UNIVERSITATEA "POLITEHNICA" BUCUREȘTI
FACULTATEA DE ELECTRONICĂ ȘI TELECOMUNICAȚII

Programarea FPGA-urilor folosind limbajul Handel-C

referat de doctorat

Prep. drd. ing. Răzvan Beuran

Coordonatori:
Prof. dr. ing. Vasile Buzuloiu
Prof. dr. ing. Jean-Marie Becker

- 2001 -

Conținut

1 Introducere

Noua tehnologie de implementare logică digitală, introdusă la mijlocul anilor '80, şi anume matricea de porţi programabilă, FPGA (*Field Programmable Gate Array*), a cunoscut o răspândire tot mai mare în ultimul timp. Avantajul principal pe care îl oferă FPGA-urile, precum şi celelalte componente ale familiei circuitelor logice programabile, cum ar fi PLA şi EEPROM, constă tocmai în programabilitatea lor. Aceasta implică posibilitatea de a configura un anumit circuit pentru o sarcină specifică, şi, în cele mai multe cazuri, chiar de a reprograma acelaşi circuit pentru o utilizare ulterioară diferită.

La începutul anilor '90 a apărut un nou tip de circuite logice programabile, numit DPGA (*Dynamically Programmable Gate Array*) [Tau-95], a căror funcţionalitate poate fi schimbată în intervale de ordinul milisecundelor, prin simplu control *software*. Aceste componente deschid cu adevărat calea conceptului de ***hardware* reconfigurabil**, adică de arhitecturi care se schimbă în timpul operării, pentru a sluji aplicaţia curentă într-un mod cât mai eficient cu putinţă.

La ora actuală, pentru programarea circuitelor logice se folosesc, în mod curent, limbaje de descriere a *hardware*-ului (*Hardware Description Language*, HDL), dintre care cele mai răspândite sunt VHDL şi Verilog. VHDL (VHSIC HDL — VHSIC însemnând *Very High Speed Integrated Circuits*) a devenit standard IEEE în anul 1986 şi se înrudeşte, din punct de vedere al sintaxei, cu limbajul de programare Ada, din care este inspirat de altfel şi limbajul Pascal. Verilog a fost standardizat de IEEE în 1995, fiind asemănător la nivel general cu limbajul C, ceea ce poate fi considerat drept un argument privind uşurinţa învăţării sale. Similaritatea este însă doar la nivel formal, portarea directă din C fiind rareori posibilă.

Cum o cerinţă importantă la ora actuală în industria *hardware* este scurtarea timpului între elaborarea unui algoritm şi implementarea sa, devine evidentă necesitatea utilizării unui limbaj care să fie cât mai apropiat de un limbaj de programare frecvent întrebuinţat, un candidat perfect fiind limbajul C. În acest fel devine posibilă realizarea sistemelor *hardware* de către persoane care nu au urmat lungi cursuri speciale de limbaje de descriere a *hardware*-ului. Un alt avantaj este facilitarea transformării unui algoritm scris în C într-o implementare *hardware*.

Un limbaj care răspunde acestor cerinţe este Handel-C, furnizat de compania *Celoxica* [Cel-**]. Acest limbaj este bazat pe limbajul Handel elaborat de Ian Page, profesor la *Imperial College* din Londra şi cofondator al *Celoxica*. La rândul său limbajul Handel este similar cu un subset al lui *occam* [Hoa-88], un limbaj folosit pentru descrierea sistemelor de

aplicații concurente. Utilizând limbajul Handel-C este așadar posibilă programarea FPGA-urilor într-un mod firesc, natural, cu rezultate satisfăcătoare din punctul de vedere al eficienței implementării.

Aplicațiile care pot beneficia de aportul sistemelor *hardware* reconfigurabile sunt numeroase, de la cele de tipul recunoașterea formelor, comunicația video, criptarea/decriptarea, până la sisteme de testare a rețelelor de calculatoare ș.a.m.d. În general vorbind, pentru orice domeniu în care sunt necesare operații în timp real, iar prelucrările *software* nu sunt suficient de rapide, o soluție practică extrem de eficientă este folosirea unor sisteme pe bază de FPGA-uri (sau DPGA-uri), programate cu un limbaj de nivel înalt, cum ar fi Handel-C. Aceasta paradigmă a migrării dinspre *software* spre *hardware* este foarte bine sintetizată de A. S. Tannenbaum în următoarea declarație:

> *"Designers with different goals may, and often do, make different decisions [...] the boundary between hardware and software is arbitrary and constantly changing. Today's software is tomorrows's hardware and viceversa."*[*]

Prezenta lucrare este structurată după cum urmează. În Capitolul 2 se vor prezenta tipurile de circuite logice programabile existente, cu accent pe FPGA-uri, precum și modalitățile uzuale de programare a acestora. Capitolul 3 va detalia structura limbajului Handel-C, iar în Capitolul 4 se va prezenta o aplicație care utilizează în mod intensiv acest limbaj. În Capitolul 5 se trec în revistă o serie de posibile aplicații ale conceptului de *hardware* reprogramabil în prelucrarea și analiza imaginilor. Lucrarea se încheie cu o serie de considerații finale și bibliografia folosită.

* Designeri cu scopuri diferite pot lua decizii diferite, și deseori o fac [...] frontiera între *hardware* și *software* este arbitrară și în continuă schimbare. *Software*-ul de astăzi este *hardware*-ul de mâine, și viceversa.

2 Circuite logice programabile

După cum s-a menționat și în introducere, circuitele logice programabile (în lb. engleză FPD — *Field Programmable Devices*), au căpătat o răspândire tot mai mare la ora actuală. Denumirea este folosită în prezent pentru toate circuitele integrate a căror funcționalitate poate fi modificată ulterior fabricației [Sea-97].

La începutul anilor 1995, termenul FPD se folosea doar pentru circuitele digitale, însă ulterior au fost incluse deasemenea circuitele analogice programabile (în lb. engleză FPAD — *Field Programmable Analog Devices*), precum și circuitele cu semnal mixt (în lb. engleză FPMSD — *Field Programmable Mixed-Signal Devices*).

Datorită evoluției caracteristicilor acestui tip de circuite, o tendință foarte puternică în prezent este aceea de a exploata la maxim caracterul de reprogramabilitate al acestor circuite, și în special a FPGA-urilor, prin includerea lor în sisteme reprogramabile [Hau-98].

În cele ce urmează se va face o trecere în revistă a tehnologiilor folosite, a principalelor tipuri de astfel de circuite, iar apoi se vor prezenta în detaliu FPGA-urile și posibilități ile care există în prezent pentru a programa astfel de circuite [Max-96].

2.1 Tehnologii de fabricație

Tehnologiile de fabricație se referă la metodele folosite pentru a creea componente programabile pentru FPD-uri. Cele mai utilizate tehnologii sunt:

- conexiunile fuzibile;
- conexiunile anti-fuzibile;
- celulele EPROM și EEPROM;
- celulele SRAM.

2.1.1 Conexiuni fuzibile

Conexiunile fuzibile sunt similare cu siguranţele casnice, în sensul că, prin aplicarea unei cantităţi de curent excesive, acestea îşi schimbă substanţial caracteristicile electrice. Există două tipuri de astfel de conexiuni: cu legătură laterală sau verticală.

Conexiunea laterală este compusă dintr-un fir de aliaj tungsten-titanium în serie cu un tranzistor cu joncţiune bipolară, care poate permite trecerea unui curent suficient de mare pentru topirea firului. Acest tip de conexiune este iniţial în scurt-circuit şi devine circuit deschis după programare.

Prin comparaţie, dioda între bază şi emitor a unui tranzistor cu joncţiune bipolară formează o conexiune verticală. Acest tip de legătură este iniţial în circuit deschis, deoarece tranzistorul se comportă ca două diode cuplate spate în spate, împiedicând astfel trecerea curentului. Dacă însă se forţează trecerea unui curent prin emitor, se produce un efect de avalanşă, emitorul se topeşte şi se crează un scurt-circuit.

Acest tip de conexiune este programabil o singură dată (în lb. engleză OTP — *One-Time Programmable*), însă conexiunile rămase neprogramate se pot bineînţeles modifica în continuare.

2.1.2 Conexiuni anti-fuzibile

Ca o alternativă, unele FPD-uri (în special circuitele complexe), folosesc tehnologia conexiunilor anti-fuzibile. Acestea sunt compuse dintr-un strat se siliciu amorf (necristalin) între două straturi metalice. În stare neprogramată, siliciul amorf este un izolator, cu rezistenţă la tensiuni de peste 1 GV, dar conexiunea poate fi programată prin aplicarea unui curent relativ mare (aproximativ 20 mA) la intrările componentei. Acest semnal duce la apariţia unei conexiuni prin transformarea siliciului amorf izolator în polisiliciu conductor.

Şi acest tip de conexiune este programabil o singură dată, datorită ireversibilităţii procesului implicat.

2.1.3 Celule EPROM şi EEPROM

Toate componentele conţinute de memoriile PROM (*Programmable Read Only Memory*), incluzând diodele, tranzistoarele şi elementele fuzibile, sunt create pe un singur substrat de siliciu. Conceptual, aceste elemente fuzibile, programabile o sigură dată, pot fi înlocuite cu tranzistori EPROM (*Electrically Programmable ROM*) sau EEPROM (*Electrically-Erasable Programmable ROM*), astfel încât circuitul obţinut devine reprogramabil. Aceste circuite prezintă anumite avantaje faţă de circuitele cu conexiuni

fuzibile sau anti-fuzibile, de exemplu ele pot fi testate în mod mai riguros după fabricație, printr-o suită de programări și ștergeri. Circuitele ce se pot programa direct în cadrul sistemului sunt de preferat, căci nu necesită scoaterea lor de pe placă și folosirea unor tehnici speciale pentru reprogramare, cum ar fi radiațiia ultravioletă.

2.1.4 Celule SRAM

SRAM este acronimul pentru *Static RAM*, RAM semnificând *Random Access Memory*. În această tehnologie, conexiunile programabile sunt constituite din tranzistori de trecere, porți de transmisie sau multiplexoare care sunt controlate de celule SRAM. Avantajul acestei arhitecturi este faptul că permite o reconfigurare rapidă în sistem, de aceea proiectele care implică o schimbare frecventă a funcționalității ii folosesc de preferință elemente SRAM. Dezavantajul principal ar fi dimensiunile mai mari implicate de tehnologia SRAM, ceea ce înseamnă că vor fi mai puține puncte de configurație decât în cazul celorlalte tehnologii.

2.2 Clasificarea arhitecturilor

În prezent sunt disponibile o serie de arhitecturi de logică programabilă, fiecare clasă majoră prezentând variante specifice diverşilor producători. O posibilitate de clasificare este următoarea [Pro-**]:

- circuite logice programabile simple;
- circuite logice programabile complexe;
- matrici de porţi programabile (FPGA-uri);
- interconexiuni programabile.

2.2.1 Circuite logice programabile simple

Circuitele logice programabile simple (în lb. engleză SPLD — *Simple Programmable Logic Device*) mai sunt cunoscute şi sub următoarele denumiri: PAL (*Programmable Array Logic*), GAL (*Generic Array Logic*, de la compania *Lattice*), PLA (*Programmable Logic Array*), PLD (*Programmable Logic Device*) .

Ele reprezintă cea mai mică, şi în consecinţă cea mai ieftină, formă de logică programabilă. Un astfel de circuit conţine între 4 şi 22 de macrocelule, complet interconectate între ele. Majoritatea circuitelor utilizează memorii non-volatile, de tipul EPROM, EEPROM sau Flash*, pentru definirea funcţionalităţii.

În mod tradiţional, SPLD-urile sunt bazate pe un şir de porţi AND (ŞI), ale căror ieşiri sunt conectate la un şir de porţi OR (SAU). Forma cea mai versatilă o constituie PLA-urile, în care utilizatorul controlează ambele matrici de intrare ale porţilor. Numărul de porţi AND este independent de numărul de intrări, iar numărul de porţi OR (şi în consecinţă de ieşiri) este independent de numărul de intrări şi de numărul de porţi AND.

Combinaţia de porţi AND şi OR nu este singura soluţie posibilă, unele circuite având două şiruri de porţi NAND (ŞI-NEGAT, NUMAI), sau două şiruri de porţi NOR (SAU-NEGAT, NICI), iar în unele cazuri un şir de porţi NAND ca intrare a unui şir de porţi NOR. Există chiar circuite bazate pe un singur şir de porţi ale cărui ieşiri se constituie în intrări (prin conexiune inversă), pentru a implementa expresii cu termeni de tipul "sume de produse"*.

Multe aplicaţii nu necesită ca ambele şiruri de porţi să fie programabile. De exemplu, în circuitele PAL, matricea porţilor AND este programabilă, iar cea a porţilor OR este

* O memorie de tip EEPROM, însă mai rapidă decât cipurile EEPROM clasice, deoarece datele sunt scrise în blocuri şi nu la nivel de octeţi.

* Termeni numiţi şi *maxtermi*, sau *termen canonic produs* din cadrul dezvoltării în forma canonică disjunctivă.

predefinită. PLA-urile sunt mai flexibile decât PAL-urile, însă acestea operează mai rapid, deoarece conexiunile cablate (în lb. engleză *hard-wired*) comută mai rapid decât echivalentele lor programabile. Datorită faptului că sunt rapide şi ieftine, PAL-urile sunt cel mai folosit tip de circuite SPLD.

Tot o formă de circuit simplu sunt PROM-urile, care pot fi văzute ca un şir predefinit de porţi AND la intrarea unui şir programabil de porţi OR. (În realitate, arhitectura internă PROM este mai asemănătoare cu un decodor ce constituie intrarea unui şir programabil de porţi OR.) Deşi PROM-urile sunt privite în general ca şi circuite de memorie, în care pentru o adresă drept intrare, la ieşire apare o valoare programată în circuit, ele pot fi interpretate ca şi circuite logice programabile clasice, fiind folosite pentru stocarea tabelelor de adevăr, sau implementarea funcţiilor cu un număr mare de *maxtermi*.

În plus faţă de funcţionalitatea de bază, SPLD-urile sunt disponibile şi cu o serie de opţiuni programabile, cum ar fi ieşiri cu trei stări sau prin regiştri. În acest al doilea caz, multe circuite permit utilizatorului alegerea tipului registrului, cum ar fi *latch*-uri D sau SR, ori bistabile D, T sau J-K. Deasemenea circuitele permit în general programarea pinilor externi drept intrări, ieşiri, sau conexiuni bidirecţionale.

2.2.2 Circuite logice programabile complexe

Circuitele logice programabile complexe (în lb. engleză CPLD — *Complex Programmable Logic Device*) mai poartă şi următoarele denumiri: EPLD (*Erasable Programmable Logic Device*), EEPLD (*Electrically-Erasable Programmable Logic Device*), MAX (*Multiple Array matriX*, de la *Altera*).

Aceste circuite sunt similare cu cele precedente, dar au o capacitate semnificativ mai mare, un circuit complex tipic fiind echivalent cu 2 până la 64 de circuite simple. Un CPLD conţine zeci sau sute de macrocelule, grupuri de 4 până la 16 dintre acestea constituind un bloc funcţional. Macrocelulele din acelaşi bloc funcţional sunt complet interconectate între ele, iar dacă un circuit conţine mai multe blocuri funcţionale, acestea sunt la rândul lor interconectate, însă nu neapărat complet, această chestiune depinzând de producător. O consecinţă a acestui fapt este eventuala imposibilitate de rutare şi probleme cu repartiţia pinilor între versiuni diferite de proiectare.

CPLD-urile au o arhitectură asemănătoare cu SPLD-urile, de aceea tranziţia este uşoară pentru proiectanţi. În plus majoritatea ciruitelor complexe suportă limbajele de dezvoltare ale circuitelor simple, cum ar fi ABEL, CuPL, PALASM etc. CPLD-urile sunt cele mai potrivite pentru proiectări orientate pe control, datorită vitezei mari de lucru, iar

interconectivitatea mare le face adaptate pentru automate (în lb. engleză *state machines*) de înaltă performanță.

Unele din variațiile majore între arhitecturile de CPLD-uri includ: numărul de termeni produs per macrocelulă, dacă aceștia pot fi "împrumutați" între macrocelule, dacă matricea de interconectare este complet sau doar parțial populată. "Împrumutarea" termenilor produs menționată mai sus, ce este posibilă în unele arhitecturi, lărgește gama de aplicații. De remarcat faptul că, în anumite cazuri, macrocelule de la care s-a "împrumutat" devin nefuncționale, în timp ce în alte arhitecturi ele își păstrează o funcționalitate de bază. Trebuie însă subliniat că, prin astfel de "împrumuturi", timpul de propagare de obicei crește.

O altă diferență între arhitecturi este dată de numărul de conexiuni în matricea de comutare. Dacă toate variantele sunt posibile, matricea este numită complet populată, în caz contrar — parțial populată. Numărul de conexiuni determină ușurința cu care se face plasarea componentelor (fitare/mapare — în lb. engleză *fitting/mapping*) și crearea interconexiunilor (rutare — în lb. engleză *routing*). Astfel, în cazul unei matrici de interconectare populată complet, plasarea se face cu ușurință, chiar dacă majoritatea resurselor sunt folosite, iar întârzierile sunt fixe și predictibile.

Un circuit cu o matrice de interconectare parțial populată poate cauza dificultăți de rutare. Schimbările de proiectare sunt mai dificile, și poate fi necesară modificarea repartiției pinilor, ceea ce constituie un inconvenient major, căci este mult mai ușor să se schimbe structura internă a unui circuit programabil, decât să se reproiecteze întreaga placă. Întârzierile pentru acest fel de matrice nu sunt fixe și mai greu de prezis. Folosirea unor astfel de circuite, în ciuda limitărilor amintite, este bineînțeles dictată de costul lor mai mic.

CPLD-urile sunt circuite CMOS (*Complementary Metal Oxide Semiconductor*), folosind una dintre tehnologiile EPROM, EEPROM sau Flash pentru definirea funcționalității. Circuitele bazate pe EPROM sunt de obicei programabile o singură dată, cu excepția cazului când sunt incluse într-o capsulă cu o fereastră pentru programarea prin radiație ultravioletă. Programarea se face de către producător sau distribuitor.

Multe din familiile de circuite recente folosesc însă tehnologiile EEPROM sau Flash și au fost proiectate pentru a putea fi programate direct în circuit. Dintre producători, doar *Atmel* și *Philips* mai produc doar CPLD-uri de primul tip, restul companiilor (*Altera*, *Cypress*, *Lattice*, *Vantis*, *Xilinx*) producând circuite de ambele tipuri.

Unele CPLD-uri bazate pe conexiuni programabile de tip SRAM au o flexibilitate mai mare prin faptul că permit folosirea blocurilor individuale de SRAM fie drept conexiuni programabile, fie ca blocuri de memorie propriu-zise. Conexiunile programabile, ce pot

conține mai mult de 100 de trasee, sunt interfațate cu blocurile SPLD prin intermediul unui multiplexor programabil.

După cum s-a amintit deja, unul din avantajele principale ale CPLD-urilor este acela că structura lor regulată permite estimarea rezonabilă a întârzierilor. Piața acestor circuite a crescut considerabil în ultimii ani, ele găsindu-și întrebuințări în multe aplicații comerciale, încluzând regândirea unor proiecte bazate pe SPLD-uri, astfel încât implementarea să necesite mai puține circuite.

2.2.3 Matrici de porți programabile (FPGA-uri)

FPGA-urile (*Field Programmable Gate Array*) mai sunt cunoscute sub următoarele denumiri: LCA (*Logic Cell Array*), pASIC (*programmable ASIC*), FLEX, APEX (de la *Altera*), ACT (de la *Actel*), ORCA (de la *Lucent*), Virtex (de la *Xilinx*).

FPGA-urile diferă de SPLD-uri și CPLD-uri și, în mod tipic, oferă cea mai mare capacitate de logică. Un astfel de circuit constă într-o matrice de blocuri logice, înconjurată de blocuri de intrare/ieșire programabile și legate prin interconexiuni care sunt la rândul lor programabile (vezi secțiunea 2.3 pentru o tratare mai detaliată).

2.2.4 Interconexiuni programabile

Interconexiunile programabile (în lb. engleză FPIC — *Field Programmable Interconnect Device*) nu sunt propriu-zis circuite logice, ci mai degrabă circuite programabile de interconectare. Prin programare se stabileşte o conexiune între un pin al circuitului şi un altul, deci se realizează respectiva interconectare.

FPIC-urile folosesc fie tehnologia SRAM, fie pe cea anti-fuzibilă. Compania *I-Cube* oferă astfel de circuite ca şi componente de sine stătătoare, în timp ce *Aptix* le comercializează ca părţi ale sistemelor lor de emulare *hardware*.

2.3 FPGA-urile

Circuitele SPLD şi CPLD sunt utile într-o varietate de scopuri, dar structura lor, bazată pe şiruri de porţi programabile AND şi OR este totuşi o limitare. La celălalt capăt al spectrului se află ASIC-urile (*Application-Specific Integrated Circuit*), care includ matrici de porţi, elemente standard şi circuite adaptate cerinţelor. Acestea sunt foarte generice, au o arhitectură de granularitate fină (la nivel de porţi de bază şi regiştri) şi au capacităţi de 800.000 de porţi sau chiar mai mult. Pe de altă parte, aceste circuite necesită costuri mari de punere în producţie şi perioade lungi de proiectare. Deci între SPLD-uri/CPLD-uri, la un capăt, şi ASIC-uri, la capătul celălalt al scării, se găsea un gol.

2.3.1 Caracteristici de bază

Spre sfârşitul anilor '80, un nou tip de circuite şi-a făcut apariţia pentru a umple acest gol, şi anume FPGA-urile (*Field Programmable Gate Array*). Aceste circuite combinau multe din caracteristicile matricilor de porţi, cum ar fi densitatea mare, cu cele ale circuitelor programabile timpurii, în principal programabilitatea. Un factor diferenţiator este acela că majoritatea FPGA-urilor are o granularitate mare, ceea ce înseamnă că acestea constau într-o matrice de blocuri logice, înconjurată de blocuri de intrare/ieşire programabile şi legate prin interconexiuni care sunt la rândul lor programabile (vezi figura 1).

Un FPGA tipic conţine între 64 şi zeci de mii de blocuri logice, şi un număr mai mare de circuite bistabile. Majoritatea FPGA-urilor nu asigură o interconectare completă între blocurile logice, ceea ce ar fi prohibitiv din punctul de vedere al costurilor. În schimb, pachete sofisticate de *software* sunt folosite pentru maparea elementelor logice pe blocurile funcţionale şi crearea interconexiunilor înter acestea.

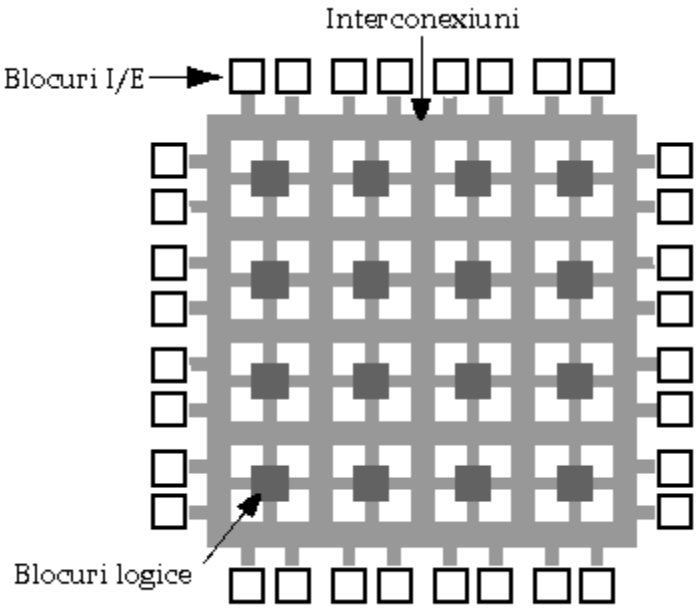


Figura 1: Structura de principiu a unui FPGA.

2.3.2 Tipuri de arhitecturi

Există o serie de sub-arhitecturi care se încadrează în modelul general prezentat anterior. Secretul densității ii și performanțelor înalte despre care s-a vorbit rezidă în funcțiile logice ale blocurilor logice prezente, precum și în eficiența arhitecturii de interconectare.

Tehnologia de bază o constituie în general conexiunile anti-fuzibile sau controlate de celule SRAM, însă există două clase principale de arhitecturi FPGA, și anume :

- arhitecturi cu granulație mare;
- arhitecturi cu granulație fină.

Arhitecturile cu granulație mare se bazează pe blocuri logice relativ mari, conținând adeseori două sau mai multe tabele de căutare și două sau mai multe bistabile. Principalele două abordări pentru circuitele cu granularitate mare sunt:

- tabelele de căutare (în lb. engleză *look-up tables*, LUT);
- multiplexoarele.

Presupunându-se un tabel cu 3 intrări, o tehnică implică folosirea unui decodor 3-la-8 pentru a selecta una din cele 8 celule SRAM care conține valoarea de ieșire dorită din tabela de adevăr a funcției logice. Ca o alternativă se pot folosi celule SRAM pentru a controla o structură piramidală de multiplexoare și a genera ieșirea. În alte cazuri se folosesc arhitecturi bazate aproape în întregime pe multiplexoare.

FPGA-urile cu granularitate mare prezintă însă o serie de probleme. În primul rând,

întârzierile nu sunt la fel de predictibile ca în cazul SPLD-urilor şi CPLD-urilor. În al doilea rând, toţi producătorii utilizează programe speciale pentru a plasa schemele de principiu în circuitele lor, ceea ce face aproape imposibilă migrarea de la un producător la altul şi menţinerea aceloraşi întârzieri de propagare. În al treilea rând, majoritatea instrumentelor de sinteză este orientată spre arhitecturi ASIC cu granularitate fină. Acestea produc liste de componente şi conexiuni (în lb. engleză *netlist*) la nivel de porţi logice, iar programele dedicate FPGA-urilor nu reuşesc în general o mapare optimă a acestora pe un circuit. Astfel, în plus faţă de utilizarea relativ slabă a resurselor, devine dificilă o estimare realistă a întârzierilor de propagare înainte de rutare, ceea ce înseamnă ca uneori sunt necesare intervenţii pe nivelele de jos ale ciclului de proiectare pentru a face sistemul funcţional.

Totuşi unele tehnici sunt adaptate în particular arhitecturilor FPGA cu granularitate mare, cum ar fi, de exemplu, biblioteca de module parametrizate EDIF (*Electronic Design Interoperability Format*). Deasemenea se observă o tendinţă în creştere către utilizarea arhitecturilor FPGA cu granularitate fină, cum ar fi circuitele pASIC-2 ale companiei *QuickLogic*, ale căror celule se pot comporta în ambele moduri (cu granularitate mare sau granularitate fină), sau arhitectura CrossFire a *Crosspoint Solutions*, care foloseşte blocuri de bază ultrafine "jumătate de poartă logică".

Celălalt tip de arhitectură este aşadar cel cu granulaţie fină. Aceste circuite conţin un număr mare de blocuri logice relativ simple. Un astfel de bloc conţine, de obicei, fie o funcţie logică de două variabile, fie un multiplexor 4-la-1 şi un bistabil. Aceste circuite prezintă avantaje clare pentru proiectele create prin sinteză logică, în care celulele de bază sunt foarte simple, uneori chiar la nivelul porţilor logice.

2.3.3 Tehnologii de fabricaţie

O altă diferenţă o constituie tehnologia folosită pentru fabricarea circuitelor. La ora actuală, FPGA-urile cu cea mai mare densitate sunt produse folosind tehnologia SRAM (*Static Random Access Memory*). Un al doilea proces tehnologic uzitat este cel numit anti-fuzibil, care are avantajul unei interconectibilităţi sporite.

Circuitele bazate pe SRAM sunt în mod inerent reprogramabile, chiar în cadrul sistemului, dar necesită o sursă externă de memorie de configuraţie. Aceasta conţine programul care defineşte modul în care funcţionează blocurile logice, care blocuri I/E sunt intrări şi care sunt ieşiri, modul în care blocurile sunt interconectate. FPGA-ul respectiv, fie îşi auto-încarcă configuraţia dintr-o memorie, fie un procesor extern descarcă configuraţia în el. Când se auto-încarcă, FPGA-ul adresează o memorie PROM standard asemeni unui procesor la iniţializare, sau foloseşte un tip special de PROM, cu acces secvenţial. În a doua

variantă FPGA-ul este oarecum echivalent cu un periferic de microprocesor standard. Configurația durează, în mod uzual, mai puțin de 200ms, depinzând de dimensiunea circuitului și de metoda de configurație.

În contrast cu circuitele descrise mai sus, în cazul celor anti-fuzibile programarea se face o singură dată, și funcționalitatea nu mai poate fi modificată, rămânând valabilă chiar și după întreruperea alimentării. Aceste circuite sunt în consecință programate doar de către fabricant sau distribuitor.

Unele FPGA-uri au integrate anumite caracteristici de nivel mai înalt, cum ar fi magistrale, memorii pentru mici blocuri de regiștri sau cozi, suport pentru JTAG*, ceea ce face ca ele să fie preferate pentru schemele mai complexe.

2.3.4 Programarea FPGA-urilor

Programarea convențională a FPGA-urilor poate fi descrisă prin următoarea succesiune de pași (vezi și figura 2):

1) etapa de proiectare;
2) etapa de mapare/rutare;
3) etapa de verificare/simulare;
4) eventual modificarea sau corectarea circuitului.

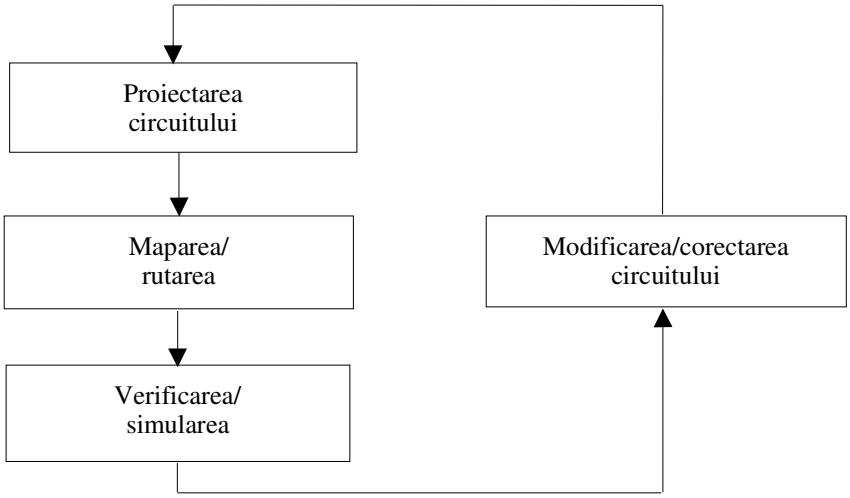


Figura 2: Ciclul de proiectare convențional pentru un FPGA.

În etapa de proiectare, schema digitală este creată cu ajutor unui editor sau a unui limbaj de descriere a *hardware*-ului. Un editor de scheme permite folosirea de simboluri

* Cadru pentru testarea componentelor logice electronice definit de *Joint Test Action Group,* standardul IEEE 1149.1.

grafice pentru precizarea componentelor și a circuitelor. Limbajele de tip HDL folosesc un limbaj descriptiv, cum ar fi Verilog, ABEL sau VHDL [Toa-96], în același scop. Orice variantă s-ar alege trebuie verificat că programul respectiv are încorporată biblioteca specifică FPGA-ului ce va fi folosit, deoarece ele trebuie să poată produce și mapa *netlist*-ul asociat proiectului.

Etapa de mapare/rutare realizează convertirea *netlist*-ului produs în faza anterioară într-un fișier binar de configurare a FPGA-ului. Cei trei pași incluși în această etapă sunt:

a) maparea schemei pe resursele circuitului;
b) asignarea blocurilor logice, create anterior prin mapare, unor locații specifice din circuit;
c) interconectarea blocurilor logice.

În final se crează o descriere exactă a configurației FPGA-ului (*Logic Cell Array File*, LCA), care este convertită într-o reprezentare binară.

Această reprezentare binară este ulterior folosită în etapa de verificare, care are drept scop testarea proiectului din punct de vedere logic şi al sincronizării/temporizării. O serie dintre aceste teste se poate efectua cu instrumentele de verificare/simulare, care permit o caracterizare detaliată a proiectului, prin efectuarea de simulări de funcţionalitate şi sincronizare. Un alt mod de testare este chiar verificarea sub tensiune, prin crearea condiţiilor normale de operare.

Uneori, după etapa de mai sus se pot constata inadecvări între funcţionalitatea circuitului şi cea dorită. În acest caz se impune reluarea ciclului de proiectare de la primul pas.

Configurarea este procesul prin care fişierul binar produs la pasul 2 este propriu-zis încărcat în FPGA. Circuitele pot fi configurate prin folosirea unui PROM, cel serial fiind cel mai folosit, dar existând şi variante paralele la nivel de octet. În acest caz FPGA-ul îşi citeşte în mod activ configuraţia din PROM-ul respectiv. O a doua posibilitate este ca datele să fie înscrise în FPGA dintr-o sursă exterioară (de obicei un PC), prin intermediul unei interfeţe, cum ar fi JTAG, ce se conectează la portul paralel al unui PC printr-un cably ByteBlaster.

Există deasemenea şi cazul în care circuitul este folosit pe o platformă reconfigurabilă, pentru care configuraţia este integrată într-o funcţie a unui limbaj de nivel înalt, de obicei C; astfel devine posibilă configurarea din cadrul unei aplicaţii.

De curând au devenit disponibile noi instrumente de proiectare a FPGA-urilor, care se bazează pe limbaje derivate din C (a se vedea capitolul 3 pentru o exemplificare detaliată a unui produs de acest tip, şi anume limbajul Handel-C). Acestea oferă în principiu o uşurinţă mai mare în programare şi permit de obicei efectuarea de simulări înainte de mapare/rutare.

3 Limbajul Handel-C

Handel-C este un limbaj de programare ce a fost proiectat pentru a permite transformarea programelor în implementări *hardware* digitale. Totuşi Handel-C nu este un limbaj de descriere a *hardware*-ului, ci mai degrabă un limbaj de programare ce transformă, prin compilare, algoritmi de nivel înalt în proiecte *hardware* la nivel de porţi logice.

În cele ce urmează se va discuta despre versiunea 2.0 a limbajului, care a fost folosită pentru implementarea plăcii de test descrisă în capitolul 4. Ultima versiune, şi anume 3.0, adaugă o serie de facilităţi limbajului, uşurând şi mai mult portarea din C.

3.1 Concepte de bază

Sintaxa limbajului Handel-C se bazează pe aceea a limbajului C convenţional [Ker-88], deci programatorii familiarizaţi cu acesta recunosc cu uşurinţă majoritatea elementelor sintactice. Limbajul permite aşadar exprimarea unui algoritm fără a ţine prea mult seama de modul în care se efectuează calculele. Într-un anumit sens Handel-C este pentru *hardware* ceea ce un limbaj de nivel înalt convenţional este pentru limbajul de asamblare al microprocesorului.

Programe secvenţiale pot fi scrise în Handel-C la fel ca şi în C, dar, pentru a beneficia cât mai mult de implementarea *hardware*, trebuie exploatat paralelismul inerent al acestuia. De aceea Handel-C conţine construcţii paralele, acesta fiind una din cele mai importante diferenţe faţă de C-ul convenţional.

3.1.1 Fluxul de execuţie

Limbajul Handel-C este aşadar în principal secvenţial, ordinea a două instrucţiuni deteminând ordinea execuţiei. Ca în oricare limbaj convenţional există instrucţiuni ce controlează cursul programului. De pildă o porţiune de cod poate fi executată în mod condiţionat, în funcţie de valoarea unei expresii, sau poate fi repetată de un număr de ori folosind instrucţiuni de ciclare. Trebuie precizat faptul că schema *hardware* produsă de Handel-C este aceeaşi cu cea specificată prin codul sursă; nu există nici un nivel intermediar de interpretare cum se întâmplă în cazul limbajelor de asamblare. Porţile logice constituie "instrucţiunile" de asamblare ale limbajului Handel-C.

Datorită faptului că prin compilare se produce un *netlist*, o creştere importantă a performanţelor se poate obţine prin utilizarea paralelismului. Este deci posibil în Handel-C (şi esenţial, din punctul de vedere al eficienţei execuţiei) a instrui compilatorul să genereze *hardware* care execută instrucţiuni în paralel.

Paralelismul în Handel-C este veritabil, nu ca cel simulat întâlnit în computerele de uz general. Cu alte cuvinte, două instrucţiuni pentru care se dispune a fi executate în paralel vor fi executate în exact acelaşi tact de ceas de către două porţiuni de *hardware* diferite.

Când se ajunge la un bloc paralel fluxul de execuţie se ramifică la începutul blocului şi ramurile sunt executate simultan. Fluxul de execuţie este apoi reluat la sfârşitul blocului, când toate ramurile au terminat execuţia. Ramurile care termină mai devreme sunt forţate să o aştepte pe cea mai înceată dintre ele, înainte de continuarea execuţiei.

3.1.2 Comunicația și accesul la variabile

Pentru comunicația între două ramuri ce se execută în paralel se folosesc canale (acestea fiind principalul element provenind din *occam*). O ramură paralelă trimite date printr-un canal și cealaltă ramură le recepționează. Canalele servesc deasemenea la sincronizarea între ramuri, căci transferul nu poate avea loc decât atunci când ambele capete ale canalului de comunicație sunt disponibile. Dacă transmițătorul nu este pregătit pentru comunicație, atunci receptorul trebuie să aștepte și viceversa.

Vizibilitatea declarațiilor de variabile este bazată pe blocuri de cod separate de acolade, ca și în limbajul C. Deoarece construcțiile paralele sunt și ele simple blocuri de cod, o variabilă ar putea fi utilizată în mod legal în două ramuri ce se execută în paralel. Aceasta poate duce însă la conflicte serioase dacă accesul se face simultan. De aceea nu se recomandă accesul aceleiași variabile în două ramuri paralele. În practică regula ce trebuie respectată este de a nu asigna o valoare unei variabile, în două ramuri paralele, în același tact de ceas; există însă posibilitatea de a citi valoarea unei variabile, ceea ce duce uneori la eficientizarea unor porțiuni de cod.

3.1.3 Ciclul de proiectare

În figura 3 se poate observa secvența operațiilor implicate de programarea în Handel-C. Se observă că simulările se efectuează cu ajutorul instrumentelor Handel-C, iar implementarea în *hardware* este ulterioară. Astfel accentul se pune pe efectuarea eventualelor corecții încă din faza "*software*" a proiectului, deci înainte de implementarea *hardware*. Motivul principal este că operațiile asociate cu aceasta din urmă (mapare, rutare) durează în general mult mai mult timp decât simpla compilare.

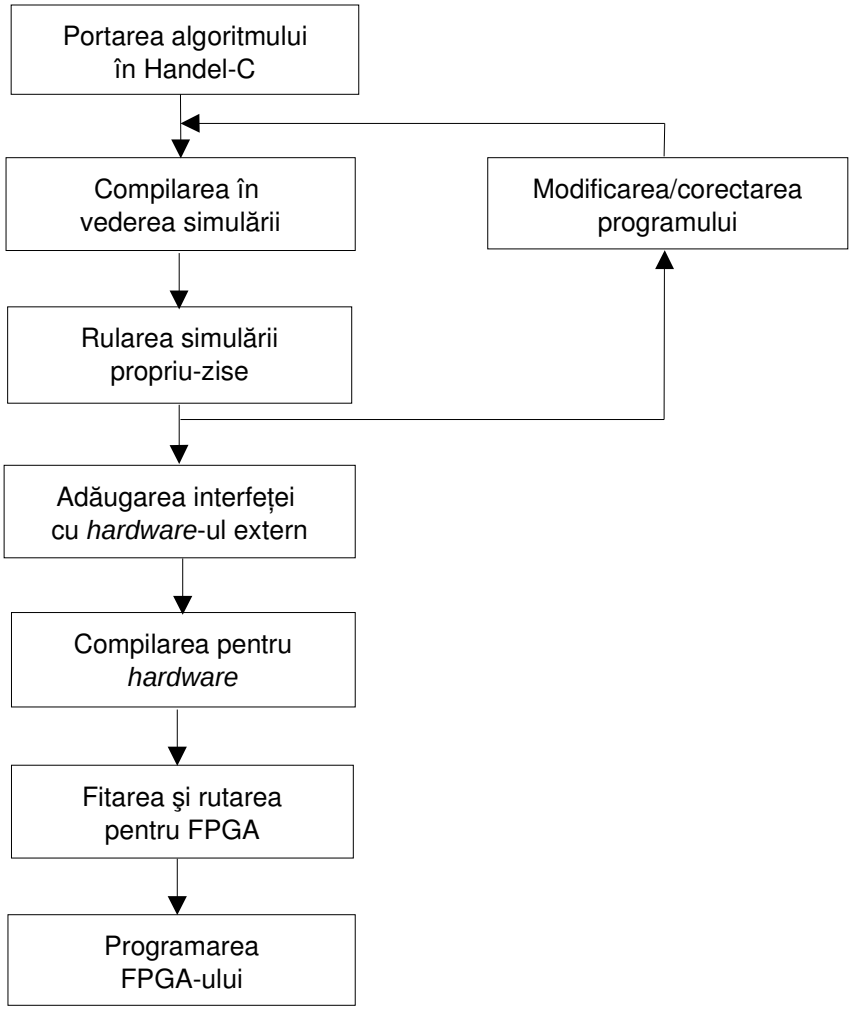


Figura 3: Detalierea ciclului de proiectare în Handel-C.

Un alt avantaj constă în faptul că în prima etapă se poate pleca de la un algoritm care este deja implementat în C şi verificat, fiind necesară doar portarea sa, care este relativ facilă.

3.2 Sintaxa limbajului

După cum s-a amintit anterior, sintaxa limbajului Handel-C este foarte asemănătoare cu cea a limbajului C. În continuare se vor prezenta succint elementele de bază ale sintaxei, cu sublinierea diferențelor care există față de limbajul C.

3.2.1 Structura unui program

Structura generală a unui program este următoarea:

```
Declaraţii globale
void main(void)
{
        Declaraţii locale
        Bloc de instrucţiuni
}
```

Deasemenea se pot folosi în cadrul codului directive pentru preprocesor, precum și comentarii în stilul C standard (`/* ... */`) sau în stilul C++ (`//`).

O diferență crucială între C și Handel-C este faptul că acesta din urmă permite manevrarea variabilelor de orice dimensiune. Biții din variabile pot fi extrași cu ușurință și alăturați pentru a forma valori de dimensiune mai mare. Tipul de bază este `int`, dar sunt definite și o serie de alte tipuri (`char`, `short`, `long`), pentru a ușura portarea din C.

3.2.2 Elemente de bază

O diferență importantă în Handel-C, în comparație cu limbajul C, este inexistența pointerilor, care se datorează bineînțeles lipsei unei memorii globale.

Tablourile de variabile pot fi declarate în același mod ca și în limbajul C. O restricție importantă în versiunea 2.0 este aceea că indicii trebuie să fie constante la compilare (această restricție dispare .

În cadrul programului se mai pot defini structuri RAM sau ROM interne. Avantajele față de folosirea tablourilor este că indicii pot fi expresii variabile și ca necesită mai puține resurse. Dezavantajul principal este că doar o singură locație poate fi accesată într-un tact de ceas.

Pentru comunicație se folosesc canale sau chiar tablouri de canale. Citirea dintr-un canal se face printr-o instrucțiune de forma:

```
Canal ? Variabilă;
```

iar scrierea prin:

```
Canal ! Expresie;
```

Pentru ca un bloc de instrucțiuni să fie executat în paralel, acesta trebuie precedat de cuvântul cheie `par`:

```
par
{
        Bloc de instrucţiuni
}
```

Instrucțiunile de atribuire sunt singurele care sunt considerate ca având o durată temporală în Handel-C, făcând astfel calculul sincronizărilor mai simplu. Se pot regăsi în limbaj și instrucțiunile de execuție condiționată sau ciclare uzuale în C: `if-else`, `switch`, `while`, `do-while`, `for`. Există deasemenea și instrucțiunea `break`.

O instrucțiune nouă este `delay`, care nu are nici un efect, ci doar cauzează o întârziere de un tact de ceas. Ea se folosește pentru a preveni conflictele asupra resurselor sau pentru a ajusta sincronizarea proceselor în timpul execuției.

Este definită o serie întreagă de operatori, de la cei de manipulare a biților (dintre care unii sunt specifici Handel-C, cum ar fi cei de selecție a biților), până la operatori aritmetici, relaționali sau operatorul condițional.

Ca și în limbajul C, Handel-C permite folosirea de *macro*-uri, fie ele expresii constante sau parametrizate. Prin folosirea cuvântului cheie `shared` în locul celui de `macro` se instruiește compilatorul să refolosească expresia în toate locurile unde apare, efectuând astfel o economie de resurse. Prin folosirea combinației `macro proc` se definește o procedură, evitându-se în acest fel rescrierea de cod; în acest caz însă pentru fiecare "apel" al procedurii se crează un nou bloc *hardware*.

Sincronizarea și durata de execuție a unui bloc de instrucțiuni sunt elemente esențiale pentru a asigura corectitudinea execuției în cazul programelor ce conțin multiple procese paralele ce interacționează între ele. După cum s-a amintit și anterior, atribuirea și instrucțiunea `delay` durează un tact de ceas, celelalte instrucțiuni fiind considerate instantanee. Ca o consecință, un ciclu care nu conține nici o instrucțiune cu "durată" este

considerat incorect, căci nu i se poate evalua timpul de execuție, și poartă denumirea de *ciclu combinatorial*. Un mod de a corecta o astfel de eroare este introducerea unei instrucțiuni de întârziere.

3.2.3 Un exemplu

În continuare este prezentat un exemplu foarte simplu, al cărui rol este de a pune în evidență sintaxa limbajului. Programul de mai jos preia valori diferite de '0' pe canalul de intrare și le însumează. La introducerea valorii '0' ciclul se încheie și suma este transmisă pe canalul de ieșire:

```
void main(void)
{
        unsigned int 16 sum;
        unsigned int 8 data;
        chanin input;
        chanout output;

        sum = 0;
        do
        {
              input ? data;
              sum = sum + (0 @ data);
        } while (data!=0);

        output ! sum;
}
```

Iată în continuare o prezentare succintă a codului. Programul definește două variabile: `sum` pentru a stoca suma intrărilor și `data` pentru a stoca temporar fiecare intrare; ele sunt întregi fără semn pe 16, respectiv 8 biți. Se definesc apoi cele două canale, unul de intrare (`input`) și unul de ieșire (`output`), folosind cuvintele cheie `chanin` și `chanout`, care realizează legătura cu intrarea și ieșirea standard a simulatorului.

Suma se inițializează la 0. Se intră apoi într-un ciclu în care, la fiecare iterație, se citește un întreg pe canalul de intrare care se adună la sumă. A se observa operatorul '@' care realizează prefixarea datelor cu 0, astfel încât ambii membri ai sumei să aibă același număr de biți. De remarcat că lungimea prefixului 0, și anume 8 biți, este calculată în mod automat de compilator.

Ieșirea din ciclu are loc când datele de la intrare au valoarea '0'. Suma este trimisă pe canalul de ieșire și apoi se încheie execuția.

3.2.4 Eficiența programelor

Eficiența unui FPGA programat în Handel-C depinde de eficiența programului

propriu-zis. Foarte importantă în acest context este frecvența/perioada ceasului folosit în sistem; cu cât frecvența este mai mare (respectiv perioada este mai mică), cu atât sistemul este mai eficient. Perioada ceasului sistemului trebuie să fie mai mare decât durata căii celei mai lungi prin logica combinațională a programului. Dacă perioada nu este suficient de mică pentru a asigura funcționarea sistemului la frecvența dorită se impune efectuarea de optimizări.

Desigur transformarea codului în Handel-C într-o configurație hardware, care este un proces automat nu asigură optimalitatea implementării algoritmului. Se pare însă că acest aspect este satisfăcător (și oricum este în afara controlului utilizatoruil), așa că ne vom ocupa doar de optimizările pe care acesta le poate întreprinde la nivelul codului.

O primă cale de optimizare este reducerea adâncimii logicii. Iată câteva soluții pentru aceasta:

1) evitarea utilizării operatorului de înmulțire, care implică folosirea unei importante cantități i de resurse logice; multe astfel de operații se pot efectua prin șiftare, sau o combinație de ciclare, șiftare și adunare;
2) reducerea circuitelor de adunare pe mai mulți biți la mai multe circuite similare pe mai puțini biți;
3) evitarea utilizării comparațiilor de tipul ≤ sau ≥ prin înlocuirea lor cu operatorii == sau !=, atunci când aceasta este posibil;
4) reducerea operațiilor complexe prin folosirea mai multor etape executate secvențial;
5) evitarea șirurilor lungi de instrucțiuni vide, ce apar de pildă în secvențe de `if` fără `else`.

Cea de-a doua cale de optimizare este utilizarea principiului benzii de asamblare, prin împărțirea operațiilor complexe în etape ce se execută în paralel. Un astfel de sistem, deși calculează o valoare în mai mult de un tact, după câteva tacte inițiale produce câte un rezultat pe tact.

3.2.5 Interfațarea cu exteriorul

Interfațarea cu simulatorul se realizează prin intermediul canalelor, și poate fi interactivă cu utilizatorul, dacă se folosesc canalele standard de intrare/ieșire, sau poate fi bazată pe fișiere în care utilizatorul stochează datele de intrare și în care simulatorul stochează datele de ieșire.

Pentru interfațarea codului, și deci a FPGA-ului, cu alte componente *hardware*, acestea trebuie declarate prin definirea unor interfețe. Componentele cu care FPGA-ul poate fi conectat prin intermediul Handel-C sunt diverse, cum ar fi module RAM externe și diverse tipuri de magistrale. Definirea acestora se face prin folosirea cuvântului cheie `interface`.

4 Studiu de caz: testarea rețelelor de calculatoare

ATLAS este unul dintre cele patru experimente ce sunt în curs de implementare la CERN pentru noul accelerator de particule LHC (*Large Hadron Collider*). În cadrul acestui experiment, ca și pentru scopuri generale de testare a rețelelor, s-a construit aici o placă de testare a rețelelor de calulatoare, numită Enet32 [Bar-01]. Ea implementează 32 de porturi Ethernet *full-duplex* a 100Mb/s folosind în principal FPGA-uri *Altera Flex 10K*, atât pentru implementarea controlerelor de acess la mediu MAC (*Media Access Controller*) cât și pentru restul componentelor care contribuie la realizarea funcționalității ce va fi descrisă în continuare.

Din punctul de vedere al prezentei lucrări acest sistem este edificator deoarece toate FPGA-urile au fost programate integral în Handel-C. Aceasta a permis o mare flexibilitate privind controlul asupra comportamentului plăcii, ea putând răspunde unei game foarte largi de cerințe, de la testarea generală rețelelor și *switch*-urilor [Tan-97] sau modelarea acestora, până la generarea unui trafic de rețea identic cu cel ce se va întâlni în cadrul experimentului ATLAS [Dob-01].

4.1 Arhitectura plăcii Enet32

Arhitectura plăcii este prezentată în figura 4. Conexiunea la PC-ul de control este efectuată printr-un port paralell IEEE 1284. Controlul asupra conexiunii este deținut de un FPGA, numit *I/O Manager* (IoMan), care trimite datele și comenzile dinspre PC spre celelalte componente ale plăcii, ca și datele dinspre acestea spre PC.

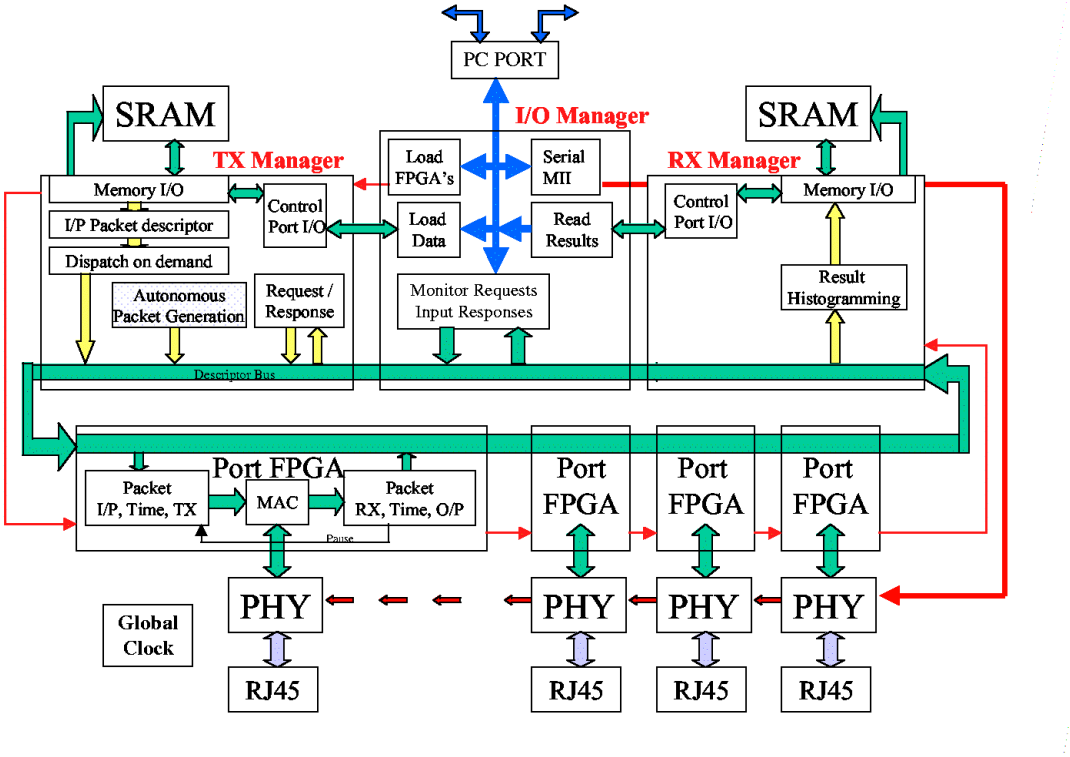


Figura 4: Arhitectura plăcii de test Enet32

Cele 32 de MAC-uri, IoMan precum și alte două FPGA-uri: managerul de transmisie a pachetelor (TxMan) și mangerul de recepție a pachetelor (RxMan), sunt conectate printr-o magistrală de tip inel, ceasul sistemului fiind de 25MHz*. Magistrala are o lărgime de 52 de biți, dintre care 32 sunt utilizați pentru transmiterea datelor, iar ceilalți pentru transportul comenzilor, semnalelor de stare precum și a ceasului global.

4.1.1 TxMan

În timpul operării, funcția magistralei este de a furniza descriptori de pachete provenind de la TxMan tuturor MAC-urilor. Un descriptor constă din 5 cuvinte de câte 32 biți și conține suficientă informație pentru generarea unui pachet Ethernet. TxMan poate accesa o memorie SRAM privată de 1Mega-cuvânt (36 de biți/cuvânt), ce este folosită pentru stocarea descriptorilor de transmisie generați de PC-ul de control. Lărgimea de bandă a magistralei este suficientă pentru a permite generarea celui mai mic pachet Ethernet (64 octeți), la debit maxim.

Descriptorii de transmisie sunt produși de TxMan în modul următor. Ei sunt preluați

* Aceasta permite generarea unui trafic de 100Mb/s prin emiterea unui grup de 4 biți (în lb. engleză *nybble*) pe fiecare tact.

din memoria TxMan, unde au fost stocați în prealabil, prin ciclare într-o manieră secvențială sau pseudo-aleatoare.

4.1.2 MAC

Descriptorii trimiși de TxMan ajung la MAC, unde sunt tratați de către un procesor implementat în Handel-C. Acesta este controlat printr-un limbaj de asamblare specializat definit de noi, deci schimbările sunt deosebit de facile, implicând doar editarea programului în asamblare și încărcarea acestuia în MAC. Pachetele transmise conțin o ștampilă temporală privind timpul la care s-a emis pachetul; datorită faptului că ceasul operează la 25MHz precizia este de 40ns.

Există și o a doua posibilitate de a transmite, configurată tot prin intermediul unui program în asamblare, ce folosește valori fixe pentru dimensiunea pachetelor, timpul înter transmiterea a două pachete și adresa destinației.

Pachetele recepționate sunt prelucrate de MAC-uri pentru a genera descriptorii de recepție (tot 5 cuvinte a 32 de biți). Aceștia conțin date esențiale ce pot fi extrase direct din pachete, sau pot fi generate pe baza informației din acestea. Direct din pachete se pot extrage adresa sursei, prioritatea pachetului (dacă se folosesc câmpuri de tip VLAN, *Virtual Local Area Network*) etc. Pe baza informației temporale privind transmiterea pachetului și a valorii curente a ceasului se poate calcula latența pachetului, iar folosind momentele de recepție a pachetelor consecutive se poate determina timpul între sosirea a două pachete.

4.1.3 RxMan

Descriptorii de recepție sunt transmiși prin magistrală spre RxMan, care folosește informația cuprinsă în descriptori pentru a realiza diverse statistici, cum ar fi totaluri ale latențelor și numărului de octeți, sau histograme ale latențelor și timpilor între sosirea a două pachete succesive, în funcție de măsurătorile ce se efectuează. Histogramele sunt stocate în memoria SRAM privată de 1 Mega-cuvânt (36 de biți/cuvânt), într-un mod configurabil de către utilizator prin intermediul unor regiștri de control. Schimbarea naturii informației extrase din pachete necesită însă intervenția în cadrul programul scris în Handel-C și implică recompilarea și refitarea acestuia. Histogramele pot fi trimise la cerere PC-ului de control prin intermediul conexiuni IEEE 1824.

4.2 Emulatorul pentru ATLAS

Placa de test Enet32, datorită flexibilității și programabilității sale, a putut fi folosită

și pentru efectuarea unor simulări în cadrul experimentului ATLAS. Este vorba de emularea traficului de rețea generat de sistemele ROB (*Read-Out Buffer*) din cadrul nivelului 2 al arhitecturii ATLAS. Aceste sisteme trebuie să răspundă la cererile efectuate de sistemele de control prin unul sau mai multe pachete conținând datele furnizate de detectorii asociați.

Enet32 a fost programată pentru a prelua informația relevantă din cereri și pentru a produce răspunsurilor necesare. Acestea au proprietăți conforme cu cererea, dar datele conținute de pachete în timpul simulării nu sunt valide. În pachetele-răspuns trimise se inserează momentul de timp la care s-a făcut cererea (provenind din pachetul-cerere), astfel încât la recepția în PC-ul de control se poate calcula direct latența procesului cerere-răspuns. Se inserează deasemeni numărul pachetului-cerere pentru a permite detectarea pierderilor de pachete de către PC-ul care face cererile.

Deocamdată s-au făcut teste doar cu 4 plăci Enet32. Un sistem de test cu 8 plăci Enet32 (echivalent cu 256 ROB-uri) plus 64 de PC-uri cu rolul de supervisor și nodurile de control, împreună cu o arhitectură de *switch*-uri, modelează aproximativ 15% din întregul sistem ATLAS. Pe lângă informația privind pierderile de pachete și latența rețelei, se vor mai putea oferi histograme privind ocuparea cozilor interne, ceea ce constituie o măsură cantitativă privind elasticitatea și robustețea sistemului.

Informațiile obținute prin simulările efectuate conform descrierii de mai sus vor fi folosite pentru evaluarea diverselor strategii-candidat privind transmiterea mesajelor (cereri și răspunsuri), precum și privind topologia rețelei.

4.3 Programarea plăcii Enet32

Programarea plăcii Enet32 s-a făcut în mod integral folosind limbajul Handel-C, versiunea 2.0. Din această cauză programarea este ușoară chiar pentru cineva care are doar cunoștințe de C. Este însă bineînțeles necesară înțelegerea conceptului de paralelism și execuție sincronă, precum și celelalte diferențe care există față de limbajul C standard (vezi capitolul 3).

Producerea fișierelor de configurare a FPGA-urilor are loc în două etape. Mai întâi se compilează programul sursă în Handel-C, obținându-se un *netlist*. Acesta este preluat de utilitarul de fitare *Altera Max+PlusII* [Alt-**] care produce fișierele de configurare în formatul RBF (*Raw Bitstream File*). Fișierele binare sunt trimise la IoMan prin portul paralel, iar acesta programează FPGA-urile respective, pe un canal JTAG. IoMan însuși, a cărui comportare se schimbă rareori, este programat prin JTAG pe un cablu ByteBlaster. Există și

opțiunea de a folosi un cip PROM, din care IoMan își citește singur programul la punerea în funcțiune a plăcii; aceasta este modalitatea folosită în mod curent.

4.4 Interfața cu utilizatorul

Interacțiunea cu utilizatorul se realizează pe două căi. În primul rând există o suită de scripturi scrise în Perl* prin intermediul cărora se pot trimite comenzi plăcii, stabilindu-se porturile care participă la test, configurându-se aceste porturi, pornindu-se și oprindu-se testul. De asemenea se pot citi rezultatele testelor: sume de latențe, numărul de pachete transmise/recepționate/pierdute, numărul de octeți recepționați sau histograme ale latenței sau ale timpului între sosirea pachetelor. Scripturile menționate comunică fizic cu placa printr-un port paralel, folosind comunicația prin *pipe*-uri și un program scris în C.

Deși scripturile oferă o mare libertate în controlul plăcii, colegul meu Cătălin Meiroșu a programat și o interfață grafică, al cărei scop principal este definirea mai intuitivă a testului și afișarea grupată, în mod dinamic, a unor parametri de bază, cum ar fi numarul de octeți și de pachete transmise/recepționate pe secundă, latența medie a pachetelor și debitul la transmisie și recepție. La cerere este posibilă și afișarea unei ferestre ce prezintă și alte detalii privind un anumit port, cum ar fi de pildă numărul de pachete pierdute.

În figura 5 se prezintă interfața grafică la momentul în care era folosită pentru controlul a două plăci de test situate pe calculatoare diferite (un total de 64 de porturi FastEthernet). Pentru fiecare din aceste plăci, pe calculatoarele gazdă rulează un server care comunică prin *socket*-uri cu interfața grafică și trimite la rândul său comenzi plăci într-un mod asemănător cu cel descris mai sus, prin scripturi Perl.

* Perl este un limbaj de programare de nivel înalt, bazat pe C, *sed*, *awk* și alte instrumente din Unix ce permite scrierea de scripturi.

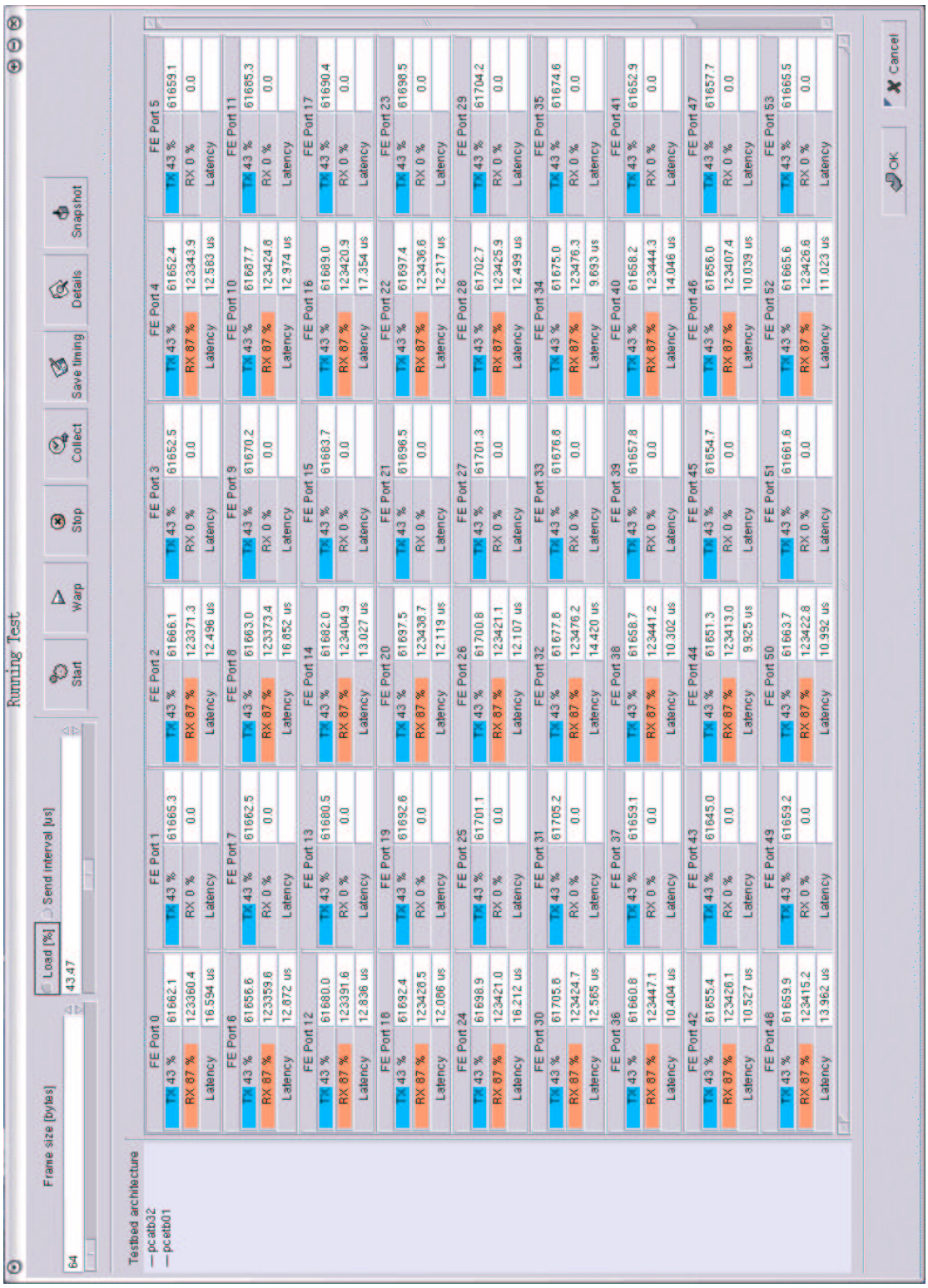


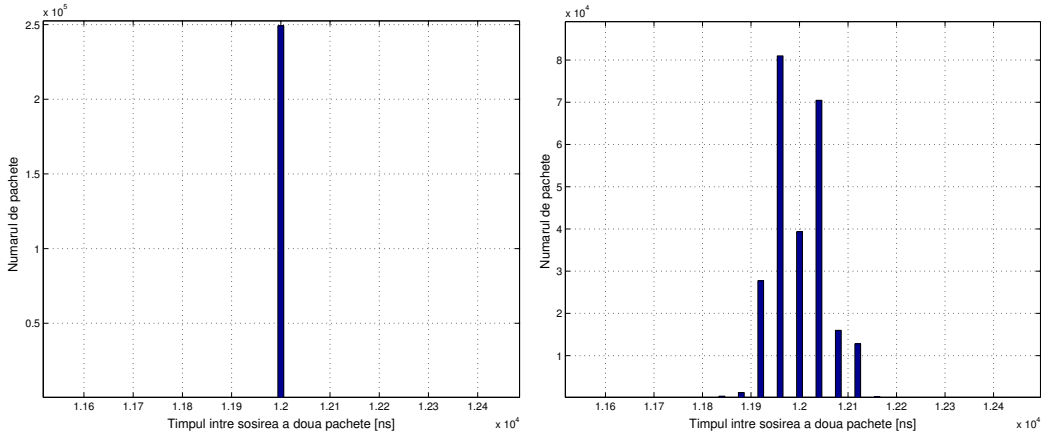
Figura 5: Interfața grafică cu utilizatorul.

4.5 Rezultate experimentale

Placa de test Enet32 se poate folosi pentru efectuarea unei game largi de măsurători, toate mărimile ce caracterizează traficul putând fi determinate. Singura problemă ce nu a fost încă rezolvată este sincronizarea înter două plăci astfel încât acestea să funcționeze ca un sistem unitar și să permită efectuarea măsurătorilor la distanță și pentru sisteme mai complexe. La momentul de față se lucrează la o soluție privind acest inconvenient.

În continuare, pentru a pune în evidență caracteristicile plăcii, sunt prezentate câteva rezultate obținute pentru generarea de trafic cu proprietăți controlate. Traficul generat este fie cu o rată constantă (CBR, *Constant Bit Rate*), fie de tip Poisson, adică cu o distribuție exponențială negativă discretă a timpilor între pachete. S-au folosit pachete de 64 și 1518 octeți (dimensiunile minimă și maximă permise în standardul Ethernet), iar caracteristicile traficului au fost astfel alese încât să determine o încărcare de aproximativ 50% a rețelei. Pentru pachetele de 64 de octeți aceasta înseamnă un trafic CBR cu 12μs între emiterea pachetelor sau un trafic Poisson cu media de 12μs; pentru pachetele de 1518, valoarea medie a timpului între emiterea pachetelor este de 250μs.

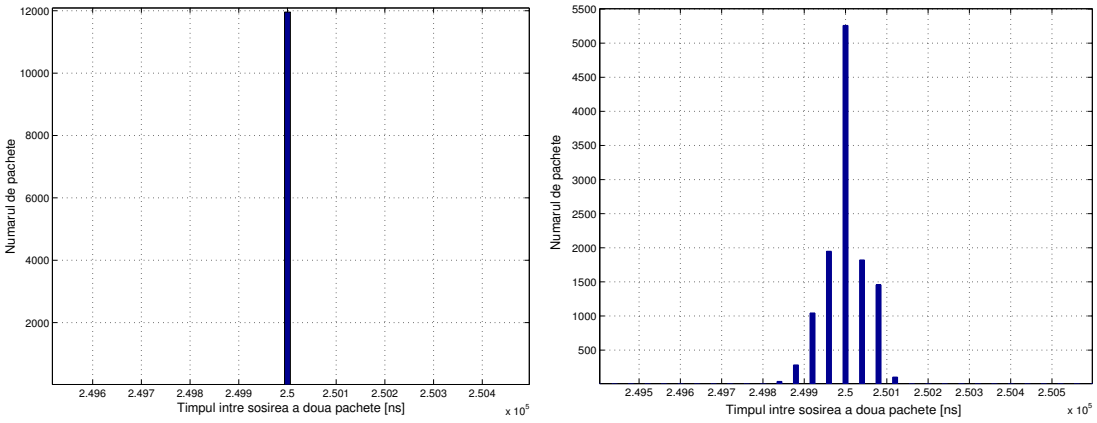
Proprietățile traficului sunt ilustrate prin intermediul histogramelor timpului între sosirea pachetelor la receptor. Pentru traficul de tip CBR se prezintă atât caracteristicile traficului la emisie, prin conectarea directă între portul ce transmite și cel ce recepționează, cât și caracteristicile traficului după trecerea printr-un *switch* (s-a folosit echipamentul T5 Compact al companiei *BATM*). Histogramele au fost efectuate folosind rezoluția maximă a plăcii, deci intervalul de cuantizare este de 40ns. Pentru figura 6 s-au alcătuit grupuri de 25 de astfel de intervale, deci rezoluția graficului este de 1μs, în scopul sporirii vizibilității caracteristicilor traficului.



(a) (b)

Figura 4: Timpul între sosirea a două pachete — (a) fără *switch*, (b) prin *switch*;
tipul traficului=CBR, dimensiunea pachetelor=64 octeți,
timpul între emiterea a două pachete=12μs.

Se observă din figurile 4 și 5 că proprietățile traficului la emitere sunt exact cele programate. După trecerea prin *switch* se constată o ușoară modificare a caracteristicilor, timpul între sosirea a două pachete îndepărtându-se cu până la 120ns față de valoarea la emitere.



(a) (b)

Figura 5: Timpul între sosirea a două pachete — (a) fără *switch*, (b) prin *switch*;
tipul traficului=CBR, dimensiunea pachetelor=1518 octeți,
timpul între emiterea a două pachete=250μs.

Figura de mai jos permite aprecierea calității bune a proprietăților traficului generat și în cazul celui de tip Poisson. Imaginea (b) pare mai zgomotoasă deoarece segmentul de timp prezentat este mai lung, aceasta deoarece pentru pachetele mari variația latenței este mult mai mare.

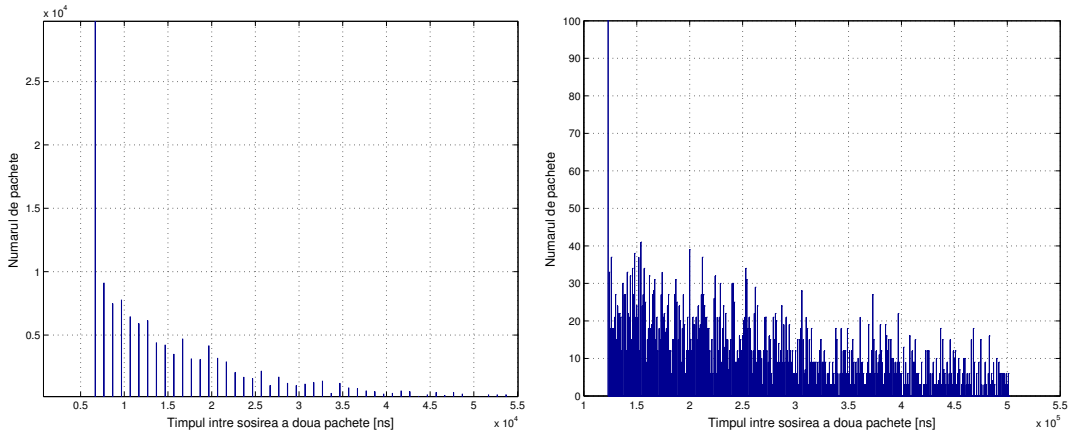


Figura 6: Timpul între sosirea a două pachete — (a) dimensiunea pachetelor=64 octeți, (b) dimensiunea pachetelor=1518 octeți; tipul traficului=Poisson,
timpul mediu între emiterea a două pachete=12µs, respectiv 250µs.

O altă mărime care se poate analiza cu ajutorul plăcii de test Enet32 este latența medie a pachetelor la trecerea printr-o rețea. În general aceasta este relativ constantă la nivelul unui *switch* pentru trafic de tip *unicast* (cu un emițător și un receptor), atât timp cât nu apar fenomene de congestie. Un aspect interesant este însă cazul *broadcast*-ului (un port este emițător și toate celelalte recepționează). Pe baza latenței medii se pot deduce în acest caz informații privind structura interna a *switch*-ului și modul de realizare a *broadcast*-ului de către acesta.

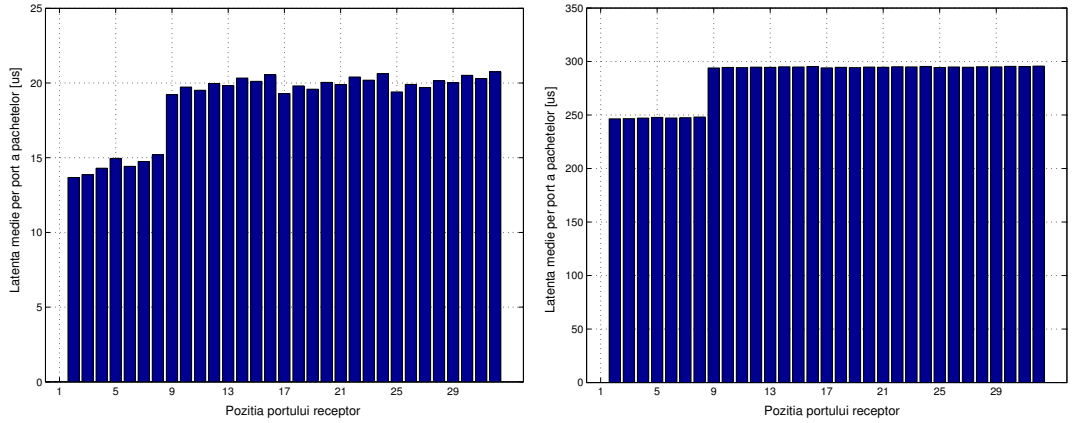


Figura 7: Latența medie per port a pachetelor pentru *broadcast* —
(a) dimensiunea pachetelor=64 octeți, (b) dimensiunea pachetelor=1518 octeți.

În figura 7 se prezintă latența medie a pachetelor ce sosesc la fiecare port din *switch*, singurul port emițător fiind cel cu indexul 0, cu trafic de tip *broadcast*. Cele 32 de porturi ale *switch*-ului care s-au folosit sunt grupate în module de câte 8 porturi. Se constată că pentru

porturile din primul modul, ce conține și portul cu numărul 0, latențele sunt mai mici decât pentru celelalte module. Explicația constă în faptul că traficul intra-modul beneficiază de legăturile foarte rapide care există în cadrul modulului.

Pentru celelalte module, ce conțin porturile de la 9 la 32, latența este cu aoproximativ 5μs mai mare. Se poate deduce așadar că pachetele au fost trimise per modul în mod centralizat de către o unitate centrală. În cadrul unui modul distribuirea se face apoi secvențial, într-o ordine fixă, care nu coincide însă cu modul de numerotare a porturilor în exterior. Efectul este mai evident pentru pachetele de dimensiuni mai mici, deoarece întârzierile introduse sunt relativ constante și devin nesemnificative în raport cu timpul efectiv de transmitere a unui pachet mare.

5 FPGA-urile în prelucrarea și analiza imaginilor

Sistemele bazate pe FPGA-uri tind să aibă un rol tot mai important în aplicațiile ce necesită o putere de calcul mare, aceasta fiind de obicei o cerință de bază în sistemele de operează în timp real. În capitolul de față se vor prezenta conceptele de bază legate de îmbinarea, ce devine tot mai strânsă, între *software* și *hardware*, accentul fiind pus de aplicațiile din domeniul prelucrării și analizei imaginilor. Motivul principal este că pentru acest tip de aplicații este întotdeauna nevoie de o putere de calcul relativ mare. În plus, operațiile simple la nivel de pixeli care se regăsesc în multe dintre tehnicile de prelucrare și analiză a imaginilor [Ver-00] — cum ar fi filtrarea, marcarea digitală a imaginilor [Beu-00], recunoașterea formelor, compresia imaginilor [Pae-91], [Wal-90] etc. — se pretează foarte bine la folosirea FPGA-urilor.

5.1 Calcul reconfigurabil

O chestiune de bază în proiectarea sistemelor de calcul actuale este raportul adecvat între viteză și generalitate. Se pot produce cipuri versatile, care au o funcționalitate largă, dar care execută relativ încet o sarcină, sau se pot crea cipuri specifice aplicațiilor, care realizează un număr limitat de sarcini, dar o fac foarte rapid. Microprocesoarele, cum ar fi *Intel Pentium*, *Athlon* sau *Motorola PowerPC*, ce se găsesc în computere, sunt cipuri cu scop general: instrucțiuni codate în formă binară pot conduce procesorul prin orice secvență de operații logice sau matematice posibilă. Procesorul *Pentium*, de pildă, nu a fost proiectat pentru a executa *Netscape Navigator* sau *Microsoft Word*, dar le poate rula pe ambele.

Prin contrast, circuitele *hardware* specializate (ASIC-uri), oferă exact funcționalitatea necesară într-un anumit context. Datorită acestui fapt este posibilă producerea de cipuri mai mici, mai ieftine și mai rapide, care consumă mai puțin. Un cip grafic poate afișa pe ecran forme geometrice sau imagini de 10 până la 100 de ori mai rapid decât o unitate centrală de uz general. Dezavantajul este că un ASIC nu poate în general rezolva o problemă ușor modificată față de cea pentru care a fost conceput. Chiar dacă un ASIC modificat poate fi reproiectat, este posibil ca dependența circuitelor de problemă să împiedice reutilizarea lor pentru generații succesive de produse. Aceasta face ca efortul depus la proiectare să trebuiască amortizat printr-un număr relativ redus de unități comercializate.

Dacă însă se înlocuiesc ASIC-urile cu FPGA-uri, constrângerile de mai sus dispar [Vil-97]. Este adevărat că se poate pierde în viteză, dar având în vedere că FPGA-urile actuale operează și la frecvențe de 200 de Mhz, această pierdere poate fi neglijată. Ceea ce a

deschis însă calea calculului configurabil este faptul că noile FPGA-uri pot fi reconfigurate foarte rapid. Dacă la început acest timp era de o secundă şi chiar mai mult (perfect însă pentru testarea proiectelor), în prezent a coborât până la nivelul milisecundelor şi se aşteaptă ca în curând ca acesta să ajungă de ordinul a 100 de microsecunde. În acest fel sistemele de calcul se pot adapta aproape în mod continuu la schimbările în datele de intrare sau în mediul de operare.

Paradigma de configurabilitate poate fi pusă în practică în cadrul sistemele de calcul în moduri diferite. Tehnica cea mai puţin pretenţioasă este alternarea la comandă între diverse funcţionalităţi; aceasta este echivalentul alternării între diverse programe înntr-un computer. Dacă reconfigurarea se poate face rapid, atunci devine posibilă trecerea printr-o succesiune de etape, fiecare adaptată unei faze a rezolvării problemei. Folosind această tehnică se poate construi, de pildă, un sistem de transmisie video într-un singur cip, care se reconfigurează de 4 ori pentru fiecare cadru [Vil-97]. În prima fază FPGA-ul stochează datele video în memorie, apoi se aplică două transformări de prelucrare a imaginilor şi în final FPGA-ul capătă funcţionalitatea unui modem pentru a transmite datele mai departe. În acest fel sunt necesare numai un sfert din resursele implicate de folosirea unui ASIC.

O modalitate mai spinoasă, dar mai puternică, de a folosi conceptul de calcul configurabil implică *hardware* care se auto-reconfigurează pe măsură ce execută o sarcină, rafinându-şi programarea pentru a creşte perfomanţa. Un cip destinat recunoaşterii formelor s-ar putea adapta ca urmare a unei încercări de a identifica un obiect: dacă acesta reprezintă un automobil, părţi ale circuitului care serveau la recunoaştere aeronavelor sau a persoanelor pot fi reconfigurate pentru a se concentra pe vehicule terestre. O astfel de abordare poate avea o influenţă importantă asupra performanţelor globale ale sistemului. Dacă la început reconfigurarea menţionată se va face pe baza unor proceduri precalculate, nu este exclusă venirea unui moment când, în mod inteligent, programul ce rulează în FPGA se va optimiza pe sine însuşi.

5.2 Recunoaşterea formelor

Una din aplicaţiile cele mai promiţătoare pentru calculul reconfigurabil o reprezintă recunoaşterea formelor. Această tehnică este folosită în scopuri diverse, cum ar fi recunoaşterea scrisului de mână, identificarea persoanelor după fotografii, obţinerea imaginilor din baze de date, sau recunoaşterea automată a ţintelor în domeniul militar. O operaţie fundamentală în majoritatea aplicaţiilor de recunoaştere a formelor o constituie compararea unui set de biţi de intrare (reprezentând o imagine, un şir de caractere sau orice

alt tip de informaţie) cu un set de şabloane corespunzând diverselor modele ce trebuie recunoscute. Sistemul declară recunoaşterea dacă numărul de biţi de intrare care se potrivesc cu biţii unui şablon depăşeşte un anumit prag.

În cazul recunoaşterii ţintelor, de exemplu, provocarea cea mai importantă este compararea rapidă a imaginii de intrare cu mii de modele. Un model poate reprezenta, de pildă, imaginea frontală sau laterală a unui anumit tip de vehicul. Fiecare imagine conţine în mod obişnuit mii de pixeli, iar o ţintă poate apare oriunde în imagine. Pentru a le recunoaşte suficient de rapid, conform cerinţelor aplicaţiilor militare, un sistem trebuie să efectueze comparaţii la o rată de câteva trilioane de operaţii pe secundă, deoarece toţi pixelii imaginii de intrare trebuie comparaţi cu cei ai şabloanelor.

În [Vil-97] se raportează construirea, cu sprijin din partea DARPA (*Defense Advanced Research Projects Agency*), a unui prototip de sistem de recunoaştere a formelor cu *hardware* configurabil, care realizează economii de resurse adaptându-se la fiecare model în parte. Mulţi din pixelii unui şablon tipic nu contribuie la deciziile luate, astfel încât ei ar putea fi omişi din calculele ulterioare. Un sistem convenţional nu ar putea face asta cu uşurinţă, deoarece poziţiile pixelilor diferă de la model la model. Se poate merge mai departe în exploatarea flexibilităţii sistemelor reconfigurabile prin luarea în consideraţie a similarităţilor care există între şabloane. Astfel, un set de modele poate fi prelucrat în paralel, folosind o singură unitate de comparaţie pentru toţi pixelii cu aceeaşi valoare în şabloanele respective. Rezultatul se propagă apoi pentru toate modelele din setul respectiv.

Un alt sistem prezentat în [Vil-97] este un prototip ce realizează encripţia folosind algoritmul DES (*Data Encryption Standard).* Acesta implică folosirea unei chei, ce rămâne de obicei fixă în timpul comunicaţiei, pentru criptarea mesajului în blocuri de biţi. Prin folosirea unui sistem bazat pe FPGA este posibilă adaptarea structurii interne la cheia folosită, obţinându-se astfel atât o economie de resurse (de la 25000 de porţi logice s-a ajuns la folosirea a numai 13000), cât şi o creştere a eficienţei. Când se schimbă cheia de criptare, noua structura internă este descărcată rapid în FPGA.

Din aceste exemple se deduce aşadar potenţialul care există pentru *hardware*-ul configurabil, prin adaptarea la tipuri de date diverse şi în continuă schimbare. Alte aplicaţii care ar putea beneficia de aceste avantaje sunt comunicaţiile digitale, prelucrările digitale în diverse domenii, cum ar fi cele pentru sistemele radar, etc.

5.3 Urmărirea obiectelor

Grupul condus de Ian Page la *Oxford University* are o lungă istorie privind implicarea

în proiectarea și realizarea de sisteme ce folosesc FPGA-uri. Ei au construit o platformă de calcul reconfigurabil numită HARP, ce constă dintr-un procesor RISC pe 32 de biți (un transputer T805 sau T425) cu o memorie DRAM de 4 Mocteți, strâns cuplat cu un sistem bazat pe un FPGA de la compania *Xilinx* (XC3195A), ce are o memorie locală de 2 bancuri SRAM a 32kbiți x 16 biți. În [Pag-97] se prezintă o aplicație care folosește sistemul HARP, aplicație ce va fi descrisă succint în continuare.

5.3.1 Prezentarea aplicației

Scopul aplicației este extragerea dintr-o secvență video, în timp real, pentru fiecare cadru, a dreptunghiurilor înglobante pentru obiectele care se mișcă în scena respectivă. Un context posibil este un sistem automat de siguranță, care se bazează pe o cameră video cu un câmp de vedere larg, dar care se poate mișca și poate face *zoom* pe obiectele aflate în mișcare. Imaginile ce conțin obiecte în mișcare pot fi înregistrate, sau un operator poate fi alertat în legătură cu detecția obiectului în mișcare, care constituie un potențial intrus.

Fluxul video este digitizat și subeșantionat de către transputer, astfel încât se obțin cadre pe nivele de gri de 128 x 128 x 8 biți. Acestea sunt trimise FPGA-ului folosind un canal de comunicație de tip magistrală. Transferul limitează dimensiunile imaginilor și ar fi de dorit existența unui sistem în care imaginile sunt accesibile direct FPGA-ului, totuși performanțele obținute prin utilizarea sistemului HARP sunt satisfăcătoare din punctul de vedere al raportului preț/performanță.

Aplicația funcționează prin efectuarea diferenței între imagini digitizate succesive și apoi o prăguire a imaginii-diferență rezultate. Un algoritm de umplere a regiunilor e folosit pentru identificarea frontierelor zonelor care s-au schimbat. Algoritmul este foarte simplu, de aceea unele regiuni concave nu sunt umplute, însă s-a doveditt suficient de performant pentru aplicația în discuție. Regiunile cu o arie aflată sub un anumit prag sunt ignorate.

În continuare se presupune că frontierele calculate mai sus corespund obiectelor în mișcare. Acestea sunt asociate dreptunghiurilor înglobante (ai căror parametri au fost calculați tot la pasul precedent) și din nou se elimină acele obiecte a căror arie se situează sub un anumit prag.

Pentru a crește robustețea sistemului, în primul rând față de zgomot, dreptunghiurile înglobante sunt modelate și urmărite cu un filtru Kalman predictiv, multi-obiect. Acesta modelează și prezice comportarea lor în timp, pe baza poziției, a dimensiunii și a derivatelor de ordinul întâi ale acestor mărimi. Astfel se reușeste chiar și urmărirea unui obiect care este temporar ocultat de un obstacol. Pentru fiecare dreptunghi se menține un nivel de încredere, ce corespunde probabilității ca acesta să fie asociat unui obiect real în mișcare; dacă nivelul

de încredere scade sub un anumit prag, dreptunghiul este înlăturat. Noi descrieri sunt create de fiecare dată când un obiect în mișcare este detectat.

5.3.2 Implementarea algoritmului

Algoritmul este implementat în așa fel încât operațiile la nivel de pixel rulează în FPGA, iar filtrul Kalman și interfața cu utilizatorul pe microprocesor. Aceasta deoarece *hardware*-ul este potrivit pentru calculele rapide cu complexitate scăzută, iar *software*-ul pentru calcule mai lente dar mai complexe. În consecință, diferențierea între cadre, prăguirea, umplerea regiunilor și calcularea dreptunghiurilor înglobante se fac toate într-un FPGA, care se ocupă deasemenea de comunicația cu microprocesorul.

Fiecare cadru de 128 x 128 pixeli este transferat în FPGA printr-o instrucțiune a microprocesorului de mutare a unui bloc. FPGA-ul calculează și returnează microprocesorului o listă, de lungime variabilă, de dreptunghiuri înglobante.

Manipularea fluxului video, filtrul Kalman și interfața pe bază de meniuri cu utilizatorul sunt rulate de microprocesorul programat în *occam*. Deși sistemul HARP este situat într-un PC, acesta nu are nici un rol la rulare, ci doar la inițializarea plăcii.

5.3.3 Performanțe

Sistemul produce la ieșire un flux de imagini video ce sunt primite de la placa HARP și afișate pe un monitor. Unul din modurile de afișare este prin supraimpunerea pe imaginea video a dreptunghiurilor înglobante. La utilizare se obține o rată de 12-23 de cadre pe secundă, în funcție de ceea ce se întâmplă în scenă. Aceste rezultate sunt destul de bune, având în vedere generalitatea sistemului folosit și faptul că programatorul nu a intervenit în nici un fel în proiectarea de *hardware*, mai jos de nivelul programului propriu-zis.

Dacă aceeași aplicație rulează integral pe microprocesor, performanța scade de șase ori. Acest factor ar fi fost și mai mare dacă sistemul inițial ar fi permis ca intrarea/ieșirea video să folosească o memorie direct accesibilă FPGA-ului. Oricum, performanțe similare ar necesita utilizarea a cel puțin încă 5 microprocesoare asemănătoare, ceea ce implică un efort în plus pentru distribuirea algoritmului în vederea calculului în paralel.

6 Concluzii

Calculul configurabil este un domeniu încă nou. Deşi conceptul a fost propus pentru prima oară de Gerald Estrin de la *University of California*, Los Angeles, la sfârsitul anilor '60, primele demonstraţii nu au apărut decât la mijlocul anilor '90, iar FPGA-urile utilizate în mod curent cu un număr de porţi de ordinul sutelor de mii, nu oferă încă suportul pentru exploatarea completă a posibilităţilor acestei tehnici. FPGA-urile de ultimă oră au însă un număr mai mare de porţi (de pildă cele din familia *Virtex* de la *Xilinx* au peste 1.000.000 de porţi, iar pentru circuitele *APEX II* de la *Altera* se ajunge până la 4.000.000), ceea ce permite implementarea unor aplicaţii mai complexe.

Cercetătorii şi producătorii încearcă depăşirea limitărilor care au împiedicat adoptarea pe o scară mai largă a FPGA-urilor. Acestea sunt potrivite pentru operaţii la nivel de biţi sau cu valori întregi, dar nu efectuează eficient operaţii numerice, cum ar fi înmulţirea sau calculul cu valori în virgulă mobilă. Circuitele de înmulţire dedicate pot fi optimizate pentru a realiza aceste operaţii mai eficient decât echivalentele lor create cu resurse interne. În plus, FPGA-urile oferă un spaţiu intern relativ redus pentru stocarea rezultatelor intermediare ale calculelor; de aceea, în multe aplicaţii, este necesară folosirea de memorii externe mari. Deasemenea, transferul de date dinspre şi înspre FPGA cresc consumul şi încetinesc calculele.

Unele cercetări sunt îndreptate spre dezvoltarea de FPGA-uri care conţin memorie, unităţi aritmetice şi alte blocuri speciale. André DeHon şi Thomas Knight, de la *Massachusetts Institute of Technology* au propus un FPGA care stochează configuraţii multiple într-o serie de bancuri de memorie. Într-un singur tact de ceas, ceea ce este de ordinul zecilor sau sutelor de nanosecunde, cipul îşi poate schimba configuraţia, fără a pierde datele parţial prelucrate.

Brad Hutchings, de la *Brigham Young University*, a construit un computer cu un set de instrucţiuni dinamic, care combină un microprocesor cu un FPGA şi demonstrează potenţialul reconfigurării automate folosind configuraţii prestocate. Pe măsură ce un program rulează, FPGA-ul cere reconfigurarea sa, dacă structura de porţi necesară nu este rezidentă. Este permisă astfel crearea de configuraţii multiple, adaptate diverselor sarcini, care pot fi activate similar cu iniţierea unui apel de subrutină într-un microprocesor.

Grupul Colt, condus de Peter Athanas, de la *Virginia Polytechnic Institute and State University*, investighează o tehnică de reconfigurare numită "gaură de vierme". În cadrul acesteia fluxul de date crează o structură logică adaptată pe măsură ce parcurge sistemul reconfigurabil.

John Wawrzynek și colegii săi de la *University of California*, Berkley, dezvoltă deasemenea un sistem care combină un microprocesor și un FPGA. Un compilator special preia un program obișnuit și generează în mod automat o combinație de instrucțiuni în cod-mașină pentru microprocesor și un ansamblu de configurații pentru FPGA, astfel încât performanțele globale ale sistemului să fie maximizate.

FPGA-urile nu vor înlocui niciodată microprocesoarele pentru activități de tip generic, dar conceptul de calcul configurabil va juca un rol tot mai mare în dezvoltarea sistemelor de calcul ultra-performante. Puterea de calcul oferită de FPGA-uri va face ca ele să fie preferate în aplicații ce necesită o adaptare rapidă la datele de intrare.

În plus, linia despărțitoare ce separă procesoarele programabile și FPGA-urile va deveni tot mai neclară. Viitoarele generații de FPGA-uri vor avea caracteristici ce sunt acum specifice microprocesoarelor: memorie locală mai mare, multiplicatoare dedicate etc. În schimb, noile generații de microprocesoare ce sunt dezvoltate în prezent posedă zone cu proprietăți asemănătoare reconfigurării FPGA-urilor.

Un prim pas într-o direcție similară a fost deja făcut de compania *Transmeta* prin procesorul *Crusoe*, apărut în ianuarie 2000. Acesta transferă cea mai mare parte a rolului procesorului — determinarea instrucțiunilor ce trebuie executate și a momentului execuției — în software, într-un proces numit adaptarea codului (în lb. engleză *code morphing*). Acesta efectuează în primul rând transformarea instrucțiunilor x86 în instrucțiuni specifice unității de execuție, precum și reordonarea acestora astfel încât mai multe instrucțiuni pot fi executate simultan. În paralel se realizează o statistică a instrucțiunilor procesate, astfel încât procesorul se adaptează la fluxul de intrare, iar instrucțiunile ce se repetă nu mai sunt din nou transformate, ci se utilizează direct instrucțiunile corespunzătoare stocate în *cache*.

Așa cum în prezent computerele își pot aduce prin intermediul Internetului componente *software* necesare pentru rezolvarea anumitor probleme, viitoarele mașini vor putea primi la cerere noi configurații *hardware*, în măsura necesităților. Este cert oricum că sistemele de calcul vor include o combinație strânsă de *hardware* programat prin *software* și de logică *hardware* reconfigurabilă. Aceasta face ca rolul limbajelor de programare facilă a *hardware*-ului, de tipul Handel-C, să continue să crească și în continuare.

Bibliografie

[Alt-**] Altera, Inc. (www.altera.com).

[Bar-01] F. R. M. Barnes, R. Beuran, R. W. Dobinson, M. J. LeVine, B. Martin, J. Lokier, C. Meiroşu, "Ethernet Networks for the ATLAS Data Collection System: Emulation and Testing", *Proc. of the 12th IEEE Real Time Congress on Nuclear and Plasma Sciences*, Valencia, iunie 2001, pp. 6-10.

[Beu-00] R. Beuran, "Marcarea digitală a imaginilor", *lucrare de disertație*, Universitatea "Politehnica" București, mai 2000.

[Cel-**] Celoxica, Ltd. (www.celoxica.com).

[Dob-01] R. W. Dobinson, S. Haas, K. Korcyl, M. J. LeVine, J. Lokier, B. Martin, C. Meiroşu, F. Saka, K. Vella, "Testing and modeling Ethernet switches for use in ATLAS high-level triggers", *IEEE Trans. on Nuclear Science*, vol. 48, no. 3, 2001.

[Hau-98] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems", *Proc. of IEEE*, vol. 86, nr. 4, aprilie 1998, pp. 615-638.

[Hoa-88] C. A. Hoare (editor), "Occam 2 Reference Manual", *Prentice Hall International Series in Computer Science*, Cambridge, 1988.

[Ker-88] B. Kernighan, D. Ritchie, "The C Programming Language", *Prentice Hall*, 1988.

[Max-96] C. Maxfield, "Field-programmable devices", *EDN Magazine*, mai 1996.

[Pae-91] A. W. Paeth, "Image File Compression Made Easy", *Graphics Gems II*, James Arvo (editor), *Academic Press*, San Diego, 1991.

[Pag-97] I. Page, "Hardware-software Co-synthesis Tesearch at Oxford", *Proc. of IEE Vacation School on Hardware/Software Co-design*, IEE, iulie 1997.

[Pro-**] *The Programmable Logic Jump Station* (www.optimagic.com).

[Sea-97] R. C. Seals, G. F. Whapshott, "Programmable Logic: PLDs and FPGAs", *Mc-Graw Hill*, 1997.

[Tan-97] A. S. Tanenbaum, "Rețele de calculatoare", ediția a 3-a, *Computer Press Agora*, 1997.

[Tau-95] E. Tau, D. Chen, I. Eslick, J. Brown, A. DeHon, "A First generation DPGA Implementation", *Proc. of the 3rd Canadian Workshop on Field-Programmable Devices*, mai 1995, pp. 138-143.

[Toa-96] Gh. Toacșe, D. Nicula, "Electronică digitală", *Editura Teora*, București, 1996.

[Ver-00] C. Vertan, M. Ciuc, V. Buzuloiu, R. Beuran, "New Trends in Color Image Analysis and Processing", *EEA - Electrotehnică, Electronică și Automatică*, vol. 48, no. 5-6, mai - iunie 2000, pp. 12-16.

[Vil-97] J. Villasenor, W. H. Mangione-Smith, "Configurable Computing", *Scientific American*, iunie 1997.

[Wal-90] G. K. Wallace, "Overview of the JPEG Still Picture Compression Algorithm", *Electronic Imaging East*, 1990.