Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Implementation of Quadruped Robot's Motion Control on SoC FPGA

Διπλωματική Εργασία

του

Καρακάση Χρυσόστομου

Επιβλέπων: Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων
Αθήνα, Μάρτιος 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Implementation of Quadruped Robot's Motion Control on SoC FPGA

Διπλωματική Εργασία

του

Καρακάση Χρυσόστομου

Επιβλέπων: Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Μαρτίου 2019.

.....	.....	.....
Δημήτριος Σούντρης	Ευάγγελος Παπαδόπουλος	Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων
Αθήνα, Μάρτιος 2019

.....
Καρακάσης Χρυσόστομος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Παρόλο που τα πεδία της Ρομποτικής και των FPGA μπορεί να φαίνονται ασυσχέτιστα εκ πρώτης όψεως, η συνεργασία τους έχει τη δυνατότητα να οδηγήσει σε αξιοσημείωτα κατορθώματα, λαμβάνοντας υπόψιν τις ανάγκες του πρώτου πεδίου και τα προτερήματα του δεύτερου. Από την μια πλευρά, τα ρομπότ έχουν εξελιχθεί ραγδαία τα τελευταία χρόνια, ωθώντας τα όρια τους ακόμα παραπέρα και προσφέροντας νέες ευκαιρίες. Ωστόσο, καθώς οι στόχοι τους επεκτείνονται, το ίδιο συμβαίνει και για τις ανάγκες τους. Σήμερα, ο κινηματικός έλεγχος είναι ιδιαίτερα απαιτητικός σε πόρους και σε δεδομένα, ενώ ο αριθμός των ενσωματωμένων αισθητήρων σε ρομποτικές συσκευές συνεχώς αυξάνεται. Από την άλλη πλευρά, τα FPGA συνεχώς βελτιώνονται από την ημέρα της δημιουργίας τους, αποκτώντας μια εξέχουσα θέση στο πεδίο των Μέσων Ψηφιακής Σχεδίασης Κυκλωμάτων. Το χαμηλό τους κόστος σε συνδυασμό με την δυνατότητα τους για παράλληλη επεξεργασία, χωρίς να περιορίζονται σε μια αμετάβλητη διαμόρφωση, έχουν οδηγήσει στην αξιοποίησή τους σε πληθώρα εφαρμογών. Ταυτόχρονα, το μεγάλο πλήθος περιφερειακών που διαθέτουν, τα καθιστούν ιδανικά για την διαχείριση πολλών εξωτερικών σημάτων. Επομένως, μια πιθανή συνεργασία μεταξύ των δύο κλάδων φαντάζει ιδιαίτερα λογική, γεγονός που μας οδήγησε στην απόφαση να εξερευνήσουμε αυτή τη δυνατότητα, στα πλαίσια αυτής της διπλωματικής εργασίας.

Συνοπτικά, πραγματοποιήθηκε η ανάπτυξη ενός κεντρικού συστήματος ελέγχου για το τετράποδο ρομπότ Λαίλαψ ΙΙ, αξιοποιώντας τη SoC αναπτυξιακή πλατφόρμα Zybo, που συγκροτείται από έναν επεξεργαστή ARM-Cortex A9 και ένα FPGA της οικογένειας Zynq-7000 από την Xilinx. Συγκεκριμένα, έπειτα από μια σύντομη εισαγωγή στα Τετράποδα ρομπότ και τα FPGA, στο τρίτο κεφάλαιο παρουσιάζονται η FPGA πλατφόρμα, καθώς και το τετράποδο ρομπότ που χρησιμοποιήθηκαν. Στη συνέχεια, αναλύεται το υπεύθυνο υλισμικό για την διαχείριση των κινητήρων και των κωδικοποιητών, ενώ παρατίθενται τα αντίστοιχα διαγράμματα λειτουργίας, καθώς επίσης και πειράματα που επιβεβαιώνουν την ορθότητα τους.

Κατόπιν, αναλύεται το λογισμικό κομμάτι του συστήματος, το οποίο είναι υπεύθυνο για την υλοποίηση του κινηματικού ελέγχου, καθώς επίσης και το AXI4-Lite πρωτόκολλο, με το οποίο επιτυγχάνεται η επικοινωνία του επεξεργαστή και του FPGA. Εν συνεχεία, στο κεφάλαιο 6 απεικονίζεται η ενοποίηση του συστήματος, μαζί με ορισμένες ηλεκτρονικές πλακέτες που κατασκευάστηκαν εξ ολοκλήρου για λόγους συνδεσιμότητας. Τέλος, στο κεφάλαιο 7 περιλαμβάνεται η πειραματική αξιολόγηση του προτεινόμενου συστήματος, όπου επιτυγχάνεται ο κινηματικός έλεγχος του ρομπότ.

Συμπεραίνοντας, το προτεινόμενο σύστημα καταφέρνει να διαχειριστεί όλα τα απαιτούμενα περιφερειακά του ρομπότ, προσφέροντας παράλληλα πλήρη λειτουργικότητα, σχετικά σύντομο χρόνο υλοποίησης, αλλά και προοπτική για επέκταση, σε ιδιαίτερα προσιτή τιμή. Συγκεκριμένα, στο τελευταίο κεφάλαιο, προτείνονται μερικές προτάσεις σχετικά με μελλοντικές αλλαγές και βελτιώσεις που θα μπορούσαν να υλοποιηθούν εύκολα. Συνολικά, η δομή των κεφαλαίων σχεδιάστηκε με τέτοιο τρόπο, ώστε ακόμα και κάποιος με περιορισμένο υπόβαθρο στον προγραμματισμό υλισμικού ή στη Ρομποτική, να μπορεί εύκολα να κατανοήσει το περιεχόμενό τους.

Λέξεις Κλειδιά: FPGA, Τετράποδα Ρομπότ, Ενσωματωμένα Συστήματα, Υλοποίηση Hardware, Συ-σχεδίαση Λογισμικού και Hardware, SoC FPGA, Ρομποτική, Κινηματικός Έλεγχος

Abstract

Although the fields of Robotics and FPGAs may seem quite dissimilar at first glance, their collaboration has the potential to give birth to remarkable achievements, considering the former's necessities and the latter's assets. On the one hand, robots have evolved drastically over the years, pushing their boundaries even further, while presenting new opportunities. However, as their goals expand, so do their requirements. Nowadays, motion control is highly demanding, both in resources and data, while the number of sensors employed in robotic devices has been considerably increased. On the other hand, FPGAs have been constantly improving since their discovery, gaining a renowned place in the field of digital circuit implementation media. Their low cost in conjunction with their ability for parallel processing, without being limited to an immutable configuration, have led to their utilization in an abundance of applications. Simultaneously, the vast number of peripherals they are equipped with, renders them as ideal for the management of multiple external signals. Consequently, a possible cooperation between the two fields seemed only logical to us, and hence we decided to explore that possibility, in the context of this thesis.

In summary, the development of a centralized control scheme for the quadruped robot Laelaps II is introduced, using a System-on-Chip that comprises an ARM-Cortex A9 processor and an FPGA of the Zynq-7000 Xilinx family, as part of the Zybo Development Board. Specifically, after a brief introduction to both Quadruped Robots and FPGAs, the utilized SoC FPGA platform and quadruped robot are presented, in chapter 3. Following, the hardware responsible for the management of the robot's encoders and motors is being thoroughly analyzed, while being accompanied by the respective block diagrams of the designs and experiments that prove their validity.

Consequently, the software segment of the scheme is elaborated, responsible for the implementation of the motion control, along with the AXI4-Lite protocol, which handles the communication between the processor and the FPGA. Ultimately, the system's integration is illustrated in chapter 6, along with several custom made circuit boards that were manufactured for connectivity purposes. Lastly, an experimental evaluation of the proposed scheme is included in chapter 7, where the motion control of the robot is achieved.

In conclusion, the proposed scheme manages to handle all I/O peripherals required, whilst providing full functionality, comparatively short development time, and potential for expansion, at a highly affordable cost. Namely, in the final chapter, numerous suggestions are being proposed concerning future modifications and enhancements that could be easily implemented. In general, the formulation of the chapters was composed in such a way that even someone with a limited background in Hardware programming or Robotics, can easily comprehend their content and concept.

Keywords: FPGA, Quadruped Robots, Embedded Systems, Hardware Implementation, SoC FPGA, Robotics, Motion Control, Co-design.

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τους καθηγητές μου, Δημήτριο Σούντρη ΕΜΠ και Ευάγγελο Παπαδόπουλο ΕΜΠ, για την ευκαιρία που μου προσέφεραν, να συνδυάσω δύο ιδιαίτερα αγαπητούς μου κλάδους, την Ψηφιακή Σχεδίαση και τη Ρομποτική. Στα πλαίσια της συνεργασίας των εργαστηρίων Μικροϋπολογιστών και Ψηφιακών Συστημάτων και Αυτομάτου Ελέγχου-ΕΠ, μου δόθηκε η ευκαιρία να εφαρμόσω τις γνώσεις μου και κυρίως να εξερευνήσω τις δυνατότητές μου σε ένα εγχείρημα που ποτέ δεν θα φανταζόμουν.

Ιδιαίτερα, θα επιθυμούσα να εκφράσω την ευγνωμοσύνη μου στον Καθηγητή Δημήτριο Σούντρη, για την αμέριστη εμπιστοσύνη που επέδειξε στο πρόσωπο μου από την πρώτη στιγμή, ενώ στάθηκε δίπλα μου καθόλη την διάρκεια της διπλωματικής μου, τόσο σε θέματα της διπλωματικής όσο και στην προσπάθεια μου να επιτύχω τους στόχους μου. Αναμφίβολα, οφείλω ένα τεράστιο ευχαριστώ από καρδιάς στους υποψήφιους διδάκτορες Χαράλαμπο Μάραντο και Κωνσταντίνο Μαχαιρά, καθώς επίσης και στον διδάκτορα Ιωσήφ Παρασκευά, για την υποστήριξη, την καθοδήγηση, αλλά κυρίως για την υπομονή τους, όλους αυτούς τους μήνες και όλα αυτά τα βράδια. Επίσης, θα ήθελα να ευχαριστήσω τα μέλη των εργαστηρίων Mlab και CLS-EP, που με υποδέχθηκαν σε ένα ιδιαίτερο οικογενειακό περιβάλλον, αλλά και για την συμβολή τους σε αρκετές πτυχές της διπλωματικής μου. Συνολικά, αποτέλεσε μία αξέχαστη εμπειρία, από την οποία δεν κέρδισα μονάχα γνώση και όμορφες αναμνήσεις, αλλά σίγουρα και φίλους.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου, την οικογένειά μου, και ιδιαίτερα τον Δημήτρη και τον αδερφό μου Πάρι, για την συμπαράσταση τους καθόλη την διάρκεια της έως τώρα σταδιοδρομίας μου, και ιδιαίτερα τον τελευταίο χρόνο.

Καρακάσης Χρυσόστομος 2019

Contents

List of Figures **13**

List of Tables **14**

Abbreviations **15**

1 Εκτεταμένη Ελληνική Περίληψη **17**
1.1 Εισαγωγή . . . 17
1.1.1 Τετράποδα Ρομπότ . . . 17
1.1.2 FPGA . . . 18
1.1.3 FPGA σε Τετράποδα Ρομπότ . . . 18
1.1.4 Κίνητρο . . . 19
1.2 Περιγραφή Υλισμικού . . . 20
1.2.1 Προγραμματιζόμενες στο Πεδίο Διατάξεις Πύλης (FPGA) . . . 20
1.2.2 Αναπτυξιακή Πλακέτα Zynq™- 7000 (ZYBO) . . . 21
1.2.3 Περιγραφή του Ρομπότ Laelaps II . . . 22
1.2.4 Επισκόπηση της EtherCat αρχιτεκτονικής ελέγχου . . . 23
1.3 Περιγραφή Συστήματος . . . 24
1.4 Περιβάλλον Λογισμικού . . . 27
1.5 Ενοποίηση Συστήματος . . . 28
1.6 Αξιολόγηση Συστήματος . . . 29
1.7 Συμπεράσματα και Μελλοντική Εργασία . . . 31
1.7.1 Συμπεράσματα . . . 31
1.7.2 Μελλοντική Εργασία . . . 31

2 Introduction **32**
2.1 Literature Review . . . 32
2.1.1 Quadruped Robots . . . 32
2.1.2 FPGAs . . . 33
2.1.3 FPGAs in Quadruped Robots . . . 35
2.2 Motivation . . . 36
2.3 Thesis Outline . . . 37

3 Hardware Platform **38**
3.1 Field-Programmable Gate Arrays (FPGAs) . . . 38
3.2 SoC FPGAs . . . 39
3.3 Zynq™- 7000 Development Board (ZYBO) . . . 42
3.4 Laelaps II Robot Adumbration . . . 44
3.4.1 Overview of the EtherCat control architecture . . . 44
3.4.2 Electrical System . . . 45
3.4.3 Laelaps II Encoders . . . 45
3.4.4 Laelaps II gearheads and motors . . . 46
3.4.5 Leg design and motion planning . . . 47

4 System Description **49**
4.1 Management of the Encoders Output . . . 49
4.2 Quadrature Encoder Interface . . . 50
4.3 Velocity Estimation . . . 52
4.4 Combinatorial QEI Block . . . 56
4.5 Pulse Width Modulation . . . 57
4.6 Hardware/Software Codesign . . . 59

5 Software Environment **65**
5.1 High-Level Controller . . . 65
5.2 Inverse Kinematics & Trajectory Planning . . . 66
5.3 PD-Controller . . . 67

6 System Integration **73**
6.1 Connection of Laelaps II and ZYBO Board . . . 73
6.2 Finalized Architecture . . . 75

7 System Evaluation **77**
7.1 Laelaps II Locomotion Experiment . . . 77

8 Conclusions and Future Work **83**
8.1 Conclusions . . . 83
8.2 Future Work . . . 84

List of Figures

1.1 Τετράποδα Ρομπότ της Boston Dynamics. . . . 17
1.2 Το πρώτο εμπορικά διαθέσιμο FPGA το 1985 – το XC2064. . . . 18
1.4 Γενική αρχιτεκτονική των FPGA [4]. . . . 20
1.5 Αναπτυξιακή Πλακέτα ZYBO Zynq-7000 [6]. . . . 21
1.6 AXI4 Αρχιτεκτονικές καναλιών ανάγνωσης και εγγραφής [5]. . . . 22
1.7 Laelaps II [7]. . . . 23
1.8 Λειτουργικό Διάγραμμα του ελεγκτή για ένα πόδι [8]. . . . 23
1.9 Processing Flow of the SoC FPGA Implementation 24
1.10 Λειτουργικό διάγραμμα σχεδίασης υλισμικού για ένα πόδι του Laelaps II. . . . 24
1.11 Λειτουργικό διάγραμμα σχεδίασης υλισμικού για δύο πόδια του Laelaps II. . . . 25
1.12 Λειτουργικό διάγραμμα σχεδίασης υλισμικού για τέσσερα πόδια του Laelaps II. . . . 25
1.13 Λειτουργικό Διάγραμμα του Λογισμικού. . . . 27
1.14 Πύργος Κεντρικού Ελέγχου. . . . 28
1.15 Κεντρικός Πύργος Ελέγχου ενοποιημένος με το Λαίλαψ II. . . . 28
1.16 Καταγεγραμμένη Απόκριση των άκρων των ποδιών μαζί με τις επιθυμητές τροχιές. . . . 30

2.1 Boston Dynamics Legged Robots. . . . 32
2.2 The first commercially viable field-programmable gate array in 1985 – the XC2064. . . . 33
2.3 Xilinx's Zynq-7000 All Programmable SoC. . . . 34
2.4 Zynq UltraScale+ MPSoC. . . . 34
2.5 Prototype of the improved 3D-printed amphibious spherical robot [1]. . . . 35
2.6 Quadruped robot setup with the FPGA board on the left [2]. . . . 35
2.7 Hexabot Robot [3]. . . . 36

3.1 General architecture of FPGAs [4]. . . . 38
3.2 Comparison between standalone CPU and FPGA solution and SoC FPGA solution. . . . 40
3.3 AXI4 read and write channel architectures [5]. . . . 41
3.4 Detailed architecture of the Zynq-7000 devices [5]. . . . 41
3.5 ZYBO Zynq-7000 development board [6]. . . . 42
3.6 Pmod diagram [6]. . . . 43
3.7 Zybo Z7-20 peripherals and components [6]. . . . 43
3.8 Laelaps II [7]. . . . 44
3.9 Block diagram of the controller for a single leg [8]. . . . 45
3.10 Electrical System of Laelaps II [7]. . . . 46
3.11 HEDS-5640 Encoder of Avago [9]. . . . 46
3.12 Leg model [7]. . . . 47

4.1 Processing Flow of the SoC FPGA Implementation 49
4.2 Preliminary experimental setup. . . . 50
4.3 Filter block realized in the FPGA. . . . 50
4.4 Encoder waveform and state transition table [10]. . . . 51
4.5 *Encoder* Block. . . . 51
4.6 *Velocity Estimation* Block. . . . 55
4.7 FPGA-ARM comparison of velocity estimation. . . . 56
4.8 *Qei* Block in VHDL. . . . 56
4.9 PWM Block in VHDL. . . . 57
4.10 Simple behavioral simulation of the component. . . . 58
4.11 Duty Cycles 58
4.12 Custom AXI4-IP for one leg. . . . 60

4.13 Pmod pinout [6]. 60
4.14 Address Map for the ARM Cortex-A9 processor. 61
4.15 Block design of the hardware system for one leg of Laelaps II. 62
4.16 Block design of the hardware system for two legs of Laelaps II. 62
4.17 Block design of the hardware system for four legs of Laelaps II. 63

5.1 Comparison between the Inverse Kinematics outputs in the ARM and Matlab implementations. 67
5.2 Experimental Platform with motor. 68
5.3 P-Control Experiments. 69
5.4 PD-Control Experiments. 71
5.5 Operating diagram of Software Partition. 72

6.1 8 Channels Bi-Directional Logic Level Converter. 73
6.2 Intermediate Circuit Board between Zybo and Secondary Boards. 74
6.3 Secondary Circuit Board for connection with encoders and actuators. 74
6.4 Centralized Control Tower of Hardware. 75
6.5 Centralized Control Tower unified with Laelaps II. 76

7.1 Recorded Response of all legs toe along with the desired elliptical trajectories. 79
7.2 Recorded Response of all hip and knee angles along with the desired positions. 80
7.3 Velocity Estimation of each leg's joint along with their desired values. 81
7.4 Duty Cycles of each leg's motor and their respective predefined saturations limits. 82

List of Tables

1.1 Έκθεση κατανάλωσης πόρων του Field Programmable Gate Array (FPGA) για τη σχεδίαση των τεσσάρων ποδιών. 26
1.2 Σύγκριση της κατανάλωσης πόρων (%) των σχεδιάσεων για ένα, δύο και τέσσερα πόδια. . . . 26
1.3 Παράμετροι του Πειράματος Κίνησης 29

4.1 Generic Variables of *laelaps_one_leg_qei_vel_pwm* Intellectual Property (IP) 59
4.2 Utilization report of the FPGA resources for the four legs design. 64
4.3 Comparison of Resources Utilization (%) between One, Two and Four Leg Designs. 64

7.1 Parameters of Locomotion Experiment 78

Abbreviations

DoF	Degrees of Freedom
FPGA	Field Programmable Gate Array
PROM	Programmable Read Only Memory
PAL	Programmable Array Logic
PLA	Programmable Logic Array
ASIC	Application Specific Integrated Circuit
SoC	System on Chip
MPSoC	Multi-Processor SoC
RFSoC	Radio Frequency SoC
SNN	Spiking Neural Network
CPG	Central Pattern Generator
PWM	Pulse Width Modulation
EtherCat	Ethernet for Control Automation Technology
CLBs	Configurable Logic Blocks
FF	Flip-Flops
LUT	Lookup Table
BRAM	Block Random-access Memory
LUTRAM	LUT Random-access Memory
DSP	Digital Signal Processing
I/O	Input/Output
HDL	Hardware Description Language
VHDL	VHSIC Hardware Description Language
HLS	High-Level Synthesis
RTL	Register Transfer Level
CPU	Central Processing Unit
GPPs	General Purpose Processors
IP	Intellectual Property
PSoC	Programmable SoC
PS	Processing System
PL	Programmable Logic

AXI Advanced Extensible Interface

AMBA Advanced Microcontroller Bus Architecture

AP SoC All Programmable System-on-Chip

Pmod Peripheral Module

VCC Voltage Common Collector

VDC Volts of Direct Current

MCU Micro Controller Units

IC Integrated Circuit

QEI Quadrature Encoder Interface

USB Universal Serial Bus

PC Personal Computer

SD Secure Digital

XSDK Xilinx Software Development Kit

IDE Integrated Development Environment

OS Operating System

P-Control Proportional Control

PD-Control Proportional–Derivative Controller

BUFG Global Clock Buffer

ISR Interrupt Service Routine

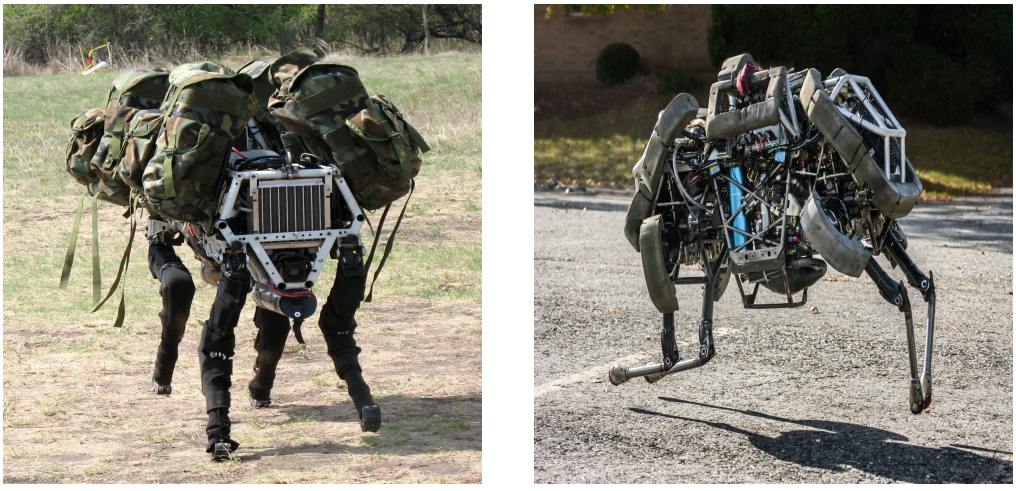
Chapter 1

Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

1.1.1 Τετράποδα Ρομπότ

Σε γενικές γραμμές, τα αυτόνομα ρομποτικά συστήματα μπορούν να ταξινομηθούν σε δύο κατηγορίες, στα συστήματα που προορίζονται για Χειρισμό και σε αυτά που σχετίζονται με την Κίνηση. Τα ρομπότ με πόδια αποτελούν έναν εκ των τριών θεμελιωδών διαμορφώσεων που υπάρχουν για τη κίνηση των Ευκίνητων ρομπότ στο έδαφος, μεταξύ των περιστροφικών συσκευών, όπως οι ερπύστριες, οι ρόδες, καθώς επίσης και περίτεχνες δομές παρόμοιες στο σώμα φιδιού. Συγκεκριμένα, τα ρομπότ με πόδια λόγω των ξεχωριστών χαρακτηριστικών τους, έχουν θεωρηθεί κατάλληλα για ποικίλες περιπτώσεις εφαρμογών [11]. Ως αποτέλεσμα, θεωρούνται ως ένας ιδιαίτερα προχωρημένος κλάδος της σύγχρονης μηχανικής που συνεχίζει να εξελίσσεται ραγδαία τα τελευταία χρόνια. Ερευνητικά εργαστήρια συνεχώς αναπτύσσουν πολύπλοκα και καινοτόμα σχέδια, τα οποία είτε ξεπερνούν προηγούμενες προσπάθειες, είτε παρουσιάζουν καινούργιες και ενδιαφέρουσες ιδιότητες. Μερικά από τα πιο διάσημα παρουσιάζονται στο Σχήμα 1.1 , τα οποία προέρχονται από την εταιρεία Boston Dynamics.



(a) BigDog (b) WildCat

Figure 1.1: Τετράποδα Ρομπότ της Boston Dynamics[†].

[†] https://www.bostondynamics.com/robots

Συνοπτικά, τις τελευταίες πέντε δεκαετίες, τα τετράποδα ρομπότ έχουν γίνει πιο γρήγορα, πιο αποδοτικά και πιο στιβαρά, ενώ δύνανται να αντιμετωπίσουν αξιόλογα μεγέθη φορτίων. Επίσης, η ανώτερη κινητικότητα που παρουσιάζουν σε διαφορετικούς τύπους εδάφους, με ποικίλους τρόπους βάδισης και ταχύτητες, τα ξεχωρίζει από τα υπόλοιπα Ευκίνητα ρομπότ και τα καθιστά στο κέντρο της ρομποτικής έρευνας. Επομένως, αυτές οι ιδιότητες, σε συνδυασμό με την εύπορη ανάπτυξη της ρομποτικής βιομηχανίας, υποδεικνύουν πως τα τετράποδα ρομπότ αποτελούν αναμφίβολα ένα πεδίο άξιο προς επένδυση [12, 13].

1.1.2 FPGA

Παρότι τα θεμέλια τους τέθηκαν κατά την διάρκεια της ευρείας ανάπτυξης των ολοκληρωμένων κυκλωμάτων από τις αρχές του 1960 μέχρι και τα μέσα του 1980, το πρώτο μοντέρνο FPGA (Προγραμματιζόμενη στο Πεδίο Διάταξη Πύλης) παρουσιάστηκε το 1984 από την Xilinx (Σχ.1.2).

Figure 1.2: Το πρώτο εμπορικά διαθέσιμο FPGA του 1985 – το XC2064 †.

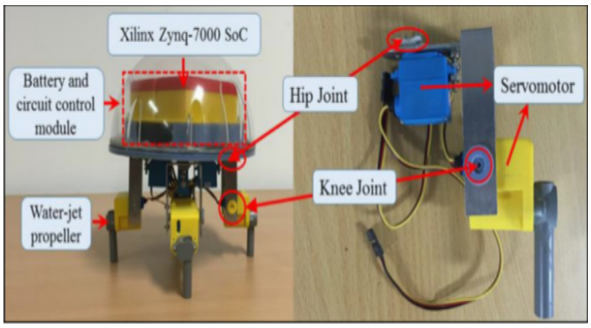
Βασισμένο στις υπάρχουσες αρχιτεκτονικές των Προγραμματιζόμενων Μνημών Μόνο Ανάγνωσης (PROM), των Προγραμματιζόμενων Λογικών Τύπου Πίνακα (PAL) και των Προγραμματιζόμενων Διατάξεων Λογικής (PLA), τα FPGA παρουσιάσαν σημαντικά πλεονεκτήματα έναντι της τεχνολογίας των Ολοκληρωμένων Ειδικών Εφαρμογών (ASIC) και των PAL, τα οποία άκμασαν στην αγορά των ηλεκτρονικών στα μέσα του 1980. Συγκεκριμένα, η δυνατότητα τους για άμεση διαμόρφωση σε συνδυασμό με το χαμηλό τους κόστος, επέτρεψε στα FPGA να ξεπεράσουν τα ASIC, τα οποία απαιτούσαν για τη κατασκευή τους μήνες και εκατοντάδες χιλιάδες δολάρια. Επιπροσθέτως, λόγω της κατάργησης του πίνακα AND, τα FPGA απέκτησαν ένα αρχιτεκτονικό πλεονέκτημα έναντι των PAL. Τόσο η απόδοση όσο και η χωρητικότητα αποδεσμεύτηκαν από τους περιορισμούς που έφερε η συστοιχία AND, ενώ οι σχεδιαστές FPGA απέκτησαν σημαντική αρχιτεκτονική ελευθερία [14, 15].

1.1.3 FPGA σε Τετράποδα Ρομπότ

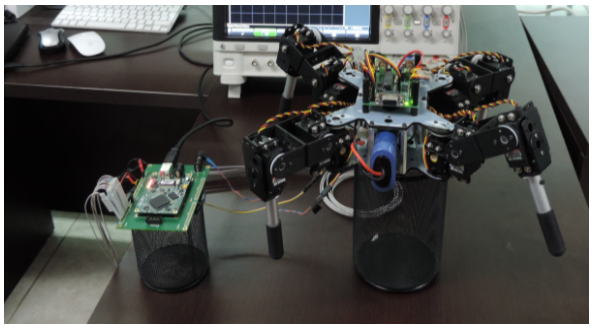
Σύμφωνα με όσα γνωρίζει ο συγγραφέας, έπειτα από μια διεξοδική έρευνα σε ερευνητικές δημοσιεύσεις, συμπεραίνεται πως η συνεργασία μεταξύ ρομπότ και FPGA είναι ιδιαίτερα σπάνια. Αξιοσημείωτα είναι τα παρακάτω παραδείγματα, όπου επιτεύχθηκε συνεργασία και σημειώθηκαν αξιέπαινα αποτελέσματα.

Στο [1] παρουσιάζεται ένα καινοτόμο και κινητό αμφίβιο σφαιρικό ρομπότ, εμπνευσμένο από τη χελώνα, το οποίο αξιοποιεί ένα Xilinx all-programmable Zynq-7000 SoC FPGA (Z-7000) προκειμένου να ελαχιστοποιήσει την κατανάλωση ενέργειας των ολοκληρωμένων κυκλωμάτων και να εισαγάγει μια αρθρωτή σχεδίαση (Σχ.1.3α).

Στο [2] παράγονται μοτίβα κίνησης για τρία διαφορετικά ρομπότ με πόδια (δίποδο, τετράποδο, εξάποδο), μέσω ενός νευρομορφικού συστήματος υλοποιημένο σε μια πλακέτα FPGA Spartan 6 (Σχημ.1.3β).



(a) Πρωτότυπο του βελτιωμένου 3D-εκτυπωμένου αμφίβιου σφαιρικού ρομπότ [1].



(b) Διάταξη τετράποδου ρομπότ με FPGA πλακέτα στα αριστερά [2].

1.1.4 Κίνητρο

Ο στόχος αυτής της διπλωματικής είναι να παρουσιάσει ένα κεντρικό σύστημα ελέγχου για το τετράποδο ρομπότ Λαίλαψ ΙΙ του Εργαστηρίου Αυτόματου Ελέγχου - ΕΠ (Σχ.3.8), αξιοποιώντας την Αναπτυξιακή SoC Πλακέτα Zybo που συγκροτείται από έναν επεξεργαστή ARM-Cortex A9 και ένα FPGA της οικογένειας Zynq-7000 από τη Xilinx. Συγκεκριμένα, προτείνεται μια αρχιτεκτονική συσχεδίασης Υλισμικού/Λογισμικού, όπου το FPGA θα αναλάβει τη διαχείριση των κινητήρων και των κωδικοποιητών του ρομπότ, ενώ ο κινηματικός έλεγχος θα εκτελείται στον επεξεργαστή ARM.

Έχοντας υπόψιν τις ανάγκες ενός τετράποδου ρομπότ, θεωρούμε πως η ενσωμάτωση ενός SoC FPGA θα παρουσιάσει τα παρακάτω χαρακτηριστικά. Πρώτα από όλα, η σχεδίασή μας στοχεύει στη μείωση τόσο των διαστάσεων όσο και του βάρους. Η τωρινή σχεδίαση χρησιμοποιεί ένα αποκεντρωμένο σύστημα, το οποίο αποτελείται από έναν EtherCat Αφέντη και τέσσερις πύργους EtherCat τύπου Slave, οι οποίοι επιβαρύνουν το σύστημα σε βάρος, κατανάλωση ενέργειας και καλωδιώσεις. Από την άλλη, ο στόχος είναι να χρησιμοποιηθεί μονάχα μία από τις Αναπτυξιακές Πλακέτες Zybo, οδηγώντας σε ένα πιο ελαφρύ και συμπαγές τελικό σύστημα.

Επιπροσθέτως, ακόμα ένα κίνητρο πίσω από την πρόταση να συνδυαστεί ένα τετράποδο ρομπότ με μια πλατφόρμα SoC FPGA, αποτελεί το ισχυριζόμενο χαμηλό τους κόστος και η μειωμένη κατανάλωση ενέργειας. Εφόσον και οι δύο αυτές παράμετροι είναι κρίσιμες για τη σχεδίαση οποιουδήποτε ρομπότ, θα ήταν συναρπαστικό, αν αυτά τα χαρακτηριστικά ίσχυαν και σε αυτή τη απόπειρα.

Επιπλέον, η συνύπαρξη ενός FPGA και ενός ARM επεξεργαστή στην ίδια ψηφίδα, μειώνει δραματικά τυχόν επιβαρύνσεις στην επικοινωνία, ενώ προσφέρει υψηλότερο εύρος ζώνης. Ταυτόχρονα, η πολύτιμη ιδιότητα των FPGA για παράλληλη επεξεργασία, πιθανώς θα μειώσει περαιτέρω το χρόνο επεξεργασίας , ενώ θα περιορίσει τυχόν καθυστερήσεις.

Τέλος, η πλακέτα Zybo παρουσιάζει τη δυνατότητα να υποστηρίξει αρχιτεκτονικές με περισσότερες αρθρώσεις, δεδομένου πως διαθέτει πολλαπλά περιφερειακά E/E. Επομένως, σε περίπτωση που η προτεινόμενη σχεδίαση δεν εξαντλήσει του πόρους του SoC FPGA, αυτή η ιδιότητα θα αποδειχθεί ιδιαίτερα ελκυστική για μελλοντικές εφαρμογές.

1.2 Περιγραφή Υλισμικού

1.2.1 Προγραμματιζόμενες στο Πεδίο Διατάξεις Πύλης (FPGA)

Οι Προγραμματιζόμενες στο Πεδίο Διατάξεις Πύλης (FPGA) είναι ψηφιακά ολοκληρωμένα κυκλώματα που μπορούν να επαναπρογραμματιστούν και μετά από την παραγωγή τους 1.4. Συγκεκριμένα, ορίζονται ως προκατασκευασμένες συσκευές πυριτίου, οι οποίες μπορούν να διαμορφωθούν ως οποιοδήποτε δυνατό ψηφιακό κύκλωμα, μέσω του αντίστοιχου ηλεκτρονικού προγραμματισμού.

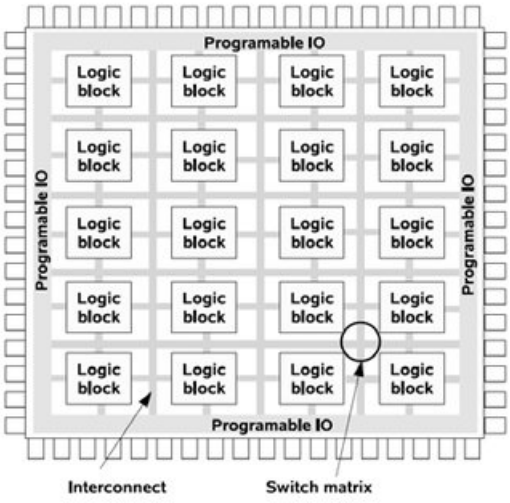


Figure 1.4: Γενική αρχιτεκτονική των FPGA [4].

Τα πολλαπλά περιφερειακά Ε/Ε, σε συνδυασμό με το μεγάλο πλήθος των διαθέσιμων διασυνδέσεων, επιτρέπουν στο FPGA να επεξεργαστεί πολλές εισόδους παράλληλα. Σε σύγκριση με την Κεντρική Μονάδα Επεξεργασίας CPU, μπορεί να προκύψει σημαντική επιτάχυνση στην επεξεργασία, δεδομένου ότι μια CPU δεν μπορεί να εκτελέσει ταυτόχρονα παραπάνω από μία εντολές. Την ίδια στιγμή, τα FPGA λειτουργούν σε αρκετά χαμηλότερες συχνότητες από τις CPU (μέχρι και τα 550MHz), οδηγώντας σε αρκετά χαμηλότερη κατανάλωση ενέργειας. Επιπλέον, αντί να είναι περιορισμένα σε μια αμετάβλητη εσωτερική λειτουργικότητα, όπως αυτή σχεδιάστηκε κατά την παραγωγή τους, τα FPGA μπορούν να επαναπρογραμματιστούν ανά πάσα στιγμή, σύμφωνα με τις ανάγκες του σχεδιαστή. Επομένως, πέρα από τη δυνατότητα της υλοποίησης οποιουδήποτε ψηφιακού κυκλώματος, οι σχεδιαστές μπορούν να τροποποιήσουν το υλισμικό τους είτε επειδή αποδείχθηκε λανθασμένο, είτε για να δημιουργήσουν κάτι διαφορετικό.

Σε σύνοψη, τα πλεονεκτήματα των FPGA είναι τα εξής:

- **Επαναδιαμόρφωση**
- **Παράλληλη Επεξεργασία**
- **Χαμηλή Κατανάλωση Ενέργειας**
- **Πολλαπλά Περιφερειακά**

Ωστόσο, όπως είναι φυσικό, τα FPGA παρουσιάζουν και ορισμένα μειονεκτήματα. Το δίκτυο διασύνδεσης που φέρει σε επικοινωνία όλα τα δομικά στοιχεία τους, καταλαμβάνει περίπου το 90% της επιφάνειας των FPGA, χαρακτηριστικό το οποίο βλάπτει σημαντικά την απόδοσή τους. Συγκεκριμένα, το βασικό τους πλεονέκτημα για ευελιξία, τα καθιστά αισθητά πιο αργά, πιο μεγάλα σε όγκο και περισσότερο ενεργοβόρα, σε σχέση με τα ισοδύναμα ASIC. Παρόλα αυτά, το χαμηλό τους κόστος σε συνδυασμό με τον υψηλό ρυθμό παραγωγής, καθιστά τα FPGA μια συναρπαστική επιλογή για την υλοποίηση ψηφιακών συστημάτων.

1.2.2 Αναπτυξιακή Πλακέτα Zynq™- 7000 (ZYBO)

Στα πλαίσια αυτής της διπλωματικής αξιοποιήθηκε η Αναπτυξιακή Πλακέτα Zybo(ZYnq BOard), κατασκευασμένη από την εταιρεία Digilent, η οποία χρησιμοποιεί το μικρότερο μέλος από την οικογένεια Zynq-7000 της Xilinx, το Z-7010 [6]. Το Z-7010 βασίζεται στην αρχιτεκτονική All Programmable System-on-Chip (AP SoC) της Xilinx, η οποία ενσωματώνει ένα διπύρηνο επεξεργαστή ARM Cortex-A9 με ένα FPGA από την 7η σειρά της Xilinx (Σχ.1.5). Επιπλέον, έξι θύρες Peripheral Module (Pmod) είναι διαθέσιμες για κάθε υλοποίηση.

Το AP SoC Zynq 7010 προσφέρει τα παρακάτω χαρακτηριστικά:

- Επαναπρογραμματίσιμη Λογική αντίστοιχη με αυτή των Artix-7 FPGA
    - Look-up Tables(LUTs): 17600
    - Flip Flops: 35200
    - Digital Signal Processing (DSP) κομμάτια: 80
    - 240 KB γρήγορης RAM
    - Ταχύτητες Εσωτερικών Ρολογιών που ξεπερνάνε τα 450MHz
- Διπύρηνος επεξεργαστής Cortex-A9 στα 650MHz
- Περιφερειακοί ελεγκτές υψηλού εύρους ζώνης: 1G Ethernet, USB 2.0, SDIO
- Περιφερειακοί ελεγκτές χαμηλού εύρους ζώνης: SPI, UART, CAN, I^2C
- Ελεγκτής Μνήμης: DDR3 με 8 DMA κανάλια
- Υποδοχή MicroSD (υποστηρίζει σύστημα αρχείων Linux)
- OTG USB 2.0 PHY (υποστηρίζει συσκευή και εξυπηρετητή)
- GPIO: 6 πιεστικά κουμπιά, 4 διακόπτες ολίσθησης, 5 LED
- Έξι θύρες Pmod (1 processor-dedicated, 1 διπλή αναλογική/ψηφιακή, 3 διαφορικές υψηλής ταχύτητας, 1 logic-dedicated)

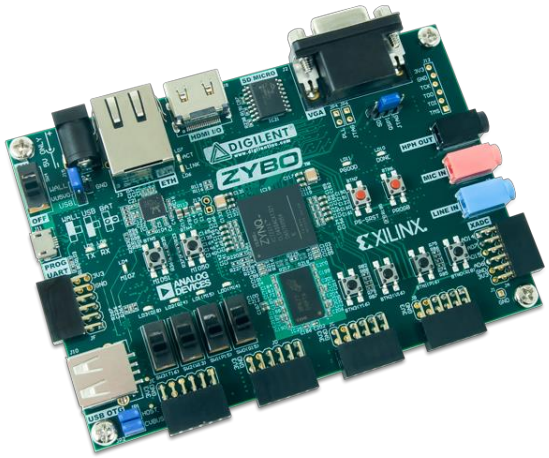


Figure 1.5: Αναπτυξιακή Πλακέτα ZYBO Zynq-7000 [6].

Πρωτόκολλο AXI

Ως μέλος του ανοιχτού προτύπου ARM AMBA® 3.0, το Πρότυπο πρωτόκολλο Advanced Extensible Interface (AXI) συνεχώς εξελισσόταν και έφτασε την τωρινή εκδοχή AXI4, η οποία θεωρείται ευρέως ως η βέλτιστη τεχνολογία διασύνδεσης για σχεδιάσεις σε FPGA. Κυρίως, AXI δίαυλοι είναι υπεύθυνοι για τις συνδέσεις μέσα στην ενσωματωμένη συσκευή, μεταξύ της επεξεργαστικής μονάδας και των υπόλοιπων πακέτων IP. Ωστόσο, υπάρχουν τρία ξεχωριστά πρωτόκολλα διαύλου του AXI4, καθένα κατάλληλο ανάλογα με την επιθυμητή φύση της σύνδεσης.

- **AXI4 — Η υψηλότερης απόδοσης διεπαφή, κατάλληλη για συνδέσεις που αντιστοιχούν στην μνήμη: μια διεύθυνση τροφοδοτείται, ακολουθούμενη από μια μεταφορά δεδομένων μέχρι και 256 λέξεων πληροφορίας.**
- **AXI4-Lite — Μια απλοποιημένη σύνδεση που υποστηρίζει μονάχα μονές συναλλαγές αντιστοιχούμενες σε μνήμη: Έχει το προτέρημα του αποτυπώματος μικρότερης λογικής: σε αυτή την περίπτωση μια διεύθυνση και μονάχα μία λέξη πληροφορίας μεταφέρονται.**
- **AXI4-Stream — Για ροή δεδομένων υψηλής ταχύτητας, που υποστηρίζει και συναλλαγές απεριόριστου μεγέθους. Δεν υπάρχει μηχανισμός διεύθυνσης, αυτός ο τύπος διαύλου ταιριάζει περισσότερο για την διεύθυνση απευθείας ροής δεδομένων μεταξύ πηγής και προορισμού. Μονάχα ένα κανάλι ορίζεται για την μεταφορά των δεδομένων, βασισμένο στο κανάλι εγγραφής δεδομένων στο Σχήμα 1.6**

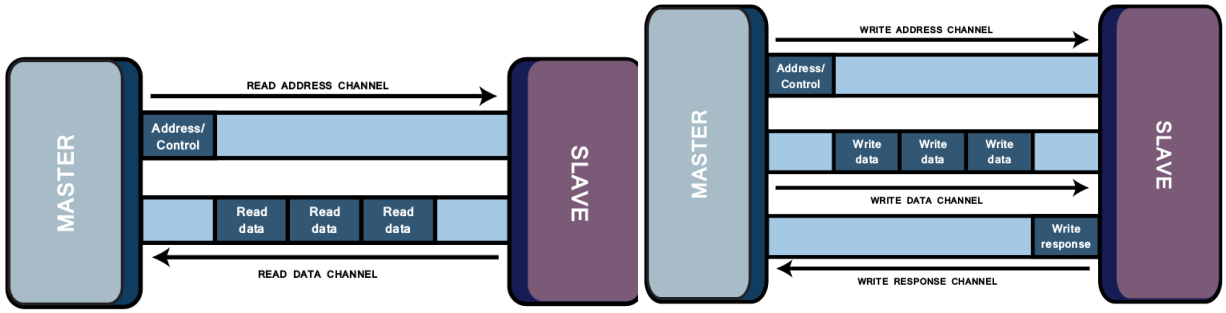


Figure 1.6: AXI4 Αρχιτεκτονικές καναλιών ανάγνωσης και εγγραφής [5].

1.2.3 Περιγραφή του Ρομπότ Laelaps II

Το Laelaps II είναι ένα τετράποδο ρομπότ εμπνευσμένο από το γατόπαρδο, το οποίο συνίσταται από ένα σώμα και και τέσσερα τρισκελή πόδια (Σχ.1.7). Τρεις αρθρώσεις υπάρχουν σε κάθε πόδι, του γοφού, του γονάτου και του αστραγάλου. Αν και οι αρθρώσεις των γοφών και των γονάτων ελέγχονται από επενεργητές, στην άρθρωση του αστράγαλου έχει τοποθετηθεί ένα στρεπτικό ελατήριο υψηλής ακαμψίας, το οποίο πρακτικά μειώνει τον αριθμό των σκελών σε δύο, με αντίστοιχα μήκη $l_1 = 250mm$ και $l_2 = 350mm$. Σχετικά με το σχεδιασμό του ελεγκτή, χρησιμοποιείται για δυναμικό τριποδισμό μια δομή ελέγχου ενεργής προσαρμοστικότητας, η οποία αποτελείται από δύο ελεγκτές χαμηλού και υψηλού επιπέδου (Σχ.1.8). Το χαμηλού επιπέδου κομμάτι είναι υπεύθυνο για την οδήγηση των ποδιών σε ελλειπτικές τροχιές μέσω της χρήσης ενός στοιχείου υπεύθυνο για το σχεδιασμό της τροχιάς, ενός άλλου για την αντίστροφη κινηματική και ενός κομματιού ενεργής προσαρμοστικότητας σε επίπεδο άρθρωσης που ελέγχει κάθε πόδι. Αντίστοιχα, το υψηλού επιπέδου τμήμα αφιερώνεται στη ρύθμιση των παραμέτρων του χαμηλού επιπέδου. Αναφορικά, στα πλαίσια του κομματιού της ενεργής προσαρμοστικότητας, εφαρμόζεται ένας ελεγκτής θέσης-ταχύτητας (PD), του οποίου η έξοδος καθορίζει τις ροπές κάθε κινητήρα σε κάθε πόδι [8].

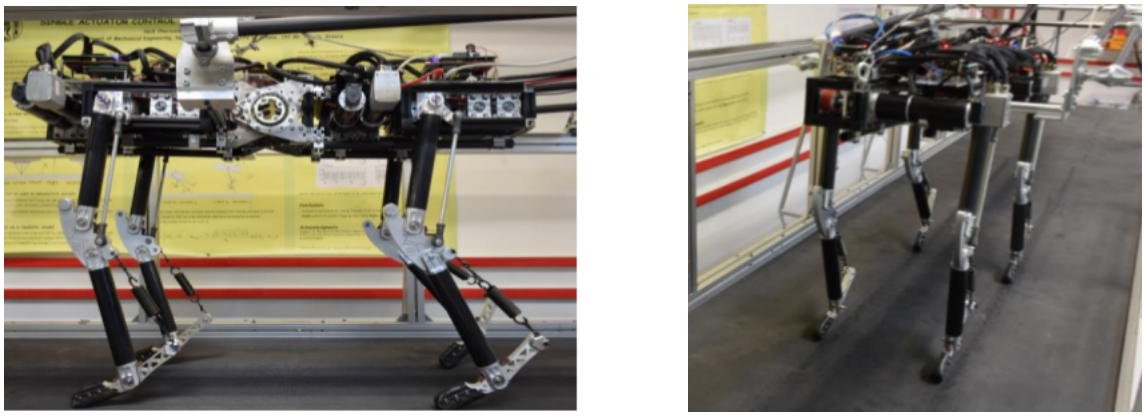


Figure 1.7: Laelaps II [7].

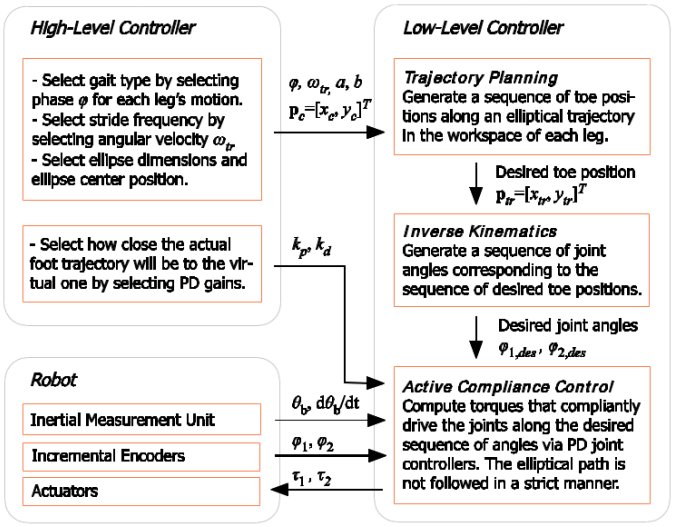


Figure 1.8: Λειτουργικό Διάγραμμα του ελεγκτή για ένα πόδι [8].

1.2.4 Επισκόπηση της EtherCat αρχιτεκτονικής ελέγχου

Προς το παρόν, εφαρμόζεται μια αποκεντρωμένη αρχιτεκτονική ελέγχου στο Laelaps II, βασισμένη στην τεχνολογία Ethernet for Control Automation Technology (EtherCat), όπου η κίνηση κάθε ποδιού ελέγχεται από μία slave συσκευή. Συγκεκριμένα, έχει σχεδιαστεί ένα δίκτυο μικροελεγκτών, υπεύθυνο για τον κινηματικό έλεγχο, τη σχεδίαση της τροχιάς και το συγχρονισμό των ποδιών. Οι παράμετροι της κίνησης ρυθμίζονται μέσω του EtherCat Master (υψηλού επιπέδου ελεγκτής), ο οποίος επικοινωνεί με τέσσερις μικροελεγκτές Delfino (χαμηλού επιπέδου ελεγκτές), οδηγώντας το σύστημα στον επιθυμητό τρόπο βάδισης. Στο [7] υπάρχουν διαθέσιμες όλες οι πληροφορίες της υλοποίησης, καθώς επίσης και το λογισμικού των μικροελεγκτών και η απαραίτητη διαδικασία διαμόρφωσης του EtherCat Master για τον έλεγχο των τεσσάρων slave συσκευών. Επιπλέον, εμπεριέχονται πληροφορίες σχετικά με το μηχανολογικό και ηλεκτρολογικό σχεδιασμό του Laelaps II.

1.3 Περιγραφή Συστήματος

Σε αντίθεση με την τωρινή υλοποίηση, που αξιοποιεί μικροελεγκτές και μία ΚΜΕ, η προτεινόμενη υλοποίηση χρησιμοποιεί ένα SoC FPGA, προκειμένου να ληφθούν οι πληροφορίες των κινητήρων και να εκτελεστεί ο έλεγχός τους. Συγκεκριμένα, η διαχείριση των κωδικοποιητών και η οδήγηση των κινητήρων πραγματοποιούνται στο FPGA, ενώ οι ελεγκτές υψηλού και χαμηλού επιπέδου υλοποιούνται στον επεξεργαστή ARM.

Η όλη διαδικασία χωρίζεται σε τέσσερις φάσεις (Σχ.1.9). Αρχικά, οι έξοδοι των κωδικοποιητών φιλτράρονται και εισάγονται στο FPGA (Phase 1). Στη συνέχεια, το FPGA εξακριβώνει τη θέση και την ταχύτητα των κινητήρων, αξιοποιώντας τις πληροφορίες που αποκτά από τους κωδικοποιητές, και τα μεταφέρει στο επεξεργαστή ARM μαζί με ορισμένα βοηθητικά σήματα, μέσω του διαύλου AXI4-Lite (Phase 2). Κατόπιν, ο συνολικός ελεγκτής εκτελείται στον επεξεργαστή μέσω της Xilinx Software Development Kit (XSDK) πλατφόρμας και αφού παραχθούν οι ροπές για κάθε κινητήρα σε κάθε πόδι, προωθούνται στο FPGA, και πάλι μέσω του διαύλου AXI4-Lite (Phase 3). Εν τέλει, κατάλληλα Pulse Width Modulation (PWM) σήματα δημιουργούνται στο FPGA, ικανά να οδηγήσουν τους κινητήρες των ποδιών, όπως ακριβώς έχει υποδείξει ο συνολικός ελεγκτής (Phase 4).

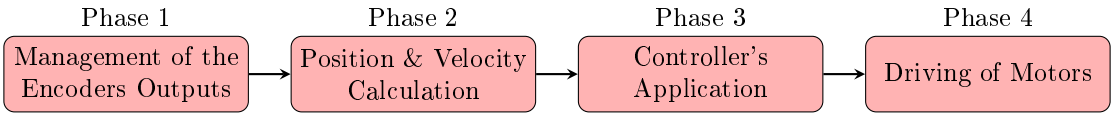


Figure 1.9: Processing Flow of the SoC FPGA Implementation

Οι λειτουργίες αυτές απεικονίζονται για ένα, δύο και τέσσερα πόδια μέσω των αντίστοιχων λειτουργικών διαγραμμάτων υλισμικού (Σχήματα 1.10, 1.11, 1.12)

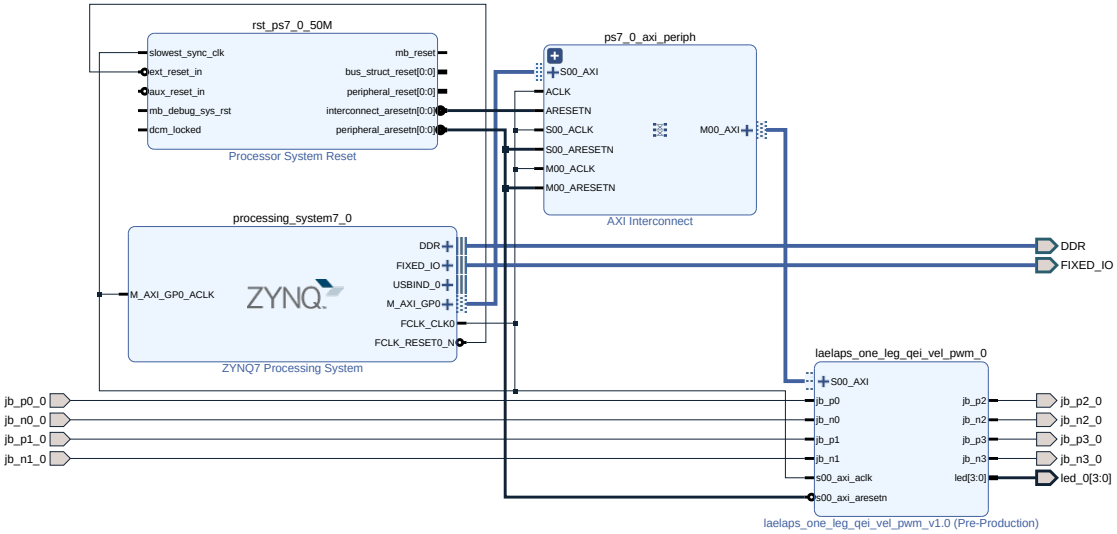


Figure 1.10: Λειτουργικό διάγραμμα σχεδίασης υλισμικού για ένα πόδι του Laelaps II.

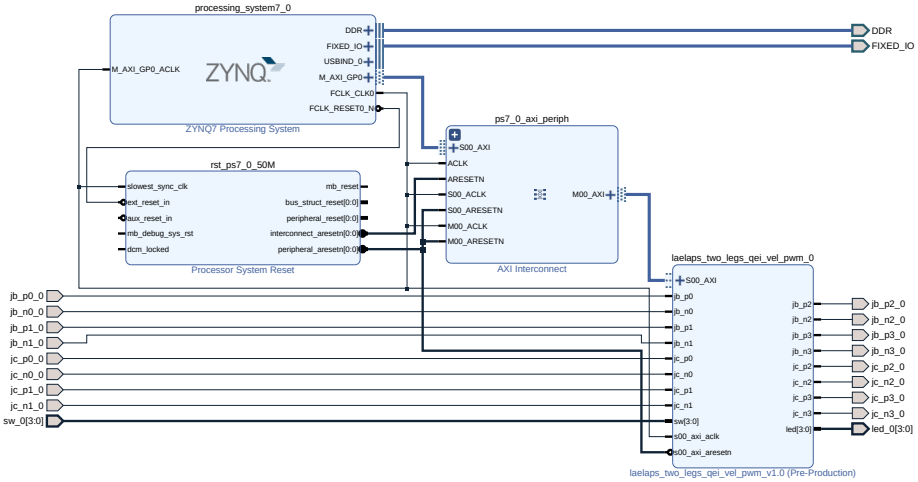


Figure 1.11: Λειτουργικό διάγραμμα σχεδίασης υλισμικού για δύο πόδια του Laelaps II.

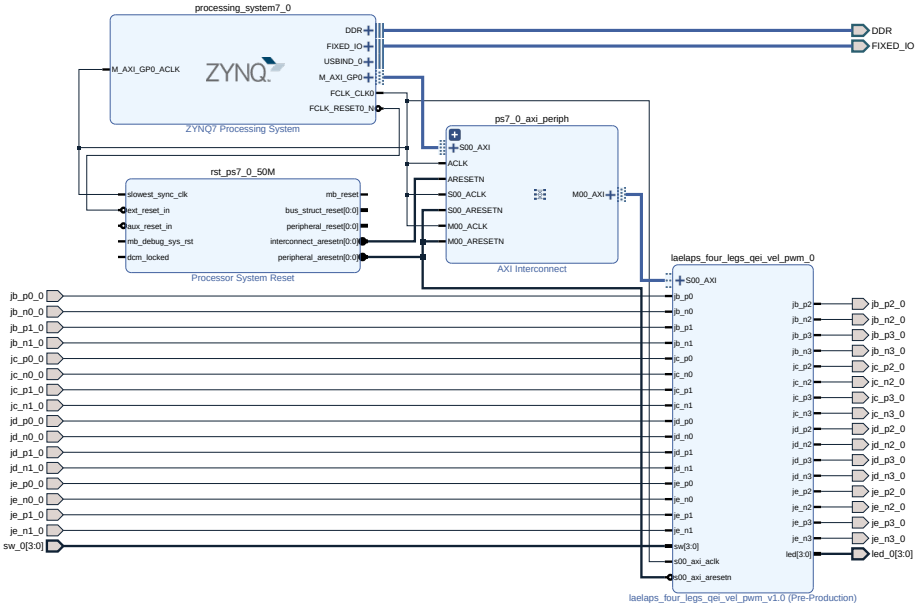


Figure 1.12: Λειτουργικό διάγραμμα σχεδίασης υλισμικού για τέσσερα πόδια του Laelaps II.

Στον πίνακα 1.1, αναλύεται η κατανάλωση των πόρων του FPGA, όπως αυτή προκύπτει από την υπηρεσία σχεδίασης Vivado Design Suite. Όπως παρατηρείται, οι κύριοι πόροι που καταναλώνονται είναι τα Lookup Tables (LUTs) και τα Input/Outputs (I/Os), ενώ οι υπόλοιποι χρησιμοποιούνται ελάχιστα. Εκτός από τα LUT Random-access Memory (LUTRAM) και τα Global Clock Buffer (BUFG), τα οποία δεν σχετίζονται με το στόχο της διπλωματικής εργασίας, τα Flip-Flops (FF) και τα DSP θα μπορούσαν να αξιοποιηθούν, ώστε να βελτιωθεί η τωρινή υλοποίηση ή για να εκτελεστούν νέες διεργασίες, χρήσιμες για την συνολική λειτουργικότητα του ρομπότ Laelaps II.

Resource	Utilization	Available	Utilization %
LUT	10709	17600	60.9
LUTRAM	60	6000	1.0
FF	2147	35200	6.1
DSP	8	80	10.0
I/O	40	100	40.0
BUFG	1	32	3.13

Table 1.1: Έκθεση κατανάλωσης πόρων του FPGA για τη σχεδίαση των τεσσάρων ποδιών.

Ένα ενδιαφέρον εύρημα απεικονίζεται στον Πίνακα 1.2, όπου συγκρίνονται οι σχεδιάσεις για ένα, δύο και τέσσερα πόδια. Ουσιαστικά, υπάρχει μια γραμμική σχέση μεταξύ της κατανάλωσης των πόρων και τον αριθμό των ποδιών, γεγονός που αποδεικνύει την δυνατότητα του συστήματος για επέκταση. Αν και το System on Chip (SoC) FPGA που χρησιμοποιείται τώρα μπορεί να μην είναι σε θέση να διαχειριστεί πολύ μεγαλύτερο αριθμό αρθρώσεων, πρέπει να τονιστεί το γεγονός ότι η Αναπτυξιακή Πλακέτα Zybo είναι μία από τις πιο απλές πλατφόρμες, τόσο σε πόρους όσο και σε επεξεργαστική ισχύ, και προορίζεται για αρχάριους μηχανικούς υλισμικού. Ως αποτέλεσμα, η κατανάλωση που παρατηρείται είναι ελάχιστη, σε σύγκριση με άλλες εναλλακτικές παρόμοιου κόστους.

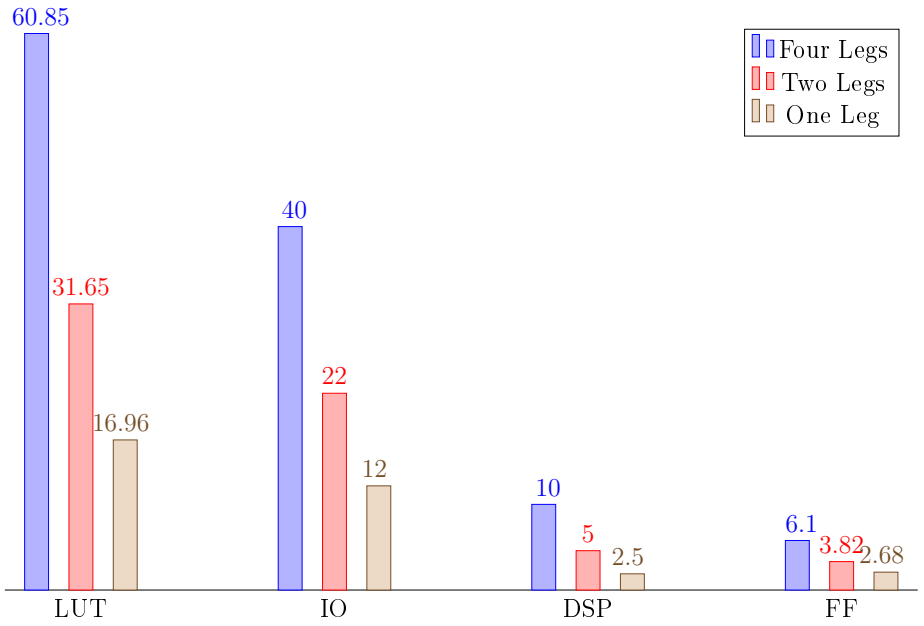


Table 1.2: Σύγκριση της κατανάλωσης πόρων (%) των σχεδιάσεων για ένα, δύο και τέσσερα πόδια.

1.4 Περιβάλλον Λογισμικού

Η απόφαση να υλοποιηθούν τα παραπάνω στοιχεία μέσω υλισμικού, οφείλεται στο γεγονός ότι δεν προβλέπεται να μεταβληθούν στο άμεσο μέλλον. Αντίθετα, το κομμάτι του ελεγκτή (υψηλό και χαμηλό επίπεδο) πρέπει να είναι ευμετάβλητο, επειδή εμπεριέχει στοιχεία τα οποία ενημερώνονται και αλλάζουν συχνά, για παράδειγμα η σχεδίαση τροχιάς και η αντίστροφη κινηματική. Ως συνέπεια, υλοποιήθηκε μια συσχεδίαση Υλισμικού/Λογισμικού, όπου ο ελεγκτής πραγματοποιήθηκε σε λογισμικό. Ο αντίστοιχος κώδικας εκτελείται στον επεξεργαστή ARM και είναι υπεύθυνος για τα δύο επίπεδα ελέγχου, καθώς και για την αποθήκευση των απαραίτητων πληροφοριών για την μετέπειτα επεξεργασία μέσω Matlab. Ο αντίστοιχος κώδικας γράφτηκε σε γλώσσα προγραμματισμού C και εκτελέστηκε σε λειτουργικό Bare-Metal, μέσω του περιβάλλοντος σχεδίασης XSDK, το οποίο βασίζεται στο IDE Eclipse 4.5.0. Κατά τη διάρκεια κυρίως των πρώτων πειραμάτων, εκτός από τη Matlab χρησιμοποιήθηκε και η υπηρεσία σχεδίασης Gnuplot για το λειτουργικό σύστημα Linux. Επιπλέον, χρησιμοποιήθηκε το εργαλείο Minicom για τη διαχείριση και την προσομοίωση του τερματικού της εφαρμογής XSDK, λόγω ασυμβατότητας με το λειτουργικό Linux.

Συνολικά, το ολοκληρωμένο διάγραμμα λειτουργίας του λογισμικού απεικονίζεται στο Σχήμα 1.13. Η κύρια συνάρτηση του C κώδικα αρχικά θέτει τις παραμέτρους της αντίστροφης κινηματικής και εκτελεί σειριακά έναν επαναληπτικό βρόχο (1), όπου καλεί τις συναρτήσεις *inv_kin* και *position_controller* για κάθε πόδι. Η πρώτη επιστρέφει τις επιθυμητές γωνίες αρθρώσεων για κάθε πόδι, οι οποίες μεταβιβάζονται ως παράμετροι στη δεύτερη (2). Κατόπιν, η συνάρτηση *position_controller* δέχεται τις απαραίτητες θέσεις και ταχύτητες από το FPGA (3) και αμέσως τα προωθεί στην συνάρτηση *pd_control* (4). Μόλις ο ελεγκτής θέσης-ταχύτητας εφαρμοστεί, η έξοδος του επιστρέφεται στη συνάρτηση *position_controller* (5) και εν τέλει μεταφέρεται πίσω στο FPGA (6). Έπειτα από την ολοκλήρωσή τους, η συνάρτηση *usleep* εισάγει την επιθυμητή χρονική καθυστέρηση και μετά το πέρας της, ξεκινά ακόμα μία επανάληψη του βρόχου. Μόλις ολοκληρωθεί ο επαναληπτικός βρόχος του ελέγχου, οι καταγεγραμμένες τιμές αποθηκεύονται σε ένα αρχείο .txt και αργότερα απεικονίζονται για λόγους εποπτείας. .

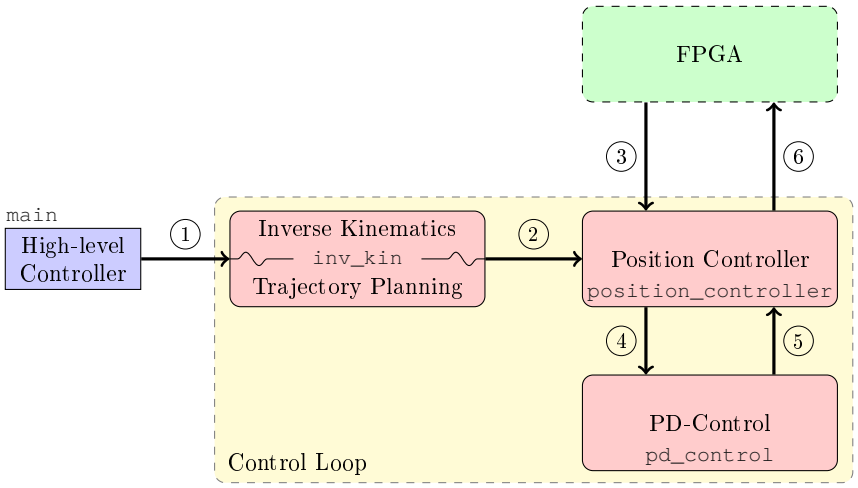


Figure 1.13: Λειτουργικό Διάγραμμα του Λογισμικού.

1.5 Ενοποίηση Συστήματος

Για τη σύνδεση της Αναπτυξιακής Πλακέτας Zybo και του ρομπότ Λαίλαψ ΙΙ, χρησιμοποιήθηκαν μία ενδιάμεση και τέσσερις δευτερογενείς πλακέτες, οι οποίες συνδυάστηκαν σε ένα συμπαγή πύργο, του οποίου η τελική εκδοχή απεικονίζεται στο Σχήμα 1.14.

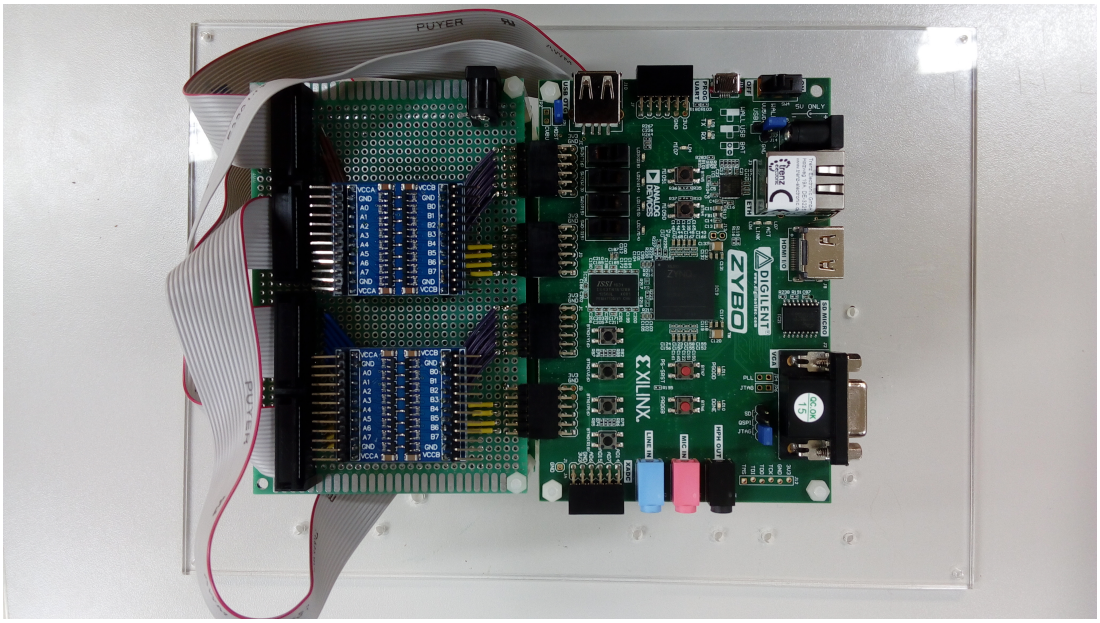


Figure 1.14: Πύργος Κεντρικού Ελέγχου.

Ο πύργος ελέγχου τοποθετήθηκε στον κορμό του Λαίλαψ ΙΙ, για χωροταξικούς λόγους και προκειμένου να ελαχιστοποιηθούν οι αποστάσεις από τα μπροστά και πίσω πόδια (Σχ.1.15). Στη συνέχεια, οι κωδικοποιητές και οι drivers των κινητήρων συνδέθηκαν στον πύργο, ενώ οι πλακέτες τροφοδοτήθηκαν από εξωτερική πηγή. Ο προγραμματισμός και η επικοινωνία με την Πλακέτα Zybo επιτεύχθηκε μέσω καλωδίου USB το οποίο συνδεόταν με Προσωπικό Υπολογιστή.

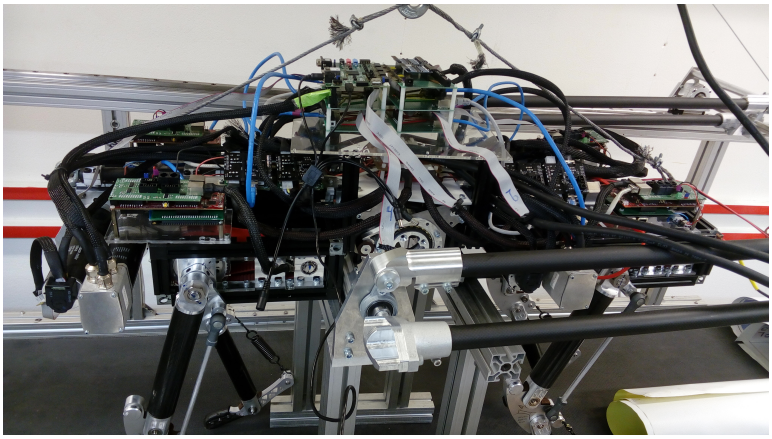


Figure 1.15: Κεντρικός Πύργος Ελέγχου ενοποιημένος με το Λαίλαψ ΙΙ.

Το κεντρικό σύστημα ελέγχου αποτελείται από δύο μονάδες, το τετράποδο ρομπότ Λαίλαψ ΙΙ και το SoC FPGA. Αρχικά, οι κωδικοποιητές παράγουν εξόδους της τάξης των 0-5V, οι οποίες μεταφέρονται μέσω των δευτερογενών πλακετών στην ενδιάμεση και σε έναν μετατροπέα τάσης.

Μετά την μετατροπή τους σε σήματα εύρους 0-3.3V, συνδέονται κατευθείαν με τις θύρες Pmod της Αναπτυξιακής Πλακέτας Zybo. Σε αυτό το σημείο, το FPGA εκκινεί το φιλτράρισμα τους και υπολογίζει την αντίστοιχη θέση και ταχύτητα για κάθε κινητήρα. Κατόπιν, οι πληροφορίες αυτές προωθούνται σε slave καταχωρητές της διεπαφής AXI4-Lite, οι οποίες είναι διασυνδεδεμένες με το δίαυλο AXI4-Lite. Συγχρόνως, στην άλλη πλευρά του διαύλου, ο επεξεργαστής έχει ήδη εκτελέσει τον ελεγκτή υψηλού επιπέδου και έχει διαμορφώσει όλες τις απαραίτητες παραμέτρους για τη σχεδίαση και τον έλεγχο της κίνησης των ποδιών. Συγκεκριμένα, ο βρόχος ελέγχου έχει ήδη ξεκινήσει και η αντίστροφη κινηματική μαζί με τη σχεδίαση της τροχιάς, από τον ελεγκτή χαμηλού επιπέδου, έχουν ήδη εκτελεστεί για αυτή την επανάληψη του βρόχου. Ως συνέπεια, οι επιθυμητές θέσεις για κάθε άρθρωση έχουν υπολογιστεί και μαζί με τις θέσεις και τις ταχύτητες των κινητήρων, όπως αυτές λαμβάνονται από το FPGA μέσω του διαύλου AXI4-Lite, αποστέλλονται στη μονάδα του ελέγχου Θέσης-Ταχύτητας. Βάση των προκαθορισμένων κερδών ελέγχου, ο ελεγκτής θέσης-ταχύτητας εφαρμόζεται και η έξοδός του, η οποία θα καθορίσει τη ροπή του εκάστοτε κινητήρα, επιστρέφεται στο FPGA μέσω της διεπαφής AXI4-Lite. Στο σημείο αυτό, οι αντίστοιχοι slave καταχωρητές, που περιέχουν τις εξόδους του ελεγκτή θέσης-ταχύτητας μαζί με ορισμένα βοηθητικά σήματα, διασυνδέονται με τα αντίστοιχα PWM κομμάτια του FPGA. Κατάλληλα PWM σήματα δημιουργούνται και κατανέμονται στις ενδιάμεσες πλακέτες και τελικά στους επενεργητές των κινητήρων που ελέγχουν τα πόδια του ρομπότ. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς καθόλη τη διάρκεια του εκάστοτε πειράματος, επιτρέποντας στο σύστημα να συγκλίνει στην επιθυμητή κατάσταση.

1.6 Αξιολόγηση Συστήματος

Σε αυτό το σημείο παρουσιάζονται τα αποτελέσματα από το τελικό πείραμα, στα πλαίσια του οποίου έξι αρθρώσεις ελέγχθηκαν με επιτυχία και εκτέλεσαν ένα μοτίβο τριποδισμού σε συγχρονισμό, ενώ το Λαίλαψ ΙΙ βρισκόταν στον αέρα και επομένως δεν υπήρχε επαφή με το έδαφος. Σε όλα τα πειράματα, η συχνότητα του επεξεργαστή ήταν 650MHz και του FPGA 83MHz.

Όλες οι σχετικές παράμετροι του πειράματος αναφέρονται στον Πίνακα 1.3, ενώ η επιθυμητή ελλειπτική τροχιά για το άκρο κάθε ποδιού συγκρίνεται με την πραγματική απόκριση τους. Σε αυτό το πείραμα, το σύστημα ρυθμίζεται με τις κατάλληλες παραμέτρους, εκτός από τις ελλειπτικές παραμέτρους, *a ellipse* και *b ellipse*, οι οποίες είχαν τεθεί στο μηδέν. Μόλις ξεκινήσει ο βρόχος ελέγχου, στα πρώτα *Lerp Inteval* δευτερόλεπτα του πειράματος, οι τιμές τους αυξάνονται γραμμικά με το χρόνο. Ταυτόχρονα, το ρομπότ αρχίζει να κινείται αργά, μέχρι να φτάσει την μόνιμη κατάσταση. Κατά τα τελευταία *Lerp Inteval* δευτερόλεπτα του πειράματος, οι ελλειπτικές παράμετροι μειώνονται γραμμικά στις αρχικές τους τιμές και το ρομπότ επιβραδύνει και τελικά σταματά την κίνησή του. Τέλος, οι αποθηκευμένες πληροφορίες επεξεργάζονται μέσω Matlab και προκύπτουν κατάλληλα διαγράμματα.

Trajectory Parameters						
x center	y center	a ellipse	b ellipse	Frequency	Phase	Flattening Parameter
0cm	57cm	3cm	3cm	1Hz	0/180°	0

Control Gains		PWM Saturation Limits		Experiment Duration	Lerp Interval
K_p	K_d	Positive	Negative	20sec	5sec
500	300	+30%	-30%	Control Frequency	1kHz

Table 1.3: Παράμετροι του Πειράματος Κίνησης

Στο Σχήμα 1.16, απεικονίζονται οι καταγεγραμμένες αποκρίσεις των άκρως κάθε ποδιού, μαζί με τις αντίστοιχες επιθυμητές ελλειπτικές τροχιές. Αν και παρατηρούνται σφάλματα, οι επιθυμητές τροχιές ακολουθούνται σε αποδεκτά πλαίσια. Λόγω της ανύψωσης του ρομπότ, δεν υπήρχαν διαταραχές από το έδαφος, ενώ επιλέχτηκαν σχετικά υψηλά κέρδη ελέγχου.

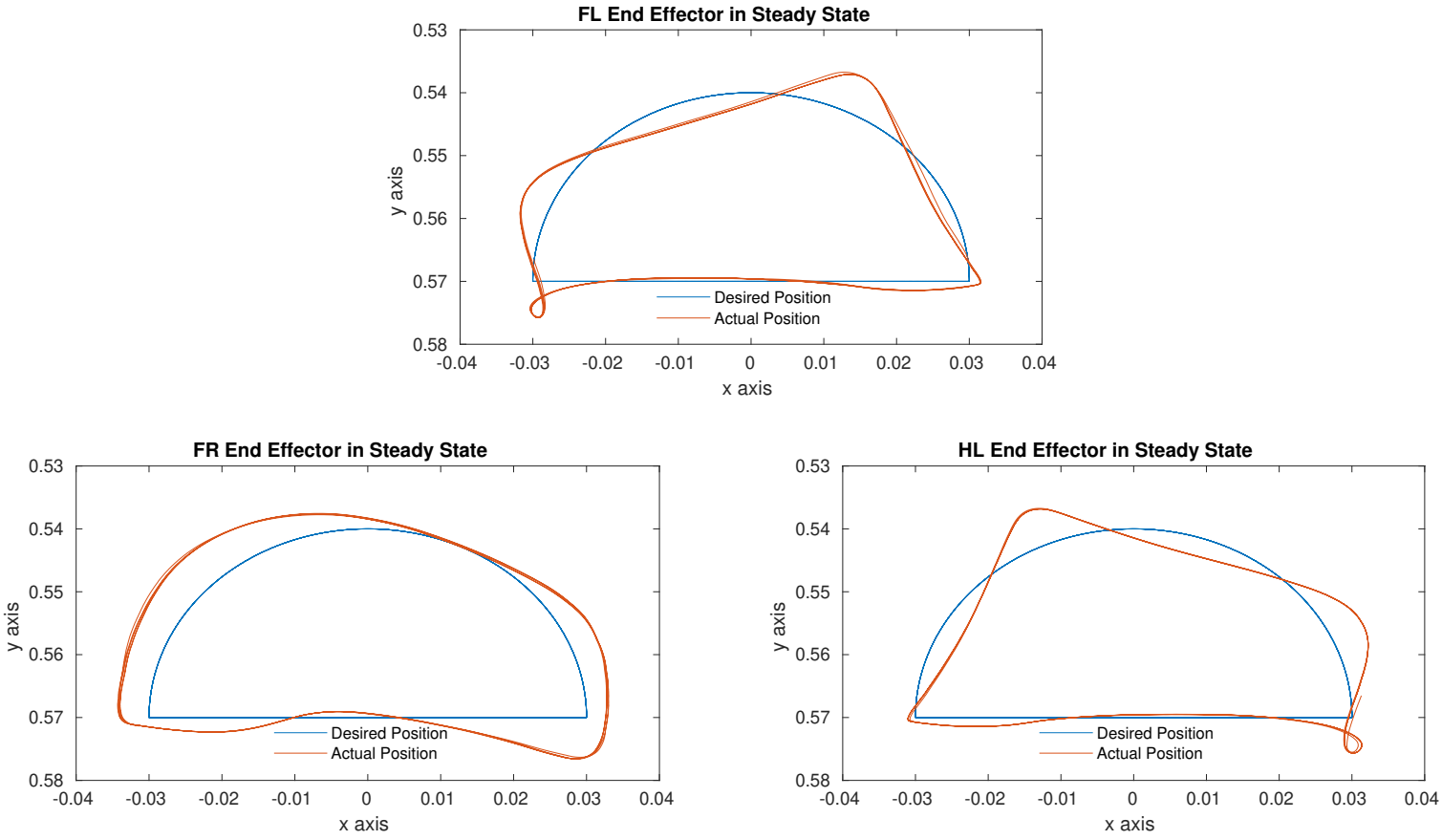


Figure 1.16: Καταγεγραμμένη Απόκριση των άκρων των ποδιών μαζί με τις επιθυμητές τροχιές.

1.7 Συμπεράσματα και Μελλοντική Εργασία

1.7.1 Συμπεράσματα

Ο στόχος αυτής της διπλωματικής εργασίας στέφθηκε με επιτυχία, εφόσον παράχθηκε ένα πλήρες λειτουργικό σύστημα ελέγχου, ικανό να ελέγξει και τα τέσσερα πόδια του τετράποδου ρομπότ Λαίλαψ ΙΙ. Το συνολικό σύστημα φιλοξενείται σε μία ιδιαίτερα οικονομική SoC FPGA πλακέτα, η οποία όχι μόνο παρουσιάζει ικανοποιητική απόδοση, αλλά επιπροσθέτως απαρτίζεται από μονάχα μία συσκευή, και επομένως αναιρεί την ανάγκη ύπαρξης ενός εκλεπτυσμένου πρωτόκολλου επικοινωνίας, όπως το EtherCat, το οποίο είναι υποχρεωτικό σε αποκεντρωμένες αρχιτεκτονικές. Το συνολικό κόστος αντιστοιχεί μονάχα στο κόστος της αναπτυξιακής πλακέτας Zynq™- 7000 (ZYBO), το οποίο αποτελεί το 2% του κόστους της προηγούμενης κεντρικής και 20% του κόστους της τωρινής αποκεντρωμένης αρχιτεκτονικής. Ως αποτέλεσμα, η αγορά εφεδρικών ανταλλακτικών είναι σίγουρα οικονομικά εφικτή, ενώ αντιμετωπίζεται ένα από τα σημαντικότερα μειονεκτήματα των κεντρικών αρχιτεκτονικών, αφού ακόμα και σε περίπτωση βλάβης, η πλακέτα μπορεί εύκολα να αντικατασταθεί και επαναπρογραμματιστεί σε λίγα δευτερόλεπτα.

Επίσης, το σύστημα διαθέτει υψηλή ανταπόκριση. Από την μία πλευρά, η συνύπαρξη του FPGA μαζί με τον επεξεργαστή ARM στην ίδια ψηφίδα, μειώνει δραματικά τυχόν επιβαρύνσεις στην επικοινωνία, ενώ προσφέρει υψηλότερο εύρος ζώνης. Από την άλλη πλευρά, η απασχόληση του επεξεργαστή είναι αρκετά χαμηλή, δεδομένου πως οι υπολογισμοί που χρειάζεται να εκτελεστούν απαιτούν αμελητέα προσπάθεια, ειδικά όταν λαμβάνεται υπόψιν και η συχνότητα λειτουργίας του. Συγκεκριμένα, διεργασίες όπως ο υπολογισμός της θέσης και της γωνιακής ταχύτητας για κάθε κινητήρα, εκτελούνται στο FPGA, ενώ αντιθέτως στην τωρινή υλοποίηση μικροελεγκτές αναλαμβάνουν αυτό το φορτίο. Το γεγονός αυτό εκτός του ότι αποσυμφορεί τον επεξεργαστή, επιπλέον εκμεταλλεύεται τη χρήσιμη ιδιότητα των FPGA για παράλληλη επεξεργασία, η οποία ελαττώνει σημαντικά το χρόνο επεξεργασίας, ενώ περιορίζει πιθανές καθυστερήσεις.

Συμπερασματικά, κρίνοντας από τα αισιόδοξα αποτελέσματα έως τώρα, μπορούμε να χαρακτηρίσουμε τη συνεργασία μεταξύ των πεδίων της ρομποτικής και των FPGA ως επιτυχημένη και συνεπώς ότι αποτελεί κλάδος στον οποίο αξίζει να επενδύσει κάποιος.

1.7.2 Μελλοντική Εργασία

Παρά την αποδεδειγμένη λειτουργικότητα της περιγεγραμμένης υλοποίησης του κινηματικού ελέγχου, τόσο σε λογισμικό όσο και σε υλισμικό, το ανεπτυγμένο σύστημα έχει αρκετό περιθώριο για βελτίωση. Επιπροσθέτως, η ανάπτυξη του συστήματος θα συνεχίσει και μετά την ολοκλήρωση αυτής της διπλωματικής εργασίας, με σκοπό να φτάσει στο μέγιστο δυνατό αποτέλεσμα. Παρ' όλα αυτά, θα αναφερθούν για μελλοντική αναφορά τόσο οι προσεχείς όσο και οι τυχόν μελλοντικές αναβαθμίσεις.

Αρχικά, θα προστεθεί στο λογισμικό του συστήματος μια Υπηρεσία Διαχείρισης Διακοπών, προκειμένου να επιτευχθεί καλύτερη χρονική ακρίβεια, καθώς επίσης και διάφορες βολικές υπηρεσίες. Η ιδιότητα αυτή θα μπορούσε να επιτρέψει την παράλληλη εκτέλεση τόσο του αλγορίθμου ελέγχου όσο και της αποθήκευσης των καταγεγραμμένων πληροφοριών, ενώ ενδεχομένως μπορεί να δώσει στο χρήστη τη δυνατότητα να εισαγάγει εντολές κατά τη διάρκεια των πειραμάτων και συνεπώς να μεταβάλλει οποιαδήποτε παράμετρο σε πραγματικό χρόνο ή να τερματίσει ομαλά την εκτέλεση του.

Επίσης, στην τωρινή υλοποίηση μονάχα ένας εκ των δύο πυρήνων του επεξεργαστή ARM Cortex-A9 χρησιμοποιείται, ενώ ο δεύτερος παραμένει ανενεργός. Πιθανώς το επεξεργαστικό φορτίο θα μπορούσε στο μέλλον να διαιρεθεί και ένας επεξεργαστής θα μπορούσε να αναλάβει τον κινηματικό έλεγχο και την υπηρεσία διακοπών, ενώ ο δεύτερος θα μπορούσε να αναλάβει την επικοινωνία με το χρήστη. Επιπροσθέτως, η συχνότητα λειτουργίας του επεξεργαστή θα μπορούσε να μειωθεί, δεδομένου ότι η τωρινή υλοποίηση δεν είναι υπολογιστικά απαιτητική, και συνεπώς η κατανάλωση ενέργειας θα μειωνόταν.

Chapter 2

Introduction

2.1 Literature Review

2.1.1 Quadruped Robots

In general, autonomous robotic systems may be classified in two major areas, Manipulation and Mobile robotics. Legged robots are one of the three fundamental configurations available for mobile robots locomotion on ground, among the rotational devices, such as tracks and wheels, and articulated structures similar to a snake's body. In fact, they display distinct attributes, which have proved them to be rather fitting for a multitude of cases [11]. As a result, they are rendered as a highly advanced section of modern engineering that keeps evolving rapidly the last years. Research labs constantly develop complex and innovative designs that either exceed previous or exhibit new and interesting qualities. Some of the most renowned are presented in Fig.2.1, both designs by Boston Dynamics.

(a) BigDog (b) WildCat

Figure 2.1: Boston Dynamics Legged Robots[†].

The main advantage of legged robots, compared to other mobile vehicles, appears to be their superior mobility in natural terrains [11]. In particular, by modifying their legs configuration, they adapt to surface irregularities and hence achieve contact with selective points of the ground in accordance with the terrain conditions. Thus, they confront both obstacles and stairs much easier, while maintaining smoothness in their motion. Moreover, compared with tracked or wheeled vehicles, legged devices require less energy to elude from sunken areas. Lastly, a recently investigated advantage concerns failure tolerance, since legged vehicles may present redundant number of joints and hence maintain balance and continue their locomotion even with one or more of their legs damaged, in contrast with wheeled vehicles,

[†] https://www.bostondynamics.com/robots

where the failure in one of the wheels, severely deteriorates overall mobility. These features, in conjunction with their small size, renders them a suitable candidate for missions, where terrain properties are unknown a priori and access might be restricted, like explorations, excavations and rescues.
Inspired by nature, legged robots can also be classified into mammal and reptile robots, according to their mechanical structures. In fact, mammal robots are able to efficiently carry much more payload than reptilian, while maintaining adequate walking speed. Inasmuch as most mammalian animals have four legs, quadruped robots became an important development direction in the robotics field. Many scientific organizations have been working on them, as they are optimal for the implementation of biologically inspired locomotion of dynamical gaits. Although the concept of an even number of legs is considered by itself universal, as it allows efficient gaits and stability performance, quadruped robots stand out as they are the first to ensure such a stability. Specifically, a stable tripod is entrenched, when moving only one leg, making them far more stable than bipedal robots . Despite the fact that further increase of joints provides even better stability, it requires far more complex control and computational resources, meaning that it is not always preferable. Concurrently, the construction and preservation of four legs is less intricate and tedious than six ones or more.
In spite of the aforementioned aspects, it should be noted that legged vehicles still suffer from limitations, since they require advanced control algorithms, are difficult to manufacture and exhibit low speeds. Simultaneously, most of them are heavy and quite energy consuming due to their large number of actuators, required to handle the multiple Degrees of Freedom (DoF) legs. On top of that, the research of posture control strategy of stability faces numerous difficulties, since it relies on the effects of complicated factors, like the instability in the changing terrains, impacts from external forces and the large time varying of parameters.
To sum up, over the last five decades, quadruped robots have become faster, more efficient, more robust, and have a considerable payload capability. Additionally, their demonstration of superior mobility of different terrains and at different gaits and speeds, distinguishes them for all Mobile robots and places them at the center of attention in the robotics research. Therefore, those features, coupled with the prosperous development of the robotics industry, indicate that quadruped robots are undoubtedly a field worth investing in [12, 13].

2.1.2 FPGAs

Although its foundations were set during the wide development of integrated circuits from the early 1960s until the mid-1980s, the first modern FPGA was introduced in 1984 by Xilinx (Fig.2.2). In spite of their poor performance at first, through constant evolution, FPGAs managed to acquire a leading place in the field of digital circuit implementation media over the last years.

Figure 2.2: The first commercially viable field-programmable gate array in 1985 – the XC2064 [†].

Based on the preexisting architectures of Programmable Read Only Memory (PROM), Programmable Array Logic (PAL) and Programmable Logic Array (PLA), FPGAs exhibited significant advantages over Application Specific Integrated Circuit (ASIC) and PAL technologies that thrived in the electronics market during the mid-1980s. Specifically, the

ability of instant configuration in conjunction with their low cost allowed them to overtake ASICs, which required months and hundreds of thousands of dollars for fabrication purposes. Additionally, due to the elimination of the AND array, FPGAs gained an architectural advantage over PALs. Both performance and capacity were disengaged from limitations imposed by the AND array, while FPGA designers acquired major architectural freedom [14, 15].

Nowadays, new products associated with FPGAs have emerged that provide intriguing capabilities. In fact, both Xilinx and Altera (Subsidiary of Intel), leading companies in the FPGA market, followed the ASIC migration to SoC technologies that integrate the hardware programmability of an FPGA with the software programmability of a processor. Notable examples are the Xilinx Zynq All Programmable SoC (Fig.2.3) and the Altera SoC FPGA classes. Furthermore, Xilinx has further expanded the SoC portfolio and introduced the Multi-Processor SoC (MPSoC) (Fig.2.4) and Radio Frequency SoC (RFSoC) families that produce overall benefits of lower cost and power reduction, while adapting to a broad base of applications.

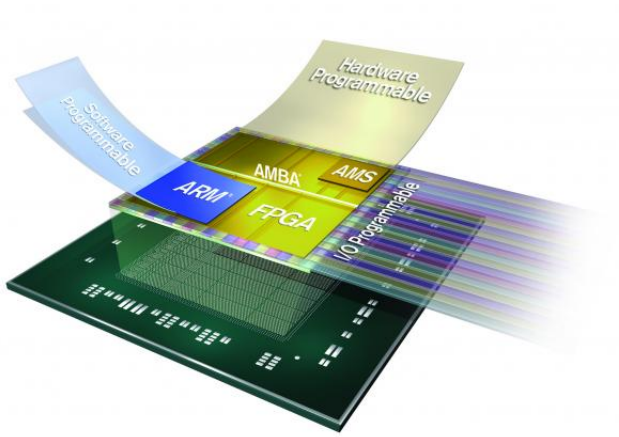


Figure 2.3: Xilinx's Zynq-7000 All Programmable SoC †.

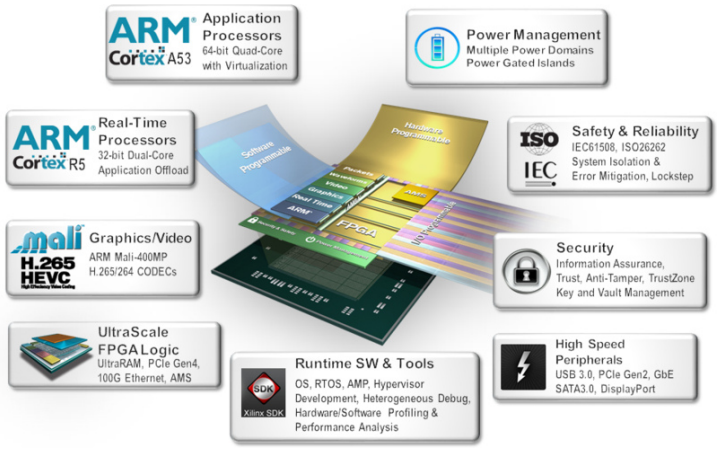


Figure 2.4: Zynq UltraScale+ MPSoC †.

2.1.3 FPGAs in Quadruped Robots

According to the best knowledge of the author, after a thorough survey in research publications, it was concluded that collaboration between robots and FPGAs is rather rare. Notable are the following examples where cooperation is achieved and praiseworthy results are introduced.

In [1] an innovative turtle-inspired mobile amphibious spherical robot is presented, which utilizes a Xilinx all-programmable Zynq-7000 SoC FPGA (Z-7000) in order to minimize the power consumption of the integrated circuits and introduce an articulated design (Fig.2.5). In this example, certain power reduction mechanisms of the Zynq SoC were utilized, pertinent with dynamical voltage frequency scaling techniques and dynamical power management.

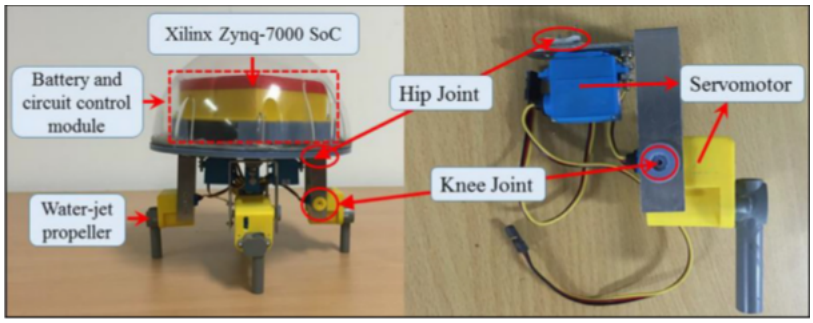


Figure 2.5: Prototype of the improved 3D-printed amphibious spherical robot [1].

A rather interesting finding was the fact that certain attempts have been made to exploit the FPGA technology in multi-legged robots. For instance, in [2] locomotion patterns for three different legged robots (biped, quadruped and hexapod) are generated through a neuromorphic system developed on a Spartan 6 FPGA board (Fig.2.6). A Spiking Neural Network (SNN) has been implemented on the FPGA, which acts as a Central Pattern Generator (CPG) programmed to create the necessary gaits for each robot and hence manage to control its motion. Similar work is presented in [16], where a CPG architecture was attached as a specialized co-processor to a Microblaze soft-processor embedded in a Spartan-6 FPGA device.

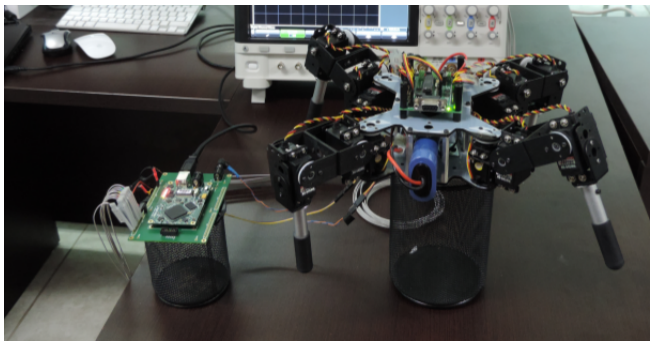


Figure 2.6: Quadruped robot setup with the FPGA board on the left [2].

In [3] a miniature legged Hexapod Robot is presented, where its processing layer is implemented on a FPGA (Fig.2.7). Specifically, for each leg's low-level control a separate processor was used, while another processor was in charge of the high-level control. In addition, the

[†]https://www.xilinx.com/

required motion control was achieved via PWM signals from the FPGA. Noteworthy is the alleged versatility of the design, concerning the number of joints, as an increase in their number would require, in general, rather simple modifications.

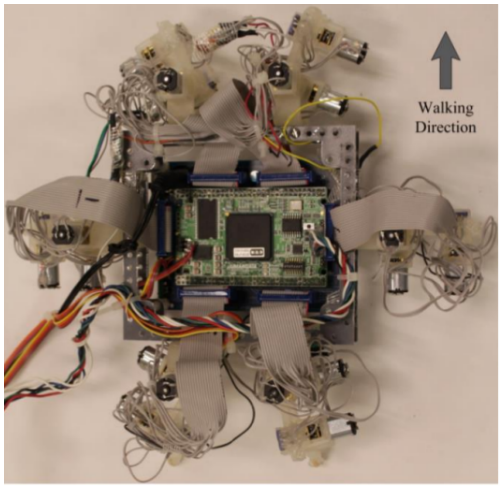


Figure 2.7: Hexabot Robot [3].

2.2 Motivation

The goal of this thesis is to introduce a centralized control scheme for the quadruped robot Laelaps II of the CSL - EP Laboratory (Fig.3.8), using a SoC that comprises an ARM-Cortex A9 processor and an FPGA of the Zynq-7000 Xilinx family, as part of the Zybo Development Board. Specifically, a Hardware/Software Co-design architecture is proposed, where the management of the robot's motors and encoders will be handled on the FPGA, while the kinematic control will be executed on the ARM processor.
Considering the requirements of a quadruped robot, it is believed that integrating a SoC FPGA would display the following attributes. First of all, our design aims at reducing both physical dimensions and weight. The current design employs a decentralized scheme that consists of one EtherCat Master and four EtherCat Slave towers, which burden the system both in weight, power consumption and wiring. On the other hand, only one of the Zybo Development Boards is planned to be utilized, meaning that the final system would be more compact and light.
Furthermore, another motive of the suggestion to combine a quadruped robot with a SoC FPGA platform, is their alleged attribute of low cost and reduced power consumption. Since both aspects are crucial for the design of any robot, it would be fascinating, if those characteristics applied in this attempt as well.
Additionally, the coexistence of the FPGA and the ARM processor on the same chip dramatically reduces their communication overhead, while establishing a higher bandwidth. Simultaneously, the FPGA's valuable feature of parallel processing will probably further reduce the processing time, while restricting any possible latencies.
Lastly, the Zybo Board exhibits the potential to support architectures with more joints, as it is equipped with multiple I/O peripherals. Therefore, if the proposed design does not deplete the SoC FPGAs resources, this aspect would prove to be highly alluring for future applications.

2.3 Thesis Outline

Overall, the thesis is organized in seven chapters, including the introductory chapter, where the literature review and motivation of this project are presented.
In the **second** chapter, an introduction to the hardware components that were utilized is presented, including FPGAs and SoCs. Consequently, a brief description of the Laelaps II robot is included.
In the **third** chapter, the whole system's composition is explained, in addition to an extensive breakdown of its functionality. Next, the hardware segment of the design, realized on the FPGA, is expounded, which concerns the handling of the robot's motors and its encoders.
In the **fourth** chapter, the software implementations are depicted, which involve the ARM processor and the post-processing procedures that took place in Matlab and Gnuplot.
In the **fifth** chapter, the interconnection between the Hardware and the Software sections is clarified, while the finalized architecture is introduced, where the SoC Board is connected with the Laelaps II robot via a custom made circuit board.
In the **sixth** chapter, an experimental validation of the robot is displayed containing the results of several locomotion experiments executed with Laelaps II using our finalized architecture.
In the **seventh** chapter, the conclusions of the thesis are summarized and future work is suggested.

Chapter 3

Hardware Platform

3.1 Field-Programmable Gate Arrays (FPGAs)

Field-programmable gate arrays (FPGAs) are digital integrated circuits that can be reprogrammed after manufacturing. Specifically, they are defined as pre-fabricated silicon devices, which can be configured as almost any possible digital circuit via the respective electrical programming.
FPGAs consist of arrays of programmable logic blocks that can be interconnected in various ways (Configurable Logic Blocks (CLBs)). In particular, some of the basic ones are made of NAND gates, FF, LUT and PAL style wide input gates, as well as multiplexers. Frequently, modern FPGAs also contain certain specific purpose blocks, which contain a diverse combination of blocks, like memory (Block Random-access Memory (BRAM), LUTRAM), multipliers, adders and DSP blocks. Apart from the core programmable fabric, programmable I/O blocks are included, around the periphery of the chip, that serve as connections with off-chip units. Both the logic and I/O blocks are linked with each other by programmable routing resources that consist a programmable routing interconnect (Fig.3.1).

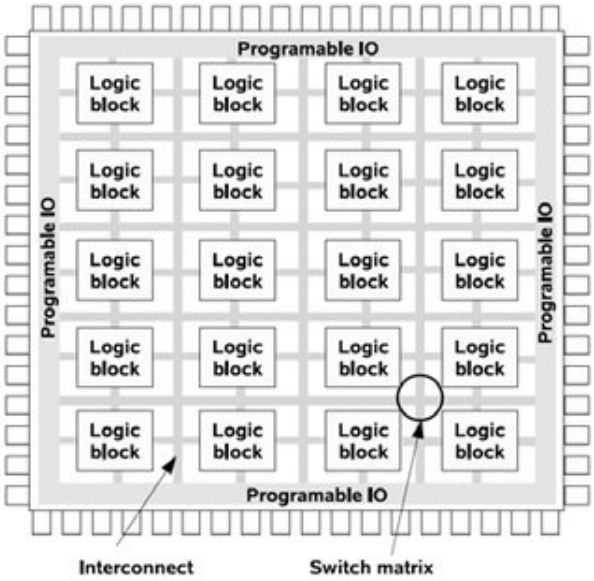


Figure 3.1: General architecture of FPGAs [4].

The actual specification of the FPGA designs can be realized through two standard approaches: Hardware Description Language (HDL)-based and Schematic-based, depending upon the complexity of the design. In particular, for computationally demanding and intricate algorithms the most prevalent method is the HDL-based (VHDL or Verilog) design entry. Another appealing method is the Xilinx High-Level Synthesis software, Vivado HLS, which transforms a C specification into a Register Transfer Level (RTL) implementation that is synthesized into a Xilinx FPGA [17].
The multiple I/O, combined together with the vast number of available interconnections, allow the FPGA to process a lot of inputs in parallel. Compared to Central Processing Unit (CPU), it can result in significant process acceleration, considering that a CPU cannot process more than one instruction at a time. At the same time, FPGAs function in much slower frequencies than CPUs (up to 550MHz), resulting in much smaller power consumption. Moreover, instead of being limited to an inalterable internal functionality, hardwired during fabrication, FPGAs can be reprogrammed when desired, according to the needs of the designer. Therefore, besides the opportunity to implement any desired digital circuit, the designers can modify their hardware either because it may prove faulty or to simply create something different.

In summary the advantages of FPGAs are:

- **Reconfigurability**
- **Parallel Processing**
- **Low Power Consumption**
- **Multiple Peripherals**

However, as natural, FPGAs also exhibit certain drawbacks. The aforementioned routing interconnect covers almost 90% of their area, a characteristic that heavily damages their performance. In fact, their main asset of flexibility makes them appreciably slower, larger and more power consuming than their ASIC counterparts. Additionally, it should be noted that writing parallel code with HDLs is more challenging than developing software. Thus, for some applications, General Purpose Processors (GPPs) or DSPs, which are configured using sequential languages like C or C++, may be able to meet the desired performance requirements and hence an FPGA implementation would be needless. Nevertheless, their low volume cost in conjunction with their fast producing rate renders them an enthralling option for digital system implementations [4, 18].

3.2 SoC FPGAs

Thanks to major technological breakthroughs, new architectures have emerged that combine FPGA's logic blocks and the routing interconnect, with memory blocks, one or more microprocessors, embedded IP cores and DSP blocks integrated on a single chip to corroborate the implementation of Programmable SoC (PSoC) designs. Similarly, certain SoC FPGA devices combine FPGA architectures with one or more hard-core 10, embedded on the same chip, while soft processors are also an available option (Fig.3.2).
Due to the fact that in this thesis a SoC FPGA, which comprises an ARM processor with an FPGA, was utilized, only that architecture will be analyzed. In fact, the review will mainly focus on the new generation of the All-Programmable SoC by Xilinx, the Zynq.
The main distinctive trait of Zynq is its amalgamation of traditional FPGA logic fabric, based on Xilinx 7-series FPGA architecture, with a dual-core ARM Cortex-A9 processor, capable of supporting operating systems such as Linux. Basically it encapsulates two parts, the Processing System (PS), revolved around the processor, and the Programmable Logic (PL), which encloses the FPGA. In addition, the communication between the two units is achieved

[†] https://www.datarespons.com/soc-fpga-evaluation-guidelines/

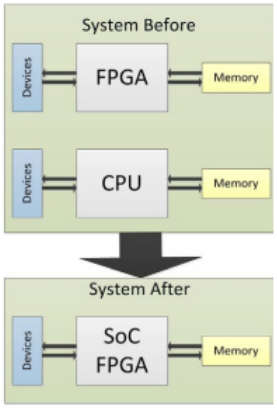


Figure 3.2: Comparison between standalone CPU and FPGA solution and SoC FPGA solution [†].

via industry standard AXIs, which ensure low latency and high bandwidth connections (Fig.3.4).
As a consequence, hardware-software cooperation is attainable, where both the FPGA logic and processor can reach their full potential. Namely, the negative overhead of interfacing between two physically separate devices is avoided, while the overall cost and physical size drop substantially. Apart from the PS and the PL, a variety of peripherals, an embedded memory and the previously mentioned AXI communication interface are featured in the Zynq architecture.

Summarized, the following benefits of the SoC technology emerge:

- **Higher Bandwidth Communication between Processor and FPGA**
- **Hardware/Software Co-design**
- **Compact Size**
- **Low Cost**

AXI Protocol

As part of the ARM AMBA® 3.0 open standard, the AXI Standard protocol has been constantly upgraded and reached the current version of AXI4, which is widely considered as the optimal interconnect technology for FPGA designs. Mainly, AXI buses are responsible for the connections within the embedded system, between the processing unit and other IP blocks. Nevertheless, three separate bus protocols of AXI4 exist, each suitable according to the desirable nature of the connection.

- **AXI4 — The highest performance interface, suited for memory-mapped links: an address is supplied followed by a data burst transfer of up to 256 data words.**
- **AXI4-Lite — A simplified link supporting only memory mapped single transactions (no bursts): Has the benefit of a smaller logic footprint: in this case an address and single data word are transferred.**
- **AXI4-Stream — For high-speed streaming data, supporting burst transfers of unrestricted size. There is no address mechanism; this bus type is best suited to direct data flow between source and destination. A single channel is defined for the transmission of streaming data, modeled after the Write Data Channel in Figure 3.3**

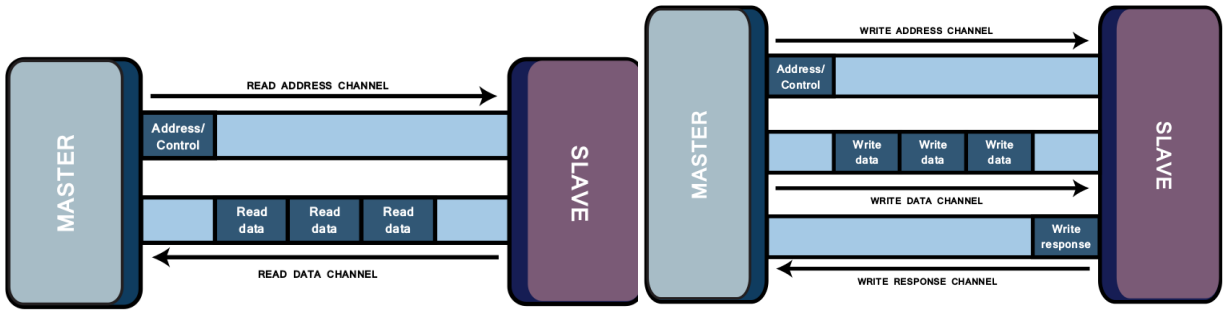


Figure 3.3: AXI4 read and write channel architectures [5].

Concerning the aforementioned term 'memory mapped', if a protocol is memory mapped, an address is specified within the transaction issued by the master (read or write), which corresponds to an address in the system memory space. In the case of AXI4-Lite, which supports a single data transfer per transaction, data are then written to, or read from, the specified address. In detail, an AXI master transfers data to an AXI slave via the AXI interconnect using the write data channel and similarly the AXI slave transmits data via the read data channel to the AXI master. As described in figure 3.3, the write transactions involve an extra write response channel, through which the slave signals completion of the write transaction. Similarly, during a read transaction, both control data and address are transmitted to the slave before a burst of read data is sent to the master [5].

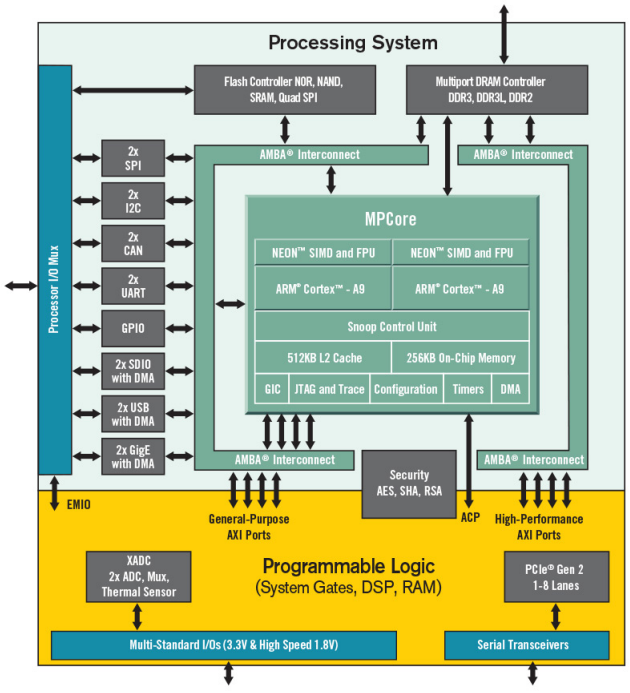


Figure 3.4: Detailed architecture of the Zynq-7000 devices [5].

3.3 Zynq™- 7000 Development Board (ZYBO)

In the context of this thesis the Zybo(ZYnq BOard) Development Board was utilized, manufactured by Digilent using the smallest member of the Xilinx Zynq-7000 family, the Z-7010 [6]. The Z-7010 is based on the Xilinx AP SoC architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic (Fig.3.5). Additionally, six Pmod ports are available to put any design on an easy growth path.

The Zynq 7010 AP SoC offers the following features:

- Reprogrammable logic equivalent to Artix-7 FPGA
    - Look-up Tables(LUTs): 17600
    - Flip Flops: 35200
    - DSP slices: 80
    - 240 KB of fast block RAM
    - Internal clock speeds exceeding 450MHz
- 650MHz dual-core Cortex-A9 processor
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I^2C
- DDR3 memory controller with 8 DMA channels
- MicroSD slot (supports Linux file system)
- OTG USB 2.0 PHY (supports host and device)
- GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs
- Six Pmod ports (1 processor-dedicated, 1 dual analog/digital, 3 high-speed differential, 1 logic-dedicated)

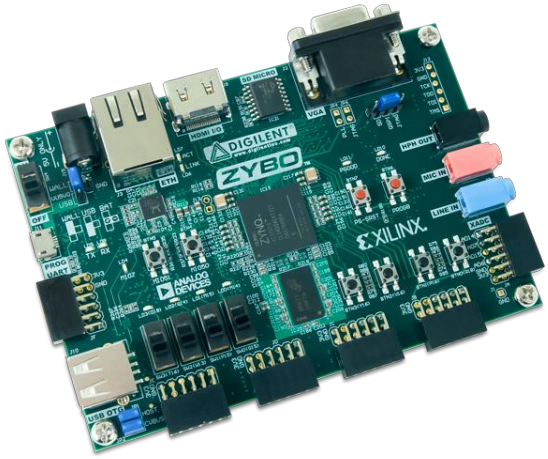


Figure 3.5: ZYBO Zynq-7000 development board [6].

At this point, a brief presentation of the board's Pmod ports and power supplies will be stated, as both aspects are crucial to this thesis.

Pmod Ports

Pmod ports are 2x6, right-angle, 100-mil spaced female connectors that mate with standard 2x6 pin headers. Each 12-pin Pmod port provides two 3.3V VCC signals (pins 6 and 12), two Ground signals (pins 5 and 11), and eight logic signals, as shown in Fig.3.6. The VCC and Ground pins can deliver up to 1A of current, but care must be taken not to exceed any of the power budgets of the on-board regulators or the external power supply. As for the logic signals, each one of them can either serve as an input or as an output, depending on the designer's choice.
The ZYBO has six Pmod ports, some of which behave differently than others. In fact four

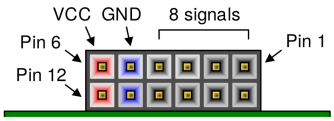
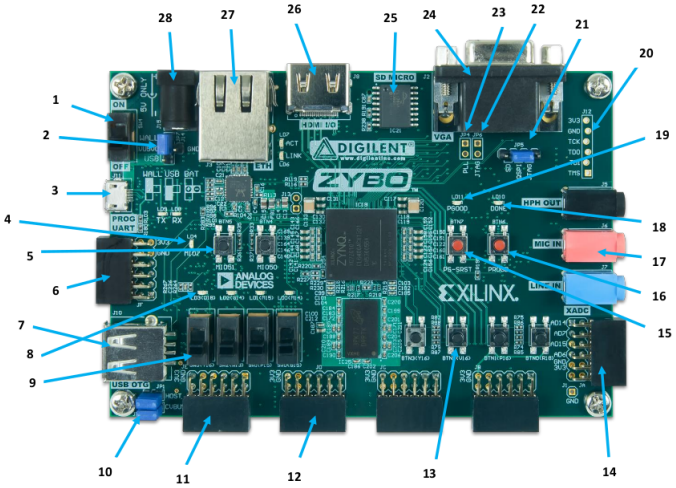


Figure 3.6: Pmod diagram [6].

categories exist: standard, MIO connected, XADC, or high-speed. Figure 3.7 specifies which category each Pmod falls into.
The following paragraphs describe the types of Pmods that were deployed: Standard and High-speed Pmods.
The Standard Pmod port is connected to the PL of the Zynq via 200 Ohm series resistors. The series resistors prevent short circuits that can occur if the user accidentally drives a signal that is supposed to be used as an input. The downside to this added protection is that these resistors can limit the maximum switching speed of the data signals. If the Pmod being used does not require high-speed access, then the standard Pmod port should be used to help prevent damage to the devices.
The High-speed Pmods use the standard Pmod port, but have their data signals routed as impedance matched differential pairs for maximum switching speeds. They have pads for loading resistors for added protection, but the ZYBO ships with these loaded as 0-Ohm shunts. With the series resistors shunted, these Pmods offer no protection against short circuits, but allow for much faster switching speeds.



Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Figure 3.7: Zybo Z7-20 peripherals and components [6].

Power Supplies

The ZYBO can be powered from the Digilent Universal Serial Bus (USB)-JTAG-UART port (3 in figure 3.7), or from an external power supply. Jumper JP7 (1 in figure 3.7) determines which power source is used. There are three valid configurations for this jumper corresponding to the three powering options: USB, wall wart with barrel jack, and battery pack. An external power supply (wall wart) can be used by plugging into to the power jack

(28 in figure 3.7) and setting jumper JP7 to "wall". The supply must use a coax, center-positive 2.1mm internal-diameter plug, and deliver 4.5VDC to 5.5VDC and at least 2.5A of current (i.e., at least 12.5W of power). Power supply voltages above 6VDC might cause permanent damage.

3.4 Laelaps II Robot Adumbration

In this section, the current version of Laelaps II quadruped robot will be presented, along with noteworthy references regarding its leg design, motion planning and electrical system.

Laelaps II is a quadruped robot inspired by the cheetah, which consists of a main body and four three-segment legs (Fig.3.8). Three joints are comprised in each leg, the hip, the knee and the ankle. Although the hip and knee joints are driven via actuators, a torsional spring of high stiffness is located at each ankle joint, which practically reduces the number of links to two, with respective lengths $l_1 = 250mm$ and $l_2 = 350mm$ (Fig.3.12). Concerning the controller's design, an active compliance control framework for dynamic trotting is introduced, which comprehends a low and a high-level controller (Fig.3.9). The low-level segment is responsible for driving the legs along elliptical trajectories through the utilization of a trajectory planning part, an inverse kinematics part and a joint-level active compliance part driving each leg, while the high-level tends to the tuning of the low-level controller's parameters. Namely, in the context of the active compliance part, a PD-Control is implemented, whose output shapes the control torques of each leg's motors [8].

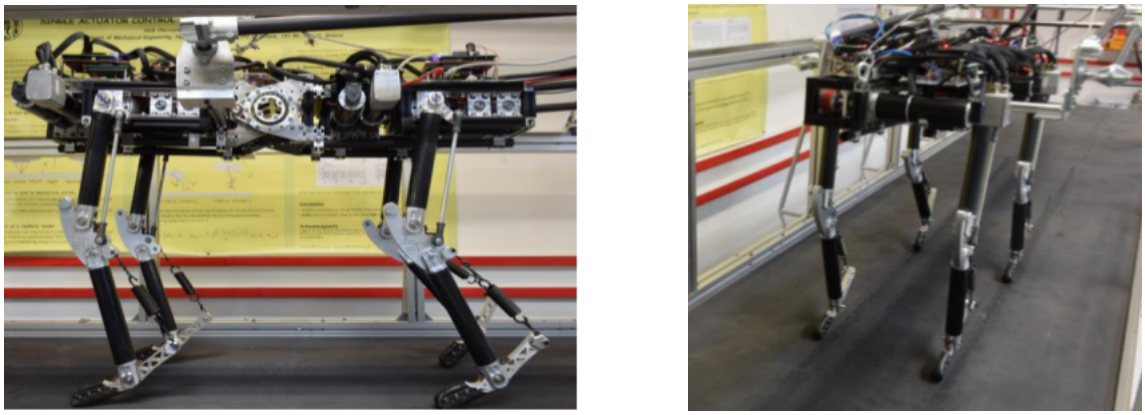


Figure 3.8: Laelaps II [7].

3.4.1 Overview of the EtherCat control architecture

Currently, a decentralized control architecture has been implemented on Laelaps II, based on EtherCat technology, where the motion of each leg is controlled by a slave device. In particular, a network of Micro Controller Units (MCU) has been designed, responsible for motion control, trajectory planning and synchronization of the legs. The actual motion parameters are configured through the EtherCat Master (high-level controller), which communicates with four Delfino MCUs (low-level controllers), resulting in the desired motion gait. In [7] one can find all available details of this implementation, as well as the firmware of

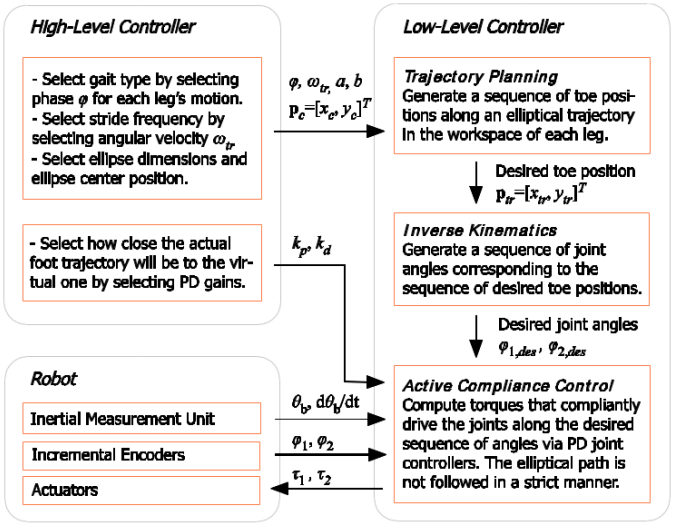


Figure 3.9: Block diagram of the controller for a single leg [8].

the MCUs and the configuration procedure of the EtherCat Master to control all four slaves devices. Moreover, information regarding the mechanical and electrical design of Laelaps II is included.

3.4.2 Electrical System

Although the electrical system of Laelaps II is described in [7], a basic overview of the electrical scheme is given to familiarize the readers with the complete infrastructure of the robot.

The main components of the system include:

- Four EtherCat Control Tower Assembly slaves connected to the motor drivers and the encoders of each leg
- Logic Power supply system with voltage regulators (5V) supplying all EtherCat towers
- Eight motor driver boards (amplifiers) (along with their designated extension boards mounted on top) configured for current control. Each leg consists of two motors, each responsible for the control of the knee and hip motion respectively
- High Power Distribution board which provides high power to all drivers

3.4.3 Laelaps II Encoders

In order to track the position of each joint and thus the position of each leg, it is essential to monitor the position of the respective motor. In this case quadrature encoders are deployed, which are commonly used in a wide variety of applications from robotics to opto-mechanical mice [19]. Optical incremental encoders contain a lensed LED source, an integrated circuit with detectors and output circuitry, and a patterned codewheel which rotates between the emitter and the detector IC. The codewheel follows the motion of the motor and through an optical switch, like a photodiode, electrical pulses are generated in accordance to the

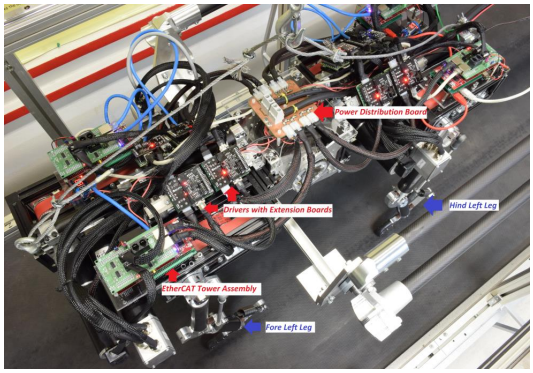


Figure 3.10: Electrical System of Laelaps II [7].

motion in question. Most encoders' outputs are two square waves A and B in quadrature (90° out of phase), while some also have a third channel index output in addition to the two channel quadrature. This index output is a 90 electrical degree, high true index pulse which is generated once for each full rotation of the codewheel.

Currently the HEDL-5640 line drivers are mounted on the Laelaps II legs, which encapsulate the HEDS-5640 (Fig.3.11) three channel optical incremental encoder [9].

Figure 3.11: HEDS-5640 Encoder of Avago [9].

3.4.4 Laelaps II gearheads and motors

Both knee and hip joints of Laelaps II are driven via motors, which are mounted at each joint. However, the motors themselves are not sufficient enough for this case. In fact, every application has power requirements and specific values of speed and torque. With a load demanding high torque at low speed, use of a large motor capable of developing the torque would not be economical, and system efficiency would be low. In such cases, a better solution is to introduce some gearing between the motor and the load. Gearing adapts the motor to the load, be it for speed, torque or inertia, and thus rendering the motor/gearbox assembly more efficient, as well as an economical solution [20].
On Laelaps II case, the gearing aims to downscale the speed by a factor equal to the reduction ratio, while upscaling the torque proportionally. Moreover, a pulley is mounted on both kinds of joints-motors, with a specific gear ration (48/26), in order to further increase their output torque and reduce their rotational speed.

According to [7], different combinations of motors and gearheads are employed to drive the knee and hip joints of Laelaps-II. Thus, the following total reduction gear ratios emerge (pulley and gearhead combined):

- Hip Motor: $\frac{1029}{13} = 79.1539$
- Knee Motor: 98

3.4.5 Leg design and motion planning

All four legs of Laelaps II are based on the leg design shown in Fig.3.12. According to this model, the respective motion planning equations can be derived.

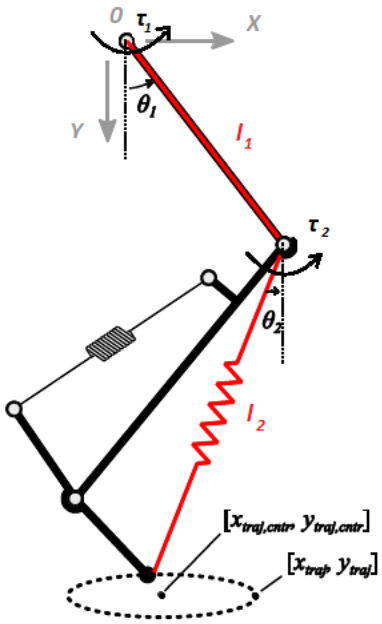


Figure 3.12: Leg model [7].

In order to achieve the desirable gait, an appropriate trajectory is selected for the tow of the leg. Since the toe also happens to be the end of the robotic leg, the robotics term "end-effector" can be applied. The position of the end-effector can be computed using the forward kinematics equations. In particular, the coordinates (x_E, y_E) of the end-effector can be computed from specified values for the joint parameters ($\theta 1$ and $\theta 2$ in this case). Parameters l_1 and l_2 refer to the lengths of the leg.

Forward Kinematics

$$\begin{aligned} x_E &= l_1 sin(\theta_1) + l_2 sin(\theta_2) \\ y_E &= l_1 cos(\theta_1) + l_2 cos(\theta_2) \end{aligned} \tag{3.1}$$

Conversely, the inverse kinematics equations determine the joint parameters that provide a desired position for the robot's end-effector. The first equation is derived via the cosines law.

Inverse Kinematics

$$\begin{gathered} \phi = \theta_2 - \theta_1 \\ x_E^2 + y_E^2 = l_1^2 + l_2^2 - 2l_1 l_2 cos(\pi - \phi) = l_1^2 + l_2^2 + 2l_1 l_2 cos(\phi) \\ cos(\phi) = \frac{x_E^2 + y_E^2 - (l_1^2 + l_2^2)}{2l_1 l_2} \\ sin(\phi) = -\sqrt{1 - cos^2(\phi)} \\ \phi = atan2(sin(\phi), cos(\phi)) \end{gathered} \tag{3.2}$$

$$\begin{gathered} \boldsymbol{\theta_2} = \frac{\boldsymbol{\pi}}{\mathbf{2}} - \mathbf{atan2}(\mathbf{y_E}, \mathbf{x_E}) + \mathbf{atan2}(\mathbf{l_1 sin}(\boldsymbol{\phi}), \mathbf{l_2} + \mathbf{l_1 cos}(\boldsymbol{\phi})) \\ \boldsymbol{\theta_1} = \boldsymbol{\theta_2} - \mathbf{atan2}(\mathbf{sin}(\boldsymbol{\phi}), \mathbf{cos}(\boldsymbol{\phi})) \end{gathered} \tag{3.3}$$

Eqs.(3.3) emerge, which have only the position of the end-effector as an argument. For the motion control of each leg, Eq.3.4 are deployed, which delineate semi-elliptical trajectories. The elliptical shape is defined with reference to point 0 (hip axis) (marked in Fig.3.12), while its parameters are explained in Eq 2.4. To model the impedance of the treadmill's floor, a flattening parameter (*param*) has been added on the y axis amplitude (b), altering the shape of the elliptical trajectory. Depending on whether ($angle \bmod 2\pi$) $< \pi$ or not, b^* is either equal with $param * b$ or b respectively.

$$\begin{gathered} angle = \omega_{traj} t + \phi \\ x_{traj} = x_{traj,cntr} + acos(angle) \\ y_{traj} = y_{traj,cntr} + b^* sin(angle), \end{gathered} \tag{3.4}$$

where

a	is the semi-major axis's length of the ellipse
b	is the semi-minor axis's length of the ellipse
ω_{traj}	is the angular velocity of the elliptical motion
ϕ	is the initial phase of the elliptical motion
t	is the time
$x_{traj,cntr}$	is the x coordinate of the ellipse's center
$y_{traj,cntr}$	is the y coordinate of the ellipse's center
b^*	is the flattened semi-minor axis's length
$param$	is the flattening parameter of the y axis amplitude (b)

Chapter 4

System Description

In contrast with the current configuration, which utilizes micro-controllers and a CPU, the proposed implementation utilizes a SoC FPGA, in order to acquire information regarding the motors, as well as carry out their control. In particular, the management of the encoders and the driving of the motors are realized in the FPGA, while both the high and low-level controllers (Fig.3.9) are actualized on the ARM processor.
The whole procedure is divided into four phases, each equally precious with the rest (Fig.4.1). Initially, the encoders' outputs are filtered and imported in the FPGA (Phase 1). Subsequently, the FPGA extracts the motors' position and velocity, exploiting the information obtained from the encoders, and transmits them to the ARM processor along with certain auxiliary signals, via the AXI4-Lite bus protocol (Phase 2). Next, the overall controller is executed on the ARM processor via the XSDK platform and the control torques of each leg's motors are generated and forwarded back to the FPGA, again via the AXI4-Lite bus protocol (Phase 3). Finally, appropriate PWM signals are produced in the FPGA, capable of driving the legs' motors as instructed by the controller (Phase 4).

In this chapter, the phases realized on Hardware will be analyzed, i.e., Phases 1,2 and 4. All units presented below were designed and produced using the Vivado 2017.4 Design Suite by Xilinx.

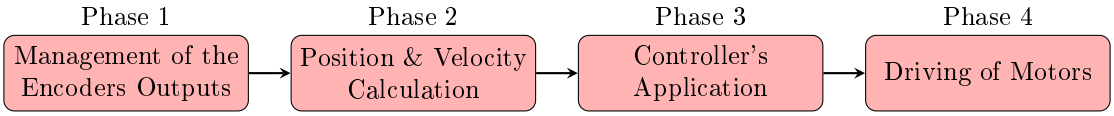


Figure 4.1: Processing Flow of the SoC FPGA Implementation

4.1 Management of the Encoders Output

Concerning "Phase 1", in order to later specify the position and consequently the velocity of a motor, a quadrature encoder is exploited, whose nature will be explained in the Section 4.2. An important characteristic that should be noted in this section, is that they generate three digital signals, which have to be processed by the proposed system. As mentioned before (Section 3.3), the Zybo Board provides the Pmod ports as I/O connectors, through which

digital signals can be accessed. Hence, the outputs of each utilized encoder were connected to the Pmod ports of the Zybo Board. At this point, it should be noted that if the encoders are supplied with 3.3V, they can be connected directly to the Pmods (Fig.4.2). Otherwise, a level shifter should intervene, as described in Section 6.1.

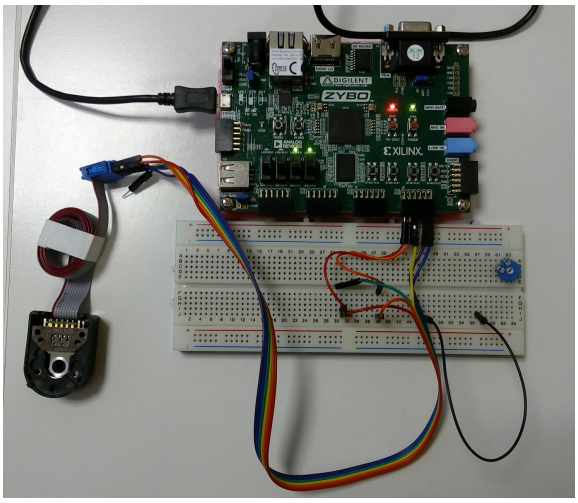


Figure 4.2: Preliminary experimental setup.

Before the process of the inputs, it was decided to filter them, as advised by [19]. In particular, a fast clock of the FPGA (*clk*) is used to sample the inputs in order to determine whether they are stable or not. Specifically, four samples of the input *QEX* are taken and if it is stable, it is transfered to the next level of computations as output *fout* (Fig.4.3).

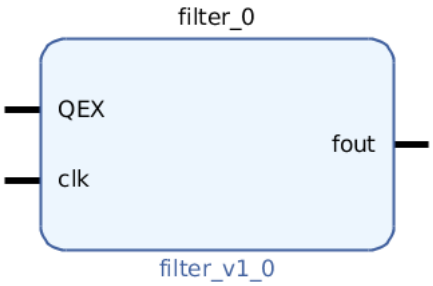
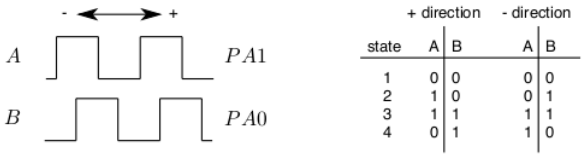


Figure 4.3: Filter block realized in the FPGA.

4.2 Quadrature Encoder Interface

As mentioned above (Section 3.4.3), encoders provide two or three output signals (A, B, I), which can be exploited to figure the position, as well as the direction of the motor's rotation. The determination of these data can be achieved via a Quadrature Encoder Interface (QEI). In the context of this thesis, the QEI was designed in VHDL and it was successfully tested using the Zybo Development Board. Although the proposed system ultimately does not utilize the Index signal, the architecture described below supports its handling.
In this section, the *encoder* part of the QEI system will be analyzed, responsible for the determination of a motor's position.

First of all, when the codewheel rotates in the counter-clockwise direction (as viewed from the encoder end of the motor), channel A will lead channel B. If the codewheel rotates in the clockwise direction, channel B will lead channel A. Next, in Fig.4.4 it can be observed that for each direction only one channel switches between two consecutive states. Additionally, if the signals A of each state (A_N) are compared with the signals B of each previous state (B_{N-1}), it can be derived that $A_N \oplus B_{N-1} = 1$ when moving to the "+" direction, and $A_N \oplus B_{N-1} = 0$ when moving to the "-" direction, where $\oplus$ is the exclusive-or logical operation (XOR, or 1-bit ADD).



	+ direction		- direction	
state	A	B	A	B
1	0	0	0	0
2	1	0	0	1
3	1	1	1	1
4	0	1	1	0

Figure 4.4: Encoder waveform and state transition table [10].

Based on that, when the signal I is equal to '0' and the direction is "+", a counter increases (+1) and when the direction is "-", the counter decreases (-1). If signal I is equal to '1' or the counter overflows, then the counter resets to zero.

This counter calculates the encoder's counts, which subsequently determine the motor's angle. The encoders in use (Fig.3.11) have a standard resolution of 2000 counts per revolution, meaning that 2000 counts correspond to 360°. However, if the total reduction ratios of the gearheads (Section 3.4.4) are taken into consideration, the counter should be able to count up to $2000 * 98 = 196000$ counts (98 being the highest value of the two reduction ratios). Since 196000 requires 18 bits to be represented in binary, the counter's resolution is currently equal to 18 bits, meaning that the maximum number of counts is equal to 262143. Of course, due to the fact that two gear ratios exist, the generic variables *ratio_numerator* and *ratio_divisor* were introduced and the maximum value of the counter is reconfigured according to their quotient. Likewise, given that a future modification of the motors or their gearheads is possible, the generic variable *bits* was also introduced for convenient revision. Generic variables provide static information to blocks similarly to constants with the exception that generics can be supplied externally from its environment, either in a component instantiation statement or in a configuration specification [§].
In conclusion, the *encoder* block (Fig.4.5) receives as inputs the signals A,B,I of the encoder and the master clock signal for synchronization purposes (all signals of 1-bit each). After the process of the inputs, the *encoder* block outputs the direction of motion (signal of 1-bit) and the counts of the encoder (signal of *bits* number of bits).

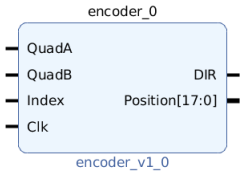


Figure 4.5: *Encoder* Block.

[§]http://vhdl.renerta.com/mobile/source/vhd00034.htm

4.3 Velocity Estimation

Apart from the position of each joint and hence the respective motor, in order to implement the PD-Controller, it is necessary to have access to the velocity of the above units. In this section, the final version of the velocity estimation system is explicated.

Initially, the system that is currently used in the quadruped was studied. A detailed breakdown of the system is quoted below, as described in [7].

Due to the fact that the encountered rotational velocities are relatively low, a custom function was created to calculate the velocity of both joints using the eQEP Edge Capture Unit – Low Speed Calculation feature (refer to TMS320x2833x, 2823x Enhanced Quadrature Encoder Pulse (eQEP) Module [21]). This approximation is based on Eq.4.1, where on every unit position event (X reaches the predefined number of quadrature edges [UPPS]) the capture timer [QCTMR] value is latched into the capture period register [QCPRD] and then [QCTMR] is reset. Then, the velocity is converted from [counts/time_register] to [rad/s].

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta t} \tag{4.1}$$

Regarding the aforementioned constant UPPS, after reading thoroughly [21] it was concluded that UPPS = 4, meaning that an event happens every four quadrature edges and the velocity is calculated.

The VHDL corresponding algorithm proposed in the context of this thesis, although similar to the one presented in [7], presents a few nuances.
Firstly, due to the reason that the system filters and synchronizes the encoder's inputs with its clock, it was considered wise to keep track of the changes that occur on the number of the encoder's counts and calculate the velocity every four changes instead of every four quadrature edges.
Secondly, the current implementation, described in [21], classifies a time measurement (Δt) between unit position events as valid, if the following conditions are met:

- No more than 65535 counts have occurred between unit position events.
- No direction change between unit position events.

Concerning the first condition, it is included to prevent overflow errors in case that the counts register exceeds its maximum possible number ($2^6 - 1 = 65535$). However, in the proposed implementation, even if the counts register overflows it will not affect the velocity estimation, since the changes of that register are monitored and not its value. The second condition is preserved in the proposed system as well and specifically the velocity resets to zero in the case of a direction change.

Mathematical analysis of the Velocity Estimation

Initially, the FPGA calculates the position of the encoder in counts and time in clock cycles. Therefore, the first goal is to express 1 $\frac{count}{clockcycle}$ to $\frac{radians}{s}$. As stated in Section 4.2, one revolution is equal to 2000 counts of the encoder. Due to the reduction ratio of the gearhead

and the pulley (Section 3.4.4), 1 revolution of the gearhead is equal to $gr * 2000$ counts, where gr is the total reduction ratio of the respective joint. Therefore:

$$360^\circ = 2000 * gr \ \ counts \tag{4.2}$$

or

$$1 \ \ count = 360^\circ/(2000 * gr) \tag{4.3}$$

Given the equation 4.4, we can express 1 count in radians.

$$1^\circ = 0.0174532925 \ \ radians \tag{4.4}$$

$$1 \ \ count = 6.2831853/(2000 * gr) \ \ radians \tag{4.5}$$

Concerning the time, the $(f_{FPGA-Hz})$ variable is introduced that describes the operating frequency of the FPGA. However the outputs of the system are updated with a frequency of $(\frac{f_{FPGA-Hz}}{4})$. As mentioned before (Section 4), the fast clock of the $(f_{FPGA-Hz})$ frequency is used to sample the inputs in order to determine whether they are stable or not. Therefore, 1 clock cycle is equal to $\frac{4}{f_{FPGA-Hz}}s$.

$$1 \ \ \frac{count}{clock \ \ cycle} = \frac{6.2831853}{2000 * gr * \frac{4}{f_{FPGA-Hz}}} = \frac{6.2831853 * f_{FPGA-Hz}}{8000 * gr} \ \ \frac{rad}{s} \tag{4.6}$$

The calculation of the velocity occurs once every 4 counts and hence the final equation for the estimation of the velocity is:

$$u(k) \ \ = \frac{4 \ \ counts}{\Delta t \ \ clock \ \ cycles} = \frac{0.00314159265 * f_{FPGA-Hz}}{\Delta t * gr} \ \ \frac{rad}{s} \tag{4.7}$$

Lastly, due to the fact that the VHDL code utilizes integer division, it was opted to multiply the divider by 10000, instead of using a complex and demanding in resources float division, in order to maintain accuracy of four decimal places. Afterwards, the result will be divided by 10000 in the ARM Processor, before the execution of the control loop feedback mechanism.

$$u(k) \ \ = \frac{31.4159265 * f_{FPGA-Hz}}{\Delta t * gr} \ \ \frac{rad}{s} \tag{4.8}$$

Implementation of the Velocity Estimation on the FPGA (VHDL)

Due to the aforementioned changeable variables, the velocity equation was expressed using generics that allow the designer to parametrize the entity during the component instantiation. Specifically, the following generics are utilized:

- $ratio\_numerator$
- $ratio\_divisor$
- sys_clk
- $bits$

where $gr = \frac{ratio\_numerator}{ratio\_divisor}$, sys_clk is the operating FPGA frequency measured in Hz and $bits$ is the number of bits dedicated for the reduction ratio.

Given that the FPGA frequency can also be measured in MHz, the more handy variable f_FPGA could be introduced, which represents the FPGA operating frequency in MHz.

$$u(k) = \frac{31.4159265 * f\_FPGA * 10^6}{\Delta t * \frac{ratio\_numerator}{ratio\_divisor}} \ \frac{rad}{s} \tag{4.9}$$

or

$$u(k) = \frac{31415926.5 * f\_FPGA}{\Delta t * \frac{ratio\_numerator}{ratio\_divisor}} \ \frac{rad}{s} \tag{4.10}$$

At this point, the constant variable $vel\_constant = 31415927$ is defined (rounding of the 31415926.5 value), since it remains fixed.

$$u(k) = \frac{vel\_constant * f\_FPGA}{\Delta t * \frac{ratio\_numerator}{ratio\_divisor}} \ \frac{rad}{s} \tag{4.11}$$

In conclusion, the estimated velocity is estimated as the integer quotient of the following division:

$$\mathbf{u(k)} = \frac{\mathbf{vel\_numerator}}{\mathbf{\Delta t}} \ \frac{\mathbf{rad}}{\mathbf{s}}, \tag{4.12}$$

where

$$vel\_numerator = temp1 * f\_FPGA \tag{4.13}$$

$$temp1 = vel\_constant * \frac{ratio\_divisor}{ratio\_numerator} \tag{4.14}$$

At this point, it should be mentioned that the minimum possible value of the velocity estimation algorithm is equal to 0.001 rad/s, which means that $u(k) = 10000 * 0.001 = 10$ [†] and hence this instance occurs when $\Delta t = \frac{vel\_numerator}{10}$. As a result, in the event that the Δt variable reaches the value $zero\_constant = \frac{vel\_numerator}{10}$, then the velocity is automatically set to zero.

[†]The actual velocity is multiplied by 10000 in the FPGA for precision reasons.

Altogether, the velocity estimation block (Fig.4.6) consists of three inputs : the overall system's clock signal for synchronization purposes (signal of 1-bit) and the direction of motion (signal of 1-bit) along with the counts of the encoder (signal of *bits* number of bits), as produced from the *Encoder* block (Fig.4.5). After the completion of their process, the velocity's value (signal of 32-bits) and its sign (signal of 1-bit) are exported from the block.

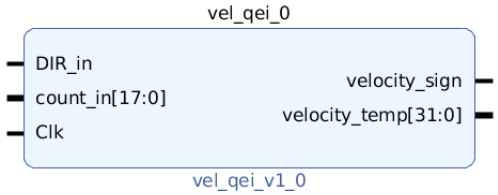


Figure 4.6: *Velocity Estimation* Block.

Current configuration for the legs of Laelaps-II

Taking into account the different combinations of motors and gearheads that are employed to drive the knee and hip joints of Laelaps-II (Section 3.4.4), as well as the current FPGA operating frequency (approximately 91MHz), the following equations for the Hip and Knee Motors' velocities emerge:

$$Hip\ \ Motor: \quad u(k) \ = \frac{31.4159265 * 91 * 10^6}{\Delta t * \frac{1029}{13}} = \frac{36117629.7857}{\Delta t} \ \frac{rad}{s} \tag{4.15}$$

$$Knee\ \ Motor: \quad u(k) \ = \frac{31.4159265 * 91 * 10^6}{\Delta t * 98} = \frac{29171931.75}{\Delta t} \ \frac{rad}{s} \tag{4.16}$$

Due to the fact that integer variables are used to represent the above values, we cannot maintain the same precision. The corresponding equations in VHDL are presented below:

$$Hip\ \ Motor: \quad u(k) \ = \frac{36117627}{\Delta t} \ \frac{rad}{s} \tag{4.17}$$

$$Knee\ \ Motor: \quad u(k) \ = \frac{29171870}{\Delta t} \ \frac{rad}{s} \tag{4.18}$$

In order to verify the validity of the velocity estimation algorithm, an experiment was conducted. In particular, one of the motors from the experimental platform (Fig.5.2) was manually handled, while its position and hence velocity were monitored via the system. As observed from the figure 4.7, the FPGA implementation is almost identical with the velocity estimation executed on the ARM processor (biggest difference equal to 0.1470). In fact, it is the author's belief that the FPGA version is more accurate, due to the fact that the ARM estimation sometimes returns significantly large values (spikes), probably due to a division of extremely small values, while the FPGA delivered reasonable results. This proves that the velocity estimation unit is valid.

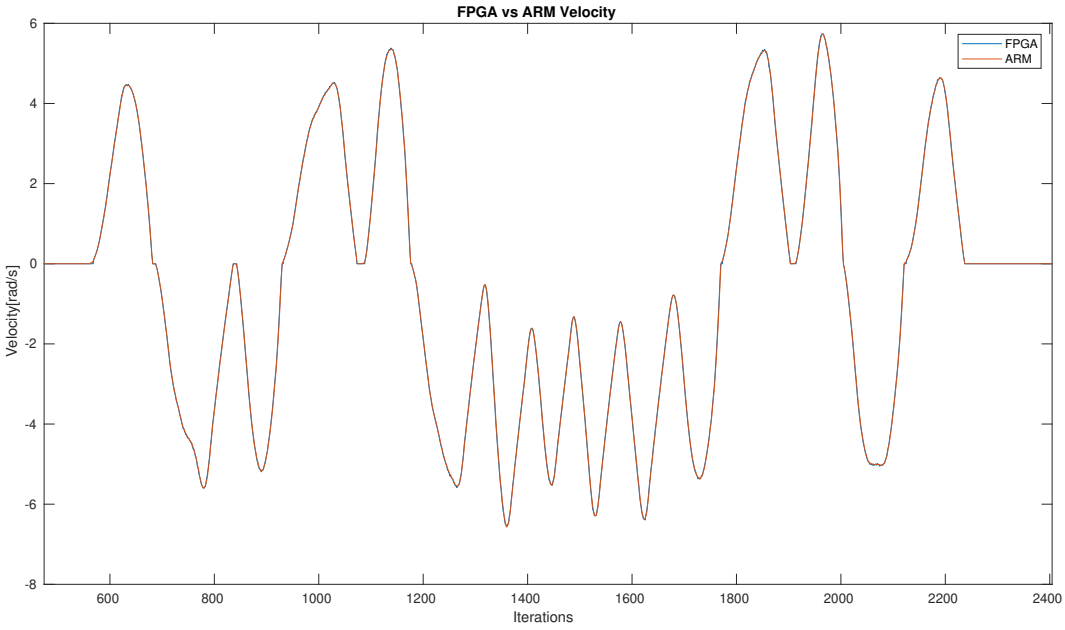


Figure 4.7: FPGA-ARM comparison of velocity estimation.

4.4 Combinatorial QEI Block

At this point, the *qei* block that conflates the *filter* block with the *encoder* and the *vel_ qei* blocks needs to be introduced. Furthermore, the *qei* block is responsible for the interconnection and synchronization of the embedded units.

Namely, the *qei* block (Fig.4.8) receives as inputs the signals A,B,I of the encoder and the master clock signal for synchronization purposes (all signals of 1-bit each). Following the fitting redirection and process of the inputs, the *qei* block outputs the counts of the encoder (signal of *bits* number of bits), the velocity of the respective motor (signal of 32-bits) and its sign (signal of 1-bit). Naturally, the same generic variables *ratio_ numerator*, *ratio_ divisor*, *sys_ clk* and *bits* are included in this block and according to their values, the equivalent generic variables of the subsidiary blocks are defined.

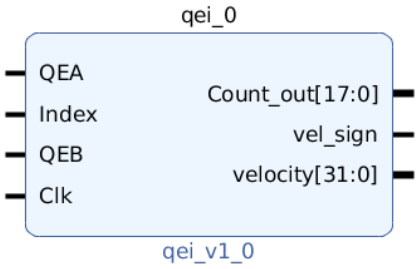


Figure 4.8: *Qei* Block in VHDL.

Apart from the aforementioned qualities, the *qei* block entails the creation of the slower clock, which is four times slower than the FPGA's operating frequency, i.e, the fast clock that contributes to the sampling of the encoders' outputs (as mentioned in Section 4.1).

4.5 Pulse Width Modulation

A key factor to the control of the motors is the creation of Pulse Width Modulation (PWM) signals of a specific frequency and duty cycle, which allow the control of the power supplied to electrical devices, such as motors. According to the duty cycle of the PWM signal, the torque of the motor is controlled, since a current control architecture is implemented to Laelaps II motor drivers.

Based on the [22] design, a PWM component was programmed in VHDL, which was successfully tested on the Zybo Development Board (Fig.4.9). The following generic variables are contained in the PWM block: the FPGA's clock frequency (*sys_clk*), the desired PWM frequency (*pwm_frequency*), the defined resolution of the duty cycle in bits (*bits_resolution*) and the desired number of phases (*phases*). The number of phases defines how many PWM signals are generated. If set to multiple phases, the component generates one PWM signal for each phase, evenly spaced. Although the proposed system ultimately does not utilize this variable and only one phase is generated, the architecture described below supports the handling of multiple phases.

Basically, the system clock divided by the desired PWM frequency equals the number of master clock pulses in one PWM period, which is defined via a counter that increments on each system clock and resets once it reaches the end of its period. On the other hand, the duty cycle determines the fraction of the period, during which the PWM signal will be active (on). Thus, a 24% duty cycle will result in a signal being on for the 24% of the period and off during the rest 76% of the period.

In total, the component's inputs are the master clock signal for synchronization purposes (*clk* 1-bit), a reset signal that asynchronously sets the duty cycle to zero (*reset* 1-bit), an enable signal that latches in new duty cycle values and adjusts the PWM outputs at the center of their pulses (*ena* 1-bit) and the desired duty cycle value (*duty bits_resolution*-bits). The outputs include the PWM signal (*pwm_out* 1-bit) and its complementary (*pwm_n_out* 1-bit), which is not currently utilized.

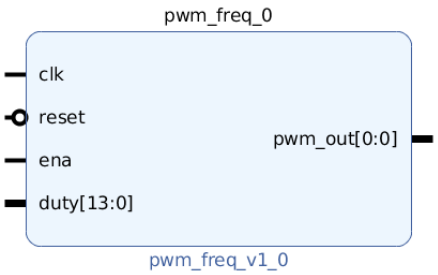


Figure 4.9: PWM Block in VHDL.

The *duty* input can represent any integer number from 0 to $2^{bits\_resolution} - 1$. In figure 4.10, the *bits_resolution* signal was equal to 8, meaning that the maximum duty cycle value was 255. In that case, if a duty cycle of 30% is intended, then the duty input signal should be equal to 0.3×255=77, as we round the result to the nearest integer. Thus, when the *duty* signal has the value "128", it corresponds to a duty cycle of 128/255=0.50%.

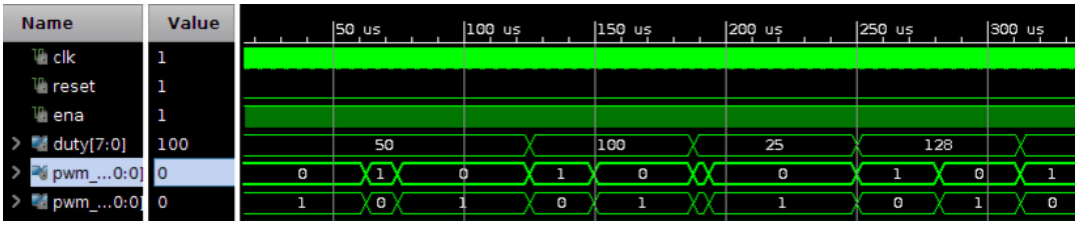
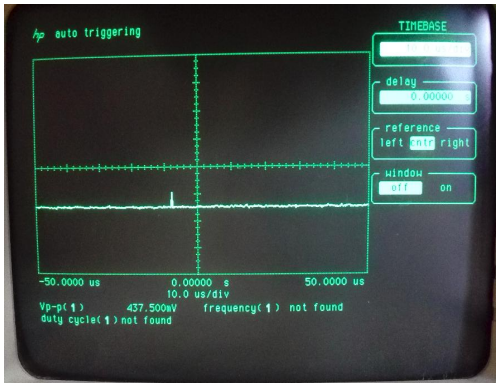
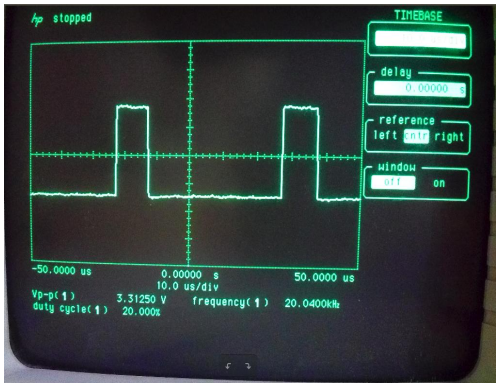


Figure 4.10: Simple behavioral simulation of the component.

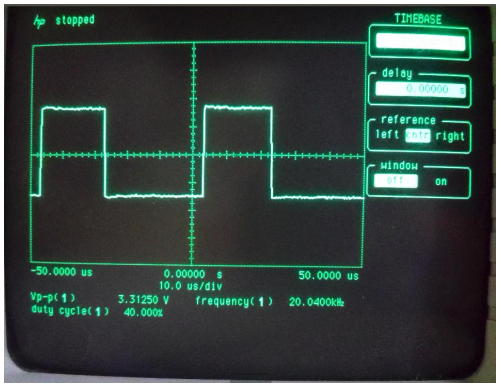
The PWM frequency is set to 20kHz, as specified in the employed motors' datasheet (range 10-25 kHz). For verification reasons, the outputs of the block were tested via an oscilloscope, which confirmed the rectitude of the design (Fig.4.11).



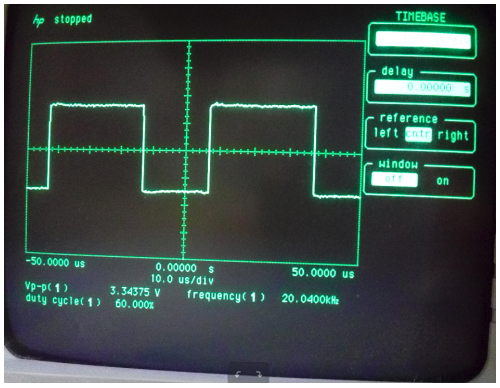
(a) Duty Cycle=0%



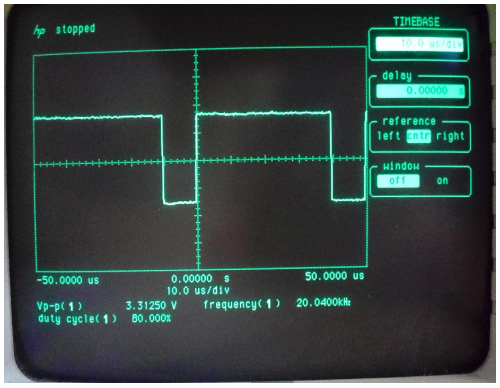
(b) Duty Cycle=20%



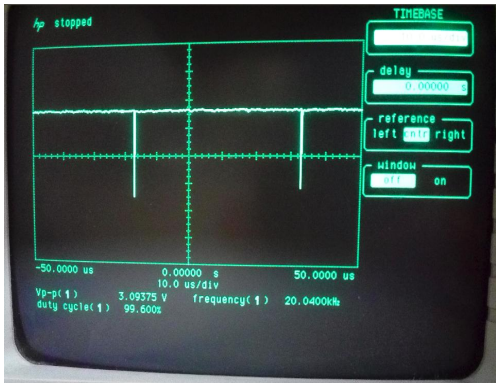
(c) Duty Cycle=40%



(d) Duty Cycle=60%



(e) Duty Cycle=80%



(f) Duty Cycle=100%

Figure 4.11: Duty Cycles

4.6 Hardware/Software Codesign

The decision to realize the aforementioned components in hardware was made, due to the fact that those units are not planned to be modified in the near future. On the other hand, the controller segment (both high and low-level) has to be easily adjustable, since it involves elements that are regularly updated or alternated, for instance the trajectory planning and the inverse kinematics. As a consequence, a Hardware/Software Codesign was implemented, where the controller was realized in software, as it will be explained thoroughly in the next chapter (5.3). Undoubtedly, the pith of the Hardware/Software Codesign, is the custom AXI4 IP that integrates the designed hardware along with the AXI4-Lite protocol (explained here 3.2), and hence establishes the communication between the Hardware and the Software.

Only the custom IP for one leg will be discussed, as the equivalent IPs for more legs constitute extensions of this version. Specifically, a pair of the *qei* and *pwm* blocks is dedicated for the handling of one joint-motor. Therefore, since one leg consists of two joints, an equal number of pairs are utilized as components for the *laelaps_one_leg_qei_vel_pwm* IP. Naturally, the IP also contains the same generic variables as its components and in fact it is responsible for defining their values, as illustrated in Table 4.1.

Variable	Value	Variable	Value	Variable	Value
sys_clk	90909088 Hz	*phases*	1	*hip_ratio_divisor*	13
pwm_frequency	20 kHz	*bits*	18	*knee_ratio_numerator*	98
bits_resolution	14	*hip_ratio_numerator*	1029	*knee_ratio_divisor*	1

Table 4.1: Generic Variables of *laelaps_one_leg_qei_vel_pwm* IP

As illustrated in Fig.4.12, the IP features seven inputs, even though the *S00_AXI* input encapsulates more than one signals. The *s00_axi_aclk* port is connected to the system's clock and synchronizes all computations that occur inside the IP, while the *s00_axi_aresetn* is a negative-logic reset signal. The *jb_p0*, *jb_n0*, *jb_p1*, *jb_p1* inputs correspond to the identifying names of the Pmod ports (Section 3.3), as designated in the *Zybo-Master.xdc* constraints file included in the overall Vivado project. For instance, the *jb_p0* and *jb_p1* represents the JB1:T20 and JB1:U20 Pmod female connectors respectively (Fig.4.13). Lastly, the *S00_AXI* is the AXI4-Lite bus through which the processor communicates with the FPGA. The prospective reader should keep in mind that the *S00_AXI*, *s00_axi_aclk*, *s00_axi_aresetn* ports exist by default and were not added by the designer.
Practically, the ports connected to the Pmods transfer the encoders' outputs to the IP and hence the *qei* blocks. After their filtering and processing, the respective positions and velocities are calculated, which are being transmitted promptly to the processor via the AXI4-Lite Interface.
As previously mentioned, the communication between the FPGA and the processor is achieved through the AXI4-Lite protocol (explained here 3.2). The design was based on the instructions mentioned in [23] and [5, pp. 91-104]. Basically, the AXI4-Lite Interface consists of a user-defined number of slave registers that can be used for communication from the FPGA to the processor and vice versa. For the one-leg IP six 32-bit slave registers were employed, four for the data transmission from hardware to software and two for the opposite.

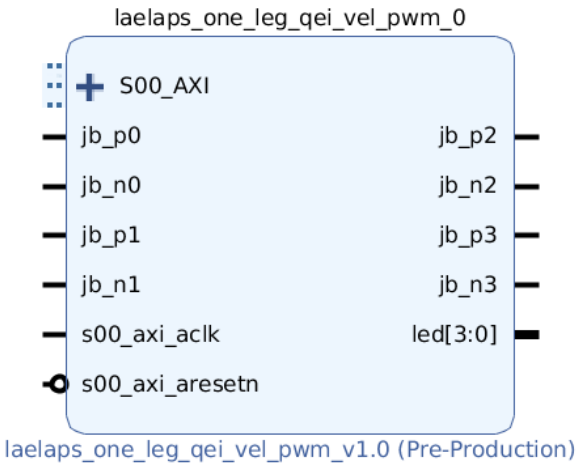


Figure 4.12: Custom AXI4-IP for one leg.

Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Figure 4.13: Pmod pinout [6].

Once the position and velocity (value and sign) of each joint is computed, they are redirected to the corresponding slave registers and through them they reach the processor. On the other side, the processor can view this information by accessing a specific memory address. This address is visible on the Address Map of the processor (Fig.4.14) in the XSDK and can also be referred to by using the pointer *XPAR_LAELAPS_ONE_LEG_QEI_VEL_PWM_0_S00_AXI_BASEADDR*. In fact, that address is linked with the first slave register, while the next address block relates with the second slave register and so on. Likewise, the processor can send data to the FPGA, by accessing the memory address designated to the desired slave register.
Once the software segment has concluded its procedure (detailed in 5.3), the computed duty cycle values along with the desirable direction signals for each motor are transfered via the AXI4-Lite Interface back to the FPGA. The designated slave registers are linked to an equal number of *pwm* blocks, which formulate suitable PWM signals, and in addition with the directions signals they comprise the *laelaps_one_leg_qei_vel_pwm* IP's outputs. Similarly with the inputs, the outputs *jb_p2*, *jb_n2*, *jb_p3*, *jb_n3* are named after the Pmod connectors, with which they will be connected. Additionally, the two PWM and

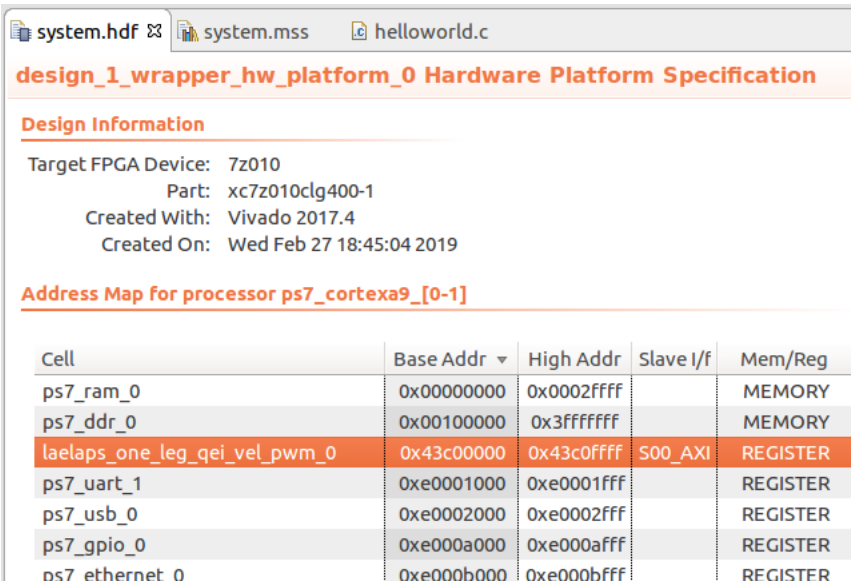


Figure 4.14: Address Map for the ARM Cortex-A9 processor.

direction signals are exported to the four-bit *led* signal, in order to later be connected to the four leds of the Zybo Board.

Eventually, the whole hardware system is summarized in Fig.4.15. The custom IP has been connected with the AXI Interconnect, which is also connected with the processor (*ZYNQ7 Processing System*). The clock and reset signals of the block have been linked with the appropriate blocks that handle their operation, while the rest of the inputs and outputs have been connected with the external ports interlinked with the suitable Pmod ports and leds, as defined in the *Zybo-Master.xdc* constraints file. It is important to point out that the *Processor System Reset*, *AXI Interconnect* and *ZYNQ7 Processing System* blocks are automatically created and hence the designer is solely occupied with the development of the custom IP.

Since the complete hardware implementation for one leg was presented, the procedure followed for the rest of the legs can be explained. In particular, the same components were utilized and additional custom IP blocks were composed, which involved the appropriate number of *qei*, *pwm* blocks and slave registers. Similarly, the communication with the processor was fulfilled via the AXI4-Lite Interface, while the extra input and output ports were connected again with the suitable external ports interlinked with the Pmod ports. Moreover, the four-bit *sw* port has been added to the IPs of more than one legs, which connects with the Zybo Board's four switches. Depending on their position, the correspondingly PWM and direction signals are transmitted to the Board's leds for supervision reasons. If the first switch is on and the rest are off, the data from the first leg are shown, while the same logic applies for the rest. The block designs for two (Fig.4.16) and four legs (Fig.4.17) are illustrated below.

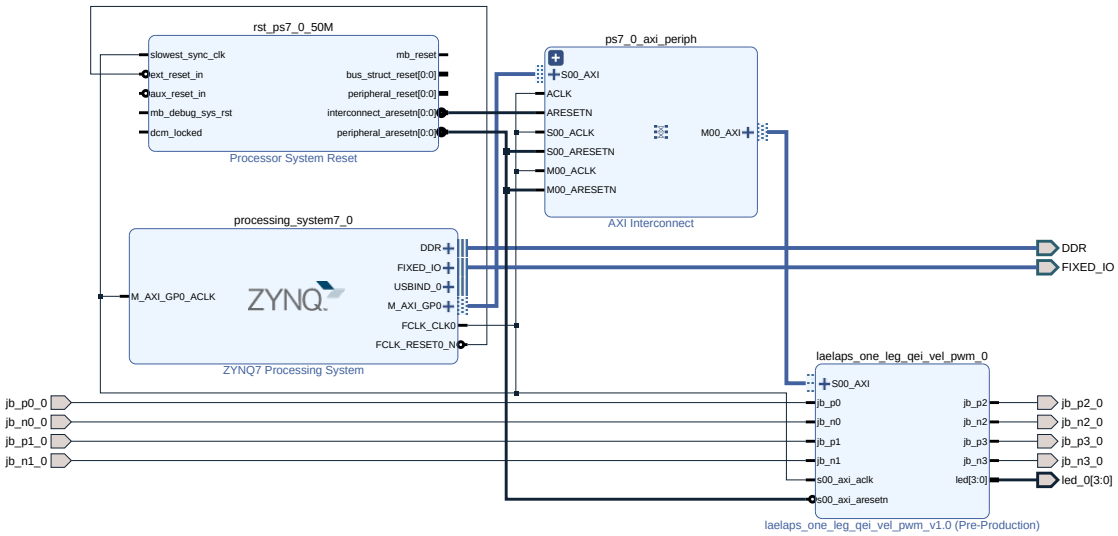


Figure 4.15: Block design of the hardware system for one leg of Laelaps II.

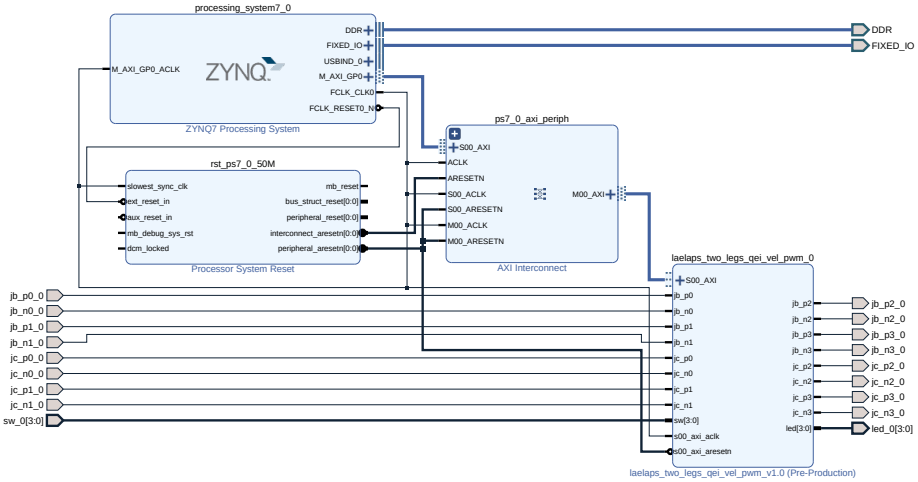


Figure 4.16: Block design of the hardware system for two legs of Laelaps II.

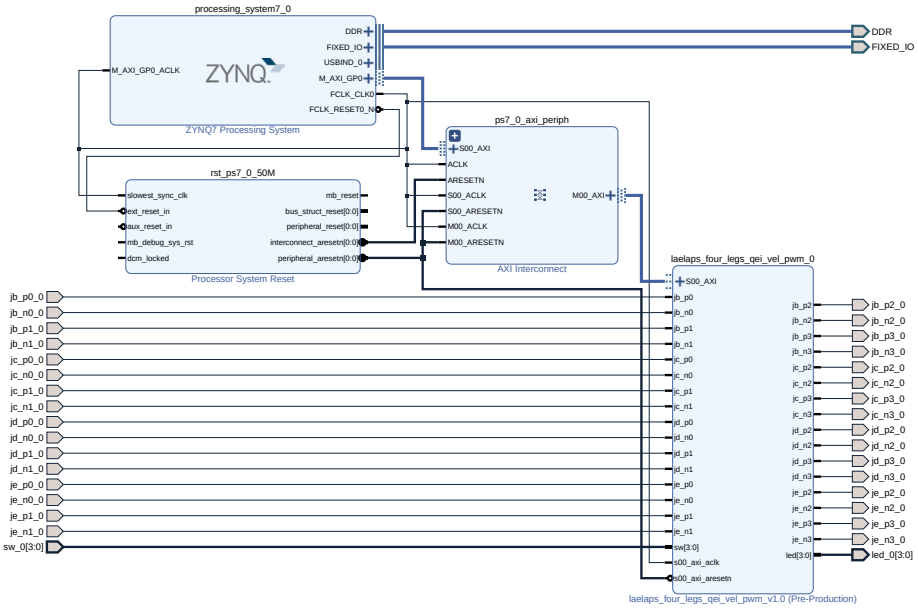


Figure 4.17: Block design of the hardware system for four legs of Laelaps II.

In Table 4.2, the FPGA resource utilization is analyzed, as provided from the Vivado Design Suite. As it can be observed, the main resources that were deployed, are the LUTs and I/Os, while the rest of them are barely depleted. Apart from the LUTRAM and BUFG components, which are not associated with the goal of this thesis, the FF and DSP units could be exploited in order to improve the current implementation or undertake new tasks, useful for the overall functionality of the Laelaps II robot.

Resource	Utilization	Available	Utilization %
LUT	10709	17600	60.9
LUTRAM	60	6000	1.0
FF	2147	35200	6.1
DSP	8	80	10.0
I/O	40	100	40.0
BUFG	1	32	3.13

Table 4.2: Utilization report of the FPGA resources for the four legs design.

An intriguing finding is illustrated in Table 4.3, where the hardware designs for one, two and four legs are compared. Basically, a linear correlation exists between the resources utilization and the number of legs, proving the scalability of the developed system. Although it might seem that the current employed SoC FPGA may not be able to handle a much larger number of joints, it should be noted that the Zybo Development Board is one of the simplest platforms both in resources and processing power, designated for novice hardware engineers. Therefore, the observed consumption is marginal, compared with other alternatives around the same price range.

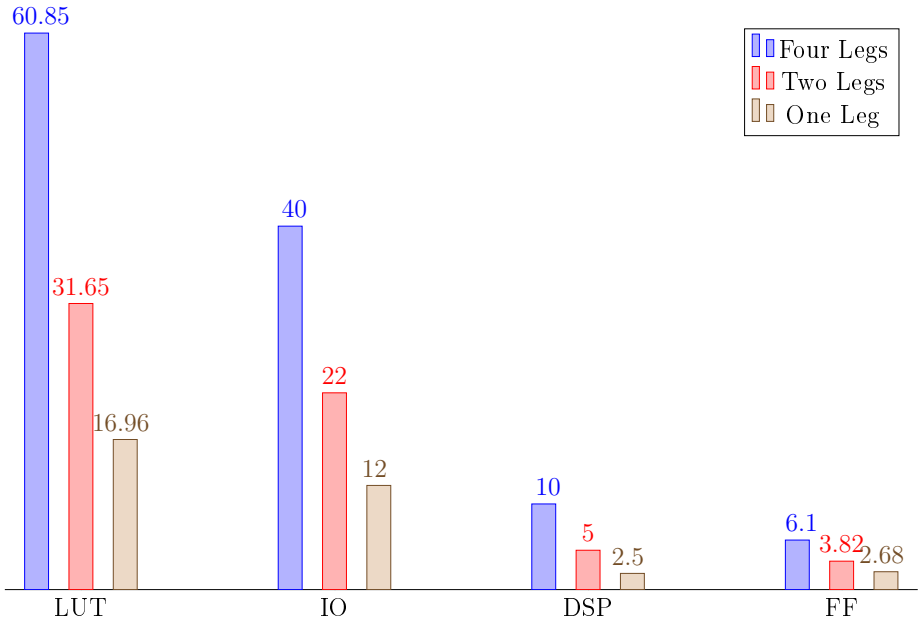


Table 4.3: Comparison of Resources Utilization (%) between One, Two and Four Leg Designs.

Chapter 5

Software Environment

As stated in previous chapters, a Hardware-Software Codesign is proposed in this thesis, for the implementation of a centralized control architecture on Laelaps II quadruped robot via a SoC FPGA. This chapter presents the software running on the ARM Cortex-A9 processor, responsible for the application of both the low and high-level controllers, as well as the retention of the necessary data for post processing using Matlab. The corresponding source code was written in C and executed on a Bare-Metal Operating System (OS) through the Xilinx Software Development Kit (XSDK) design environment, based on Eclipse 4.5.0 Integrated Development Environment (IDE). Mainly during the first experiments, the Gnuplot graphing utility for Linux OS was also employed besides Matlab. In addition, Minicom was used for the manipulation and emulation of the XSDK' terminal, due to its incompatibility with the Linux OS.

5.1 High-Level Controller

First, the high-level part of the current controller's block diagram (Fig.3.9) will be represented, followed by the low-level, which is realized through the *inv_kin*, *pd_control* and *position_controller* functions. All of them will be directed by the main function of the C project, which will embody the high-level of the current controller.

Initially, all parameters pertinent to the inverse kinematics of each leg are configured according to the user's preference, such as the gait type for each leg, the stride frequency, and the ellipse characteristics. In addition, the time duration of the control loop is set, along with another timing parameter, relevant to the transitional phases at the beginning and termination of the loop. Lastly, the Proportional–Derivative Controller (PD-Control) gains can be modified, to define how close the actual foot trajectory will be to the virtual one.

Secondly, the system awaits the pressing of a specific key by the user, in order to initiate the control loop. Inside the control loop, during the *lerp_t_interval* first seconds of the experiment, the ellipse variables a and b are set using the *lerp* function, before the call of the *inv_kin* and *position_controller* functions for each leg. This function constitutes a linear interpolation, in order for the ellipse variables to segue to their final values, starting from the initial values *a_init* and *b_init*. In this case, both initial values were set to zero, while the *lerp_t_interval* varied from two to five seconds, depending on the experiment.

As a result, the robot's legs smoothly transition to the elliptical trajectory.

Next, since the ellipse parameters have reached their destined values, the control loop remains at a steady state for $experiment\_duration - 2 * lerp\_t\_interval$ seconds. During that period, the loop constantly calls the control functions on a frequency specified in the initialization part of the code. Regarding the control loop frequency, after the call of the control functions, a sleep function is included, which inserts a volitional delay of $time\_of\_sleep$ milliseconds. By adjusting that variable, the frequency of the control loop can be regulated, since it determines the temporal duration of the process iteration. It should be noted that the durations of the loop without the sleep function were approximately 9, 17 and 32μs, for one, two and four legs respectively.

Finally, during the last $lerp\_t\_interval$ seconds of the control loop, the ellipse parameters segue to their initial values, again via the *lerp* function, and hence the legs return to their initial positions. After the termination of the control loop, the system once again awaits for the pressing of a specific key. As soon as it is pressed, a new loop is executed, responsible for the printing of the stored data to the XSDK terminal. Due to incompatibility reasons between the XSDK terminal and the Linux OS, the terminal emulation program Minicom is utilized to illustrate the printed data to a Linux terminal, while via a Bash Script all data are simultaneously recorded in a .txt file. Once the software program has been terminated, the .txt file is post processed using a Matlab script, which plots all necessary figures regarding the position, velocity and PWM command of each joint over time.

5.2 Inverse Kinematics & Trajectory Planning

Based on Section 3.4.5, the function inv_kin was created, which is the equivalent C version of the Matlab code described in [¶]. The two versions proved to be tantamount, since disparities existed only after the second decimal place (average error = 0.0022) (Fig.5.1). Apart from the application of the inverse kinematics equations, the trajectory planning segment of the low-level controller was also included in this function. Specifically, semi-elliptical trajectories were deployed, as explained in Eq.3.4.

To be precise, the inv_kin function receives as arguments the following variables:

- in_t : time
- xc : x coordinate of the ellipse's center
- yc : y coordinate of the ellipse's center
- a : semi-major axis's length of the ellipse
- b : semi-minor axis's length of the ellipse
- $traj\_freq$: frequency of elliptical motion
- $phase$: initial phase of the elliptical motion
- $param$: flattening parameter of the y axis amplitude (b)
- *thdes[2]* : float pointer responsible for the storage of the desired positions

After the completion of its computations, the inv_kin function calculates the desired angles

[¶] "7.1 Matlab Leg Modelling Code" of [7]

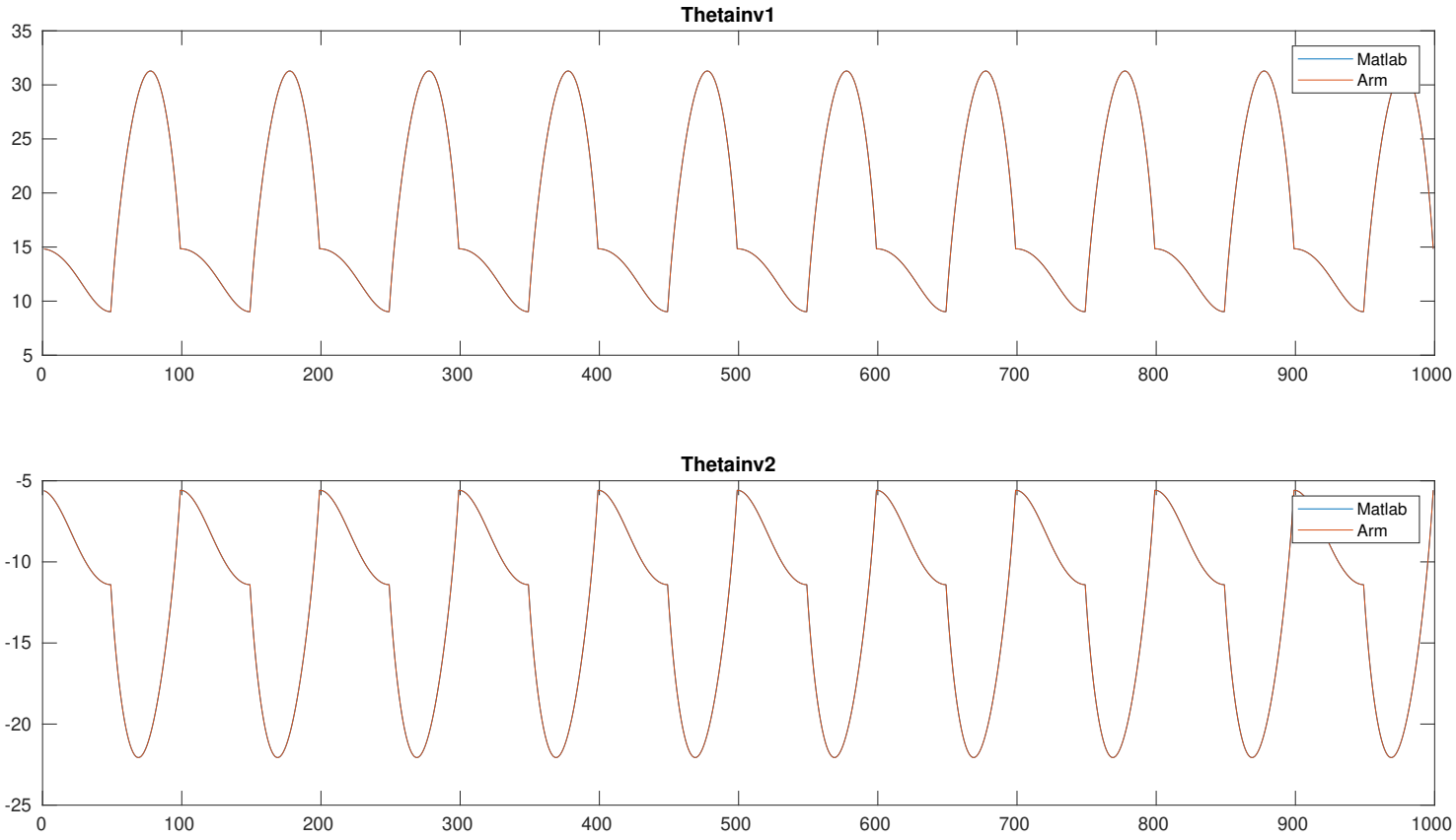


Figure 5.1: Comparison between the Inverse Kinematics outputs in the ARM and Matlab implementations.

of the hip and knee motors in degrees, which are transferred to the main function via the float pointer *thdes*.

5.3 PD-Controller

Based on the block diagram of the current controller (Fig.3.9), the first step was to emulate the low-level controller, and specifically the employed PD joint controllers. Therefore, the first experiments of the FPGA-Processor collaboration revolved around the application of simple control loop feedback mechanisms, such as Proportional Control (P-Control) and PD-Control, in order to determine a fair estimation of the control gains. Before mounting the proposed design on Laelaps II, an experimental platform was utilized (Fig.5.2), equipped with a motor of a total reduction ratio of 51.

First, the P-Control system was studied and carried out. According to the P-Control theory [24], the output of the controller is equal to

$$P_{out} = K_p * e(t) + p0$$

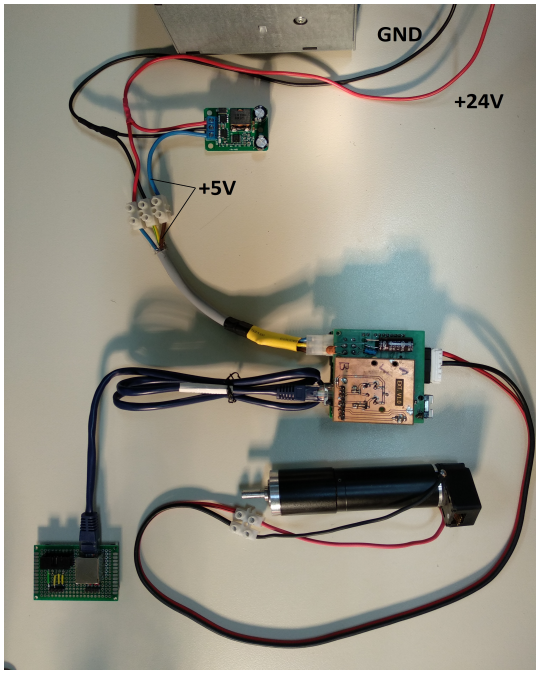


Figure 5.2: Experimental Platform with motor.

where

- K_p: Proportional Gain.
- e(t): Instantaneous process error at time t. e(t) = DA - EA.
- p0: Controller output with zero error (set to 0 in this application).
- DA: Desired Angle.
- EA: Encoder's Angle.

As stated in Section 4.6 , the encoder's angle in counts per revolution is available through the access to the *XPAR_LAELAPS_ONE_LEG_QEI_VEL_PWM_0_S00_AXI_BASEADDR* Memory Address, and it is translated to degrees using Eq.5.1:

$$EA = counts * 360/(51 * 2000), \tag{5.1}$$

where *counts* is the encoder's angle in counts per revolution, 51 is the total reduction ratio of the motor and 2000 is the number of counts that correspond to a full revolution or 360°. As for the P_{out}, its absolute value is the duty cycle that will be transmitted to the PWM system (rounded to the closest integer), while its sign plays the role of the motor's direction signal. Due to the 14 bits of resolution in the PWM segment, the following inequality emerge:

$$0 \leq P_{out} \leq 16383$$

For preservation and safety reasons, a saturation limit of 5000 was introduced to the duty cycle, in order to set a maximum duty cycle of about 30%. As a result, in case the controller's output exceeds that value, it is automatically set to 5000.

In Figure 5.3, the results from several P-Control experiments are illustrated using Gnuplot, where different values for the Proportional Gains were employed. In all cases, the desired

angle was set to 240°and the motor successfully converged to it. Metrics such as overshoot, settling time and the steady-state error were taken into consideration and it was decided to distinguish the value 50 as optimal for the Proportional Gain K_p. Of course, more values were also tested, but they were not comparable with the ones depicted.

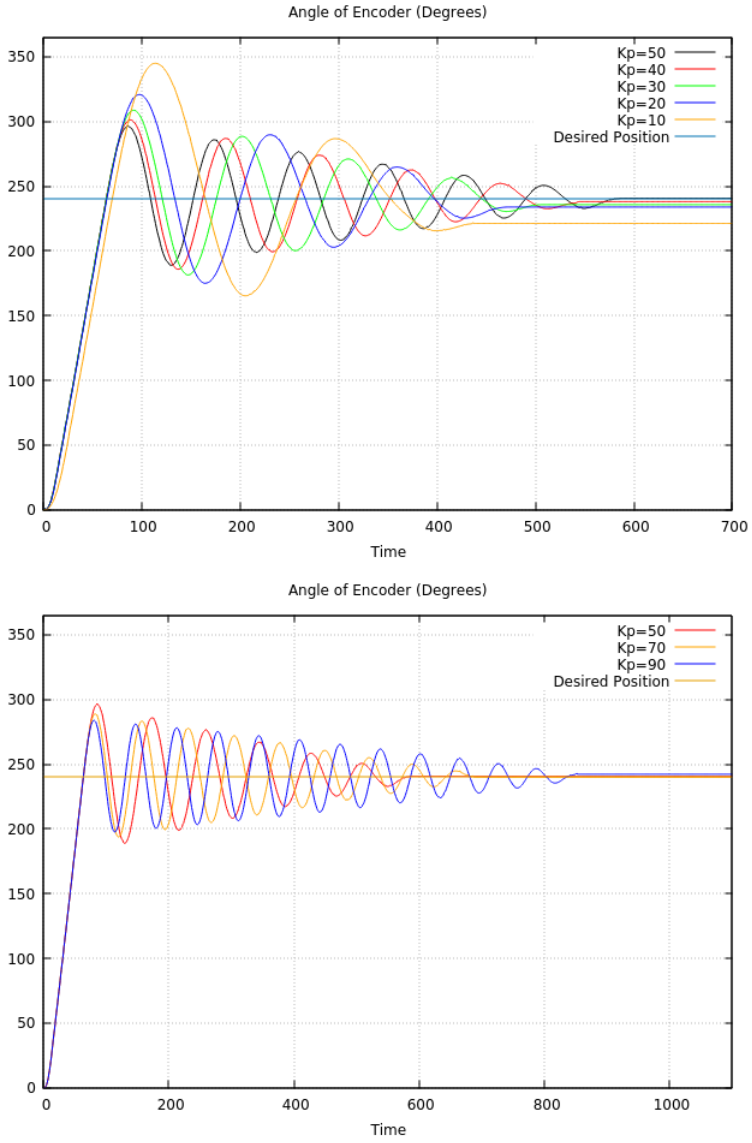


Figure 5.3: P-Control Experiments.

Next, the PD-Control was invoked, by further extending the existing P-Control version. According to the PD-Control theory [24], the equation that describes the PD Control is Eq.5.2:

$$PD_{out} = K_p * e(t) + K_d * \frac{de(t)}{dt}, \tag{5.2}$$

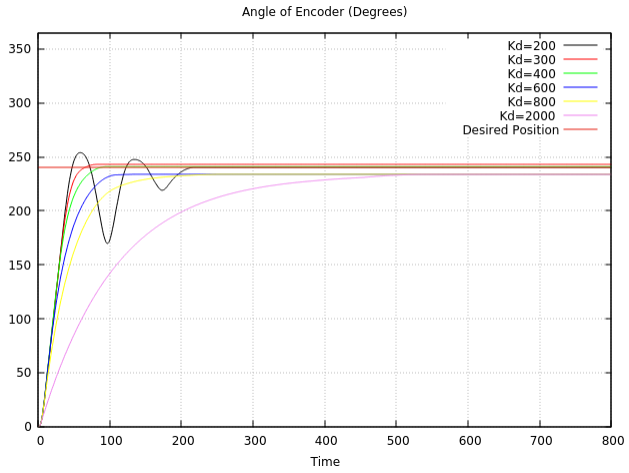
where

- K_p : Proportional Gain
- K_d : Derivative Gain
- $e(t)$: Position error (same as in Eq.5.1)
- $\frac{de(t)}{dt} = DV - MV$: Velocity error
- DV : Desired Velocity
- MV : Motor's Velocity

Similarly with the motor's position, its velocity (value and sign) is also accessed through the suitable Memory Adress and it is divided by 10000, due to the fact that the calculated velocity in the FPGA is multiplied by 10000 for precision reasons, as stated in Section 4.3. Naturally, the same settings apply here as in P-Control, regarding the saturation of the controller's output. Moreover, a saturation limit was incorporated to the calculated velocity, in order to eliminate extremely large values, which may jeopardize the function of the motors. In case the velocity's absolute value exceeds 100 rad/s, then it is automatically set to zero.

In Figure 5.4, the results from several PD-Control experiments are illustrated using Gnuplot, where different values for the Derivative gain were employed. In all cases, the K_p was set to 50, the desired angle was set to 240°and the desired velocity to 0 rad/s, while the motor successfully converged to the desired state. Metrics such as overshoot, settling time and the steady-state error were taken into consideration and it was decided to distinguish the value 300 as optimal for the Derivative Gain K_d.

Manual tuning was chosen as the tuning method of the control gains, since a template preexisted for their values in [7], $K_p = 40$ and $K_d = 0.03$. However, they were not as valid in this instance, probably due to the disparity of the equipped motors.



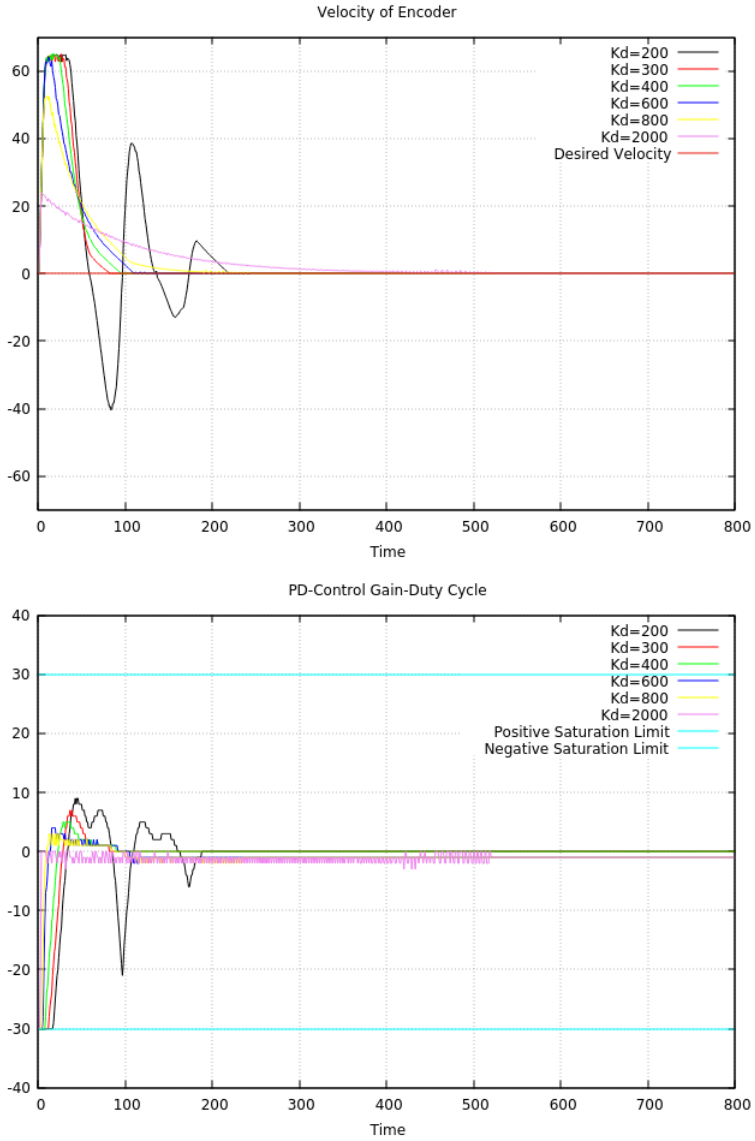


Figure 5.4: PD-Control Experiments.

Current configuration of the PD-Control on ARM

An integer function, entitled *pd_ control*, has been written in C to implement the PD Control of each motor. In particular, the function receives as arguments the following variables:

- t : time
- *num_ of_ motor* : a unique id to distinguish between hip and knee motor
- *counts* : the position of the motor in counts
- *velocity* : the velocity of the motor in rad/s
- *des_ angle* : the desired angle of the motor in degrees

Next, the position of the motor in degrees is calculated according to the type of motor, and the PD Control is applied using the global variables K_p, K_d and *desired_velocity*. The absolute value of the calculation's result is rounded to the nearest integer, while it is truncated in case it exceeds the defined saturation limit (5000 or 30%). The sign of the controller's output will be the direction signal of the motor, while its absolute value will determine the duty cycle of the generated PWM signal that drives the motor. Both of them are integrated into one variable, which is the return value of the integer function.

The *pd_control* function is called by the integer *position_controller* function, which has as arguments the time (t) and the two desired angles for the hip and knee respectively. Additionally, the *position_controller* function receives the positions and velocities of both motors, in counts and rad/s respectively, from the FPGA and consequently calls the *pd_control* function. Thus, after the function receives the return value of the *pd_control* function, it transmits the value to the FPGA, where the PWM and Direction signals are generated. Lastly, the necessary data for post-processing and illustration are stored inside the *pd_control* function via a global two dimensional array, including the number of the motor, its position and velocity, the calculated duty cycle, the corresponding time instant of their usage and the desired position.

In total, the complete software framework is illustrated in Fig.5.5. The main function of the C code initially sets the parameters of the inverse kinematics and serially executes a while loop (1), where it calls the *inv_kin* and *position_controller* functions for each leg . The former returns the desired angles for each leg, which are passed as arguments to the latter (2). Consequently, the *position_controller* function receives the necessary positions and velocities from the FPGA (3) and promptly forwards them to the *pd_control* function (4). Once, the PD-Controller has been applied, its output is returned to the *position_controller* function (5) and it is ultimately transfered back to FPGA (6). Following their completion, the *usleep* function inserts the intended temporal delay and after its expiration another iteration of the loop launches. Upon conclusion of the control loop, the stored values are saved in a .txt file and later they are plotted for supervision purposes.

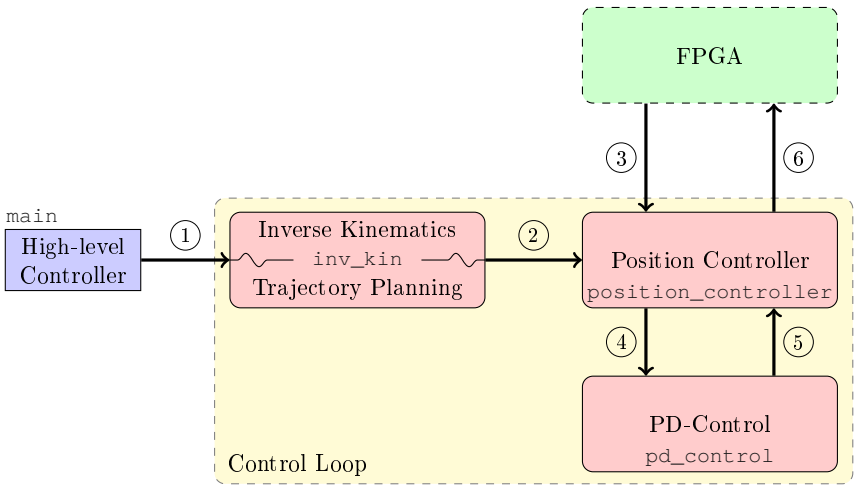


Figure 5.5: Operating diagram of Software Partition.

Chapter 6

System Integration

In this chapter, the physical interconnection between the Zynq Development Board and the Laelaps II quadruped robot will be elaborated and illustrated. Subsequently, since both the Hardware and Software segments of the processing system have been presented in the previous chapters, the finalized architecture will be presented.

6.1 Connection of Laelaps II and ZYBO Board

During the initial experiments on the preliminary experimental platform (Fig.4.2), the Zybo Board received its inputs through direct wiring with the encoders. In fact, a breadboard was deployed, not only for the wiring, but also for the power supply distribution of the encoders (3.3V). Later on, a temporary Circuit Board was manufactured and used as mediator between the Pmods of the Board and the encoders and motors' drivers, as part of the second experimental platform (Fig.5.2). This platform was more advanced than the first one, whilst providing a more convenient and distinct structure. In addition, in the second platform the encoders were supplied with separate power than the SoC FPGA Board. Specifically, they were supplied with 5V, while the Zybo Board's Pmods can handle up to 3.3V. For the purpose of overcoming this obstacle, the AC-LLC8-V2 level shifter was purchased (Fig.6.1), capable of converting signals from 5V to 3.3V (and vice-versa). As a consequence, the level shifter intervened between the Pmods and the second experimental platform, enabling their connection.

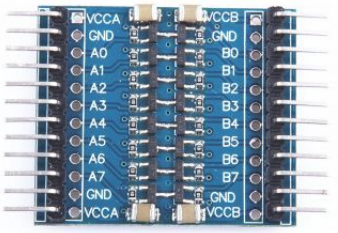


Figure 6.1: 8 Channels Bi-Directional Logic Level Converter.

Although the same approach was also taken concerning the PWM signals, it was discovered

that the level shifter distorted the signals, while a direct connection with 3.3V PWM signals produced positive results, without generating any issues. Therefore, it was decided to employ the level shifter exclusively for the encoders-Pmods connection.

As soon as the conducted experiments produced promising results, two new circuit boards were constructed, aiming to fabricate a structured system, without loose cables that not only add noise, but additionally can tangle the user.

On the one hand, an intermediate board was introduced, which firmly connects with the Zybo Board through the Pmod ports (Fig.6.2). The main purpose of this board is to link the encoders' outputs with the corresponding inputs of the level shifters and afterwards with the respective Pmods. Moreover, the PWM outputs of the SoC FPGA are redirected to a socket, through which the connection with the secondary board is established. In total, the board connects with four of the Pmod ports, utilizes two level shifters and includes four 2x8 sockets, each destined for one leg of the robot. Furthermore, a power jack is included, which provides the necessary 5V supply to the level shifters. It should be noted that the Zybo Board is also supplied with 5V and hence shares the same power supply as this board.

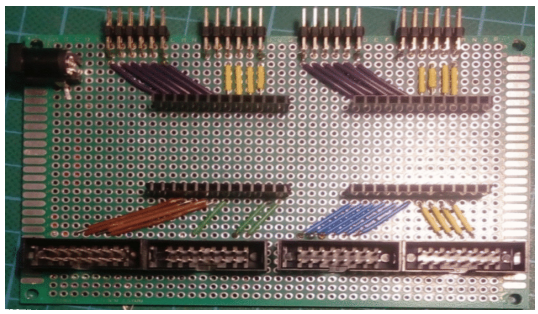


Figure 6.2: Intermediate Circuit Board between Zybo and Secondary Boards.

On the other hand, four secondary Circuit Boards were fabricated, responsible for the direct connection with the encoders and the motors' drivers. Each board entails one 2x8 socket for the connection with the intermediate board, two 2x5 sockets for the connection with the encoders and two Ethernet UTP sockets for the connection with the drivers (Fig.6.3). For the interconnection between the intermediate and secondary boards, IDC Ribbon Cables were exploited.

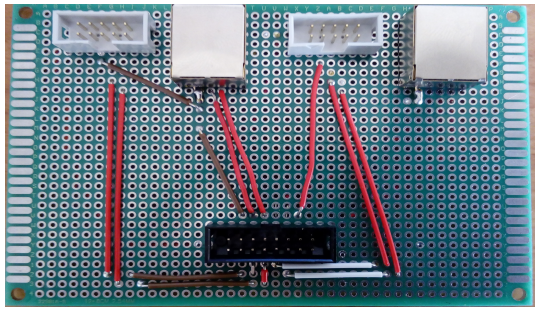


Figure 6.3: Secondary Circuit Board for connection with encoders and actuators.

In total, the Zybo Board together with the Intermediate Board and the four Secondary

Boards were unified into one compact block tower, in position for connection with the Laelaps II robot. As a result, the finalized centralized control tower of hardware is illustrated in Fig.6.4.

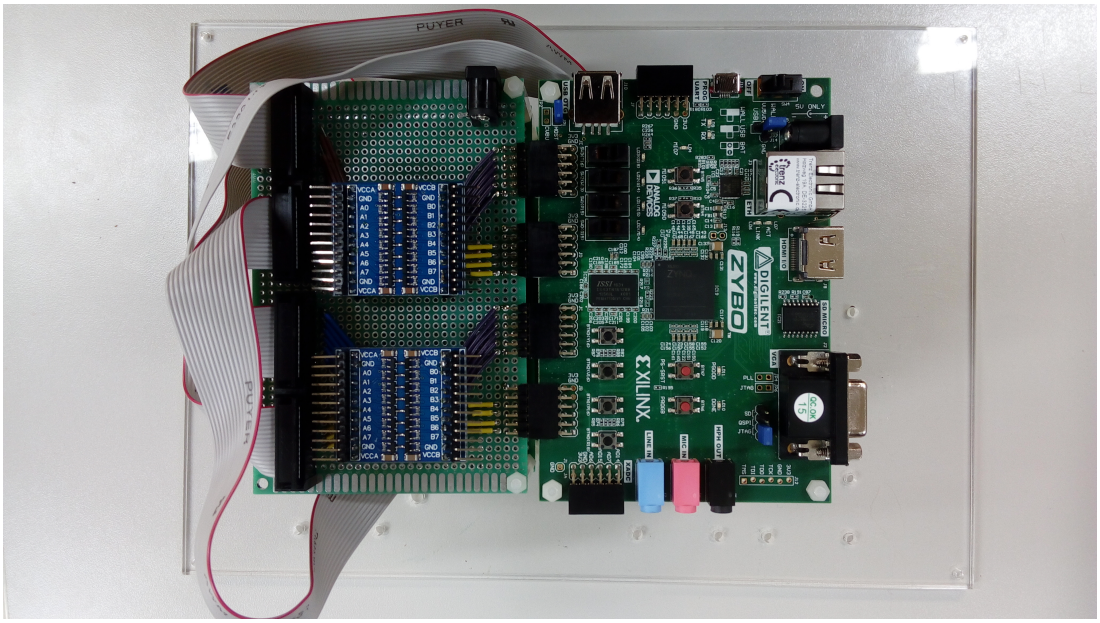


Figure 6.4: Centralized Control Tower of Hardware.

6.2 Finalized Architecture

At this point all segments, both physical and theoretical, of the proposed implementation have been presented and clarified. Hence, in this section, the integration of the centralized control tower with the Laelaps II robot will be depicted, along with the full-scale operating description of the realized scheme.

The control tower was positioned on the torso of the Laelaps II, for spatial purposes and in order to minimize its distance from the forward and hinder legs (Fig.6.5). Next, both encoders and motors' drivers were linked with the tower, while an external power supply was provided to the Zybo and Intermediate Boards. The programming and communication with the Zybo Board was achieved via a USB cable that connected with a Personal Computer (PC).
The centralized control scheme consists of two blocks: the Laelaps II quadruped robot and the SoC FPGA. Initially, the mounted encoders generate their 0-5V outputs, which are transferred through the Secondary Boards to the Intermediate Board and thus the Level Shifter. After their conversion to 0-3.3V signals, they are directly linked with the Pmod ports of the Zybo Development Board. At this point, the FPGA initiates their filtering and calculates the corresponding position and velocity of each motor. Consequently, these data are forwarded to the slave registers of the AXI4-Lite Interface, which are interconnected with the AXI4-Lite bus. Concurrently, on the other side of that bus, the ARM processor has already executed the high-level controller and configured all necessary variables for the legs'

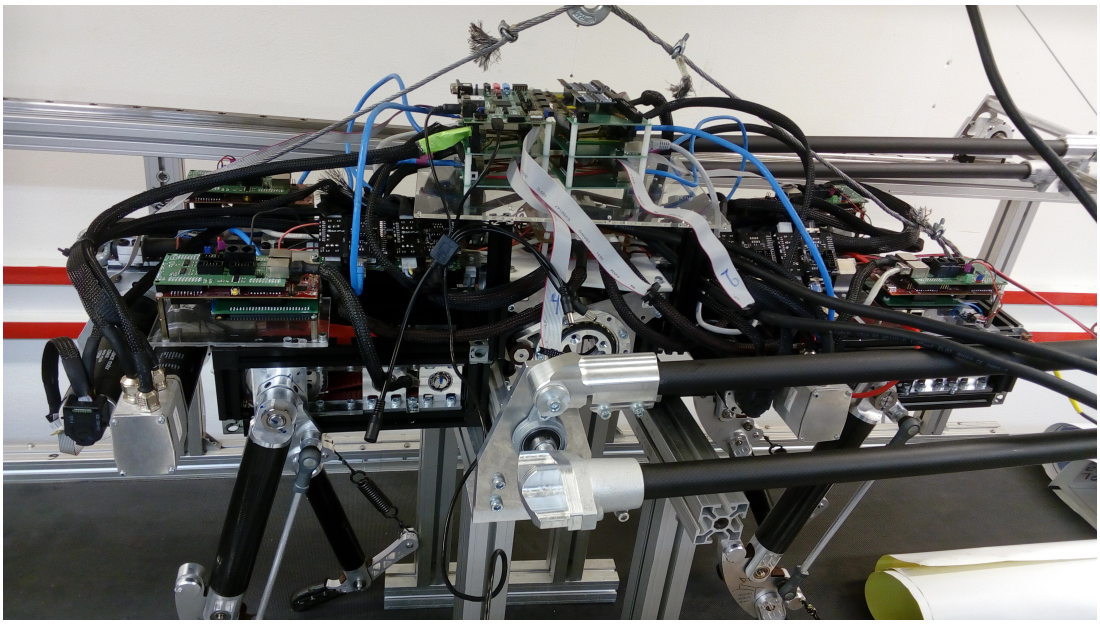


Figure 6.5: Centralized Control Tower unified with Laelaps II.

motion planning and control. In fact, the control loop has already been initialized and both the inverse kinematics and trajectory planning of the low-level controller have been carried out for this iteration of the loop. As a consequence, the desired positions for each joint have been calculated and along with the motors' positions and velocities, accessed via the AXI4-Lite bus, they are passed as arguments to the PD-Control unit. Based on the preset control gains, the PD-Control is applied and its output, which will determine the motors' torques, is sent back to the FPGA via the AXI4-Lite Interface. Now, the respective slave registers, which contain the PD-Controller's outputs along with some auxiliary data, are interlinked with the *PWM* blocks of the FPGA. According to the computed outputs, these blocks produce PWM signals of appropriate Duty Cycles that are exported to the Pmod ports and thus the Intermediate Board. Lastly, the PWM signals are distributed to the Secondary Boards and subsequently to the motors actuators that control the robot's legs. In total, this procedure is repeated constantly throughout the duration of each locomotion experiment, allowing the system to converge to the desired state.

Chapter 7

System Evaluation

Following an extensive number of experiments with the developed control scheme, either on experimental platforms or Laelaps II, the validity of the proposed system was verified and it was decided to attempt the control of all four legs of Laelaps II. This chapter includes the results from the final experiment, in the context of which, six joints were successfully controlled and performed a trotting pattern in sync, while Laelaps II was elevated and hence there was no contact with the ground. This kind of quadruped movement was selected, as it is considered one of the simplest symmetrical gaits. In all experiments on four legs, the processor's running frequency was 650MHz, while the FPGA's was 83MHz. However, only the sampling of the inputs operates at that frequency, whereas the rest of the FPGA operations are executed on a frequency of approximately 20MHz.

The procedure of the experiment initially involved the definition of the system parameters in the main function of the processor code, including the desired elliptical trajectory and the control gains K_p and K_d. Next, the execution of the control process was launched and after the completion of the experiment, the recorded data were logged to a txt file and plotted using a custom made Matlab script.

All parameters pertinent with the experiment are stated in a table, while four figures are provided, depicting:

- The desired elliptical trajectory of each leg's end-effector (toe), compared with their actual response, with reference to coordinate systems located in the hip joints of the legs
- The desired positions of both knee and hip joints for all eight joints, compared with their respective actual response.
- The duty cycles of each PWM signal sent to each joint's motor, along with their respective predefined saturations limits (both positive and negative)
- The velocity estimation of each leg motors and its respective desired value

7.1 Laelaps II Locomotion Experiment

In this experiment, Laelaps II is originally elevated with all four legs set in their initial positions, meaning that all joints are at 0°. Next, the system is configured with the pa-

rameters shown in Table 7.1, except for the trajectory parameters, *a ellipse* and *b ellipse*, which are set to zero. After the control loop begins, during the first *Lerp Inteval* seconds of the experiment, their values are increased to 3cm linearly with time. Simultaneously, the Laelaps II starts moving slowly, until it has reached its steady state. During the last *Lerp Inteval* seconds of the experiment, the *a ellipse* and *b ellipse* segue to their initial values and Laelaps II decelerates and eventually stops moving. The user is notified of the experiment's completion and after a specific key is pressed, the data retention initiates. Lastly, the data are post processed in Matlab and respective plots are created.

Trajectory Parameters						
x center	y center	a ellipse	b ellipse	Frequency	Phase	Flattening Parameter
0cm	57cm	3cm	3cm	1Hz	0/180°	0

Control Gains		PWM Saturation Limits		Experiment Duration	Lerp Interval
K_p	K_d	Positive	Negative	20sec	5sec
500	300	+30%	-30%	Control Frequency	1kHz

Table 7.1: Parameters of Locomotion Experiment

Following the expiration of the first *Lerp Inteval* seconds of the experiment, the *a ellipse* and *b ellipse* parameters have reached their final values. As a result, the toe of each leg performs a specific trajectory, while trying to converge to the desired elliptical trajectory. In Fig.7.1, the recorded responses of each toe is depicted, along with the corresponding desired elliptical trajectory. Although errors exist, the desired elliptical orbits are tracked within acceptable bounds. Due to the fact that the robot was elevated, there was no disturbance from contact with the ground, while considerably high values of the Control gains were selected.

Figure 7.2 displays the desired value of each leg's hip and knee joint angle and the actual – real response of every respective joint throughout the experiment. Both the transition and the steady state phases are illustrated. In this figure, it is evident that the Forward Right leg operated at a different phase (180°), than the rest of the legs that functioned with zero phase. As one may observe in these figures, the desired values are closely tracked by all legs, since large values were considered for the control gains of all motors.

Analogously, Fig.7.3 illustrates the velocity estimation of each leg's joints using the hardware implementation, as described in Section 4.3. Once again, as anyone can understand, the velocities of every motor are always within acceptable range, and thus there is no need to integrate a filter, to smooth out any spikes.

Figure 7.4 depicts the Duty Cycles of the PWM signals for each leg's motor with their respective predefined limits. These values are the output of the PD controller exploited in our application and are directly translated in torque commands since a current control architecture is implemented. As one may observe, the commands in both hind legs are always within the limit range, except for certain instances, where spikes can be observed. The nature of this phenomenon are not known, however they did not affect the robot's functionality in any way.

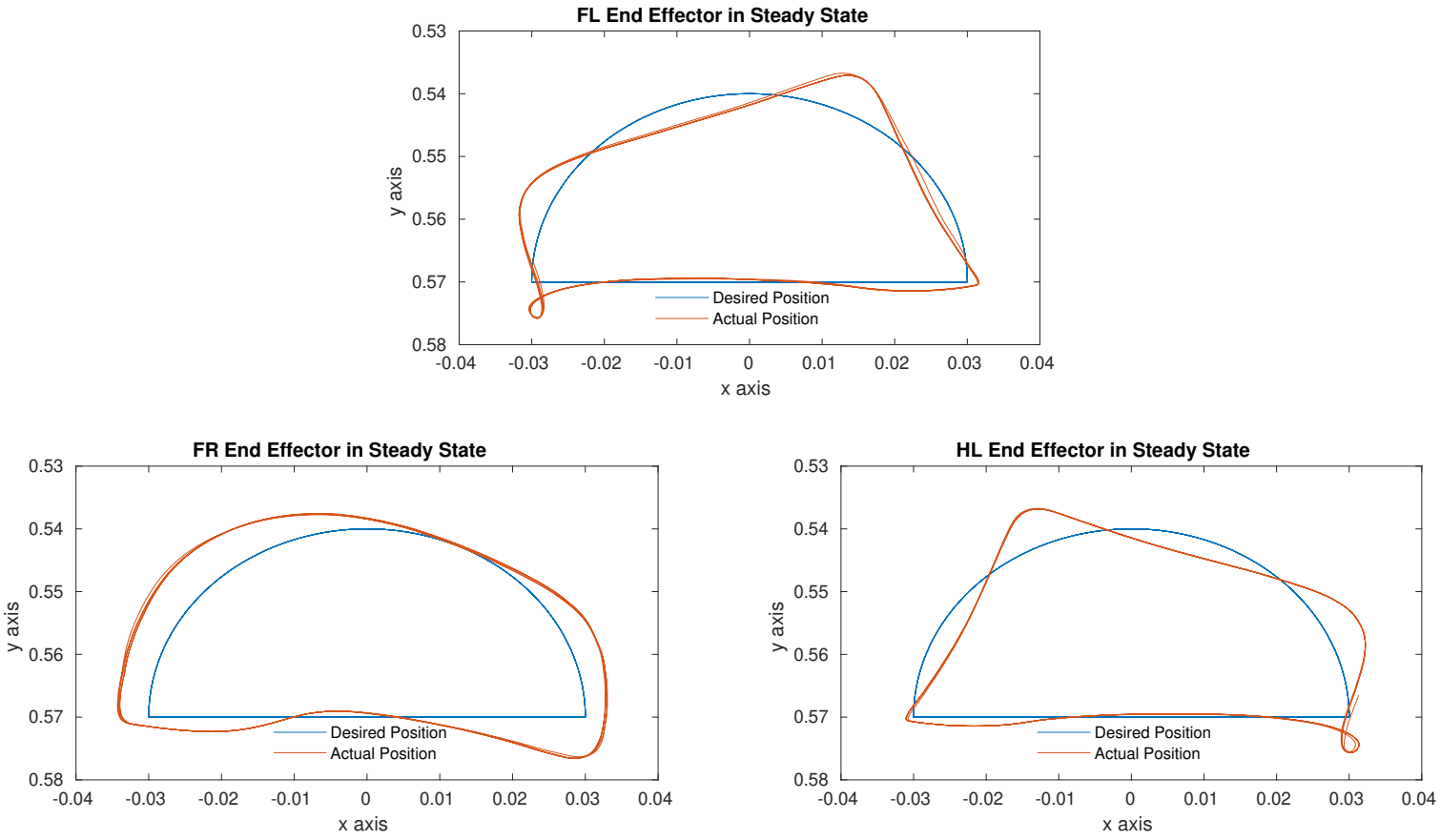


Figure 7.1: Recorded Response of all legs toe along with the desired elliptical trajectories.

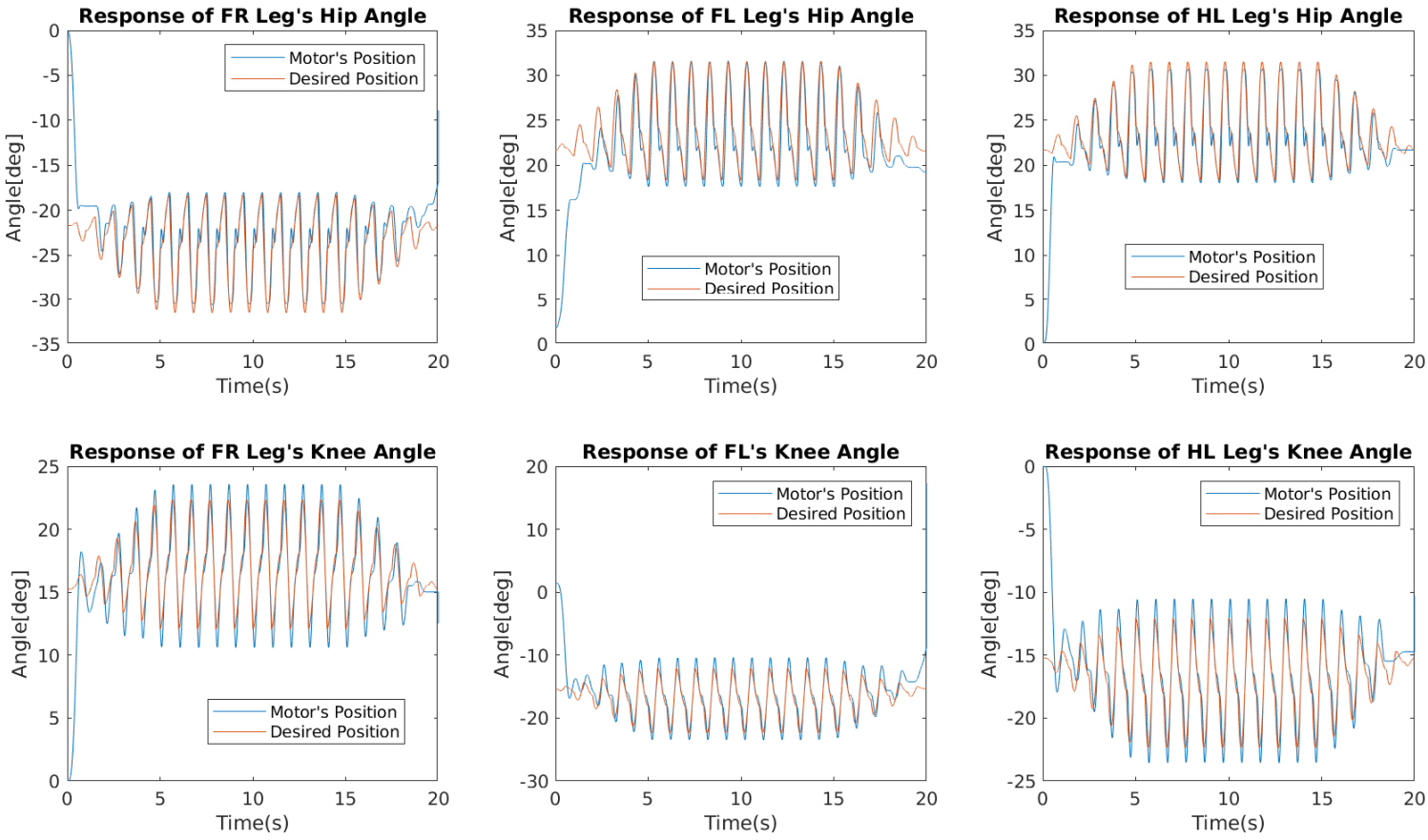


Figure 7.2: Recorded Response of all hip and knee angles along with the desired positions.

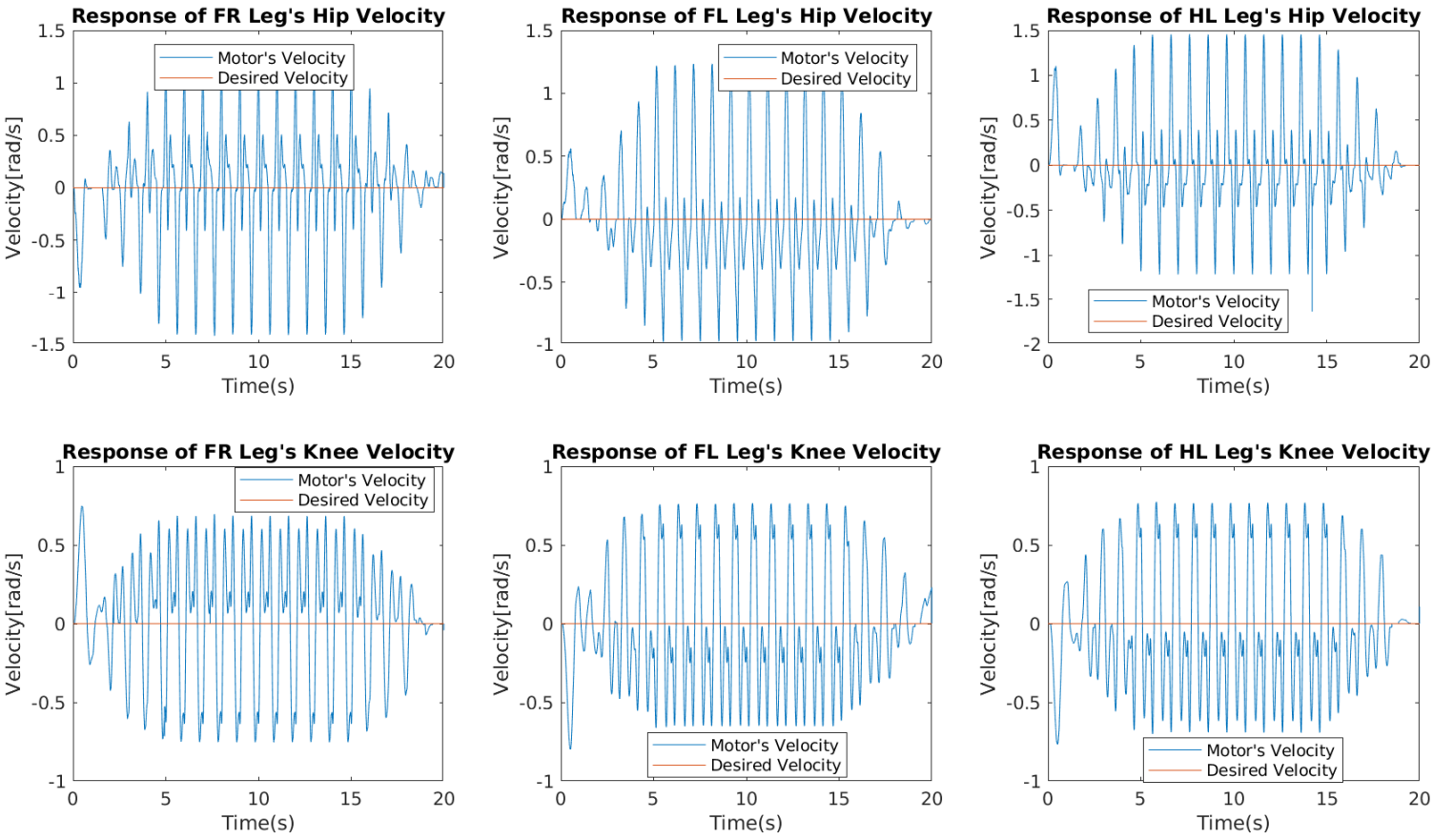


Figure 7.3: Velocity Estimation of each leg's joint along with their desired values.

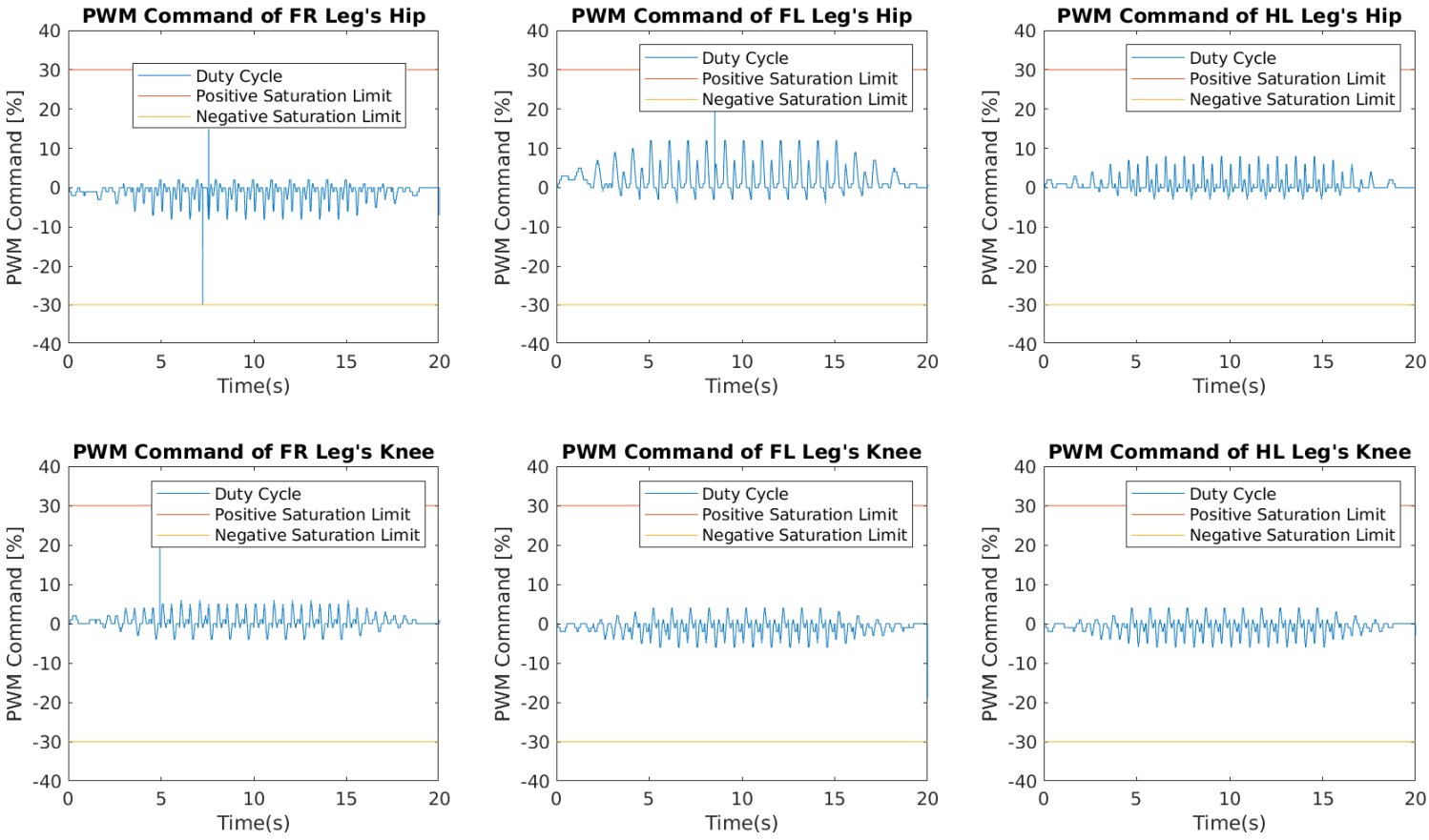


Figure 7.4: Duty Cycles of each leg's motor and their respective predefined saturations limits.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The goal of this thesis was successfully achieved, since a complete functional centralized control scheme was produced, capable of controlling all four legs of Laelaps II quadruped robot. The whole system is hosted on a highly affordable SoC FPGA Board, which not only exhibits sufficient performance, but additionally it consists of only one device, and hence negates the need for a sophisticated communication protocol, as EtherCat, mandatory in decentralized architectures. The total purchasing cost corresponds only to the cost of the Zynq™- 7000 Development Board (ZYBO), which is almost 2% of the previous centralized and 20% of the current decentralized architecture. As a result, the purchase of spare parts is certainly economically feasible, while one of most important drawbacks of centralized architectures is confronted, since even in the event of damage, the board can be easily replaced and reconfigured in a matter of seconds.

Additionally, the system could be characterized as highly responsive. On the one hand, the coexistence of the FPGA and the ARM processor on the same chip dramatically reduces their communication overhead, while establishing a higher bandwidth. On the other hand, the processor's occupation is rather low, since the calculations that need to be executed require inconsiderable effort, especially when compared to its operating frequency. In fact, tasks such as the calculation of the position and rotational velocity of each motor are executed in the FPGA, whereas in the current implementation microprocessors handle this payload. This does not only disencumbers the processor, but also utilizes the valuable feature of parallel processing in FPGAs, which minimizes substantially the processing time, while restricting any possible latencies.

Next, an important aspect of our system is its programming versatility. After a meticulous research of the current design as well as its future needs, an appropriate architecture was designed, where volatile parts of the system can be easily reconfigured, while pieces that are planned to remain unvarying were programmed in hardware description language. Nevertheless, even some significant segments realized in the FPGA, can be easily altered, through the included generic variables, which allow the user to reconfigure a number of variables. Furthermore, the simplicity of the proposed scheme is also proved by its rather small development time, considering that hardware programming is frequently time consuming and challenging.

Lastly, an effortless transition to architectures with more joints is supported, as the Zybo Board is equipped with multiple I/O peripherals, while the expansion of the corresponding software and hardware is rather convenient and straightforward. Moreover, even if it is decided to replace the current SoC Board with another board (for instance a more resourced one), the configuration data could be readily adjusted to the capabilities of the new device.

In conclusion, judging by the promising results so far, it is safe to declare the collaboration between the robotics and FPGA fields as prosperous and hence definitely worth investing in.

8.2 Future Work

In spite of the proven functionality of the described implementation of motion control, in both software and hardware, the developed system has plenty room for improvement. Additionally, the development of the system will continue even after the completion of this thesis, in order to reach the maximum possible potential. Nevertheless, both forthcoming and future upgrades are stated for future reference.

First of all, an Interrupt Service Routine (ISR) could be included in the software segment of the system, in order to provide increased temporal precision along with several convenient capabilities. This feature could allow the parallel execution of both the control algorithm and the storage of the recorded data, while it may enable the user to insert commands during the experiments and hence modify any parameter on real-time or terminate the execution smoothly.

Secondly, in the current implementation, only one of the ARM Cortex-A9 cores is utilized, whilst the second one remains idle. Perhaps, the process load could be divided in the future and one processor could handle the control loop along with the ISR, while the second one could take over the communication with the user. Furthermore, the operating frequency of the processor could be reduced, since the current implementation is not computationally demanding, and hence the power consumption would lessen.

In addition, since the resources utilization was insignificant, the FPGA could take over some of the operations that are currently executed by the processor and hence reduce its payload. For instance, the PD-Control segment could be easily realized in hardware, whilst the inverse kinematics and trajectory planning along with the high-level controller remain in the processor's scope. Another interesting approach would be to devise a complete hardware implementation either in VHSIC Hardware Description Language (VHDL) or via the High-Level Synthesis (HLS) software, although the current SoC FPGA's resources may not suffice. In those cases, the processor will be solely responsible for the recording of the necessary data as well as the communication with the user.

Moreover, another FPGA-based platform could be purchased that provides either more hardware resources, I/O peripherals or processing power. Namely, two appealing options are the second generation of current employed device, Zybo Z7-20, as well as the Z-turn Board, which are equipped with more I/O ports and hardware resources than the Zybo device. As a result, more signals could be processed, meaningful for the overall functionality of the robot, such as the handling of more sensors, pertinent with the ground's roughness or the identification of obstacles near the robot.

Ultimately, concerning the processor segment, a complete OS could be installed such

as the Linux distribution by Xilinx called Xillinux, which is based on the Ubuntu TS 12.04 for ARM and can be immediately deployed on Zybo [25]. Likewise, another useful option is the Linux-ROS operating system, which is avowedly an appropriate tool for robotic devices and is already familiar to the CSL - EP Laboratory research team. An OS would ease the data storage via a file descriptor, while there would be no need for a PC to cross compile the execution code, as in the current implementation. Especially, the system would become more portable, whilst enabling its configuration even in remote areas.

Bibliography

[1] S. Guo, Y. He, L. Shi, S. Pan, R. Xiao, K. Tang, and P. Guo, "Modeling and experimental evaluation of an improved amphibious robot with compact structure," *Robotics and Computer-Integrated Manufacturing*, vol. 51, pp. 37–52, 2018.

[2] E. I. Guerra-Hernandez, A. Espinal, P. Batres-Mendoza, C. H. Garcia-Capulin, R. De J. Romero-Troncoso, and H. Rostro-Gonzalez, "A FPGA-Based Neuromorphic Locomotion System for Multi-Legged Robots," *IEEE Access*, vol. 5, pp. 8301–8312, 2017.

[3] A. Ahmed, M. Henrey, P. Bloch, P. Gupta, C. Panaitiu, D. Naaykens, S. Strbac, L. Shannon, and C. Menon, "A Miniature Legged Hexapod Robot Controlled by a FPGA," *Journal ISSN*, vol. 1929, p. 2724, 2013.

[4] S. M. Qasim, A. A. Telba, and A. Y. AlMazroo, "Fpga design and implementation of matrix multiplier architectures for image and signal processing applications," *International Journal of Computer Science and Network Security*, vol. 10, no. 2, pp. 168–176, 2010.

[5] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* Strathclyde Academic Media, 2014.

[6] Digilent Inc., "ZYBO™FPGA Board Reference Manual," 2014. Revised February 27, 2017.

[7] S. Athiniotis, "Firmware design for microcontrollers on EtherCAT network for quadruped robot motion control." `http://dspace.lib.ntua.gr/handle/123456789/47669`, 2018.

[8] K. Machairas and E. Papadopoulos, "An active compliance controller for quadruped trotting," in *2016 24th Mediterranean Conference on Control and Automation (MED)*, pp. 743–748, June 2016.

[9] Avago Technologies, "HEDS-5640 Encoder of Avago Datasheet." `https://docs.broadcom.com/docs/AV02-0993EN`, 2014.

[10] "Quadrature Decoding in Software." `http://www.me.unm.edu/~starr/teaching/me470/quad.pdf`.

[11] J. T. Machado and M. F. Silva, "An overview of legged robots," in *International symposium on mathematical methods in engineering*, MME Press Ankara, Turkey, 2006.

[12] Y. Li, B. Li, J. Ruan, and X. Rong, "Research of mammal bionic quadruped robots: A review," in *2011 IEEE 5th International Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 166–171, Sep. 2011.

[13] G. A. Bekey, "Autonomous robots: from biological inspiration to implementation and control," p. 304, 2005.

[14] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, pp. 318–331, March 2015.

[15] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. now, 2008.

[16] J. H. Barron-Zambrano and C. Torres-Huitzil, "Fpga implementation of a configurable neuromorphic cpg-based locomotion controller," *Neural Networks*, vol. 45, pp. 50–61, 2013.

[17] V.-H. Xilinx, "Vivado Design Suite User Guide-High-Level Synthesis," 2014.

[18] U. Farooq, Z. Marrakchi, and H. Mehrez, "Fpga architectures: An overview," in *Tree-based heterogeneous FPGA architectures*, pp. 7–48, Springer, 2012.

[19] Glen Young, "How to use FPGAs for quadrature encoder-based motor control applications." `https://www.edn.com/design/programmable-logic/4015131/How-to-use-FPGAs-for-quadrature-encoder-based-motor-control-applications`, September 11, 2007.

[20] "Why a Gearhead." `https://www.portescap.com/products/motor-accessories/understanding-gearheads`.

[21] Texas Instruments, *TMS320x2833x, 2823x Enhanced Quadrature Encoder Pulse (eQEP) Module*, August 2008.

[22] "PWM Generator (VHDL)." `https://eewiki.net/pages/viewpage.action?pageId=20939345`.

[23] "Creating a custom IP block in Vivado." `http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html`.

[24] B. W. Bequette, *Process control: modeling, design, and simulation*. Prentice Hall Professional, 2010.

[25] Δ. Γουρουνάς, "Hardware acceleration of image registration algorithm on fpga-based systems on chip," 2018.