MINISTERUL EDUCAŢIEI NAŢIONALE
UNIVERSITATEA TRANSILVANIA DIN BRAŞOV
BRAŞOV, EROILOR NR. 29, 500036, TEL. 0040-268-413000,
FAX 0040-268-410525

Universitatea Transilvania din Brasov

Școala Doctorală Interdisciplinară

Facultatea: Inginerie Electrică și Știința Calculatoarelor

Ing. Szabolcs HAJDU

Contribuții la implementarea în timp real a diferitelor sarcini de control în circuite FPGA prin reconfigurare parțială

Contributions to real-time implementation of different control tasks in FPGAs by partial reconfiguration

Rezumat/Abstract

Conducător ştiinţific
Prof.dr.ing. Iuliu SZEKELY

BRASOV, 2019

D-lui (D-nei) .....

COMPONENŢA

Comisiei de doctorat

numită prin ordinul Rectorului Universităţii Transilvania din Braşov

Nr. 9824 din 08.04.2019

PREŞEDINTE:	Conf.dr.ing. Carmen GERIGAN
	Universitatea Transilvania din Braşov
CONDUCĂTOR ŞTIINŢIFIC:	Prof.dr.ing. Iuliu SZEKELY
	Universitatea Transilvania din Braşov
REFERENŢI:	Prof.dr.ing. Alexandru SĂLCEANU
	Universitatea Tehnică „Gheorghe Asachi” din Iași
	Prof.dr.ing. Radu MUNTEANU
	Universitatea Tehnică din Cluj-Napoca
	Prof.dr.ing. Petre Lucian OGRUȚAN
	Universitatea Transilvania din Brașov

Data, ora şi locul susţinerii publice a tezei de doctorat: 22.06.2019, ora 10,00, sala N II 1.

Eventualele aprecieri sau observaţii asupra conţinutului lucrării vă rugăm să le transmiteţi în timp util, pe adresele:

hajdu.szabolcs@unitbv.ro sau diana.thierheimer@unitbv.ro

Totodată vă invităm să luaţi parte la şedinţa publică de susţinere a tezei de doctorat.

Vă mulţumim.

Cuvânt înainte

Circuitele reconfigurabile FPGA au trecut printr-o revoluție mare din anii 2000, când au apărut pentru prima dată, datorită cărora dezvoltarea și implementare circuitelor digitale complexe a devenit mai ușoară și mai simplă. Tradițional circuitele implementate au fost dezvoltate în limbajele HDL de exemplu VHDL sau Verilog, dar în timp s-au dezvoltat metode pentru generarea circuitelor în Matlab Simulink sau în limbajul C. În lucrarea de licență și în disertația de masterat m-am ocupat cu diferite circuite de control implementate în circuitele FPGA. În cursul acestor studii și cercetări a devenit problematică dezvoltarea circuitelor adiționale, necesare monitorizării și testării circuitelor din mers. Din acest motiv scopul următor al cercetărilor mele a fost crearea unui proiect cadru care combină metodele de configurare, monitorizare și testare într-un mod care simplifică dezvoltarea circuitelor de control pentru sarcini complexe.

În teza de doctorat se vor prezenta diferite moduri pentru implementarea și configurarea circuitelor FPGA, completată cu tehnologia de reconfigurare parțială, care dă șansa reconfigurării circuitelor FPGA din mers. Se vor prezenta avantajele și dezavantajele diferitelor tehnici, pe baza studierii și sistematizării tehnologiilor dezvoltate de cei mai importanți producători de circuite FPGA. Aplicația de bază pentru demonstrarea posibilităților de configurare și reconfigurare este un sistem dezvoltat pentru controlul complex al unui vehicul aerian de tip quadcopter, mai precis cu scopul de a controla poziția quadcopterului.

Aș dori să exprim mulțumiri în primul rând d-lui prof. dr. ing. Székely Iuliu care m-a îndrumat, sprijinit și ajutat pe toată durata elaborării tezei în calitate de conducător ştiinţific.

Mulțumesc dr. ing. Bakó László și dr. ing. Brassai Sándor Tihamér pentru sprijinul acordat în rezolvarea problemelor pe toată durata elaborării tezei și îndrumarea științifică la structurarea, dezvoltarea și prezentarea articolelor publicate.

În final aș dori să mulțumesc membrilor familiei mele pentru încurajare, răbdare și suport.

Cuprins

Table of contents

Listă de abrevieri

ALU - Arithtmetical Logical Unit
API - Application Programming Interface
ARM - Advanced RISC Machine
ASIC - Application Specific Circuit
AXI - Advanced Microcontroller Bus
BRAM - Block Random Access Memory
BUFG - Global Clock Buffer
BUFR - Regional Clock Buffer
CB - Connection Block
CLB - Configurable Logic Block
CORDIC - Coordinate Rotation Digital Computer
CPI- Cycles per Instruction
CPU - Central Processing Unit
CRC -Cyclic redundancy check
CU - Control Unit
DCM - Digital Clock Manager
DCP -Design Checkpoint
DMA - Direct Memory Access
DRC -Design Rule Checking
DSP - Digital Signal Processing
EDK - Embedded Development Kit
EPROM - Erasable Programmable Read-only Memory
EMMC - Embedded Multicore MicroController
FF - Flip-Flop
FPGA - Field Programmable Gate Array
GSR - Global Set/Reset
GUI - Graphical User Interface
HDL - Hardware Description Language
HLS - High Level Synthesis
HPR - Hybrid Partial Reconfiguration
ICAP - Internal Configuration Access Port
IDE - Integrated development environment
IMU - Inertial Measurement Unit
IOB - Input-Output Block
IP - Intellectual Property Core
JTAG - Joint Test Action Group
LSB - Least Significant Bit

LUT - Look-up table
MEMS - Micro Electro Mechanical System
MGT - Multi-Gigabit Transceiver
MMCM - Mixed Mode Clock Manager
MSB - Most Significant Bit
OF - Optical Flow
OOC - Out Of Context Synthesize
PC - Personal Computer
PCAP - Processor Configuration Access Port
PD - Partition Definition
PID - Proportional–Integral–Derivative Controller
PLL - Phase Locked Loop
PR - Partial Reconfiguration
PRM - Partially Reconfigurable Module
PU - Programmable Unit
PWM - Pulse Width Modulation
RC -Radio Command
RM - Reconfiguration Module
RTL - Register Transfer Level
QoR - Quality of Reconfiguration
SB - Switch Box
SCC - Single Cycle Computer
SDM - Secure Device Manager
SLL - Super Long Lines
SLP- Sapentia Lab Processor
SLR -Super Logic Region
SoC - System on Chip
SRAM - Static Random Access Memory
SRL - Shift Register LUT
TBUF - Three State Buffer
UAV -Unmanned aerial vehicle
VHDL - Very High Speed Integrated Circuit Hardware Description Language
WNS -worst negative setup slack

Introducere

Datorită dezvoltării rapide a circuitelor FPGA (Field Programmable Gate Array) acestea sunt folosite din ce în ce mai mult în aplicațiile industriale și comerciale. Cu ajutorul circuitelor FPGA dezvoltarea aplicațiilor logice de control devine mai rapidă și mai optimizată, implementarea este direct la nivel hardware. Un circuit FPGA este mai flexibil decât un circuit de tip ASIC, care are o arhitectură fixă și poate fi folosit doar pentru sarcinile pentru care a fost realizat în timpul dezvoltărilor înainte de fabricație.

Folosind circuitele FPGA algoritmii de control sunt dezvoltați într-un limbaj de programare pentru descrierea hardware, cum ar fi VHDL sau VERILOG, respectiv folosind mediile de dezvoltare mai avansate circuitele pot fi descrise în C sau MATLAB, apoi se compilează pentru circuitul FPGA, după care algoritmul poate fi descărcat și testat imediat. Beneficiul evident al utilizării unor dispozitive reconfigurabile, cum ar fi FPGA, este că funcționalitatea de care dispune un dispozitiv la un moment dat poate fi modificată și actualizată ulterior. Deoarece sunt disponibile funcționalități suplimentare sau devin necesare îmbunătățiri ale proiectului, circuitul FPGA poate fi complet reprogramat cu o nouă logică. În mai multe situații acest lucru nu este suficient. Ce se întâmplă dacă logica într-o parte a unui FPGA trebuie să fie schimbată, fără a perturba întregul sistem? De exemplu, la un proiect compus din mai multe blocuri logice și, fără a perturba sistemul și a opri fluxul de date, trebuie actualizată funcționalitatea într-un singur bloc. În astfel de situație este necesară o modalitate de reconfigurare parțială a aplicației pe dispozitivul FPGA.

În acest scop se folosește tehnica de reconfigurare parțială, care este un proces de proiectare, care permite reconfigurarea unei porțiuni limitate, predefinite a unui FPGA, în timp ce restul dispozitivului continuă să funcționeze. Acest lucru este deosebit de util în cazul în care dispozitivele funcționează într-un mediu critic, în care procesul nu poate fi întrerupt, în timp ce unele subsisteme se redefinesc. Folosind reconfigurarea parțială, este posibilă creșterea în mod dramatic a funcționalității unui singur FPGA, permițând utilizarea unor dispozitive mai mici, decât ar fi fost nevoie altfel. Aplicații importante pentru această tehnologie includ sistemele de comunicații reconfigurabile și sistemele criptografice. Un sistem bazat pe circuitele FPGA integrat cu tehnica de reconfigurare parțială este o platformă bună pentru implementarea circuitelor de control complexe pentru diferite sarcini industriale.

Obiectivele tezei

Primul obiectiv al, tezei este testarea diferitelor abordări pentru dezvoltarea algoritmilor de control în circuite FPGA. În acest scop se prezintă diferite metode de dezvoltare. Spre sfârșitul lucrării se va prezenta un proiect cadru bazat pe un sistem de operare. Pentru testarea circuitelor, a algoritmilor dezvoltați, proiectul final va fi aplicat pe un vehicul aerian de tip quadcopter. Algoritmii dezvoltați cu diferite metode de proiectare (VHDL, Simulink, HLS) vor fi folosite pentru stabilizarea unghiurilor de înclinare ale quadcopterului.

A doilea obiectiv este integrarea tehnicii de reconfigurare parțială în proiectul dezvoltat, testarea sistemului dat și dezvoltarea unei platforme universale, care poate să fie modificată cu ușurință pentru alte sarcini de control. Această platformă este bazată pe un sistem de operare încorporat Linux, care controlează și monitorizează reconfigurarea parțială și circuitele implementate în partea reconfigurabilă a FPGA.

Structura tezei

Lucrarea prezentă este compusă din șase capitole principale. Primul capitol a acestei lucrări prezintă circuitele FPGA: componentele principale, structura, metoda de proiectare cu limbajul VHDL și Verilog. Se prezintă două procesoare dezvoltate în VHDL, implementate în FPGA, dedicate pentru implementarea algoritmilor de control.

Al doilea capitol prezintă gradual metoda de implementare a unor logici în FPGA prin utilizarea Xilinx System Generator bazat pe Matlab Simulink și dezvoltarea unui algoritm de control PID în acest mediu de dezvoltare. Circuitul dezvoltat va fi folosit pentru stabilizarea unghiurilor de înclinare ale unui quadcopter.

Capitolul al treilea introduce mediul de dezvoltare HLS, în care algoritmul de control este dezvoltat în limbajul de programare C. Se prezintă o metodă pentru detectarea deplasării în imagini captate cu camere video, utilizând filtrul Canny. Metoda este implementată direct în circuitul logic FPGA. Modulul de detecție a deplasării va fi folosit în continuare la quadcopter, pentru a da acestuia o autonomie mai mare.

Al patrulea capitol este dedicat analizei și prezentării metodelor de reconfigurare a circuitelor FPGA. Se prezintă un studiu asupra metodologiei de reconfigurare parțială folosită de firmele Xilinx și Altera. În finalul capitolului se prezintă un proiect bazat pe circuitul Xilinx Zynq și se vor prezenta avantajele și dezavantajele variantelor analizate.

Capitolul al cincilea prezintă un proiect bazat pe tehnologia de reconfigurare parțială care a fost dezvoltat pentru aplicația de control al unui quadcopter. Se prezintă algoritmii de control, metoda de proiectare, implementarea la nivel de hardware și rezultatele obținute. În final se prezintă un proiect independent care poate fi utilizat pentru implementarea diferitelor circuite de control.

Lucrarea se încheie cu capitolul al șaselea de concluzii finale, contribuții originale, diseminarea rezultatelor cercetării și direcții de cercetare viitoare.

Capitolul 1

Circuite Field Programmable Gate Array - FPGA

Un FPGA (*Field Programmable Gate* Array) [1] este un circuit integrat digital configurabil de către utilizator, după ce a fost fabricat. Descrierea algoritmilor pentru implementarea în circuite FPGA se realizează în general cu ajutorul unui limbaj de descriere hardware (HDL) similar cu limbajul folosit în circuitele de tip ASIC [2].

Circuitele FPGA sunt capabile să profite de avantajele procesului de dezvoltare în mediul hardware și în mediul software. Algoritmii implementați în hardware au un avantaj considerabil din punctul de vedere al consumului și performanțe comparabile cu algoritmii implementați în software. În circuitele FPGA, spre deosebire de circuitele ASIC, circuitele dezvoltate de către utilizator sunt încărcate, spre deosebire de circuitele ASIC, care sunt fabricate cu arhitectura fixă. Acest fapt înseamnă că circuitul FPGA poate fi configurat și reconfigurat de mai multe ori. Astfel un singur circuit devine capabil de realizarea sarcinilor specifice ale mai multor circuite integrate dedicate [3].

Pe lângă avantaje apar și unele dezavantaje. Circuitele FPGA pot fi programate cu ușurință dar crearea unui circuit optimizat și construirea acestuia corect este o sarcină complexă spre deosebire de programarea unui microcontroler. În general, în comparație cu circuitele ASIC, circuitele FPGA au o performanță de 5-25x mai slabă din punctul de vedere al consumului, spațiului și întârzierea semnalelor. Pe de altă parte dezvoltarea unui algoritm în circuite ASIC poate dura ani de zile și cu costuri foarte mari, realizarea algoritmului în circuite FPGA poate dura câteva zile cu un buget semnificativ mai mic. În sistemele unde nu este importantă realizarea performanței maxime sau minimizarea consumului de energie electrică, un circuit FPGA reprezintă o alternativă simplă față de circuitele ASIC.

Figura 1.1 prezintă diagrama bloc simplificată a unui circuit FPGA, elementele de bază ale acestui circuit fiind blocurile logice încorporate într-o matrice de rutare. Blocurile logice conțin unități de procesare pentru executarea operațiilor logice combinaționale și circuite logice bistabile (flip-flop) pentru implementarea operațiilor logice secvențiale.

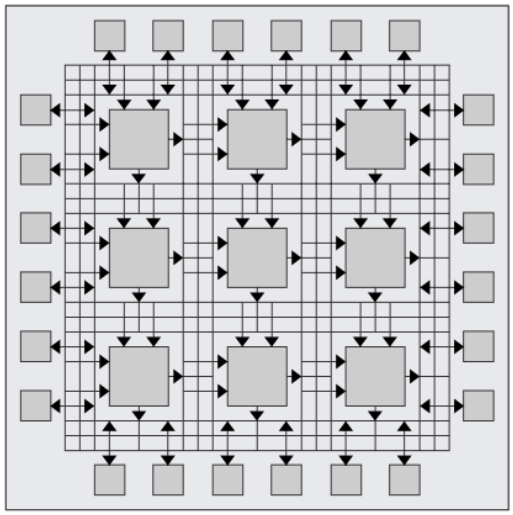

Figura 1.1 Diagrama bloc simplificată a unui circuit FPGA [3]

În majoritatea cazurilor blocurile logice sunt memorii simple prin urmare sunt capabile de implementarea oricărei funcții booleene cu 5-6 intrări. Matricea de rutare poate fi programată liber, astfel orice celulă logică este conectabilă cu ușurință cu alte celule logice. Datorită acestei flexibilități circuitele FPGA sunt capabile de implementarea de algoritmi complecși. Circuitele FPGA actuale sunt capabile de implementarea unor funcții logice complexe construite din milioane de porți logice cu o frecvență de ceas de sute de MHz. Pentru creșterea vitezei circuitele FPGA pot conține pe lângă unitățile logice și alte circuite cum ar fi: blocuri de memorie, circuite de multiplicare, circuite sumator Carry-Look-ahead, sau microcontrolere complete. Unitățile logice și componentele de rutare în circuitele FPGA sunt controlate prin puncte de programare bazate pe tehnologiile Antifuse, FLASH, sau SRAM. Majoritatea circuitelor reconfigurabile folosite în practică sunt bazate pe tehnologia SRAM.

În aceste circuite punctele de programare, adică toate punctele de conectare și toate funcțiile logice sunt implementate cu ajutorul unui bit de memorie. Cu configurarea biților de memorie obținem un fișier de configurare numit în limba engleză *Bitstream*. Implementarea unui circuit în FPGA este un proces în care este creat un fișier care este încărcat/programat în dispozitiv. Proiectarea unui circuit începe cu descrierea comportamentului algoritmului cu ajutorul unui limbaj de descriere hardware, cele mai frecvente limbaje de descriere utilizate fiind VHDL și Verilog. Unitatea abstractă astfel obținută este optimizată ca să se potrivească în blocurile logice disponibile în circuitul FPGA [4]. Optimizarea și implementarea au următoarele etape:

- Sinteză Logică (*Logic Synthesis*) - traducerea de unități logice de nivel înalt și cod de descriere funcțională în porți logice;
- Mapare Tehnologică (*Technology Mapping*) - implementarea porților logice conform resurselor disponibile în circuitul FPGA;

- Plasare (*Placement*) - plasarea grupurilor logice obținute în blocurile logice în circuitul FPGA;
- Rutare (*Routing*) - conectarea matricelor de rutare pentru interconectarea componentelor din circuitul FPGA astfel încât să se realizeze/valideze circuitul descris de utilizator;
- Crearea fișierului bitstream de configurare (*Bitstream Generation*) - crearea unui fișier binar care conține informațiile de configurare pentru toate punctele/conexiunile de programare.

După compilare circuitul FPGA poate fi programat cu fișierul de configurare. După programare circuitul FPGA va realiza funcționalitatea circuitului dorit de către utilizator.

1.1 Arhitectura de bază a circuitelor FPGA

În cea mai generală formă circuitul FPGA are două resurse principale: resursele logice și matricea de rutare (*Interconnect*). Resursele logice sunt programabile pentru realizarea de operațiuni aritmetice și logice, datele sunt trimise între resursele logice prin matricea de rutare.

1.1.1 Elemente logice

În electronica digitală toate operațiile sunt reprezentabile cu funcții booleane și aceste funcții booleene pot fi reprezentate cu tabele de adevăr. Bazându-ne pe funcții booleene se pot construi structuri complexe care sunt capabile de executarea operațiilor aritmetice complexe ca însumare, multiplicare sau elemente decizionale, de exemplu structura *if-then-else*. Combinând acestea se pot implementa algoritmi complecși folosind doar tabelele de adevăr. Pentru implementarea tabelelor de adevăr cea mai folosită componentă este *Look-up Table* (LUT), una dintre componentele principale ale unităților logice ale circuitelor FPGA.

Un LUT este capabil de rezolvarea oricărei funcții cu N intrări doar prin programarea acestora cu tabelul de adevăr. Pentru rezolvarea funcțiilor mai complexe se pot combina mai multe componente LUT. Pe lângă componente LUT blocurile logice conțin circuite logice bistabile (flip-flop). Folosind doar componente LUT pentru construirea blocurilor logice ar fi imposibilă crearea circuitelor secvențiale care sunt capabile de a memora o anumită stare. Cel mai simplu bloc logic este alcătuit dintr-un LUT și un flip-flop D prezentat în figura 1.2.

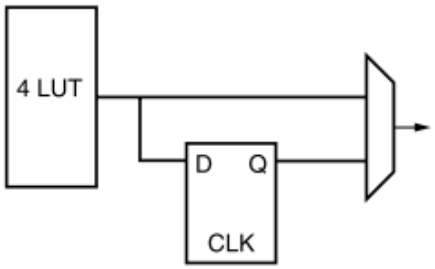

Figura 1.2 Schema simplificată a unui bloc logic

Punctele programabile ale blocului logic prezentat în figura 1.2 sunt: conținutul LUT, semnalul de selectare a multiplexorului la ieșire, și starea principală a flip-flop-ului [5].

1.1.2 Matrice de rutare (*Interconnect*)

În figura 1.3 se prezintă schema bloc simplificată a unei matrici de rutare, care cuprinde blocurile logice. Blocurile logice pot fi interconectate cu ajutorul acestor matrici de rutare.

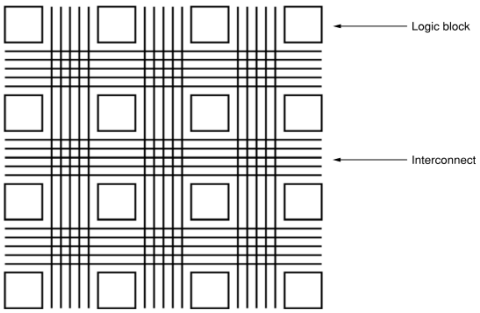

Figura 1.3 Schema simplificată a unei matrici de rutare

1.2 Elemente logice adiționale

1.2.1 Lanțuri rapide Carry look-ahead

Cea mai importantă operațiune aritmetică care poate fi implementată la circuitele FPGA este însumarea. Această operațiune necesită minimum două blocuri logice. De exemplu un LUT calculează suma și un alt LUT calculează bitul de transfer (*carry bit*). Prin conectarea în cascadă a acestor blocuri este posibilă formarea unui sumator de N-biți. Întârzierea apare în linia de carry unde semnalul trebuie să ajungă de la bitul cel mai puțin semnificativ la cel mai semnificativ. O soluție este scurtarea liniei carry între blocurile logice. Acest lucru poate fi realizat folosind liniile dedicate care nu necesită programare pentru conectarea semnalelor carry.

1.2.2 Circuite de multiplicare

Circuitul de multiplicare poate fi realizat folosind componente de LUT ca însumarea dar necesită multe blocuri logice care introduc întârzieri. O modalitate mai simplă este utilizarea circuitelor de multiplicare dedicate pe lângă blocurile logice care permit multiplicarea rapidă.

1.2.3 Memorii RAM

O metodă frecventă este utilizarea de blocuri de memorii dedicate în arhitectura circuitelor FPGA care permit implementarea de tabele de căutare care funcționează la viteză mare și cu spațiu de stocare mare. O soluție frecventă este utilizarea blocurilor de memorie de tip Dual-port. Memoriile de acest tip au două seturi de linii de adrese, date, și ceas

1.2.4 Unități de procesare

Circuitele FPGA actuale conțin unități de procesare complexe ca circuite dedicate DSP.

1.3. Moduri de configurare a circuitelor FPGA

O caracteristică majoră a circuitelor FPGA este că acestea ajung la utilizator ca un hardware configurabil, utilizatorul având posibilitatea să construiască arhitectura preferată, oferind flexibilitate mai mare decât soluțiile de dezvoltare software. În circuitele FPGA fiecare dintre elementele configurabile are nevoie de 1 bit de memorie pentru stocarea configurării primite de la utilizator. În cazul unui FPGA simplu bazat pe componentele LUT punctele de programare sunt constituite prin conținutul blocurilor logice și punctele de conectare ale matricei de rutare. În procesul de configurare se ajustează biții de memorie conectați la punctele de programare. Fișierul de programare în majoritatea cazurilor este un fișier binar simplu în care fiecare bit corespunde unui punct de programare. Structura fișierului de bit variază de la producător la producător. Pentru stocarea biților de configurare cele mai folosite tehnici sunt următoarele:

a. SRAM

Soluția cea mai frecventă pentru stocarea biților de configurare este memoria SRAM (*Volatile Static RAM*). Memoria SRAM este populară pentru că este rapidă și permite reconfigurări de nelimitate ori.

b. Memoria Flash

Nu este o soluție atât de populară ca memoriile SRAM dar este utilizată în multe circuite FPGA. Memoria FLASH nu este volatilă dar numărul reconfigurărilor este limitat. Oricum păstrează datele introduse după întreruperea alimentării de tensiune, după realimentare rămâne configurat, nu necesită reconfigurare.

c. Tehnologia Antifuse

A treia abordare de programare este utilizarea tehnologiei Antifuse [6]. După cum sugerează și numele tehnologia Antifuse este o conectivitate bazată pe metal care se comportă contrar tehnologiei Fuse. În mod normal comutatorul Antifuse este deschis. În timpul programării în condiții de laborator se topesc comutatoarele Antifuse, realizând un scurtcircuit între terminalele comutatorului Antifuse. Circuitele FPGA bazate pe tehnologia aceasta pot fi programate o singură dată.

1.4. Programarea circuitelor FPGA în limbaje HDL

Pe piața circuitelor FPGA doi dintre cei mai mari producători sunt Xilinx și Altera. Altera a fost fondată în anul 1983, primele circuite PLD s-a lansat în anul 1984; Xilinx a fost fondat în anul 1984, primele produse s-au lansat în anul 1985. Soluții de implementare prezentate la subsecțiunile următoare sunt bazate pe tehnologia Xilinx. Pe piața românească Xilinx este dominant parțial datorită companiei Digilent cu filială la Cluj-Napoca.

Prima generația a circuitelor FPGA Xilinx a fost programată cu mediul de dezvoltare Xilinx ISE (Integrated Synthesis Enviroment) ultima versiune a venit la 23 octombrie 2013 [7]. Ultima generație a FPGA Xilinx este programată în mediul de dezvoltare Xilinx Vivado care combină mai multe opțiuni pentru implementarea circuitelor FPGA în diferite limbaje hardware. În acest mediu de dezvoltare utilizatorul are posibilitatea de a crea circuite FPGA în limbajele VHDL și Verilog sau să folosească modulele mai avansate ca HLS cu care utilizatorul dezvoltă circuitele în limbajul C, care este interpretată și compilată de către mediul de dezvoltare pentru circuitul FPGA. O altă abordare este Matlab Simulink. Xilinx are un modul integrat numit *System Generator,* care după instalare este integrat în mediul Matlab Simulink.

1.4.1. Limbajul Verilog

Verilog este un limbaj de descriere hardware standardizat (IEEE 1364) [8], folosit pentru modelarea sistemelor electronice. Verilog este cel mai frecvent utilizat în proiectarea și verificarea circuitelor digitale la nivelul transferului de registre al abstractizării. Se utilizează, de asemenea, în verificarea circuitelor analogice și a circuitelor cu semnale mixte, precum și în proiectarea circuitelor genetice. Limbajul de descriere Verilog, este similar cu limbajele de programare software, deoarece includ modalități de descriere a timpului de propagare și a intensității semnalului (sensibilitate).

1.4.2. Limbajul VHDL

VHDL este un acronim pentru *Very High Speed Integrated Circuit Hardware Description Language* care este un limbaj de programare, care descrie un circuit logic prin funcție, comportament, flux de date [9]. Această descriere hardware este utilizată pentru a configura un dispozitiv programabil (PLD) cum ar fi un circuit FPGA cu un proiect logic personalizat. Formatul general al unui program VHDL este construit în jurul conceptului de blocuri (BLOCKS), care sunt unitățile de bază de construcție a unui design VHDL. În cadrul acestor blocuri de proiectare poate fi descrisă ușor funcția unor circuite logice. Un program VHDL începe cu un bloc numit ENTITY care descrie interfața de conectare. Interfața definește semnalele de intrare și ieșire ale circuitului proiectat. Blocul ARHITECTURE descrie funcționarea internă a circuitului dezvoltat. Figura 1.4 prezintă structura a unui program VHDL.

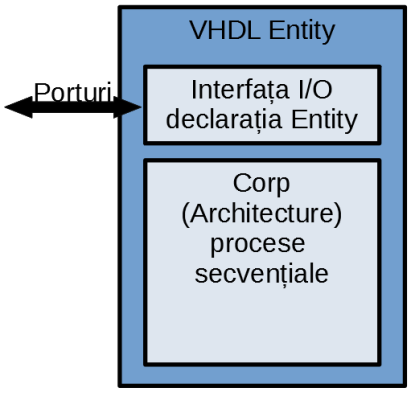

Figura 1.4 Structura unui program VHDL

1.5. Implementarea și evaluarea procesoarelor de control în limbajul VHDL

În acest sub-capitol vor fi prezentate trei arhitecturi de microprocesoare dezvoltate în VHDL și optimizate [10], pentru a fi ușor implementabile pe circuitele FPGA cu costuri reduse: primele două procesoare (EMMC și SLP) realizate de către autor, al treilea procesor (SCC) de către Morgan Fearghal (NUI Galway) partener din grupul de cercetare. Procesorul de referință este (PicoBlaze) - Xilinx. Acestea sunt procesoare minimaliste de 8 și 16 biți destinate funcțiilor utilitare și, de asemenea, pentru utilizarea educațională în design-uri pe bază de FPGA, System-on-Chip (SoC) [11]. Scopurile de proiectare au fost utilizarea optimă de celule logice și memorii de pe chip, performanța rezonabilă și frecvența maximă a semnalului de ceas. Acest lucru este important atunci când este utilizat drept coprocesor într-un SoC, pentru a evita reducerea frecvenței de ceas la întregul proces. Printre arhitecturile prezentate se găsesc modele care sunt construite pe o arhitectură de acumulatoare de tip pipe-line cu registre adresabile direct, stocate în memoria integrată în circuitul FPGA.

Problemele de întârziere ale semnalului de ceas cresc complexitatea rutei fizice în diferite părți ale procesorului. Prin urmare, una dintre cele mai importante provocări în astfel de modele este distribuția semnalului ceas, creșterea frecvenței de ceas, optimizarea consumului de energie și reutilizarea. Dezvoltarea pentru o amprentă digitală mai mică și creșterea continuă a vitezei [12], gestionarea ceasului și creșterea complexității disipării de putere afectează performanța designului. Crearea unui microprocesor într-un FPGA pe bază de soft-core de 8 biți sau 16 biți este posibilă, dar performanța ce poate fi realizată depinde de o serie de factori cum ar fi setul de instrucțiuni, modurile de adresare și tipul de arhitectură selectat (etape de pipeline, analize de dependență etc.). S-au dezvoltat trei microprocesoare, fiecare cu avantaje și dezavantaje, în comparație cu microprocesorul de referință, reprezentat în acest caz de arhitectura Xilinx PicoBlaze.

1.5.1. Microcontroller încorporat pe bază de FPGA pentru algoritmi de control

Microcontrolerul (EMMC - *Embedded Multicore MicroController*) prezentat este o mașină RISC cu arhitectură modificată de tip Harvard, cu două unități pipeline cu patru trepte pentru executarea instrucțiunilor. Acest procesor a fost prima încercare pentru dezvoltarea a unui controler dedicat pentru sarcini de control. Instrucțiunile dependente sunt verificate la nivel hardware. În mod ideal, după umplerea pipeline, procesorul poate executa opt instrucțiuni în paralel. Pentru a asigura o precizie ridicată, unitatea aritmetică a procesorului utilizează valori de 32 de biți în virgulă mobilă. Unitatea aritmetică are patru module pentru fiecare operație aritmetică (ADD, SUB, MUL, DIV). Toate instrucțiunile sunt executate în 6 cicluri de ceas, frecvența maximă a ceasului este de 25 MHz. În mod ideal opt instrucțiuni sunt executate în paralel din acest motiv unitățile aritmetice sunt formate din patru module, câte două în paralel pentru fiecare instrucțiune aritmetică, ca să se poată executa în același timp patru operații. În figura 1.5 se prezintă unitatea aritmetică.

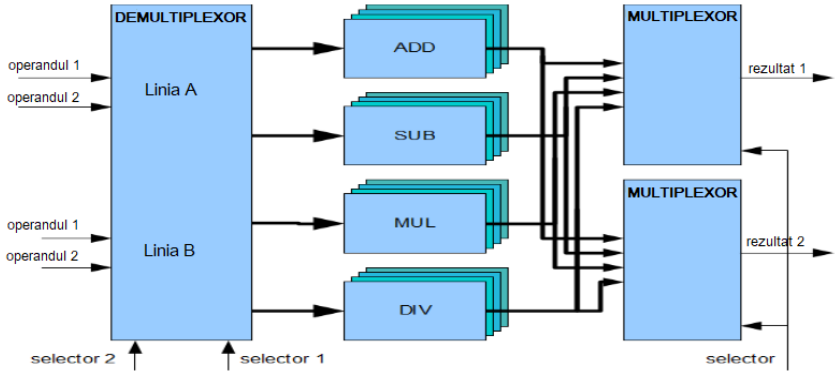

Figura 1.5 Structura unității aritmetice

Arhitectura procesorului este prezentată în figura 1.6. Deoarece arhitectura a fost construită pentru aplicații de control, microcontrolerul conține module periferice A/D și D/A, instrucțiunile sunt descărcate în memoria programului prin modulul UART. Este încorporat un modul encoder suplimentar, pentru a citi codificatoare incrementale. Pentru implementarea unor algoritmi de control aceste unități periferice sunt necesare pentru interfațarea cu mediul exterior. După compilare este generat un singur bloc (circuit) care poate fi folosit în orice FPGA Xilinx fără circuite adiționale. Deoarece circuitul conține și circuitele periferice și folosește numerele cu virgulă mobilă, va utiliza practic aproape toate resursele unui circuit Xilinx Spartan3 XC3S1200E [13].

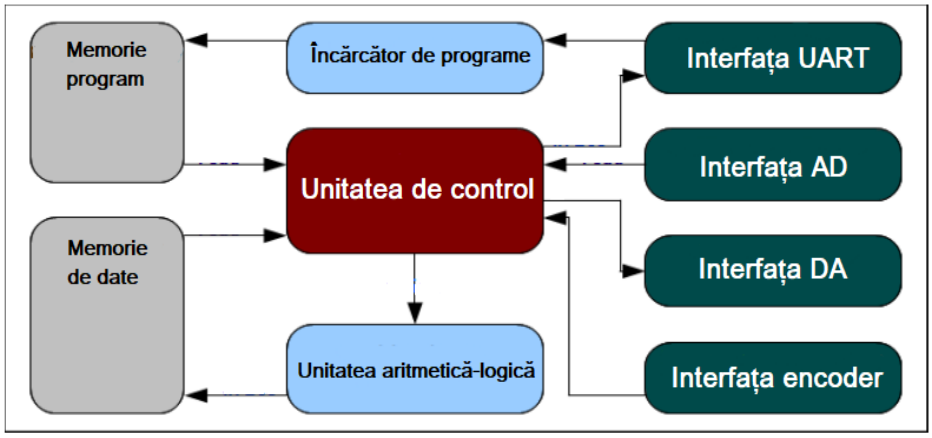

Figura 1.6 Arhitectura procesorului EMMC

În timpul executării instrucțiunilor, unitatea de control citește două instrucțiuni consecutive din memoria program și verifică dependențele de date. Dacă nu există dependențe, instrucțiunile sunt trimise la unitatea pipline pentru a fi executate. Dacă se găsesc dependențe de date, instrucțiunea este salvată într-o memorie FIFO și va fi executată mai târziu. În cazul dependențelor de date structura pipline nu este utilizată pe deplin. Microcontrolerul are un set simplu de instrucțiuni cu doar zece instrucțiuni: patru instrucțiuni aritmetice (ADD, SUB, MUL, DIV), patru instrucțiuni pentru mișcarea datelor (READ, OUTPUT, READEN, MOV) în memoria de date. În

tabelul 1.1 se prezintă gradul de utilizare a resurselor. Anexa 1 prezintă simularea procesorului în cursul execuției algoritmul PID.

Utilizarea logică	Utilizat	Disponibil	Grad de utilizare
Unități de Slice	8459	8672	97%
Unități Flip-Flop	2842	17344	16%
LUT	16305	17344	94%
BRAM	24	28	85%

Tabelul 1.1 Utilizarea resurselor

1.5.2. Sapientia Lab Processor (SLP)

Acest nucleu de procesor a fost dezvoltat în primul rând ca un instrument educațional pentru studenții din domeniul ingineriei. Cu toate acestea, poate fi util în multe aplicații datorită utilizării reduse a resurselor, ușurinței de programare și setului de module periferice atașabile. Scopul a fost de a proiecta un microcontroler RISC pe 16 biți în VHDL și de a se implementa pe un FPGA, luând ca referință procesorul Xilinx PicoBlaze, prezentat pe scurt în cele ce urmează. Acest microprocesor a fost proiectat folosind blocuri modulare care urmează arhitectura von Neumann. Procesorul este construit pe o arhitectură RISC cu un format de instrucțiuni pe două câmpuri. Dispune de o stivă stocată în memorie de 256 de niveluri și o unitate logică aritmetică (ALU) pentru operații cu instrucțiuni de date pe 16 biți și instrucțiuni de nivel logic. Modulul de memorie utilizează un BlockRAM al unui FPGA Spartan 3E Xilinx, care este împărțit în trei secțiuni. Memoria programului se poate potrivi cu instrucțiuni de 1Kx16-biți, în timp ce memoria de date poate stoca variabilele de 768x16-biți. Schema bloc a arhitecturii procesorului este prezentată în figura 1.7.

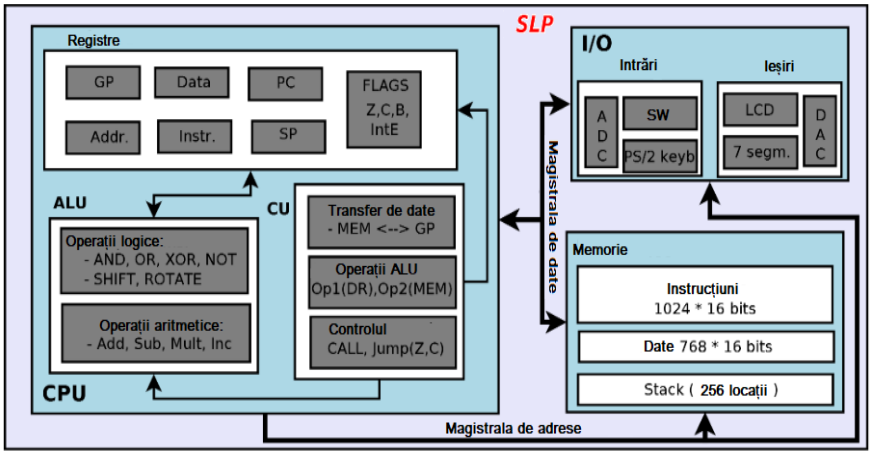

Figura 1.7 Arhitectura procesorului SLP

Modulele principale ale SLP sunt blocul de registre, *ALU* și unitatea de comandă (*CU*). Blocul de registre conține următoarele registre: de uz general (GP), adresă, date, instrucțiuni, numărător de program (*PC*), pointer de stivă (*SP*) și de stocare semnalele Flag (pentru Zero, Carry, Borrow

și biți de întrerupere). *ALU* poate procesa operații de bază cu operanzi pe 16 biți, cu excepția multiplicării, pentru care se utilizează numai octeții inferiori ale variabilelor. *CU* este un modul hibrid cu logică integrată pentru faza de preluare a instrucțiunilor, decodare și microprogramarea fazei de execuție. Acest lucru duce la o durată de execuție a instrucțiunilor de la 5 la 11 cicluri de ceas la o frecvență maximă de 35 MHz, dar este ușor de utilizat ca instrument educațional. Opțional, poate fi activat un sistem de întrerupere pentru controlul diferitelor module de intrare / ieșire care pot servi la o gamă largă de aplicații. Acest procesor este bazat pe o architectură generală de procesor cu unitățile de bază prezente la majoritatea procesoarelor actuale. Procesorul SLP are un set de instrucțiuni mai mare decât procesorul EMMC, care a fost dedicat pentru calcularea de algoritmi de control. Unitatea ALU folosește numere intregi de 16 biți, fără fracțiuni, fără unitatea pipline pentru executarea paralelă.

1.5.3. Single Cycle Computer (SCC)

Microprocesorul pe 16 biți este un calculator cu ciclu unic (SCC) și se bazează pe elementele arhitecturale prezentate în [14]. Arhitectura a fost adaptată și extinsă pentru a include elementele ilustrate în figura 1.8. ALU a SCC pot efectua operații tipice de calcul cu unul sau doi operanzi (ADD, SUB, ROTATE, SHIFT), precum și biții de FLAG utilizabili în declarații de branșament. Un modul separat al SCC este responsabil pentru gestionarea datelor și memoria de stivă. Componentele de returnare salvează rezultatele operațiilor ALU într-una din cele opt locații ale băncii de registre cu scop general și actualizează și registrele de funcții speciale. Procesorul are, de asemenea, un modul timp / contor și are un sistem simplu de întreruperi. SCC a fost prototipat pe hardware-ul de la distanță FPGA folosind viciLogic [15]. ViciLogic permite asamblarea, compilarea și transferul programului de asamblare în memoria de instrucțiuni SCC. IDE-ul bazat pe Python acceptă depanarea extensivă, incluzând rularea pas cu pas și ilustrarea animată a comportamentului fiecărui semnal din circuitul FPGA pentru a descoperi comportamentul detaliat al procesorului.

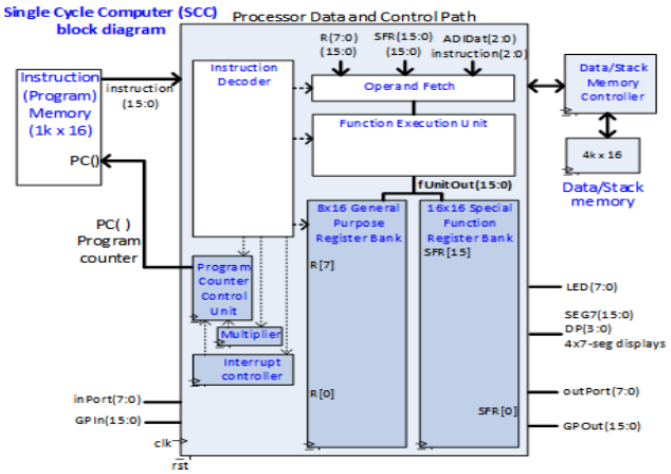

Figura 1.8 Arhitectura procesorului SCC [10]

Obiectivul viciLogic este de a privi hardware-ul procesorului în timp real, pentru a verifica cum se execută programele.

1.5.4. Arhitectura de referență Xilinx PicoBlaze

Procesorul Xilinx PicoBlaze este dezvoltat de către firma Xilinx. Procesorul PicoBlaze (figura 1.9) este un microcontroler RISC compact, capabil și rentabil, complet integrat pe 8 biți, optimizat pentru familiile Xilinx FPGA. Versiunea KCPSM3 descrisă în ghidul său de utilizare ocupă doar 96 de blocuri de tip SLICE într-un FPGA de generare Spartan®-3, care este de numai 12,5% dintr-un dispozitiv XC3S50 și un minuscul 0,3% dintr-un dispozitiv XC3S5000. În implementările tipice, un singur RAM bloc FPGA stochează până la 1024 instrucțiuni de program, care sunt încărcate automat în timpul configurării FPGA. Chiar și cu o astfel de eficiență a resurselor, microcontrolerul PicoBlaze realizează 44 până la 100 de milioane de instrucțiuni pe secundă (MIPS), în funcție de familia FPGA țintă și gradul de viteză [16]. Pentru a obține rezultate relevante, în timpul experimentelor s-a utilizat versiunea Spartan®-3 cu o frecvență maximă de 125.3 MHz.

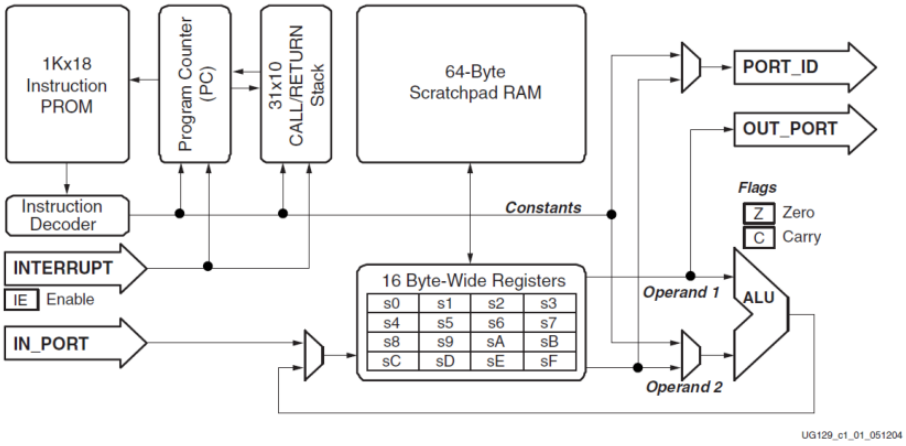

Figura 1.9 Arhitectura procesorului PicoBlaze [16]

1.5.5. Metode de evaluare a arhitecturii procesoarelor

În această secțiune, se analizează problemele de testare ale blocurilor de bază ale procesoarelor. Se pot proiecta module speciale, care implementează funcții fundamentale ce pot fi văzute ca un kernel de test pentru elaborarea de scheme complexe de testare.

Testarea *ALU* se efectuează prin definirea unor faze separate. Prima fază este calculul valorilor de testare (algoritmul de generare a numerelor pseudo-aleatoare), în timp ce a doua fază este cererea de testare urmată de evaluarea răspunsului la teste. Pentru a testa modulele *ALU* care lucrează cu valori întregi de 2*n* biți, avem nevoie de doi vectori de 2*n* biți, bitul de Input-CARRY și de codul de operare. Acești vectori de testare pot fi generați prin acumularea unei constante de 4*n* biți care este divizată în două câte două valori de *n* biți (S_1 și S_2). Fiecare pereche de vectori de testare de 2*n* biți, Z și W, este generată utilizând două adunări executate de *ALU*. Având în vedere

că ***i*** este indexul de buclă, Z_i este generat mai întâi prin adăugarea S_1 la Z_{i-1} cu Input-Carry = 0. În continuare, W_i este generat într-un mod similar. Codul de testare este obținut cu suma exclusivă a Z și W deplasată o dată la stânga. Pe lângă implementarea metodei de mai sus pe procesoarele descrise în secțiunile precedente, o metodă de testare personalizată a fost implementată prin dezvoltarea codurilor pentru controlerul de sistem proportional P și proporțional integrativ diferențial PID. O probă a algoritmului PID implementat este dată în figura 1.10. Pentru a asigura rezultate comparabile, măsurătorile semnalului de intrare și generarea semnalului de comandă au fost considerate a fi realizabile numai printr-o singură executare a instrucțiunilor I/O.

Prin urmare, timpul necesar pentru conversiile A/D și D/A nu a fost luat în considerare atunci când au avut loc măsurătorile de viteză de execuție, chiar dacă arhitectura testată a avut acele module. SLP, SCC și PicoBlaze toate au avut rezultate similare, așa cum este prezentat în tabelul 1.2 (f_{max} al plăcii Spartan3E folosit, Digilent Nexys3 este de 100 MHz, în timp ce valorile f_{max} din tabel sunt date după sinteza VHDL de către instrumentele de dezvoltare Xilinx), în timp ce EMMC le depășește puțin în anumite fragmente de cod. Acest lucru se întâmplă atunci când EMMC găsește un bloc de instrucțiuni fără dependențe de date, prin urmare cele două pipline pot fi păstrate pline și nu au loc evenimente de blocare.

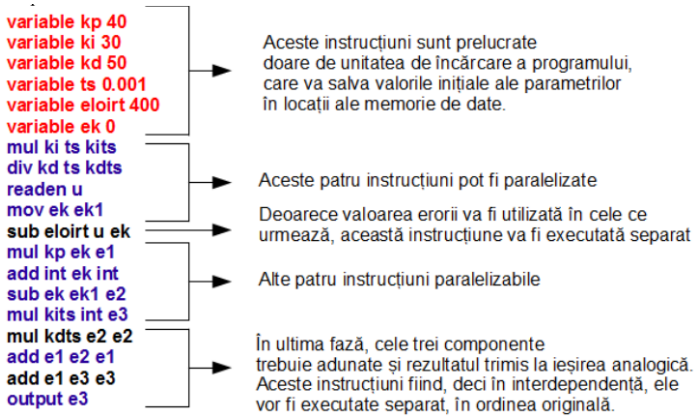

Figura 1.10 Algoritmul PID implementat cu instrucțiunile procesorului EMMC

Deoarece celelalte trei arhitecturi prezentate (SLP, SCC, PicoBlaze) nu au unități cu structură pipeline, timpul de execuție, în aceste cazuri, este o măsură a eficienței designului (numărul ciclurilor de ceas per instrucțiune CPI și diversitatea setului de instrucțiuni) și valoarea maximă realizabilă este viteza circuitului dat în interiorul XC3S250E Xilinx Spartan FPGA.

Microprocesor	Frecvența maximă de ceas f_{max} (MHz)	Cicluri de ceas/ instrucțiune	Utilizarea logică (FPGA Slice)	Memoria utilizată	ALU test timp de executie	P algoritm timp de executie
PicoBlaze	125	2	76	1	12	32
SLP	35	10	550	1	62	150
EMMC	25	6	320	1	40	96
SCC	45	1	410	1	16	36

Tabelul 1.2. Rezultate ale evaluării comparate

În tabelul 1.2 se prezintă rezultatele testelor comparative ale celor trei implementări în FPGA de procesoare, comparativ cu procesorul Xilinx Pico Blaze, considerat ca procesor de referință. În primul rând se constată că cel mai rapid procesor din punct de vedere al frecvenței maxime, a timpului de execuție și a resurselor consumate este procesorul dezvoltat de către Xilinx. Acest procesor este rapid și eficient, dar nu este ideal pentru implementarea algoritmilor matematici complecși, deoarece utilizează numere întregi cu 8 biți, fără fracțiune și este sensibil la erori cumulative. Procesorul EMMC are cel mai lent timp de execuție pentru algoritmul P, dacă algoritmul este optimizat pentru executarea paralelă, dar cu ajutorul de unități pipeline viteza de execuție poate fi redusă. Toate operațiunile sunt executate în virgulă mobilă și procesorul EMMC calculează rezultatele cel mai precis. Procesorul SLP este cel mai lent în execuția algoritmilor cu numere de 16 biți; acest procesor a fost dedicat nu numai pentru algoritmi de control, ci și ca un instrument educațional. Procesorul SCC este destul de rapid cu un set de instrucțiuni moderat, fiind bazat pe microprocesoarele cu un singur ciclu de ceas per instrucțiune.

Concluzii

Se analizează structurile de bază FPGA, resursele de care dispun și metodele de programare / configurare ale acestor categorii de circuite. Prezentarea are la bază familia de circuite FPGA tip Xilinx.

În partea finală a capitolului se prezintă dezvoltarea de către autor a două procesoare, dedicate pentru controlul sistemelor (EMMC și SLP), implementate în FPGA, care apoi se compară, prin testare cu două procesoare (unul de catalog, celălalt dezvoltat în FPGA la NUI Galway). Procesoarele s-au dezvoltate în limbajul VHDL, având nevoie de resurse minime. Procesoarele dezvoltate au fost capabile pentru rularea de algoritmi de control în timp real, procesorul EMMC fiind mai precis, operând în virgulă mobilă. Problema principală în calea performanței este setul de instrucțiuni redus, din care cauză procesoarele dezvoltate nu sunt capabile pentru implementarea de sarcini complexe, deoarece arhitectura devine destul de complicată. Aceste procesoare pot fi utilizate în calitate de coprocesor într-un proiect mai complex cu algoritmi de control dedicați.

Capitolul 2

Dezvoltarea circuitelor FPGA în Matlab Simulink – System Generator

Dezvoltarea circuitelor în limbajele VHDL sau Verilog, prezentată în capitolul 1, este un proces relativ lent și complex. Pentru realizarea mai ușoară de circuite complexe, Xilinx a dezvoltat mediul *System Generator* [17], un software care se integrează pe platforma Matlab Simulink.

2.1 Elementele principale ale mediului System Generator

Xilinx System Generator este un instrument de programare FPGA furnizat de firma Xilinx. System Generator se concentrează în mod specific asupra FPGA-urilor Xilinx, permițând utilizatorului să lucreze în mediul Simulink și să genereze nuclee parametrizate special optimizate pentru circuitele de FPGA Xilinx. Xilinx System Generator este livrat cu Xilinx ISE Design Suite (System Edition) și Xilinx Vivado HLS (System Edition). În mod implicit, blocul Xilinx Blockset conține peste 90 de blocuri DSP, de la blocuri simple ca sumatoare, multiplicatoare etc. la blocuri complexe, cum ar fi blocuri de corecție a erorilor, FFT, filtre și memorii.

Unele dintre aceste blocuri suportă de asemenea DSP în virgulă mobilă. Biblioteca cu virgulă mobilă furnizată de Xilinx afișează astfel de blocuri. Mai mult, System Generator include, de asemenea, blocurile Mcode și Black Box, care pot fi folosite pentru a integra codurile mcode și HDL, respectiv, direct în mediul de proiectare Simulink.

Cu ajutorul blocurilor Xilinx utilizatorul poate să creeze și să testeze algoritmi matematici mai complecși, implementați direct pe circuitele FPGA. Blocurile Xilinx nu pot fi conectați direct cu blocurile Simulink. Aceste blocuri au nevoie și de timp de eșantionare. Când se conectează domeniile Simulink pentru verificarea proiectului, fiecare intrare în domeniul de aplicare trebuie trecută prin blocul Gateway Out pentru a se coverti de la virgulă fixă înapoi la virgulă mobilă.

În ceea ce privește restul procesului de modelare a modelului, System Generator funcționează destul de similar cu Coder HDL [18], în care utilizatorii trag și elimină diferite blocuri în mediul Simulink pentru a proiecta sistemul global. Utilizatorii pot folosi apoi blocul Scop MATLAB și alte blocuri similare, precum și elementele bibliotecii Surse pentru a verifica rezultatele proiectului. Deoarece sistemul de generare este deja parte a Xilinx ISE sau Vivado HLS, nu sunt necesare alte instrumente de sinteză suplimentare, iar utilizatorii pot genera fișierul de biți direct din mediul Simulink. În acest scop, utilizatorul trebuie să adauge blocul Simulink numit System Generator. Blocul System Generator furnizează utilizatorilor funcționalități care permit utilizatorilor să selecteze fluxul de lucru și platforma țintă specifice.

Pentru a verifica codurile HDL prin simulări, System Generator poate fi folosit pentru a apela la simulatorul HDL implicit de la Xilinx, care vine ca parte a mediul de dezvoltare și aici, din nou, spre deosebire de HDL Coder și HDL Verifier; nici un instrument de simulare HDL de la alți furnizori nu este necesar. Cu toate acestea, System Generator este, de asemenea, compatibil cu unelte de simulare de la alți furnizori, cum ar fi ModelSim, oferind astfel un flux integrat.

2.2. Implementarea algoritmul de control PID în System Generator pentru un vehicul quadcopter

Un sistem complex din punctul de vedere al controlului este vehiculul aerian quadcopter. În cursul cercetărilor quadcopterul ne-a atras atenția pentru că circuitele montate la quadcopter pot avea resurse mai reduse, care sunt greu de îmbunătățit, astfel se potrivesc destul de bine pentru reconfigurarea parțială în circuite FPGA. În acest sistem unul din procesoarele dezvoltate și prezentate în capitolului precedent (EMMC) nu are instrucțiuni suficiente pentru controlarea tuturor aspectelor necesare pentru vehicul aerian quadcopter. Algoritmul PID [19] a fost implementat pe un circuit dedicat cu virgulă fixă în Matlab – System Generator cu resurse reduse. Scopul a fost stabilizarea unghiurilor de înclinare.

Un senzor IMU BOSHH BNO055 [20] care furnizează datele unghiulare condensate direct, fără calcule suplimentare, a fost conectat la circuitul FPGA. Cu ajutorul acestui senzor s-a dezvoltat controlerul PID, accentul fiind pus pe implementarea controlerului PID și a sistemului general [21]. Toate modulele necesare pentru stabilizarea quadcopterului sunt implementate la nivel hardware, făcând algoritmul mai rapid și independent de restul sistemului. Schema sistemului implementat este prezentată în figura 2.1. Microcontrolerul digital încorporat Xilinx PicoBlaze a fost utilizat pentru a interacționa cu senzorul IMU BOSCH. Acest procesor utilizează resurse reduse și este adecvat pentru citirea și transmiterea datelor. Controlerul inițializează senzorul după pornire și începe citirea datelor unghiulare. Senzorul este capabil să furnizeze date la 100 Hz: la fiecare 10 ms se poate citi o valoare nouă a poziției unghiulare, timpul de eșantionare a sistemului fiind de 10 ms. Circuitul FPGA funcționează cu un semnal de ceas de 50 MHz. După citirea unui nou set de date unghiulare, microcontrolerul PicoBlaze semnalizează la unitatea de comandă să înceapă calcularea unui nou set de semnale de control, care vor fi aplicate la motoarele quadrecopter.

Controlerul proporțional-integral-derivativ (PID) este unul dintre controlerele industriale utilizate pe scară largă. Controlerul PID calculează valoarea de eroare *e(t)*, care este diferența dintre o valoare presetată dorită și o variabilă de proces măsurată și produce corecția pe baza modulelor proporționale, integrale și derivative: coeficientul K_p pentru erorile actuale, K_i pentru erorile din trecut și K_d pentru eventuale erori în viitor.

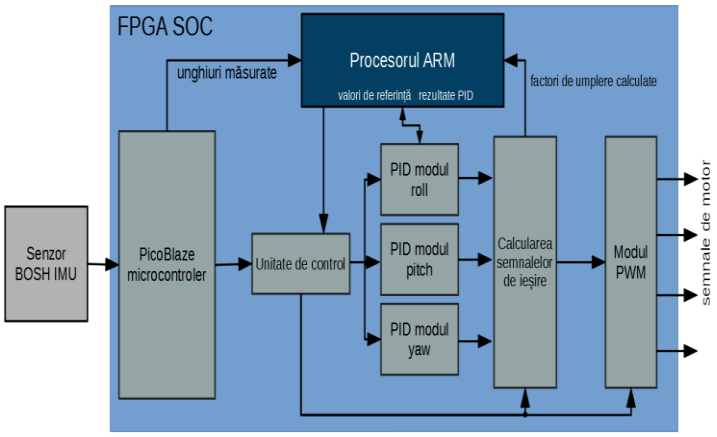

Figura 2.1 Schema bloc a controlerului dezvoltat

Funcția de control analogică se calculează cu următoarea ecuație.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \quad . \qquad (2.1)$$

Forma discretă utilizată în proiectarea prezentată este exprimată în ecuațiile (2.2)...(2.8) [22] [23], formula fiind calculată în situația în care controlerul are un filtru aplicat pe partea derivativă:

$$u[k] = \frac{-a1}{a0}u[k-1] - \frac{a2}{a0}u[k-2] + \frac{b0}{a0}e[k] + \frac{b1}{a0}e[k-1] + \frac{b2}{a0}e[k-2], \qquad (2.2)$$

unde coeficienții se calculează cu relațiile:

$$b0 = K_p(1 + NT_s) + K_i T_s(1 + NT_s) + K_d N \qquad (2.3)$$

$$b1 = -\left(K_p(2 + NT_s) + K_i T_s + 2K_d N\right) \qquad (2.4)$$

$$b2 = K_p + K_d N \qquad (2.5)$$

$$a0 = (1 + NT_s) \qquad (2.6)$$

$$a1 = -(2 + NT_s) \qquad (2.7)$$

$$a2 = 1, \qquad (2.8)$$

K_p fiind coeficientul proporțional, K_d - coeficientul derivativ, K_i - coeficientul integrativ, T_s timpul de eșantionare și N este parametrul filtrant aplicat pe partea derivată. Fiecare controler PID are parametrii proprii programați de sistemul de operare care rulează pe procesorul ARM. Unitatea implementată calculează doar ecuația $u[k]$, valorile coeficienților *b0 ... a2* , conform ecuațiilor de mai sus. Ele sunt sunt calculate de sistemul de operare la pornire și transmise în circuit. Schema de calcul a controlerului PID este prezentată în figura 2.2.

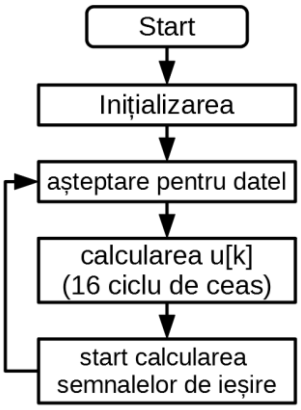

Figura 2.2 Schema de calcul PID

După ce este activat modulul PID, este nevoie de 16 cicluri de ceas pentru a calcula noua ieșire de control *u[k]*. Toate variabilele sunt valori de tip cu punct fix de 24 biți, cu 16 biți partea întreagă, și cu o parte facțională de 8 biți. După finalizarea calculului PID, cele patru semnale de comandă ale motorului sunt calculate folosind ecuațiile:

$$m1 = Throttle + Pitch_{PID} - Yaw_{PID} \quad (2.9)$$

$$m2 = Throttle - Pitch_{PID} - YAW_{PID} \quad (2.10)$$

$$m3 = Throttle + Roll_{PID} + YAW_{PID} \quad (2.11)$$

$$m4 = Throttle - Roll_{PID} + YAW_{PID}, \quad (2.12)$$

care necesită 2 cicluri de ceas, cele patru calcule fiind paralele. În ecuații notațiile au semnificația: throttle – accelerație, pitch – tangaj, yaw – girație și roll – ruliu.

Numerotarea motoarelor este prezentată în figura 2.3.

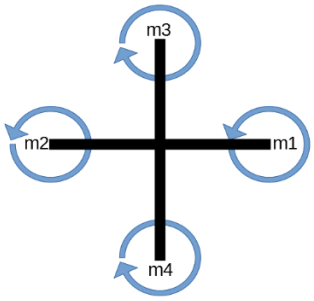

Figura 2.3 Numerotarea motoarelor quadrocopterului

După aceste calcule, modulul PWM generează semnale de control pe baza ciclurilor calculate. Întregul proces are 28 de cicluri de ceas pe sistemul de 50 MHz, ciclul având 560 ns.

Pentru testarea proiectului au fost utilizate două abordări: un mecanism cu două grade de libertate prezentat la figura 2.4 și un quadrocopter echipat cu o placă de dezvoltare Digilent Zynq Zybo [24]. Tabelul 2.1 prezintă utilizarea resurselor circuitului implementat.

Resursa	Utilizat	Disponibilitate	Grad de utilizare %
LUT	3738	17600	21.24
LUTRAM	98	6000	1.63
FF	3880	35200	11.02
BRAM	1	60	1.67
DSP	7	80	8.75
IO	19	100	19.00
BUFG	1	32	3.13

Tabelul 2.1 Utilizarea resurselor

În timpul testării, un software dedicat s-a rulat pe procesorul ARM tip SoC Zynq care a pornit circuitul implementat și a înregistrat valorile unghiulare provenite de la senzorul IMU, ieșirile PID calculate și ieșirile semnalelor PWM calculate de către unitatea implementată.

Când circuitul a fost testat prin descărcarea directă în circuitul FPGA pe quadcopter, o axă a quadcopter-ului a fost fixată, astfel încât quadcopter-ul să se poată rostogoli liber de-a lungul celeilalte axe. După mai multe teste a fost construit un mecanism dedicat; scopul construirii mecanismului [25] utilizat în această proiectare a fost de a se realiza o bibliotecă de testare adecvată pentru proiectul UAV de tip quadcopter autonom (acesta va fi prezentat în capitolele următoare). Această structură este un sistem cu două grade de libertate și este alcătuit dintr-un profil din plastic, cu un rulment liniar care rulează pe un arbore de oțel de doi metri lungime. La profilul plastic s-au atașat două brațe cu câte un motor fără perie cu elice pe fiecare braț (figura 2.4). Folosind acest mecanism testarea algoritmilor de control pentru un singur ax al quadcopter-ului a devenit mai ușoară și mai rapidă. Senzorul cu ultrasunete este utilizat pentru a determina altitudinea dispozitivului și magnetometrul pentru determinarea orientării orizontale. Mișcările simulate cu dispozitivul sunt cazurile care apar atunci când un quadcopter este în aterizare sau decolare.

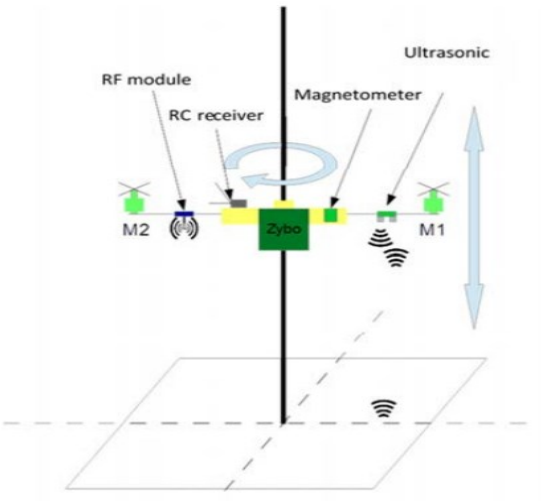

Figura 2.4 Structura fizică a sistemului construit [25]

Rezultatele măsurate sunt prezentate în figurile care urmează. Ieșirea circuitului este un semnal PWM care este aplicat la motoare. În figura 2.5 se prezintă variația poziției unghiulare de referință a unghiului de tangaj (pitch angle), cu stabilzare la zero. Din răspunsul sistemului se poate constata că quadcopterul a fost capabil să atingă valoarea de referință dorită, cu timp de răspuns anticipat.

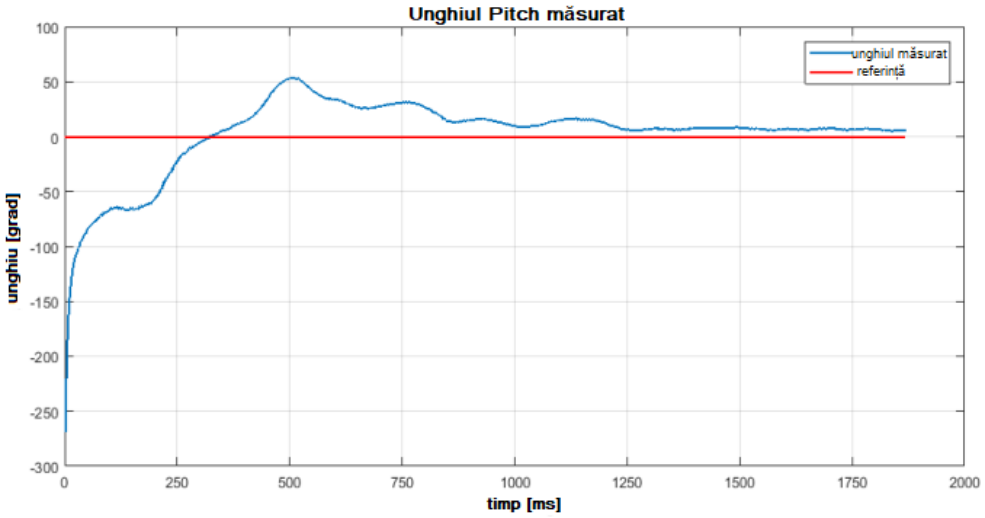

Figura 2.5 Unghiul Pitch măsurat

Motoarele sunt controlate în pereche, motoarele care sunt față în față au câte un semnal de control reflectat. Valorile coeficienților PID au fost stabilite prin rezultatele testului de încercare și de eroarea mimimă, la valorile $K_p = 4,5$, $K_i = 8,2$, $K_d = 1,2$. Semnalele de control generate care vor fi aplicate pe motoarele opuse sunt prezentate în figura 2.6.

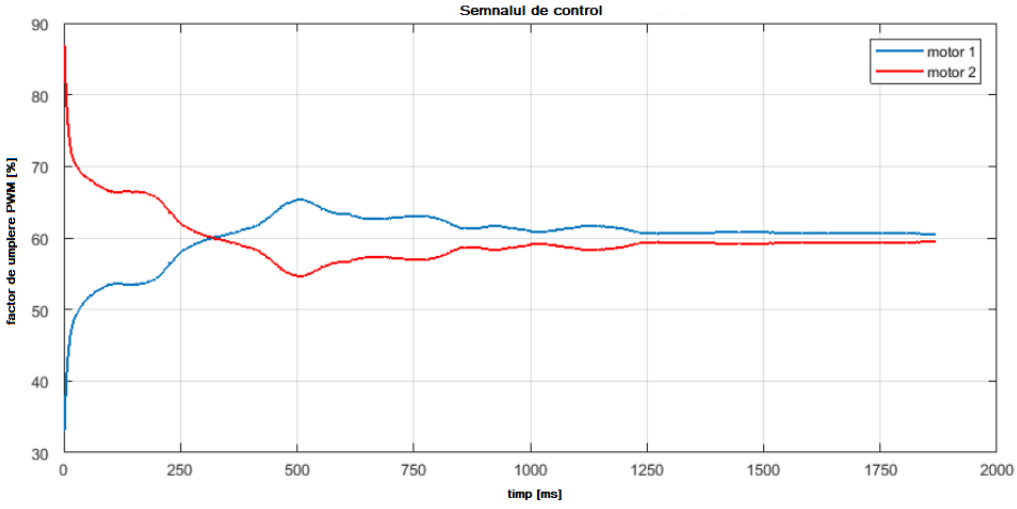

Figura 2.6 Semnalele de control generate de PID pentru motoare

Pentru a testa în continuare sistemul, a fost setat ca referință un semnal sinusoidal pentru unghiul de tangaj. Rezultatele sunt prezentate în figura 2.7. Linia roșie reprezintă semnalul de

referință sinusoidal și linia albastră este unghiul Pitch măsurat. La momentul măsurării quadcopterul a fost fixat de-a alungul unghiului Roll într-un mod în care numai unghiul Pitch poate să se schimbe.

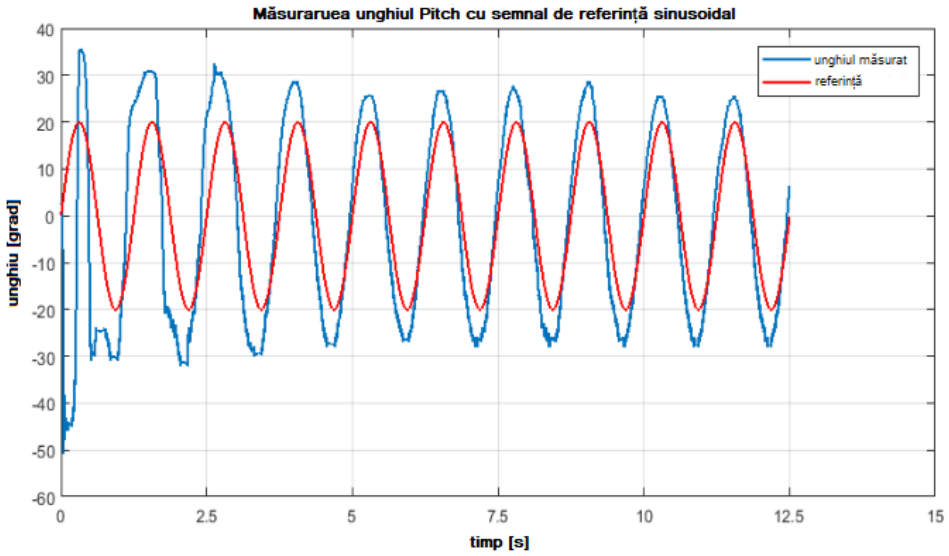

Figura 2.7 Măsurarea unghiulară de înclinare cu o referință sinusoidală

Concluzii

În acest capitol s-a prezentat un circuit de control PID dedicat, care a fost dezvoltat pentru stabilizarea unghiurilor de înclinare, implementat în Simulink System-Generator cu virgulă fixă, prin care se reduce substanțial complexitatea și utilizarea resurselor. Circuitul este rapid, are o amprentă de resurse minimală. System-Generator permite modelarea și implementarea algoritmilor matematici complecși în circuite FPGA. Procesarea are nevoie de 28 de cicluri de ceas pe sistemul de 50 MHz. Un avantaj mare este folosirea numerelor cu virgulă fixă pentru operațiuni matematice care sunt deja implementate în *Blocksetul* System-Generator. Proiectul folosește numere cu virgulă fixă pe 16 biți, cu partea întreagă și 8 biți partea fracționară, care este adecvată pentru calcularea precisă a rezultatelor. Pentru proiectarea și implementarea circuitelor de control în mediul de dezvoltare System-Generator s-a folosit și limbajul VHDL pentru implementarea circuitelor care controlează și sincronizează treptele de execuție.

Capitolul 3

Vivado HLS: implementarea circuitelor FPGA în limbajul C++

High-Level Synthesis (HLS) [26] permite descrierea diferiților algoritmi folosind limbajele C, C++ sau SystemC. După descrierea unui algoritm în unul din aceste limbaje, acesta poate fi compilat, poate fi executat pentru validarea faptului că algoritmul este corect din punct de vedere funcțional, iar apoi poate fi sintetizat într-un modul RTL. Modulul implementat poate fi analizat și depanat, iar apoi poate fi instanțiat în proiect sau poate fi împachetat într-un modul IP pentru a fi utilizat în proiect sau în proiecte viitoare. HLS pune la dispoziție mai multe facilități pentru generarea unei implementări optime a algoritmului [27].
Într-un caz tipic, programatorul exprimă algoritmul dorit folosind elementele lexicale și funcții descrise în limbajul C. O astfel de funcție în C ar putea avea multe interpretări ca: un modul hardware, care diferă pe o scară largă în timpul execuției, structura căilor de date sau interfețe periferice. Maparea codului C pe o cale de date: Vivado-HLS poate face în mod automat opțiuni implicite de implementare, care nu sunt specificate în codul C. Alternativ, programatorul are opțiunea de a direcționa multe din deciziile de mapare folosind directivele de tip #pragma oferit de Vivado-HLS. Programatorul are posibilitatea să scrie specificațiile C în C, C ++, SystemC, sau API C Open Language Computing (OpenCL) kernel, iar circuitul FPGA oferă o arhitectură masiv paralelă cu beneficii în performanță, cost și putere peste procesoarele tradiționale.

3.1. Fazele HLS

Sinteza la nivel înalt include următoarele faze:

Programare (Scheduling)

Stabilește operațiile care apar în timpul fiecărui ciclu de ceas pe baza:

- Lungimii ciclului de ceas sau frecvența ceasului;
- Timpului necesar pentru finalizarea operației, așa cum este definit de dispozitivul țintă;
- Instrucțiunii de optimizare specificate de utilizator.

În cazul în care perioada de ceas este mai lungă sau este folosit un FPGA mai rapid, mai multe operațiuni sunt finalizate într-un singur ciclu de ceas, și toate operațiunile s-ar putea încheia într-un singur ciclu de ceas. În contrast, dacă perioada de ceas este mai scurtă sau este folosit un FPGA mai lent, sinteza la nivel înalt programează automat operațiunile peste mai mult cicluri de ceas și unele operații ar putea fi implementate ca resurse de tip multicycle.

Legarea (Binding)

Aici se stabilește ce resurse hardware implementează fiecare operație programată. Pentru a implementa soluția optimă, sinteza la nivel înalt utilizează informații despre dispozitivul țintă.

Extracția logică de control (Control logic extraction)

Vivado HLS extrage logica de control pentru a creea o mașină cu stări finite (FSM) care secvențează operațiile din proiectul RTL. Figura 3.1 prezintă etapele de proiectare în Xilinx Vivado HLS:

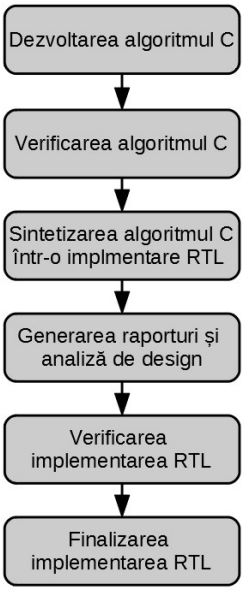

Figura 3.1 Etapele de dezvoltare

- Primul pas este dezvoltarea algoritmilor la nivelul C. Dezvoltarea se desfășoară la un nivel care este abstract de detaliile implementării, care consumă timpul de dezvoltare.
- Verificarea algoritmului la nivelul C. Validarea corectitudinii funcționale a designului se efectuează mai repede decât într-un limbaj de descriere hardware tradițional.
- Sinteza algoritmului C într-o implementare RTL. Instrumentul Vivado HLS va crea automat o implementare RTL a algoritmului planificat C. Vivado HLS va crea automat căi de date și module de cale de control necesare pentru implementarea algoritmului planificat în hardware.
- Generarea de rapoarte detaliate și analiza designului. După sinteză, Vivado HLS creează automat rapoarte de sinteză pentru a ajuta utilizatorul să înțeleagă performanța designului implementat.
- Verificarea implementării RTL, prin care utilizatorul are șansa ca să verifice dacă implementarea RTL este identică din punct de vedere funcțional cu codul C original.
- Implementarea pachetului RTL într-o selecție de formate IP-Core. Folosind Vivado HLS, proiectul RTL și pachete fișiere de ieșire RTL poate fi exportat ca IP-Core.

3.2. Detectarea deplasării în fluxurile video folosind o arhitectură distribuită pe platforma SoC pentru aplicații de control în timp real

Principalele rezultate ale acestui sub-capitol [28] constau în aplicabilitatea metodei pe platforme cu putere redusă și cu resurse reduse folosind tehnica de implementare HLS. Obiectivul secundar al cercetării descrise mai jos este pregătirea unui cadru pentru utilizarea metodei de detectare a mișcării cu unitatea de execuție la bord a roboților mobili sau a UAV-urilor. Acest lucru va oferi o precizie sporită în localizarea adaptivă și în capacitatea de evitare a obstacolelor. Scopul propus, este bine prezentat în literatura de specialitate [29] [30] [31] totuși majoritatea rezultatelor au fost obținute pe sisteme bine modelate, cu randament ridicat de energie și cu o mulțime de resurse de procesare. Este mult mai dificilă proiectarea și construirea unui astfel de sistem cu constrângeri de resurse reduse, cu consum redus de energie. Cele mai bune platforme pentru a atinge acest obiectiv sunt sistemele bazate pe FPGA sau sistemele pe chip (SoC). Această platformă oferă și alte oportunități, deoarece proprietățile lor de reconfigurare în timp real pot fi exploatate pentru a implementa algoritmi evolutivi pentru a da un adevărat comportament adaptiv dispozitivelor autonome. Pentru a atinge aceste obiective, cel mai important pas este acela de a găsi cea mai potrivită arhitectură de paralelizare a pașilor necesari pentru extragerea informațiilor de mișcare și deplasare, în timp real, din fluxul video achiziționat de o cameră de bord. Fluxul optic [29] este o condiție prealabilă a multor algoritmi de vizionare la nivel înalt [32], [30] și a aplicațiilor. Distribuția vitezei relative a mișcării aparente a modelului în cadrele video este referit ca flux optic (Optical Flow sau OF). OF este, de fapt, un câmp de vector cu valori aparente de viteză obținute prin măsurarea deplasării obiectului față de un punct de observare. Există mai multe metode pentru a calcula matematic OF. Primele metode s-au bazat pe gradient, principalele referințe fiind lucrările lui Horn și Schunck [29] sau Lucas și Kanade [33]. Una din metodele de cea mai înaltă precizie se bazează pe estimarea deplasării marginilor (edge displacements) în cadrele consecutive, dacă observăm intensitatea imaginii la momentul t cu $I(x, y, t)$. Pentru a simplifica calculele, se pot face două ipoteze: intensitatea $I(x, y, t)$ are dependența redusă de valorile coordonatelor x, y și timpul nu influențează intensitatea obiectelor care nu se mișcă față de cea a obiectelor în mișcare. Presupunem că există margini pe imagine care se vor deplasa cu (dx, dy) în timpul dt. Prin utilizarea celei de-a doua ipoteze, valorile intensității $I(x, y, t)$ pot fi dezvoltate în seria Taylor:

$$-\frac{\delta I}{\delta t} = \frac{\delta I}{\delta x}\frac{dx}{dt} + \frac{\delta I}{\delta y}\frac{dy}{dt} \qquad (3.1)$$

Această expresie este denumită ecuația constrângerii OF, unde $dx/dt=u$ și $dy/dt=v$ sunt componentele câmpului vectorial în direcțiile x și y. Valoarea u și v trebuie să fie determinate, pentru care se poate utiliza una dintre tehnicile diferențiale sau tehnicile de corelare.

Sunt preferate metodele bazate pe gradient cu tehnici de corelare, deoarece acestea prezintă o complexitate computațională mai scăzută, oferind în același timp un câmp vectorial cu o precizie

bună. OF este, de obicei, implementat într-un mediu PC, dar pentru a proiecta o arhitectură cu putere redusă, cu utilizarea redusă a resurselor, trebuie îndeplinite anumite compromisuri.

3.2.1. Etape de implementare a OF în platforme încorporate

Primul pas este, de obicei, segmentarea imaginilor de cadru. Găsirea obiectelor este esențială pentru calculul OF, deoarece deplasarea globală este puternic corelată cu mișcările obiectului din imagini. Localizarea zonelor omogene ale imaginii este un proces bine dezvoltat, cu toate acestea, poate fi solicitat pe platforme cu proprietăți restrictive. Prin urmare, ar trebui utilizate metode mai simple de prag (thresholding) cu obiectivele propuse, adică pragul clasic, semipragul sau clasificarea simplă. Este imperativă alegerea valorilor corecte de prag pentru aceste calcule. Una dintre căile posibile de a face acest lucru este utilizarea histogramei imaginelor. Obiectul și fundalul are o culoare medie globală cu o anumită distribuție dată de funcțiile Gaussian în histogramă. Intersecția parcelelor de funcții poate fi considerată drept valoarea optimă a pragului. De multe ori sunt două valori dominante, cea mai mică dintre acestea este valoarea de prag convențională.

3.2.2. Detectarea marginilor utilizând filtrul Canny

Prin utilizarea metodei de filtrare Canny dimensiunea datelor care urmează a fi procesate poate fi redusă drastic și se pot obține informații valoroase asupra imaginilor. Prin urmare merită a fi proiectate și implementate circuitele necesare care vor efectua această operație în timp real. Criteriile generale pentru ca o metodă de detectare a marginii să fie fezabilă sunt:

- Precizie mare, găsirea a cât mai multor margini în imagine;
- Punctele de margine detectate trebuie să fie la mijlocul benzii de margine;
- Evitarea detectării multiple a aceleiași margini.

Algoritmul Canny poate fi împărțit în patru etape:

- Eliminarea zgomotului din imagine utilizând un filtru Gaussian;
- Calcularea valorilor de gradient orizontal și vertical;
- Suprimarea margini non-maxime;
- Calculul histerezisului;

Implementarea unui filtru Gaussian pentru reducerea zgomotului. Imaginile zgomotoase sunt dificil de procesat atunci când se pune problema unei detecții precise a marginilor, care să satisfacă criteriile menționate mai sus. Cadrele video sunt netezite prin aplicarea unei convoluții cu filtrul Gaussian. Pentru un filtru cu nucleu $(2k + 1)$ x $(2k + 1)$ relația este:

$$H_{ij} = \frac{1}{2\pi\sigma^2}\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right), \qquad (3.2)$$

$$1 <= i, j <= (2k + 1).$$

Un filtru Gaussian de 5x5 cu distribuție σ = 1,4 are forma următoare:

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 4 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A, \qquad (3.3)$$

unde *B* este rezultatul filtrării, adică subimaginea cu filtrul Gauss aplicat, iar *A* este matricea de pixeli a subimaginii peste care se aplică filtrarea.

Este important de menționat faptul că dimensiunea filtrului Gaussian are un impact direct asupra calității de detectare a marginilor. Un filtru de 5x5 este un compromis bun între precizie și costul calculat.

Calculul gradientului orizontal și vertical. Acest pas este necesar pentru a putea determina mărimea gradientului global din imagine:

$$G = \sqrt{G_x^2 + G_y^2}. \quad (3.4)$$

În mod similar, direcția vectorului de gradient în fiecare punct poate fi calculat:

$$\theta = atan2\frac{G_y}{G_x} \qquad (3.5)$$

Suprimarea marginilor non-maxime. Eliminarea sau suprimarea marginilor de intensitate non-maxime produce margini mai subțiri cu un contrast mai mare. Se păstrează numai maximele locale, stabilind toate celelalte valori pentru gradient la zero. Următorii pași sunt calculați pentru fiecare pixel din imagini: compararea intensităților de margine cu pixelii vecini în direcții gradient pozitive și negative, valoarea este păstrată numai dacă este cea mai mare dintre cele trei posibile.

Postprocesare pentru filtrul Canny. În timp ce detectorul clasic de margine Canny este o metodă relativ simplă și precisă, nu poate funcționa bine în anumite cazuri speciale. Unele dintre principalele deficiențe sunt:

- Utilizând un filtru Gauss pentru a elimina zgomotul la imagine, deși se obține o imagine netezită, unele dintre marginile mai slabe, dar utile, ar putea fi pierdute.
- Pentru calculul intensității gradientului, detectorul original Canny utilizează o fereastră 2x2, producând o sensibilitate ridicată la zgomot și producând margini false.
- Algoritmul utilizează numai două valori de prag globale, deși imaginile complexe pot avea mai multe zone care necesită valori de prag diferite, care sunt dificil de calculat automat.

Pe lângă metoda Canny, valoarea gradientului și direcția pot fi calculate utilizând mai mulți operatori de detectare a marginii (Sobel, Prewitt, Roberts) [34] [35] [36]. Selectarea operatorului optim are un impact deosebit asupra calității rezultatelor. Pentru a rezolva problema pragului dublu, poate fi utilizată metoda Otsu pe imagine cu margini non-maxime suprimate, pentru a

genera limita superioară a valorii de prag [37]. Limita inferioară este de obicei jumătate din limita superioară. Acest proces adaptiv are următorii pași:

- $K = 1$, setarea de n iterații si coeficientul amplitudinii h;
- Calcularea valorile gradientului $G_x(x, y)$ și $G_y(x, y)$;
- Calcularea valorilor de pondere folosind:

$$d(x,y) = sqrt(G_x(x,y)^2 + G_y(x,y)^2) \quad (3.6)$$

$$w(x,y) = \exp(-\frac{sqrt(d(x,y))}{2h^2}). \quad (3.7)$$

- Filtrul adaptiv este obținut prin:

$$f(x,y) = \frac{1}{N}\sum_{i=-1}^{1}\sum_{j=-1}^{1} f(x+i, y+j)w(x+i, y+j) \quad (3.8)$$

$$N = \sum_{i=-1}^{1}\sum_{j=-1}^{1} w(x+i, y+j) \quad (3.9)$$

- Dacă $K = n$ se oprește ciclul, dacă nu $k = k + 1$.

Preprocesarea imaginilor în FPGA utilizând descrierea hardware-ului de sinteză la nivel înalt (HLS). Cadrele video sunt reduse și transformate în tonuri de gri, ca un prim pas de pre-procesare, pentru a minimiza costul de calcul al platformei încorporate.

Controlul contrastului cu egalizarea histogramei. Echilibrarea histogramei este una dintre metodele care poate corecta contrastul unei imagini. Utilizează valorile minime și maxime ale pixelilor din imagine:

$$Y = \left(\frac{X - X_{min}}{X_{max} - X_{min}}\right) * Y_{max} \quad (3.10)$$

Un exemplu pentru această etapă de preprocesare, obținut prin soluția implementată pe hardware, este prezentată în figura 3.2.

Figura 3.2 Rezultate corecției de contrast

Convoluția bidimensională. Atunci când s-a implementat această etapă a procesării, s-au utilizat instrumentele de paralelizare oferite de Xilinx Vivado Mediul HLS. Prin urmare, în loc de stocare de cadre întregi de imagine pe placa FPGA, consumând cantități mari din resursele limitate de memorie, fluxul video a fost procesat în mișcare, așa cum este primit de la modulul video printr-un modul de flux de date prezentat în figura 3.3. Pixelii care sosesc completează memoria tampon, formând o fereastră de eșantionare care va fi utilizată în calculul convoluției (figura 3.4) cu nucleul

prezentat. Figura 3.5 prezintă diagrama bloc a nucleului de preprocesare. Se poate observa că nucleul poate fi modificat pentru a testa performanța mai multor metode.

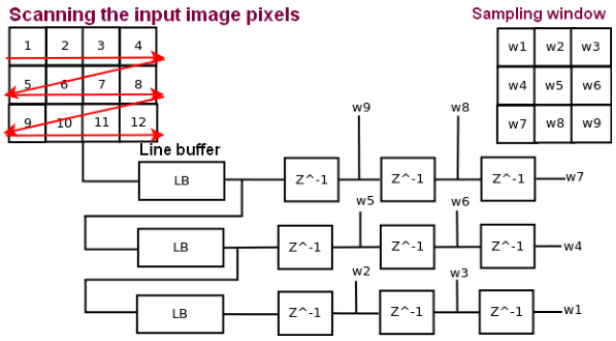

Figura 3.3 Diagrama bloc a memoriei tampon implementate în HLS [28]

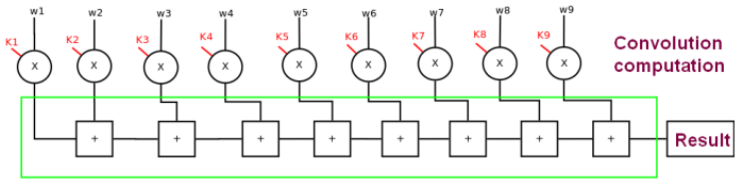

Figura 3.4 Diagrama de calculare a convoluției [28]

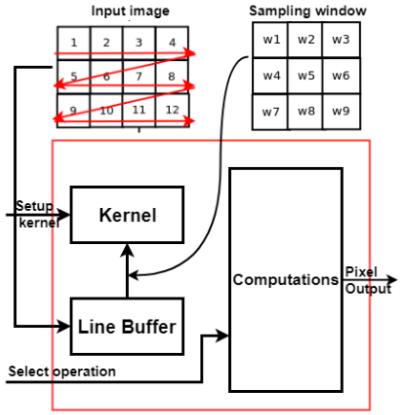

Figura 3.5 Diagrama bloc unităților de preprocesare [28]

În figura 3.6 se prezintă o diagramă bloc a întregului sistem, în care se poate constata că semnalele de ceas și date provin de la camera video, care este conectată la modulul *VideoIn*. Astfel semnalele sunt transmise printr-o linie AXI pe portul AXI-Slave de înaltă performanță de pe procesor. De acolo, procesorul transmite fluxul de date printr-o linie AXI-Master-General-Purpose către un modul de interconectare AXI.

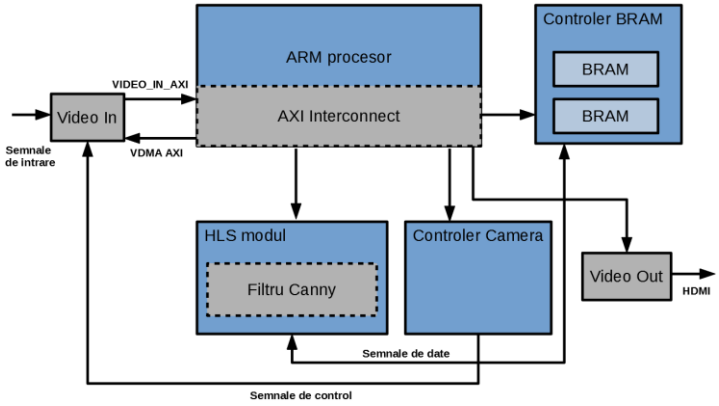

Figura 3.6 Diagrama generală a fluxului optic OF

Din modulul AXI Interconnect datele video se transmit într-o memorie BRAM, sau datele video pot fi trimise direct pe un afișaj prin modulul *VideoOut*. În plus modulele *Camera-Controll*, *Filtru Canny, VideoOut* pot fi controlate și monitorizate cu programul rulat în procesor. Prin linia CTRL-BUS a modulului *Canny*, modulul este controlat când este pornit, oprit sau trece într-o stare de așteptare. Rolul acestui modul de detectare a marginilor este de a citi din memoria BRAM datele de intensitate a imaginii colorate pentru prelucrare, de a obține marginile exacte și de a le stoca ca o imagine binară în cea de-a doua memorie BRAM. Imaginea binară salvată poate fi apoi utilizată de celelalte module de procesare a semnalului digital pentru calculul OF. Modulul *CameraControl* folosește linia Camera-Enable pentru a activa camera Raspberry Pi conectată, adresând și încărcând registrele corespunzătoare pe placa de dezvoltare și pe linia Culoare pentru a alege dacă imaginea primită este colorată sau în tonuri de gri.

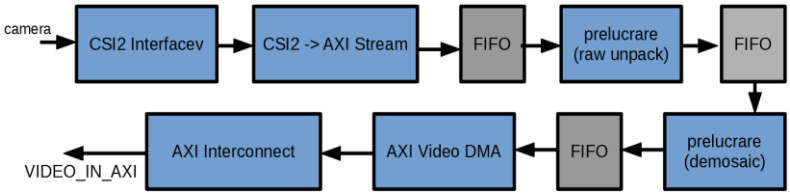

Figura 3.7 Schema bloc a modulului Input Video

În figura 3.7 se prezintă diagrama bloc a modulului care citește semnalul video recepționat. Pentru a primi date de la camera Raspberry Pi, trebuie utilizată o interfață CSI2. Aceste date de ieșire sunt transformate într-un flux-AXI și redirecționat către o unitate de stocare FIFO. Modulul FIFO are rolul de a împiedica sistemul să se prăbușească sau să funcționeze defectuos din cauza supraîncărcării cu date primite. *AXI-VDMA* este un modul care oferă acces la lățime de bandă mare între memorie și perifericele video *AXI4-Stream*. Acest modul oferă operații DMA eficiente bidimensionale cu operații independente de scriere și citire asincrone. În cele din urmă, datele care trec printr-un AXI -Interconnect *VIDEO-IN-AXI* vor fi adăugate la portul AXI cu viteză mare a procesorului. Diagrama bloc din figura 3.8 arată structura unității de ieșire video a sistemului. Semnalul video ajunge printr-o interfață *AXI-VDMA* în modulul de conversie *AXI-Stream-to-Video-out*, sincronizarea timpului este controlată de un controler de sincronizare video. Ulterior,

semnalul video generat este introdus într-un *RGB-to-DVI-coder* care transformă intrarea astfel încât imaginea capturată de camera video să poată fi văzută prin conectarea unui ecran la portul de ieșire HDMI. Hardware-ul folosit a fost o placă de dezvoltare Digilent Zybo FPGA.

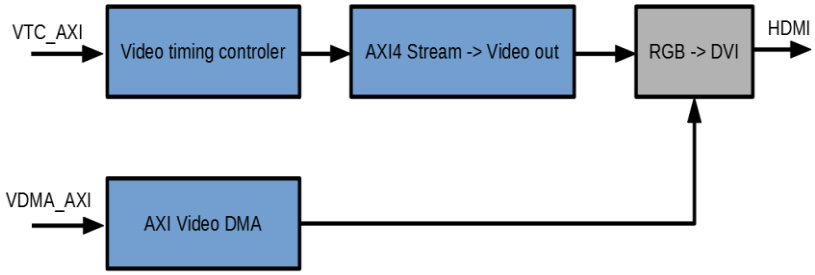

Figura 3.8 Unitate de ieșire Video

În lucrările anterioare ale autorului tezei [38] a fost dezvoltat un algoritm de procesare video capabil să detecteze mișcarea în timp real într-un sistem încorporat prin efectuarea unor operații matematice relativ simple. Cu algoritmul prezentat utilizatorul poate să determine exact poziția unei linii pe un fond omogen în ceea ce privește camera, cu aplicație la un robot mobil. Pentru a detecta marginile deplasate, este necesară procesarea cadrelor secvențiale. Cadrele sunt acoperite cu kernele (nucleu) de 5x5 pixeli, fiecare dintre acestea preluând o direcție de mișcare atunci când este detectată activitate. Un nucleu este împărțit în 4 cadrane (A, B, C, D) cu 3x3 pixeli. Evaluarea acestor cadrane determină direcția deplasării într-un kernel în funcție de cadranul activat. Cea mai mare creștere a punctelor vii determină cadrele active. Această metodă avea o rezoluție de 45 de grade. Rezultatele experimentale au demonstrat aplicabilitatea metodei de definire a orientării în timp real a vehiculelor aeriene fără pilot. Metoda a fost extinsă prin creșterea rezoluției la 15 grade. În locul primei alocări cu patru zone, blocurile au fost împărțite în 30x30 pixeli în 24 de zone. Acest calcul va oferi direcția de mișcare calculată din perechea de cadre curente. Rezoluția imaginii care conține numai datele de localizare a marginilor a fost aleasă astfel încât să poată fi stocată în elementele BRAM de pe placa de dezvoltare și în același timp aliniată cu dimensiunea cadranului prezentată mai sus. Prin urmare au fost obținute imagini de intrare cu 11 OF zone de calcul pe linie, repetate de opt ori. Imaginea care trebuie procesată este de 240x330 = 79200 pixeli. Memoria video are o lățime celulară de 15 biți și reprezintă 15 pixeli într-o locație adresabilă. Acesta trebuie să fie descompus în zone de 15x15 pixeli, generând zone 16x22. 79200 de pixeli stocați într-o memorie de 5280x15 biți. Pentru a adresa memoria 5280, avem nevoie de un buffer de adresă de 13 biți. Pentru a alimenta datele în unitatea de procesare OF, a fost construit un modul *Framedemux*, arătat în Figura 3.9.

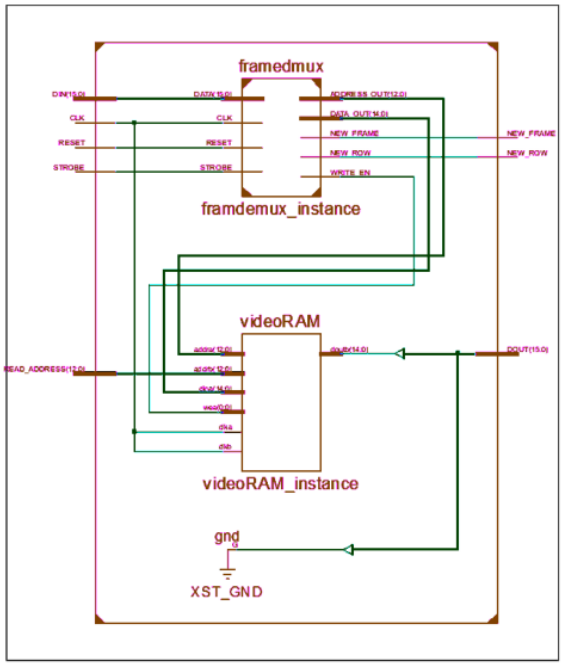

Figura 3.9 Unitatea Framedemux

Semnalele wea_s, $addr_s$, $data_s$ sunt semnale conectate la memoria utilizată ca intrare, adresare, introducere de date noi și permițând scrierea în memorie. După cum se arată în Figura 3.10, transferul de date este coordonat de mașina de stare între sistemul de operare Linux și modulul procesorului de imagine.

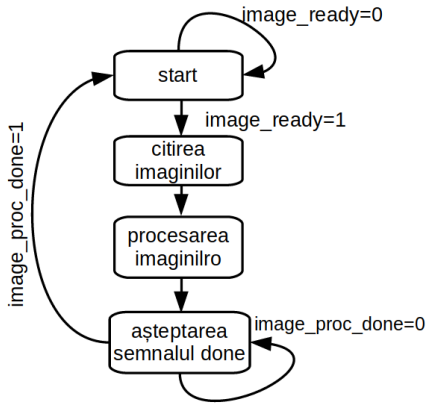

Figura 3.2 Maţina de stare conectată la unitatea Frameloader

Sistemul de operare înregistrează imaginea care are nevoie de procesare într-un bloc de memorie BRAM. Apoi, semnalează mașinii de stare cu semnalul pregătit pentru imagine, că în memorie este descărcată o imagine nouă. Mașina de stare semnalează modulului de procesare a imaginii să citească imaginea din *portB* a memoriei BRAM. După procesare, scrie imaginea rezultată înapoi. În urma procesului de înregistrare, modulul de procesare a imaginii semnalează

prin semnalul de procesare a imaginii că calculul OF este completat. De asemenea, sistemul de operare vede acest semnal și poate citi imaginea prelucrată.

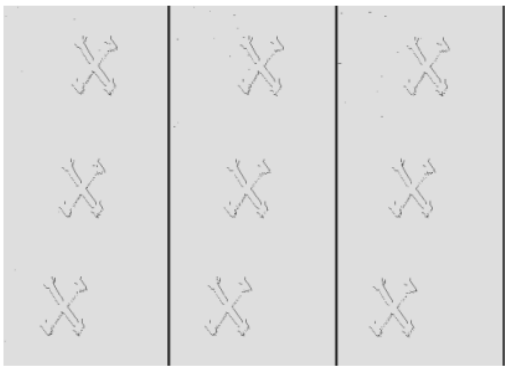

Figura 3.3 Marginile găsite cu sistemul implementat

În figura 3.11 se prezintă o serie de imagini ale unui model în formă de X în mișcare orizontală, detectat de sistemul implementat de hardware. Marginile sunt extrase pe chip, ca și determinarea direcției de mișcare folosind calculul OF

Concluzii

În acest capitol s-a prezentat o metodă pentru prelucrarea imaginilor și detectarea deplasării în imagine. Cu introducerea mediului de dezvoltare HLS algoritmii prezentați pentru filtrul Canny s-au implementat în C++. Algoritmul de detectare este implementat în VHDL, iar pentru o precizie mai mare această unitate trebuie să fie portată în HLS. Problema principală în această abordare este utilizarea resurselor. În partea aceasta a cercetării s-a observat că HLS consumă o mare parte din resursele disponibile. În circuitul Xilinx Zync XC7Z010 filtrul Canny consumă 38% din celulele logice disponibile. Proiectele realizate în HLS au nevoie de circuite FPGA cu capacitate mai mare. HLS simplifică implementarea algoritmilor matematici, dar necesită costuri mai mari de resurse.

Capitolul 4

Reconfigurarea parțială a circuitelor FPGA

Beneficiul evident al utilizării unor dispozitive reconfigurabile, cum ar fi FPGA [39], este că funcționalitatea de care dispune un dispozitiv se poate modifica și actualiza ulterior. Dacă apar funcționalități suplimentare sau sunt disponibile îmbunătățiri ale proiectului, circuitul FPGA poate fi complet reprogramat cu o nouă logică. Pentru mulți utilizatori acest lucru nu este suficient. Ce se întâmplă dacă utilizatorul vrea să schimbe logica într-o parte a unui FPGA, fără a perturba întregul sistem? De exemplu, avem un proiect compus din mai multe blocuri logice și, fără a perturba sistemul și a opri fluxul de date, funcționalitatea într-un singur bloc trebuie să fie actualizată. Astfel, avem nevoie de o modalitate de reconfigurare parțială a aplicației pe un dispozitiv.

Prin reconfigurare parțială se parcurge un proces de proiectare, care permite reconfigurarea unei porțiuni limitate, predefinite a unui FPGA, în timp ce restul dispozitivului continuă să funcționeze. Acest lucru este deosebit de valoros în cazul în care dispozitivele funcționează într-un mediu critic care nu poate fi întrerupt în timp ce unele subsisteme sunt redefinite. Folosind reconfigurarea parțială, este posibilă creșterea funcționalității unui singur FPGA, permițând utilizarea unor dispozitive mai mici, decât ar fi fost nevoie altfel. Aplicații importante pentru această tehnologie includ sistemele de comunicații reconfigurabile și sistemele criptografice.

Într-un FPGA bazat pe SRAM, toate caracteristicile programabile de către utilizator sunt controlate de celule de memorie care sunt volatile și trebuie configurate la pornire. Aceste celule de memorie sunt cunoscute ca memorie de configurare și definesc conținutul tabelului de căutare (LUT), rutarea semnalelor, standardele de tensiune de intrare / ieșire (IOB) și toate celelalte aspecte ale proiectului. Pentru a programa memoria de configurare, instrucțiunile pentru logica de control, care face configurarea și datele pentru memoria de configurare, sunt furnizate sub forma unui fișier numit *bitstream*, care este livrat la dispozitiv prin interfața de configurare JTAG, SelectMAP, serial sau ICAP(Xilinx).

Un circuit FPGA poate fi parțial reconfigurat folosind un fișier *bitstream* parțial. Utilizatorul poate să folosească un astfel de fișier *bitstream* parțial pentru a schimba structura unei părți a unui proiect FPGA, pe măsură ce restul dispozitivului continuă să funcționeze.

Avantajele reconfigurării parțiale

Reconfigurarea parțială este utilă pentru sistemele cu funcții multiple care pot partaja resursele disponibile ale aceluiași dispozitiv FPGA [40]. În astfel de sisteme, o secțiune a circuitului FPGA continuă să funcționeze, în timp ce alte secțiuni ale circuitului FPGA sunt dezactivate și reconfigurate pentru a oferi noi funcționalități. Acest lucru este similar cu situația în care un

microprocesor gestionează schimbarea de context între procesele software. În cazul reconfigurării parțiale a unui FPGA, este schimbat totuși hardware-ul, nu software-ul.

Reconfigurarea parțială oferă un avantaj față de folosirea unui fișier *bitstream* întreg în aplicații care necesită o funcționare continuă, ceea ce nu ar fi posibil în timpul reconfigurării complete. Ca exemplu poate fi dat un afișaj grafic care utilizează sincronizarea orizontală și verticală. Datorită mediului în care funcționează această aplicație semnalele și legăturile de radio și video trebuie să fie păstrate, dar formatul și modul de prelucrare a datelor necesită actualizări și modificări în timpul funcționării. Cu reconfigurarea parțială, sistemul poate menține aceste legături în timp real, în timp ce alte module din cadrul FPGA sunt modificate în timpul funcționării.

Metodologie pentru reconfigurarea parțială

Pentru a implementa cu succes un proiect de reconfigurare parțială, trebuie să fie urmată o metodologie de proiectare strictă. Câțiva pași de bază care trebuie să fie urmați:

- Introducerea de module macro de magistrală între modulele care trebuie înlocuite (numite module de reconfigurare parțială PRM – *Partial Reconfiguration Module*) și restul proiectului (logica statică). Acestea sunt canalele sau porturile prin care modulele comunică și transmit date.
- Folosirea pașilor de sinteză pentru a genera un fișier *netlist* parțial reconfigurabil.
- Planificarea zonei PRM și gruparea tuturor modulelor statice.
- Plasarea de macrouri de magistrală.
- Folosirea de pași specifici reconfigurării parțiale.
- Rularea fluxului de implementare a reconfigurării parțiale.

Pe piața circuitelor FPGA cei mai semnificativi participanți sunt Xilinx și Altera (cumpărat de Intel). Teoria reconfigurării parțiale este aproape similară la ambele companii, doar implementarea diferă. În continuare vor fi prezentate metodele folosite pentru reconfigurarea parțială de către Intel-Altera și Xilinx. Tehnologia folosită de către Xilinx va fi examinată în detaliu, întrucât realizarea practică s-a bazat la tehnologia furnizată de către Xilinx.

4.1.Reconfigurare parțială Intel-Altera

Reconfigurarea parțială (PR) permite reconfigurarea dinamică a unei părți din circuitul FPGA, în timp ce proiectul care a rămas în circuitul FPGA continuă să funcționeze. Mediul de dezvoltare Intel Quartus Prime Pro [41] suportă caracteristica PR pentru familia de dispozitive Intel Arria 10 și Intel Stratix 10 [42]. Figura 4.1 ilustrează modelul de reconfigurare Intel-Altera

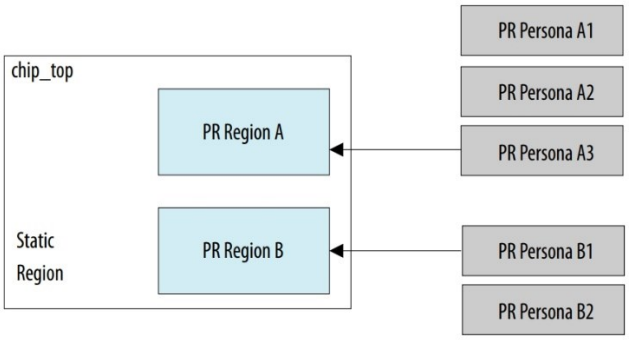

Figura 4.1 Reconfigurare Parțială Intel-Altera [42]

PR oferă următoarele beneficii față de o proiectare standard:

- Permite reconfigurarea design-ului în timpul execuției;
- Crește scalabilitatea proiectului prin multiplexarea de timp;
- Reduce costurile și consumul de energie prin utilizarea eficientă a spațiului de dispozitiv;
- Suportă funcția de timp-multiplexare dinamică în proiectare;
- Îmbunătățește timpul inițial de programare prin fișiere de *bistream* mai mici;
- Permite actualizarea ușoară a sistemului prin schimbarea hardware la distanță.

4.2.Reconfigurarea parțială Xilinx

Reconfigurarea parțială în circuitele Xilinx e similară cu circuitele Intel-Altera. La Xilinx fișierul *bitstream* de configurare se numește fișier BIT. După configurarea totală a circuitului părți din circuitul FPGA pot fi reconfigurate în timpul operației [43]. Figura 4.2 ilustrează reconfigurarea parțială definită de către Xilinx, această definiție fiind similară cu prezentarea din figura 4.1 al Intel-Altera.

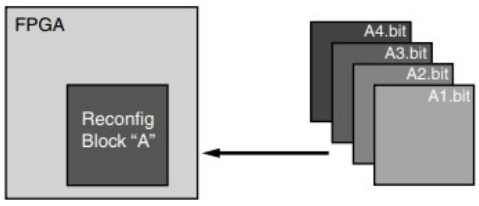

Figura 4.2 Reconfigurare parțială Xilinx [43]

Așa cum se arată în figura 4.2, funcția implementată în *Reconfig Block A* este modificată prin descărcarea unuia dintre fișierele parțiale BIT, A1.bit, A2.bit, A3.bit sau A4.bit. Logica în proiectul FPGA este împărțită în două categorii: logica reconfigurabilă și logica statică. Zona gri deschisă a Blocului FPGA reprezintă logica statică, iar blocul cu eticheta *Reconfig Block "A"* reprezintă o zonă de logică reconfigurabilă. Logica statică rămâne funcțională și nu este afectată de încărcarea unui fișier parțial BIT. Logica reconfigurabilă este înlocuită de conținutul fișierului parțial BIT descărcat.

4.2.1.Terminologia Xilinx

Sinteza de tip *Bottom-Up*

Sinteza *Bottom-Up* este sinteza proiectelor modulare, fie într-un proiect sau mai multe proiecte. Un proiect poate să devină un modul la un alt proiect. În Vivado, sinteza *Bottom-Up* este menționată ca sinteză *out-of-context* (OOC). Sinteza OOC generează un fișier *netlist* (DCP fișierul *netlist* nu conține conexiunile de intrare și ieșirile fizice, doar planul circuitului) separat pentru fiecare modul OOC, care este necesar pentru reconfigurarea parțială, pentru a ne asigura că nu există optimizare în afară limitei modulului, adică modulele sunt optimizate și sintetizate separat. În sinteza OOC, logica de nivel superior (sau statică) este sintetizată cu *black_boxmodule* pentru fiecare modul OOC.

Configurație

O configurație este un proiect complet, care are câte un modul reconfigurabil pentru fiecare partiție reconfigurabilă. S-ar putea să existe mai multe configurații într-un proiect FPGA cu reconfigurare parțială. Fiecare configurație generează un fișier BIT complet, precum și un fișier parțial BIT pentru fiecare modul reconfigurabil (RM – *Reconfiguration Module*).

Cadru de configurare

Cadrele de configurare sunt cele mai mici segmente adresabile ale memoriei de configurare FPGA, fiind construite din elemente de la cele mai joase niveluri. În dispozitivele Xilinx, corpurile reconfigurabile sunt următoarele: există un element de bază larg (CLB, bloc RAM, DSP, acestea sunt prezente fizic în circuitul FPGA) și o regiune de ceas (unități dedicate pentru generarea, modularea și transmiterea semnalului de ceas) mare.

Portul de acces pentru configurare internă

Portul de acces pentru configurare internă (ICAP - *Internal Configuration Access Port*) este, în esență, o versiune internă a interfeței *SelectMAP*. Primitivul ICAP Xilinx oferă acces intern la logica de configurare a FPGA din interiorul circuitului FPGA. Prin interfața ICAP, datele de configurare pot fi încărcate dinamic în memoria de configurare a circuitul FPGA, în timpul funcționării. Interfața ICAP conține porturi de date separate pentru citirea (O) și scrierea (I) datelor de configurare.

Reconfigurarea parțială (PR)

Reconfigurarea parțială modifică un subset de logică într-un proiect FPGA operațional descărcând unui fișier *bitstream* parțial.

Partiție

O partiție este o secțiune logică a proiectului, definită de utilizator, la o limită ierarhică, luată în considerare la refolosirea proiectului. O partiție este implementată ca nouă sau este păstrată de

la implementarea anterioară. O partiție care este păstrată, păstrează nu numai funcțiile implicite dar și implementările implicite.

Definirea partițiilor (PD)

Acesta este un termen utilizat numai în fluxul proiectării. Aici se definesc un set de module reconfigurabile care sunt asociate cu instanța modulului (sau partiția reconfigurabilă). Un PD este aplicat tuturor instanțelor modulului și nu poate fi asociat cu un subset de module instanțiate.

Pini de partiție

Pinii de partiție reprezintă conexiunea logică și fizică între logica statică și o logică reconfigurabilă. Instrumentele în Vivado creează automat, plasează și gestionează pinii de partiție.

Portul de acces pentru configurarea procesorului (PCAP)

Portul de acces pentru configurarea procesorului (PCAP - *Processor Configuration Access Port*) este similar cu configurația internă (ICAP) și este portul principal utilizat pentru configurarea unui dispozitiv SoC (System on Chip - SoC este un circuit integrat care conține toate unitățile principale ale unui calculator într-un singur chip) Zynq-7000. Lângă partea reconfigurabilă este integrat un procesor ARM cu o arhitectură fixă care conține unități periferice, un controler pentru magistrala AXI și un controler pentru reconfigurare PCAP care este integrat direct în procesor și este disponibil încă de la sistemul de operare care rulează pe procesorul ARM.

Modulul DevC (*Device Configuration*) conține modulul PCAP. Portul de acces pentru configurarea procesorului este o interfață relativ nouă, care oferă procesoarelor de tip SoC capacitatea de a accesa memoria de configurare cu un canal DMA de lățime de bandă mare. Această interfață este disponibilă numai pe FPGA / SoC dispozitive care conțin procesoare precum Zynq-7000, toate programabile SoC.

Unitate programabilă (PU)

În arhitectura UltraScale unitățile programabile reprezintă resursele minime necesare pentru reconfigurare. Dimensiunea unui PU variază în funcție de tipul de resurse. Deoarece site-urile adiacente au o resursă de rutare (sau placa de interconectare) comună în arhitectura UltraScale, un PU este definit în termeni de perechi.

Cadru reconfigurabil

Cadrul reconfigurabil (CF - *Reconfiguration frame*) reprezintă cea mai mică regiune reconfigurabilă într-un FPGA. Dimensiunile de fișier *bitstream* de cadru reconfigurabil variază în funcție de tipurile de logică conținute în cadru.

Logica reconfigurabilă

Logica reconfigurabilă este constituită din orice element logic care face parte dintr-un modul reconfigurabil. Aceste elementele logice sunt modificate atunci când este încărcat un fișier parțial

BIT. Pot fi reconfigurate multe tipuri de componente logice, cum ar fi LUT-uri, flip-flop-uri, blocuri RAM și blocuri DSP.

Modul reconfigurabil

Un modul reconfigurabil (RM – *Reconfiguration Module*) este descrierea *netlist* sau HDL care este implementată în cadrul partiției reconfigurabile. Pentru o partiție reconfigurabilă sunt disponibile mai multe RM-uri.

Partiția reconfigurabilă

Partiția reconfigurabilă (RP) este un atribut setat pe o instanțiere care definește instanța ca reconfigurabilă. Partiția reconfigurabilă este nivelul de ierarhie în cadrul căruia sunt implementate diferite module reconfigurabile. Comenzile Tcl (limbaj script folosit de către Vivado pentru generarea proiectelor), cum ar fi *opt_design*, *place_design* și *route_design,* detectează proprietatea HD.RECONFIGURABLE pe instanță și o procesează.

Logică statică

Logica statică este orice element logic care nu face parte dintr-o reconfigurare parțială. Elementul logic nu este niciodată reconfigurabil parțial și este întotdeauna activ, chiar și în timpul când sunt reconfigurate RP-urile. O logică statică este cunoscută și ca logică de nivel superior.

Designul static

Proiectul static reprezintă partea din proiect care nu se schimbă în timpul reconfigurării parțiale. Proiectul static include modulele de nivel superior și toate modulele care nu sunt definite ca reconfigurabile. Proiectul static este construit cu o logică statică și o rutare statică.

4.2.2.Cerințe de proiectare și orientări

- Reconfigurarea parțială necesită utilizarea programului Vivado 2013.3 sau mai nou.
  - ♦ PR este susținută și în ISE Design Suite. Se utilizează ISE Design Suite numai pentru PR cu dispozitive Virtex-6, Virtex-5 și Virtex-4.
- Planificarea *floorplan* este necesară pentru a defini regiuni reconfigurabile, per tip de element.
  - ♦ Pentru seria 7, Pblock-urile trebuie să fie alinate vertical cu limitele cadrului de configurare CF / ceasului. Această aliniere produce cel mai bun QoR (*Quality of Reconfiguration*) și permite să fie activată *RESET_AFTER_RECONFIG*.
  - ♦ Pentru UltraScale, planificarea *floorplaning* este mai flexibilă. Xilinx recomandă oprirea Pblock pe cadrul de configurare / ceasul de regiune, pentru a permite rutarea extinsă, care poate îmbunătăți semnificativ QoR.
  - ♦ Se aplică, de asemenea, normele de aliniere orizontală.
- Sinteza *Bottom-up* / OOC (pentru a crea mai multe fișiere *netlist* / DCP) și managementul fișierelor *netlist* de module reconfigurabile sunt responsabilitatea utilizatorului.
  - ♦ Pentru instrumentele de sinteză, din altă sursă decât Xilinx, inserarea I/O trebuie să fie dezactivată.

- ♦ Pentru sinteza Vivado OOC, inserarea I/O este dezactivată automat în modul *out_of_context*.
- În mediul de dezvoltare Vivado a fost creat un set unic de verificări ale regulilor de proiectare (DRC) pentru a asigura finalizarea proiectului cu succes.
- Un design PR trebuie să ia în considerare inițierea reconfigurării parțiale, precum și livrarea de fișiere parțiale BIT, fie în cadrul FPGA, fie ca parte a proiectării sistemului.
- *Black Boxes* - cutii negre, sunt acceptate pentru generarea fișierului *bitstream*.

4.2.3.Criterii de design

Unele tipuri de componente pot fi reconfigurate, iar altele nu pot. Pentru dispozitivele din seria 7, regulile componentelor sunt după cum urmează:

-resursele reconfigurabile includ tipurile de componente CLB, RAM bloc și DSP precum și resursele de rutare;

-logica de ceas nu poate fi reconfigurată și, prin urmare, trebuie să rămână în regiunea statică. Logica include BUFG, BUFR, MMCM, PLL și componente similare.

4.2.4. Fluxul de reconfigurare parțială Vivado

Fluxul de proiectare a reconfigurării parțiale este similar cu un flux de proiectare standard, cu unele diferențe notabile. Mediul de implementare Vivado gestionează automat detaliile de nivel scăzut pentru a satisface cerințele circuitului integrat. Utilizatorul trebuie să definească structura de proiectare și planul *floorplan*. Procesarea unui design parțial reconfigurabil cuprinde următorii pași:

1. Sintetizarea modulelor statice și reconfigurabile separat.
2. Crearea constrângerilor fizice (*Pblocks*) pentru a defini regiunile reconfigurabile – *floorplaning*.
3. Setarea proprietății de HD.RECONFIGURABLE pe fiecare partiție reconfigurabilă.
4. Implementarea unui proiect complet (modulul static și unul reconfigurabil per partiție reconfigurabilă) în context.
5. Salvarea unui punct de proiectare pentru proiectarea completă.
6. Eliminarea modulelor reconfigurabile din acest proiect și salvarea proiectului static.
7. Fixarea locului static și rutarea.
8. Adăugarea de noi module reconfigurabile la proiectarea statică și implementarea acestora din nou.
9. Configurare, salvând un punct de control pentru proiectarea completă.
10. Repetarea pașilor 8 și 9 până când sunt implementate toate modulele reconfigurabile.
11. Rularea unui utilitar de verificare *(pr_verify*) în toate configurațiile.
12. Crearea fișierul *bitstream* pentru fiecare configurație

Pentru configurarea acestei partiții reconfigurabile se pot considera două metode:

a.. Interfața ICAP

Folosirea interfeței ICAP împreună cu un modul DMA [44]: fișierul bitstream este încărcat în memoria RAM externă folosind un controler implementat. Fișierul bitstream parțial este citit din memorie, este descărcat la partiția reconfigurabilă dedicată, partiția este resetată, apoi noul circuit este pornit.

Această metodă este cea mai rapidă fără sisteme adiționale, fiind necesar doar un singur circuit care controlează funcționalitățile ICAP și AXI DMA [45] [46] [47]. Problema cu metoda aceasta este că circuitul DMA are nevoie de resurse adiționale: la un circuit Xilinx Zynq XC7Z10 consumă 3278 Slice LUT logice din 17600 și 2 blocuri de memorie BRAM din 60. Alternativa este folosirea doar a circuitului ICAP și încărcarea fișierului bitstream parțial în memorie internă BRAM a circuitului FPGA, dar memoria BRAM este limitată la circuitul XC7Z010 la 2.1 MB, astfel cu mai multe fișiere parțiale memoria internă este consumată rapid.

b. Interfața PCAP

Folosirea interfeței PCAP [48] care este integrată fizic în circuitele Zynq nu consumă resurse adiționale în partea reconfigurabilă și are conectivitate cu memoria externă. În circuitele Zynq pentru controlarea reconfigurării utilizatorul poate să implementeze un program din C++ care rulează la procesorul ARM, sau reconfigurarea parțială este posibilă cu modulele kernel Xilinx în sistemul de operare Linux. Controlarea cu programul C++ este cea mai rapidă, modulul kernel integrat cu sistemul de operare Linux are nevoie aproximativ de două ori mai mult timp. O abordare netestată de autor a fost separarea procesorului. Procesorul ARM integrat în circuitele Zynq are două nuclee: unul rulează în sistem de operare linux care se poate folosi pentru diferite sarcini pentru control, monitorizare etc. și cealălalt nucleu rulează doar programul care face reconfigurarea. Metoda separării procesorului nu a fost finalizată pentru că au apărut dificultăți la comunicarea între sistemul de operare și programul rulat.

Concluzii

În acest capitol s-au studiat diferite metode de reconfigurare în tehnologiile Xilinx și Altera. Cu circuitele FPGA actuale, ca Xilinx Zynq, reconfigurarea parțială a devenit o tehnică mai simplă care se poate utiliza în mai multe proiecte cu accentul pe control. În urma studiilor s-a ales o metodă de reconfigurarea parțială, la care se folosește unitatea PCAP împreună cu un sistem de operare Linux. Această metodă este cea mai ieftină din punct de vedere al resurselor, iar folosirea sistemului de operare oferă un mediu software mai flexibil.

Capitolul 5

Reconfigurarea parțială a circuitelor FPGA pentru comanda unui quadcopter

Pentru realizarea proiectelor și modulelor prezentate în secțiunea următoare am utilizat placa de dezvoltare Digilent Zynq Zybo [49] cu circuitul FPGA SoC Zync XC7Z010-1CLG400C integrat. Cu evoluția circuitelor reconfigurabile de tip FPGA a venit o nouă generație de circuite care conține pe lângă componentele reconfigurabile și procesoare încorporate care sunt capabile pentru rularea unui sistem de operare. Acest circuit FPGA este capabil pentru reconfigurarea parțială folosind modulul prezentat PCAP. Circuitul integrat conține două nuclee de procesor ARM Cortex A9 care este dotat cu o unitate NEON cu virgulă mobilă, 256 KB memorie integrată, controler DMA cu opt canale, și două controlere SPI, I2C, CAN, UART, un controler GPIO, două controlere Ethernet.

5.1. Tehnologia de reconfigurare parțială în circuite FPGA la un vehicul aerian fără pilot (quadcopter)

Folosirea vehiculelor aeriene fără pilot a devenit un lucru comun în toată lumea. Aceste vehicule sunt utilizate în domeniul industrial, militar și privat. Cele mai folosite sunt vehiculele aeriene de tip quadrocopter (quadcopter), deoarece nu au nevoie de pistă adițională. Aceste vehicule sunt comandate cu un controler mic montat direct pe șasiu, fără capacitatea de a servi mai multe sarcini sau de a executa sarcini în paralel. Pentru sarcini mai complexe este nevoie de ajutorul (serviciile) unui calculator gazdă situat la distanță (la sol) și trebuie să aibă o legătură directă și permanentă cu calculatorul.

Ideea principală este folosirea circuitelor reconfigurabile de tip FPGA pentru controlarea vehiculelor aeriene de tip quadcopter, adică construirea a unui quadcopter care este capabil să efectueze mai multe sarcini în paralel și să aibă capacitatea de a se adapta la o sarcină nouă fără ajutor de la distanță. Calculatorul gazdă transmite doar instrucțiile și realizează monitorizarea, dar nu rezolvă sarcinile. Quadcopterul realizat va avea un circuit de tip FPGA cu un procesor încorporat care rulează un sistem de operare în timp real, având capacitatea de a reconfigura arhitectura circuitului și să schimbe algoritmii realizați în hardware. Adică vehiculul devine capabil de a se adapta la sarcini noi, să schimbe arhitectura în zbor. Figura 5.1 prezintă întregul sistem, accentul fiind pus pe arhitectura realizată.

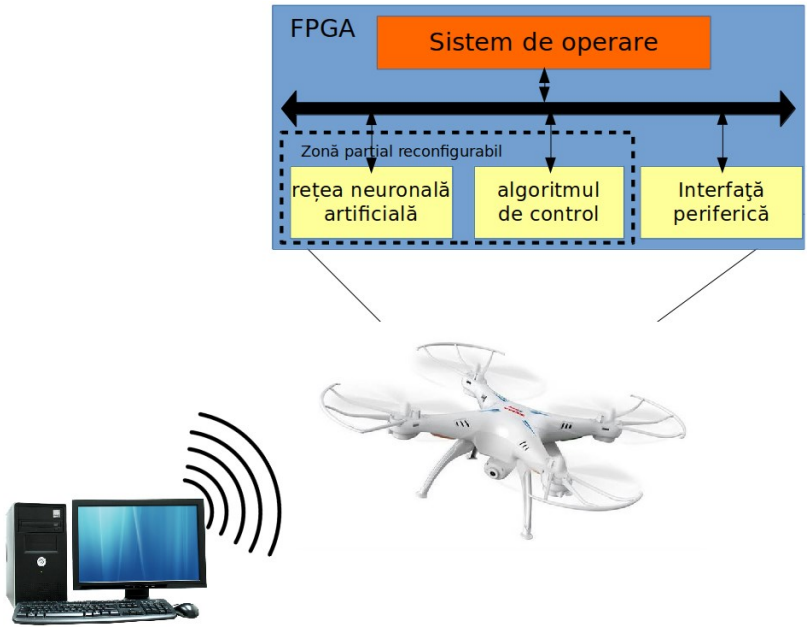

Figura 5.1 Arhitectura unui sistem quadcopter (ilustrare)

Primul pas a fost realizarea unei mediu (cadru) de bază în circuitul FPGA care poate rula un sistem de operare în timp real și acest sistem de operare să fie capabil pentru comunicarea cu circuitele reconfigurabile [50]. Sistemul de operare cel mai folosit în acest domeniu este sistemul de operare Linux [51]. Ideea principală a fost folosirea sistemului de operare Linux pentru monitorizarea circuitelor de control implementate în partea reconfigurabilă a circuitul FPGA. Circuitele de control implementate în hardware sunt mai rapide decât cele implementate în software, ele sunt capabile de a opera în timp real și de funcționare paralelă. Operatorul poate utiliza o interfață simplă și cunoscută, iar sistemul de operare poate asigura monitorizarea și parametrizarea circuitului implementat în partea reconfigurabilă.

Pasul al doilea a constat în analiza timpilor de răspuns ai sistemului prezentat ca să descoperim dacă sistemul de operare prezentat este capabil pentru controlarea și monitorizarea unui proces în timp real [52]. Timpul de răspuns este timpul care trece între intrarea unui semnal la procesul controlat și ieșirea răspunsului generat de sistemul de operare. Schema bloc a sistemului realizat este prezentată în figura 5.2.

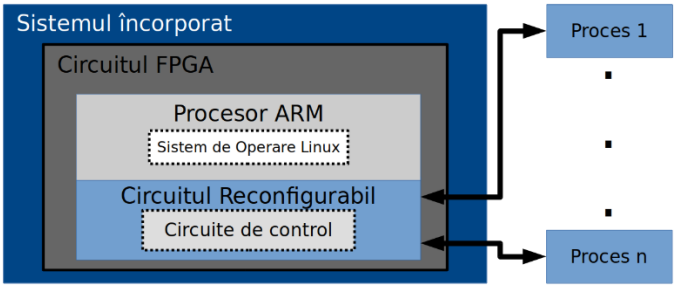

Figura 5.2 Primul proiect realizat pentru testarea sistemului de operare Linux

Sistemul încorporat este placa de dezvoltare dezvoltată de către Digilent și folosește circuitul SOC Zynq. Circuitul conține un procesor ARM dual-core care rulează sistemul de operare Linux și este modificat cu ajutorul unui *patch* numit RT-PREEMPT [53] ca să fie capabil pentru funcționare în timp real. Partea reconfigurabilă conține circuitele de control, al căror număr depinde de capacitatea circuitului FPGA.

Pentru măsurarea timpului s-a scris un kernel driver în limbajul C++, în sistemul de operare Linux [54], care așteaptă întreruperea generată de circuitul de control și este descărcat în partea reconfigurabilă a circuitului FPGA. La circuitul de control în momentul când la intrare s-a aplicat un semnal (în timpul testului acest semnal a fost generat cu un generator de semnale cu perioada de 10 ms), acesta generează un semnal de întrerupere și lansează un contor.

Când driverul kernel percepe întreruperea, se generează un răspuns și se trimite spre circuitul de control, care în momentul recepției oprește contorul. Timpul de răspuns este calculat de către numărul din contor. Timpul de răspuns a fost testat în condițiile a trei nivele diferite de utilizare ale procesorului.

Metoda de cercetare constă în măsurarea timpului care trece între intrarea unui semnal în sistem până la ieșirea semnalului de răspuns. Schema bloc sistemului de testare este prezentată în figura 5.3.

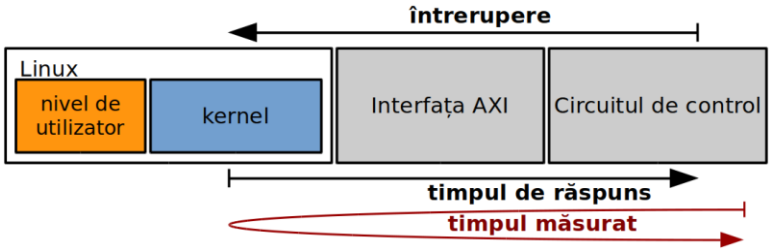

Figura 5.3 Sistemul de testare pentru operare în timp real

5.1.1. Sistemul de operare

Sarcina sistemului de operare este de a monitoriza circuitul de control implementat și de a oferi feedback utilizatorului conectat de la un calculator la distanță. Au fost examinate întârzierile cauzate de sistemul de operare, nu și întârzierile în rețea. Sistemul trebuie să gestioneze erorile și excepțiile printr-un set predeterminat de reguli care pot fi actualizate de la distanță. S-a ales sistemul de operare Linux deoarece este un sistem de operare open-source utilizat pe scară largă în industrie. Distribuția Linux selectată a fost o distribuție deja existentă numită Arch ARM Linux [55], cu un kernel modificat 4.09. Pentru ca kernel-ul să fie compatibil cu arhitectura cu chip Xilinx Zynq, s-a utilizat sursa kernel furnizată și întreținută și modificată de Xilinx. Pentru a face sistemul de operare compatibil cu placa de dezvoltare, Digilent oferă un fișier de configurare de bază a kernel-ului. Sursa kernel-ului a fost recompilată cu *patch*-ul RT-Preempt, care oferă un mecanism de blocare preemptibil, astfel încât sarcinile să poată fi întrerupte chiar și atunci când se află în faza de execuție a secțiunii critice, îmbunătățind comportamentul sistemului Linux în timp real.

După compilare kernel-ul personalizat a fost integrat în distribuția Arch ARM Linux lăsând restul distribuției intactă.

5.1.2. Metoda pentru schimbarea datei între circuitul reconfigurabil și sistemul de operare

Abordare cea mai convenabilă a constat în folosirea mijloacelor de dezvoltare ale mediului Vivado, care a generat automat un circuit schelet prin mediul de dezvoltare, care implementează un controler de întrerupere și o interfață de registru în limbajul de descriere VHDL. Sarcina utilizatorului este implementarea unei mașini de stare, care va supraveghea circuitul de control și restul modulelor implementate, realizează un protocol pentru gestionarea instrucțiunilor din registre și generează vectorii de întrerupere. Figura 5.4 prezintă schema bloc a modulului VHDL.

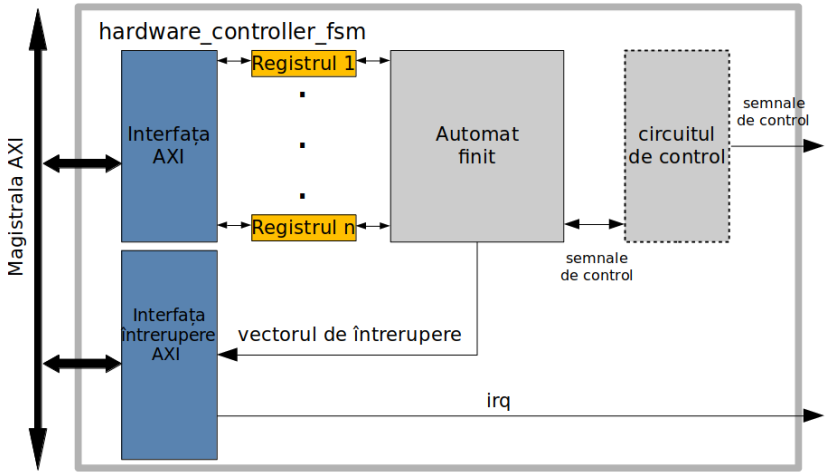

Figura 5.4 Modulul VHDL generat de mediul de dezvoltare Vivado

Interfața AXI și interfața de întrerupere AXI sunt generate automat de mediul de dezvoltare, utilizatorul trebuie doar să implementeze mașina de stare și circuitul de control. Unul dintre avantajele acestui sistem este utilizarea de întreruperi, astfel încât sistemul de operare nu trebuie să analizeze conținutul registrelor tot timpul. Atunci când intervenția este solicitată de sistemul de operare, modulul generează o întrerupere. Dezavantajul acestei abordări este acela că circuitele de control complexe sau protocoalele complexe de comunicare au nevoie de o mașină de stare sofisticată, iar la orice schimbare în protocolul de comunicație sau în circuitul de comandă, utilizatorul trebuie să rescrie mașina de stare pentru a se potrivi noului mediu. Această abordare este perfectă pentru implementarea circuitelor de control simple sau mai puțin sofisticate, cum ar fi generarea de semnale PWM sau pentru a implementa o buclă pentru scopuri de testare.

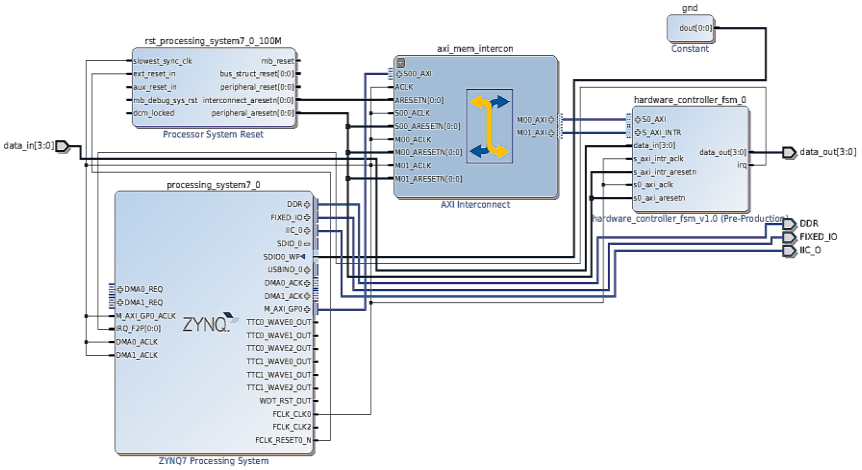

Figura 5.5 Proiectul minim pentru conectarea sistemului de operare cu circuitul reconfigurabil

Figura 5.5 prezintă proiectarea implementată a *hardware_controller_fsm_0* care conține codul VHDL (prezentat în figura 5.4). Tabelul 5.1. prezintă estimarea resurselor proiectului controler.

Resursa	Utilizare	Disponibilitate	Utilizare (%)
Slice LUTs	1094	17600	6.22
Slice Registers	1063	35200	3.02
IO	8	102	7.84
Clocking	1	32	3.12

Tabelul 5.1. Estimarea resurselor controlerului

5.1.3. Algoritmi pentru calcularea poziției unghiulare

Pasul al treilea constă în dezvoltarea algoritmilor pentru citirea și fuziunea datelor furnizate de către senzorii de localizare pentru stabilizarea quadcopterului [56]. Una dintre cele mai importante sarcini în sistemele moderne de robotizare și de automobile este stabilizarea bazată pe unghiul *pitch-roll* (tangaj-ruliu). Pentru măsurarea poziției unghiulare un senzor utilizat pe scară largă este unitatea de măsurare inerțială (IMU) [57] [58] [59]. Cu evoluția rapidă a sistemelor MEMS (Micro Electro Mechanical Systems), senzorii IMU sunt miniaturali, cu costuri reduse, cu consum redus de energie; senzorii IMU sunt integrați în multe dispozitive de consum și dispozitive robotizate.

O modalitate de obținere a unei poziții unghiulare constă în utilizarea funcțiilor trigonometrice inverse pentru a obține unghiul de la datele măsurate de accelerometru. Problema este că accelerometrul măsoară forțele multiple care acționează asupra senzorului, astfel încât chiar și o forță mică aplicată asupra senzorului afectează măsurarea, producând date zgomotoase și inutilizabile în medii cu niveluri ridicate de vibrații cum ar fi quadcopterele, mașini și nave. Datele măsurate de un giroscop sunt vitezele unghiulare în jurul celor trei axe, acestea nefiind afectate de

forțele externe, ca la accelerometru. Totodată giroscopul are un dezavantaj propriu, datele măsurate fiind vitezele unghiulare din care se poate calcula unghiul prin integrarea vitezei pe o perioadă lungă de timp, această operație producând o deviație semnificativă. Pentru a obține o măsurare unghiulară corectă, este necesară fuziunea datelor măsurate din giroscop și accelerometru. Metoda cea mai frecvent utilizată în literatura studiată pentru fuziunea senzorilor este filtrul Kalman [60] [61] [62]. Unul dintre obiectivele stabilite a fost acela de a construi un circuit cu o amprentă de circuit redusă care să poată fi integrată în sistemul FPGA. Filtrul Kalman este foarte complex din punct de vedere al implementării hardware. Utilizează matrice, are nevoie de o mare capacitate de calcul, prin urmare au fost explorate alte alternative. O abordare promițătoare a fost filtrul complementar.

Rezultatele din articolele [63] [64] [65] [66] prezentate despre fuziunea senzorilor utilizând filtrul complementar au fost convingătoare, arătând că algoritmul complementar este o alternativă bună pentru fuziunea senzorilor, algoritmul necesind un set redus de pași pentru a calcula rezultatul și are o precizie bună. Unii cercetători au susținut chiar că, în unele cazuri, filtrele complementare obțin rezultate mai bune decât filtrul Kalman [67]. De cele mai multe ori acești algoritmi au fost testați și implementați pe microcontrolere, implementarea lor pe circuite FPGA ar oferi oportunități mai mari de paralelizare. De exemplu dacă calculul poziției unghiulare se face printr-un circuit separat, care este conectat la un microprocesor, singura sarcină pentru microprocesor este de a citi datele, astfel nefiind necesare instrucțiuni suplimentare pentru obținerea valorilor poziției unghiulare de la un senzor IMU dat. Algoritmul implementat pe circuite FPGA ar fi portabil între dispozitive reconfigurabile, făcându-l ideal pentru a fi utilizat pe dispozitive hardware dedicate pentru stabilizarea unghiulară.

Scopul final a fost de a implementa și testa filtrul la nivel de circuit pentru a obține o precizie relativ ridicată și o utilizare echilibrată a resurselor și timpilor de execuție.

Filtrul complementar pentru fuziunea datelor IMU a fost propus de S. Colton în 2007 [68]. Pentru a obține poziția unghiulară folosind accelerometrul, se determină orientarea vectorului de gravitație, care este întotdeauna măsurat de către accelerometru. Aceasta poate fi calculată folosind funcția trigonometrică inversă arctangentă. Principala problemă, așa cum s-a menționat în introducere, este că accelerometrul măsoară forțele multiple care acționează asupra senzorului. Dacă senzorul este static și acționează numai forța G asupra senzorului, în sens vertical, unghiul măsurat de accelerometru este corect. Într-un mediu dinamic în care senzorul se deplasează de-a lungul celor trei axe, chiar și o mică forță aplicată asupra senzorului afectează măsurarea, făcând ca unghiul calculat să fie incorect.

Principiul în care filtrul complementar combină datele obținute de la cei doi senzori este ilustrat în figura 5.6.

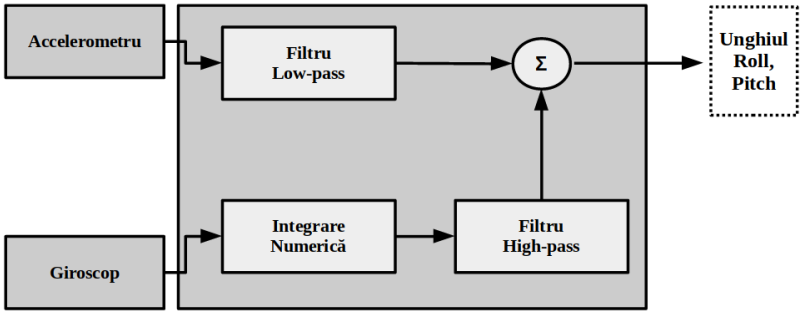

Figura 5.6 Schema bloc a filtrului complementar

Datele măsurate de către accelerometru sunt fiabile pe perioade lungi de timp. Un filtru trece jos (*low-pass*) aplicat pe datele măsurate filtrează zgomotul de la forțele mici care acționează asupra senzorului (influențe cu caracter de zgomot), corectând unele dintre perturbațiile din date.

Giroscopul măsoară viteza unghiulară (grad / s), care nu este influențată de forțele exterioare. Dacă viteza unghiulară este integrată, se poate obține poziția unghiulară. Totuși, datorită procesului de integrare în timp, datele giroscopului încep să nu se întoarcă la zero când senzorul este întors în poziția inițială. Datele giroscopului sunt fiabile pe termen scurt. Prin urmare, aplicarea unui filtru de tip trece sus (*high-pass*) corectează eroarea introdusă de derivă.

Filtrul complementar combină filtrul *low-pass*, care filtrează fluctuațiile schimbărilor pe termen scurt (vârfurile cauzate de forțele mici care acționează asupra accelerometrului) cu filtrul *high-pass* aplicat pe datele giroscopului, care permite semnalelor cu durată scurtă să treacă prin întreruperea fluctuațiilor pe termen lung care cauzează deriva datelor giroscopului.

Reprezentarea matematică a filtrului complementar este exprimată prin ecuația:

$$\theta angle=\alpha*(\theta angle+\omega gyro*dt)+(1-\alpha)*aacl \quad , \qquad (5.1)$$

unde $\theta angle$ este unghiul estimat, α - coeficientul filtrului, $\omega gyro$ - viteza unghiulară măsurată de giroscop, dt este perioada de timp a eșantionului și $aacl$ este unghiul obținut utilizând funcția arctangentă de la accelerometru. Coeficientul α este calculat de ecuația:

$$\alpha= \tau/(\tau+dt) \quad , \qquad (5.2)$$

unde τ este constanta de timp a filtrului. Înainte de aplicarea ecuației, datele măsurate de giroscop și accelerometru trebuie să fie reglate la zero și scalate. Datele scalate greșit conduc la rezultate false sau înșelătoare. Filtrul *low-pass* filtrează semnale care sunt mai scurte decât constanta de timp, iar filtrul *high-pass* face invers. În fiecare perioadă de timp, datele giroscopului sunt integrate cu unghiul curent filtrat de filtrul *high-pass*, apoi combinate cu datele din accelerometrul filtrat de filtrul *low-pass*. Rezultatul este poziția unghiulară care este mai precisă, decât pozițiile unghiulare calculate de la datele accelerometrului prin funcția trigonometrică arctangentă. Coeficientul α al filtrelor se modifică prin reglarea constantei de timp τ care este stabilită prin încercare și eroare. S-au obținut rezultate bune cu valoarea coeficientului de aproximativ $\alpha = 0{,}95$.

5.1.4.Implementarea filtrului complementar

Pentru a atinge un nivel relativ mare de precizie, numerele de 24 biți au fost utilizate cu punct fix în complementul față de doi pentru reprezentarea numărului cu partea fracționară de 16 biți. În continuare, procesul de dezvoltare va fi prezentat prin pașii întreprinși pentru implementarea algoritmului pentru atingerea scopurilor de proiectare menționate anterior. Figura 5.7 prezintă schema bloc a circuitul implementat:

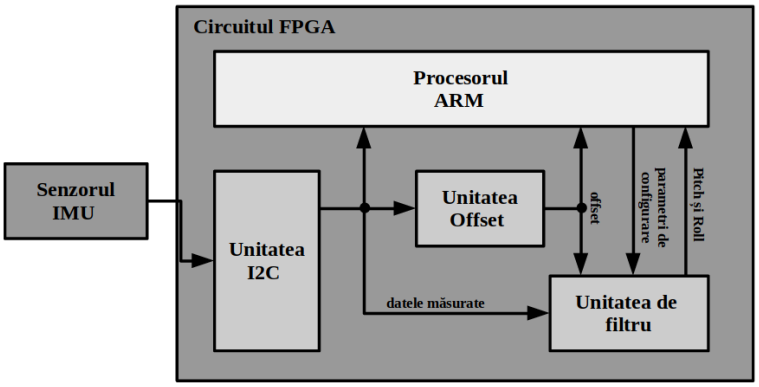

Figura 5.7 Schema bloc a circuitul complementar implementat

Modulul I2C este o mașină de stare VHDL care a fost implementată pentru interfațarea cu senzorul IMU. Modulul se ocupă de comunicarea I2C cu senzorul. Unitatea Offset calculează valorile de offset necesare pentru ajustarea la zero a datelor senzorului. Unitatea de Filtru implementează filtrul complementar. Procesorul ARM se ocupă de parametrizare și de înregistrarea datelor.

Pentru ca algoritmul prezentat în ecuația de mai sus să funcționeze, datele brute trebuie să fie ajustate la zero și scalate. Ajustarea la zero se face prin măsurarea unui set de date de la poziția zero a senzorilor unde se fixează valoare zero pentru unghi și se calculează o medie care este scăzută din fiecare măsurătoare următoare, astfel încât în poziția de bază valorile sunt zero. Offsetul este calculat de modulul offset care calculează o valoare medie din 64 de eșantioane de date; acest număr a fost ales astfel încât divizarea se poate face printr-o operație simplă de *shift right*, oferind o valoare medie destul de precisă, cu o dimensiune redusă a resursei. După ce se calculează offsetul, modulul de filtrare poate să eșantioneze datele și să calculeze valorile de *roll* și *pitch*. Valorile offset sunt calculate o dată în timpul procesului de inițializare, motiv pentru care latența acestui modul nu a fost calculată în latența generală a filtrului complementar implementat.

Pentru calcularea valorilor unghiulare din datele brute modulul de filrare urmează secvențele prezentate în figura 5.8.

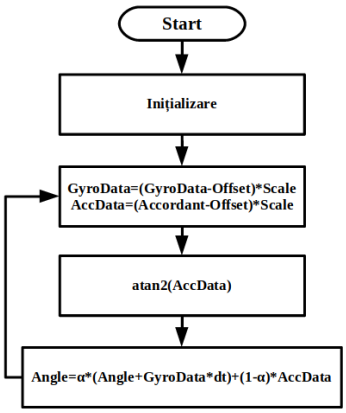

Figura 5.8 Schema de executare a filtrul complementar

Urmărind procesul de implementare, s-au testat mai multe scheme de proiectare pentru a optimiza dimensiunea și latența circuitului.

Mașinile de stare de control au fost implementate astfel încât datele de la accelerometru să fie procesate în paralel cu datele din giroscop în două linii de procesare separate. Pe lângă aceste două linii de procesare paralelă, la fiecare pas ecuația prezentată a fost de asemenea executată paralel pe toate axele senzorilor (axele *x, y ,z*). Pentru a reduce utilizarea resurselor s-au considerat trei abordări diferite. **Prima abordare** a dat cel mai rapid timp de calcul, dar a rezultat și un circuit mai complex și mai mare în dimensiune. Paralelizarea înseamnă că sunt utilizate mai multe unități aritmetice pentru fiecare calcul separat. Avantajul acestei implementări a fost că a dat cel mai rapid timp de execuție: un ciclu este calculat în 260 ns, cu frecvența de ceas de 100 MHz. Calculul cel mai complex este necesar pentru calcularea funcției trigonometrice inverse arctangentă (*atan2*). Calculul poate fi realizat prin utilizarea algoritmului CORDIC [69], circuitul prezentat utilizează nucleul IP CORDIC furnizat de Xilinx. Circuitul FPGA folosit pentru cercetare a fost Xilinx Zynq-XC7Z010. Acest circuit are cea mai mică capacitate de celule programabile de 28K, în familia de circuite Zynq FPGA. Prima abordare de proiectare, descrisă până aici, utilizează aproximativ 16% din suprafața reconfigurabilă disponibilă, deoarece mărimea mică a fost un criteriu de optimizare a obiectivelor pentru a reduce dimensiunea circuitului. **A doua abordare** a constat în eliminarea paralelizării circuitelor CORDIC, adică calcul paralel pe axe, care a redus imediat dimensiunea circuitului folosind doar 9% din suprafața reconfigurabilă. Timpul de execuție a fost ridicat la 440 ns. Această mică modificare a redus dimensiunea circuitului la jumătate din dimensiunea originală fără alte modificări.

Pentru a reduce unitățile aritmetice necesare pentru calcule, a fost dezvoltată o a **treia abordare** de proiectare**.** Executarea se face în același mod numai că în fiecare etapă unitatea aritmetică a fost refolosită pentru fiecare calculare a axei, adică toate calculele de adunare, multiplicare au fost de asemenea fuzionate cu aceeași unitate aritmetică. Acest lucru a fost obținut prin utilizarea unei mașini de stare de control mai complexe și a unor registre suplimentare și a multiplexorilor care să țină datele între etape. Rezultatele nu au fost convingătoare, deoarece unitățile aritmetice au fost

înlocuite cu registre și multiplexoare care au condus la o schimbare de 0,04% a dimensiunii comparativ cu a doua abordare. Singura modificare semnificativă a fost nevoia redusă a modulelor DSP. Timpul de execuție a crescut la 610 ns. Un rezumat mai detaliat privind utilizarea zonelor poate fi văzut în tabelul 5.2.

Timpii de execuție între 260 și 610 ns oferă o latență bună. Majoritatea MEMS [70] [71] [72] au o rată medie de ieșire a datelor de aproximativ 1 kHz atât la accelerometru cât și la giroscop ca să actualizeze datele.

Abordare	Zonă	Utilizare/Disponibil	Grad de utilizare %
Design 1.	Slice LUT	2693/17600	15.30
	Slice Flip-Flop	2910/35200	8.26
	LUT-RAM	8/6000	0.13
	DSP modules	21/80	26.25
Design 2.	Slice LUT	1556/17600	8.84
	Slice Flip-Flop	1799/35200	5.11
	LUT-RAM	4/6000	0.07
	DSP modules	21/80	26.25
Design 3.	Slice LUT	1548/17600	8.80
	Slice Flip-Flop	1765/35200	5.01
	LUT-RAM	4/6000	0.07
	DSP modules	4/80	5

Tabel 5.2 Utilizarea resurselor zonelor reconfigurabile

Calculul cu date noi de 1 kHz, disponibile la fiecare 1 ms, în cel mai neavantajos caz modulul implementat calculează unghiurile în 0.61 µs sumat cu latența I2C, timpul necesar pentru a obține toate datele axei de la giroscop și accelerometru și a da o ieșire paralelă avem o latență generală de 371 µs.

Dacă circuitele prezentate citesc date noi în intervale de 1 ms și pozițiile unghiulare se calculează în 0,371 ms, atunci este un interval de 0,629 ms între datele citite consecutiv pentru orice circuit de comandă sau procesor ca să calculeze ieșirile de comandă utilizând pozițiile unghiulare date de circuitul prezentat. Protocolul de comunicație I2C este unul serial cu o viteză maximă de 400 kHz. Din timpul de execuție prezentat mai sus este vizibil faptul că, chiar și cu un modul hardware dedicat, este nevoie de 370 µs pentru a colecta toate datele axelor de la senzori, ceea ce face ca colectarea datelor să fie lentă.

Rezultatele măsurate în poziția zero cu filtrul complementar implementat sunt prezentate în figurile următoare, în care se poate constata că rezultatele calculate cu Matlab sunt aproape identice cu rezultatele calculate de către circuitul implementat. În figura 5.9 se arată o măsurare în mediul static: se poate constata că unghiul *roll* obținut, folosind numai funcțiile arctangentă (ACC cu roșu), este zgomotos, cu multe vârfuri în timpul măsurătorii. Poziția unghiulară calculată de funcția arctangentă este incorectă. Rezultatele măsurării statice ar trebui să producă valori în jurul valorii

de zero, deoarece nu se face nicio mișcare de către senzor, în schimb se calculează valori între -30 și 30 grade care ar perturba orice algoritm de control aplicat pentru controlul poziției unghiulare.

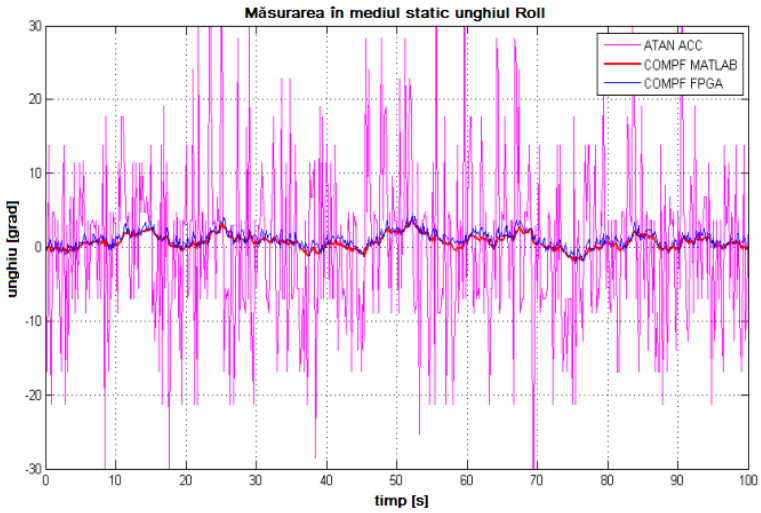

Figura 5.9 Rezultatele măsurate cu filtrul complementar implementat, comparat cu calcularea arctangentei

Rezultatele obținute cu filtrul complementar sunt mult mai fluide fără oscilații aleatoare mari, diferențele dintre rezultatele obținute din MATLAB și din circuitul implementat sunt mici. Figura 5.10 prezintă doar rezultatele din MATLAB și cu circuitul implementat.

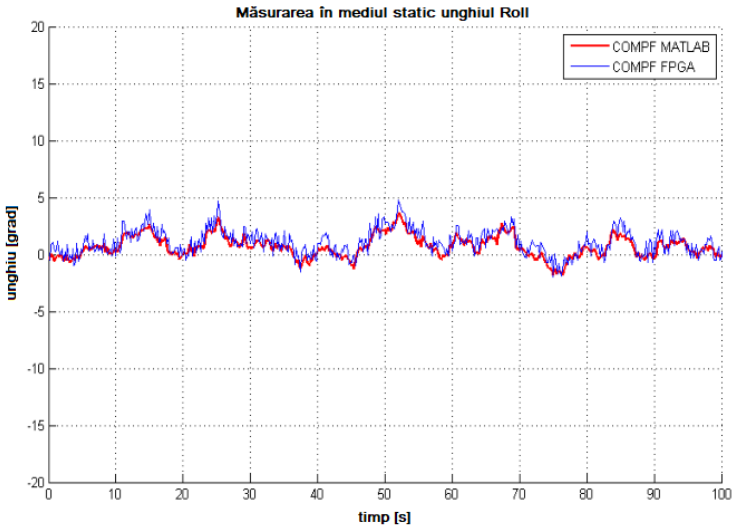

Figura 5.10 Rezultatele măsurate cu filtrul complementar implementat, comparat doar cu rezultatele obținute în Matlab

Diferențele minime sunt observabile între cele două rezultate. Rezultatele obținute din MATLAB au fost calculate utilizând valori în virgulă mobilă cu precizie dublă utilizând funcția arc-tangentă încorporată. Circuitul FPGA utilizează valori de 24 biți cu o parte fracționară de 8 biți. Rezultatele diferă datorită erorilor cumulate aduse de mărimea fracției mici.

Figura 5.11 prezintă rotații de măsurate dinamic în jurul axei Y (*pitch*). Filtrul implementat reușește să calculeze valorile corecte comparativ cu filtrul MATLAB. Figura 5.12 prezintă aceeași măsurătoare pe un interval mai mare, în care au fost de asemenea reprezentate pentru comparație și rezultatele funcției arctangentă. În comparație cu rezultatul filtrului complementar, unghiul calculat de funcția arctangentă aduce un zgomot semnificativ cu vârfuri multiple pe parcursul măsurătorilor. Unghiul calculat are o deplasare globală semnificativă de 50 de grade. Datele obținute numai de accelerometru sunt inadecvate pentru măsurători precise ale poziției unghiulare.

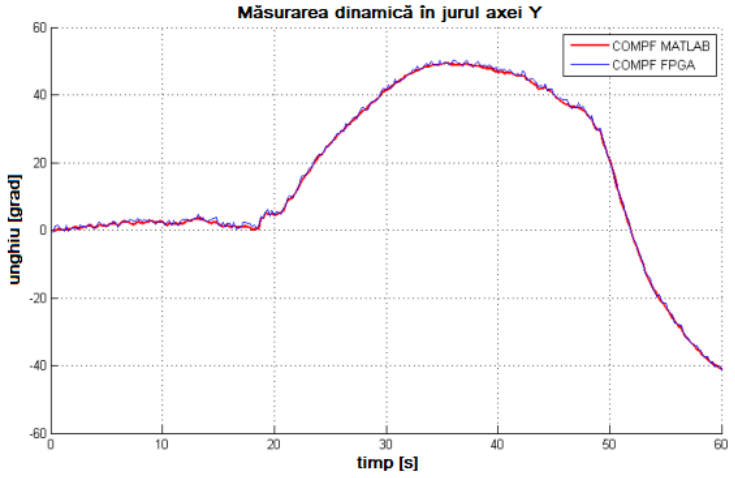

Figura 5.1 Rezultatele măsurării dinamice în jurul axei Y (Pitch)

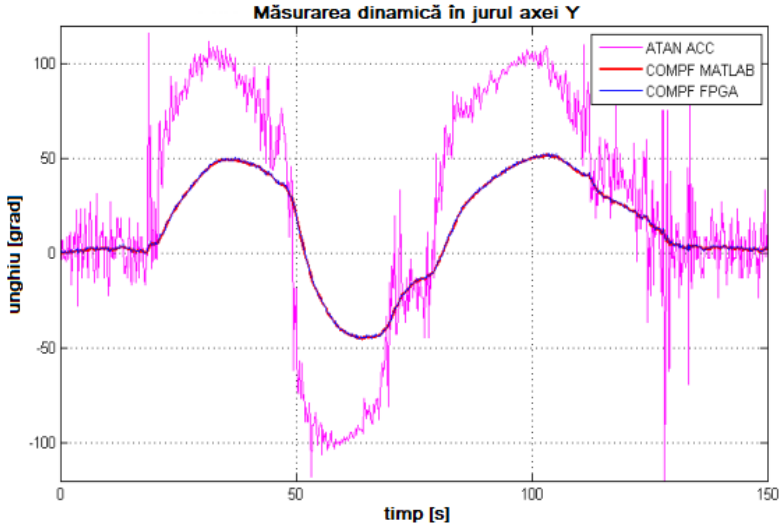

Figura 5.2 Rezultatele măsurării dinamice în jurul axei Y (Pitch) îngtr-un interval mai lung

Figura 5.13 prezintă o măsurare în care, în timpul rotației, s-a făcut o mișcare rapidă. Se observă că această mișcare rapidă a perturbat datele accelerometrului obținute prin funcția arctangentă, care devine distorsionată, inutilizabilă. Filtrul complementar reușește să urmărească modificarea

vârfurilor de mici dimensiuni, care sunt, de asemenea, observabile la măsurare, dar datele globale rămân neîntrerupte.

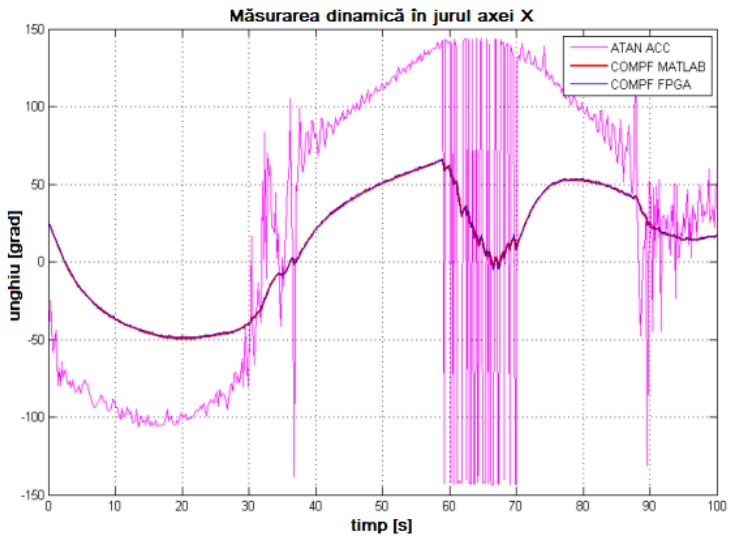

Figura 5.3 Rezultatele măsurărilor dinamice în jurul axei X (Roll)

5.2. Implementarea reconfigurării parțiale în mediul de bază

Pentru reconfigurarea parțială un aspect important a fost mărimea zonei reconfigurabile, pentru a fi potrivită implementării unor circuite mai complexe. Pentru calcularea poziției unghiulare partiția reconfigurabilă trebuie să fie atât de mare încât să se potrivească cu circuitul, iar când quadcopterul nu este în zbor, să fie posibilă refolosirea partițiile pentru alte sarcini. Fără partiționare proiectul minimal care este necesar pentru rularea unui sistem de operare consumă un spațiu mai puțin de 10 % a circuitului FPGA.

În figura 5.14 regiunile cu culoarea albastru închis sunt unitățile logice programabile, numite Slice. Un Slice conține două celule logice CLB. Zona cu culoare portocalie este procesorul integrat ARM. Zona aceasta a circuitului nu este reconfigurabilă. Utilizatorul poate efectua modificări la registrele de configurare, dar nu în arhitectura principală. Zona reconfigurabilă este divizată în șase zone majore. În aceste zone sunt folosite diferite nivele logice de semnal. Aceste zone conțin logica reconfigurabilă, adică celule logice. Zonele cu culoarea albastru-deschis sunt unitățile logice folosite de către utilizator adică proiectul dezvoltat care este descărcat în circuitul FPGA, restul circuitului este inițializat dar nu este folosit.

După planificare *floorplanning* și implementarea filtrului complementar structura circuitului FPGA va fi modificată, aceasta fiind prezentată și în figura 5.14:

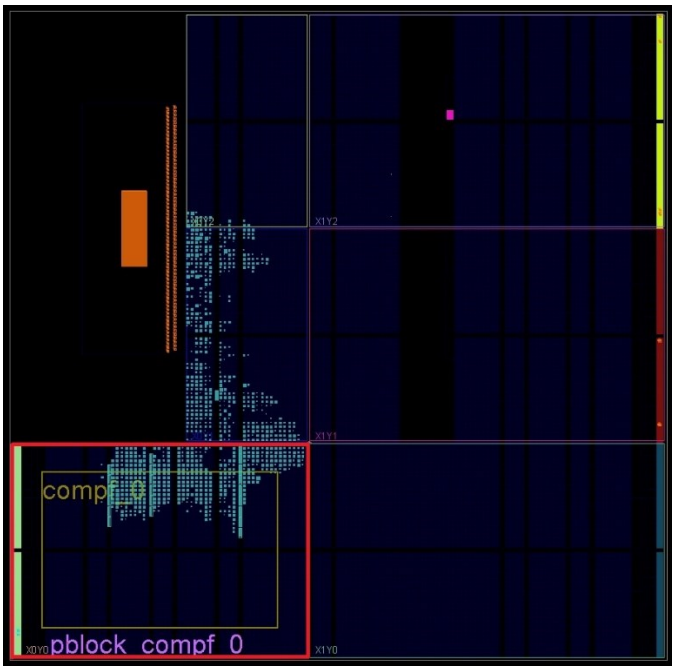

Figura 5.4 Schema hardware FPGA cu zona reconfigurabilă configurată

Blocul *pblock_compf_0* este partiția reconfigurabilă care este inițializată cu filtrul complementar. Această partiție va conține după prima configurare filtrul complementar, dar utilizatorul are posibilitatea de reconfigurare a oricărui circuit care se potrivește în zona marcată cu culoarea roșie. Partea statică de lângă procesorul ARM, la care s-a schimbat reconfigurarea necesită elemente logice suplimentare, iar pentru interfațarea semnalelor cu filtrul complementar circuitul trebuie să fie plasat fizic lângă granița partiției reconfigurabile. Reconfigurarea parțială este efectuată și monitorizată din sistemul de operare Linux prezentat. Xilinx oferă un driver kernel numit *devcfg* care citește fișierul *bitstream* parțial din memorie sau orice dispozitiv de stocare a datelor atașate și face reconfigurarea. În timpul reconfigurări magistrala AXI este blocată pentru evitarea semnalelor false. După reconfigurare *devcfg* resetează partiția reconfigurabilă sistemul se întoarce la funcționarea normală. Timpul de reconfigurare depinde de mărimea fișierului *bitstream* parțial. Un fișier bitstream complet pentru circuitul Zync ZC7010 este de 2083 kB. Fișierul parțial prezentat pentru zona *pblock_compf_0* este de 462 kB, reconfigurarea parțială durează între 1-2 µs (doar zona *pblock_compf_0* este schimabtă). Timpul depinde cât de utilizat este sistemul de operare, câte operațiuni critice trebuie să fie terminate înainte de blocarea magistralei AXI. Reconfigurarea totală a circuitului în sistemul de operare prezentat durează cca ~25 ms.

5.2.1. Algoritmul de control pentru stabilizare

Pentru stabilizare s-a utilizat algoritmul de control PID și circuitul implementat la subcapitolul 2.2. Senzorul IMU BOSCH BNO055 a fost schimbat cu Polulu MinIMU [73] care dă doar datele brute, calcularea unghiurilor este efectuată de circuitul complementar prezentat în subcapitolul 5.1. Schema bloc a sistemului implementat este prezentată în figura 5.15.

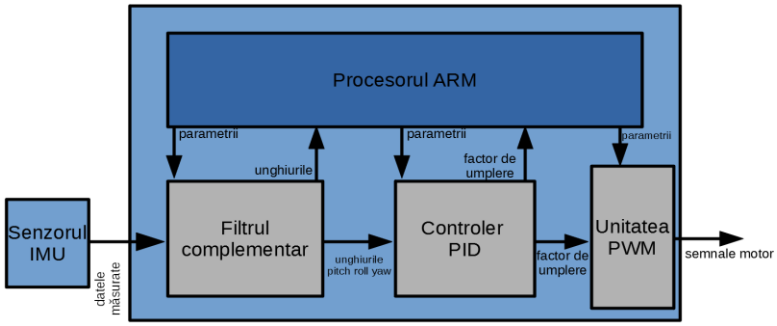

Figura 5.5 Schema bloc a circuitelor de control ale quadcopterului

Algoritmul complementar calculează poziția unghiulară în 610 ns, unitatea PID calculează semnalul de control a motoarelor în 510 ns cu o fervență de ceas 50 MHz. Sistemul de operare Arch Linux monitorizează și parametrizează sistemul. Utilizatorul trimite semnalele de referință pentru unghiurile pitch, roll, yaw și o valoare de referință pentru motoarele care asigură viteza minimă de rotație. Sistemul de operare primește aceste valori prin rețea wireless și le trimite spre circuitul PID, care calculează valorile de control. Sistemul implementat este capabil să ruleze cu o frecvență de eșantionare de 1,2 ms. Pentru stabilizarea quadcopterului sunt necesare aceste două circuite plus mașina de stare, care controlează etapele implementate în software la sistemul de operare. Acest sistem poate fi controlat de către utilizator pentru un comportament autonom, pentru care sunt necesare circuite adiționale.

5.2.2. Integrarea reconfigurării parțiale

După dezvoltarea algoritmilor pentru stabilizarea quadcopterului, rezultatul proiectării s-a integrat în sistemul reconfigurabil (figura 5.16).

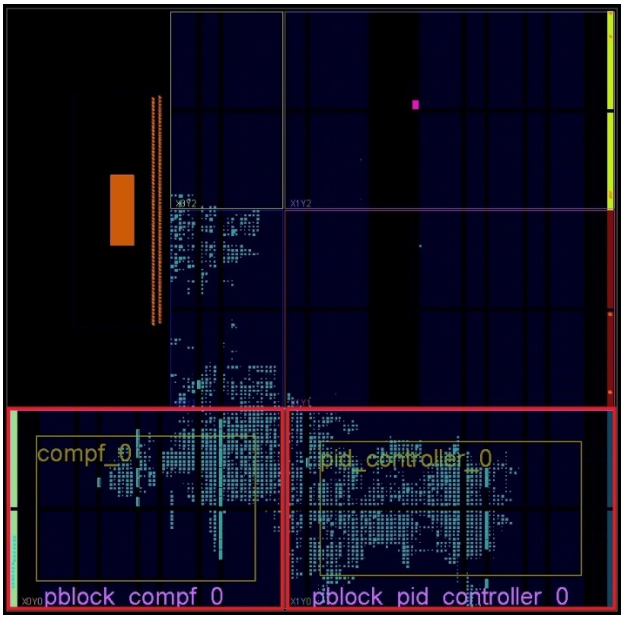

Figura 5.6 Schema hardware FPGA: zona reconfigurabilă pentru filtrul complementar și zona reconfigurabilă pentru circuitul de control

Partiția *pblock_pid_controller_0* conține circuitul de control PID. Pentru ca circuitul să fie capabil pentru controlul zborului quadcopterului, este necesar și filtrul complementar (partiția *pblock_compf_0*)și circuitul de control să fie înglobate la partiția *pblock_pid_controller_0.*

5.2.3.Integrarea circuitelor de detectarea deplasării în proiectul reconfigurabil

Acest sub-proiect a fost dezvoltat în mare parte din mediul HLS pentru simplificarea implementării și paralelizarea algoritmilor la un nivel mai bun. Dezavantajul a fost complexitatea și mărimea circuitului. În figura 5.17 se arată schema hardware implementată pe circuitul Zynq ZC7010.

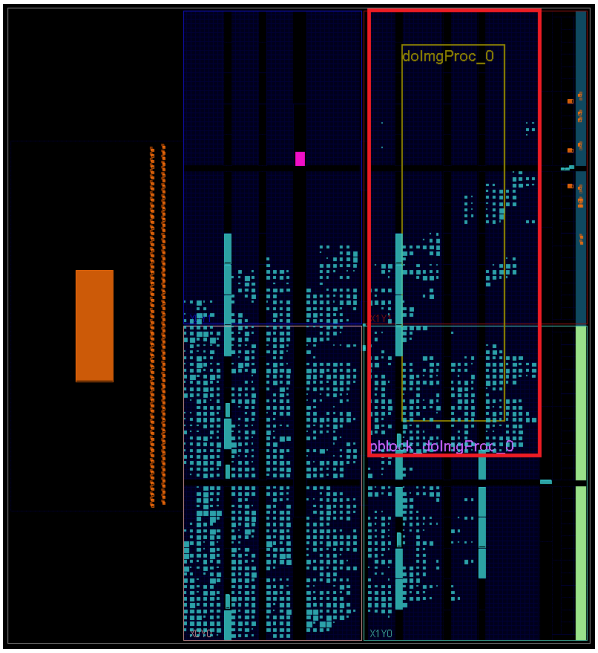

Figura 5.17 Schema hardware a zonei de reconfigurare a filtrului Canny

În zona statică este circuitul care citește datele de la camera video (figura 3.6). Circuitul care generază semnalele HDMI, adică semnalul VideoOut, nu este necesar, deoarece quadcopterul nu conține monitoare. Zona *pblock_doImgProc_0* conține filtrul Canny și circuitul de procesare prezentat (capitolul 3). Este vizibil că partiționarea prezentată în capitolele precedente nu este atât de mare ca să conțină și filtrul Canny. Zona a fost lărgită dar partea statică este încă prea mare. Pentru un proiect mai general partea reconfigurabilă trebuie să conțină și circuitele de VideoInput. Dacă folosim un UAV fără cameră video atunci circuitul VideoInput devine inutil. Proiectul a fost mutat la un circuit Zynq cu capacitate mai mare ZC7020. Figura 5.18 prezintă proiectul implementat în acest circuit.

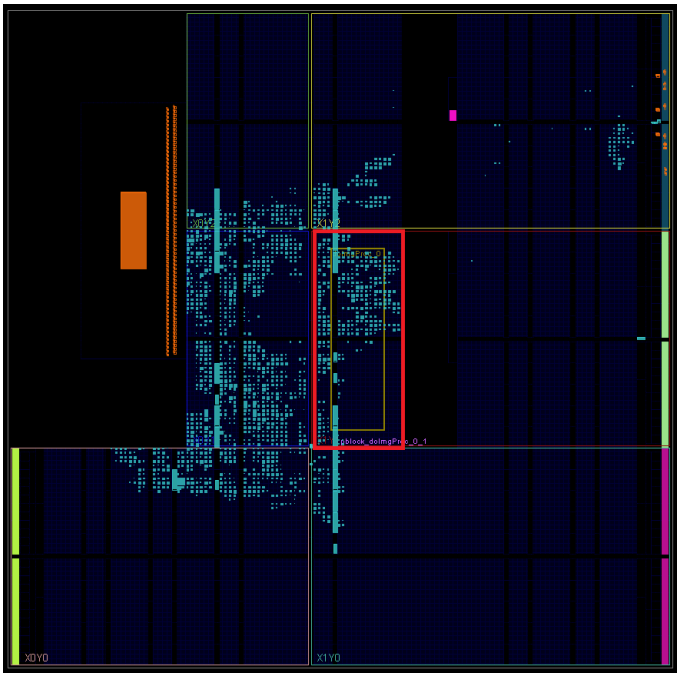

Figura 5.18 Schema hardware ZC7020

Se poate constata că același bloc care conține doar filtrul Canny *pblock_doImgProc_0* ocupă doar 12% a circuitul FPGA. Fișierul de bitstream este 3950 de kB, fișierul parțial 156 kB, timpul reconfigurării parțiale este de cca 1 µs.

Recapitulând în acest capitol s-a prezentat o metodă pentru controlarea unui quadcpoter cu implementare pe circuite FPGA. Proiectul este compus din circuite prezentate în capitolele precedente. Cu folosirea tehnicii de reconfigurare utilizatorul are posibilitatea să folosească mai multe circuite în același sistem. Folosirea limbajului VHDL împreună cu mediul de dezvoltare System Generator și HLS face implementarea și testarea algoritmilor mai rapide și eficiente. În paralel cu dezvoltarea circuitelor de control pentru quadcopter se prezintă în figura 5.19 un mediu de control și monitorizare universală., care s-a realizat și testat.

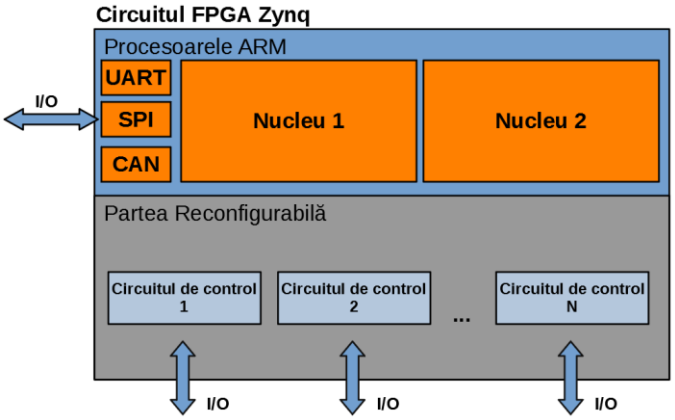

Figura 5.19 Schema bloc a proiectului

Pe procesorul ARM se rulează un sistem de operare Arch linux, care are un kernel compilat cu toți parametrii necesari pentru funcționarea în timp real, incluzând modificarea RT-PREEMPT. Sistemul de operare a fost capabil pentru rulare în timp real cu limita de a putea să perceapă o întrerupere în 9,13 µs [50], ceea ce înseamnă că semnalele generate de către algoritmii implementați în software au nevoie de 9,13 µs ca să ajungă la circuitul FPGA în partea reconfigurabilă și întreruperea generată de către acest circuit are nevoie de încă 9,13 µs pentru ca semnalul să se întoarcă. Algoritmii complementari pentru calcularea unghiurilor și algoritmul PID pentru stabilitate, care s-au dezvoltat mai sus, fiind implementați, rezultă că sistemul a reușit să stabilizeze quadcopterul (figura 5.20).

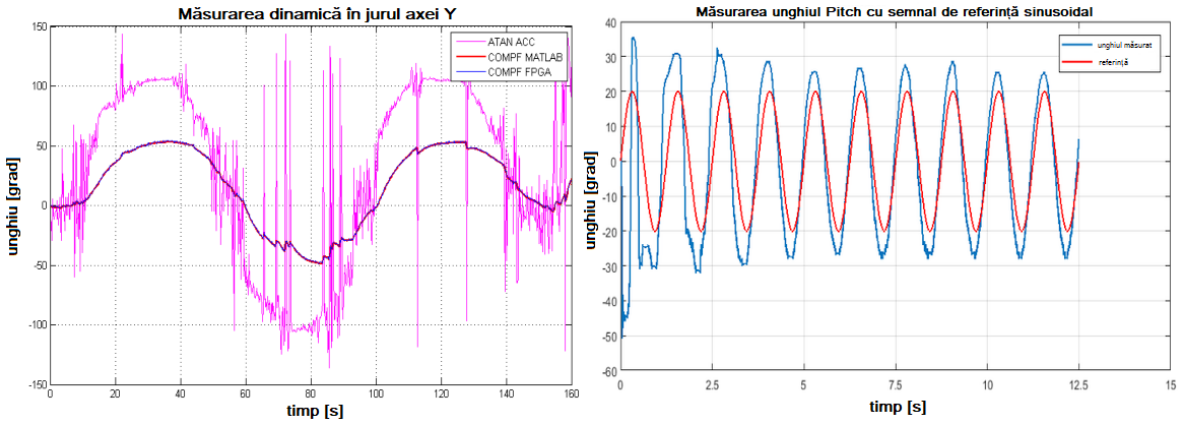

Figura 5.20 Rezultatele algoritmilor de stabilizare

În final metoda de reconfigurare parțială a fost introdusă și pentru optimizarea utilizării resurselor și pentru extinderea capabilităților în proiectul quadcopter cu rețele neurale sau procesarea imaginilor. Dar mediul final prezentat și în figura 5.21 poate fi utilizat ca un proiect de bază și în alte sarcini de control.

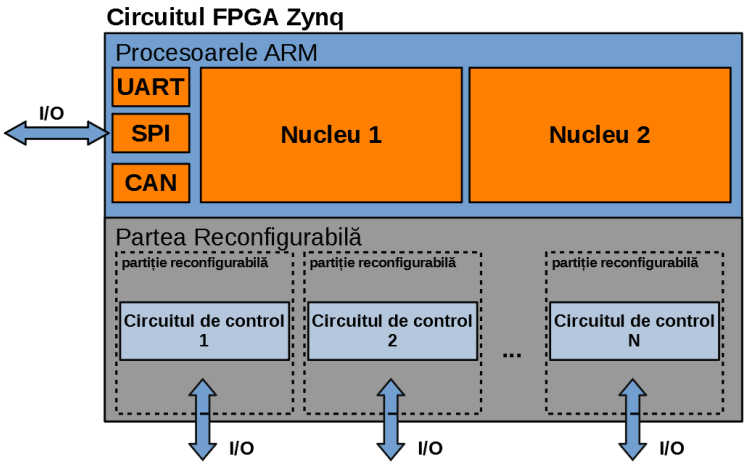

Figura 5.21 Schema bloc a proiectului cu partițiile reconfigurabile

Utilizatorul dezvoltă circuitele de control nr 1, 2 etc., le descarcă în partiția reconfigurabilă cu sistemul de operare, resetează partiția și poate să utilizeze, să monitorizeze sau să parametrizeze imediat circuitul dezvoltat.

Concluzii

În acest capitol se dezvoltă în detaliu modul de implementare a tehnicii de reconfigurare pe circuite FPGA Xilinx Zynq, pe baza aplicației de control al zborului unui quadcopter. Se utilizează arhitectura disponibilă, în sensul construirii proiectului în jurul prosecorului ARM și a părții configurate pentru filtrul Canny (procesarea de indentificare a muchiilor imaginilor mobile), cu părți de reconfigurare. Astfel se prezintă utilizarea sistemului de operare Linux pentru sarcinile de control, blocurile pentru filtrarea semnalelor de la senzorii de accelerație și giroscopici (filtru complementar), algoritmul de control PID. Se analizează prin comparație performanțele de sistem în cazul determinării parametrilor de zbor cu Matlab, cu implementarea în FPGA și cu calculul matematic (numai cu arctangenta) de procesare.

Se prezintă trei abordări ale implementării sistemului de control și reconfigurare FPGA, analizând performanțele sistemelor, gradul de utilizare a resurselor. În urma cercetării s-a prezentat un sistem cadru bazat pe tehnica de reconfigurare în circuite FPGA, care are capacitatea pentru implementarea diferitelor sarcini de control,.

Capitolul 6.
Concluzii finale. Contribuţii originale. Diseminarea rezultatelor. Directii viitoare de cercetare

Scopul cercetării a fost crearea unui mediu de dezvoltare pentru controlul și monitorizarea diferitelor sarcini de control în circuite FPGA. Mediul dezvoltat trebuie să fie flexibil și ușor de modificat pentru diferiți algoritmi. Din acest motiv mediul de dezvoltare trebuie să fie capabil pentru utilizarea tehnicii de reconfigurare parțială.

Cercetarea a început cu proiectarea și implementarea în FPGA a unui procesor complet dedicat pentru sarcini de control (procesorul EMMC, capitolul 1.). Procesorul a fost dezvoltat în limbajul VHDL și este capabil pentru calcularea cu mare precizie a diferiților algoritmi matematici, dar are un set relativ mic de instrucțiuni, ceea ce a făcut dificilă integrarea în sisteme încorporate mai complexe. În grupul de cercetare s-au dezvoltat încă două arhitecturi de procesoare implementate în FPGA (procesorul SLP – dezvoltat la Universitatea Sapientia-Tg.Mureș și procesorul SCP – dezvoltat la NUI Galway) și s-a procedat la compararea performanțelor celor trei implementări în FPGA cu un procesor de referință, Pico Blaze de la Xilinx. Rezultatele comparării sunt satisfăcătoare, rezultă că se pot implementa chiar și procesoare în FPGA pentru dezvoltări de control, pe același suport FPGA.

În capitolul al doilea s-a prezentat un circuit dedicat de control PID care a fost dezvoltat pentru stabilizarea unghiurilor de înclinare ale unui vehicul aerian tip quadcopter, implementat în Simulink System-Generator. System-Generator permite modelarea și implementarea algoritmilor matematici complecși în circuite FPGA. Acest capitol s-au prezentat avantajele dezvoltării de aplicații în FPGA utilizând mediul System-Generator.

În al treilea capitol se prezintă o metodă pentru prelucrarea imaginilor și detectarea deplasării în imaginile captate de camere video, ca parte integrantă a unor aplicații de control în sisteme. În acest capitol s-a introdus mediul de dezvoltare HLS (High Level Synthesis), prin intermediul căruia algoritmii prezentați pentru filtrul Canny s-au implementat în C++. În partea aceasta a cercetării s-a făcut observația că HLS consumă o mare parte din resursele disponibile. În circuitul Xilinx Zync XC7Z010 filtrul Canny consumă 38% din celulele logice disponibile. Proiectele realizate în HLS au nevoie de circuite FPGA cu capacitate mai mare. HLS simplifică implementarea algoritmilor matematici, dar necesită mai multe resurse.

Capitolul al patrulea analizează diferite metode de reconfigurare în tehnologiile Xilinx și Altera. Este o sinteză selectivă și critică a documentărilor în domeniul reconfigurării circuitelor FPGA. În urma studiilor s-a optat pentru metoda de reconfigurarea parțială, care folosește unitatea PCAP (tehnologie Xilinx) împreună cu un sistem de operare Linux, opțiune utilizată pentru dezvoltarea sistemului din capitolul următor. Această metodă este cea mai ieftină din punct de

vedere al resurselor FPGA, iar folosirea sistemului de operare oferă un mediu software mai flexibil.

Capitolul al cincilea are ca scop prezentrea în detaliu a modului de implementare a tehnicii de reconfigurare pe circuite FPGA XILINX Zynq, pe baza aplicației de control al zborului unui quadcopter. Se utilizează arhitectura disponibilă, în sensul construirii proiectului în jurul procesorului ARM și a părții configurate pentru filtrul Canny (procesarea de indentificare a muchiilor imaginilor mobile), cu părți de reconfigurare. Astfel se prezintă utilizarea sistemului de operare Linux pentru sarcinile de control, blocurile pentru filtrarea semnalelor de la senzorii de accelerație și giroscopici (filtru complemenetar), algoritmul de control PID. În final se prezintă un sistem cadru care are capacitatea pentru implementarea diferitelor sarcini de control bazat pe tehnica de reconfigurare în circuitele FPGA.

6.1. Contribuții originale

- Implementarea și testarea în circuitele FPGA a filtrului complementar [56] pentru calcularea unghiul de înclinare a unei vehicul aerian.
- Implementarea și testarea algoritmului de control PID pentru stabilizarea vehicul aerian quadcopter [21].
- Dezvoltarea și adaptarea sistemului de operare încorporat Arch Linux la circuitele FPGA Zynq [50]
- Implementarea în FPGA a unui procesor dedicat EMMC pentru algoritmi de control [10].
- Compararea performanțelor a două procesoare dezvoltate pe FPGA (EMMC, SLP) cu procesorul de referință Xilinx Pico Blaze.
- Dezvoltarea mediului de testare pentru algoritmi neuro-fuzzy aplicate la quadcopter [25] [74].
- Implementarea în limbajul VHDL a circuitelor de control și monitorizare pentru un vehicul aerian quadcopter și testarea sistemului proiectat [42].
- Implementarea mediului de interconectare între procesorul ARM și partea reconfigurabilă a circuitului FPGA, dezvoltarea circuitelor de testare și dezvoltarea sistemului de monitorizare și de logare [38].

6.3 Diseminarea rezultatelor

1. L. Bako, **Sz. Hajdu** și F. Morgan, „Evaluation and Comparison of Low FPGA Footprint,Embedded Soft-Core Processors,” în *MACRo 2017 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics*, Volume 2, pp. 23-30, 2017.

2. **Sz. Hajdu**, S.T. Brassai, I. Szekely, „FPGA based angular stabilization of a quadcopter,” în *MACRo 2017 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics*, Volume 2, pp. 79-86, 2017.

3. T. Tamas, **Sz. Hajdu** și S. T. Brassai, „Adaptive Neuro-Fuzzy Structure Based Control Architecture," în, „Procedia Technology",Volume 22, pp. 600-605, 2015, *WOS:000383949300083*.

4. L. Bako, **Sz. Hajdu** și A. Bacs, „Displacement detection method in video feeds using a distributed architecture on SoC platform for real-time control applications," în *IEEE International Conference and Workshop in Óbuda on Electrical and Power Engineering*, DOI: 000255-000260. 10.1109/CANDO-EPE.2018.8601161, 2018.

5. L. Bako, **Sz. Hajdu**, T. S. Brassai, F. Morgan și C. Enachescu, „Embedded Implementation of a Real-Time Motion Estimation Method in Video Sequences" în, „Procedia Technology", Volume 22, pp. 897-904, 2015, *WOS:000383949300126*.

6. **Sz. Hajdu** și S.T. Brassai, „Implementation Of Embedded Linux Systems On Fpga Based Circuits For Real Time Process Control," în *MACRo 2015 5th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics*, Volume 1, pp. 145-154, 2015, *WOS:000364568800014*.

7. **Sz. Hajdu**, S.T. Brassai, I. Szekely, „Complementary filter based sensor fusion on FPGA platforms," în *2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP)*, pp. 851-856, 2017, *WOS:000426909600131*.

8. S.T. Brassai, **Sz. Hajdu**, T. Tibor, L. Bako, „Hardware Implemented Adaptive Neuro Fuzzy System,"*Proceedings of the 2015 16th International Carpathian Control Conference (ICCC)* „ pp.58-63, 2015.

6.2 Direcții viitoare de cercetare

Mediul dezvoltat necesită mai multe teste, cu reconfigurarea parțială nu s-au rezolvat toate problemele de resurse, circuitele mai mari necesită mai mult spațiu. Proiectul trebuie să fie mutat de la circuitul Xilinx Zync XC7Z010 la un circuit cu capacitate mai mare ca XC7Z020.

Implementarea algoritmilor bazați pe rețele neurale făcând quadcopterul mai puțin dependent de utilizator. Scăderea timpului de reconfigurare care este acum ~20 ms.

Bibliografie

[1] C. Unsalan și B. Tar, Digital System Design with FPGA: Implementation Using Verilog and VHDL, McGraw-Hill Education, 2017.

[2] H. Bhatnagar, ADVANCED ASIC CHIP SYNTHESIS, Kluwer Academic Publishers, 2002.

[3] S. H. A. Dehon, Reconfigurable Computing, Elsevier, 2008.

[4] S. Kilts, Advanced FPGA Design: Architecture, Implementation, and Optimization, Wiley-IEEE Press, 2007.

[5] Xilinx, „7 Series FPGAs Configurable Logic Block," 2016. [Interactiv]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.

[6] M. J. S. Smith, Application Specific Integrated Circuits, Addison-Wesley Professional, 1997.

[7] F. Gruian, „Xilinx Vivado/SDK Tutorial," 2017. [Interactiv]. Available: http://fileadmin.cs.lth.se/cs/Education/EDAN15/labs/lab1/vivado_tutorial.pdf.

[8] S. Sutherland, Verilog-2001, Springer, 2002.

[9] J. Bhasker, A VHDL Primer, Prentice Hall, 1999.

[10] L. Bako, **Sz. Hajdu** și F. Morgan, „Evaluation and Comparison of Low FPGA Footprint,Embedded Soft-Core Processors," *MACRo 2017 6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics,* pp. 23-30, 2017.

[11] M. Holland, J. Harris și S. Hauck, „Harnessing FPGAs for computer architecture education," *Proceedings 2003 IEEE International Conference on Microelectronic Systems Education. MSE'03,* pp. 12-13, 2003.

[12] E. R. P. Julio, J. G. Edwar și M. S. Fernando, „Implementation of an 8-Bit Softcore Microcontroller on Xilinx Spartan FPGA," *Indian Journal of Science and Technology,* pp. 1-7, 2017.

[13] Xilinx, „Spartan-3E FPGA Family Data Sheet," 2018. [Interactiv]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf.

[14] C. R. K. M. Morris Mano, "Logic and Computer Design Fundamental" 2nd Edition, ISBN 0130314862, 2002.

[15] F. Morgan, „viciLogic: Online learning and prototyping platform for digital logic and computer architecture," *eChallenges e-2014 Conference Proceedings,* pp. 1-9, 2014.

[16] Xilinx, „PicoBlaze 8-bit Embedded Microcontroller User Guide for Extended Spartan®-3 and Virtex®-5 FPGAs Introducing PicoBlaze for Spartan-6, Virtex-6, and 7 Series FPGAs," 2011.

[17] Xilinx, „Vivado Design Suite User Model-Based DSP Design Using System Generator," 2018. [Interactiv]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug897-vivado-sysgen-user.pdf.

[18] K. Kintali și Y. Gu, „Model-Based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP," 2015. [Interactiv]. Available: https://www.mathworks.com/tagteam/86457_92077_v01_Xilinx_WhitePaper.pdf.

[19] K. J. Åström, Pid Controllers, International Society for Measurement and Control, 1995.

[20] Bosh Sensortec, „Bosh," June 2016. [Interactiv]. Available: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf.

[21] **Sz. Hajdu**, S.T. Brassai, I. Szekely, „FPGA based angular stabilization of a quadcopter," *6th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics,* pp. 79-86, 2017.

[22] D. Toochinda, „Discrete-time PID Controller Implementation,” 2015. [Interactiv]. Available: https://www.scilab.org/discrete-time-pid-controller-implementation.

[23] A. Michael și H. Mohammad, PID Control New Identification and Design Methods, London: Springer, 2005.

[24] Digilent, „Zybo Reference Manual,” 2017. [Interactiv]. Available: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual.

[25] T. Tamas, **Sz. Hajdu** și S. T. Brassai, „Adaptive Neuro-Fuzzy Structure Based Control Architecture,” *9th International Conference Interdisciplinarity in Engineering,* pp. 600-605, 2015.

[26] P. Coussy și A. Morawiec, High-Level Synthesis from Algorithm to Digital Circuit, Springer, 2008.

[27] Z. Zhao și J. C. Hoe, „Using Vivado-HLS for Structural Design: a NoC Case Study,” *2017 ACM/SIGDA International Symposium,* pp. 289-289, 2017.

[28] L. Bako, **Sz. Hajdu** și A. Bacs, „Displacement detection method in video feeds using a distributed architecture on SoC platform for real-time control applications,” *IEEE International Conference and Workshop in Óbuda on Electrical and Power Engineering,* pp. 255-260, 2018.

[29] B. Horn și B. Schunck, „Determining optical flow,” *Artificial,* vol. 17, pp. 185-204, 1981.

[30] F. Ruffier și N. Franceschini, „Optic flow regulation: the key to aircraft automatic guidance,” *Robotics and Autonoumous Systems,* pp. 177-194, 2005.

[31] R. Chaudhry, A. Ravichandran, G. Hager și R. Vidal, „Histograms of Oriented Optical Flow and Binet-Cauchy Kernels on Nonlinear Dynamical Systems for the Recognition of Human Actions,” *2009 IEEE Conference on Computer Vision and Pattern Recognition,* pp. 1932 - 1939, 2009.

[32] J. Chase, B. Nelson, J. Bodily, Z. Wei și D.-J. Lee, „Real-Time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study," *2008 16th International Symposium on Field-Programmable Custom Computing Machines,* pp. 173 - 182, 2008.

[33] B. Lucas și T. Kanade, „An Iterative Image Registration Technique with an Application to Stereo Vision," *7th International Joint Conference on Artificial Intelligence (IJCAI),* pp. 674-679, 1981.

[34] C. Deng, W. Ma și Y. Yin, „An edge detection approach of image fusion based on improved Sobel operator," *2011 4th International Congress on Image and Signal Processing,* pp. 1189 - 1193, 2011.

[35] L. Yang, X. Wu, D. Zhao, H. Li și J. Zhai, „An improved Prewitt algorithm for edge detection based on noised image," *4th International Congress on Image and Signal Processing,* pp. 1197 - 1200, 2011.

[36] H. Qing-hui și L. Xiao-gang, „The research of an improved Roberts algorithm used in welding line identification," *IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design,* pp. 786 - 788, 2009.

[37] H. Vala și A. Baxi, „A review on Otsu image segmentation algorithm," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* pp. 387-389, 2013.

[38] L. Bako, **Sz. Hajdu**, T. S. Brassai, F. Morgan și C. Enachescu, „Embedded Implementation of a Real-Time Motion Estimation Method in Video Sequences," *9th International Conference Interdisciplinarity in Engineering,* pp. 897-904, 2015.

[39] W. Lie și W. Feng-yan, „Dynamic Partial Reconfiguration in FPGAs," *2009 Third International Symposium on Intelligent Information Technology Application,* pp. 445-448, 2009.

[40] D. Koch, Partial Reconfiguration on FPGAs, Springer, 2013.

[41] Altera, „Quartus II Handbook Volume 1: Design and Synthesis," 2014. [Interactiv]. Available: https://courses.cs.washington.edu/courses/cse467/15wi/docs/Quartus_II_Handbook.pdf.

[42] Intel, „Partial Reconfiguration User Guide," [Interactiv]. Available: https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug-qpp-pr.pdf.

[43] Xilinx, „Vivado Design Suite User Guide Partial Reconfiguration," [Interactiv]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug909-vivado-partial-reconfiguration.pdf.

[44] Xilinx, „AXI DMA v7.1," 2018. [Interactiv]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf.

[45] M. Liu, W. Kuehn, Z. Lu și A. Jantsch, „Run-time Partial Reconfiguration speed investigation and architectural design space exploration," *2009 International Conference on Field Programmable Logic and Applications,* pp. 498 - 502, 2009.

[46] C. Claus, F. H. Muller, J. Zeppenfeld și W. Stechele, „A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *2007 IEEE International Parallel and Distributed Processing Symposium,* pp. 1-7, 2007.

[47] A. Ebrahim, K. Benkrid, X. Iturbe și C. Hong, „A novel high-performance fault-tolerant ICAP controller," *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS),* pp. 259 - 263, 2012.

[48] A. Stoddard, A. Gruwell, P. Zabriskie și M. Wirthlin, „High-speed PCAP configuration scrubbing on Zynq-7000 All Programmable SoCs," *2016 26th International Conference on Field Programmable Logic and Applications (FPL),* pp. 1-8, 2016.

[49] Digilent inc., April 2016. [Interactiv]. Available: https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf.

[50] **Sz. Hajdu** și S.T. Brassai, „Implementation Of Embedded Linux Systems On Fpga Based Circuits For Real Time Process Control," *5th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics,* p. 145–154, 2015.

[51] B. Huang, J. Harrison, O. Goudard, G. Naylor, R. Myers, R. Sharples și R. Vann, „Embedded Linux on FPGA Instruments for Control Interface and Remote Management," *European Synchrotron Radiation Facility ESRF,* pp. 1293-1295, 2012.

[52] R. Williams, Real-Time Systems Development, Butterworth Heinemann, 2005.

[53] D. W. Sven-Thorsten Dietrich, The Evolution of Real-Time Linux, OSADL Project: Real Time Linux Workshops, 2005.

[54] J. Corbet, A. Rubini și G. Kroah-Hartman, Linux Device Drivers, 3rd Edition, O'Reilly Media, 2005.

[55] P. Dusty, Arch Linux Handbook 3.0: A Simple, Lightweight Survival Guide, CreateSpace Independent Publishing Platform, 2012.

[56] **Sz. Hajdu**, S.T. Brassai, I. Szekely, „Complementary filter based sensor fusion on FPGA platforms," *2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP),* pp. 851 - 856, 2017.

[57] S. Ailneni, S. K. Kashyap și N. S. Kumar, „Real Time Sensor Fusion for Micro Aerial Vehicles using Low cost Systems," *2016 Indian Control Conference (ICC),* pp. 292-297, 2016.

[58] H. Ahmed și M. Tahir, „Accurate Attitude Estimation of a Moving Land Vehicle Using Low-Cost MEMS IMU Sensors," *IEEE Transactions on Intelligent Transportation Systems,* vol. 19, nr. 99, pp. 1-17, December 2016.

[59] E. Trinklein și G. Parker, „Ship Motion Sensor Isolation System Development and Testing for use with Low Cost IMUs," *Sensors Applications Symposium (SAS),* pp. 1-6, 2016.

[60] H. Ferdinando, H. Khoswanto și D. Purwanto, „Embedded Kalman Filter for Inertial Measurement Unit (IMU) on the ATMega8535," *2012 International Symposium on Innovations in Intelligent Systems and Applications,* pp. 1-5, 2012.

[61] D. Barbosa, A. Lopes și R. E. Araújo, „Sensor Fusion Algorithm Based on Extended Kalman Filter for Estimation of Ground Vehicle Dynamics," *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society,* pp. 1049-1054, 2016.

[62] S. Sabatelli, M. Galgani, L. Fanucci și A. Rocchi, „A Double-Stage Kalman Filter for Orientation Tracking With an Integrated Processor in 9-D IMU," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT,* vol. 62, nr. 3, pp. 590-598, March 2013.

[63] A. E. Hadri, L. Benziane, A. Seba și A. Benallegue, „Sensors model based data fusion using complementary filters for attitude estimation and stabilization," *2016 IEEE International Conference on Robotics and Automation (ICRA),* pp. 2978-2983, 2016.

[64] K. Madhira, A. Gandhi și A. Gujral, „Self balancing Robot using Complementary filter Implementation and analysis of Complementary filter on SBR," *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT),* pp. 2950-2954, 2016.

[65] P. Marantos, Y. Koveos și K. J. Kyriakopoulos, „UAV State Estimation Using Adaptive Complementary Filters," *IEEE Transactions on Control Systems Technology,* vol. 24, nr. 4, pp. 1214 - 1226, July 2016.

[66] G. G. Redhyka, D. Setiawan și D. Soetraprawata, „Embedded Sensor Fusion and Moving-average Filter for Inertial Measurement Unit (IMU) on the Microcontroller-based Stabilized Platform," *2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT),* pp. 72-77, 2015.

[67] P. Gui, L. Tang și S. Mukhopadhyay, „MEMS Based IMU for Tilting Measurement: Comparison of Complementary and Kalman Filter Based Data Fusion," *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA),* pp. 2004-2009, 2015.

[68] S. Colton, „The balance filter," *Presentation, Massachusetts Institute of Technology,* June 2007.

[69] J. Volder, „The CORDIC Computing Technique," *Proceedings of the Western Joint Computer Conference (WJCC),* pp. 257-261, 1959.

[70] STMicroelectronics N.V., September 2012. [Interactiv]. Available: http://www.st.com/content/ccc/resource/technical/document/datasheet/bd/61/af/53/b5/f5/4d/7b/DM00037200.pdf/files/DM00037200.pdf.

[71] InvenSense Inc., August 2013. [Interactiv]. Available: https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/.

[72] STMicroelectronics N.V., November 2013. [Interactiv]. Available: http://www.st.com/content/ccc/resource/technical/document/datasheet/1c/9e/71/05/4e/b7/4d/d1/DM00057547.pdf/files/DM00057547.pdf.

[73] Pololu, „MinIMU-9 v5 Gyro, Accelerometer, and Compass (LSM6DS33 and LIS3MDL Carrier)," 2018. [Interactiv]. Available: https://www.pololu.com/product/2738.

[74] S.T. Brassai, **Sz. Hajdu**, T. Tibor, L. Bako, „Hardware Implemented Adaptive Neuro Fuzzy System," *Proceedings of the 2015 16th International Carpathian Control Conference (ICCC),* pp. 58 - 63, 2015.

Abstract

Due to the rapid development of FPGA circuits, implementation of complex and fast control algorithms became faster and easier. The above work presents different design methods for FPGA circuit development including HDL languages (VHDL, Verilog) and Matlab Simulink through Xilinx System Generator and high level languages as C and C++ through Xilinx Vivado HLS. The final goal was mixing the advantages of these different design methods to design a system capable for controlling, monitoring, and deploying different control algorithms. This is achieved by using the Xilinx Zynq FPGA that incorporates an ARM hardcore processor that runs a real-time embedded Linux operating system. The operating system hides the underlining architecture of the design from the user and gives a universal control interface. For a better resource utilization and a more general build partial reconfiguration was introduced. Partial reconfiguration is the capability for reconfiguring part of the FPGA circuit without disturbing the data flow in the rest of the circuit. The operating system controls this process additional circuits can be downloaded to the board.

The first part of the thesis presents different methods for circuit implementation and gives example designs, the advantages and disadvantages of each method studied. Each chapter presents a design or circuit that will be used to construct the final project. This project is the control and angular stabilization of a quadcopter. The last chapter presents the whole design completed with partial reconfiguration capabilities.

Rezumat

Datorită dezvoltării rapide a circuitelor FPGA, implementarea algoritmilor de control complecși a devenit mai rapidă și mai ușoară. În lucrare se prezintă diferite metode de proiectare pentru dezvoltarea circuitelor FPGA, inclusiv limbajele HDL (VHDL, Verilog) și Matlab Simulink prin Xilinx System Generator și limbaje de nivel înalt ca C și C ++ prin Xilinx Vivado HLS. Scopul final a fost mixarea avantajelor acestor metode de proiectare diferite pentru a proiecta un sistem capabil să controleze, să monitorizeze și să implementeze algoritmi de control diferiți. Acest lucru este realizat prin utilizarea Xilinx Zynq FPGA, care încorporează un procesor ARC hardcore, pe care rulează un sistem de operare Linux în timp real încorporat. Sistemul de operare ascunde practic arhitectura designului față de utilizator și oferă o interfață de control universală. Pentru o utilizare mai bună a resurselor fost introdusă reconfigurarea parțială. Reconfigurarea parțială este capacitatea de a reconfigura o parte a circuitului FPGA fără a deranja fluxul de date în restul circuitului. Sistemul de operare controlează acest proces, configurațiile de circuite suplimentare pot fi descărcate pe placă. Prima parte a lucrării prezintă diferite metode de implementare a circuitelor și oferă exemple de modele, sunt studiate avantajele și dezavantajele fiecărei metode. Fiecare capitol prezintă un design sau un circuit care va fi utilizat pentru a construi proiectul final. Acest proiect este controlul și stabilizarea unghiulară a unui quadcopter. Ultimul capitol prezintă întregul proiect finalizat cu capacități parțiale de reconfigurare.

Curriculum Vitae

Date personale:

Nume: Hajdu

Prenume: Szabolcs

Data și locul nașterii:

Adresa:

E-mail: hajdu.szabolcs@unitbv.ro

Studii:

2015 – prezent: Universitatea Transilvania din Brașov

Doctorand, Facultatea de Inginerie Electrică și Știința Calculatoarelor

2013-2015: Universitatea Sapientia din Cluj - Napoca

Facultatea de Științe Tehnice și Umaniste din Tg. Mureș

Masterat: Sisteme de control inteligente

2007-2011: Universitatea Sapientia din Cluj - Napoca

Facultatea de Științe Tehnice și Umaniste din Tg. Mureș

Licență: Automatică și informatică aplicată

Experiență profesională:

2012 - 2017 : tehnician de laborator la Universitatea Sapientia din Cluj - Napoca ,

Facultatea de Științe Tehnice și Umaniste din Tg. Mureș

2017 - prezent: Inginer de rețea de calculatoare la Universitatea Sapientia din Cluj – Napoca, Facultatea de Științe Tehnice și Umaniste din Tg. Mureș

Curriculum Vitae

Personal information:

Surname: Hajdu

Name: Szabolcs

Data and place of birth:

Address:

E-mail: hajdu.szabolcs@unitbv.ro

Education:

2015-present: Transilvania University of Brașov

Ph.D. studies, Faculty of Electrical Engineering and Computer Science

2013-2015 - Sapientia University of Cluj-Napoca

Faculty of Technical and Human Sciences, Tg. Mureș

MSc: Intelligent control systems

2007-2011 Sapientia University of Cluj-Napoca

Faculty of Technical and Human Sciences, Tg. Mureș

BSc: Automation and applied informatics

Professional experience:

2012 - 2017 : lab technician - Sapientia University of Cluj-Napoca Technical and Human Sciences, Tg. Mureș.

2017 - present: system administrator - Sapientia University of Cluj-Napoca Technical and Human Sciences, Tg. Mureș.