Universitatea “Politehnica” din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

uClinux pentru sisteme FPGA

Conducător ştiinţific Student

*Conf.dr.ing. Ștefan STĂNCESCU Cătălina-Alexandra RUSU*

##### Anul 2013

**CUPRINS**

[INTRODUCERE 3](#_Toc373253902)

[CAPITOLUL 1. Embedded Linux 4](#_Toc373253903)

[1.1. Definiții și caracteristici 4](#_Toc373253904)

[1.2. Nivelele Linux 4](#_Toc373253905)

[1.3. Embedded Linux vs. Desktop Linux 6](#_Toc373253906)

[1.4. Sistemul de gestiune a memoriei 7](#_Toc373253907)

[1.5. Sistemul de gestiune a operațiilor I/O 7](#_Toc373253908)

[1.6. Sistemul de gestiune a fișierelor 8](#_Toc373253909)

[1.7. Sistemul de gestiune a proceselor 9](#_Toc373253910)

[CAPITOLUL 2. SISTEME RECONFIGURABILE DE CALCUL 12](#_Toc373253911)

[2.1. Generalități 12](#_Toc373253912)

[2.2. Field Programmable Gate Arrays (FPGA) 12](#_Toc373253913)

[2.2.1. Arhitectură 12](#_Toc373253914)

[2.2.2. Fluxul de proiectare 14](#_Toc373253915)

[CAPITOLUL 3. Arhitectura de ansamblu a uneltelor Și sistemelor dedicate 18](#_Toc373253916)

[3.1. Descriere generală 18](#_Toc373253917)

[3.2. Prezentarea generală a procesului de proiectare 19](#_Toc373253918)

[3.2.1. Dezvoltare de tip hardware 19](#_Toc373253919)

[3.2.2. Dezvoltare de tip software 20](#_Toc373253920)

[3.2.3. Verificare 20](#_Toc373253921)

[A.Verificarea hardware folosind simularea 20](#_Toc373253922)

[B. Verificarea software folosind opțiunea „debug” 20](#_Toc373253923)

[3.2.4. Configurarea dispozitivului 20](#_Toc373253924)

[3.3. MicroBlaze (Xilinx) 21](#_Toc373253925)

[3.3.1. Introducere în lumea “MicroBlaze” 21](#_Toc373253926)

[3.3.2. Configurarea mediului Xilinx 23](#_Toc373253927)

[3.3.3. Transferul sistemului de operare Linux pe MicroBlaze 24](#_Toc373253928)

[CAPITOLUL 4. PLATFORMA Z4800 25](#_Toc373253929)

[4.1. Noțiuni generale 25](#_Toc373253930)

[4.2. Proiectare 26](#_Toc373253931)

[4.3. Instrucțiunile load-linked și store-conditional 26](#_Toc373253932)

[CONCLUZII 28](#_Toc373253933)

[BIBLIOGRAFIE 29](#_Toc373253934)

# INTRODUCERE

Bariera existentă între sistemele de operare dedicate și cele de uz general devine din ce în ce mai obscură pe măsură ce dispozitivele dedicate evoluează pentru a îndeplini cerințele pieței. Aceste dispozitive trebuie să aibă capacitatetea de a rula aplicații din ce în ce mai sofisticate și să fie dependente de o administrare mai inteligentă. Însă este de așteptat ca ele să rămână soluții eficiente din punct de vedere al costului pentru misiunea lor limitată.

Pentru companiile care furnizează soluții pentru piețele dedicate, timpul redus de fabricație și fiabilitatea platformei sunt caracteristicile cheie ale dezvoltării. Soluțiile pentru mediul industrial ar trebui să furnizeze o conectivitate a rețelei la dispozitivul terminal, putând să suporte mai multe protocoale diferite de rețea. Pentru o soluție totală de rețea, trebuie luate în considerare și cerințele folosite pentru securizarea și administrarea dispozitivelor terminale. Acestea și mai multe cereri vor crește nivelul de complexitate, fiind necesară folosirea unui sistem de operare complet.

În [17] se arată că pentru aceste provocări, modelul embedded Linux are avantajul de a fi total gratuit, de tip open source și reprezintă o soluție compactă, furnizând o fundație puternică pentru aplicațiile ce vor fi rulate pe el. Oferă și un suport puternic pentru rețea, cum ar fi protocoale de administrare a rețelei, dar și opțiuni de securitate pentru email și pentru alte aplicații. Avantajele pe care le oferă modelul standard Linux pot fi găsite în versiunile dedicate. Caracterul deschis al Linux-ului permite realizarea de modificări rapide pentru cerințele fizice, realizând în același timp și un sistem de bază standardizat și fiabil pentru producători.

# CAPITOLUL 1. Embedded Linux

## Definiții și caracteristici

Majoritatea definesc „embedded Linux” (Linux dedicat) ca partea software care rulează pe un dispozitiv cu funcție fixă sau limitată, permițând acestuia să profite din plin de avantajele puterii de procesare și de perifericele conexe. Într-un dispozitiv cu funcție fixă, embedded Linux „strălucește” când vine vorba de eficiență în ceea ce privește puterea, performanța și ușurința în utilizarea dispozitivului pentru îndeplinirea scopului propus. Cu toate acestea, majoritatea sistemelor dedicate pe care rulează Linux sunt tablete și smartphone-uri; acestea nu fac parte din categoria dispozitivelor cu funcție fixă.

Avantajele utilizării sistemului embedded Linux [11] sunt :

**1.Furnizor independent**

 Folosirea Linux-ului înseamnă independență față de un anumit distribuitor de unelte. Pentru Linux, totul este disponibil prin comunitatea de tip open-source (sursă-deschisă). Așadar, utilizatorul poate schimba cu ușurință un distribuitor cu altul.

**2. Disponibilitate rapidă a uneltelor folosite**

În cazul sistemului embedded Linux, o mare parte din utilități și unelte de dezvoltare sunt disponibile gratuit. Utilizatorul le poate descărca și folosi așa cum dorește. Acest lucru duce la un timp de dezvoltare rapid în cazul produselor specifice sistemelor dedicate.

**3. Suporturi hardware variate**

Comunitatea Linux este foarte activă, asigurând în mod regulat suport pentru noile echipamente hardware. Linux este folosit în diferite universități și laboratoare de cercetare din lume, ceea ce înseamnă că este mereu actualizat cu cele mai noi suporturi hardware.

**4. Dezvoltare ieftină**

Utilizarea sistemului de operare Linux în producția de sisteme dedicate duce la dezvoltarea produselor la un preț redus. Uneltele de dezvoltare pentru Linux sunt gratuite și ușor de găsit. Astfel nu este nevoie să fie plătit un preț mare pentru producerea unui număr de produse.

## 1.2. Nivelele Linux

Fiecărui model Linux îi corespunde  un kernel Linux general, caracterizat pe scurt în cele ce urmează. Un sistem de operare Linux este reprezentat printr-o structură ierarhică pe mai multe nivele [19]:

A. **Un nivel hardware** – format din componente hardware precum CPU, memorie principală de tip RAM,  memorie secundară de tip HDD/Flash, elemente de rețea;

B. **Un nivel sistem** – reprezentat prin kernel-ul sistemului, care  oferă interfața între aplicații și partea hardware, pentru ca aceste aplicații folosite să poată fi portabile.

Nucleul  are trei componente principale:

* Sistem de gestiune pentru procese (process manager);
* Sistem de gestiune pentru fișiere (file system manager);
* Componenta pentru comunicațiile în rețea (IP + TCP/UDP);

C. **Un nivel de utilizator** – în acest nivel  se găsesc limbajele de comandă (shell-urile), programele utilitare și aplicații de tip utilizator.

Fiecare nivel are la bază serviciile sau resursele pe care le poate oferi nivelul imediat inferior.



Figura 1.1. Structura sistemului Linux [15]

Din figura 1.1 se poate observa că nucleul (kernel-ul) Linux separă partea hardware de sarcinile nivelului utilizator. Kernel-ul are capacitatea de a suspenda orice sarcină la nivel de utilizator.

## 1.3. Embedded Linux vs. Desktop Linux

Diferențele între un sistem de operare Linux de tip desktop și unul dedicate [11] sunt prezentate în tabelul 1.

|  |  |
| --- | --- |
| **EMBEDDED LINUX** | **DESKTOP LINUX** |
| Kernel-ul Linux-ului rulează pe sistem dedicat/ calculator de tip single board/ placă de dezvoltare | Kernel-ul Linux-ului rulează pe Desktop/ Laptop |
| Este folosit un kernel Linux de tip real-time. Răspunsul kernel-ului este în timp real sau este determinist. | Kernel-ul care lucrează pe desktop sau laptop nu lucrează în timp real. Răspunsul lui nu este determinist în cazul răspunsurilor la evenimente. |
| Scopul folosirii kernel-ului Linux-ului este realizarea unei anumite funcții. Exemplu: Kernel-ul Linux-ului rulează în Video Streamer pentru a converti formatul video în MPEG4 și transmiterea semnalului video în rețea. | Scopul folosirii kernel-ului Linux-ului este acela de a ușura munca utilizatorului. Exemplu: Kernel-ul Linux-ului care rulează pe desktop realizează mai multe aplicații diferite pentru sarcini precum rularea fișierelor video, a lui Internet Explorer, a unui program de tip Java sau C, etc. Sunt implicate mai multe obiective. |
| Este foarte clar din cele spuse mai sus că Linux trebuie să îndeplinească o anumită sarcină, astfel că modulul adăugat alături de componentele sale sunt limitate din punct de vedere al cerințelor aplicației. Ca și în exemplul de mai sus, pot fi necesare următoarele module sau librării alături de kernel:Stiva TCP/IPEthernet DriverDriver-ul pentru controlerul graficLibraria graficaStiva USB daca este implicat un conector USBDriverul de sunet.  | Astfel, se poate spune că sunt necesare mai multe module/librării sau drivere în funcție de componentele de pe placă, precum:Stiva TCP/IPEthernet DriverDriver-ul pentru controlerul graficLibraria grafica Driverul de sunet.Driverul pentru USBDriverul PCIDriverul pentru portul serialEtc. |
| Pe scurt, kernel-ul folosit în cazul sistemelor dedicate este o versiune personalizată a celui original. Utilizatorul configurează kernel-ul în funcție de sarcina pe care trebuie să o îndeplinească procesorul, de componentele de pe placă, de nevoia de drivere, etc. | Versiunea completă a kernel-ului este folosită alături de toate driverele și librăriile disponibile. Atunci când un dispozitiv sau protocol nou este scos pe piață, driverele sunt furnizate de comunitatea Linux sau de distribuitor. |

Tabel 1. Embedded Linux vs. Desktop Linux

uClinux are proprietăți similare ca cele ale modelului standard Linux, dar are ca avantaj optimizarea pentru aplicații și dispozitive dedicate. Este optimizat în mod special pentru a minimiza dimensiunea codului necesar atât pentru aplicații cât și pentru kernel. Acest kernel este derivat dintr-un kernel standard de  Linux v2.0 fără o administrare a memoriei. Kernel-ul suportă multe familii de procesoare, precum Alpha, ARM,  Blackfin, i386, m68k, MicroBlaze, MIPS, PPC, SH, SPARC, etc.

## 1.4. Sistemul de gestiune a memoriei

Diferența principală între uClinux și alte sisteme Linux o constituie lipsa unității de gestiune a memoriei. În ceea ce privește sistemul Linux, această unitate de gestiune a memoriei este realizată cu ajutorul memoriei virtuale (virtual memory-VM). În schimb, uClinux a fost creat pentru sisteme care nu acceptă VM [20].

Utilizând VM, toate procesele rulează la aceeași adresă; deși este una virtuală, memoria VM se ocupă de memoria fizică mapată la aceste locații. Chiar dacă memoria virtuală pe care o vede procesul este alăturată celei fizice, aceasta din urmă poate fi răspândită pe o suprafață mai mare. O parte din această memorie poate fi găsită pe un hard disk în memoria Swap a acestuia. Deoarece memoria localizată arbitrar poate fi mapată oriunde în spațiul de adrese al procesului, este posibilă adăugarea de memorie la un proces care deja funcționează.

Fără VM, fiecare proces trebuie să fie localizat într-un loc în memorie unde să poată rula. În general, nu poate fi extinsă deoarece pot exista procese în derulare în locațiile de deasupra sau de dedesubtul celei curente. Acest lucru înseamnă că un proces în uClinux nu poate crește dimensiunea memoriei existente în timpul rulării ca și un proces tradițional în Linux.

Neutilizarea memoriei VM reprezintă o problemă pentru fabricanții de uClinux. Rezultă că nu este oferită o protecție a memoriei - este posibil ca anumite aplicații sau chiar kernel-ul să corupă o parte din sistem. Unele arhitecturi CPU permit anumitor suprafețe de I/O, instrucțiuni sau regiuni din memorie să fie protejate de programele folosite de utilizatori, însă acest lucru nu este garantat. Un efect mai nociv decat coruperea care distruge un sistem este cea neobservabilă, iar depistarea coruperilor aleatoare interproces este una foarte dificilă.

## 1.5. Sistemul de gestiune a operațiilor I/O

Sistemul I/O al kernel-ului Linux reprezintă unul din componentele critice care afectează performanța volumului de informații pe server. Porturile de I/O sunt implementate printr-un ansamblu de drivere de dispozitive, unul pentru fiecare tip de dispozitiv. Funcția driver-ului este de a izola restul sistemului de posibilele anomalii care apar din cauza hardware-ului. Prin folosirea unei interfețe standard între drivere și restul sistemului de operare, o mare parte din sistemul de I/O poate fi pusă în partea independentă a kernel-ului.

Principalele funcții pe care trebuie să le genereze un sistem de operare în scopul comunicării cu perifericele sunt următoarele:

* Generarea comenzilor către dispozitivele pericerice;
* Tratarea întreruperilor specifice de I/O;
* Tratarea eventualelor erori cauzate de I/O;
* Furnizarea unei interfețe utilizator cât mai standardizată și mai flexibilă.

În versiunea uClinux, majoritatea driverelor din kernel pentru dispozitive sunt eliminate, păstrându-se doar acelea care vor fi folosite de dispozitivul creat. De asemenea, tratarea erorilor se poate face la nivelul kernel sau se poate transfera această responsabilitate nivelului aplicație (aplicația va trebui construită astfel încât să solicite retransferul în cazul în care se depășește pragul de acceptare a erorii unei transmisiuni).

## 1.6. Sistemul de gestiune a fișierelor

În sistemul de operare Linux, datele și programele sunt păstrate în fișiere identificate prin nume. Numele fișierelor pot avea până la 255 de caractere și pot conține oricâte caractere “.”, singurele restricții fiind nefolosirea caracterelor neprintabile sau a caracterelor NULL, “/” și a spațiilor albe TAB si SPACE. Spre deosebire de MS-Windows (adică de sistemele de fișiere FAT și FAT32), numele fișierelor în Linux sunt case-sensitive, adică se face distincția între majuscule și minuscule.

În sistemele embedded Linux, există foarte rar un hard disk. De aceea, în majoritatea situațiilor se pune problema cum se poate crea un sistem de fișiere care să nu se bazeze pe un hard disk. Există mai multe tipuri de memorii Flash create special pentru stocarea non-volatilă a datelor, cum ar fi memoriile flash NAND [18] (negative AND – simulează comportamentul unei porți NAND la nivelul unui grup de tranzistori) sau NOR (negative OR) – simulează comportamentul unei porți NOR la nivelul unui grup de tranzistori) și dispozitivele „disk-on-chip”.

Memoriile NAND sunt mai ieftine, dar pot prezenta câteva erori hardware (se folosesc în general la dispozitive USB și carduri de memorie).

Memoriile NOR sunt mult mai rar predispuse la erori, de aceea se folosesc drept suport de stocare a BIOS-ului în sistemele embedded.

Cum memoria Flash NOR este folosită de obicei pentru stocarea codului (numită și „memorie de cod”), implementarea unui sistem de gestiune a fișierelor în această memorie Flash NOR este soluția cea mai eficientă din punct de vedere al costului și al numărului minim de modificări pe placa de dezvoltare.

Există mai multe tehnologii Linux ce funcțioanează împreună pentru a implementa sistemul de gestiune a fișierelor pentru un sistem de operare embedded Linux. Figura 1.2 ilustrează legătura între componentele standard din această structură.



Figura 1.2. Componente sistemului de gestiune a fișierelor într-un sistem de operare embedded Linux [18]

## 1.7. Sistemul de gestiune a proceselor

Linux este un sistem de operare multitasking, ceea ce înseamnă că permite rularea mai multor programe la un anumit moment de timp prin comutarea CPU între diferite task-uri. Când un program este executat, se generează o instanță numită „proces”. Fiecărui proces îi corespunde un identificator unic de proces (PID), a cărui valoare este un număr pozitiv cuprins între 2 și 32768 [17].

Într-un mediu multitasking, un proces poate fi în mai multe stări în timpul execuției. Stările posibile sunt Running (rulare), Ready (pregătit de lucru), Waiting (așteptare), Stopped (oprit) și Zombie. O sarcină este gata de lucru atunci când se poate executa, dar nu are suficientă prioritate să facă acest lucru. Starea Running este starea în care procesul primește o cuantă de timp pentru a fi executat în cadrul procesorului. Starea de așteptare se referă la așteptarea procesului pentru ca un eveniment să fie lansat. Un proces este oprit atunci când nu mai este programat temporar pentru execuție. Starea Zombie reprezintă starea în care un proces trece atunci când procesul său părinte nu i-a determinat corect încercarea execuției sau zona de memorie pe care a solicitat-o nu a putut fi eliberată, ocupând astfel inutil loc în coada de așteptare.

Timpul de calcul pe sistemele Linux este alocat în „jiffies” [16] (perioade de timp alocate pe microprocesor). Pe majoritatea sistemelor Linux, acest jiffie reprezintă 1/100 dintr-o secundă.

|  |  |
| --- | --- |
| Nume câmp | Semnificație |
| PID |

|  |
| --- |
| Identificatorul procesului  |

 |
| USER |

|  |
| --- |
| Proprietarul procesului  |

 |
| PR |

|  |
| --- |
| Prioritatea procesului  |

 |
| NI |

|  |
| --- |
| Valoarea parametrului nice pentru ajustarea nivelului de prioritate pentru proces. O valoare cât mai negativă înseamnă o prioritate cât mai mare.  |

 |
| VIRT |

|  |
| --- |
| V olumul total al memoriei virtuale utilizate de proces, exprimat în unități kb (en. Virtual Image). Include: codul și datele procesului, bibliotecile dinamice partajate. VIRT = SWAP+RES; |

 |
| RES |

|  |
| --- |
| Volumul de memorie fizică utilizată de proces, care nu a fost transferată în swap, exprimat în unități kb (en. Resident size). RES=CODE+DATA;  |

 |
| SHR |

|  |
| --- |
| Volumul de memorie partajată utilizată de proces, exprimată în unități kb (Shared memory size). Reflectă volumul de memorie care ar putea fi partajat cu alte procese.  |

 |
| S |

|  |  |
| --- | --- |
| Starea rocesului. Poate avea valorile:

|  |
| --- |
| D = “uninterruptible sleep”; R = “running”; S = “sleeping”; T = “traced or stopped”; Z = “zombie”;  |

 |

 |
| %CPU |

|  |
| --- |
| Interval de timp CPU alocat unui proces, exprimat în procente din timpul total CPU măsurat de la ultima afiare.  |

 |
| %MEM |  Volumul de memorie fizică utilizat de un process, exprimat în procente. |

Tabel 2. Semnificațiile câmpurilor unui proces

Există 3 tipuri de planificare a proceselor:

* **normal** – acest tip de planificare este specific pogramelor normale;
* **FIFO** – planificare specifică sistemelor real-time, se referă la faptul că primul proces pornit (first in) va fi și primul proces care termină execuția (first out). Procesul iese din execuție dacă intră în una din stările S, D, T, terminare sau zombie.
* **RR** (round robin) – planificare specifică de asemenea sistemelor real-time, în care fiecare proces primește o anumită cuantă de timp în care își poate desfășura activitatea după care trebuie să se opreasca și să predea controlul unui alt proces (din coadă).

Procesele Linux au următoarele caracteristici:

* **Politica** – normală sau real-time. Procesele real-time au o prioritate mai mare decât

procesele normale.

* **Prioritate** – valoarea sa este un număr întreg între -20 (prioritate maximă) și 19 (prioritate minimă).

Procesul care apelează funcția fork() pentru a crea un fișier se numește proces părinte, iar procesul creat se mai numește proces fiu.

Procesele create astfel au segmentul de text identic cu al procesului părinte, însă nu partajează segmentul de date și stiva cu acesta.

# CAPITOLUL 2. SISTEME RECONFIGURABILE DE CALCUL

## 2.1. Generalități

Precum multe alte tehnologii, sistemele de calcul reconfigurabile sunt construite folosind o mare varietate de tehnici şi tehnologii existente. Întotdeauna este dificil de identificat la momentul exact un nou domeniu de tehnologie, care ia naştere sau chiar pentru a indica care este primul sistem într-o nouă clasă de maşini.

 Sistemele reconfigurabile sunt acele sisteme care utilizează componente hardware ce pot fi reconfigurate la nivel logic cu scopul de a rezolva probleme caracteristice. Acestea au trei instanţe definitorii: paralelism adaptat proporţional cerinţelor design-ului, logică direct orientată spre funcţii corespunzătoare şi reconfigurare la nivel hardware [3].

 Cercetări recente arată că sistemele reconfigurabile de calcul sunt un trend pentru o gamă variată de aplicaţii, de la procesarea de imagini până la operaţii cu virgulă mobilă. Deşi este importantă,viteza nu este singurul avantaj al calculului reconfigurabil. Un alt avantaj îl constituie consumul redus de energie şi putere. Într-un sistem reconfigurabil, circuitistica este optimizată pentru aplicaţie, iar consumul de putere va scădea faţă de cel al unui procesor de uz general. Rezultatele unui studiu recent [4] arată că schimbarea buclelor din software cu un hardware reconfigurabil va duce la o ecomonisire de energie de la 35% până la 70% cu o creştere a vitezei de la 3 la 7 ori, în funcţie de dispozitivul ales. Alte avantaje ale circuitelor de calcul reconfigurabil sunt: reducerea dimensiunilor şi a numărului de componente, alături de cost, creşterea flexibilităţii şi posibilităţii de îmbunătăţire. Aceste avantaje sunt importante în special pentru aplicaţiile încorporate.

## 2.2. Field Programmable Gate Arrays (FPGA)

### 2.2.1. Arhitectură

O arhitectură complet diferită care a fost introdusă la jumătatea anului 1980 şi utilizează memoria RAM bazată pe tabela de căutare( look-up tables-LUT) în loc de porţi AND şi OR pentru a implementa logica combinaţională, poartă denumirea de *FPGA(Field Programmable Gate Arrays)*. Acest dispozitiv electronic este format din blocuri logice configurabile (Configurable Logic Blocks-CLB) înconjurate de blocuri de intrare/ieşire (I/O). Placa Spartan- 3E de la firma Xilinx conţine de asemenea şi câteva blocuri de memorie RAM, dar şi multiplicatoare.

FPGA-ul poate conţine trei tipuri de circuite:

* nevolatile - circuite care pot fi programate o singură dată, denumite anti-fuzibili
* nevolatile – circuite reprogramabile(flash)
* volatile (SRAM - static RAM)

 bloc logic

 bloc I/O Bloc logic

latch

D Q

 Q

1

0

LUT cu 4 intrări

 intrări

 ieșire

 Figura 2.1. Arhitectura FPGA [3]

În [5] se specifică faptul că sistemele reconfigurabile de calcul au fost influenţate foarte mult de dezvoltarea circuitelor de tip FPGA. Pe de o parte, circuitele FPGA reprezintă o sursă de părţi componente ale produsului, oferind piese ieftine şi uşor de programat în unul dintre cele mai avansate procese de fabricaţie disponibile. Pe de altă parte, ele nu sunt optimizate pentru sistemele reconfigurabile de calcul din simplul motiv că marea majoritate a clienţilor de circuite FPGA le consideră ca fiind circuite ASIC ieftine şi de calitate redusă, cu durată limitată pe piaţa de desfacere.

Pornind de la această observaţie a logicii digitale, percepem tabelele de adevăr ca pe inima sistemelor de calcul din FPGA. Mai specific, un element hardware care poate, cu uşurinţă, să implementeze o tabelă de adevăr este tabela de căutare LUT .

Un bloc logic configurabil este format din mai multe tabele de căutare (LUT). Această denumire poate fi diferită de la un producător de circuite la altul. De exemplu, firma Xilinx foloseşte termenul de CLB. Blocurile logice se conectează între ele cu ajutorul unui ansamblu de resurse programabile de rutare.

Cercetări atât în cazul dimensiunilor LUT cât şi a numărului de blocuri LUT încep să adreseze întrebarea complexă a granularităţii calculului într-un FPGA. Într-o parte a spectrului, cea mai simplă structură a unui mic tabel de căutare ( exemplu 2-LUT) o reprezintă granulaţia mică ("fine grained") a capacităţii de calcul. La celălalt capăt, "coarse grained" sau circuitele cu granulaţie mare, pot fi imaginate ca blocuri de calcul mari, precum unităţile logice aritmetice pe 8 biţi ( ALU), mai tipice pentru CPU. Ca şi în cazul dimensiunii tabelului de căutare, blocurile cu granulaţie mică pot fi mai bine adaptate pentru manipularea şi aritmetica la nivel de bit, dar necesită combinarea mai multor astfel de blocuri pentru a implementa sisteme mai complexe de logică de calcul. Acestea sunt comparate cu blocurile cu granulaţie mare, fiind folosite în special pentru sistemele de calcul orientate pe calea de date, care lucrează cu dimensiunile standard ale cuvintelor ( pe 8/16/32 de biţi), dar pot fi utile şi atunci când se implementează operaţii cu logică foarte simplă.

Spunem că implementăm un design în FPGA atunci când descompunem design-ul şi mapăm în blocul logic. Circuitele FPGA mai conţin pe lânga CLB şi blocuri I/O, blocuri de memorie RAM, multiplicatoare, unitate aritmetico-logică (ALU) şi sumatoare.

În fabricarea lor, FPGA-urile combină logica programabilă cu resurse adiționale dedicate din următoarele motive:

* resursele dedicate consumă mai puțină putere;
* resursele dedicate pot funcționa la frecvențe ridicate.

### 2.2.2. Fluxul de proiectare

Noţiunea de “flux de proiectare” se fundamentează pe un ansamblu de etape ce trebuie respectate într-o ordine prestabilită, logică, paşi ce vor fi prezentaţi în continuare.



Figura 2.2. Etapele fluxului de proiectare [6]

1. Prima etapă importantă a complexului constă în *descrierea* circuitului logic cu ajutorul modului grafic (schematic) sau a unui limbaj de descriere hardware (Hardware Description Language -HDL), ca element reprezentabil esenţial, din categoria căruia fac parte limbajele VHDL şi Verilog.

2.Consecutiv primului pas important, urmează implementarea unui program care are rolul de a *sintetiza* logica, convertindu-se limbajul HDL sau schema într-o listă de legături (netlist) în care se prezintă porţile logice din design şi modul de interconectare specific acestora.

3. Faza trei mai este numită şi faza de *implementare* şi este formată din translatare, mapare, plasare şi rutare, ca procese de baza, specifice şi utile. Procesul de translatare reuneşte toate intrările listei de legături şi constrângerile într-un fişier de proiectare logică, termenul de “constrângere” nereprezentând în acest context nimic altceva decat o simplă atribuire a porturilor din proiect componentelor fizice (pini,butoane, comutatoare, etc) ale dispozitivului ţintă, specificând cerinţele de timp pentru proiectare; ulterior, informaţiile sunt salvate într-un fişier cu extensia .ucf (User Constraints File). În cea de-a doua sub-etapă, se mapează porţile logice şi interconexiunile din FPGA, utilizând uneltele de implementare disponibile. Procesul de plasare şi rutare are ca principiu de derulare procesul prin care sunt reintegrate sub-blocurile din etapa de mapare şi plasate în blocurile logice conform constrângerilor şi modului de interconectare.

4. După terminarea fazei de implementare, din matricea de rutare se extrage starea comutatoarelor şi se genereaza şirul de biţi (bitstream), unde “1” corespunde stării de deschis, iar “0” corespunde stării de închis a comutatoarelor.

5. Consecutiv, şirul de biţi generat în etapa anterioară este înscris în circuitul FPGA şi are extensia .bit.

Securitatea listei de legături (netlist), descrisă în [7], capătă o latură esenţială în cadrul importanţei deosebite pentru mulţi utilizatori de FPGA-uri de tip hardware. Fişierul de programare a FPGA-ului conţine informaţia listei de legături în asa fel încât producătorii de FPGA oferă opţiuni puternice de criptare pentru protejarea conţinutului din şirul de biţi. Însă, suportul criptografic este disponibil doar la FPGA-urile de ultimă generaţie, unele implementări având nevoie de o baterie externă pentru stocarea cheilor criptografice. Astfel, multe proiecte încorporate nu îşi pot permite aceste FPGA-uri care sunt foarte mari, scumpe şi consumatoare de putere, toate aceste caracteristici fiind definitorii unui proces ce ascunde şi subtilităţi, nu permanent convenabile, cel puţin din câteva puncte de vedere, specificate anterior. De asemenea, proiectele pe domenii critice nu pot fi rulate pe aceste baterii, adăugând un punct de eşec sensibil în hardware. Pentru aceşti utilizatori, confidenţialitatea netlist-ului depinde doar de complicaţiile formatului bitstream-ului cu adăugarea de către utilizator a unor nivele ascunse. Securitatea în schemele obscure este cunoscută ca fiind, într-o măsură mai mult sau mai puţin specifică, dintr-un punct de vedere  teoretic şi practic, plină de erori. Însă, este cunoscut în general faptul că ingineria inversă necesară pentru a readuce netlist-ul  dintr-un bitstream compilat nu este un lucru banal pentru un FPGA.

HDL e folosit la crearea a ceea ce se integrează în aplicația embedded, cea care este condusă prin sistemul de operare. Deci, sistemul de operare embedded Linux este ca un fel de interfață de folosire a aplicațiilor proiectate pe platforme FPGA cu limbaje de proiectare hardware.

În [3] se arată că limbajele de descriere hardware (HDL) permit implementarea unităților **Intellectual Property (IP),** numite  **IP cores.** Acestea sunt componente reutilizabile care descriu și implementează functionalități hardware. Pot fi folosite de mai multe ori în același proiect sau în proiecte diferite.

IP core includ 3 categorii:

* Hard core
* Firm core
* Soft cores

**Hard IP cores** implică implementarea la nivel *silicon-level circuit î*n aria reconfigurabilă FPGA ( terminologie utilizată: *FPGA fabric*).

**Soft IP cores** reprezintă elemente de design care pot fi implementate în aria FPGA, fiind realizate în limbaj HDL (VHDL sau Verilog).

**Firm IP cores** au un grad mai mare de optimizare și sunt proiectate pentru un anumit circuit sau arhitectură. Sunt mai puțin portabile decât variantele soft core.

Un exemplu de soft IP core îl constituie implementarea procesoarelor prin folosirea componentelor FPGA. Proiectarea care utilizează soft IP core prezintă o serie de avantaje, printre care și refolosirea unităților de proiect (design reuse), simplificarea proiectării, putând fi updatat și modificat un proiect, dar și folosirea modularității într-un design.



Figura 2.3. Exemple de soft IP cores [3]

În capitolul următor va fi prezentat un exemplu de soft IP core și anume, MicroBlaze de la firma Xilinx.

# CAPITOLUL 3. Arhitectura de ansamblu a uneltelor Și sistemelor dedicate

## 3.1. Descriere generală

Kit-ul de instalare Xilinx Embedded Development Kit (EDK) prezentat în [9] permite proiectarea unui sistem cu procesor complet dedicat cu scopul de a-l implementa într-un dispozitiv FPGA de la firma Xilinx.

EDK este o componentă a Mediului Integrat Software (ISE) din Ediția de Sisteme Dedicate și Pachete de Proiectare. ISE este un produs dezvoltat de Xilinx  pentru sisteme care au la bază FPGA, fiind necesar la implementarea proiectelor într-un dispozitiv logic programabil. EDK include:

* Pachetul de unelte pentru sistem - „Platform Studio”, furnizat de Xilinx (XPS) se ocupă cu dezvoltarea procesoarelor hard dedicate.
* Kitul de dezvoltare software (SDK), bazat pe o sursă deschisă framework de la Eclipse, care se folosește pentru dezvoltarea aplicațiilor software dedicate. SDK este disponibil de asemenea și ca program autonom.
* Nuclee dedicate pentru procesarea Proprietății Intelectuale (IP) ce includ procesoare și periferice. În timp ce mediul EDK suportă proiecte de creare și implementare, calea recomandată este de a începe într-un proiect ISE, apoi adăugându-se un procesor sursă dedicat în proiectul ISE.

EDK depinde de componentele ISE pentru sintetizarea proiectului microprocesorului hard, pentru a mapa acel proiect spre o tință FPGA și pentru a genera și descărca bitstream-ul.

 Figura 3.1. Proiectarea sistemelor dedicate folosind FPGA

## 3.2. Prezentarea generală a procesului de proiectare

Uneltele furnizate de EDK sunt concepute pentru a asista la toate etapele procesului de proiectare a sistemelor dedicate, prezentate în figura 3.2.

****

Figura 3.2. Fluxul de proiectare a sistemelor dedicate [9]

### 3.2.1. Dezvoltare de tip hardware

Tehnologia FPGA de la Xilinx permite personalizarea părții logice de tip hardware din subsistemul procesorului. O asemenea personalizare nu este posibilă cu microprocesoarele sau controlerele standard. Termenul de „Platformă hardware” descrie subsistemul dedicat, flexibil de procesare creat cu ajutorul tehnologiei Xilinx pentru nevoile aplicației. Platforma hardware constă în unul sau mai multe periferice și procesoare conectate la magistralele procesorului. XPS evidențiază descrierea acestei platforme în fișierul de “Specificații Hardware ale Microprocesorului „(MHS). Acest fișier este fișierul sursă principal care menține descrierea platformei hardware și reprezintă în cod ASCII componentele hardware ale sistemului dedicat. Atunci când descrierea platformei este completă, platforma poate fi dată în folosință lui SDK [9].

### 3.2.2. Dezvoltare de tip software

Un Pachet de Suport pentru Placă (BSP) este o colecție de drivere software și, opțional, sistemul de operare pe care urmează să fie construită aplicația. Imaginea software-ului creat conține doar părțile din biblioteca Xilinx de care a fost nevoie în cadrul sistemului dedicat. Se pot crea aplicații multiple care să ruleze pe BSP. Platforma hardware trebuie să fie importată în SDK înainte de crearea aplicațiilor software și a BSP-ului.

### 3.2.3. Verificare

 EDK oferă atât unelte de verificare software, cât și hardware.

### A.Verificarea hardware folosind simularea

Pentru a verifica dacă funcționalitatea platformei hardware este corectă, trebuie creat un model de simulare și apoi rulat pe un simulator care utilizeză limbajul HDL( Hardware Design Language). Când se simulează sistemul, procesorul/procesoarele execută programele software. Se poate alege un model de simulare comportamentală, structurală sau de sincronizare exactă.

Simulatorul ISE(ISim) suportă simularea design-urile dedicate. Atunci când se creează un proiect în ISE și se adaugă sursa unui proiect dedicat, se poate lansa ISim din cadrul uneltei de dezvoltare ISE. În cazul în care nu se utilizeză niciun proiect ISE, se poate lansa programul ISim direct din Platform Studio.

### B. Verificarea software folosind opțiunea „debug”

Pentru verificarea software sunt disponibile următoarele opțiuni:

* Se poate încărca design-ul pe o placă de dezvoltare suportată și se folosește o unealtă debug pentru a controla procesorul țintă;
* Se poate măsura performanța sistemului prin diagnosticarea execuției codului.

### 3.2.4. Configurarea dispozitivului

Atunci când platformele hardware și software sunt complete, se poate configura bitstream-ul pentru dispozitivul țintă FPGA.

* Pentru realizarea prototipurilor, se descarcă bitstream-ul împreună cu orice software necesar pentru rularea pe platforma dedicata în timp ce este conectată la calculatorul gazdă;
* În faza de producție, se stochează configurarea bitstream-ului și a software-ului într-o memorie nevolatilă conectată la FPGA.

## 3.3. MicroBlaze (Xilinx)

### 3.3.1. Introducere în lumea “MicroBlaze”

Selectarea unui procesor de tip soft core pentru implementarea într-un FPGA ține cont de [3]:

* costul implementării;
* puterea consumată;
* design-ul și uneltele de dezvoltare;
* considerente referitoare la sistemul de operare;
* opțiuni de debug și capabilități.

MicroBlaze este un tip de procesor soft core, bazat pe o arhitectură Harvard flexibilă care utilizează un set optimizat de instrucțiuni RISC pe 32 de biți pentru aplicații dedicate, oferind sistemul de procesare dorit la cel mai mic preț posibil.

Microblaze are la dispoziție peste 70 de opțiuni de reconfigurare ce pot fi folosite de către utilizator, permițînd folosirea oricărui procesor, de la o mașină de stare de dimensiuni foarte mici sau un microcontroler la un sistem bazat pe microcontroler cu performanțe înalte ce lucrează în Linux. Acesta operează într-un mod tip pipeline pe trei nivele pentru optimizarea dimensiunii sau mod tip pipeline pe cinci nivele ce furnizează performanțe mai bune pentru circuitele DMIP față de celelelte soluții bazate pe procesoare soft cu FPGA.



Figura 3.3. Procesorul MicoBlaze pentru circuite FPGA de tip Virtex II PRO, Virtex 4, Virtex 5, Spartan 3 și Spartan 3E

Configurabilitatea este considerată centrul flexibilității procesorului MicroBlaze, însă nu este necesară memorarea manualului pentru a învăța modul lui de configurare. Fereastra principală pentru Configuration Wizard a procesorului MicroBlaze, inclusă în Platform Studio de la Xilinx, oferă selectarea celor 6 tipuri de modele de procesoare disponibile.

Fereastra de configurare furnizează imediat un feedback printr-un mediu de afișare în legatură cu utilizarea resurselor și a performanțelor, utilizând modul wizard pentru setarea rapidă sau modul avansat pentru accesul la detaliile legate de nivelele de bază.



##### Figura 3.4. MicroBlaze Configuration Wizard [12]

Pentru automatizarea procesului de personalizare al kernel-urilor uClinux folosite în sistemele MicroBlaze, trebuie descărcat pachetul uClinux BSP de la Petalogix  de la adresa http://www.petalogix.com/resources/downloads/uclinux-bsp.

Pachetul uClinux BSP este folosit pentru a genera fișierul auto-config.in ce definește toți parametrii software necesari în compilarea kernel-ului uClinux.

Mai jos este evidențiat un exemplu al structurii necesare pentru mediul Linux:

/opt/xilinx/edk8.1                                    Directorul de instalare pentru EDK 8.1

/opt/xilinx                                              Directorul părinte pentru EDK8.1 existent

/opt/xilinx/edk\_user\_repository               Directorul de utilizator EDK (include fișierele de suport pentru uClinux BSP)

### 3.3.2. Configurarea mediului Xilinx

Mediul de instalare Xilinx este necesar pentru a dezvolta un proiect hardware cu instrumente de la firma Xilinx. În continuare sunt prezentați pașii necesari pentru configurarea mediului Xilinx [13].

* Click dreapta mouse pe Desktop și selectare “Open Terminal” pentru deschiderea terminalului.
* În terminal, se scrie scriptul necesar pentru setarea sursei de la Xilinx:

[host] $ source <Path to the installed Xilinx ISE>/settings32.sh

[host] $ source <Path to the installed Xilinx EDK>/settings32.sh

Se utilizează settings32.sh în cazul derivatelor bash și settings32.csh pentru C shell.

* Confirmarea setării variabilelor mediului Xilinx:

[host] $ set

Următorul text ar trebui să apară în consolă:

XILINX=<Path to the installed Xilinx ISE>

XILINX\_EDK=<Path to the installed Xilinx EDK>

* Trebuie asigurat faptul că este permisă citirea și scrierea în driver-ul JTAG deoarece acesta este necesar pentru programarea FPGA-ului. Următoarea comandă permite tuturor utilzatorilor să citească și sa scrie în driverul JTAG:

 [host] $ sudo chmod a+wr /dev/windrvr6

Cerințele minime necesare construirii unei platforme hardware care utilizează uClinux sunt:

* procesor soft MicroBlaze;
* timer;
* controler de întreruperi;
* dispozitive standard de intrare și ieșire;
* controler pentru memoria externă
* ROM – memorie non-volatilă (opțională); exemplu: Flash;
* RAM – memorie accesibilă de date precum home dir; exemplu: DDR-SDRAM .

### 3.3.3. Transferul sistemului de operare Linux pe MicroBlaze

 Modelul original uC-Linux este derivat din kernel-ul sistemului de operare Linux 2.0 destinat microprocesoarelor bazate pe sisteme fără unitatea de gestiune a memoriei (Memory Management Unit - MMU)**.** Deșiversiunea actuală de uC-Linux 2.6 suportăunitățile de gestiune a memoriei, în acest proiect nu le voi folosi în procesul de implementare. Figura 3.5 descrie pașii necesari pentru mutarea kernel-ului Linux pe procesorul soft MicroBlaze.



Figura 3.5.Mutarea kernul-ului Linux pe procesorul MicroBlaze

# CAPITOLUL 4. PLATFORMA Z4800

## 4.1. Noțiuni generale

În arhitectura calculatoarelor, există o nevoie tot mai mare de a evalua performanțele și costul unor idei noi, complexe din punct de vedere structural. În prezent, majoritatea evaluărilor se realizează cu ajutorul simulărilor, dar aceste simulatoare sunt destul de lente. Este dificil de obținut rezultate precise cât mai rapid. În consecință, au fost dezvoltate diverse cercetări privind accelerarea hardware-ului în cazul acestor simulatoare folosind FPGA [10].

Simulatoarele accelerate pentru FPGA nu au devenit populare în comunitatea de cercetare a arhitecturii calculatoarelor. Astfel, au fost necesare mai multe platforme FPGA pentru a le face „remarcate”. Aceste sisteme ar trebui să ofere o soluție completă: ușurință în utilizare, modificare, depanare și verificare, dar și o configurare automată pentru experiment. Pentru a aborda această necesitate, voi prezenta platforma de explorare bazată pe arhitectura FPGA-ului și anume, Z4800.

Z4800 nu este gândită ca o soluție la înlocuirea platformelor existente de simulare accelerate cu FPGA: pentru tipuri diferite de studii de cercetare, dar și pentru obiective educaționale, se vor folosi unelte specifice. Platforma Z4800 se axează pe testarea și analiza sistemelor cu un singur nucleu sau a sistemelor multi-nucleu de dimensiune mai mică (2-10 nuclee), în detrimentul analizării sistemelor de dimensiuni mari. Se poate încărca sistemul de operare nemodificat Linux din sistemul de fișiere de rețea precum NFS sau din unitatea de stocare locală USB.

Etapele principale ale proiectării platformei Z4800 se bazează pe ușurință la adaptare și utilizare. Este susținută de un sistem complet de sinteză și testare a performanțelor care furnizează un mecanism cu ajutorul căruia se poate rula un număr mare de experimente fără intervenții umane. De asemenea, se poate realiza verificarea interfețelor și se oferă un timp mai mare pentru debug. În concluzie, Z4800 este o platformă complet bazată pe circuit de tip FPGA care poate fi folosit nu numai pentru cercetarea arhitecturii unui calculator, dar și în cercetarea sistemelor de operare și în depanarea programelor software.

 Platforma Z4800 este sintetizabilă pe multe circuite de tip FPGA de la Altera; în cazul circuitului Altera DE2-115, o placă FPGA de 300$ bazată pe un cip ieftin Cyclone IV-E, un sistem cu 4 nuclee poate fi foarte practic.

## 4.2. Proiectare

Design-ul este modularizat, astfel încât componentele independente să poată fi înlocuite și reproiectate pentru a testa noi idei arhitecturale. Mai multe numărătoare utilizate în calculul performanțelor (actualmente 80) sunt folosite în măsurarea simultană a unei game largi de elemente de statistică în timp real. De asemenea, design-ul furnizează un sistem automatizat de sinteză a părții hardware și de testare a performanțelor, dar și o interfață extinsă pentru depanare (debug). Acest design este bazat pe simplitate, usurință la utilizare și adaptabilitate.

Z4800 preia o filozofie de proiectare unică. Ideea centrală este că funcționalitatea și ușurința proiectării sunt mai importante decât celelalte criterii de proiectare (arie, viteză, putere, etc.).

În locul focusării pe maximizarea performanțelor pe bloc logic și micro-optimizare, Z4800 folosește o construcție de tip RTL (register-transfer level - nivelul de transfer al registrilor) de nivel înalt. Aria sintetizată este schimbată cu un RTL mai ușor care conține un circuit de modularitate. Design-ul rezultant funcționează la perioade de ceas moderate, însă poate fi modificat ușor pentru studiul îmbunătățirilor arhitecturale variate.

Blocuri precum controlerul pentru memoria cache sau TLB (translation lookaside buffer) sunt distribuite în entități separate VHDL (VHSIC (very high speed integrated circuit) hardware description language) astfel încât acestea pot fi rescrise și înlocuite în mod independent. Cu toate acestea, modularizarea nu se extinde mult dincolo de acest nivel. Acest lucru este benefic deoarece modularizarea peste un anumit nivel ar eclipsa partea de proiectare. Trebuie să se ia în considerare partea scrisă de utilizator în limbajul HDL ca fiind parte a procesului de proiectare: o modularizare prea mică sau prea mare poate face ca proiectarea să devina greu de înțeles. Timpul și efortul implicat în optimizarea manuală a limbajului HDL ar putea reprezenta diferența dintre o parte de proiectare care funcționează și una care nu funcționează. În plus, având la dispoziție un design modularizat corespunzător, se poate reveni la optimizarea părților-problemă ale design-ului odată ce corectitudinea deplină este realizată.

## 4.3. Instrucțiunile load-linked și store-conditional

Instrucțiunile de încărcare și stocare condiționată descrise în [10] și folosite de MIPS, furnizează o cale eficientă de implementare a secțiunilor critice de citire, modificare și scriere din codul de “multithreading” (presupune execuția în paralel a mai multor procese, pentru a îmbunătăți eficiența sistemului). Acestea sunt folosite la blocarea nucleului sistemului de operare, la fel ca și primitivele de așteptare în bucla de la Linux. Aceste instrucțiuni au nevoie de un suport special în memoriile cache de nivel 1.

* Încărcare-conectare: cu un cuvânt de încărcare la fel ca cel normal, însă cu o parte suplimentară de contorizare a evidenței în sistem. La Z4800, apar două instrucțiuni suplimentare:
* Dacă linia de referință din cache nu este în starea „MODIFIED”, atunci ea avansează. Acest procedeu poate declanșa o interogare nedorită care duce la obținerea unei linii din memoria cache cu proprietate exclusivă (cu exclusivitate).
* Căutarea adresei fizice efective a instrucțiunii de referință. O stocare condiționată ulterioară va eșua dacă adresa fizică nu se potrivește cu adresa salvată în această etapă.
* Stocarea condiționată: Se stochează un cuvânt. Stocarea are loc doar dacă partea hardware poate garanta că datele de la adresa efectivă nu au fost modificate de orice alt procesor de la ultima operație de încărcare-conectare. Dacă această încărcare are loc, valoarea „1” este scrisă în registru; altfel apare valoarea „0”. La Z4800, se aplică următoarele condiții:
* Stocarea condiționată va avea loc în starea „MODIFIED” a memoriei cache. Cum ultima operație de stocare-conectare a forțat intrarea în starea „MODIFIED”, stocarea condiționată va acționa asupra memoriei cache. Însă, dacă nu se întamplă așa, stocarea condiționată va eșua imediat și va returna valoarea „0”. Acest lucru se produce în cazul în care două procesoare vor intra în secțiunea corespunzătoare de stocare-conectare în același timp.
* Adresa fizică efectivă a instrucțiunii de stocare condiționată trebuie să corespundă celei de stocare-conectare anterioare. Mai mult, toate operațiile ce implică memoria cache vor anula această adresă stocată; orice operație dintre intrucțiunile de stocare-conectare și stocare condiționată va garanta eșecul stocării condiționate.

Condițiile de mai sus sunt suficiente pentru garantarea faptului că secțiunea critică există în toate procesoarele. Datorită modului în care este implementată memoria cache la Z4800, acest lucru este ușor de înțeles.

Automatizarea se realizează prin rularea unei comenzi paralele pentru procesele de pe sistemul gazdă cu scopul de a iniția sinteza .

  De fiecare dată când programarea pe FPGA este completă , se încarcă automat kernel-ul de pe memoria Flash internă, pentru ca apoi să fie executat direct scriptul benchinit. Acest script aduce un mediu minimal Unix, și execută fiecare test de performanță enumerat în benchlist.pl. Un program ajutător , z48perf , care se încadrează în executarea testului,  trebuie să citească datele obținute de la contoarele parametrilor de performanță și să le transpună în fișierele pentru  rezultate. Fișierele rezultate sunt scrise folosind NFS (Network FileSystem ) prin  Ethernet la PC-ul gazdă .

# CONCLUZII

Scopul acestui proiect a fost explorarea arhitecturii și a implementării kernel-ului uClinux, dar și a instrumentelor sale de dezvoltare. Astfel, s-a studiat procesorul soft MicroBlaze de la firma Xilinx.

Modelul uClinux nu este foarte bine documentat. Interfețele au fost schimbate datorită lipsei unității de gestiune a memoriei MMU, iar derivarea diferitelor părți din kernel au dus la anumite schimbări în funcționalitatea de bază. Acest lucru înseamnă că toată documentația pentru Linux nu va putea fi folosită și pentru uClinux. În general, acest lucru necesită mai multă atenție din partea producătorilor.

Platforma și uneltele uClinux formează o fundație gratuită și bine venită în procesul de dezvoltare. În mod tradițional, Linux deține o platformă stabilă, cu un suport excelent pentru rețea. Acesta este cazul și uClinux-ului care este o soluție bună ca sistem de operare furnizând conectivitate și administrare pentru dispozitive. Platforma de dezvoltare uClinux este prevăzută cu un set eficient de unelte, bazate pe uneltele de dezvoltare GNU. Acești factori, în general, ajută la realizarea unui proces de fabricație mai ușor și mai rapid.

uClinux va avea nevoie ca dezvoltarea să fie făcută ținându-se cont de îmbunătățirile aduse părții software și kernel-ului. Este esențial ca o mașină industrială care furnizează conectivitatea rețelelor să aibă cele mai bune actualizări în materie de securitate.

# BIBLIOGRAFIE

 [1] Nicola Bombieri, Franco Fummi and Graziano Pravadelli, "Hardware Design Languages and Simulation", 2006

 [2] Santanu Chattopadhyay, „Embedded System Design”, PHI Learning Private Limited, 2013

[3] Ioana.Dogaru, Curs “Sisteme Reconfigurabile de Calcul”, 2013

[4] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk and P.Y.K. Cheung, „Reconfigurable Computing: Architectures, Design Methods, and Applications ”, Computers and Digital Techniques, IEE Proceedings Vol.152, pp.193-207, 2005

 [5] Scott Hauck and Andre Dehon, “Reconfigurable Computing”, Elsevier,2008

[6] Ioana Dogaru, “Exemplificarea fluxului de proiectare, 2013

[7] Jean-Baptiste Note and Éric Rannaud, „From the bitstream to the netlist”, 2008

[8] Ron Sass and Andrew G. Schmidt, “Embedded Systems Design with Platform FPGAs”, 2011

[9] "Embedded System Tools Reference Manual EDK"- Xilinx

[10] Will Simoneau and Resit Sendag, "An FPGA-based Multi-Core Platform for Testing and Analysis of Architectural Techniques ", Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island

[11] <http://embeddedcraft.org/embedlinuxdesktoplinux.html>

[12] <http://www.xilinx.com/tools/microblaze.htm>

[13] PetaLogix / XUP Professors’ Workshop, „Embedded Linux for the Xilinx MicroBlaze Soft Processor”- Lab Manual, V11.3 XUPV5 Development Board

[14] JasonWu, Irfan Syed & Dr. JohnWilliams, „MicroBlaze uClinux Creating a Simple Clinux ready MicroBlaze Design”, 2006

[15] <http://www.oocities.org/richardfuu/rtlinux.html>

[16] <http://www.comptechdoc.org/os/linux/howlinuxworks/linux_hlprocess.html>

[17] Kimmo Nikkanen, “UCLINUX AS AN EMBEDDED SOLUTION - Bachelor's Thesis ”, 2003

[18] <http://www.linuxjournal.com/article/4678>

[19] <http://stst.elia.pub.ro/news/SOIII2009/labs/>

[20] <http://www.linuxjournal.com/article/7221>