/circuit existent
- Limbaj de descriere – instructiuni – genereaza un nou circuit hardware!

DOMENII DE APLICARE

Obiective VHDL

- **Specificare** sisteme hardware
  - Descrierea functionalitatii/structurii
  - Pregatire pentru etapele de sintetizare + implementare
- **Simulare** evoluţie temporală a descrierilor
  - Instrumentele de simulare – tot structuri VHDL
  - Realizează simularea (“execuţia”) codului VHDL în paralel
  - Codul nu descrie modul de proiectare sau de realizare a funcţiei, ci doar ce trebuie să facă aceasta
  - De exemplu – numarator
    - Asteptam ca la fiecare ciclu de ceas sa se incrementeze
    - Trebuie sa creem un semnal de ceas
    - Trebuie sa verificam la fiecare interval de timp noua valoare a semnalului

DOMENII DE APLICARE

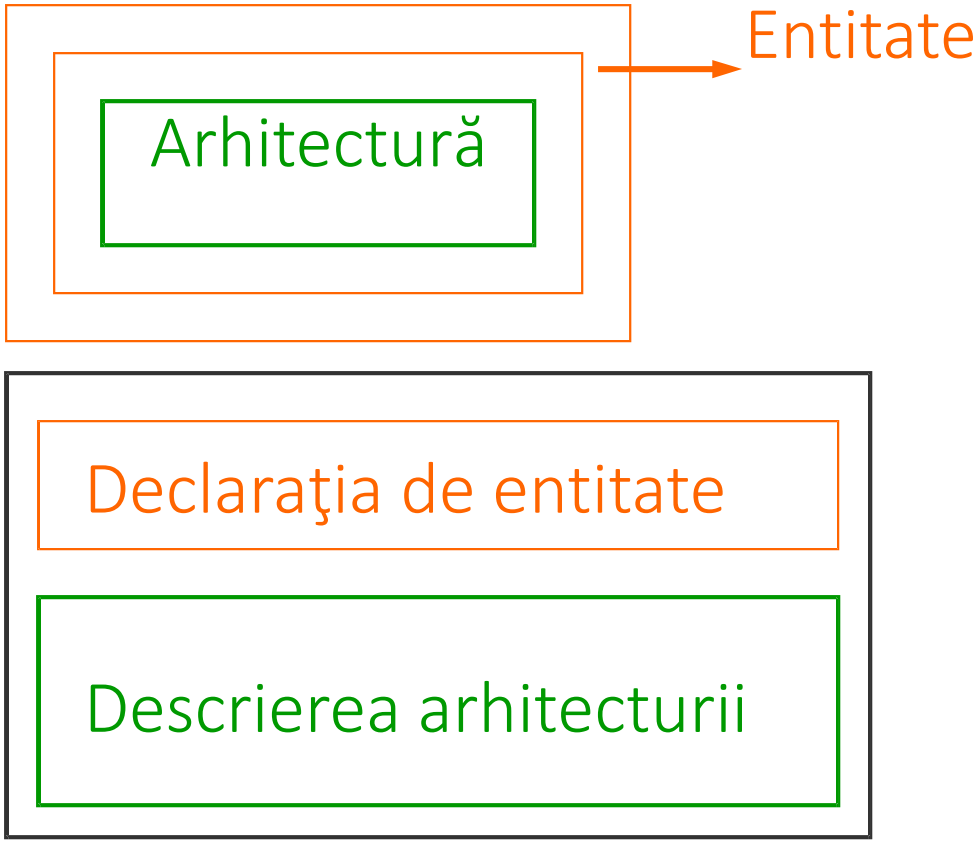
Obiective VHDL (continuare)

- **Sinteza logică** în cadrul instrumentelor CAD care integrează VHDL (fază automatizată)
  - Descrierea proiectării unui system
    - Descrierea funcţionării
    - Descrierea structurii exacte a fiecărei părţi
  - Descrierea realizării finale - interconexiuni de componente logice elementare
  - Porneşte de la **o descriere VHDL sintetizabilă**
  - Conduce la o **schemă logică clasică** (porţi logice + bistabile)

STRUCTURA VHDL

Proiectare ierarhică

- model VHDL: **pereche entitate + arhitectură**



STRUCTURA VHDL

- Entitatea - declaraţie a intrărilor şi ieşirilor modulului
- Arhitectura
  - Descriere detaliată a **structurii** modulului
    - Exemplu: utila pt descrierea unui sumator pe 4 biti
  - Descriere detaliată a **funcţionării** modulului
    - Exemplu: utila pt descrierea unui counter
- **Proiectare ierarhică**
  - Se pot genera componente de complexitate mare, folosind sub-componente de complexitate mica
  - Exemplu
    - un procesor e format din: memorie date, ALU, memorie instructiuni;
    - Un ALU e format din sumatoare, scazatoare, porti logice
    - Un sumator e format din porti logice

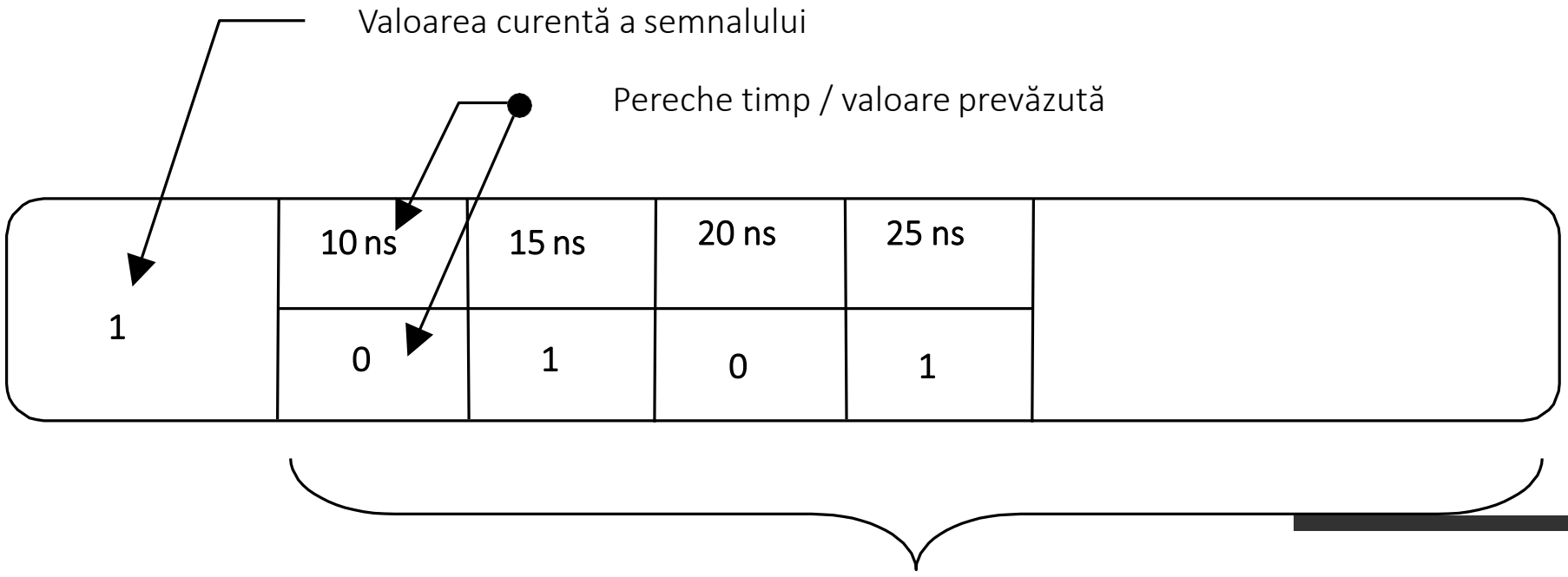
CARACTERISTICI

- Sistemele hardware - în mod natural concurente
  - Diferite componente, au acces simultan la diferte resurse
  - Se poate stabili o ordine a accesului la resurse
  - Ca urmare, toate instructiunile vor fi executate in **paralel**
  - Atentie: nu putem conecta simultan 2 componente la aceeasi resursa!
- Problema: Nu toate componentele pot fi declarate de o singura instructiune!
- **Modelare VHDL (2 moduri)**:
  - domeniul concurent
  - domeniul secvential

CARACTERISTICI

Obiecte in VHDL

- Constante si variabile (mai putin utilizate)
- **Semnale – s**pecifice sistemelor hardware
  - Modelează informaţia care tranzitează între componente (legătură fizică prin fire)
  - Există tot timpul simulării, indiferent de zona de vizibilitate
  - Pot modela intrari/iesiri sau informatii intermediare
  - Pilot (driver) de semnal:



CARACTERISTICI

Obiecte in VHDL

- Orice obiect clasificat într-un **tip**
  - tipul este obligatoriu şi nu se schimbă niciodată
  - tipurile impun valori şi operaţii permise şi interzise
- 4 tipuri:
  - scalare (întregi, flotante, fizice, enumerate)
  - compuse (tablouri, articole)
  - acces (pointeri)
  - fişier
- tipuri predefinite:
  - **bit, bit-vector – most used! (precizie variablila, specificata)**
  - boolean, character, integer, real (precizie predefinita)
  - severity-level, string

CARACTERISTICI

Funcţii şi proceduri

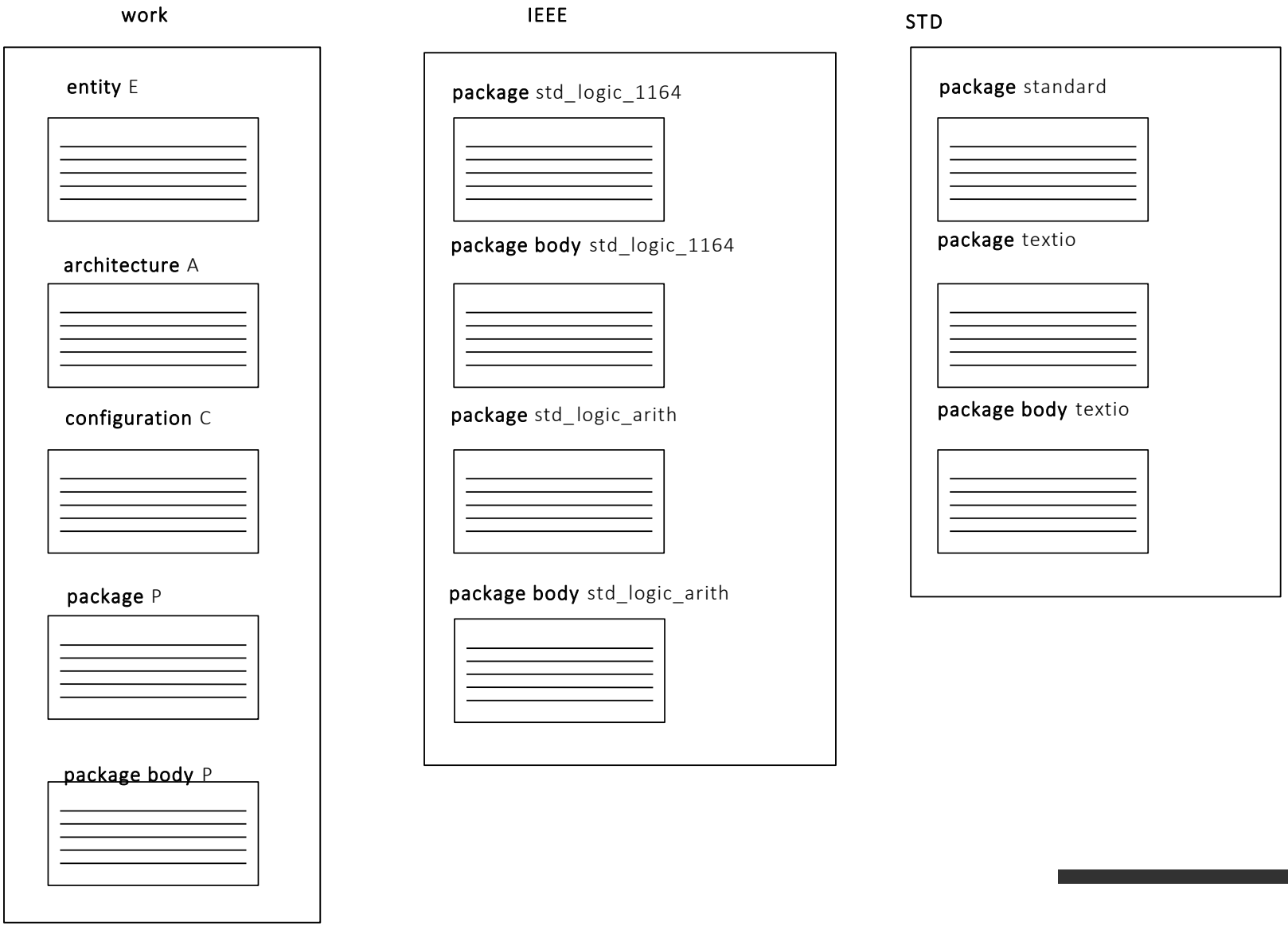
- Sunt folosite mai putin pentru descrierea componentelor (spre deosebire de limbajele de programare)
- In general au rol de generare a unor expresii **la un anumit moment (nu pot tine minte o stare)**
- Exemplu: avem nevoie de un numarator pana la X
  - Trebuie sa decidem pe cati biti functioneaza numaratorul
  - Putem scrie o functie care sa calculeze log(X)
- Funcţiile:
  - argumente - au tip definit
  - returnează rezultat - are tip definit
- Procedurile
  - argumente - au tip definit
  - se pot folosi în locul unei instrucţiuni secvenţiale

CARACTERISTICI

Biblioteci şi pachete

- VHDL limbaj modular
  - Se descriu unităţi mici, ierarhizate
  - Descrieri compilate separat = **unităţi de proiectare**
    - Fiecare componenta separata, poate fi testata
    - Apoi poate fi incapsulata intr-o alta componenta
  - Unităţile de proiectare salvate în **biblioteca de lucru (“proiectul curent”)**
    - generată de mediul VHDL: **WORK**
  - **Biblioteci de resurse** - apelare cu **library**
    - Contin componente realizate anterior
    - Se declara inainte de unitatea de proiectare
    - Folosire cu **use** unitate**.all**

CARACTERISTICI



CARACTERISTICI

Biblioteci şi pachete

- **Bibliotecile conţin doar unităţi de proiectare**
- Fişierele sursă (cu cod VHDL) analizate şi compilate nu mai există pentru proiectant
- După compilarea fişierelor - **utilizam (referim) doar unitati de proiectare**
- UP – “black box”
  - Trebuie cunoscuta functionalitatea
  - Trebuie cunoscute semnalele de in/out

CARACTERISTICI

Biblioteci şi pachete

- Pentru utilizare ulterioara – trebuie create UP
- Continutul unei unităţi de proiectare:
  - **Entitate (interfaţa sistemului)**
  - **Arhitectura (descrierea sistemului)**
  - Specificaţie de pachet (vedere externă a posibilităţilor puse la dispoziţie)
  - Corp de pachet (descrierea internă a funcţionalităţilor)
  - Configuraţie (asociere componentă - model)

Concluzii

- Circuite hardware
  - Circuite integrate
  - FPGA
- Limbajul VHDL
  - Generalitati
  - Domenii de aplicare – specificare, simulare, sinteza
- Structura unui program VHDL
  - Structura ierarhica
- Caracteristici VHDL
  - Obiecte in VHDL
  - Unitate de proiectare
  - Biblioteci si pachete

Multumesc pt atentie!