

# Laboratorul 3 - Sintetizarea modulelor Verilog

În laboratoarele anterioare au fost prezentate elementele principale ale limbajului Verilog și au fost descrise o serie de circuite digitale, folosind acest limbaj de descriere a hardware-ului. Verificarea comportamentului descris s-a făcut folosind simulatorul ISim, inclus în mediul de dezvoltare Xilinx ISE.

Scopul final pentru un cod Verilog este însă de a realiza implementări hardware care se comportă precum descrierea Verilog. Acest lucru poate fi obținut în mod automat printr-un proces numit sintetizare (eng. *synthesis*). Un utilitar de sintetizare preia descrierea Verilog sau VHDL și poate genera fișiere de configurare pentru circuite integrate reconfigurabile (ex. PLA, CPLD, FPGA) sau chiar măștile necesare pentru realizarea unui ASIC (*Application-Specific Integrated Circuit*) prin diferite tehnologii \*MOS.

Există mai multe utilitare capabile de sintetizarea circuitelor digitale, de la diverși producători, care sunt în principal axate pe anumite tehnologii specifice acestora. Pentru programarea plăcilor de dezvoltare de la laborator, care sunt dotate cu FPGA-uri de la Xilinx, se folosește mediul de dezvoltare Xilinx ISE care, pe lângă simulatorul ISim, conține și un utilitar de sintetizare.

## FPGA-uri

Un FPGA (Field-Programmable Gate Array) este un circuit integrat care poate fi programat pentru a se comporta ca orice alt circuit digital. Spre deosebire de un procesor, care stochează și execută instrucțiuni, programarea unui FPGA înseamnă reconfigurarea hardware a acestuia pentru a realiza funcționalitatea dorită.

Primele FPGA-uri au fost introduse în anii '80, iar utilizarea lor principală era în testarea prototipurilor pentru ASIC-uri. Avantajele aduse de FPGA-uri au făcut însă ca ele să fie folosite acum, atât pentru testare, cât și în multe alte domenii precum prelucrarea de semnale, criptografie și HPC (High Performance Computing).

FPGA-urile sunt construite dintr-un număr mare de blocuri logice configurabile (eng. *configurable logic block* - CLB), identice, interconectate printr-o matrice de fire și switch-uri programabile (Fig. 1).

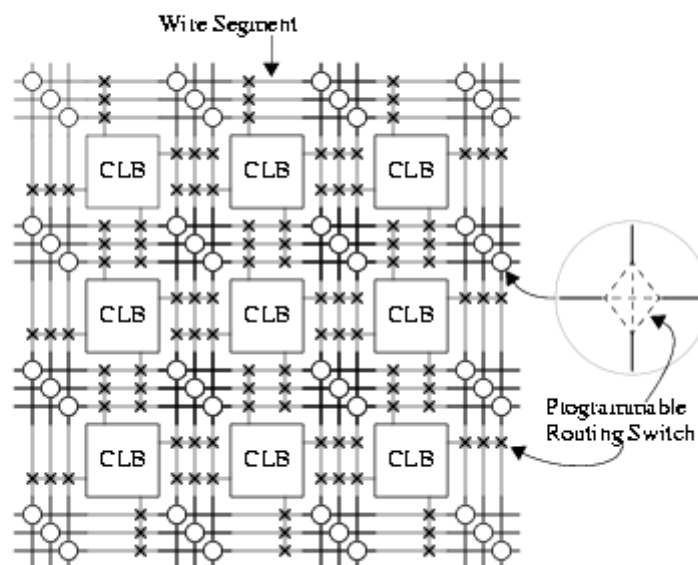


Fig. 1: Structura unui FPGA

În general, un CLB este compus dintr-un element combinațional reconfigurabil și un bistabil. De obicei elementul combinațional este un LUT (Look-Up Table), iar bistabilul este de tip D. Arhitectura unui bloc diferă însă de la producător la producător, motiv pentru care trebuie folosite utilitățile de sintetizare specifice producătorului. Fig. 2 prezintă un CLB tipic.

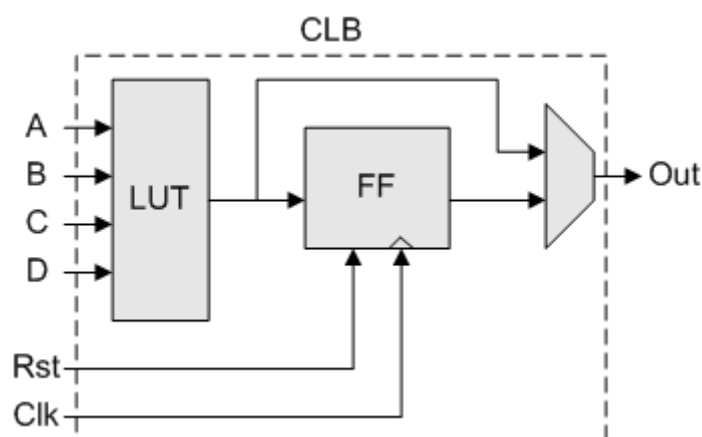


Fig. 2: Configurable Logic Block

Numărul de celule dintr-un FPGA variază de la model la model, putând ajunge până la câteva sute de mii și chiar milioane. Un exemplu este familia Virtex-7 de la Xilinx care conține peste 2 milioane de celule.

În cadrul laboratorului se folosește placa de dezvoltare Spartan3E Starter Board de la Digilent, care este dotată cu un FPGA din familia Spartan-3E (XC3S500E) produs de către Xilinx. Acest FPGA oferă ~500.000 de porți logice organizate în ~10.000 de celule. Pe lângă cipul FPGA, placa de dezvoltare oferă și o serie de periferice conectate la acesta, cum ar fi:

- o memorie volatilă DDR-SDRAM de 64MB
- o memorie nevolatilă Flash paralelă de 16MB
- o memorie nevolatilă Flash serială de 16MB
- un display LCD cu 2 linii și 16 caractere

- butoane, switch-uri și LED-uri
- generator de ceas de 50MHz

Lista completă a perifericelor, precum și modul de interconectare și folosire a acestora împreună cu cipul FPGA pot fi consultate în [manualul de utilizare a plăcii](#).

## Sintetizarea folosind Xilinx ISE

Mediul de dezvoltare Xilinx ISE permite programarea cipurilor FPGA produse de către Xilinx. Versiunea WebPACK poate fi folosită pentru a sintetiza circuite și programa cipurile FPGA din gama low-cost ale acestui producător, cum ar fi familia Spartan.

În Xilinx ISE sintetizarea unui circuit se face selectând modul *Implementation* și este împărțită în două etape. În prima etapă, reprezentată de procesul *Synthesize*, este generată o descriere generică a circuitului ce urmează a fi implementat, sub forma unei liste de primitive și conexiunile dintre ele. În a doua etapă, reprezentată de procesul *Implement Design*, primitivele sunt mapate pe resursele oferite de cipul FPGA țintă, iar apoi conexiunile dintre ele sunt rutate prin matricea de interconectare din interiorul FPGA-ului.

Schema circuitului sintetizat de Xilinx ISE se poate vizualiza rulând procesul *Synthesize - XST→View RTL Schematic*. Selectând opțiunea *Start with a schematic of the top-level block*, inițial schema va conține doar un *black-box* pentru modulul *top-level*. Implementarea unui *black-box* poate fi expandată prin *dublu-click*.

Nu toate modulele pot fi expandate până la nivel de porți logice. În procesul de sintetizare, Xilinx ISE va infera anumite funcționalități precum: sumatoare (eng: *adders*), comparatoare (eng: *comparators*), multiplicatoare (eng: *multipliers*), numărătoare (eng: *counters*) etc. care nu mai pot fi expandate. Lista tuturor funcționalităților inferate se găsește în *Synthesis Report* disponibil în *Design Summary/Reports* din lista de procese.

La crearea proiectului Xilinx, trebuie selectat cipul FPGA țintă pentru care vor fi sintetizate circuitele. Pentru a putea programa placa de dezvoltare acesta trebuie să corespundă cu cipul FPGA de pe placă.

Pentru a putea testa un modul Verilog pe placa de dezvoltare, intrările și ieșirile acestuia trebuie rutate la pinii cipului FPGA. Acest lucru se face cu un fișier de constrângeri care asignează fiecărui bit dintr-un port al modulului un anumit pin al cipului. Alegerea pinilor care se conectează la porturile modulului Verilog se face în funcție de perifericele care vrem să producă intrările și să primească ieșirile modulului. Legăturile dintre pinii cipului FPGA și perifericele plăcii de dezvoltare se găsesc în [manualul de utilizare](#).

Fișierul de constrângeri trebuie creat înainte de sintetizarea circuitului. Urmăriți [tutorialul de asignare a pinilor de IO](#) pentru a vedea cum se creează acest fișier.

În final, programarea cipului FPGA se face cu fișierul de configurare generat pentru acesta (extensia .bit). Urmăriți [tutorialul de programare a FPGA-ului](#) pentru a vedea cum se realizează acest lucru.

Un singur modul, și dependențele acestuia, poate fi sintetizat la un moment dat. Acest modul se numește modulul *top-level* și este marcat cu un simbol special în lista de module ale proiectului. Fișierul de constrângeri va referi porturile acestui modul.

Modulul *top-level* poate fi modificat prin *click-dreapta* pe un alt modul și selectarea opțiunii *Set as Top Level Module*. În această situație, fișierul de constrângeri trebuie actualizat pentru a folosi porturile noului modul *top-level*.

## Exerciții

1. **(2p)** Urmăriți [tutorialul de debugging](#) și rezolvați bug-urile din sumator, apoi setați un *breakpoint* la linia 35 și rulați simularea până la a 5-a iterație (bitul 4) a celei de-a treia perechi de numere testate. ⚠ Implementarea sumatorului este una didactică, Verilog permițând o descriere mult mai simplă pentru un sumator pe oricâți biți cu operatorul +. ⚠
2. **(2p)** Testați pe placa de dezvoltare modulul *comp1*. Generați și vizualizați schema circuitului.
  - Hint: Funcționalitatea modulului *comp1* este explicată în [exercițiul 5](#) din laboratorul 1.
  - Hint: Scheletul de cod conține deja fișierul de constrângeri (*comp1.ucf*) și configurațiile necesare pentru programarea plăcii (*comp1.ipf*).
  - Hint: Rulați procesul *Configure Target Device* din modul *Implementation* pentru programarea plăcii. Placa este programată în momentul în care se aprinde LED-ul portocaliu.
  - Hint: Studiați fișierul de constrângeri și manualul de utilizare a plăcii de dezvoltare pentru a descoperi corespondența dintre porturile modulului *comp1* și perifericele plăcii.
3. **(4p)** Testați pe placa de dezvoltare modulul *simple\_alu*. Folosiți ca intrări pentru a switch-urile *SW0-SW3*, pentru b butoanele *BTN\_NORTH*, *BTN\_EAST*, *BTN\_SOUTH* și *BTN\_WEST*, iar pentru op butonul *ROT\_CENTER*. Afișați ieșirea modulului pe LED-urile *LED0-LED7*.
  - Hint: Porniți de la scheletul de cod.
  - Hint: Funcționalitatea modulului *simple\_alu* este explicată în [exercițiul 4](#) din laboratorul 2.
  - Hint: Urmăriți [tutorialul de asignare a pinilor de IO](#) pentru generarea fișierului de constrângeri. ⚠ Nu uitați configurarea de tip *PULLDOWN* a porturilor pentru butoane și switch-uri.
  - Hint: Urmăriți [tutorialul de programare a FPGA-ului](#) pentru configurațiile necesare programării plăcii.
4. **(4p)** Implementați și testați pe placa de dezvoltare un circuit care poate compara 2 numere pe câte 4 biți. Circuitul trebuie să aibă 3 ieșiri corespunzătoare situațiilor: *mai mic*, *egal*, *mai mare*.
  - Hint: Urmăriți [tutorialul de creare a unui proiect](#).
  - Hint: Folosiți o descriere comportamentală a circuitului.

## Resurse

- [Schelet de cod](#)
- [Soluție laborator](#) (disponibilă începând cu 19.10.2018)
- [PDF laborator](#)
- [Manualul plăcii de dezvoltare](#)

## Referințe

- [http://en.wikibooks.org/wiki/Programmable\\_Logic/FPGAs](http://en.wikibooks.org/wiki/Programmable_Logic/FPGAs)
- Datasheet-ul cipului XC3S500E (FPGA-ul folosit pe placa de laborator)

From:

<http://elf.cs.pub.ro/ac/wiki/> - **AC Wiki**

Permanent link:

<http://elf.cs.pub.ro/ac/wiki/lab/lab03>

Last update: **2018/10/11 16:35**

