ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Έλεγχος Μηχανής Αριθμητικού Ελέγχου με FPGA

Του φοιτητή
Ιακώβου Πέτρος
Αρ. Μητρώου: 52308Μ

Επιβλέπων
Δημήτριος Παπακώστας
Βαθμίδα: Κοσμήτορας

Ημερομηνία Ιούνιος 2026

Τίτλος Δ.Ε. Έλεγχος Μηχανής Αριθμητικού Ελέγχου με FPGA
Κωδικός Δ.Ε. …
Ονοματεπώνυμο φοιτητή: Ιακώβου Πέτρος
Ονοματεπώνυμο εισηγητή …
Ημερομηνία ανάληψης Δ.Ε. …
Ημερομηνία περάτωσης Δ.Ε. …

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Κατά την εκπόνηση της διπλωματικής εργασίας χρησιμοποιήθηκαν εργαλεία τεχνητής νοημοσύνης Codex της OpenAI και Claude της Anthropic για υποβοήθηση στην συγγραφή κώδικα. Το σύνολο του τεχνικού περιεχομένου, της υλοποίησης και της επιστημονικής τεκμηρίωσης ανήκει στον συγγραφέα, ο οποίος φέρει την πλήρη ευθύνη.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Αφορμή για το θέμα της διπλωματικής μου εργασίας αποτέλεσε η ενασχόλησή μου με προγενέστερο project που αφορά την μελέτη και την κατασκευή CNC μηχανής δύο αξόνων. Στο project αυτό για τον έλεγχο της μηχανής χρησιμοποιήθηκε ο μικροελεγκτής ATmega328P. Όπως τελικά αποδείχτηκε η υλοποίηση με μικροελεγκτή θέτει κάποιους περιορισμούς όσον αφορά:

- Τον πραγματικό παράλληλο έλεγχο μεταξύ πολλαπλών αξόνων.
- Την σταθερότητα στην περιοδικότητα των παλμών (Jitter).
- Την προσαρμογή του ελεγκτή σε διαφορετικές αρχιτεκτονικές μηχανών.

Φυσικά υπάρχουν μέθοδοι και τεχνικές που περιορίζουν σε σημαντικό βαθμό τα αρνητικά αποτελέσματα που θέτουν οι παραπάνω περιορισμοί, καθιστώντας την λύση με μικροελεγκτή αποδεκτή σε μη βιομηχανικές εφαρμογές. Στα πλαίσια όμως εφαρμογών όπου η ακρίβεια και ευελιξία αποτελεί καθοριστικό παράγοντα για την ποιότητα και την αποδοτικότητα ό έλεγχος με μικροελεγκτή δεν μπορεί να επιφέρει το επιθυμητό αποτέλεσμα. Σύγχρονες μελέτες (Santos et al., 2018· Zhang et al., 2015) υποστηρίζουν πως οι αρχιτεκτονικές βασισμένες σε FPGA μπορούν να αντιμετωπίσουν αποτελεσματικά τα παραπάνω προβλήματα.

Παράλληλα με την μεταφορά του ελέγχου σε FPGA, διαπίστωσα (μέσω από την εμπειρία που έχω στο βιομηχανικό σχεδιασμό) πως θα μπορούσε να γίνει χρήση εργαλείων CAM λογισμικού έτσι ώστε ο ίδιος ο εκλεκτής να μπορεί να εξυπηρετήσει μηχανές διαφορετικών χαρακτηριστικών χωρίς να απαιτείται αλλαγή στο Hardware.

Τα οφέλη που θα προσκομίσω μετά την ολοκλήρωση της διπλωματικής εργασίας είναι: 1) η βελτίωση της λειτουργίας της μηχανής που σχεδίασα, 2) η περαιτέρω εξέλιξη των δεξιοτήτων μου στον σχεδιασμό και την ανάπτυξη εργαλείων λογισμικού CAD-CAM και 3) η εκμάθηση στην ανάπτυξη ψηφιακών κυκλωμάτων με FPGA.

Το project που αφορά την υλοποίηση με χρήση μικροελεγκτή βρίσκεται στον παρακάτω σύνδεσμο:

Design_and_Implementation_of_a_Computerized_Numerical_Control_CNC_System

Περίληψη

Η παρούσα διπλωματική εργασία αποτελεί την τελική εργασία του μεταπτυχιακού προγράμματος σπουδών Εφαρμοσμένα Ηλεκτρονικά Συστήματα του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου Ελλάδος. Πραγματεύεται την ανάπτυξη ενός FPGA ελεγκτή μηχανών CNC γενικής χρήσης. Η φιλοσοφία στον σχεδιασμό της αρχιτεκτονικής του συστήματος προβλέπει την προσαρμογή του σε μηχανές διαφορετικών μηχανολογικών χαρακτηριστικών.

Η υλοποίηση του συστήματος χωρίζεται σε δύο μέρη. Το πρώτο αφορά την ανάπτυξη εργαλείων λογισμικού CAD-CAM σε γλώσσα AutoLISP και περιλαμβάνει πέντε εργαλεία. Το πρώτο εργαλείο παρέχει μέσω φόρμας τη δυνατότητα καταχώρησης μοντέλου μηχανής και ορισμού των παραμέτρων της ανά άξονα. Το δεύτερο εργαλείο επιτρέπει την επιλογή τμήματος του σχεδίου που πρέπει να αναπαράγει η μηχανή, τον ορισμό των παραμέτρων κίνησης και τη δημιουργία G Code με κριτήριο είτε τη βέλτιστη διαδρομή είτε τη σειρά που επιλέγει ο χρήστης. Το τρίτο εργαλείο αναλαμβάνει τον υπολογισμό των παλμών ανά άξονα που πρέπει να αποσταλούν στον ελεγκτή του κινητήρα για κάθε εντολή του G Code, με κριτήριο την επιλεγμένη μηχανή. Το τέταρτο εργαλείο παρέχει τη δυνατότητα επιλογής πρωτοκόλλου επικοινωνίας και ορισμού των παραμέτρων του στην πλευρά του host υπολογιστή. Το πέμπτο εργαλείο επιτρέπει την οδήγηση της κεφαλής στο σημείο εκκίνησης που έχει οριστεί στο σχέδιο, ώστε να επιτευχθεί σωστή στοίχιση σε σχέση με τη θέση του δοκιμίου στον φυσικό χώρο.

Το δεύτερο μέρος αφορά την ανάπτυξη του FPGA ελεγκτή σε γλώσσα Verilog και αποτελείται από πέντε υποσυστήματα. Ο δέκτης σειριακής επικοινωνίας υλοποιεί ασύγχρονη σειριακή λήψη δεδομένων από τον host υπολογιστή, ενώ ο αντίστοιχος αποστολέας υλοποιεί την ασύγχρονη αποστολή σήματος επιβεβαίωσης προς αυτόν. Ο αποκωδικοποιητής πακέτων συναρμολογεί τα εισερχόμενα bytes της σειριακής επικοινωνίας και διακρίνει δύο τύπους πακέτων: το πακέτο αρχικοποίησης που μεταφέρει τις παραμέτρους της μηχανής και το πακέτο κίνησης που περιέχει τον τύπο κίνησης, τα βήματα ανά άξονα και την ταχύτητα. Η γεννήτρια παλμών παράγει τα σήματα STEP και DIR για τους οδηγούς των κινητήρων ανά άξονα, διασφαλίζοντας την παράλληλη και συντονισμένη κίνηση μεταξύ τους. Τέλος, ο ελεγκτής κεφαλής laser ελέγχει την ενεργοποίησή της με βάση τον τύπο κίνησης που λαμβάνεται από τον αποκωδικοποιητή πακέτων.

Λέξεις-κλειδιά: CNC, G Code, CAD, CAM, AutoLISP, Γραμμική και Κυκλική Παρεμβολή, FPGA, Verilog

FPGA-Driven Computerize Numerical Control Machine

Iakovou Petros

Abstract

This thesis is the final project of the MSc program in Applied Electronic Systems at the Department of Information and Electronic Engineering of the International Hellenic University. It focuses on the development of a general-purpose FPGA-based CNC machine controller, designed with adaptability to machines of varying mechanical characteristics.

The implementation of the system is divided into two parts. The first concerns the development of CAD-CAM tools in AutoLISP. The first tool enables the registration of machine characteristics, such as motor type, motion transmission method, pitch, and driver microstepping mode, allowing parameter configuration per axis. The second tool generates the G-code of a selected part from the drawing, incorporating parameters such as machining, cutting, and rapid-movement speeds, and supports two modes: path optimization or user-defined execution order. The third tool calculates the required pulses per axis, translating the generated G-code into motion commands based on the selected machine configuration. A fourth tool provides configuration of communication protocols and parameters, and includes a UI to align physical machine space with the digital design by positioning the tool head at the defined starting point.

The second part concerns the development of an FPGA controller in Verilog, which consists of the following modules: a UART receiver and transmitter for serial communication, a packet decoder that handles initialization packets and motion packets, a step/dir pulse generator for coordinated multi-axis motor control, and a laser controller that activates the laser during machining and disables it during rapid movements.

The system demonstrates a modular and scalable architecture, enabling efficient CNC control across different machine configurations while integrating design, toolpath generation, and hardware-level execution.

Keywords: CNC, G-code, CAD, CAM, AutoLISP, Linear and Circular Interpolation, FPGA, Verilog.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

ACK Acknowledgement (byte επιβεβαίωσης λήψης)

ARM Advanced RISC Machine

AVR Alf and Vegard's RISC processor (αρχιτεκτονική Atmel)

CAD Computer-Aided Design

CAM Computer-Aided Manufacturing

CCW Counter-Clockwise (αντίστροφη φορά περιστροφής)

CISC Complex Instruction Set Computer

CNC Computerized Numerical Control

CTC Clear Timer on Compare

CW Clockwise (φορά ωρολογίου)

DDA Digital Differential Analyzer

DIR Direction (σήμα φοράς κινητήρα)

DXF Drawing Exchange Format (μορφή αρχείου AutoCAD)

FIFO First In – First Out

FPGA Field-Programmable Gate Array

GPIO General-Purpose Input/Output

HDL Hardware Description Language

IC Integrated Circuit

ISO International Organization for Standardization

LED Light-Emitting Diode

LSB Least Significant Bit

MSB Most Significant Bit

NC Numerical Control

OCR Output Compare Register (καταχωρητής σύγκρισης AVR)

PCB Printed Circuit Board

PWM Pulse Width Modulation

RTL Register-Transfer Level

SoC System on Chip

UART Universal Asynchronous Receiver/Transmitter

UI User Interface

USB Universal Serial Bus

VHDL VHSIC Hardware Description Language

Κεφάλαιο 1ο: Εισαγωγή

1.1 Κίνητρα και αντικείμενο εργασίας

Οι μηχανές αριθμητικού ελέγχου (CNC), αποτελούν αναπόσπαστο μέρος της σύγχρονης βιομηχανικής παραγωγής. Κάθε μονάδα CNC περιλαμβάνει το σύστημα ελέγχου που είναι υπεύθυνο για τη μετατροπή των εντολών του κώδικα G στα κατάλληλα σήματα οδήγησης στους κινητήρες. Η αρχιτεκτονική του συστήματος ελέγχου καθορίζει την ακρίβεια, την ταχύτητα και την αξιοπιστία ολόκληρου του συστήματος.

Η παρούσα εργασία έχει ως αφετηρία προγενέστερο project [1] στο οποίο μελετήθηκε, σχεδιάστηκε και κατασκευάστηκε μια CNC μηχανή δύο αξόνων. Στο project αυτό για τον έλεγχο της κίνησης χρησιμοποιήθηκε ο μικροελεγκτής ATmega328P (αρχιτεκτονική AVR, 8-bit). Κατά την λειτουργία αυτής της μηχανής ο host υπολογιστής υπολογίζει των αριθμό των παλμών ανά άξονα με βάση τον κώδικα G, εφαρμόζοντας έναν αλγόριθμο παρεμβολής. Τα δεδομένα αποστέλλονται μέσω σειριακής επικοινωνίας UART στον μικροελεγκτή, ο οποίος με την σειρά του παράγει τους παλμούς για την οδήγηση των βηματικών κινητήρων μέσω των drivers.

Κατά την αξιολόγηση του συστήματος διαπιστώθηκαν ορισμένοι περιορισμοί που αποδίδονται στη φύση της σειριακής εκτέλεσης εντολών ενός μικροελεγκτή. Συγκεκριμένα, ο ATmega328P διαθέτει δύο 8-bit timers (timer0 και timer2) και έναν 16-bit timer (timer1), οι οποίοι για τις ανάγκες της εφαρμογής ρυθμίστηκαν σε λειτουργία CTC (Clear Timer on Compare) για την παραγωγή παλμών σε συγκεκριμένη συχνότητα. Η κίνηση στον άξονα X ελέγχεται μέσω του timer0 (ακροδέκτης OC0A – PD6) και στον άξονα Y μέσω του timer2 (ακροδέκτης OC2B – PB3). Παρότι οι δύο timers λειτουργούν ανεξάρτητα, η ρύθμιση των καταχωρητών σύγκρισης (OCR0A, OCR2A) και η εκτέλεση των αντίστοιχων ρουτινών των interrupts γίνεται σειριακά από τον επεξεργαστή, με αποτέλεσμα να μην επιτυγχάνεται πραγματικός παράλληλος έλεγχος μεταξύ των αξόνων.

Ένα δεύτερο ζήτημα αφορά την αστάθεια στην περιοδικότητα των παλμών (jitter). Εφόσον ο μικροελεγκτής εκτελεί ταυτόχρονα πολλαπλές εργασίες (λήψη δεδομένων μέσω UART, υπολογισμό μετατόπισης, interrupts) το χρονικό διάστημα για την εκτέλεση τους δεν είναι σταθερό. Η διακύμανση αυτή εισάγει μεταβολές στον χρόνο μεταξύ διαδοχικών παλμών, οι οποίες μεταφράζονται σε ανομοιομορφία στην ταχύτητα κίνησης και κατ' επέκταση σε απόκλιση από την επιθυμητή τροχιά. Σε βιομηχανικές εφαρμογές, όπου η ακρίβεια στην τροχιά αποτελεί κρίσιμο παράγοντα ποιότητας, η χρονική αυτή καθυστέρηση δεν είναι αποδεχτή.

Τρίτος περιορισμός είναι η δυσκολία προσαρμογής του ελεγκτή σε μηχανές διαφορετικών χαρακτηριστικών. Στην υλοποίηση με μικροελεγκτή, κάθε αλλαγή στον αριθμό αξόνων, στον τύπο του κινητήρα ή στη μέθοδο μετάδοσης κίνησης απαιτούσε τροποποίηση τόσο στο κύκλωμα (PCB) όσο και στο firmware. Αυτό καθιστά τη λύση δύσκαμπτη και αυξάνει τον χρόνο ανάπτυξης κάθε φορά που το σύστημα πρέπει να υποστηρίξει μια μηχανή διαφορετικών χαρακτηριστικών ή μια προσθήκη στην ίδια την μηχανή.

Η τεχνολογία των FPGA (Field Programmable Gate Array) προσφέρει μια εναλλακτική προσέγγιση προκυμμένου να ξεπεραστούν οι παραπάνω περιορισμοί. Σε ένα FPGA, η λογική δεν εκτελείται σειριακά μέσω ενός σετ εντολών, αλλά απευθείας σε επίπεδο hardware. Η κάθε επιμέρους λειτουργία (παραγωγή παλμών, αποκωδικοποίηση πακέτων, σειριακή επικοινωνία κ.α.) μπορεί να «ανατεθεί» σε

ξεχωριστό κύκλωμα που να λειτουργεί παράλληλα με τα υπόλοιπα. Κατά συνέπεια, η οδήγηση πολλαπλών αξόνων γίνεται ταυτόχρονα σε επίπεδο υλικού, χωρίς να απαιτείται εναλλαγή μεταξύ εργασιών και χωρίς τον κίνδυνο καθυστέρησης. Η χρονική ακρίβεια στην παραγωγή παλμών εξαρτάται αποκλειστικά από τη συχνότητα του ρολογιού του FPGA.

Σύγχρονες ερευνητικές εργασίες επιβεβαιώνουν τα πλεονεκτήματα των αρχιτεκτονικών FPGA στον έλεγχο κίνησης CNC μηχανών. Οι Santos et al. (2018) παρουσίασαν ένα σύστημα ελέγχου κίνησης βασισμένο σε FPGA, αναδεικνύοντας τη δυνατότητα πραγματικής παράλληλης επεξεργασίας και τον ακριβή χρονισμό ως βασικά πλεονεκτήματα έναντι λύσεων με μικροελεγκτή [2]. Αντίστοιχα, οι Zhang et al. (2015) πρότειναν μια αρχιτεκτονική CNC βασισμένη σε FPGA, εστιάζοντας στην ευελιξία αναδιαμόρφωσης και στην υψηλή ακρίβεια που επιτυγχάνεται μέσω αποκλειστικών κυκλωμάτων παραγωγής παλμών [3].

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η ανάπτυξη ενός ολοκληρωμένου συστήματος ελέγχου μηχανής CNC, στο οποίο ο ελεγκτής κίνησης υλοποιείται με FPGA σε γλώσσα Verilog. Η σχεδιασμός του συστήματος στοχεύει στη δημιουργία αρχιτεκτονικής, ικανή να προσαρμόζεται σε μηχανές διαφορετικών μηχανολογικών χαρακτηριστικών, χωρίς να απαιτείται αλλαγή στο υλικό ή εκ νέου σύνθεση του κυκλώματος. Βασικό στοιχείο αυτής αρχιτεκτονικής είναι η δημιουργία ενός πακέτου αρχικοποίησης (init packet) που περιέχει τις βασικές παραμέτρους της μηχανής (τύπος κινητήρα, βήμα, μέθοδος μετάδοσης κίνησης, microstepping) και αποστέλλεται από τον host υπολογιστή πριν την έναρξη κάθε κατεργασίας.

Παράλληλα με τον ελεγκτή FPGA, η εργασία περιλαμβάνει την ανάπτυξη εργαλείων λογισμικού CAM σε γλώσσα προγραμματισμού AutoLISP, τα οποία εκτελούνται εντός του σχεδιαστικού λογισμικού και εξυπηρετούν: την καταχώρηση μοντέλων μηχανών και των χαρακτηριστικών τους, την παραγωγή κώδικα G από το σχέδιο που έχει επιλεχθεί, τον υπολογισμό των απαιτούμενων παλμών ανά άξονα με βάση τα μηχανολογικά δεδομένα, και τη γεωμετρική ταύτιση μεταξύ του ψηφιακού σχεδίου και του φυσικού χώρου εργασίας της μηχανής. Η προσέγγιση αυτή μετατοπίζει την παραμετροποίηση από το επίπεδο του firmware στο επίπεδο του λογισμικού CAM, επιτρέποντας στον ίδιο ελεγκτή να εξυπηρετεί μηχανές με διαφορετικά κατασκευαστικά χαρακτηριστικά.

Συνοψίζοντας, τα κίνητρα για την ανάπτυξη της εργασίας είναι: η βελτίωση της απόδοσης μιας προγενέστερης μηχανής CNC, αντικαθιστώντας τον έλεγχο με μικροελεγκτή από έναν ελεγκτή βασισμένο σε FPGA, ο σχεδιασμός μιας γενικευμένης αρχιτεκτονικής ελεγκτή που να μπορεί να προσαρμόζεται σε διαφορετικές μηχανές χωρίς αλλαγές υλικού, και η εξοικείωση με τις αρχές σύνθεσης ψηφιακών κυκλωμάτων σε FPGA.

1.2 Βιβλιογραφική ανασκόπηση

Οι πρώτοι ελεγκτές των μηχανών αριθμητικού ελέγχου (1950) βασίζονταν σε αναλογικά κυκλώματα όπου η υλοποίηση του ελέγχου γινόταν μέσω πολύπλοκης καλωδίωσης. Η εισαγωγή των μικροεπεξεργαστών στη δεκαετία του 1970 οδήγησε σε συστήματα όπου το software αναλαμβάνει τον έλεγχο, επιλύοντας αλγόριθμους παρεμβολής που εκτελούνται σειριακά μέσω εντολών. Η αρχιτεκτονική αυτή αποτελεί ακόμη τη βάση πολλών εμπορικών ελεγκτών, ιδίως σε εφαρμογές χαμηλού και μεσαίου κόστους.

Σε μη βιομηχανικές εφαρμογές μηχανών CNC, οι μικροελεγκτές αρχιτεκτονικής AVR (όπως ο ATmega328P) και ARM αποτελούν δημοφιλείς επιλογές λόγω χαμηλού κόστους, ευκολίας

προγραμματισμού και πλήθους διαθέσιμων βιβλιοθηκών. Υπάρχουν αρκετές πλατφόρμες ανοιχτού κώδικα που βασίζονται σε τέτοιους μικροελεγκτές και εξυπηρετούν ικανοποιητικά εφαρμογές χαμηλών απαιτήσεων. Ωστόσο, η φύση της αρχιτεκτονικής του μικροελεγκτή (σειριακή εκτέλεση εντολών) θέτει περιορισμούς στον πραγματικό παράλληλο έλεγχο πολλαπλών αξόνων, στη σταθερότητα της περιοδικότητας παλμών (jitter) και στη δυνατότητα κλιμάκωσης σε πιο σύνθετες μηχανές [2]. Η προγενέστερη εργασία [1] ανέδειξε πρακτικά αυτούς τους περιορισμούς κατά τη λειτουργία μηχανής δύο αξόνων με ελεγκτή ATmega328P, επιβεβαιώνοντας ότι η εξυπηρέτηση πολλαπλών interrupts, και η ταυτόχρονη διαχείριση σειριακής επικοινωνίας έχει αποτέλεσμα τον κακό χρονισμό στην παραγωγή παλμών.

Η τεχνολογία FPGA (Field Programmable Gate Array) αποτελεί μια εναλλακτική προσέγγιση στον σχεδιασμό ελεγκτών κίνησης, η οποία έχει προσελκύσει ερευνητικό ενδιαφέρον τα τελευταία χρόνια. Σε αντίθεση με τους μικροελεγκτές, ένα FPGA δεν εκτελεί σειριακά ένα σύνολο εντολών, αλλά υλοποιεί κάθε λειτουργία ως ξεχωριστό κύκλωμα σε επίπεδο υλικού. Η ιδιότητα αυτή επιτρέπει τον πραγματικό παραλληλισμό: κάθε άξονας οδηγείται από ανεξάρτητο κύκλωμα παραγωγής παλμών, ενώ ταυτόχρονα εκτελείται η αποκωδικοποίηση δεδομένων και η σειριακή επικοινωνία. Ο χρονισμός καθορίζεται αποκλειστικά από τη συχνότητα του ρολογιού, εξαλείφοντας το πρόβλημα του jitter που παρατηρείται σε λύσεις βασισμένες σε λογισμικό [2], [3].

Οι πρώτες συστηματικές μελέτες για τη χρήση FPGA στον έλεγχο βηματικών κινητήρων εμφανίστηκαν στις αρχές της δεκαετίας του 2000. Οι Carrica, Funes και Gonzalez (2003) παρουσίασαν έναν ελεγκτή βηματικού κινητήρα υλοποιημένο σε FPGA, ο οποίος επέτρεπε δεκαπλάσια μείωση του χρόνου υπολογισμού σε σχέση με συμβατικούς αλγορίθμους, επιτυγχάνοντας ταχύτητες λειτουργίας που υπερέβαιναν τα όρια παραδοσιακών προσεγγίσεων και μάλιστα χωρίς τη χρήση timers, οι οποίοι αποτελούν κρίσιμο στοιχείο στα συμβατικά συστήματα [4]. Η εργασία τους έδειξε ότι ένας αλγόριθμος υλοποιημένος σε hardware μπορούσε να λειτουργήσει σε συχνότητες ρολογιού έως 80 MHz, εκτελώντας μετατοπίσεις 12.000 βημάτων με υψηλή αποδοτικότητα. Αντίστοιχα, οι Jeon και Kim (2002) πρότειναν ένα FPGA κύκλωμα επιτάχυνσης και επιβράδυνσης ειδικά σχεδιασμένο για βιομηχανικά ρομπότ και μηχανές CNC, αναδεικνύοντας τη δυνατότητα υλοποίησης τραπεζοειδών προφίλ ταχύτητας απευθείας σε υλικό [5].

Όσον αφορά τον ελέγχου πολλών αξόνων, η εργασία των Cho, Le και Jeon (2009) αποτελεί σημαντικό σημείο αναφοράς. Οι ερευνητές ανέπτυξαν ένα ολοκληρωμένο κύκλωμα ελέγχου κίνησης πολλαπλών αξόνων σε FPGA, το οποίο ενσωμάτωνε λειτουργίες όπως δημιουργία προφίλ ταχύτητας, υπολογισμό παρεμβολής, αντίστροφη κινηματική, έλεγχο PID και μέτρηση ανάδρασης, υλοποιημένα εξολοκλήρου σε VHDL. Η πειραματική αξιολόγηση σε ρομποτικό βραχίονα SCARA τεσσάρων βαθμών ελευθερίας επιβεβαίωσε ότι η προσέγγιση επιτυγχάνει υψηλό ρυθμό δειγματοληψίας (10 μs), ακρίβεια, ευελιξία και χαμηλή κατανάλωση ισχύος [6]. Η σημασία αυτής της εργασίας έγκειται στο ότι απέδειξε πως ένα και μόνο FPGA μπορεί να αντικαταστήσει ολόκληρα συστήματα που παραδοσιακά απαιτούσαν συνδυασμό DSP, μικροεπεξεργαστή και πρόσθετων κυκλωμάτων.

Μια ιδιαίτερα ενδιαφέρουσα κατεύθυνση στη βιβλιογραφία αφορά την ενσωμάτωση επεξεργασίας κώδικα G απευθείας σε FPGA. Οι Saifee και Mehta (2016) σχεδίασαν και υλοποίησαν έναν ελεγκτή τόρνου CNC συμβατό με κώδικα G, ο οποίος βασίζεται σε έναν CISC επεξεργαστή ειδικού σκοπού, υλοποιημένο σε FPGA Artix 7 της Xilinx. Ο ελεγκτής υποστήριζε εντολές γραμμικής και κυκλικής παρεμβολής καθώς και γρήγορης μετακίνησης, οι οποίες υλοποιούνταν ως συνεπεξεργαστές (co-processors) του κεντρικού επεξεργαστή G-code [7]. Η αρχιτεκτονική αυτή επέδειξε ότι η υλοποίηση σύνθετων αλγορίθμων παρεμβολής σε υλικό παρέχει εγγυήσεις πραγματικού χρόνου που δεν είναι

εφικτές μέσω σειριακής εκτέλεσης σε μικροεπεξεργαστή, ιδίως όταν ο επεξεργαστής επιφορτίζεται ταυτόχρονα με πολλαπλές εργασίες.

Ο Fei και οι συνεργάτες του (2011) πρότειναν μια υβριδική αρχιτεκτονική που συνδυάζει επεξεργαστή ARM με FPGA σε ενσωματωμένη συσκευή αριθμητικού ελέγχου, όπου ο ARM αναλαμβάνει τη διαχείριση κώδικα G, ενώ το FPGA υλοποιεί τους αλγορίθμους παρεμβολής και ελέγχου κίνησης [8]. Η κατανομή εργασιών μεταξύ λογισμικού και υλικού αποτελεί κοινό μοτίβο στη σύγχρονη βιβλιογραφία και αντικατοπτρίζει την ανάγκη αξιοποίησης των πλεονεκτημάτων κάθε τεχνολογίας.

Ένα ακόμη ζήτημα που απασχολεί τη βιβλιογραφία είναι η δημιουργία προφίλ ταχύτητας κατάλληλων για ομαλή κίνηση χωρίς απώλεια βημάτων. Στις λύσεις με μικροελεγκτή, ο υπολογισμός αυτών των προφίλ σε πραγματικό χρόνο αποτελεί υπολογιστική πρόκληση, καθώς απαιτεί αριθμητικές πράξεις κινητής υποδιαστολής σε κάθε κύκλο παλμού. Πρόσφατες εργασίες προτείνουν υβριδικές μεθόδους υπολογισμού σε πλατφόρμες FPGA, οι οποίες επιτυγχάνουν υψηλή ευελιξία και χαμηλή ταλάντωση [9].

Στο ευρύτερο πλαίσιο ελεγκτών CNC βασισμένων σε FPGA, δύο εργασίες σχετίζονται άμεσα με το αντικείμενο της παρούσας διπλωματικής. Οι Santos, Ferreira και Silva (2018) παρουσίασαν ένα σύστημα ελέγχου κίνησης για μηχανές CNC υλοποιημένο εξολοκλήρου σε FPGA, αναδεικνύοντας ως βασικά πλεονεκτήματα τον πραγματικό παραλληλισμό στην επεξεργασία και τον χρονισμό σε σύγκριση με λύσεις μικροελεγκτή [2]. Οι Zhang, Li και Wang (2015) πρότειναν μια αρχιτεκτονική CNC βασισμένη σε FPGA, στην οποία η ευελιξία αναδιαμόρφωσης και η υψηλή ακρίβεια παραγωγής παλμών αποτελούσαν τα κεντρικά χαρακτηριστικά σχεδιασμού [3]. Και οι δύο εργασίες επιβεβαιώνουν ότι η μεταφορά κρίσιμων λειτουργιών ελέγχου κίνησης από το επίπεδο λογισμικού στο επίπεδο υλικού αποτελεί μια ώριμη και αποτελεσματική στρατηγική.

Ένα στοιχείο που δεν καλύπτεται επαρκώς στην υπάρχουσα βιβλιογραφία είναι η ολοκληρωμένη αντιμετώπιση του ζητήματος της παραμετροποίησης. Πώς δηλαδή ο ίδιος ελεγκτής θα μπορούσε να εξυπηρετήσει μηχανές με διαφορετικά μηχανολογικά χαρακτηριστικά χωρίς τροποποίηση του υλικού ή σύνθεση του κυκλώματος από την αρχή. Οι περισσότερες εργασίες αντιμετωπίζουν τον ελεγκτή FPGA ως αυτόνομη μονάδα, χωρίς να εξετάζουν τη διασύνδεσή του με εργαλεία λογισμικού CAD/CAM που θα μπορούσαν να μεταφέρουν τη λογική της παραμετροποίησης στο επίπεδο εφαρμογής. Τυπικά, η αλλαγή μηχανής συνεπάγεται τροποποίηση σε παραμέτρους που είναι ενσωματωμένες στον κώδικα HDL ή στο firmware, γεγονός που απαιτεί εκ νέου σύνθεση και προγραμματισμό του FPGA.

Η παρούσα εργασία τοποθετείται σε αυτό ακριβώς το κενό. Σε αντίθεση με τις προαναφερθείσες προσεγγίσεις, η αρχιτεκτονική που προτείνεται εδώ μεταφέρει την παραμετροποίηση από το επίπεδο του hardware σε ένα πακέτο αρχικοποίησης (init packet) που αποστέλλεται από τον host υπολογιστή στην αρχή κάθε λειτουργίας. Παράλληλα, η ανάπτυξη εξειδικευμένων εργαλείων CAD-CAM σε γλώσσα AutoLISP (τα οποία αναλαμβάνουν την καταχώριση χαρακτηριστικών μηχανής, την παραγωγή κώδικα G, τον υπολογισμό παλμών ανά άξονα και τη γεωμετρική ταύτιση σχεδίου με φυσικό χώρο εργασίας) δημιουργεί ένα ολοκληρωμένο σύστημα στο οποίο ο ελεγκτής FPGA λειτουργεί ως γενικού σκοπού μονάδα εκτέλεσης κίνησης. Η προσέγγιση αυτή συνδυάζει τα πλεονεκτήματα του hardware-based ελέγχου με την ευελιξία που προσφέρει η παραμετροποίηση μέσω λογισμικού, και χωρίς τα μειονεκτήματα που εντοπίζονται στις υπάρχουσες λύσεις.

1.3 Συνεισφορά της εργασίας

Από την ανασκόπηση της βιβλιογραφίας προκύπτει ότι οι υπάρχουσες εργασίες εστιάζουν κατά κανόνα στον σχεδιασμό του ελεγκτή FPGA ως αυτόνομης μονάδας, ενώ η σύνδεσή του με το ευρύτερο περιβάλλον σχεδιασμού και παρασκευής (CAD/CAM) παραμένει εκτός εστίασης. Η παρούσα εργασία επιχειρεί να καλύψει αυτό το κενό, αντιμετωπίζοντας το σύστημα ελέγχου CNC ως ενιαία αλυσίδα που ξεκινά από το σχέδιο του τεμαχίου στο CAD και καταλήγει στην παραγωγή παλμών step/dir σε επίπεδο υλικού.

Η βασική συνεισφορά εντοπίζεται στον τρόπο που κατανέμονται οι αρμοδιότητες μεταξύ λογισμικού και υλικού. Αντί να ενσωματώνονται οι παράμετροι της μηχανής στον κώδικα HDL (όπως γίνεται στην πλειονότητα των δημοσιευμένων αρχιτεκτονικών) εδώ ο ελεγκτής FPGA σχεδιάστηκε ώστε να δέχεται τις παραμέτρους κατά τη φάση εκτέλεσης, μέσω ενός πακέτου αρχικοποίησης (init packet) που αποστέλλεται από τον host υπολογιστή. Το πακέτο αυτό μεταφέρει τα μηχανολογικά χαρακτηριστικά κάθε μηχανής (τύπο κινητήρα, γωνία βήματος, μέθοδο μετάδοσης κίνησης, υποδιαίρεση βήματος) και το FPGA τα αποθηκεύει σε εσωτερικούς καταχωρητές πριν ξεκινήσει η αποστολή εντολών κίνησης. Με αυτόν τον τρόπο, η αλλαγή μηχανής δεν απαιτεί εκ νέου σύνθεση ούτε προγραμματισμό του FPGA, αλλά μόνο αποστολή νέου πακέτου αρχικοποίησης.

Η δεύτερη διάσταση της συνεισφοράς αφορά τα εργαλεία CAD-CAM που αναπτύχθηκαν σε γλώσσα AutoLISP και εκτελούνται μέσα στο περιβάλλον CAD. Τα εργαλεία αυτά καλύπτουν ολόκληρη τη ροή εργασίας, από την καταχώριση και αποθήκευση μοντέλων μηχανής (μέσω φόρμας OpenDCL) έως την παραγωγή κώδικα G από το σχέδιο, τον υπολογισμό των παλμών ανά άξονα με βάση τα μηχανολογικά δεδομένα και τη γεωμετρική ταύτιση του ψηφιακού σχεδίου με τον φυσικό χώρο εργασίας.

Στο σκέλος του υλικού, ο ελεγκτής FPGA υλοποιήθηκε σε Verilog και αποτελείται από πέντε διασυνδεδεμένα modules: τον δέκτη UART (uart_rx), τον πομπό UART (uart_tx), τον αποκωδικοποιητή πακέτων (packet_decoder) που μπορεί να επεξεργαστεί πακέτα αρχικοποίησης (9 bytes) και πακέτα κίνησης (11 bytes), τη γεννήτρια παλμών (step_gen) που υλοποιεί γραμμική παρεμβολή μεταξύ των αξόνων και τον ελεγκτή laser (laser_ctrl) που ενεργοποιεί ή απενεργοποιεί την κεφαλή ανάλογα με τον τύπο κίνησης (κατεργασία ή γρήγορη μεταφορά). Η αρθρωτή δομή διευκολύνει τόσο τη δοκιμή μεμονωμένων modules όσο και την μελλοντική επέκταση σε περισσότερους άξονες ή πρόσθετες λειτουργίες.

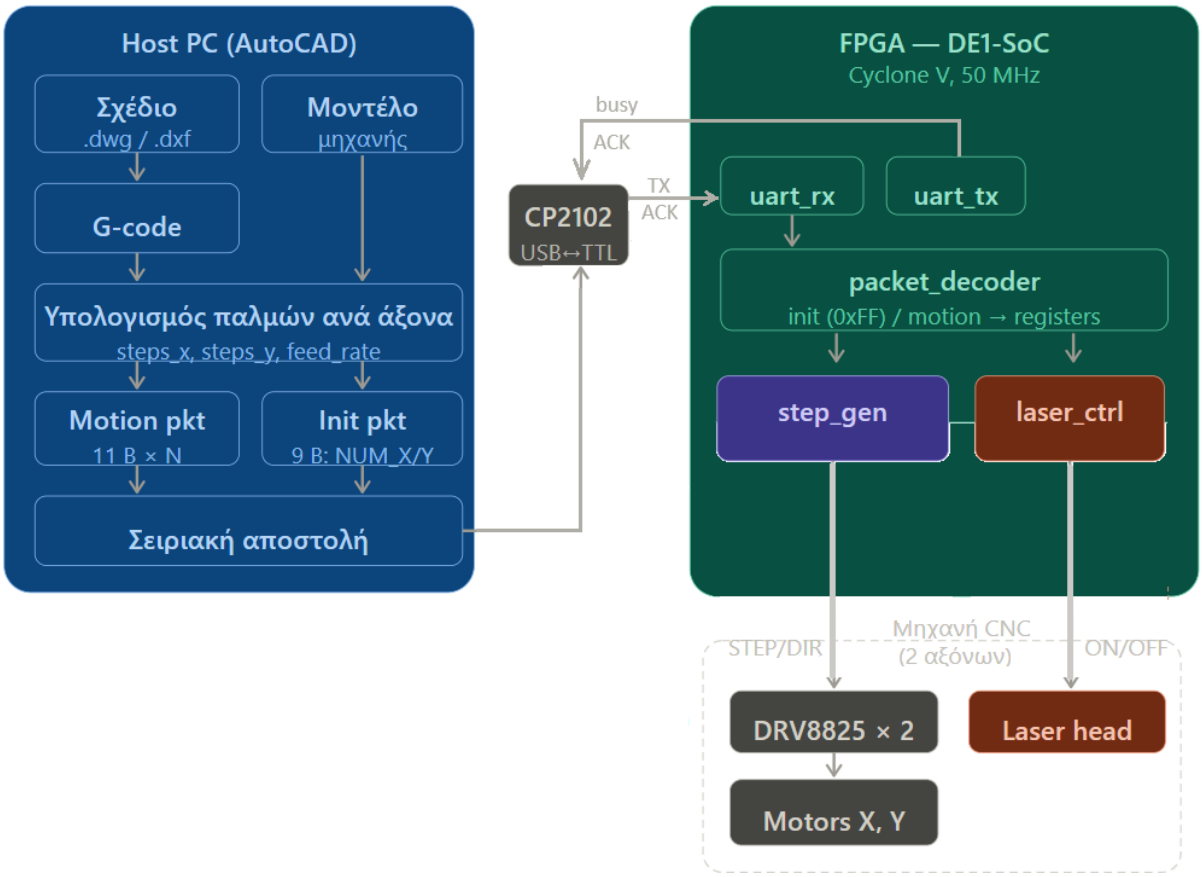
Ένα επιπλέον στοιχείο που διαφοροποιεί την παρούσα εργασία είναι ο συγκριτικός χαρακτήρας της. Δεδομένου ότι η ίδια μηχανή λειτούργησε αρχικά με ελεγκτή ATmega328P [1] και στη συνέχεια με τον ελεγκτή FPGA που αναπτύχθηκε εδώ, καθιστά δυνατή την απευθείας σύγκριση μεταξύ των δύο αρχιτεκτονικών στο ίδιο μηχανολογικό υπόστρωμα. Η σύγκριση καλύπτει χρονικό jitter στην παραγωγή παλμών, ακρίβεια τροχιάς, δυνατότητα παράλληλου ελέγχου αξόνων και ευκολία προσαρμογής σε νέα μηχανή. Πρόκειται για πρακτική αξιολόγηση σε πραγματικές συνθήκες λειτουργίας, η οποία συμπληρώνει τα θεωρητικά επιχειρήματα που εκτίθενται στη βιβλιογραφική ανασκόπηση.

Συνολικά, η εργασία δεν περιορίζεται στη σχεδίαση ενός ελεγκτή CNC σε FPGA. Η κεντρική ιδέα είναι ότι ο ελεγκτής αποτελεί τον τελευταίο κρίκο μιας αλυσίδας που ξεκινά από τον σχεδιαστικό χώρο του CAD, περνά από τη δημιουργία κώδικα G και τον υπολογισμό παλμών, και καταλήγει στη φυσική κίνηση της μηχανής. Η γνώση της μηχανής βρίσκεται στο λογισμικό και μεταδίδεται στον ελεγκτή κατά τη φάση εκτέλεσης (ο ελεγκτής δεν χρειάζεται να «γνωρίζει» εκ των προτέρων σε ποια μηχανή θα συνδεθεί). Αυτή η αρχιτεκτονική επιλογή, αν και φαινομενικά απλή, δεν εντοπίζεται στις

δημοσιευμένες εργασίες που εξετάστηκαν στην ενότητα 1.2, και αποτελεί κατά τη γνώμη μου το πιο ενδιαφέρον χαρακτηριστικό του συστήματος που παρουσιάζεται.

1.4 Αρχιτεκτονική συστήματος

Το σύστημα που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας αποτελείται από δύο κύρια τμήματα: το λογισμικό CAM, που εκτελείται στον υπολογιστή-host, και τον ελεγκτή κίνησης, που υλοποιείται σε FPGA. Η επικοινωνία μεταξύ τους πραγματοποιείται μέσω σειριακής σύνδεσης UART με τη χρήση μετατροπέα CP2102 USB-to-Serial. Στο Σχήμα 1.1 παρουσιάζεται σχηματικά η συνολική αρχιτεκτονική.



Σχήμα 1.1: Αρχιτεκτονική Συστήματος

Στην πλευρά του host υπολογιστή, η ροή εργασίας ξεκινά από το σχέδιο του τεμαχίου στο περιβάλλον CAD. Ο χρήστης επιλέγει τη γεωμετρία προς κατεργασία και τα εργαλεία CAM αναλαμβάνουν τη μετατροπή σε κώδικα G (G00 για γρήγορη μεταφορά, G01 για γραμμική κίνηση, G02/G03 για κυκλικά τόξα). Κάθε εντολή G αντιστοιχεί σε ένα ευθύγραμμο τμήμα ή ένα τμήμα τόξου, το οποίο στη συνέχεια μετατρέπεται σε αριθμό παλμών ανά άξονα. Ο υπολογισμός αυτός λαμβάνει υπόψη τα χαρακτηριστικά της συγκεκριμένης μηχανής (γωνία βήματος κινητήρα, υποδιαίρεση microstepping του driver, τύπο και βήμα του μηχανισμού μετάδοσης) τα οποία έχουν καταχωρηθεί εκ των προτέρων μέσω φόρμας OpenDCL.

Πριν ξεκινήσει η αποστολή εντολών κίνησης, ο host αποστέλλει ένα πακέτο αρχικοποίησης (init packet) μήκους 9 bytes. Το πρώτο byte είναι ο δείκτης 0xFF που σηματοδοτεί πακέτο αρχικοποίησης, και

ακολουθούν δύο τιμές uint32 (NUMERATOR_X, NUMERATOR_Y), οι οποίες εκφράζουν τη σχέση μεταξύ ρολογιού FPGA, μονάδων ταχύτητας και βημάτων ανά χιλιοστό για κάθε άξονα. Ο ελεγκτής αποθηκεύει τις τιμές αυτές σε εσωτερικούς καταχωρητές και τις χρησιμοποιεί στον υπολογισμό της περιόδου παλμών κατά την εκτέλεση κάθε εντολής κίνησης. Με αυτόν τον τρόπο, αν αλλάξει η μηχανή ή η ρύθμιση microstepping των drivers, αρκεί ένα νέο πακέτο αρχικοποίησης και όχι η εκ νέου σύνθεση του κυκλώματος.

Μετά την αρχικοποίηση, ο host αρχίζει να αποστέλλει πακέτα κίνησης (motion packets) μήκους 11 bytes. Κάθε πακέτο περιέχει τον τύπο κίνησης (motion type: 0 = G00, 1 = G01, 2 = G02, 3 = G03), τα βήματα ανά άξονα ως ακέραιους με πρόσημο (int32, MSB first) και το feed rate (uint16). Ο host στέλνει ένα πακέτο και περιμένει το byte επιβεβαίωσης (ACK = 0xAA) από τον ελεγκτή πριν στείλει το επόμενο. Ο μηχανισμός αυτός αποτρέπει την υπερχείλιση δεδομένων, δεδομένου ότι ο ελεγκτής δεν διαθέτει buffer πολλαπλών πακέτων.

Στην πλευρά του FPGA, η αρχιτεκτονική ακολουθεί αρθρωτή δομή σε πέντε modules γραμμένα σε Verilog, τα οποία διασυνδέονται στο top-level module. Ο δέκτης UART (*uart_rx*) για την λήψη των δεδομένων. Ο αποκωδικοποιητής πακέτων (*packet_decoder*) συναρμολογεί τα εισερχόμενα bytes σε πακέτα, διακρίνοντας τα σε πακέτα αρχικοποίησης (9 bytes, ξεκινά με 0xFF) και πακέτα κίνσης (11 bytes) μέσω του πρώτου byte. Ένας χρονιστής 100 ms μηδενίζει τη μηχανή καταστάσεων αν η λήψη ενός πακέτου δεν ολοκληρωθεί εντός αυτού του χρονικού ορίου, ώστε ένα κατεστραμμένο ή ελλιπές πακέτο να μην μπλοκάρει τον ελεγκτή.

Η γεννήτρια παλμών (*step_gen*) αποτελεί τον πυρήνα του ελεγκτή. Λαμβάνει τον αριθμό βημάτων ανά άξονα (steps_x, steps_y) και το feed rate, και υπολογίζει την περίοδο παλμών σε κύκλους ρολογιού: T = NUMERATOR / feed_rate. Για τη γραμμική παρεμβολή μεταξύ δύο αξόνων, η κεφαλή στην ονομαστική ταχύτητα ενώ η ταχύτητα των αξόνων προσαρμόζεται, ώστε και οι δύο να ολοκληρώσουν την κίνηση ταυτόχρονα.

Ο ελεγκτής laser (*laser_ctrl*) παρακολουθεί τον τύπο κίνησης που αποκωδικοποιεί ο *packet_decoder* και ενεργοποιεί την έξοδο laser μόνο κατά τη διάρκεια κινήσεων κατεργασίας (G01, G02, G03). Σε κινήσεις γρήγορης μετακίνησης (G00) η κεφαλή παραμένει απενεργοποιημένη. Τέλος, ο πομπός UART (*uart_tx*) αποστέλλει το byte ACK (0xAA) στον host μόλις η γεννήτρια παλμών ολοκληρώσει μια κίνηση. Ο host, με τη λήψη του ACK, προχωρά στην αποστολή του επόμενου πακέτου.

Η κεντρική σχεδιαστική επιλογή που διατρέχει ολόκληρη την αρχιτεκτονική είναι ο διαχωρισμός μεταξύ γνώσης της μηχανής και εκτέλεσης κίνησης. Τα εργαλεία CAM στον host γνωρίζουν τις ιδιαιτερότητες κάθε μηχανής (αριθμό βημάτων ανά χιλιοστό, πρόωση, γεωμετρία χώρου) και μεταδίδουν στον ελεγκτή μόνο αριθμούς (παλμούς, φορά, ταχύτητα). Ο ελεγκτής, από την πλευρά του, δεν χρειάζεται να γνωρίζει τι μηχανή οδηγεί· εκτελεί μια γενική λειτουργία παραγωγής παλμών. Η μόνη σχέση μεταξύ ελεγκτή και μηχανής είναι οι τιμές NUMERATOR που αποστέλλονται στο πακέτο αρχικοποίησης και αυτές προσδιορίζονται εξ ολοκλήρου από το λογισμικό.

Στα κεφάλαια που ακολουθούν παρουσιάζονται αναλυτικά τα επιμέρους τμήματα: η ανάπτυξη των εργαλείων CAM (Κεφάλαιο 3), η σειριακή επικοινωνία και ο μηχανισμός handshake (Κεφάλαιο 4), η σχεδίαση του κάθε module σε Verilog (Κεφάλαιο 5) και οι δοκιμές αξιολόγησης (Κεφάλαιο 6).

1.5 Δομή εργασίας

Η εργασία οργανώνεται σε επτά κεφάλαια, σε ένα παράρτημα κώδικα και τη βιβλιογραφία.

Κεφάλαιο 1 : Εισαγωγή

Παρουσιάζονται τα κίνητρα και το αντικείμενο της εργασίας. Γίνεται αναφορά στη σχετική βιβλιογραφία, και στη συνεισφορά σε σχέση με τις υπάρχουσες δημοσιεύσεις. Τέλος περιγράφετε η αρχιτεκτονική του συστήματος.

Κεφάλαιο 2 : Θεωρητικό Υπόβαθρο

Παρατίθεται το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση του συστήματος. Εξετάζονται οι βασικές αρχές λειτουργίας των μηχανών CNC, η δομή και οι εντολές του κώδικα G σύμφωνα με το πρότυπο ISO 6983, τα χαρακτηριστικά λειτουργίας βηματικών κινητήρων και οδηγών (drivers), καθώς και οι αλγόριθμοι γραμμικής και κυκλικής παρεμβολής. Ακολουθεί μια εισαγωγή στην τεχνολογία FPGA (αρχιτεκτονική, πλεονεκτήματα σε σχέση με μικροελεγκτές, εργαλεία σύνθεσης) και η περιγραφή του πρωτοκόλλου σειριακής επικοινωνίας UART.

Κεφάλαιο 3 : Ανάπτυξη λογισμικού CAM

Αφορά την ανάπτυξη των εργαλείων λογισμικού CAM. Περιγράφεται η αρχιτεκτονική των εργαλείων AutoLISP και της φόρμας OpenDCL για καταχώριση μοντέλων μηχανής, τον μηχανισμό παραγωγής κώδικα G από σχέδιο, με τις δύο εναλλακτικές στρατηγικές (βέλτιστης διαδρομής και σειρά επιλογής από τον χρήστη) και τον υπολογισμό των παλμών ανά άξονα με βάση τα μηχανολογικά δεδομένα, και τη γεωμετρική ταύτιση σχεδίου με το φυσικό δοκίμιο κατεργασίας.

Κεφάλαιο 4 : Επικοινωνία μεταξύ Host και ελεγκτή

Ανάλυση της σειριακής επικοινωνίας μεταξύ host και ελεγκτή. Περιγράφονται η φυσική σύνδεση μέσω του μετατροπέα CP2102, η δομή των δύο τύπων πακέτων (αρχικοποίησης και κίνησης), ο μηχανισμός handshake που ρυθμίζει τη ροή δεδομένων.

Κεφάλαιο 5 : Σχεδίαση και υλοποίηση ελεγκτή FPGA

Παρουσιάζεται η σχεδίαση και η υλοποίηση του ελεγκτή FPGA. Αρχικά περιγράφεται η πλατφόρμα ανάπτυξης DE1-SoC (Cyclone V) και το εργαλείο σύνθεσης Quartus II 13.1. Στη συνέχεια αναλύονται τα πέντε modules σε Verilog (*uart_rx*, *uart_tx*, *packet_decoder*, *step_gen*, *laser_ctrl*) και ο τρόπος ενοποίησής τους στο top module. Τεκμηριώνεται η διασύνδεση των ακροδεκτών και η αντιστοίχιση μέσω του Pin Planner.

Κεφάλαιο 6 : Δοκιμές και αξιολόγηση

Καλύπτει τις δοκιμές και την αξιολόγηση του συστήματος. Παρουσιάζονται η πλατφόρμα και η μεθοδολογία δοκιμών, οι μετρήσεις σε πραγματική μηχανή (ακρίβεια τροχιάς, χρονικό jitter) και η σύγκριση μεταξύ του ελεγκτή ATmega328P της προγενέστερης υλοποίησης [1] και του ελεγκτή FPGA που αναπτύχθηκε στην παρούσα εργασία.

Κεφάλαιο 7 : Συμπεράσματα και μελλοντικές επεκτάσεις

Συνοψίζονται τα συμπεράσματα και προτείνονται κατευθύνσεις μελλοντικής επέκτασης.

Παράρτημα Α

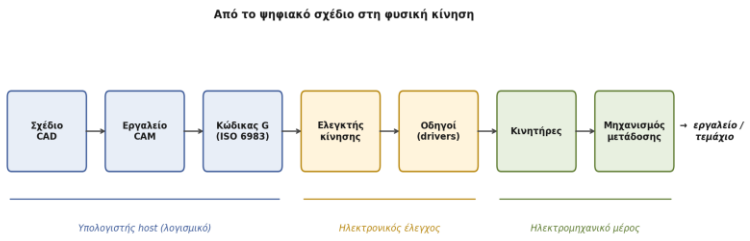
Ο πηγαίος κώδικας AutoLISP και Verilog.

Κεφάλαιο 2ο: Θεωρητικό Υπόβαθρο

2.1 Μηχανές CNC — αρχές λειτουργίας

Ο όρος μηχανή αριθμητικού ελέγχου (Computerized Numerical Control, CNC) αναφέρεται σε μια κατηγορία εργαλειομηχανών όπου η κίνηση των αξόνων και η ενεργοποίηση του εργαλείου κατεργασίας ελέγχονται από ένα πρόγραμμα αποτελούμενο από αριθμητικές εντολές. Η ουσιαστική διαφορά τους από τις συμβατικές εργαλειομηχανές δεν εντοπίζεται στον τρόπο που πραγματοποιείται η κατεργασία (αφαίρεση υλικού, χάραξη, κοπή κ.λπ.), αλλά στο επίπεδο όπου παρεμβαίνει ο χειριστής. Αντί να μετακινεί χειροκίνητα τους άξονες με μοχλούς, ο χειριστής συντάσσει (ή παράγει αυτόματα μέσω εργαλείων CAM) ένα πρόγραμμα κίνησης το οποίο εκτελεί η μηχανή. Η ορολογία προέρχεται από τη δεκαετία του 1950, όταν εμφανίστηκαν οι πρώτες εργαλειομηχανές αριθμητικού ελέγχου (NC), στις οποίες οι εντολές αναγνωρίζονταν από διάτρητες χάρτινες ταινίες. Η αντικατάσταση της ηλεκτρομηχανικής λογικής από επεξεργαστές, στη δεκαετία του 1970, οδήγησε στην προσθήκη του προθέματος *Computerized*, σηματοδοτώντας την εποχή των CNC συστημάτων όπως τα γνωρίζουμε σήμερα.

Παρά την ποικιλομορφία των εφαρμογών (κέντρα κατεργασίας και τόρνους, μηχανές κοπής laser, water-jet, τρισδιάστατους εκτυπωτές κ.α.) η αλυσίδα μετάδοσης πληροφορίας από το αρχικό σχέδιο έως τη φυσική κίνηση της κεφαλής παραμένει σε μεγάλο βαθμό κοινή. Στο πρώτο στάδιο, ο χρήστης σχεδιάζει το προς κατεργασία τεμάχιο σε λογισμικό CAD ή λαμβάνει έτοιμο σχέδιο σε ψηφιακή μορφή (DXF, DWG, STEP). Ένα εργαλείο CAM επεξεργάζεται τη γεωμετρία και παράγει τον κώδικα G (μια ακολουθία εντολών διατυπωμένων κατά το πρότυπο ISO 6983, η οποία περιγράφει τις διαδοχικές κινήσεις του εργαλείου). Ο κώδικας μεταφέρεται στον ελεγκτή της μηχανής και ερμηνεύεται εντολή προς εντολή. Για κάθε γραμμή του προγράμματος, ο ελεγκτής υπολογίζει (μέσω αλγορίθμου παρεμβολής) τις διακριτές μετατοπίσεις που πρέπει να εκτελέσει κάθε άξονας και παράγει τα αντίστοιχα σήματα οδήγησης (παλμούς step/dir στην περίπτωση βηματικών κινητήρων ή αναλογικά σήματα αναφοράς στην περίπτωση servo). Τα σήματα αυτά τροφοδοτούν τους οδηγούς (drivers), οι οποίοι παρέχουν την ηλεκτρική ισχύ ώστε να ενεργοποιηθούν οι κινητήρες. Η περιστροφική κίνηση των moter, μετατρέπεται από τον μηχανισμό μετάδοσης σε γραμμική κίνηση των αξόνων. Στο σχήμα 2.1 αποτυπώνεται η ροή αυτή.



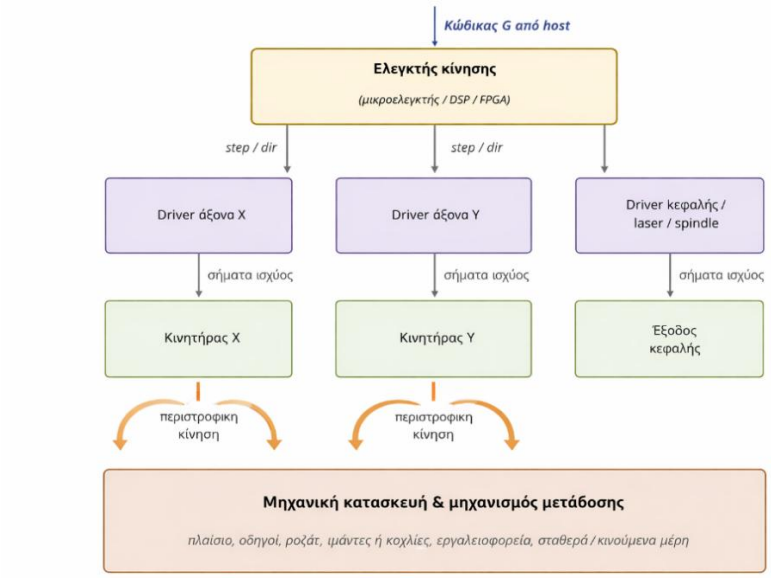
Σχήμα 2.1*: Αλυσίδα μετάδοσης πληροφορίας* από το ψηφιακό σχέδιο έως τη φυσική κίνηση της κεφαλής.

Από αρχιτεκτονικής σκοπιάς, μια τυπική CNC μηχανή αποτελείται από πέντε λειτουργικές βαθμίδες που συνδέονται διαδοχικά (Σχήμα 2.2). Στην κορυφή βρίσκεται ο ελεγκτής κίνησης (ο εγκέφαλος του

συστήματος) ο οποίος αναλαμβάνει την αποκωδικοποίηση των εντολών G, τον υπολογισμό της τροχιάς, την παραγωγή των σημάτων οδήγησης και τον συγχρονισμό με τα περιφερειακά (αισθητήρες, τερματικούς διακόπτες, βοηθητικές εξόδους όπως spindle, laser κ.α.). Η υλοποίησή του ποικίλλει: από μικροελεγκτή σε εφαρμογές χαμηλού κόστους, σε ψηφιακό επεξεργαστή σήματος (DSP) σε εφαρμογές μέσης κλίμακας, ή σε FPGA όπου απαιτείται πραγματικός παραλληλισμός (επιλογή στην οποία στηρίζεται η παρούσα εργασία).

Αμέσως μετά τον ελεγκτή τοποθετούνται οι οδηγοί κινητήρων (drivers), οι οποίοι μετατρέπουν τα ψηφιακά σήματα ελέγχου σε ηλεκτρική ισχύ ικανή να κινήσει τους κινητήρες. Σε βηματικούς κινητήρες, ο driver είναι υπεύθυνος για τη διαδοχική διέγερση των τυλιγμάτων αλλά και για την υποδιαίρεση του βασικού βήματος (microstepping). Σε servo κινητήρες, ο αντίστοιχος ρόλος αποδίδεται στον servo driver, ο οποίος στις περισσότερες περιπτώσεις υλοποιεί ένα εσωτερικό βρόχο ελέγχου ταχύτητας ή θέσης. Οι κινητήρες αποτελούν τον επόμενο κρίκο. Σε CNC χαμηλού και μεσαίου κόστους κυριαρχούν οι βηματικοί κινητήρες, λόγω της απλότητας οδήγησης σε ανοιχτό βρόχο και του χαμηλού κόστους ανά μονάδα ροπής. Όπου απαιτείται υψηλή δυναμική απόκριση και ταχύτητα (σε βιομηχανικά κέντρα κατεργασίας ή σε μηχανές μεγάλης μάζας) προτιμώνται servo κινητήρες με ενσωματωμένο encoder, που επιτρέπουν λειτουργία σε κλειστό βρόχο.

Η περιστροφική κίνηση των κινητήρων μεταφέρεται στους άξονες μέσω του μηχανισμού μετάδοσης. Δύο είναι τυπικά οι λύσεις: ο οδοντωτός ιμάντας με τροχαλία (timing belt) και ο κοχλίας ισχύος (lead ή ball screw). Η πρώτη επιτρέπει υψηλές ταχύτητες αλλά εμφανίζει μικρότερη ακρίβεια και περισσότερο διάκενο, ενώ η δεύτερη συμπεριφέρεται ακριβώς αντίστροφα (μεγάλη ακρίβεια, μικρότερες όμως ταχύτητες λειτουργίας). Επί της ουσίας πρόκειται για συμβιβασμό μεταξύ ταχύτητας και ακρίβειας, ο οποίος καθορίζεται από τις απαιτήσεις της εφαρμογής. Η μηχανική κατασκευή ολοκληρώνει το σύστημα: το πλαίσιο, οι οδηγοί, τα φορεία (carriers) και οι μηχανισμοί στερέωσης ορίζουν τη γεωμετρική σχέση μεταξύ των αξόνων και επηρεάζουν άμεσα τη δυσκαμψία του συνόλου.



Σχήμα 2.2: *Λειτουργικές βαθμίδες τυπικής CNC μηχανής.*

Οι υλοποιήσεις διαφοροποιούνται επίσης και ως προς την ύπαρξη ή μη ανάδρασης. Ένα σύστημα ανοιχτού βρόχου αποστέλλει εντολές κίνησης χωρίς να γνωρίζει αν η μηχανή τις ακολούθησε πιστά (η προσέγγιση αυτή είναι αποδεκτή στους βηματικούς κινητήρες, εφόσον η ροπή παραμένει εντός των ορίων του κινητήρα και δεν παρατηρείται απώλεια βημάτων). Ένα σύστημα κλειστού βρόχου, αντίθετα,

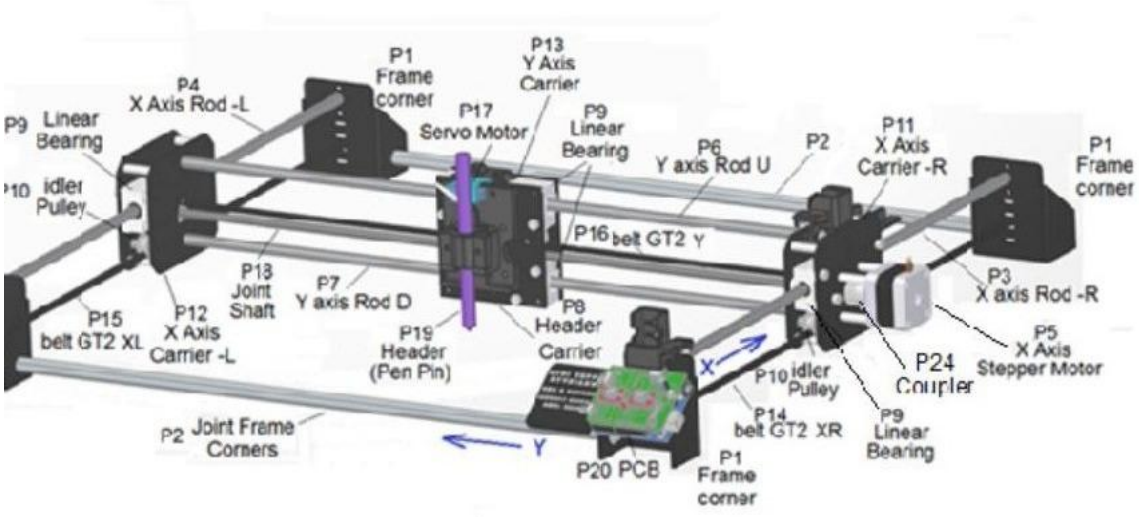
ενσωματώνει αισθητήρα θέσης (συνήθως οπτικό ή μαγνητικό encoder) που τροφοδοτεί τον ελεγκτή με την πραγματική θέση του άξονα. Με βάση το σφάλμα μεταξύ επιθυμητής και πραγματικής θέσης, ο ελεγκτής αναπροσαρμόζει διαρκώς το σήμα. Η λύση κλειστού βρόχου επιβάλλεται σε εφαρμογές υψηλών απαιτήσεων και ακρίβειας τροχιάς, αυξάνει όμως την πολυπλοκότητα και το κόστος (Suh κ.ά., 2008).

Κάθε CNC μηχανή λειτουργεί επίσης με βάση δύο τουλάχιστον συστήματα συντεταγμένων. Το σύστημα συντεταγμένων της μηχανής (Machine Coordinate System) έχει σταθερή αρχή που ταυτίζεται με μια αναγνωρίσιμη μηχανική θέση και εξαρτάται από τη γεωμετρία της κατασκευής. Το σύστημα συντεταγμένων του τεμαχίου (Work Coordinate System) όπου ορίζεται από τον χρήστη και αντιστοιχεί στη θέση του δοκιμίου επί της επιφάνειας εργασίας. Η ταύτιση των δύο συστημάτων (κεντράρισμα κεφαλής, αναζήτηση αρχικού σημείου αναφοράς) πραγματοποιείται είτε χειροκίνητα μέσω χειριστηρίου, είτε αυτόματα μέσω τερματικών διακοπτών τοποθετημένων στα φυσικά όρια της κίνησης.

Στον τύπο της κίνησης, οι CNC ελεγκτές διακρίνουν δύο βασικές κατηγορίες εντολών. Η γρήγορη μετακίνηση που εκτελείται με τη μέγιστη επιτρεπτή ταχύτητα της μηχανής και χρησιμοποιείται για μετακινήσεις όπου δεν γίνεται κατεργασία, όπως η μετάβαση ανάμεσα σε διαδοχικά τμήματα κατεργασίας. Η μετακίνηση με συγκεκριμένη ταχύτητα (πρόωση), η οποία επιλέγεται με βάση τις συνθήκες κατεργασίας. Επιπλέον, οι σύγχρονοι ελεγκτές υποστηρίζουν τόσο γραμμική παρεμβολή (όπου οι άξονες κινούνται συντονισμένα ώστε η συνισταμένη τροχιά να αποτελεί ευθεία) όσο και κυκλική παρεμβολή για τόξα.

Η CNC μηχανή που χρησιμοποιείται ως πλατφόρμα δοκιμών (πλαίσιο, κυλινδρικοί οδηγοί και γραμμικά έδρανα, μετάδοση κίνησης μέσω οδοντωτών ιμάντων GT2 με βήμα 2 mm) παρουσιάστηκε αναλυτικά στην προγενέστερη εργασία [1] και η συνολική διάταξη φαίνεται στο σχήμα 2.3. Στους δύο άξονες χρησιμοποιούνται βηματικοί κινητήρες 42BYGHW208 (NEMA17) σε ανοιχτό βρόχο, οδηγούμενοι από drivers DRV8825. Η μηχανή στην αρχική της μορφή χρησιμοποιήθηκε ως plotter με κεφαλή ένα στυλό. Η ενεργοποίηση της κεφαλής ελεγχόταν από έναν servo κινητήρα ενσωματωμένο στο φορείο του Y άξονα. Στην παρούσα διπλωματική διατηρείται η ίδια ακριβώς μηχανική υποδομή, αλλά αντί για στυλό η κεφαλή φέρει ένα laser module ισχύος 2,5 W, και ο servo κινητήρας ανύψωσης δεν χρησιμοποιείται (το laser ενεργοποιείται και απενεργοποιείται ψηφιακά).

Η ίδια μηχανή θα λειτουργήσει με την ίδια μηχανική και ηλεκτρική υποδομή, αλλάζοντας μόνο τη βαθμίδα ελέγχου. Σε σύγκριση με την προγενέστερη εργασία, αυτό που μεταβάλλεται στην παρούσα διπλωματική δεν είναι ούτε η μηχανή ούτε η αλυσίδα μετάδοσης πληροφορίας από το σχέδιο στον κώδικα G (αυτά διατηρούνται ταυτόσημα). Αλλάζει η βαθμίδα του ελεγκτή. Στη θέση του μικροελεγκτή Atmega 328p (Atmel/Microchip) που υλοποιούσε όλες τις λειτουργίες ως λογισμικό σε γλώσσα assembly [1, §2.2.2], τοποθετείται ένα FPGA (Cyclone V σε αναπτυξιακή πλακέτα DE1-SoC), στο οποίο οι ίδιες λειτουργίες υλοποιούνται ως παράλληλα ψηφιακά κυκλώματα γραμμένα σε Verilog. Η αντικατάσταση αυτή (και τα οφέλη που συνεπάγεται) αποτελεί το αντικείμενο της παρούσας διπλωματικής, και αναπτύσσεται αναλυτικά σε επόμενο κεφάλαιο.



Σχήμα 2.3: *Καρτεσιανή μηχανή CNC δύο αξόνων (X–Y)* που χρησιμοποιείται ως πλατφόρμα δοκιμών. Διακρίνονται οι δύο βηματικοί κινητήρες, οι ιμάντες GT2 ανά άξονα, τα φορεία (carriers), η κεφαλή και η πλακέτα του ελεγκτή. (Πηγή: [1])

2.2 Κώδικας G (ISO 6983) - Δομή και Εντολές

Ο κώδικας G είναι η γλώσσα στην οποία περιγράφονται οι κατεργασίες σε σχεδόν κάθε CNC μηχανή που κατασκευάζεται σήμερα. Η αρχική του μορφή προήλθε από την τυποποίηση *RS-274* της Electronic Industries Association στις αρχές της δεκαετίας του 1960. Αργότερα αναγνωρίστηκε διεθνώς ως *ISO 6983*, με την πιο πρόσφατη αναθεώρηση να είναι το ISO 6983-1:2009. Πέρα από το ότι παραμένει το πρότυπο που αναφέρεται στους ελεγκτές των βιομηχανικών εργαλειομηχανών, αποτελεί και το αρχείο εξόδου που παράγεται σχεδόν από κάθε λογισμικό CAM, οπότε η συντακτική του δομή είναι ταυτόσημη ανεξάρτητα από τη μηχανή στην οποία απευθύνεται.

Ένα πρόγραμμα G-code οργανώνεται σε γραμμές που ονομάζονται **μπλοκ** (blocks). Κάθε μπλοκ συνδυάζει μία ή περισσότερες **λέξεις** (words). Κάθε λέξη αποτελείται από ένα γράμμα και μια αριθμητική τιμή. Για παράδειγμα, η λέξη `X100.5` ορίζει συντεταγμένη X ίση με 100.5 μονάδες, ενώ η `F1500` ορίζει ταχύτητα προώθησης 1500 mm/min. Η δομή ενός τυπικού μπλοκ φαίνεται στο Σχήμα 2.4.

Σχήμα 2.4: Δομή ενός μπλοκ κώδικα G. Κάθε λέξη συντίθεται από ένα γράμμα-διεύθυνση (N, G, X, Y, F, …) και μια αριθμητική τιμή.

Το γράμμα κάθε λέξης καθορίζει και τη σημασία της. Οι λέξεις X, Y, Z δηλώνουν γραμμικές συντεταγμένες στους τρεις άξονες, οι I, J, K μετατοπίσεις του κέντρου ενός τόξου ως προς τη θέση εκκίνησης, η F την ταχύτητα προώθησης, η S την ταχύτητα περιστροφής της κεφαλής (ή την ένταση του laser στη συγκεκριμένη μηχανή), η T την επιλογή εργαλείου και η N την προαιρετική αρίθμηση γραμμών. Εντολές που χρησιμοποιούνε το γράμμα G χαρακτηρίζουν μια προπαρασκευαστική λειτουργία που ορίζουν τη μορφή της κίνησης ή τη ρύθμιση που πρόκειται να εφαρμοστεί, ενώ εντολές που χρησιμοποιούνε το γράμμα M χαρακτηρίζουν μια βοηθητική λειτουργία για τον έλεγχο διαφόρων περιφερειακών καταστάσεων της μηχανής, όπως ενεργοποίηση κεφαλής, ψύξη ή τερματισμός προγράμματος.

Το πρότυπο ορίζει δεκάδες προπαρασκευαστικές και βοηθητικές λειτουργίες. Στον Πίνακα 2.1 παρουσιάζονται οι εντολές αυτές μαζί με τη λειτουργία τους. Διακρίνονται σε εντολές κίνησης (G0–G3), που υπαγορεύουν τη μορφή της τροχιάς, σε εντολές ρύθμισης (G17, G20/G21, G90/G91, G94), που ορίζουν την ερμηνεία των επόμενων κινήσεων, και σε βοηθητικές εντολές M, που χειρίζονται την κεφαλή και τη λήξη του προγράμματος.

Πίνακας 2.1: Υποσύνολο εντολών κώδικα G που χρησιμοποιούνται στην παρούσα υλοποίηση

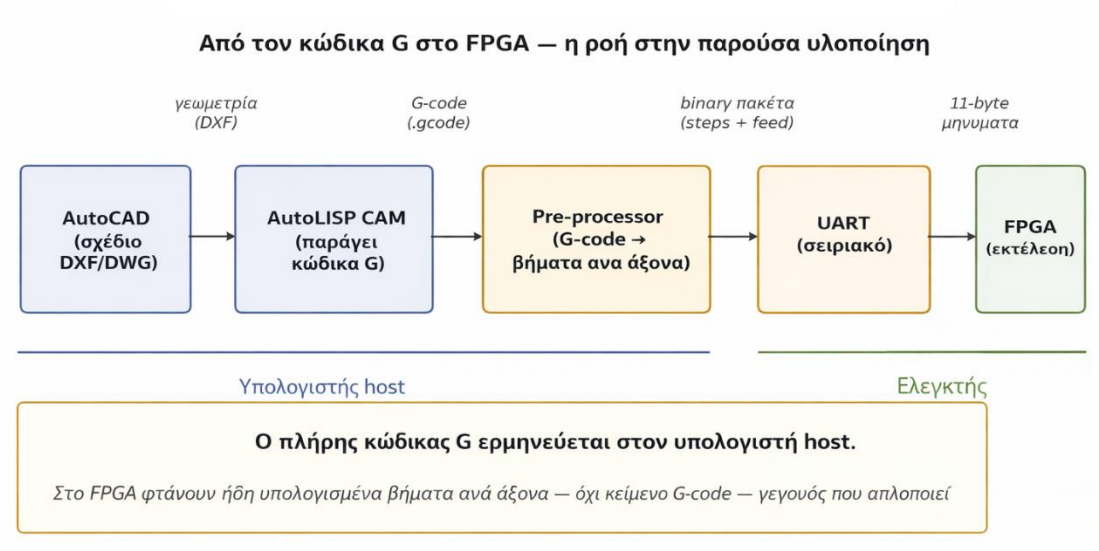
Εντολή	Τύπος	Λειτουργία
G00	Κίνηση	Γρήγορη μετακίνηση χωρίς κατεργασία
G01	Κίνηση	Γραμμική παρεμβολή με ταχύτητα προώθησης
G02	Κίνηση	Κυκλική παρεμβολή κατά τη φορά του ωρολογίου (CW)
G03	Κίνηση	Κυκλική παρεμβολή αντίθετα στη φορά του ωρολογίου (CCW)
G17	Ρύθμιση	Επιλογή επιπέδου εργασίας XY

Εντολή	Τύπος	Λειτουργία
G20 / G21	Ρύθμιση	Μονάδες σε ίντσες / χιλιοστά
G90 / G91	Ρύθμιση	Απόλυτο / σχετικό σύστημα συντεταγμένων
G94	Ρύθμιση	Ταχύτητα προώθησης εκφρασμένη σε mm/min
M03	Βοηθητική	Ενεργοποίηση κεφαλής (laser ON)
M05	Βοηθητική	Απενεργοποίηση κεφαλής (laser OFF)
M30	Βοηθητική	Τέλος προγράμματος και επαναφορά δείκτη

Ένα πλήρες (μικρό) παράδειγμα προγράμματος G το οποίο χαράζει με laser ένα ορθογώνιο διαστάσεων 40 × 20 mm με αρχή στο σημείο (10, 10) δίνεται παρακάτω:

```
; Sample G-code — orthogonio 40x20
G21 ; monades se mm
G90 ; apolyto systhma syntetagmenwn
G17 ; epipedo XY
G94 ; feed se mm/min
G0  X10.000 Y10.000  ; tachia metakinhsh sthn arxh
M3  S255 ; laser ON
G1  X50.000 Y10.000  F800  ; pleura katw
G1  X50.000 Y30.000  F800  ; pleura dexia
G1  X10.000 Y30.000  F800  ; pleura panw
G1  X10.000 Y10.000  F800    ; pleura aristera
M5   ; laser OFF
M30 ; telos programmatos
```

Στην παρούσα υλοποίηση, ολόκληρος ο κύκλος ερμηνείας του κώδικα G πραγματοποιείται στον υπολογιστή host και όχι στο FPGA (Σχήμα 2.5). Ένα προσαρμοσμένο εργαλείο CAM γραμμένο σε AutoLISP εξάγει από το σχέδιο στο CAD ένα αρχείο .nc το οποίο περιέχει αποκλειστικά τις λέξεις του Πίνακα 2.1. Στη συνέχεια, ένα δεύτερο εργαλείο διαβάζει το αρχείο αυτό και μετατρέπει κάθε εντολή σε σχετικές μετατοπίσεις βημάτων ανά άξονα, με βάση την ανάλυση steps/mm κάθε άξονα της μηχανής, ενώ τα τόξα G2/G3 μετασχηματίζονται σε πολλά μικρά γραμμικά τμήματα. Όπως θα δούμε παρακάτω ο κώδικας G, μετατρέπεται σε μια ακολουθία δυαδικών πακέτων σταθερού μήκους (11 byte ανά πακέτο), όπου κάθε πακέτο μεταφέρει τον κωδικό κίνησης, έναν προσιμασμένο αριθμό που δηλώνει το πλήθος των βημάτων ανά άξονα και την φορά περιστροφής και τέλος την επιθυμητή ταχύτητα προώθησης. Τα πακέτα αυτά αποστέλλονται μέσω UART στο FPGA, όπου εκτελούνται από τη βαθμίδα παραγωγής βημάτων (κεφάλαιο 4).



Σχήμα 2.5: *Ροή του κώδικα G στην παρούσα υλοποίηση.* Η ερμηνεία γίνεται εξ ολοκλήρου στον υπολογιστή host· στο FPGA φτάνουν προ-υπολογισμένα βήματα ανά άξονα, όχι κείμενο G-code.

Η επιλογή να μην υλοποιηθεί διερμηνέας του G-code εντός του FPGA δεν είναι αυθαίρετη. Η ερμηνεία ASCII κειμένου, οι αριθμητικές πράξεις με κινητή υποδιαστολή και η τμηματοποίηση τόξων σε γραμμικά τμήματα είναι εργασίες που ταιριάζουν στη φύση μιας CPU γενικού σκοπού και όχι στη συνδυαστική και ακολουθιακή λογική που υλοποιείται με LUT και flip-flop. Μεταφέροντας την ερμηνεία στον host, το FPGA απαλλάσσεται από την αυτή την πολυπλοκότητα και αφιερώνεται στο κομμάτι όπου προσφέρει αδιαμφισβήτητο πλεονέκτημα: παράλληλη παραγωγή παλμών step/dir με χρονική ακρίβεια της τάξης των μερικών νανοδευτερολέπτων. Η υλοποίηση πλήρους διερμηνέα G-code σε hardware, έχει επιχειρηθεί σε ερευνητικές εργασίες (Garrido κ.ά., 2010), προσφέρει όμως οφέλη μόνο σε εφαρμογές υψηλής ταχύτητας όπου η καθυστέρηση επικοινωνίας host–controller γίνεται κρίσιμη.

2.3 Βηματικοί κινητήρες και οδηγοί (drivers)

Η κίνηση κάθε άξονα της μηχανής εκτελείται από έναν βηματικό κινητήρα και έναν αντίστοιχο driver. Η συγκεκριμένη επιλογή έγινε στην προγενέστερη εργασία ύστερα από σύγκριση με την εναλλακτική των servo κινητήρων. Διαπιστώθηκε ότι ο βηματικός κινητήρας υπερτερεί στη συγκεκριμένη εφαρμογή ως προς τη λειτουργία σε ανοιχτό βρόχο χωρίς ανάγκη ανάδρασης θέσης λόγω χαμηλότερου κόστους, υψηλότερης ροπής σε μικρές ταχύτητες και απουσία δονήσεων στη στατική θέση κατακράτησης [1, πίνακας 2.2]. Σε αντίστοιχα CNC χαμηλού-μεσαίου κόστους, η συγκεκριμένη λύση (βηματικός κινητήρας χωρίς encoder σε ανοιχτό βρόχο) αποτελεί την κυρίαρχη επιλογή.

Στους δύο άξονες της μηχανής χρησιμοποιείται το ίδιο μοντέλο κινητήρα, ο 42BYGHW208 [1, draft 2.2]. Πρόκειται για διπολικό υβριδικό βηματικό κινητήρα τυποποιημένου μεγέθους NEMA17, με βασική γωνία βήματος 1,8° (που αντιστοιχεί σε 200 πλήρη βήματα ανά πλήρη περιστροφή), ονομαστική τάση 12 V, ονομαστικό ρεύμα 0,4 A ανά φάση και ροπή κατακράτησης 2.800 g·cm. Στη μηχανή, ο κινητήρας του X άξονα μεταφέρει συνολικά 3,8 kg (τα δύο φορεία του X (αριστερό και δεξί), το φορείο του Y, τον συνδετικό άξονα και τον ιμάντα του Y) ενώ ο κινητήρας του Y μεταφέρει μόνο το φορείο του Y. Η ροπή που απαιτείται για την κίνηση και την επιτάχυνση μέχρι την επιθυμητή ταχύτητα 80

mm/s υπολογίστηκε αναλυτικά στο [1, §2.1.4] και χωρίς ιδιαίτερη δυσκολία καλύπτεται από τον συγκεκριμένο κινητήρα ακόμη και με συντελεστή ασφαλείας.

Από μόνος του ο κινητήρας δεν μπορεί να συνδεθεί απευθείας στην ψηφιακή έξοδο ενός μικροελεγκτή ή ενός FPGA, καθώς απαιτείται κατάλληλη ηλεκτρονική βαθμίδα που θα διοχετεύσει το ρεύμα στα τυλίγματα με την κατάλληλη φορά και ένταση. Τη λειτουργία αυτή επιτελεί ο driver. Στην εργασία χρησιμοποιείται το ολοκληρωμένο DRV8825 της Texas Instruments [1, §2.2.1], του οποίου η διάταξη ακροδεκτών παρουσιάζεται στο σχήμα 2.4.

Σχήμα 2.6: Λειτουργικές ακίδες του οδηγού DRV8825. Από την πλευρά ελέγχου: EN (ενεργοποίηση), M0/M1/M2 (επιλογή ανάλυσης microstep), STEP (κάθε παλμός = ένα microstep), DIR (φορά περιστροφής). Από την πλευρά ισχύος και κινητήρα: VMOT, GND_M, A1/A2 (τύλιγμα A), B1/B2 (τύλιγμα B), FAULT (έξοδος ένδειξης σφάλματος).

Ο οδηγός χρησιμοποιεί στο εσωτερικό του δύο γέφυρες H (μία ανά τύλιγμα) που του επιτρέπουν να αντιστρέφει τη φορά του ρεύματος και να εκτελεί έτσι τη διαδοχική διέγερση που απαιτεί ο διπολικός κινητήρας [1, §2.2.1]. Στη λογική πλευρά απαιτεί μόνο τρεις απλές γραμμές για τη βασική του λειτουργία: το pin EN ενεργοποιεί ή απενεργοποιεί τις εξόδους του (λογικό «0» = ενεργό, «1» = ανενεργό), το pin DIR καθορίζει τη φορά περιστροφής του κινητήρα (η αντιστοίχιση μεταξύ λογικής τιμής και μηχανικής φοράς εξαρτάται από τη φυσική συνδεσμολογία των τυλιγμάτων), και το STEP δέχεται έναν παλμό για κάθε microstep που πρέπει να εκτελέσει ο κινητήρας. Επιπλέον, οι ακίδες M0, M1 και M2 ορίζουν την υποδιαίρεση του βασικού βήματος (microstepping) σύμφωνα με τον πίνακα 2.5.

Επιλογή ανάλυσης microstep μέσω M0, M1, M2 [1]

M0	M1	M2	Ανάλυση	Γραμμικό βήμα (a=1,8°, p=2 mm)
0	0	0	Πλήρες βήμα	0,200 mm
1	0	0	1/2 βήμα	0,100 mm
0	1	0	1/4 βήμα	0,050 mm
1	1	0	1/8 βήμα	0,025 mm
0	0	1	**1/16 βήμα**	**0,0125 mm**
1	0	1	**1/32 βήμα**	**0,00625 mm**

Το γραμμικό βήμα προκύπτει από την εξίσωση 2.1 του [1]: dx = 2a / 18 mm για ιμάντα GT2 με p=2 mm και τροχαλία 20 δοντιών.

Σχήμα 2.7: Επιλογή ανάλυσης microstep μέσω των ακίδων M0, M1, M2 στον DRV8825. Στην τρίτη στήλη φαίνεται η ονομαστική γωνιακή ανάλυση και στην τέταρτη η αντίστοιχη γραμμική ανάλυση όταν συνδυαστεί με τον μηχανισμό μετάδοσης της μηχανής (ιμάντας GT2 με τροχαλία 20 δοντιών). [1, πίνακας 2.3]

Ο μηχανισμός του microstepping επιτρέπει την υποδιαίρεση κάθε φυσικού βήματος σε μέχρι 32 ηλεκτρικά υπο-βήματα, μέσω κατάλληλης διαμόρφωσης των ρευμάτων στα δύο τυλίγματα. Για μικρές τιμές του microstep, η λειτουργία γίνεται με διακριτές καταστάσεις πλήρους ρεύματος, και κάθε νέος παλμός STEP μετακινεί τον κινητήρα κατά μια πλήρη γωνία βήματος. Για μεγάλες τιμές του microstep, ο οδηγός παράγει εσωτερικά κλιμακωτές προσεγγίσεις ημιτονοειδών ρευμάτων με διαμόρφωση εύρους παλμών (PWM), και κάθε παλμός STEP αντιστοιχεί σε μικρότερη γωνιακή μετατόπιση. Συνδυάζοντας την εξίσωση της μετατόπισης (dx = 2α/18 mm) με τη ανάλυση που επιτυγχάνεται από τη ρύθμιση των pin M0/M1/M2, προκύπτει η γραμμική ανάλυση της μηχανής. Στη ρύθμιση πλήρους βήματος αυτή είναι 0,200 mm, ενώ στη ρύθμιση 1/16 microstepping πέφτει στα 0,0125 mm.

Όσο μικρότερο γίνεται το microstep, τόσο μεγαλύτερος γίνεται ο αριθμός των παλμών που χρειάζονται για μια δεδομένη μετατόπιση, γεγονός που στην προγενέστερη εργασία αποτελούσε ουσιαστικό περιορισμό λόγω της χρήσης 8-bit καταχωρητών στον Atmega 328p [1, §6.1]. Στην παρούσα υλοποίηση, οι εσωτερικοί καταχωρητές του FPGA διατηρούνται σε 32 bits, και ο περιορισμός αυτός παύει να ισχύει.

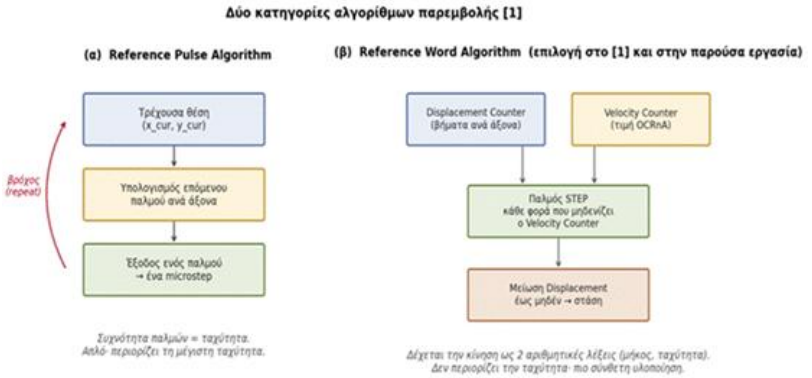
Τέλος, σχετικά με την ηλεκτρική σύνδεση, ο DRV8825 τροφοδοτείται με μια ξεχωριστή τάση κινητήρα VMOT (στη συγκεκριμένη υλοποίηση 12 V από εξωτερικό μετασχηματιστή), η οποία είναι ανεξάρτητη από την τάση τροφοδοσίας της λογικής (3,3 V από την πλακέτα του ελεγκτή). Οι έξοδοι A1, A2, B1, B2 συνδέονται απευθείας στα δύο τυλίγματα του κινητήρα. Ολόκληρο το ηλεκτρονικό κύκλωμα (δύο οδηγοί DRV8825, τα pin σύνδεσης με τους κινητήρες, οι θέσεις των τερματικών διακοπτών και η τροφοδοσία) έχει υλοποιηθεί σε μια τυπωμένη πλακέτα η οποία αναπτύχθηκε στην προγενέστερη εργασία έτσι ώστε να μπορεί να προσαρμοστεί στην αναπτυξιακή πλακέτα του Atmega 328p [1, §2.2.4]. Στην παρούσα διπλωματική η ίδια πλακέτα συνδέεται στις γενικού σκοπού ψηφιακές εξόδους (GPIO) της πλακέτας DE1-SoC.

2.4 Γραμμική και κυκλική παρεμβολή

Ο όρος παρεμβολή (interpolation) χρησιμοποιείται στη βιβλιογραφία των CNC συστημάτων για να περιγράψει το σύνολο των αλγορίθμων που μετατρέπουν τις γεωμετρικές εντολές του κώδικα G σε μια ακολουθία διακριτών παλμών για κάθε άξονα της μηχανής [1, εισαγωγή §5]. Η ακολουθία των παλμών πρέπει να εξασφαλίζει ότι η συνισταμένη τροχιά της κεφαλής ακολουθεί την επιθυμητή γεωμετρία (ευθεία ή τόξο) σε όλη τη διάρκεια της κίνησης, και ότι η ταχύτητα μετακίνησης της κεφαλής κατά μήκος αυτής της τροχιάς αντιστοιχεί στη ζητούμενη ταχύτητα προώθησης.

Στη βιβλιογραφία διακρίνονται δύο μεγάλες οικογένειες αλγορίθμων παρεμβολής (Σχήμα 2.6) [1, §5]. Η πρώτη, που στο [1] αναφέρεται ως Reference Pulse Algorithm, επιστρέφει σε κάθε κύκλο εκτέλεσης ένα διακριτό παλμό ανά άξονα. Σε κάθε επανάληψη, ο αλγόριθμος γνωρίζει την τρέχουσα θέση των αξόνων και τη ζητούμενη τελική θέση, αποφασίζει αν πρέπει να σταλεί παλμός σε κάθε άξονα και ενημερώνει την τρέχουσα θέση. Η συχνότητα με την οποία επαναλαμβάνεται ο αλγόριθμος είναι η ταχύτητα κίνησης. Σε αυτή την κατηγορία ανήκουν ο Digital Differential Analyzer, η Stairs Approximation Method και η Direct Search Method. Παρόμοιοι αλγόριθμοι χρησιμοποιούνται στην ψηφιακή επεξεργασία εικόνας, όπου το microstep αντιστοιχεί σε pixel.

Η δεύτερη οικογένεια, που αναφέρεται Reference Word Algorithm, δεν παράγει διακριτούς παλμούς αλλά αριθμητικές λέξεις. Μία n-bit λέξη για το συνολικό πλήθος βημάτων ανά άξονα και μία m-bit λέξη για την αντίστοιχη ταχύτητα. Οι λέξεις αυτές αποθηκεύονται σε εσωτερικούς καταχωρητές: έναν Displacement Counter που μετρά τα βήματα που απομένουν για κάθε άξονα και ένα Velocity Counter που ελέγχει τη συχνότητα παραγωγής των παλμών. Κάθε φορά που ο Velocity Counter μηδενίζεται, παράγεται ένας παλμός STEP και ο Displacement Counter μειώνεται κατά ένα. Η κίνηση τερματίζεται όταν ο Displacement Counter φτάσει στο μηδέν.



Σχήμα 2.8: Οι δύο κύριες κατηγορίες αλγορίθμων παρεμβολής [1, §5]. (α) Αλγόριθμος αναφοράς παλμού: σε κάθε κύκλο επανάληψης παράγεται διακριτός παλμός. (β) Αλγόριθμος αναφοράς λέξης: η κίνηση μεταβιβάζεται ως αριθμητικές λέξεις (μήκος και ταχύτητα) και η παραγωγή των παλμών γίνεται από εσωτερικούς μετρητές. Επιλογή τόσο της προγενέστερης εργασίας όσο και της παρούσας υλοποίησης.

Η επιλογή ανάμεσα στις δύο κατηγορίες είναι θέμα συμβιβασμού μεταξύ απλότητας και επιδόσεων. Οι αλγόριθμοι αναφοράς παλμού είναι απλούστεροι σε υλοποίηση αλλά εισάγουν περιορισμό στη μέγιστη ταχύτητα, καθώς η συχνότητα παραγωγής παλμών δεν μπορεί να ξεπεράσει αυτή με την οποία τρέχει ο αλγόριθμος. Οι αλγόριθμοι αναφοράς λέξης δεν εισάγουν τέτοιο περιορισμό, αλλά απαιτούν πιο σύνθετη υλοποίηση και θεωρούνται λιγότερο ακριβείς [1, §5]. Στην προγενέστερη εργασία επιλέχθηκε η δεύτερη προσέγγιση, και η ίδια επιλογή υιοθετείται στην παρούσα διπλωματική. Στο FPGA, οι ρόλοι των δύο μετρητών αντιστοιχούν στους εσωτερικούς καταχωρητές rem_x/rem_y (Displacement) και cnt_x/cnt_y (Velocity) της βαθμίδας step_gen.v.

Στη γραμμική παρεμβολή η μετακίνηση είναι ευθύγραμμη ομαλή κίνηση (σταθερή ταχύτητα προώθησης). Αν υποθέσουμε την γενική περίπτωση μετατόπησης από το σημείο (x_1, y_1) στο σημείο (x_2, y_2). Οι μετατοπίσεις σε κάθε άξονα είναι [1, εξίσωση 5.1]:

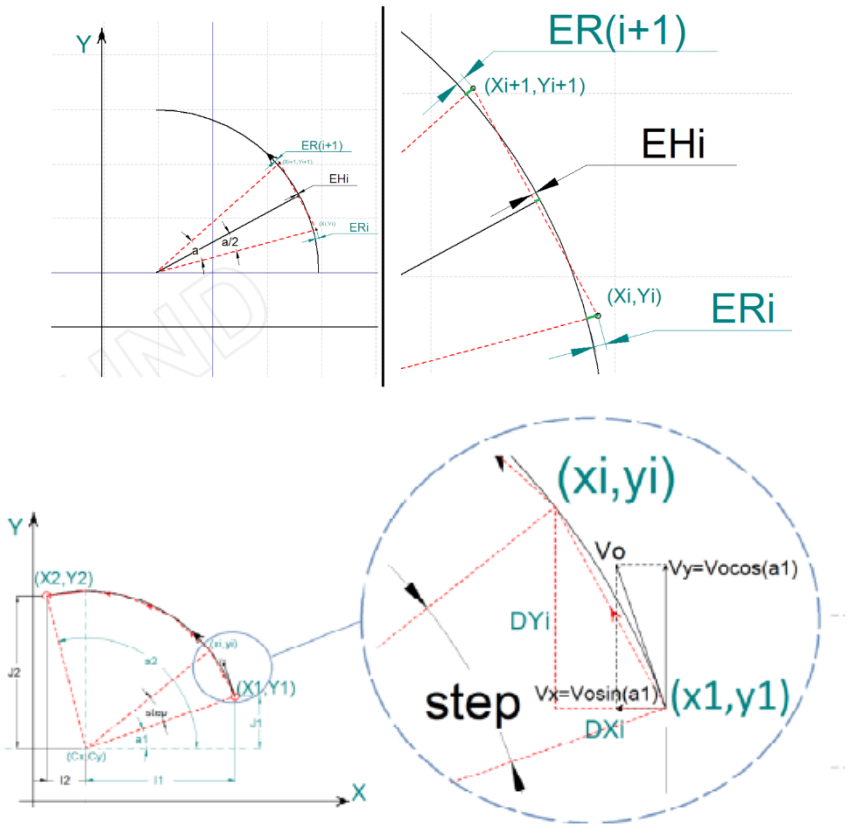
$$dx = x_2 - x_1, \quad dy = y_2 - y_1$$

Το πρόσημο των δύο τιμών καθορίζει τη φορά περιστροφής κάθε κινητήρα μέσω του pin DIR του αντίστοιχου driver, ενώ μηδενική τιμή σε έναν άξονα προκαλεί την απενεργοποίηση του αντίστοιχου οδηγού μέσω του pin EN. Η συνισταμένη μετατόπιση είναι ίση με $dz = \sqrt{(dx^2 + dy^2)}$ και η συνισταμένη ταχύτητα ίση με τη ζητούμενη ταχύτητα προώθησης v_0. Από τις εξισώσεις της ευθύγραμμης ομαλής κίνησης (σταθερή ταχύτητα), προκύπτουν οι αντίστοιχες ταχύτητες ανά άξονα [1, εξισώσεις 5.6]:

$$v\_x = v_0 \cdot (dx / dz), \quad v\_y = v_0 \cdot (dy / dz)$$

Στην προγενέστερη εργασία, ο μικροελεγκτής λάμβανε από τον υπολογιστή το πλήθος των βημάτων μόνο του X άξονα (όχι του Y), την περίοδο του Velocity Counter του X (καταχωρητής OCR0A) και την περίοδο του Velocity Counter του Y (καταχωρητής OCR2A) [1, εξισώσεις 5.7]. Στην παρούσα υλοποίηση, υπάρχουν δύο σημεία που διαφοροποιείται αυτή η λογική. Πρώτον, ο υπολογιστής στέλνει τα βήματα και των δύο αξόνων ως 32-bit πεδία StepsX και StepsY (αντί μόνο του X), ώστε ο ελεγκτής να γνωρίζει το πλήρες πλάνο της κίνησης. Δεύτερον, οι περίοδοι των Velocity Counters δεν στέλνονται προ-υπολογισμένες αλλά υπολογίζονται μέσα στο FPGA, με βάση τη ζητούμενη ταχύτητα προώθησης (FeedRate) και έναν αριθμητή NUMERATOR που έχει σταλεί στην αρχή με το πακέτο αρχικοποίησης (βλ. ενότητα 2.6). Η συγκεκριμένη επιλογή κάνει τη βαθμίδα step_gen.v ανεξάρτητη του τύπου της μηχανής. Η ίδια λογική μπορεί να δουλέψει και σε μηχανή με διαφορετικά μηχανολογικά χαρακτηριστικά, αρκεί να ενημερωθεί ανάλογα ο αριθμητής NUMERATOR.

Για την κυκλική παρεμβολή (G02 και G03), η προγενέστερη εργασία εφαρμόζει ως λύση την προσέγγιση του τόξου ως ακολουθία μικρών γραμμικών τμημάτων (χορδών) ενός εγγεγραμμένου πολυγώνου [1, §5.3]. Η ίδια ακριβώς προσέγγιση διατηρείται και στην παρούσα διπλωματική. Η λύση αυτή φαίνεται στο σχήμα 2.7: το πραγματικό τόξο διαιρείται σε ακολουθία τόξων μιας σταθερής γωνίας α (η γωνία βήματος), και η κίνηση κατά μήκος κάθε ενδιάμεσου τόξου εκτελείται ως γραμμική κίνηση από την αρχή του στο τέλος του.



Σχήμα 2.9: Προσέγγιση τόξου με εγγεγραμμένο πολύγωνο, όπως στην προγενέστερη εργασία [1, drafts 5.5–5.6]. Το τόξο διαιρείται σε χορδές γωνιακού βήματος α· κάθε χορδή εκτελείται ως ξεχωριστή γραμμική κίνηση. Όσο μικρότερο το α, τόσο μικρότερο το σφάλμα ΕΗ (chord error) — αλλά τόσο περισσότερα τμήματα στέλνονται στον ελεγκτή.

Για κάθε τόξο, η ακτίνα R υπολογίζεται από τις σχετικές μετατοπίσεις του κέντρου ως προς το σημείο εκκίνησης (που στον κώδικα G φέρουν τα γράμματα I και J), $R = \sqrt{(I_1^2 + J_1^2)}$ [1, εξίσωση 5.8]. Η αρχική

γωνία α_1 και η τελική γωνία α_2 υπολογίζονται από τις σχέσεις atan2 (J/I), και ο αλγόριθμος προωθεί τη γωνία από α_1 προς α_2 με βήμα α, υπολογίζοντας σε κάθε ενδιάμεση θέση το αντίστοιχο σημείο του τόξου από τις παραμετρικές εξισώσεις [1, εξισώσεις 5.12]:

$$x_{2i} = R \cdot \cos(\alpha_1 + i \cdot \alpha), \quad y_{2i} = R \cdot \sin(\alpha_1 + i \cdot \alpha)$$

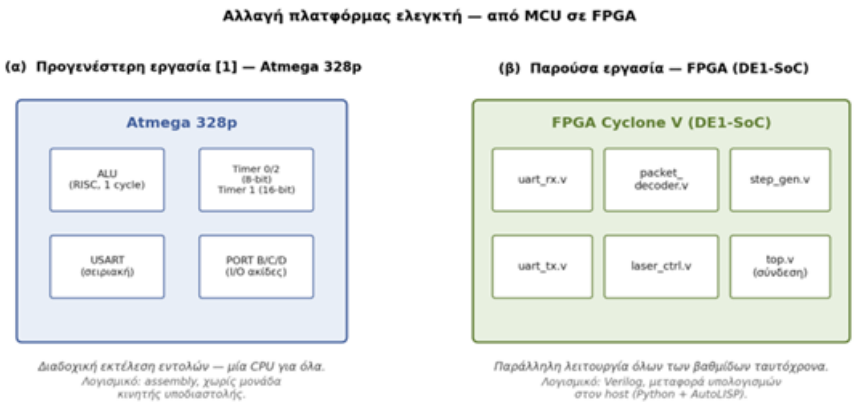
Για κάθε χορδή του πολυγώνου, ο αλγόριθμος υπολογίζει τις σχετικές μετατοπίσεις βημάτων στους δύο άξονες και τις στέλνει στον ελεγκτή ως μια ευθύγραμμη μετατόπιση. Η διαδικασία επαναλαμβάνεται μέχρι η τρέχουσα γωνία να ξεπεράσει την τελική γωνία, οπότε η εκτέλεση του τόξου τερματίζεται. Στην περίπτωση τόξου CCW (G03) χρησιμοποιείται έλεγχος $\alpha_1 \leq \alpha_2$· σε τόξο CW (G02) η εκτέλεση τρέχει αντίθετα, μέχρι $\alpha_2 < \alpha_1$ [1, §5.4].

Δύο τύποι σφάλματος χαρακτηρίζουν αυτή την προσέγγιση [1, draft 5.5]: το σφάλμα χορδής EH (chord error) που είναι η μέγιστη απόσταση μεταξύ της χορδής και του πραγματικού τόξου και το σφάλμα ακτίνας ER (radius error) που είναι η απόκλιση της μέσης απόστασης της εκτελούμενης τροχιάς από το θεωρητικό κέντρο. Και τα δύο σφάλματα μειώνονται όσο μικραίνει η γωνία βήματος α, με αντίτιμο όμως την αύξηση του πλήθους των τμημάτων που πρέπει να σταλούν στον ελεγκτή. Όπως αναφέρεται στο [1], η ποιότητα αυτής της προσέγγισης εξαρτάται από τον χρόνο που χρειάζεται για την επίλυση των εξισώσεων και τη μεταφορά των δεδομένων στον ελεγκτή.

Η μετατόπιση της λογικής της κυκλικής παρεμβολής στον υπολογιστή host (αντί στον ελεγκτή) αποτελεί συνειδητή επιλογή. Στη μηχανή της παρούσας εργασίας, ο host διαθέτει αφθονία μνήμης και υπολογιστικής ισχύος, και μπορεί να εκτελέσει με άνεση τις τριγωνομετρικές πράξεις για κάθε τόξο. Ο ελεγκτής, αντίθετα, χρειάζεται να κρατηθεί όσο πιο απλός γίνεται ώστε να εξασφαλιστεί ντετερμινιστική παραγωγή των παλμών step. Η ίδια επιλογή είχε γίνει και στην προγενέστερη εργασία [1, §5.3]: ο μικροελεγκτής Atmega 328p δεν διέθετε μονάδα κινητής υποδιαστολής, και οι τριγωνομετρικοί υπολογισμοί ήταν πρακτικά ανέφικτοι σε επίπεδο assembly και για αυτό οι παραπάνω υπολογισμοί εκτελούνται στον host υπολογιστή.

2.5 Τεχνολογία FPGA — αρχιτεκτονική και πλεονεκτήματα

Η ουσιαστική διαφοροποίηση της παρούσας διπλωματικής σε σχέση με την προγενέστερη εργασία [1] εντοπίζεται στην επιλογή της πλατφόρμας ελέγχου της μηχανής. Στην προγενέστερη υλοποίηση, ο ρόλος του ελεγκτή είχε ανατεθεί στον μικροελεγκτή Atmega 328p της Atmel, ένα ολοκληρωμένο 8-bit RISC αρχιτεκτονικής Harvard με 32 καταχωρητές γενικού σκοπού, δύο timers-counters των 8-bit και έναν των 16-bit, και μια μονάδα USART για σειριακή επικοινωνία [1, §2.2.2]. Ο μικροελεγκτής αυτός εκτελούσε όλη τη λογική του ελεγκτή (αναγνώριση εντολών, υπολογισμό κίνησης, παραγωγή παλμών step, διαχείριση τερματικών διακοπτών). Στην παρούσα εργασία, ο μικροελεγκτής αντικαθίσταται από ένα FPGA, και οι ίδιες λειτουργίες υλοποιούνται ως παράλληλα ψηφιακά κυκλώματα γραμμένα σε γλώσσα περιγραφής υλικού Verilog. Στο σχήμα 2.8 αποτυπώνεται η σύγκριση των δύο πλατφορμών.



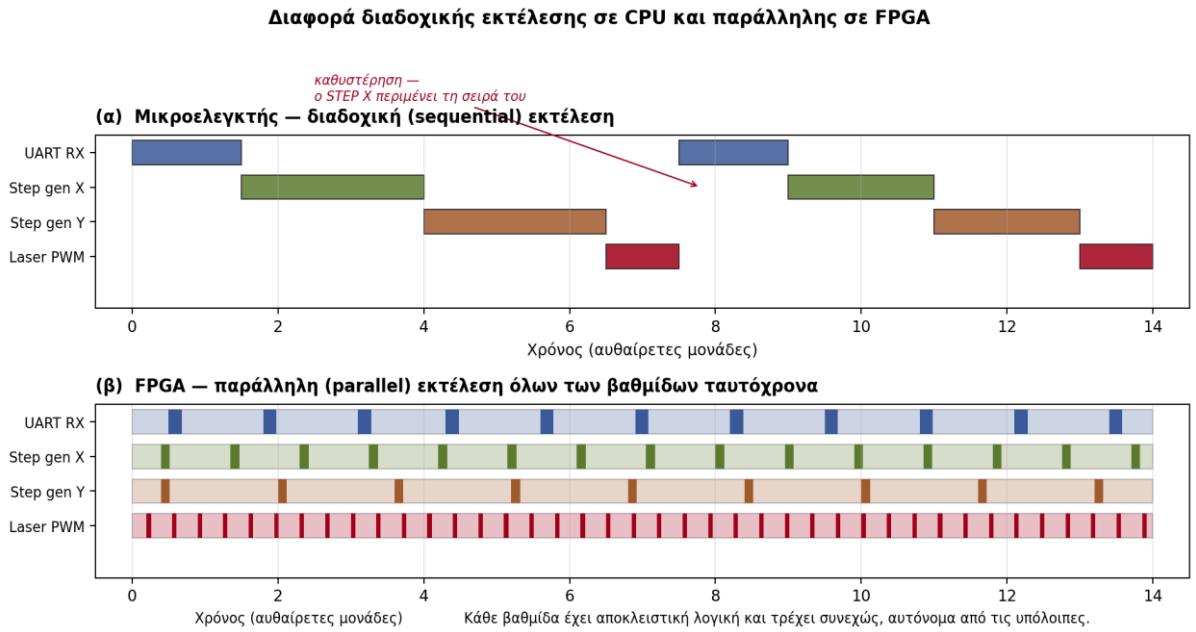
Σχήμα 2.10: Σύγκριση των δύο ελεγκτών CNC. (α) Στην προγενέστερη εργασία [1], ο Atmega 328p εκτελεί ένα πρόγραμμα assembly: μία μονάδα ALU εξυπηρετεί όλες τις λειτουργίες, οι οποίες ανταγωνίζονται. (β) Στην παρούσα υλοποίηση, οι ίδιες λειτουργίες χωρίζονται σε ξεχωριστά Verilog modules μέσα στο FPGA, και κάθε ένα διαθέτει τη δική του αποκλειστική λογική που εκτελείται ταυτόχρονα με τα υπόλοιπα.

Το ακρωνύμιο FPGA σημαίνει *Field-Programmable Gate Array*, δηλαδή πίνακας λογικών πυλών προγραμματιζόμενος στο πεδίο. Πρόκειται για μια κατηγορία ολοκληρωμένων κυκλωμάτων των οποίων η εσωτερική δομή δεν έχει προκαθοριστεί από τον κατασκευαστή. Ο τελικός χρήστης μπορεί να καθορίσει (μέσω κατάλληλου πηγαίου κώδικα σε γλώσσα Verilog ή VHDL) ποιες ακριβώς συνδέσεις θα δημιουργηθούν στο εσωτερικό του τσιπ. Η συσκευή αποτελείται από έναν εκτεταμένο πίνακα λογικών μπλοκ (ALMs στην ορολογία της Altera), που ο καθένας περιλαμβάνει πίνακες αναζήτησης (LUTs) ικανούς να υλοποιήσουν αυθαίρετες λογικές συναρτήσεις λίγων εισόδων, μερικά flip-flop για συγκράτηση τιμών, και βοηθητική λογική. Επιπλέον περιλαμβάνει ξεχωριστά μπλοκ ενσωματωμένης μνήμης, εξειδικευμένες αριθμητικές μονάδες (DSP blocks) και περιφερειακά εισόδου-εξόδου. Όλα αυτά τα στοιχεία συνδέονται μεταξύ τους μέσω ενός πλέγματος αγωγών, του οποίου η διαμόρφωση καθορίζεται κατά τον προγραμματισμό της συσκευής.

Ένα FPGA δεν εκτελεί λογισμικό. Μετά τον προγραμματισμό του, ολόκληρο το κύκλωμα που έχει περιγραφεί στον πηγαίο κώδικα υπάρχει μέσα στο ολοκληρωμένο. Όταν τα δεδομένα εισόδου του αλλάζουν, οι αντίστοιχες έξοδοι ενημερώνονται μέσα σε λίγα νανοδευτερόλεπτα, χωρίς να χρειάζεται καμιά «επόμενη εντολή» να τοποθετηθεί στη σειρά εκτέλεσης. Λόγω αυτής της αρχιτεκτονικής προκύπτουν τα τρία πλεονεκτήματα της τεχνολογίας FPGA τα οποία είναι σχετικά με την παρούσα εφαρμογή.

Πρώτον, παραλληλισμός εκτέλεσης. Στον Atmega 328p [1, §2.2.2], η παραγωγή παλμών step για τους δύο κινητήρες γινόταν με τη συνδυασμένη χρήση των timer-counters 0 και 2, οι οποίοι λειτουργούσαν σε λειτουργία CTC (Clear Timer on Compare) και παρήγαγαν παλμούς στις ακίδες PD6 και PB3 αντίστοιχα. Επιπλέον, ο έλεγχος της ολοκλήρωσης της κίνησης σε κάθε άξονα γινόταν μέσω «polling» των αντίστοιχων καταχωρητών. Η συγκεκριμένη υλοποίηση ήταν λεπτομερής και εφευρετική, αλλά είχε ένα φυσικό όριο: όλες οι διεργασίες (λήψη δεδομένων από την USART, υπολογισμοί, παραγωγή παλμών) μοιράζονταν την ίδια ALU. Στο FPGA, οι ίδιες διεργασίες χωρίζονται σε ξεχωριστά modules: η βαθμίδα uart_rx.v χειρίζεται την λήψη των bytes από το UART· η packet_decoder.v αναγνωρίζει το είδος και το περιεχόμενο των πακέτων· η step_gen.v παράγει τους παλμούς για τους δύο άξονες· η laser_ctrl.v ελέγχει την έξοδο του laser· και η uart_tx.v στέλνει το μήνυμα επιβεβαίωσης προς τον host. Όλες λειτουργούν ταυτόχρονα και εξυπηρετούνται από ξεχωριστή λογική του τσιπ χωρίς να εμποδίζουν η μια την άλλη. Στο σχήμα 2.9 αποτυπώνεται η ποιοτική αυτή διαφορά: στην εικόνα α, ο μικροελεγκτής

εκτελεί τις τέσσερις εργασίες σε διαδοχική ροή, με κάθε νέα εργασία να αναστέλλει τις προηγούμενες· στην εικόνα β, οι ίδιες εργασίες λειτουργούν ταυτόχρονα στο FPGA, η καθεμιά με τον δικό της ρυθμό και χωρίς να αλληλο-εμποδίζονται.



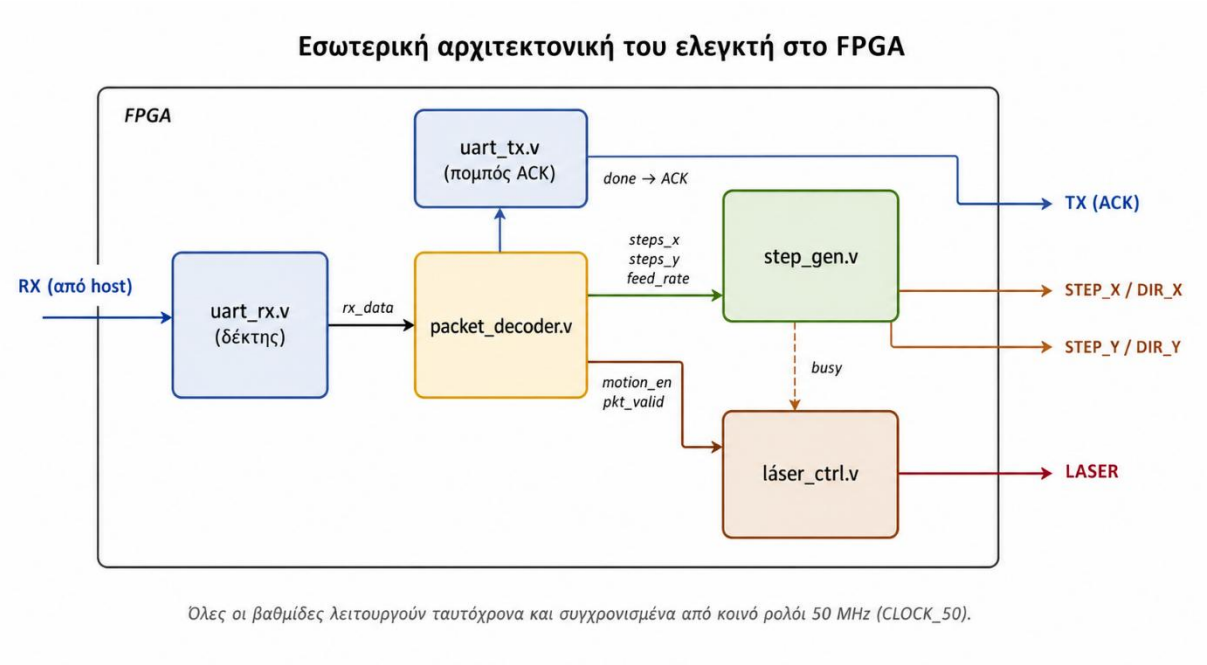
Σχήμα 2.11: Διαφορά διαδοχικής εκτέλεσης σε CPU και παράλληλης εκτέλεσης σε FPGA. (α) Στον μικροελεγκτή, οι εργασίες μοιράζονται την ίδια αριθμητική μονάδα και εκτελούνται σε σειρά, εισάγοντας καθυστερήσεις. (β) Στο FPGA, καθεμιά διαθέτει αποκλειστική λογική και τρέχει ανεξάρτητα από τις υπόλοιπες

Δεύτερον, ντετερμινιστικός χρονισμός. Σε ένα FPGA, το χρονικό διάστημα μεταξύ δύο γεγονότων (ανάμεσα σε δύο διαδοχικούς παλμούς – step) προκύπτει αυστηρά από τον αριθμό των κύκλων ρολογιού που μεσολαβούν, και είναι σταθερό σε όλη τη διάρκεια λειτουργίας του συστήματος. Στην προγενέστερη εργασία [1, §5.3] είχε αναγνωριστεί ότι «κατά τη διάρκεια του χρόνου που μεσολαβεί μέχρι να ενεργοποιηθεί ξανά ο έλεγχος μηδενισμού θέσης, εμφανίζεται μικρό σφάλμα ελέγχου θέσης. Λόγω της διαδοχικής φύσης της εκτέλεσης στον μικροελεγκτή, η ακριβής χρονική στιγμή κάθε παλμού εξαρτιόταν από το σύνολο των ενεργειών που εκτελούνταν στο σύστημα. Στο FPGA το πρόβλημα αυτό εξαλείφεται λόγω της αρχιτεκτονικής: ο παλμός παράγεται όταν ο εσωτερικός μετρητής της step_gen.v φτάσει στην προκαθορισμένη τιμή, χωρίς εξάρτηση από οποιαδήποτε άλλη βαθμίδα του συστήματος.

Τρίτον, αξιοποίηση 32-bit καταχωρητών χωρίς πρόσθετο κόστος. Στην προγενέστερη εργασία [1, §6.1] συζητείται εκτενώς το ζήτημα της επίδρασης που έχει η ρύθμιση microstep στη μέγιστη μετατόπιση που μπορεί να αναπαρασταθεί από έναν 16-bit καταχωρητή. Σε λειτουργία πλήρους βήματος (0,2 mm/step), ένας 16-bit καταχωρητής μπορεί να αναπαραστήσει έως 13.107 mm μετατόπισης· σε λειτουργία 1/32 microstep (0,00625 mm/step) η ίδια μετατόπιση πέφτει στα 410 mm, που είναι σχεδόν ακριβώς το μέγεθος της επιφάνειας εργασίας. Στο FPGA, οι αντίστοιχοι καταχωρητές είναι 32-bit, και η μέγιστη μετατόπιση που μπορούν να αναπαραστήσουν αγγίζει τα 13 km. Επιπλέον, η εργασία με 32-bit ακέραιες πράξεις δεν επιβαρύνει το FPGA σε σχέση με 16-bit, αφού αρκεί απλώς να δηλωθεί το αντίστοιχο εύρος των καταχωρητών στην περιγραφή Verilog.

Η συγκεκριμένη πλατφόρμα στην οποία υλοποιείται ο νέος ελεγκτής είναι η DE1-SoC της Terasic. Πρόκειται για μια αναπτυξιακή πλακέτα χτισμένη γύρω από το ολοκληρωμένο Cyclone V SoC της

Altera (Intel), που συνδυάζει σε ένα και μόνο τσιπ ένα FPGA της οικογένειας Cyclone V και έναν επεξεργαστή ARM Cortex-A9 διπλού πυρήνα. Στην παρούσα υλοποίηση χρησιμοποιείται μόνο η FPGA πλευρά του τσιπ. Ο επεξεργαστής παραμένει ανενεργός. Το ρολόι λειτουργίας είναι το CLOCK_50 της πλακέτας, ένας εξωτερικός κρυσταλλικός ταλαντωτής συχνότητας 50 MHz. Για την σχεδίαση χρησιμοποιήθηκε το λογισμικό Quartus II της Altera (έκδοση 13.1), το οποίο μετατρέπει τον πηγαίο κώδικα Verilog σε αρχείο διαμόρφωσης (.sof). Το αρχείο αυτό μεταφέρεται στη συσκευή μέσω της θύρας USB Blaster που διαθέτει η πλακέτα.



Σχήμα 2.12: Εσωτερική αρχιτεκτονική του ελεγκτή στην παρούσα υλοποίηση. Πέντε ξεχωριστές βαθμίδες Verilog λειτουργούν παράλληλα μέσα στο FPGA: η uart_rx.v αναλαμβάνει τη λήψη bytes, η packet_decoder.v την αναγνώριση πακέτων, η step_gen.v την παραγωγή παλμών step, η laser_ctrl.v την οδήγηση του laser και η uart_tx.v την αποστολή επιβεβαιώσεων. Όλες τρέχουν με κοινό ρολόι 50 MHz.

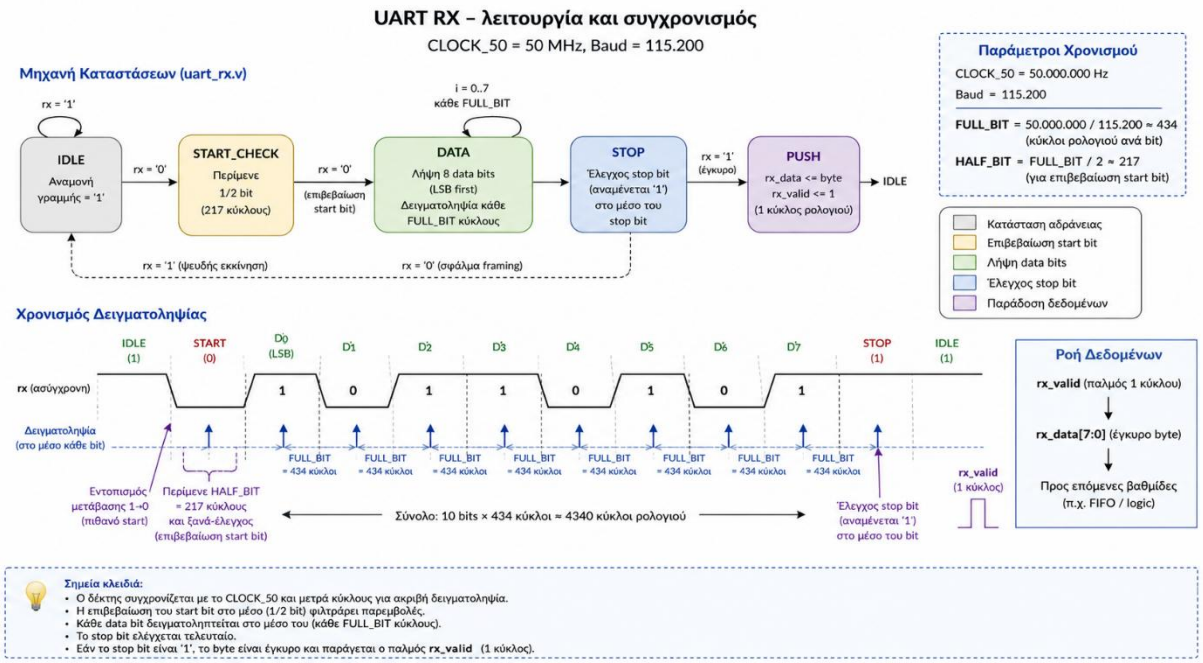
Η συνολική αρχιτεκτονική του ελεγκτή στο FPGA φαίνεται στο σχήμα 2.10. Σε αυτή τη μορφή, το FPGA έχει αναλάβει αποκλειστικά τη λειτουργικότητα που στην προγενέστερη εργασία ήταν στενά συνδεδεμένη με το χρονισμό: τη λήψη των bytes από τη σειριακή θύρα, τη συναρμολόγησή τους σε πακέτα, την παραγωγή των παλμών step με τη σωστή συχνότητα και διάρκεια, και την οδήγηση του laser σε συγχρονισμό με την κίνηση. Όλη η υπόλοιπη λογική (η ανάγνωση του αρχείου .nc, η ερμηνεία του κώδικα G, η διαίρεση των τόξων σε ευθύγραμμα τμήματα, η μετατροπή των μετατοπίσεων από mm σε βήματα) έχει ανατεθεί στον υπολογιστή host. Η συγκεκριμένη κατανομή είναι θεμιτή στη μηχανή της παρούσας εργασίας, καθώς η ταχύτητα κίνησης είναι σχετικά χαμηλή και ο χρόνος μεταφοράς ενός νέου πακέτου από τον host είναι αμελητέος σε σχέση με τη διάρκεια εκτέλεσης κάθε τμήματος κίνησης. Σε εφαρμογές μεγαλύτερων απαιτήσεων σε ταχύτητα, ένα μέρος από αυτή τη λογική θα μπορούσε να μεταφερθεί στο FPGA.

2.6 Σειριακή επικοινωνία UART

Η μετάδοση των εντολών κίνησης από τον υπολογιστή στον ελεγκτή της μηχανής γίνεται μέσω σειριακής επικοινωνίας UART (Universal Asynchronous Receiver-Transmitter). Στην προγενέστερη εργασία [1, §4.1] η αντίστοιχη επιλογή ήταν η USART του Atmega 328p, που υποστηρίζει τόσο σύγχρονη όσο και ασύγχρονη λειτουργία. Χρησιμοποιήθηκε ωστόσο μόνο στην ασύγχρονη μορφή της, που είναι κοινή με το πρωτόκολλο UART. Στην παρούσα υλοποίηση, ολόκληρη η βαθμίδα UART υλοποιείται σε Verilog, ως δύο ανεξάρτητα modules: το uart_rx.v για τη λήψη και το uart_tx.v για τη μετάδοση.

Σε επίπεδο πρωτοκόλλου, η σύνδεση στην παρούσα υλοποίηση ακολουθεί τη ρύθμιση 115200 8N1: ταχύτητα 115.200 bit ανά δευτερόλεπτο, 8 data bits ανά πλαίσιο, χωρίς bit ισοτιμίας, ένα stop bit. Σε κατάσταση αδράνειας η σειριακή γραμμή παραμένει σε λογικό «1» (HIGH). Η μετάδοση των byte ξεκινάει με ένα start bit (μετάβαση σε «0»), ακολουθούν τα 8 data bits μεταδιδόμενα από το λιγότερο σημαντικό προς το περισσότερο σημαντικό, και τερματίζεται με ένα stop bit (επιστροφή στο «1»). Απαραίτητη προϋπόθεση είναι ρύθμιση πομπού και δέκτη στο ίδιο baud rate.

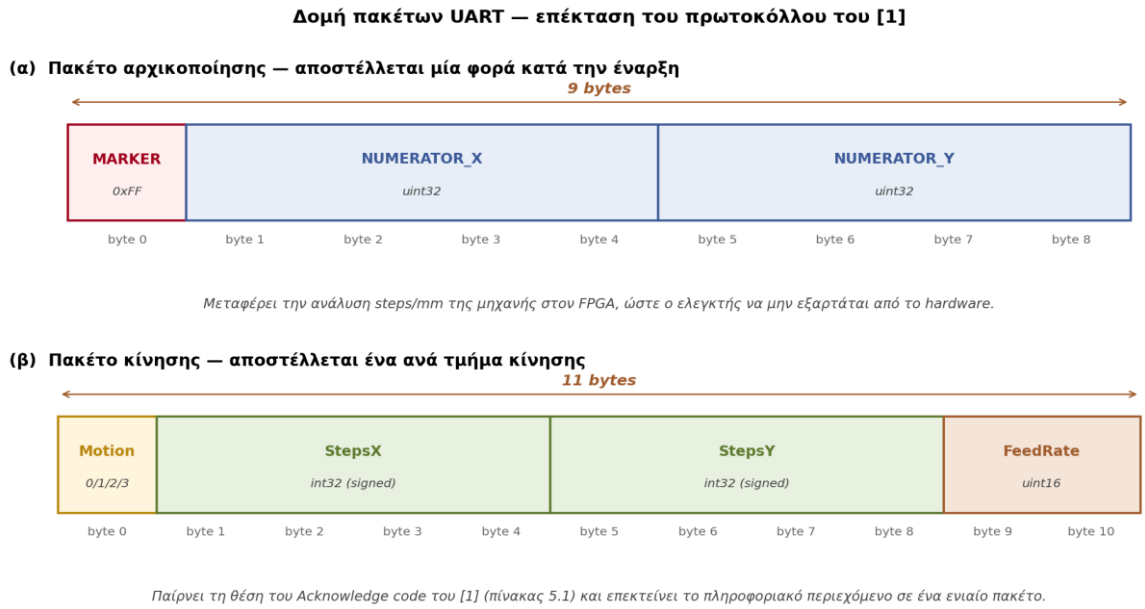
Στο FPGA, η υλοποίηση του δέκτη uart_rx.v στηρίζεται στη συχνότητα του ρολογιού CLOCK_50 (50 MHz). Στα 115.200 baud, η διάρκεια ενός bit αντιστοιχεί σε 50.000.000 / 115.200 ≈ 434 κύκλους ρολογιού — τιμή που ορίζεται ως σταθερά FULL_BIT στον πηγαίο κώδικα. Η μηχανή καταστάσεων του δέκτη ξεκινά σε κατάσταση αδράνειας· όταν εντοπίσει μετάβαση σε «0», περιμένει 217 κύκλους (το μισό του FULL_BIT) και ξανά-ελέγχει τη γραμμή για επιβεβαίωση του start bit, ώστε να φιλτράρει παρεμβολές. Στη συνέχεια δειγματοληπτεί τα 8 data bits σε διαστήματα 434 κύκλων μεταξύ τους, με αποτέλεσμα κάθε δείγμα να λαμβάνεται ακριβώς στο μέσο του αντίστοιχου bit. Το stop bit ελέγχεται τελευταίο, και αν η γραμμή είναι σε «1» όπως αναμένεται, το byte θεωρείται έγκυρο και προωθείται στις επόμενες βαθμίδες μέσω του σήματος rx_valid (παλμός ενός κύκλου ρολογιού).



Σχήμα 2.13: *UART επικοινωνία στο FPGA.*

Από την πλευρά του υπολογιστή host, η σύνδεση στη σειριακή γραμμή γίνεται μέσω εξωτερικής γέφυρας USB ↔ UART (στη συγκεκριμένη υλοποίηση χρησιμοποιείται μονάδα CP2102 της Silicon Labs). Η γέφυρα παρουσιάζεται στο λειτουργικό σύστημα του υπολογιστή ως εικονική θύρα COM, στην οποία τυπικές εφαρμογές μπορούν να γράψουν και να διαβάσουν με τις κοινές λειτουργίες ανοίγματος αρχείου. Η εφαρμογή του host είναι γραμμένη σε Python και χρησιμοποιεί τη βιβλιοθήκη pyserial: ανοίγει τη θύρα με τις παραμέτρους 115200 8N1, διαβάζει το αρχείο .nc που έχει παραχθεί από το CAD2GCode, μετατρέπει κάθε εντολή σε ένα δυαδικό πακέτο σταθερού μήκους και το στέλνει στον ελεγκτή με μία κλήση write.

Η εφαρμογή ορίζει ένα δικό της πρωτόκολλο πακέτων. Στην προγενέστερη εργασία [1, πίνακας 5.1] η ίδια ανάγκη είχε λυθεί με ένα 8-bit acknowledge code που στελνόταν πρώτα στον μικροελεγκτή για να τον ενημερώσει ποιά εντολή G ή M ακολουθεί, και αμέσως μετά τα δεδομένα της κίνησης (πλήθος βημάτων X, περίοδος OCR0A, περίοδος OCR2A, byte εκκίνησης σε PORT B). Στην παρούσα υλοποίηση η ίδια πληροφορία ενσωματώνεται σε ένα πακέτο σταθερού μήκους (motion packet) με τη μορφή που φαίνεται στο σχήμα 2.12.

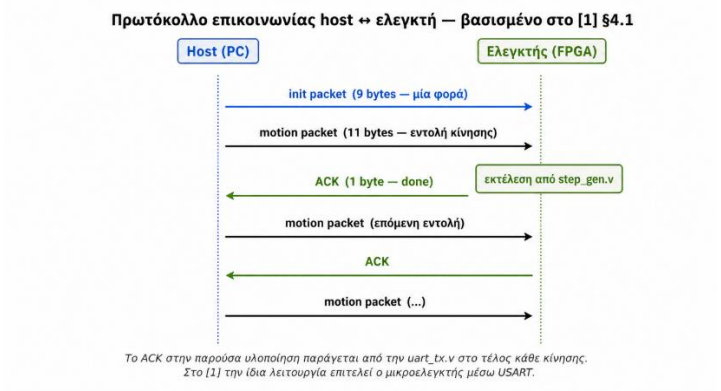


Σχήμα 2.14: Δομή των πακέτων που ανταλλάσσονται μέσω UART στην παρούσα υλοποίηση. (α) Πακέτο αρχικοποίησης: 9 bytes που στέλνονται μία φορά κατά την έναρξη και μεταφέρουν τη μηχανική ανάλυση steps/mm στο FPGA. (β) Πακέτο κίνησης: 11 bytes ανά εντολή, με τα πεδία Motion (τύπος εντολής G), StepsX, StepsY (προσημασμένος αριθμός βημάτων) και FeedRate.

Διακρίνονται δύο τύποι πακέτων. Το **πακέτο αρχικοποίησης** (init packet) έχει μήκος 9 bytes, ξεκινά με τον χαρακτηριστικό 0xFF και μεταφέρει τους δύο 32-bit αριθμητές NUMERATOR_X και NUMERATOR_Y που χρησιμοποιεί η βαθμίδα step_gen.v για τη μετατροπή της επιθυμητής ταχύτητας προώθησης σε περίοδο παλμών step. Οι αριθμητές αυτοί υπολογίζονται στον host με βάση τον τύπο NUMERATOR = f_clk · 60 / (steps_per_mm), όπου f_clk = 50 MHz. Η συγκεκριμένη επιλογή κωδικοποίησης μετακινεί την εξάρτηση από τη μηχανική ανάλυση από το FPGA στον host: το ίδιο ακριβώς bitstream FPGA θα δούλευε σε διαφορετική μηχανή με διαφορετική ρύθμιση microstep ή διαφορετικό ιμάντα, με μόνη αλλαγή την τιμή του NUMERATOR στο πακέτο αρχικοποίησης.

Το **πακέτο κίνησης** (motion packet) έχει μήκος 11 bytes και αποστέλλεται ένα ανά εντολή κίνησης. Το πρώτο byte είναι το πεδίο Motion, το οποίο κωδικοποιεί τον τύπο της εντολής G με την αντιστοίχιση 0=G00, 1=G01, 2=G02 (CW arc segment), 3=G03 (CCW arc segment). Τα επόμενα 4+4 bytes περιέχουν τους προσημασμένους αριθμούς βημάτων στους δύο άξονες (StepsX, StepsY ως 32-bit signed ακέραιοι), και τα τελευταία 2 bytes την επιθυμητή ταχύτητα προώθησης ως 16-bit unsigned. Όλα τα πεδία μεταδίδονται με MSB πρώτο. Όταν το συνολικό πλήθος των bytes ενός πακέτου φτάσει στη βαθμίδα packet_decoder.v, εκείνη υψώνει το σήμα pkt_valid για έναν κύκλο ρολογιού, ώστε να ενεργοποιηθεί η βαθμίδα step_gen.v που αναλαμβάνει την εκτέλεση.

Σε επίπεδο ροής, η ανταλλαγή ανάμεσα στον host και τον ελεγκτή ακολουθεί ένα μοτίβο που είναι ταυτόσημο σε λογικό επίπεδο με αυτό της προγενέστερης εργασίας. Στο [1, §4.1] περιγράφεται με σαφήνεια: «Η εφαρμογή στέλνει την εντολή στον μικροελεγκτή, περιμένει την επιβεβαίωση λήψης από τον μικροελεγκτή, επιστρέφει στο πρώτο βήμα». Η ίδια λογική ανταλλαγής αναπαράγεται και στην παρούσα υλοποίηση (Σχήμα 2.11): ο host στέλνει ένα motion packet, ο ελεγκτής το λαμβάνει, εκτελεί την κίνηση μέσω της step_gen.v, και στο τέλος της κίνησης η βαθμίδα uart_tx.v στέλνει πίσω ένα byte επιβεβαίωσης (ACK). Ο host δεν αποστέλλει το επόμενο πακέτο πριν λάβει το ACK, έτσι αποτρέπεται η συσσώρευση πακέτων στον buffer εισόδου του FPGA όταν η μηχανική κίνηση είναι αργότερη από τη σειριακή μετάδοση.



Σχήμα 2.15: *Πρωτόκολλο επικοινωνίας host ↔ ελεγκτή.* Η ίδια λογική «πακέτο – εκτέλεση – ACK» που είχε υλοποιηθεί στον Atmega 328p [1, §4.1] διατηρείται και στην παρούσα υλοποίηση, με το ACK να παράγεται πλέον από τη βαθμίδα uart_tx.v του FPGA.

Δύο επιπλέον βαθμίδες συμπληρώνουν την εικόνα. Πρώτον, η βαθμίδα packet_decoder.v ενσωματώνει έναν timer ασφαλείας ο οποίος μετρά κύκλους ρολογιού από τη λήψη του τελευταίου byte. Αν περάσουν 100 ms (5 εκατομμύρια κύκλοι στα 50 MHz) χωρίς νέο byte, ο μετρητής εσωτερικών byte μηδενίζεται και η βαθμίδα επιστρέφει σε κατάσταση αναμονής νέου πακέτου. Αυτή η βαθμίδα προστατεύει το σύστημα από κακό συγχρονισμό: σε περίπτωση που χαθεί έστω και ένα byte κατά τη μετάδοση, το σύστημα ανανεώνεται αυτόματα μετά από σύντομη αδράνεια αντί να συσσωρεύει σφάλμα μετάβασης. Δεύτερον, εκτός από τα motion packets, υπάρχει και ένα ξεχωριστό σύντομο πακέτο για τον άμεσο έλεγχο του laser (2 bytes με marker 0xFE), το οποίο εξυπηρετεί τη χειροκίνητη ενεργοποίηση - απενεργοποίηση της κεφαλής από τη γραφική διεπαφή χρήστη και δεν απαιτεί ACK επιβεβαίωσης.

Η συνολική σχεδίαση του πρωτοκόλλου διατηρεί την αρχιτεκτονική του αρχικού project [1] (αίτημα από host, εκτέλεση από τον ελεγκτή, επιβεβαίωση πριν το επόμενο αίτημα) και την επεκτείνει με δύο κατευθύνσεις. Πρώτον, ενοποιεί την κωδικοποίηση εντολής και δεδομένων κίνησης σε ένα ενιαίο πακέτο σταθερού μήκους, αντί για ξεχωριστή αποστολή. Δεύτερον, εισάγει ένα ξεχωριστό πακέτο

αρχικοποίησης που μεταφέρει τη μηχανική ανάλυση από τον host στον ελεγκτή, καθιστώντας το FPGA ανεξάρτητο από τα συγκεκριμένα μηχανικά χαρακτηριστικά της μηχανής. Αυτές οι δύο τροποποιήσεις δεν αλλάζουν τη φύση της επικοινωνίας αλλά την προσαρμόζουν στις δυνατότητες της νέας πλατφόρμας ελέγχου, που σε αντίθεση με τον Atmega 328p [1, §6.1], δεν εμφανίζει περιορισμούς στο εύρος των εσωτερικών της καταχωρητών.

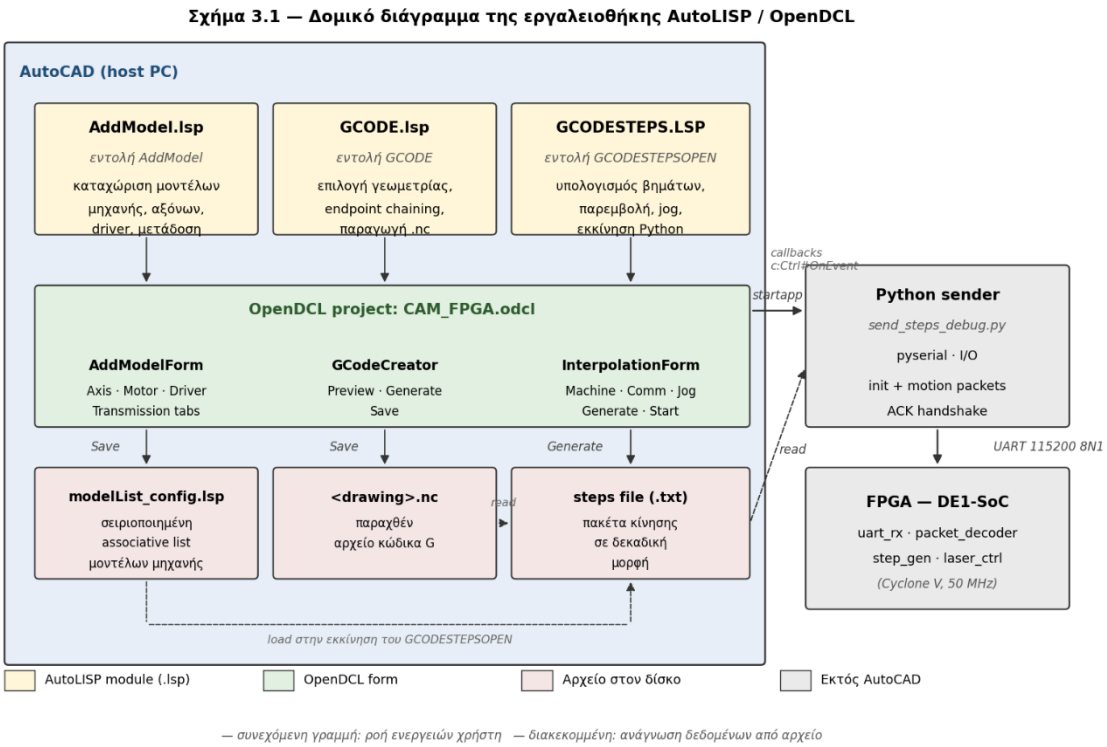
Κεφάλαιο 3ο: Ανάπτυξη λογισμικού CAD-CAM

3.1 Αρχιτεκτονική εργαλείων (AutoLISP, OpenDCL)

Στην προγενέστερη υλοποίηση δύο αξόνων [1], η μετατροπή του σχεδίου σε κώδικα G γινόταν από μια εφαρμογή (σε C++), που διάβαζε απευθείας ένα εξαγμένο αρχείο .dxf. Ο χρήστης ολοκλήρωνε τη σχεδίαση στο λογισμικό CAD, αποθήκευε το αρχείο σε μορφή .dxf και στην συνέχεια χρησιμοποιούσε την εφαρμογή. Στην παρούσα εργασία η CAM εφαρμογή βρίσκετε εντός του CAD λογισμικού, ώστε ο σχεδιαστής να εργάζεται με στο σχέδιο και όχι με το αρχείο. Η αλλαγή αυτή επιβάλλει την χρήση γλώσσας προγραμματισμού μέσα στο περιβάλλον του CAD η οποία έχει άμεση πρόσβαση στις οντότητες του σχεδίου. Η AutoLISP, είναι μια επέκταση της γλώσσας Lisp προσαρμοσμένη για CAD λογισμικό. Αναπτύχθηκε το 1986 και είναι πλήρως υποστηριζόμενη στις σύγχρονες εκδόσεις.

Η AutoLISP από μόνη της δεν παρέχει σύγχρονη γραφική διεπαφή. Η λύση για ανάπτυξη UI είναι τα αρχεία .dcl, τα οποία όμως υποστηρίζουν μόνο τα βασικά γραφικά στοιχεία - widgets (radio buttons, edit boxes, list boxes) με περιορισμένη ευελιξία διάταξης, εικόνες χαμηλής ανάλυσης και χωρίς υποστήριξη tabs ή slide previews. Για να καλυφθεί το κενό αυτό, η παρούσα εργασία βασίζεται στην OpenDCL, μια ανοιχτού κώδικα προέκταση που από το 2008 προσφέρει ένα συμβατικό toolkit παραθύρων (combo boxes με drop-down callbacks, tab controls, slide preview windows, message boxes) και ένα περιβάλλον σχεδίασης φορμών (OpenDCL Studio). Όλες οι φόρμες της εργαλειοθήκης σχεδιάζονται και αποθηκεύονται σε ένα κοινό αρχείο, το CAM_FPGA.odcl, το οποίο φορτώνεται με την κλήση *dcl_Project_Load* κατά την πρώτη εκτέλεση οποιασδήποτε εντολής και παραμένει στη μνήμη μέχρι το κλείσιμο αχρείου CAD. Κάθε φόρμα εμφανίζεται με την μέθοδο *dcl_Form_Show*, και η επικοινωνία ανάμεσα στον πυρήνα της AutoLISP και τα στοιχεία της φόρμας γίνεται μέσω callbacks. Σε κάθε event ενός widget (π.χ. κλικ σε κουμπί, αλλαγή επιλογής σε combo box, άνοιγμα drop-down) η OpenDCL καλεί την αντίστοιχη συνάρτηση (π.χ. *c:GenerateButton#OnClicked* για το πάτημα του κουμπιού GenerateButton) και την εκτελεί.

Η συνολική εργαλειοθήκη απαρτίζεται από τρία ανεξάρτητα αρχεία AutoLISP, καθένα από τα οποία αντιστοιχεί σε μία κύρια εντολή που ο χρήστης καλεί από το μενού (Ribbon Bar) του CAD: AddModel.lsp για την καταχώριση των μηχανολογικών χαρακτηριστικών της μηχανής (ενότητα 3.2), GCODE.lsp για την παραγωγή κώδικα G από την επιλεγμένη γεωμετρία (ενότητα 3.3), και GCODESTEPS.LSP για τον υπολογισμό βημάτων ανά άξονα και τη γεωμετρική ταύτιση σχεδίου με το δοκίμιο πριν από την έναρξη μιας κατεργασίας (ενότητες 3.4 και 3.5). Οι τρεις εντολές μοιράζονται το ίδιο OpenDCL project (CAM_FPGA.odcl) (με ξεχωριστή φόρμα η καθεμιά). Στο σχήμα 3.1 αποτυπώνεται συνολικά η αρχιτεκτονική, με τις τρεις βαθμίδες AutoLISP από πάνω, την φόρμα OpenDCL στη μέση, τα αρχεία που χρησιμοποιούν στο κάτω μέρος, και τον εκτελέσιμο αρχείο Python που μεταφέρει τα δεδομένα στο FPGA. Η επιλογή να κρατηθούν χωριστά τα τρία αρχεία πηγαίου κώδικα, επιτρέπει σε κάθε εργαλείο να μπορεί να χρησιμοποιηθεί, χωρίς να συνεπάγεται την παρουσία των άλλων δύο, ενώ διευκολύνει τη συντήρηση καθώς κάθε αρχείο αποτελείται από πολλές γραμμές κώδικα.



Σχήμα 3.1: *Δομικό διάγραμμα της εργαλειοθήκης AutoLISP / OpenDCL.* Οι τρεις εντολές (AddModel, GCODE, GCODESTEPSOPEN) εκκινούν ισάριθμες φόρμες από το κοινό CAM_FPGA.odcl. Η μεταξύ τους ανταλλαγή πληροφοριών γίνεται μέσω αρχείων στον δίσκο. Η σειριακή επικοινωνία με το FPGA ανατίθεται σε εξωτερικό script Python που καλείται από την InterpolationForm.

Η ανταλλαγή πληροφοριών ανάμεσα στα τρία εργαλεία πραγματοποιείται μέσω αρχείων στον δίσκο και όχι μέσω κοινών μεταβλητών μνήμης. Η σχεδιασμός αυτός εξασφαλίζει την διατήρηση των δεδομένων ακόμη και όταν ο χρήστης κλείσει το CAD. Η βασική δομή δεδομένων του συστήματος είναι μια ένθετη λίστα LISP (associative list) με ονομασία *modelList*, στην οποία κάθε καταχώριση αντιστοιχεί σε ένα μοντέλο μηχανής και περιέχει για κάθε άξονά του τα πεδία *Dimensions*, *Motor*, *Driver* και *Transmission*.

Η λίστα αυτή αποθηκεύεται από το AddModel.lsp στο αρχείο modelList_config.lsp, που όταν φορτώνεται με τη βασική συνάρτηση *load*, αναπαράγει επί τόπου τη δομή στη μνήμη. Όταν αργότερα ενεργοποιηθούν οι άλλες δύο εντολές, διαβάζουν το ίδιο αρχείο, ώστε να εμφανίσουν στα combo boxes τα ίδια ονόματα μηχανών που έχει διαμορφώσει ο χρήστης. Παρομοίως, η έξοδος του GCODE.lsp (αρχείο .nc) γίνεται είσοδος στο GCODESTEPS.LSP για μετατροπή των εντολών του κώδικα G σε βήματα.

Η σειριακή επικοινωνία δεν μπόρεσε να ενσωματωθεί στην AutoLISP διότι δεν διαθέτει υποστήριξη ανάλογης βιβλιοθήκης. Για τον λόγο αυτό χρησιμοποιήθηκε ένα Python script (send_steps_debug.py) έτσι ώστε να αποδεσμεύσει τον κώδικας της AutoLISP από το υλικό, την ανεξαρτησία από το CAD και τη μελλοντική αντικατάσταση του (π.χ. υλοποίηση με Ethernet ή WebSocket), χωρίς αλλαγές στη φόρμα του χρήστη. Σε επίπεδο αρχιτεκτονικής, η AutoLISP + OpenDCL αποτελεί το **front-end** του CAM. Αντλεί δεδομένα από το CAD, παρουσιάζει τις επιλογές στον σχεδιαστή και παράγει αρχεία· οι λεπτομέρειες της φυσικής επικοινωνίας με τον ελεγκτή ανήκουν σε άλλο επίπεδο και αναλύονται στο κεφάλαιο 4.

3.2 Καταχώριση μοντέλου μηχανής και παραμέτρων

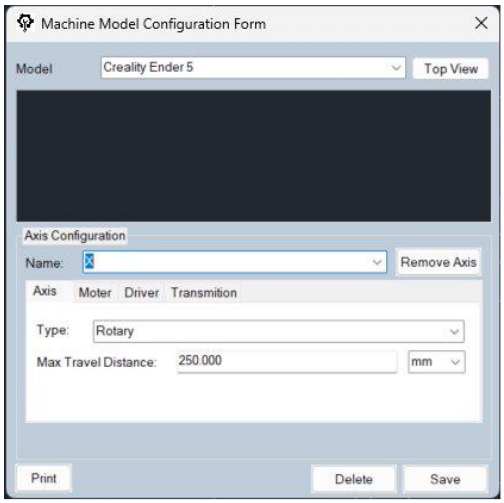
Ο ελεγκτής FPGA που αναλύεται σε επόμενο κεφάλαιο δέχεται από τον host υπολογιστή ως δεδομένα τον αριθμό των βημάτων που πρέπει να εκτελεστούν ανά άξονα. Η μετατροπή της εντολής G «μετάβαση στο σημείο (X, Y) σε χιλιοστά» στην ισοδύναμη ακολουθία βημάτων εξαρτάται από ένα πλήθος μηχανολογικών παραμέτρων της εκάστοτε μηχανής (τη γωνία ανά βήμα του κινητήρα, το microstep του driver, τον αριθμό δοντιών του τροχού και το βήμα του). Επιπλέον, κάθε μηχανή έχει συγκεκριμένα όρια διαδρομής ανά άξονα που μπορεί να μετακινηθεί η κεφαλή, και πρέπει κάθε εντολή μετακίνησης να περιορίζεται από αυτά τα όρια για αποφυγή πρόσκρουσης.

Αν οι παραπάνω παράμετροι αποθηκεύονταν σε σταθερές μέσα στον πηγαίο κώδικα, η αλλαγή κάποιον χαρακτηριστικών της μηχανής θα απαιτούσε ταυτόχρονα και ανάλογες αλλαγές μέσα στον κώδικα. Για τον λόγο αυτό σχεδιάστηκε η εντολή AddModel η οποία ανοίγει τη φόρμα Machine Model Configuration (σχήμα 3.2), που οργανώνει όλες τις παραμέτρους σε τέσσερις θεματικές ομάδες - tabs (άξονας, κινητήρας, driver, μετάδοση κίνησης) και αποθηκεύει τις τιμές των παραμέτρων σε μια δομή λίστας που ονομάζεται modelList. Παρακάτω φαίνεται ένα παράδειγμα της λίστας:

```
(setq modelList
  '(
    ("Creality Ender 5" .                  ; όνομα μοντέλου
      (("X" .                               ; πρώτος άξονας
         (("Dimensions" .  (("TravelDistance" . "250")
                            ("Type"           . "Rotary")))
          ("Motor"      .  (("Model"     . "Nema 17")
                            ("MotorType" . "Stepper")
                            ("StepAngle" . "1,8")))
          ("Driver"     .  (("Model"    . "DVR8825")
                            ("StepMode" . "1/32")))
          ("Transmission" . (("Method" . "pulley-belt")
                             ("Teeth"  . "20")
                             ("Pitch"  . "2")))))
       ("Y" .                                ; δεύτερος άξονας – ίδια δομή
         ((...)))))
    ("Custom Machine" . ())                 ; μοντέλο χωρίς αξονες ακόμη
  )
)
```

Παράδειγμα κώδικα 3.1 — Δομή της **modelList.** Παίρνει τιμές από την AddModel στο αρχείο modelList_config.lsp. Κάθε εγγραφή είναι ζεύγος (όνομα-μοντέλου . λίστα-αξόνων)· κάθε άξονας ανάγεται σε τέσσερις θεματικές υπό-λίστες (Dimensions, Motor, Driver, Transmission), και κάθε υπό-λίστα σε μεμονωμένα ζεύγη (πεδίο . τιμή).

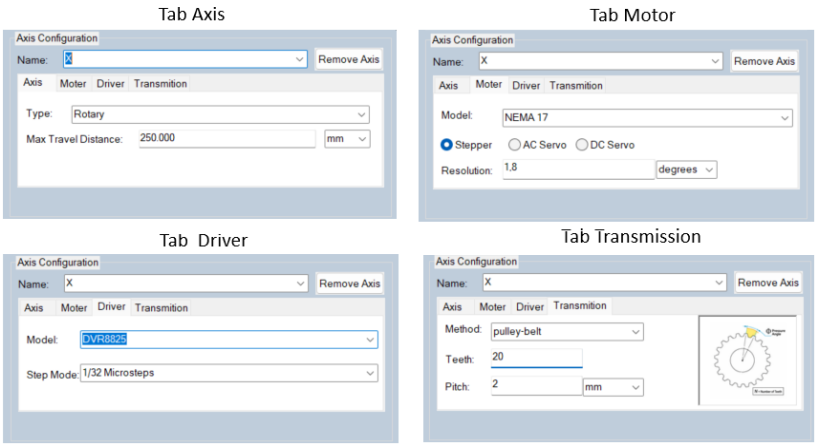
Η φόρμα είναι οργανωμένη σε τρεις λογικές ζώνες. Στο πάνω μέρος βρίσκεται ένα combo box “Model” που εμφανίζει όλες τις ήδη καταχωρημένες μηχανές, μαζί με το κουμπί “Top View” που επιτρέπει τη σύνδεση μιας κάτοψης (αρχείο .sld του CAD) με το επιλεγμένο μοντέλο. Επίσης υπάρχει η περιοχή “SlideView” όπου εμφανίζεται μια κάτοψη της μηχανής. Στο μέσο της φόρμας βρίσκεται το πλαίσιο Axis Configuration, με το combo box “Name” για την επιλογή άξονα και τέσσερις καρτέλες (Axis, Motor, Driver, Transmission). Στο κάτω μέρος υπάρχουν τα κουμπιά “Delete” και “Save”.



Σχήμα 3.2: Φόρμα Machine Model Configuration στην καρτέλα Axis. Στην κορυφή φαίνεται το combo box για επιλογή μοντέλου (Creality Ender 5) και το κουμπί Top View για τη σύνδεση εικόνας προεπισκόπησης

Όταν ο χρήστης ανοίγει το dropdown “Model”, η εφαρμογή διαβάζει τη λίστα των ήδη καταχωρημένων μοντέλων και την περνά στις επιλογές του combo box. Αμέσως μόλις ο χρήστης επιλέξει κάποιο μοντέλο μηχανής, εμφανίζει στην περιοχή SlideView την εικόνα της κάτοψης που έχει αντιστοιχιστεί στο συγκεκριμένο μοντέλο μηχανής.

Στην περιοχή Axis Configuration το combo box “Name” περιέχει το όνομα όλων των δηλωμένων αξόνων της μηχανής (X,Y,Z ….).Οι παράμετροι του εκάστοτε άξονα είναι ομαδοποιημένες σε τέσσερις καρτέλες - tabs στη ζώνη Axis Configuration (σχήμα 3.3). Στην καρτέλα Axis καταχωρείται ο τύπος της κίνησης (Rotary ή Linear) και η μέγιστη επιτρεπτή διαδρομή του άξονα. Στην καρτέλα Motor ορίζεται το μοντέλο του κινητήρα (λ.χ. NEMA 17), η ανάλυση σε μοίρες ανά βήμα (λ.χ. 1,8°) και ο τύπος του, μέσω option buttons (Stepper, AC Servo, DC Servo). Στην καρτέλα Driver ορίζετε το μοντέλο του οδηγού (π.χ. DRV8825) και το microstep (1/8, 1/16, 1/32). Τέλος, στην καρτέλα Transmission καταχωρείται η μέθοδος μετάδοσης κίνησης από τον κινητήρα στον άξονα (pulley-belt ή leadscrew) μαζί με τις δύο γεωμετρικές παραμέτρους που εμπλέκονται στους υπολογισμούς της ενότητας 3.4 (αριθμός δοντιών τροχαλίας και βήμα ιμάντα ή σπείρας).



Σχήμα 3.3: Οι τέσσερις καρτέλες της ζώνης Axis Configuration: Axis (τύπος και μέγιστη διαδρομή), Motor (μοντέλο, τύπος, ανάλυση), Driver (μοντέλο, microsteps) και Transmission (μέθοδος, δόντια, βήμα).

Η συμπλήρωση των τεσσάρων καρτελών δεν γίνεται από το μηδέν κάθε φορά. Όταν ο χρήστης επιλέξει έναν υφιστάμενο άξονα από το combo box “Name”,όλα τα αντίστοιχα πεδία παίρνουν τις τιμές που

είναι αποθηκευμένες στην λίστα. Η αντίστροφη ροή (εισαγωγή τιμής σε κάποια παράμετρο μέσω του αντίστοιχου πεδίου της φόρμας πίσω στη λίστα "modelList") εκτελείται όταν ο χρήστης πατήσει Save.

Πέρα από την αποθήκευση, η φόρμα προσφέρει τρεις λειτουργίες διαχείρισης που τροποποιούν τη "modelList". Το κουμπί "Remove Axis" που διαγράφει τον επιλεγμένο άξονα του μοντέλου μαζί με όλα τα ένθετα δεδομένα Motor, Driver και Transmission. Το κουμπί "Delete" στο κάτω μέρος, διαγράφει ολόκληρη την εγγραφή του επιλεγμένου μοντέλου από τη λίστα.

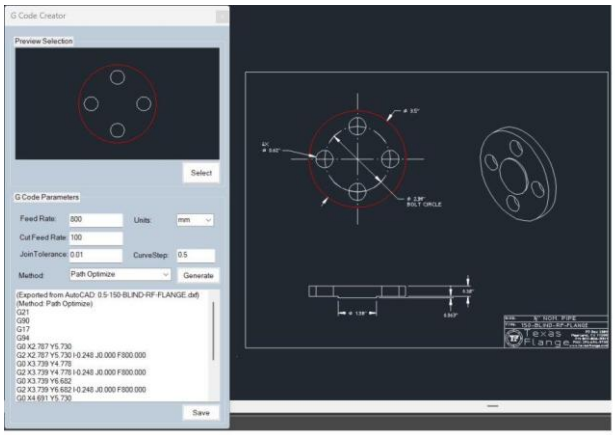
3.3 Παραγωγή G-code από σχέδιο

Ο βασικός στόχος ενός λογισμικού CAM είναι η μετατροπή του σχεδίου σε ακολουθία εντολών G που περιγράφουν την μετακίνηση της κεφαλής σε προ-καθορισμένη τροχιά. Η εντολή GCODE ανοίγει τη φόρμα **G Code Creator** (σχήμα 3.4), μέσω της οποίας ο χρήστης ορίζει τις παραμέτρους της κίνησης και παράγει το αρχείο του κώδικα (.nc).

Στο πάνω μέρος της φόρμας διακρίνετε η περιοχή "Preview Selection". Με το κουμπί "Select" ο χρήστης επιλέγει τις οντότητες που συνθέτουν την γεωμετρία στην οποία θα κινηθεί η κεφαλή. Επίσης ένα παράθυρο προεπισκόπησης εμφανίζει την εικόνα της επιλεγμένης γεωμετρίας.

Στο κέντρο βρίσκεται η περιοχή "G Code Parameters" με έξι πεδία που καθορίζουν τις παραμέτρους της κίνησης. Η παράμετρος "Feed Rate" ορίζει την ταχύτητα μετακίνησης κατά την κατεργασία, και το "Cut Feed Rate" ορίζει την ταχύτητα για κοπή. Στο πεδίο "Units" επιλέγετε μεταξύ χιλιοστών και ίντσας ως μονάδα μέτρησης του παραγόμενου κώδικα. Το πεδίο "JoinTolerance" ορίζει την ανοχή μέσα στα όρια της οποίας δύο τελικά σημεία διαφορετικών οντοτήτων θεωρούνται ταυτόσημα. Το "CurveStep" καθορίζει την πυκνότητα δειγματοληψίας για καμπύλες, ενώ το "Method" επιλέγει την σειρά αναπαραγωγής των επιλεγμένων οντοτήτων δίνοντας δύο επιλογές: "Path Optimize" για βέλτιστη διαδρομή και "On User Selection" για διατήρηση της σειράς επιλογής.

Το κάτω τμήμα της φόρμας υπάρχει ένα TextBox όπου εμφανίζεται ο παραγόμενος κώδικας G με το πάτημα του κουμπιού "Generate". Ο χρήστης μπορεί να τον ελέγξει και αν θέλει να τον επεξεργαστεί, πριν την αποθήκευση. Με το κουμπί "Save", ο κώδικας γράφεται σε αρχείο .nc στον δίσκο. Όλες οι παραπάνω ενέργειες εκτελούνται παράλληλα με το σχέδιο σε κατάσταση επεξεργασίας. Ο χρήστης μπορεί να επιστρέψει στο σχέδιο κρατώντας ανοιχτή την φόρμα και να τροποποιήσει την γεωμετρία ή να επιλέξει διαφορετικό υποσύνολο, και να παράγει εκ νέου τον G-code χωρίς να κλείσει τη φόρμα.



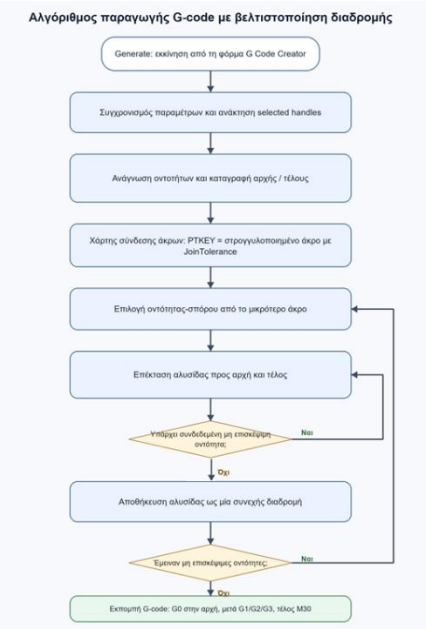
Σχήμα 3.4: *Φόρμα G Code Creator (εντολή **GCODE**).* Αριστερά διακρίνονται, η προεπισκόπηση της επιλεγμένης γεωμετρίας, οι παράμετροι κατεργασίας και ο παραγόμενος κώδικας G. Δεξιά φαίνεται το σχέδιο στο CAD από το οποίο γίνεται η επιλογή των οντοτήτων μέσω του κουμπιού Select

Όταν ο χρήστης πατήσει “Generate”, εκτελείται ο αλγόριθμος παραγωγής G-code (σχήμα 3.5). Με την επιλογή “On user selection” η σειρά των εντολών είναι ίδια με την σειρά επιλογής των οντοτήτων από τον χρήστη. Με την επιλογή “Path Optimazation” ο αλγόριθμος εντοπίζει τις οντότητες που έχουν κοινές συντεταγμένες μεταξύ τους, παράγοντας μια σειρά εντολών όπου το τέλος της μιας οντότητας συμπίπτει με την αρχή της επόμενης (όπου αυτό είναι εφικτό). Με αυτόν τον τρόπο περιορίζονται οι άσκοπες εντολές G0, δηλαδή οι γρήγορες μετακινήσεις χωρίς κατεργασία, και η διαδρομή της κεφαλής γίνεται βελτιστοποιημένη.

Ο αλγόριθμος ξεκινά από τη λίστα των handles (μοναδικό id που χαρακτηρίζει μια οντότητα) που έχουν αποθηκευτεί κατά την επιλογή των οντοτήτων. Κάθε handle επαναφέρει την αντίστοιχη οντότητα του CAD και από αυτήν καταγράφονται τα βασικά γεωμετρικά στοιχεία που χρειάζονται για την παραγωγή G-code: τύπος οντότητας, αρχικό σημείο, τελικό σημείο (για ευθείες) και, όπου απαιτείται (τόξο), κέντρο και φορά τόξου.

Στη συνέχεια δημιουργείται ένας πίνακας - «χάρτης» σύνδεσης των άκρων (αρχικό ή τελικό σημείο) οντοτήτων. Κάθε αρχικό ή τελικό σημείο οντότητας στρογγυλοποιείται με ανοχή που έχει δηλωθεί στο JoinTolerance. Έτσι δύο άκρα που στο σχέδιο θεωρούνται ενωμένα, αλλά διαφέρουν ελάχιστα αριθμητικά, αντιμετωπίζονται ως το ίδιο σημείο. Με βάση αυτόν τον χάρτη επιλέγεται μια αρχική οντότητα (seed) και η αλυσίδα επεκτείνεται προς την αρχή και προς το τέλος, αντιστρέφοντας τη φορά μιας οντότητας όταν χρειάζεται.

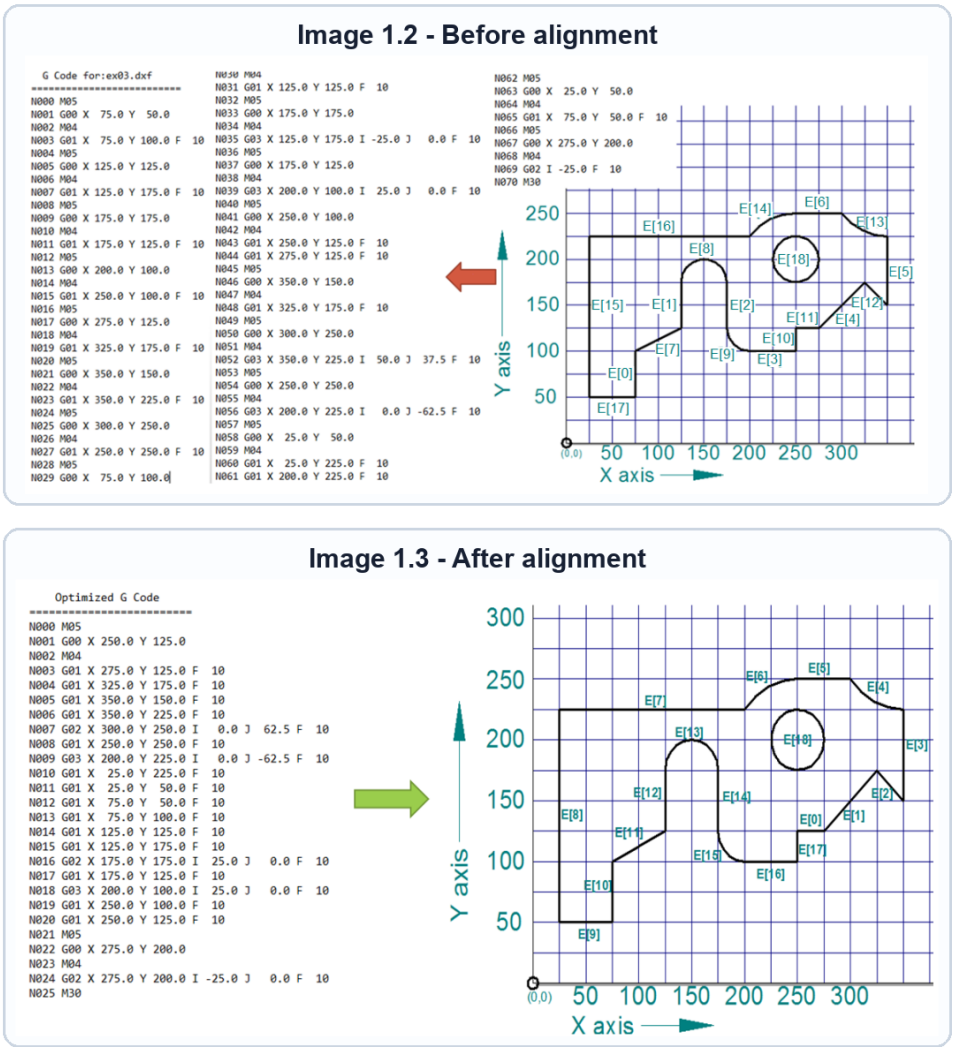
Όταν δεν υπάρχει άλλη συνδεδεμένη μη επισκέψιμη οντότητα, η τρέχουσα αλυσίδα αποθηκεύεται και η διαδικασία επαναλαμβάνεται για τις υπόλοιπες οντότητες. Το αποτέλεσμα είναι μία αλυσίδα για κάθε ανεξάρτητη γεωμετρική ομάδα. Κατά την τελική εκπομπή γράφεται μία εντολή G0 για τη μετάβαση στην αρχή κάθε αλυσίδας και έπειτα συνεχόμενες εντολές G1, G2 ή G3 για τις οντότητες που την αποτελούν.



Σχήμα 3.5: Διάγραμμα ροής του αλγορίθμου παραγωγής G-code στο GCODE.lsp. Η επιλογή του χρήστη διαβάζεται ως σύνολο οντοτήτων, καταγράφονται τα άκρα τους, σχηματίζεται χάρτης σύνδεσης άκρων και τελικά παράγονται οι εντολές G0/G1/G2/G3.

Ως πραγματικό παράδειγμα εφαρμογής χρησιμοποιείται το σχέδιο - Image 1.3 της εργασίας Two Axis CNC - Hardware and Software Development [1]. Το σχέδιο περιλαμβάνει δεκαεννέα γεωμετρικές οντότητες, οι οποίες πριν από την ευθυγράμμιση είχαν παραχθεί σε τυχαία σειρά. Στην προηγούμενη εργασία αναφέρεται ότι ο G-code πριν από την ευθυγράμμιση είχε 70 εντολές, ενώ μετά την αναδιάταξη των οντοτήτων μειώθηκε σε 25 εντολές και σε δύο μόνο εναλλαγές μεταξύ γρήγορης μετακίνησης και μετακίνησης με κατεργασία. Στο παρόν κείμενο το ίδιο παράδειγμα χρησιμοποιείται για να φανεί πώς θα το χειριζόταν η λογική του GCODE.lsp.

Το βελτιστοποιημένο σχέδιο αποτελείται από ένα κύριο κλειστό περίγραμμα και έναν ανεξάρτητο κύκλο. Η πρώτη αλυσίδα αρχίζει στο σημείο (250,125), ακολουθεί διαδοχικά γραμμές και τόξα, και επιστρέφει ξανά στο ίδιο σημείο. Ο κύκλος με αρχή και τέλος το (275,200) δεν έχει κοινό άκρο με το κύριο περίγραμμα, επομένως σχηματίζει δεύτερη ανεξάρτητη αλυσίδα και απαιτεί ξεχωριστή εντολή G0 για να ξεκινήσει.



Σχήμα 3.6: Πραγματικό παράδειγμα ευθυγράμμισης οντοτήτων της εργασίας Two Axis CNC - Hardware and Software Development [1]. Πάνω φαίνεται η αρχική σειρά των οντοτήτων πριν από την ευθυγράμμιση και κάτω η βελτιστοποιημένη σειρά εκτέλεσης μετά την ευθυγράμμιση.

Στο στάδιο 2 του αλγορίθμου, κάθε γεωμετρική οντότητα περιγράφεται με τα στοιχεία που χρειάζονται για τη σύνδεση και την εκπομπή της. Ο πίνακας 3.1 ακολουθεί σκόπιμα την αρχική σειρά σχεδίασης του Image 1.2, δηλαδή E(0) -> E(1) -> ... -> E(18), ώστε να φαίνεται καθαρά ποια είναι η είσοδος που καλείται να αναδιατάξει ο αλγόριθμος.

Πίνακας 3.1: Οντότητες του παραδείγματος στη σειρά που σχεδιάστηκαν στο Image 1.2

Entity	Τύπος	Αρχή	Τέλος	I,J
E(0)	LINE	(75,50)	(75,100)	-
E(1)	LINE	(125,125)	(125,175)	-
E(2)	LINE	(175,175)	(175,125)	-
E(3)	LINE	(200,100)	(250,100)	-
E(4)	LINE	(275,125)	(325,175)	-
E(5)	LINE	(350,150)	(350,225)	-
E(6)	LINE	(300,250)	(250,250)	-
E(7)	LINE	(75,100)	(125,125)	-
E(8)	ARC	(175,175)	(125,175)	I=-25, J=0
E(9)	ARC	(175,125)	(200,100)	I=25, J=0
E(10)	LINE	(250,100)	(250,125)	-
E(11)	LINE	(250,125)	(275,125)	-
E(12)	LINE	(350,150)	(325,175)	-
E(13)	ARC	(300,250)	(350,225)	I=50, J=37.5
E(14)	ARC	(250,250)	(200,225)	I=0, J=-62.5
E(15)	LINE	(25,50)	(25,225)	-
E(16)	LINE	(25,225)	(200,225)	-
E(17)	LINE	(25,50)	(75,50)	-
E(18)	CIRCLE/ARC	(275,200)	(275,200)	I=-25, J=0

Στο στάδιο 3, τα άκρα των οντοτήτων σχηματίζουν τον χάρτη σύνδεσης άκρων. Ο χάρτης δεν αποθηκεύει ολοκληρωμένες διαδρομές, αλλά ποια entities ξεκινούν ή τελειώνουν στο ίδιο σημείο. Έτσι, όταν η αλυσίδα φτάσει σε ένα άκρο, ο αλγόριθμος μπορεί να βρει άμεσα ποια μη επισκέψιμη οντότητα συνεχίζει από εκεί.

Πίνακας 3.2: Χάρτης σύνδεσης άκρων για τις οντότητες του πραγματικού παραδείγματος.

Κοινή	Entities στο ίδιο άκρο
(25,50)	E(15) αρχή, E(17) αρχή
(75,50)	E(0) αρχή, E(17) τέλος
(75,100)	E(0) τέλος, E(7) αρχή
(125,125)	E(1) αρχή, E(7) τέλος
(125,175)	E(1) τέλος, E(8) τέλος

(175,175)	E(2) αρχή, E(8) αρχή
(175,125)	E(2) τέλος, E(9) αρχή
(200,100)	E(3) αρχή, E(9) τέλος
(250,100)	E(3) τέλος, E(10) αρχή
(250,125)	E(10) τέλος, E(11) αρχή
(275,125)	E(4) αρχή, E(11) τέλος
(325,175)	E(4) τέλος, E(12) τέλος
(350,150)	E(5) αρχή, E(12) αρχή
(350,225)	E(5) τέλος, E(13) τέλος
(300,250)	E(6) αρχή, E(13) αρχή
(250,250)	E(6) τέλος, E(14) αρχή
(200,225)	E(14) τέλος, E(16) τέλος
(25,225)	E(15) τέλος, E(16) αρχή
(275,200)	E(18) αρχή, E(18) τέλος

Από τον πίνακα 3.2 φαίνεται ότι τα περισσότερα άκρα του κύριου περιγράμματος εμφανίζονται δύο φορές: μία ως τέλος μιας οντότητας και μία ως αρχή μιας άλλης. Αυτό επιτρέπει στον αλγόριθμο να σχηματίσει μια συνεχή αλυσίδα για το εξωτερικό περίγραμμα, ενώ το E(18), που έχει αρχή και τέλος στο ίδιο σημείο και δεν συνδέεται με άλλη οντότητα, αναγνωρίζεται ως δεύτερη ανεξάρτητη κλειστή διαδρομή.

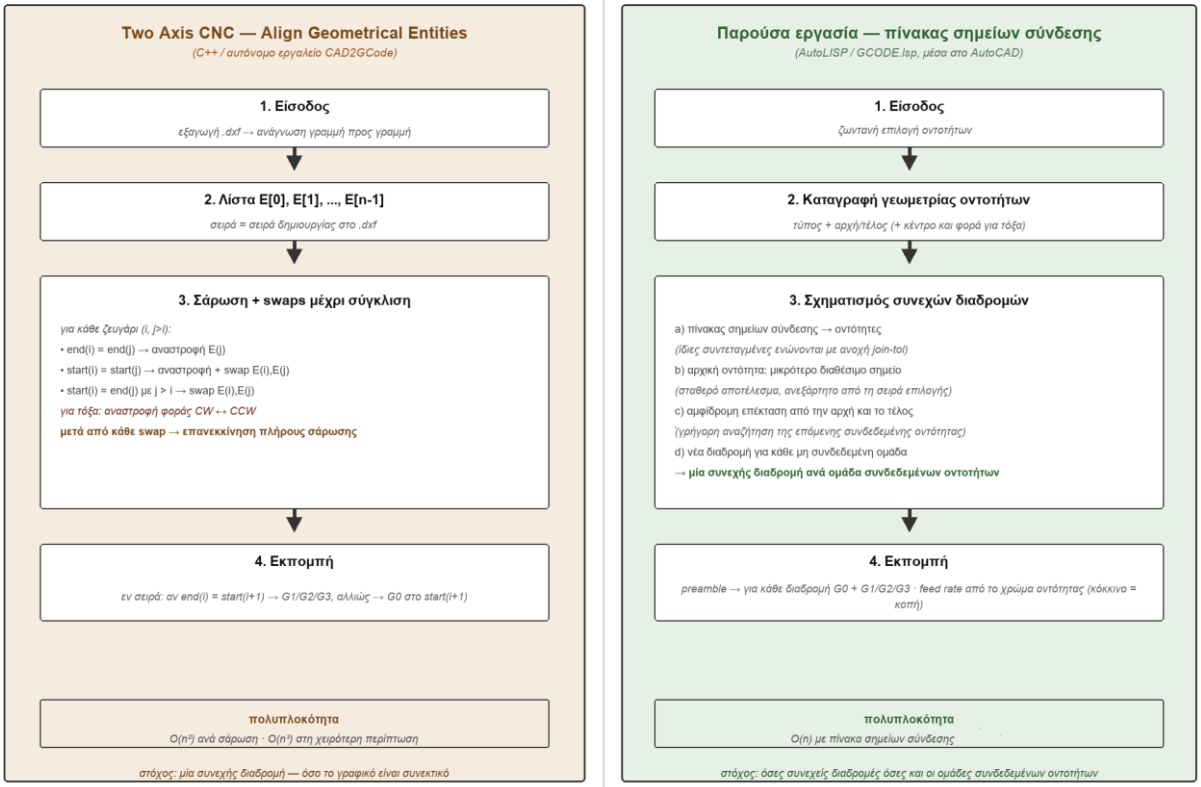
Στην εκπομπή των εντολών εφαρμόζεται μια επιπλέον σχεδιαστική επιλογή. Ο ρυθμός πρόωσης που γράφεται σε κάθε εντολή G1/G2/G3 δεν είναι σταθερός για ολόκληρο το πρόγραμμα, αλλά εξαρτάται από το χρώμα της αρχικής οντότητας από την οποία προήλθε το τμήμα: οι κόκκινες οντότητες παίρνουν την τιμή Cut Feed Rate, όλες οι υπόλοιπες την τιμή Feed Rate. Η σύμβαση είναι πρακτική: σε ένα τυπικό σχέδιο για λέιζερ, τα κόκκινα περιγράμματα αντιπροσωπεύουν διαδρομές κοπής όπου απαιτείται αργή ταχύτητα ώστε η δέσμη να διαπεράσει το υλικό, ενώ τα υπόλοιπα χρώματα χαράξεις χωρίς πλήρη διάτρηση όπου επιδιώκεται γρήγορη ταχύτητα. Έτσι ο χρήστης μεταφέρει τη διάκριση κοπής/χάραξης απευθείας από την οπτική κωδικοποίηση του σχεδίου στη μηχανή, χωρίς να γράφει χωριστά αρχεία ή να χωρίζει την επιλογή σε δύο διαδοχικά περάσματα.

Η ίδια ανάγκη (η μετατροπή ενός 2D σχεδίου σε βέλτιστο κώδικα G) είχε αντιμετωπιστεί διαφορετικά στην προγενέστερη εργασία [1] (σχήμα 3.7). Εκεί, ο αλγόριθμος Align Geometrical Entities δούλευε ως αυτόνομο εργαλείο σε C++, που διάβαζε ένα αρχείο .dxf, και η οντότητα αναπαριστάνονταν ως αντικείμενο μιας κλάσης Entity με μεταβλητές για τις συντεταγμένες των άκρων και τις παραμέτρους τόξου. Η αρχική σειρά των οντοτήτων στη λίστα ήταν αυτή με την οποία είχαν εγγραφεί στο .dxf, δηλαδή η σειρά σχεδίασης από τον χρήστη.

Η προσέγγιση της προηγούμενης εργασίας ήταν να σαρώνει **κάθε ζευγάρι οντοτήτων (i, j) με j > i** και να αναζητεί τρεις περιπτώσεις ταύτισης άκρων: όταν το τέλος της E(i) συμπίπτει με το τέλος της E(j), όταν η αρχή της E(i) συμπίπτει με την αρχή της E(j), και όταν η αρχή της E(i) συμπίπτει με το τέλος της E(j). Σε κάθε περίπτωση εκτελούσε ανάλογη πράξη (αναστροφή των άκρων ή μετάθεση των οντοτήτων στη λίστα. Όταν η οντότητα ήταν τόξο, εκτελούσε επιπλέον αναστροφή φοράς (CW ↔ CCW) και υπολογισμό νέων offsets I,J). Αυτό διότι το πρότυπο .dxf αποθηκεύει όλα τα τόξα ως CCW, ανεξάρτητα από το πώς δημιουργήθηκαν, χάνοντας την αρχική φορά τους κατά την εξαγωγή. Κάθε

μετάθεση όμως υποχρέωνε τον αλγόριθμο να επανεκκινήσει **πλήρη σάρωση** όλης της λίστας από την αρχή, ώστε να ελέγξει αν η νέα διάταξη επιτρέπει επιπλέον ταυτοποιήσεις οδηγώντας σε αρκετά μεγάλη πολυπλοκότητα.

Σχήμα 3.6 — Σύγκριση αλγορίθμων ταύτισης οντοτήτων: προηγούμενη εργασία (αριστερά) - παρούσα εργασία (δεξιά)



Σχήμα 3.7: Σύγκριση των δύο προσεγγίσεων. Αριστερά η σειριακή σάρωση με swaps της προηγούμενης εργασίας — τετραγωνική πολυπλοκότητα. Δεξιά η προσέγγιση της παρούσας εργασίας με χάρτη σύνδεσης άκρων σε γραμμικό χρόνο.

Η παρούσα εργασία διαφοροποιείται σε τρία ουσιώδη σημεία. Πρώτον, η αναζήτηση τμημάτων με κοινά άκρα δεν γίνεται με σάρωση όλης της λίστας αλλά μέσω χάρτη σύνδεσης άκρων, μειώνοντας τη συνολική πολυπλοκότητα από κυβική σε γραμμική και επιτρέποντας επεξεργασία σχεδίων με χιλιάδες οντοτήτων χωρίς αισθητή καθυστέρηση. Δεύτερον, η ταύτιση γίνεται με ανοχή στρογγυλοποίησης και όχι με ακριβή ισότητα. Τρίτον, η επιλογή του πρώτου τμήματος-σπόρου γίνεται από το μικρότερο άκρο της λίστας. Στην προηγούμενη εργασία, η αρχή ήταν πάντα η οντότητα E(0), δηλαδή αυτή που σχεδιάστηκε πρώτη, με αποτέλεσμα δύο σχέδια ταυτόσημα γεωμετρικά αλλά διαφορετικής σειράς δημιουργίας να παράγουν διαφορετικό κώδικα.

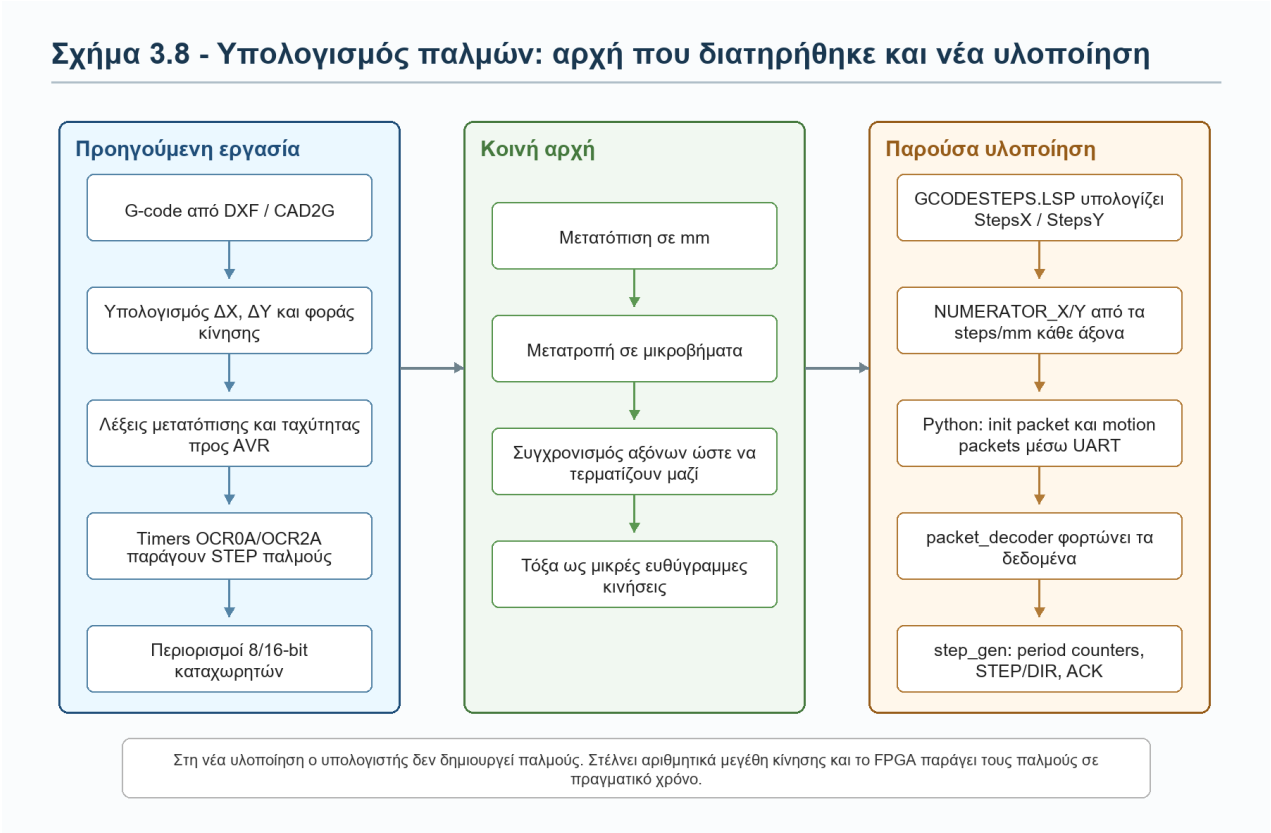
Το αρχείο .nc που προκύπτει αποτελεί την είσοδο για την επόμενη φάση (μετατροπή σε βήματα κινητήρα) η οποία αναλύεται στην ενότητα 3.4.

3.4 Υπολογισμός παλμών ανά άξονα

Στο στάδιο του υπολογισμού παλμών, το G-code μετατρέπεται από σημεία συντεταγμένων σε δεδομένα κίνησης που μπορεί να εκτελέσει το FPGA. Κάθε εντολή G0, G1, G2 ή G3 αναλύεται σε μετατόπιση στους άξονες, σε πρόσημο κατεύθυνσης, σε πλήθος παλμών και σε τιμή πρόωσης (ταχύτητα). Με αυτόν

τον τρόπο η πληροφορία του σχεδίου συνδέεται με τα πραγματικά σήματα STEP και DIR που οδηγούν τους βηματικούς κινητήρες.

Η περιγραφή βασίζεται στη λογική της προηγούμενης εργασίας «Two Axis CNC - Hardware and Software Development», όπου η κίνηση εκφράζεται ως μετατόπιση και ταχύτητα ανά άξονα. Στη νέα υλοποίηση η ίδια αρχή προσαρμόζεται στην αρχιτεκτονική AutoLISP - Python - FPGA: το GCODESTEPS.LSP υπολογίζει τα βήματα, διατηρεί την πρόωση F ως συνισταμένη ταχύτητα της κεφαλής και υπολογίζει την ταχύτητα ανά άξονα πριν από την αποστολή. Το FPGA λαμβάνει έτοιμα αριθμητικά μεγέθη και αναλαμβάνει μόνο τον ακριβή χρονισμό των παλμών [2][3].



Σχήμα 3.8: Προσαρμογή της λογικής υπολογισμού παλμών στη υλοποίηση με FPGA.

3.4.1 Στοιχεία που κρατήθηκαν από την προηγούμενη εργασία

Στην προηγούμενη εργασία γίνεται διάκριση ανάμεσα σε αλγορίθμους που παράγουν απευθείας παλμούς και σε αλγορίθμους που παράγουν αριθμητικές λέξεις μετατόπισης και ταχύτητας. Η νέα υλοποίηση ακολουθεί τη δεύτερη λογική. Ο υπολογιστής δεν στέλνει μία ακολουθία παλμών, αλλά στέλνει στον ελεγκτή το πλήθος παλμών που πρέπει να εκτελεστεί σε κάθε άξονα και την πρόωση της κίνησης. Στη συνέχεια το FPGA μετατρέπει αυτά τα αριθμητικά μεγέθη σε πραγματικούς παλμούς προς τους drivers.

Πίνακας 3.3: Σύγκριση μεταξύ των δύο υλοποιήσεων.

Στοιχείο	Προηγούμενη εργασία	Νέα υλοποίηση
Μετατόπιση ανά άξονα	Υπολογισμός dx, dy από τις συντεταγμένες της εντολής.	Υπολογισμός DX, DY στο GCODESTEPS.LSP

Κατεύθυνση	Το πρόσημο της μετατόπισης καθορίζει τα bits DIR στο PORTB.	Το πρόσημο μεταφέρεται μέσα στα signed Steps και το step_gen οδηγεί τα DIR pins.
Μετατροπή σε microsteps	Η απόσταση σε mm μετατρέπεται σε αριθμό microsteps.	Ίδια αρχή, αλλά οι παράμετροι του άξονα λαμβάνονται από το μοντέλο μηχανής.
Χρονισμός κίνησης	Οι τιμές OCR0A/OCR2A καθορίζουν τη συχνότητα παλμών στον AVR.	Το FPGA χρησιμοποιεί 32-bit counters. Το FeedRate του packet είναι η συνιστώσα του κύριου άξονα, ώστε η συνισταμένη ταχύτητα να ισούται με το F.
Τόξα	Το τόξο προσεγγίζεται από μικρές ευθύγραμμες κινήσεις.	Εφαρμόζεται η ίδια λογική
Επιβεβαίωση	Μετά από κάθε αποστολή υπάρχει μήνυμα επιβεβαίωσης.	Το FPGA στέλνει ACK 0xAA μετά την ολοκλήρωση κάθε κίνησης.

3.4.2 Μετατροπή από mm σε παλμούς

Η βασική μετατροπή είναι ίδια με την παλιότερη υλοποίηση: πρώτα υπολογίζεται πόσα μικροβήματα αντιστοιχούν σε μία περιστροφή του κινητήρα και στη συνέχεια πόση γραμμική μετατόπιση παράγει μία περιστροφή της τροχαλίας. Για άξονα με βηματικό κινητήρα 1,8°, οδηγό σε microsteping mode 1/32, ιμάντα GT2 και τροχαλία 20 δοντιών, η μετατόπιση μιας πλήρους περιστροφής είναι 40 mm και τα microsteps ανά περιστροφή είναι 6400. Άρα η ανάλυση είναι 160 παλμοί/mm.

steps_per_rev = (360 / step_angle) × microsteps

travel_per_rev = belt_pitch × pulley_teeth

steps_per_mm = (steps_per_rev × gear_ratio) / travel_per_rev

Η σχέση αυτή είναι κρίσιμη επειδή απομονώνει τη γεωμετρία της μηχανής από τον κώδικα ελέγχου. Αν αλλάξει το microstep mode, η τροχαλία ή ο τύπος μετάδοσης, αλλάζει μόνο ο συντελεστής steps_per_mm και όχι ο τρόπος με τον οποίο το FPGA παράγει παλμούς.

Πίνακας 3.4: Παράδειγμα μηχανικών παραμέτρων και ανάλυσης άξονα.

Μέγεθος	Μεταβλητή	Τιμή παραδείγματος	Αποτέλεσμα
Γωνία βήματος κινητήρα	step_angle	1,8°	200 πλήρη βήματα/rev
Mixrostep driver	microsteps	32	6400 μικροβήματα/rev
Βήμα ιμάντα	belt_pitch	2 mm/δόντι	—
Δόντια τροχαλίας	pulley_teeth	20	40 mm/rev
Λόγος μετάδοσης	gear_ratio	1	Απευθείας σύνδεση
Ανάλυση άξονα	steps_per_mm	6400 / 40	160 παλμοί/mm
Μικρότερη μετατόπιση	1 / steps_per_mm	1 / 160	0,00625 mm

3.4.3 Μετατροπή εντολών G-code σε παλμούς

Η εντολή GCODE2STEPS του αρχείου GCODESTEPS.LSP διαβάζει το αρχείο G-code γραμμή προς γραμμή. Για κάθε κίνηση εντοπίζει τον τύπο της εντολής, την τελική θέση και την ενεργή πρόωση. Έπειτα υπολογίζει τη μετατόπιση από την τρέχουσα θέση και τη μετατρέπει σε ακέραιους παλμούς. Η στρογγυλοποίηση είναι απαραίτητη, επειδή ο οδηγός βηματικού κινητήρα δέχεται μόνο ακέραιο πλήθος παλμών.

```
StepsX = round(DX × steps_per_mm,X)
StepsY = round(DY × steps_per_mm,Y)
PulsesEst = max(|StepsX|, |StepsY|)
```

Το πρόσημο των StepsX και StepsY δεν χάνεται. Χρησιμοποιείται από το FPGA για την επιλογή φοράς περιστροφής στον αντίστοιχο άξονα, ενώ το απόλυτο πλήθος παλμών χρησιμοποιείται από τους εσωτερικούς μετρητές rem_x και rem_y. Έτσι το λογισμικό δεν χρειάζεται να στέλνει ξεχωριστά bits κατεύθυνσης.

Πίνακας 3.5: Ροή υπολογισμού από εντολή G-code σε δεδομένα κίνησης.

Βήμα	Είσοδος	Ενέργεια	Έξοδος
1	Γραμμή G-code	Ανάγνωση G0/G1/G2/G3, X, Y, I, J, F.	Τύπος κίνησης και στόχος
2	Τρέχουσα και τελική θέση	Υπολογισμός DX και DY	Μετατόπιση ανά άξονα
3	DX, DY και steps/mm	Πολλαπλασιασμός και στρογγυλοποίηση.	StepsX, StepsY
4	FeedRate / PathFeed	Υπολογισμός ενεργής πρόωσης και συνιστώσας κύριου άξονα.	FeedRate packet και PathFeed
5	Steps και FeedRate	Εγγραφή στο ενδιάμεσο αρχείο και αποστολή με Python.	Δυαδικό motion packet

3.4.4 Γραμμικές κινήσεις και τόξα

Για τις γραμμικές κινήσεις G0 και G1 η διαδικασία είναι άμεση, επειδή η αρχική και η τελική θέση ορίζουν μία ευθεία μετατόπιση. Σε αντίθεση με την απλοποίηση της προηγούμενης υλοποίησης για την G0, όπου μπορούσαν να κινηθούν και οι δύο άξονες με μέγιστη ταχύτητα ανεξάρτητα, εδώ το step_gen υπολογίζει περιόδους ώστε οι άξονες να ολοκληρώνουν την κίνηση στον ίδιο χρόνο. Αυτό διατηρεί τη γεωμετρική συνέπεια της τροχιάς ακόμη και στις γρήγορες μετακινήσεις.

Για τις εντολές G2 και G3 διατηρείται η αρχή της προηγούμενης εργασίας: το τόξο δεν εκτελείται ως αναλογική καμπύλη, αλλά προσεγγίζεται με μικρές ευθύγραμμες κινήσεις. Το AutoLISP υπολογίζει το κέντρο από τα I και J, βρίσκει τη γωνιακή σάρωση και χωρίζει το τόξο σε βήματα με μέγιστο μήκος maxSegLen. Κάθε ενδιάμεση μετατόπιση περνά από την ίδια διαδικασία μετατροπής σε παλμούς.

```
centerX = curX + I, centerY = curY + J
arc_length = radius × sweep
N = ceil(arc_length / maxSegLen)
```

Η διαίρεση του τόξου γίνεται στον υπολογιστή επειδή εκεί υπάρχουν ευκολότερα οι τριγωνομετρικοί υπολογισμοί και η αριθμητική κινητής υποδιαστολής. Το FPGA λαμβάνει μόνο τις τελικές μικρές μετατοπίσεις σε παλμούς και τις εκτελεί όλες με τον ίδιο μηχανισμό.

3.4.5 NUMERATOR και παραγωγή παλμών στο FPGA

Στην προηγούμενη εργασία η ταχύτητα αντιστοιχούσε σε τιμές των καταχωρητών OCR0A και OCR2A του μικροελεγκτή. Στη νέα υλοποίηση ο ίδιος ρόλος μεταφέρεται σε μετρητές περιόδου μέσα στο FPGA. Το σύστημα λειτουργεί με ρολόι 50 MHz, επομένως η ζητούμενη πρόωση μετατρέπεται σε αριθμό κύκλων ρολογιού ανά παλμό STEP.

steps_per_second = (FeedRate × steps_per_mm) / 60

cycles_per_step = CLK_HZ / steps_per_second

NUMERATOR = CLK_HZ × 60 / steps_per_mm

period = NUMERATOR / FeedRate

Ο συντελεστής NUMERATOR υπολογίζεται μία φορά για κάθε άξονα και αποστέλλεται στο FPGA πριν από τις κινήσεις. Με 160 παλμούς/mm και ρολόι 50 MHz, προκύπτει NUMERATOR = 18 750 000. Η περίοδος που χρησιμοποιεί το FPGA δεν υπολογίζεται από το αρχικό F της διαδρομής, αλλά από τη συνιστώσα ταχύτητας του κύριου άξονα.

Όταν κινούνται και οι δύο άξονες, το F του G-code εφαρμόζεται στη συνισταμένη ταχύτητα της κεφαλής. Το GCODESTEPS.LSP υπολογίζει το μήκος της μετατόπισης και μετατρέπει το F στη συνιστώσα του άξονα με τους περισσότερους παλμούς. Αυτή η τιμή αποστέλλεται ως FeedRate στο motion packet. Το step_gen συνεχίζει να χρησιμοποιεί τον κύριο άξονα για τον βασικό χρονισμό και δίνει μεγαλύτερη περίοδο στον δεύτερο άξονα, ώστε οι δύο άξονες να τερματίζουν μαζί.

path_length = sqrt(DX^2 + DY^2)

axisFeed = F × main_axis_distance / path_length

period_main = NUMERATOR_main / axisFeed, period_other = period_main × steps_main / steps_other

period_other = period_main × steps_main / steps_other

3.4.6 Δυαδικά δεδομένα και όρια της νέας υλοποίησης

Η επικοινωνία παραμένει σειριακή, όπως και στην προηγούμενη εργασία, αλλά η μορφή των δεδομένων έχει αλλάξει. Αντί για bytes να στέλνονται απευθείας θύρες του μικροελεγκτή, η Python στέλνει δομημένα δυαδικά πακέτα. Το αρχικό πακέτο ρυθμίζει τους συντελεστές NUMERATOR_X και NUMERATOR_Y. Κάθε επόμενο πακέτο περιγράφει μία κίνηση με τύπο, παλμούς X, παλμούς Y και πρόωση.

Πίνακας 3.6: Δομή των δυαδικών πακέτων UART.

Πακέτο	Μέγεθος	Δομή	Σκοπός
Init packet	9 bytes	0xFF | NUMERATOR_X[4] | NUMERATOR_Y[4]	Φόρτωση παραμέτρων χρονισμού στο FPGA.
Motion packet	11 bytes	motion[1] | StepsX[4] | StepsY[4] | FeedRate[2]	Περιγραφή μιας κίνησης. Το FeedRate είναι η συνιστώσα του κύριου άξονα, ενώ το

			PathFeed μένει στο CSV ως τεκμηρίωση.
ACK	1 byte	0xAA	Επιστρέφεται από το FPGA όταν ολοκληρωθεί η κίνηση.

Η μετάβαση από 8-bit/16-bit καταχωρητές μικροελεγκτή σε 32-bit καταχωρητές FPGA αυξάνει σημαντικά το εύρος των υπολογισμών. Για παράδειγμα, με 160 παλμούς/mm, ένας signed 32-bit αριθμός παλμών αντιστοιχεί θεωρητικά σε πολύ μεγαλύτερη απόσταση από τις πραγματικές διαστάσεις της μηχανής. Έτσι ο πρακτικός περιορισμός είναι πλέον η μηχανική διαδρομή και όχι το μέγεθος του καταχωρητή μετατόπισης.

3.4.7 Αριθμητικό παράδειγμα

Έστω ότι η μηχανή έχει ανάλυση 160 παλμούς/mm και η αρχική θέση είναι (0,0). Για την εντολή G1 X10.000 Y5.000 F1000, η μετατόπιση είναι DX = 10 mm και DY = 5 mm. Το F=1000 mm/min είναι η ταχύτητα πάνω στην πραγματική ευθεία διαδρομή. Επειδή ο X είναι ο κύριος άξονας, στο FPGA αποστέλλεται η συνιστώσα του X, δηλαδή 894.43 mm/min.

StepsX = round(10 × 160) = 1600

StepsY = round(5 × 160) = 800

PulsesEst = max(1600, 800) = 1600

axisFeedX = 1000 × 10 / sqrt(10^2 + 5^2) = 894.43 mm/min

periodX = 18 750 000 / 894.43 = 20 963 cycles, periodY = 20 963 × 1600 / 800 = 41 926 cycles

Πίνακας 3.7: Παράδειγμα μετατροπής G-code γραμμής σε παλμούς και πακέτο κίνησης

G-code	DX	DY	StepsX	StepsY	FeedRate packet	Περιγραφή πακέτου
G1 X10 Y5 F1000	10 mm	5 mm	1600	800	894	motion=1, X=1600, Y=800, F=894, PathFeed=1000
G1 X10 Y0 F1000	0 mm	-5 mm	0	-800	1000	motion=1, X=0, Y=-800, F=1000, PathFeed=1000

Στην πρώτη κίνηση ο X είναι ο κύριος άξονας, όμως στο packet δεν αποστέλλεται το αρχικό F=1000 αλλά το διορθωμένο FeedRate=894. Με αυτή την τιμή ο X κινείται με 894.43 mm/min και ο Y με 447.21 mm/min, οπότε η συνισταμένη ταχύτητα της κεφαλής είναι 1000 mm/min. Στη δεύτερη κίνηση υπάρχει μόνο κίνηση στον Y, άρα το FeedRate του packet παραμένει 1000.

Συνολικά, η νέα υλοποίηση κρατά τη βασική ιδέα της προηγούμενης εργασίας, δηλαδή τη μετατροπή γεωμετρίας σε αριθμητικές λέξεις κίνησης και την παραγωγή παλμών από counters, αλλά τη μεταφέρει σε πιο κατάλληλη αρχιτεκτονική. Οι γεωμετρικοί και τριγωνομετρικοί υπολογισμοί γίνονται στο AutoLISP/Python, ενώ το FPGA αναλαμβάνει την ακριβή χρονική παραγωγή των παλμών STEP και τον συγχρονισμό των αξόνων.

3.5 Γεωμετρική ταύτιση σχεδίου με δοκίμιο και κεντράρισμα κεφαλής

Μετά την παραγωγή του G-code και πριν από την εκτέλεση της κίνησης, πρέπει να οριστεί σε ποιο φυσικό σημείο του δοκιμίου θα ξεκινήσει η κατεργασία. Στην παρούσα υλοποίηση η ταύτιση σχεδίου και δοκιμίου γίνεται με τον ορισμό ενός κοινού σημείο αναφοράς. Ο χρήστης μετακινεί την κεφαλή πάνω από το επιλεγμένο σημείο του υλικού και το δηλώνει ως σημείο αναφοράς.

Θεωρείται ότι το δοκίμιο έχει τοποθετηθεί σε σωστή σχετική γωνία με τους άξονες X και Y της μηχανής. Το SetOriginButton χρησιμοποιείται για να συνδέσει τη γραφική αρχή του σχεδίου με την τρέχουσα φυσική θέση της κεφαλής, δηλαδή για να εφαρμοστεί μεταφορά αρχής ή offset.

3.5.1 Συστήματα συντεταγμένων

Στην πράξη συνυπάρχουν δύο βασικά συστήματα συντεταγμένων. Το πρώτο είναι το σύστημα συντεταγμένων του σχεδίου CAD, με βάση το οποίο δημιουργήθηκαν οι γεωμετρικές οντότητες. Το δεύτερο είναι το σύστημα συντεταγμένων της μηχανής, δηλαδή το φυσικό επίπεδο στο οποίο κινείται η κεφαλή.

Ο ορισμός αρχής δεν αλλάζει τη μορφή του σχεδίου. Αλλάζει μόνο το φυσικό σημείο από το οποίο θα αρχίσει η εκτέλεση. Έτσι η ίδια διαδρομή μπορεί να μεταφερθεί σε άλλη θέση πάνω στο τραπέζι εργασίας, χωρίς να επανασχεδιαστεί η γεωμετρία.

Πίνακας 3.8: Διαχωρισμός των βασικών συστημάτων συντεταγμένων.

Σύστημα	Περιγραφή	Παράδειγμα χρήσης
Σύστημα σχεδίου	Οι αρχικές συντεταγμένες των οντοτήτων στο AutoCAD.	Κέντρο κύκλου C=(40,25) mm.
Σύστημα δοκιμίου	Οι πραγματικές θέσεις των αντίστοιχων σημείων πάνω στο υλικό.	Γωνία δοκιμίου A'=(120,75) mm.
Σύστημα μηχανής	Οι συντεταγμένες που αντιστοιχούν στην κίνηση της κεφαλής.	Θέση κεφαλής μετά από jog ή μηδενισμό.
Σύστημα G-code	Οι τελικές συντεταγμένες που θα μετατραπούν σε παλμούς.	G1 X145.200 Y90.300 F1000.

3.5.2 Σημείο αναφοράς και SetOriginButton

Η διαδικασία βασίζεται σε ένα κοινό σημείο αναφοράς. Ο χρήστης επιλέγει ποιο σημείο του σχεδίου θα θεωρηθεί αρχή της κατεργασίας, για παράδειγμα την κάτω αριστερή γωνία ή το κέντρο μιας οπής. Στη συνέχεια μετακινεί την κεφαλή με χειροκίνητα πάνω από το αντίστοιχο φυσικό σημείο του δοκιμίου.

Με το SetOriginButton η τρέχουσα θέση της κεφαλής δηλώνεται ως σημείο εκκίνησης. Από εκεί και μετά οι συντεταγμένες του G-code εκτελούνται ως σχετικές μετατοπίσεις ως προς αυτή τη νέα αρχή. Αν το δοκίμιο είναι περιστραμμένο, η παρούσα υλοποίηση δεν υπολογίζει αυτόματη γωνιακή διόρθωση. Το δοκίμιο πρέπει να ευθυγραμμιστεί μηχανικά πριν από τον ορισμό της αρχής.

Πίνακας 3.9: Λειτουργικός ρόλος του σημείου αναφοράς στην παρούσα υλοποίηση.

Στοιχείο	Ρόλος	Σχόλιο
Σημείο A στο σχέδιο	Γραφική αρχή ή σημείο εκκίνησης της διαδρομής.	Επιλέγεται από τον χρήστη στο σχέδιο.
Φυσικό σημείο A' στο δοκίμιο	Πραγματική θέση πάνω στο υλικό.	Η κεφαλή μετακινείται χειροκίνητα στο σημείο.
SetOriginButton	Ορίζει την τρέχουσα θέση ως αρχή εκτέλεσης.	Δεν υπολογίζει περιστροφή.

3.5.3 Υπολογισμός μεταφοράς αρχής

Έστω A το σημείο αναφοράς στο σύστημα του σχεδίου και A' το αντίστοιχο φυσικό σημείο πάνω στο δοκίμιο, στο οποίο έχει μετακινηθεί η κεφαλή. Για οποιοδήποτε σημείο p του σχεδίου, η αντίστοιχη θέση εκτέλεσης p' προκύπτει με απλή μεταφορά.

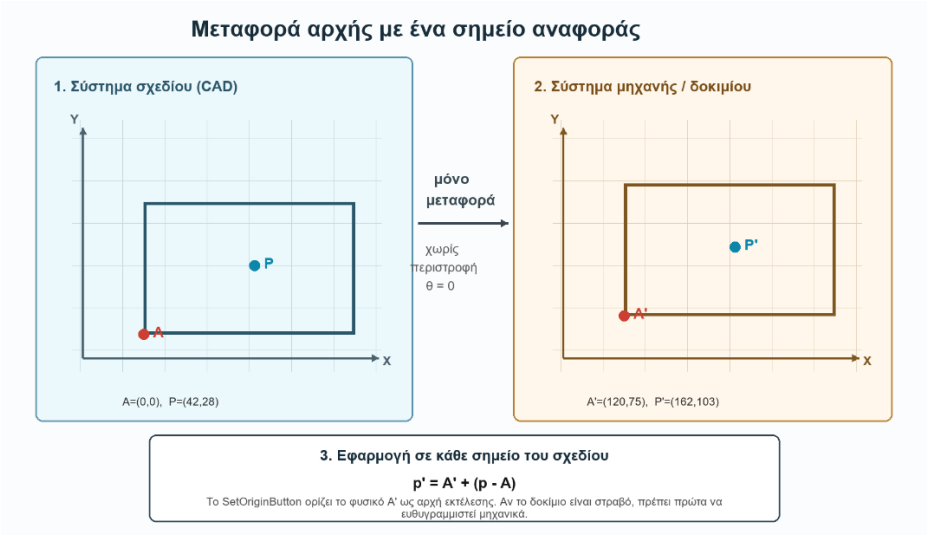
offset = A' - A

p' = p + offset

ισοδύναμα: p' = A' + (p - A)

θ = 0, s = 1 στην παρούσα υλοποίηση

Η σχέση αυτή εφαρμόζεται ως μετατόπιση σε όλες τις κινήσεις. Για ευθύγραμμα τμήματα μεταφέρονται η αρχή και το τέλος κατά το ίδιο offset. Για τόξα μεταφέρονται επίσης το κέντρο και τα άκρα, χωρίς αλλαγή ακτίνας ή φοράς. Δεν εφαρμόζεται περιστροφή ή αλλαγή κλίμακας.



Σχήμα 3.9: Μεταφορά σημείου από το σύστημα του σχεδίου στο σύστημα της μηχανής.

3.5.4 Κεντράρισμα κεφαλής

Το κεντράρισμα της κεφαλής είναι η φυσική διαδικασία με την οποία επιβεβαιώνεται ότι το θεωρητικό σημείο αναφοράς αντιστοιχεί στην πραγματική θέση της δέσμης. Η κεφαλή μετακινείται με μικρά

βήματα πάνω από το δοκίμιο μέχρι η δέσμη χαμηλής ισχύος ή ο οπτικός δείκτης να συμπέσει με το επιλεγμένο σημείο. Στη συνέχεια ο χρήστης πατά το SetOriginButton.

Η διαδικασία πρέπει να γίνεται με την κεφαλή σε ασφαλές ύψος και με την ισχύ του laser σε επίπεδο ελέγχου, ώστε να μην χαραχτεί το δοκίμιο κατά την ευθυγράμμιση. Μετά τον ορισμό της αρχής μπορεί να γίνει δοκιμαστική διαδρομή περιγράμματος χωρίς κοπή, ώστε να επιβεβαιωθεί ότι η διαδρομή βρίσκεται εντός της επιθυμητής περιοχής.

Πίνακας 3.10: Βήματα κεντραρίσματος κεφαλής πριν από την εκτέλεση.

Βήμα	Ενέργεια	Σκοπός
1	Στερέωση και μηχανική ευθυγράμμιση δοκιμίου.	Το δοκίμιο να είναι παράλληλο με τους άξονες X/Y.
2	Επιλογή σημείου αναφοράς A στο σχέδιο.	Ορισμός γραφικής αρχής ή σημείου εκκίνησης.
3	Χειροκίνητη μετακίνηση της κεφαλής στο φυσικό σημείο A'.	Ταύτιση της κεφαλής με το επιλεγμένο σημείο του δοκιμίου.
4	Πάτημα SetOriginButton.	Η τρέχουσα θέση δηλώνεται ως αρχή εκτέλεσης.
5	Έλεγχος περιγράμματος χωρίς κοπή.	Επιβεβαίωση ότι η διαδρομή βρίσκεται στη σωστή περιοχή.

3.5.5 Αριθμητικό παράδειγμα μεταφοράς αρχής

Ως απλό παράδειγμα θεωρείται ορθογώνιο σχέδιο στο οποίο το σημείο αναφοράς A βρίσκεται στο (0,0). Ο χρήστης μετακινεί την κεφαλή στο φυσικό σημείο του δοκιμίου όπου θέλει να ξεκινήσει η κατεργασία και πατά SetOriginButton. Αν η φυσική θέση αυτή είναι A'=(120,75), τότε όλο το σχέδιο μεταφέρεται κατά offset=(120,75).

A = (0,0), A' = (120,75)

offset = A' - A = (120,75)

P = (42,28) -> P' = P + offset = (162,103)

Για ένα εσωτερικό σημείο του σχεδίου P=(42,28), η φυσική θέση εκτέλεσης είναι P'=(162,103). Αν το G-code παραχθεί ως σχετική διαδρομή από το A ή αν η αρχή της μηχανής τεθεί στο A', η κεφαλή θα κινηθεί στην αντίστοιχη θέση του δοκιμίου. Η μέθοδος αυτή προϋποθέτει ότι το δοκίμιο είναι παράλληλο με τους άξονες της μηχανής.

Πίνακας 3.11: Παράδειγμα μεταφοράς σημείων σχεδίου σε συντεταγμένες δοκιμίου.

Σημείο	Συντεταγμένες σχεδίου	Συντεταγμένες δοκιμίου	Ρόλος
A	(0.00, 0.00)	(120.00, 75.00)	Σημείο αναφοράς / νέα αρχή
Offset	(+120.00, +75.00)	(+120.00, +75.00)	Μεταφορά που εφαρμόζεται
P	(42.00, 28.00)	(162.00, 103.00)	Σημείο διαδρομής μετά τη μεταφορά

Μετά τον ορισμό της αρχής, το G-code εκτελείται ως προς τη νέα θέση εκκίνησης και περνά στο στάδιο υπολογισμού παλμών που περιγράφηκε στην ενότητα 3.4. Το FPGA δεν χρειάζεται να γνωρίζει τίποτα για τη διαδικασία κεντραρίσματος. Εκτελεί απλώς την τελική ακολουθία παλμών που αντιστοιχεί στη σωστή θέση εκκίνησης πάνω στο δοκίμιο.

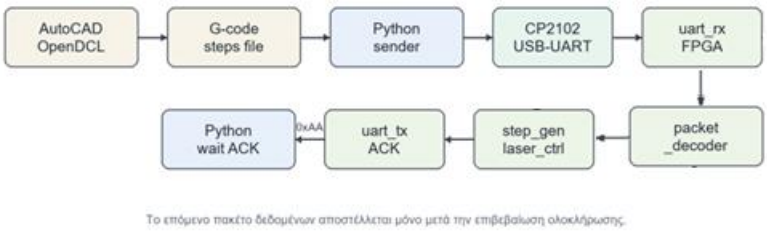
Κεφάλαιο 4ο: Επικοινωνία μεταξύ Host και ελεγκτή

4.1 Αρχιτεκτονική επικοινωνίας

Η επικοινωνία ανάμεσα στον υπολογιστή-host και τον ελεγκτή FPGA αποτελεί το ενδιάμεσο επίπεδο που συνδέει το λογισμικό CAM με το κύκλωμα παραγωγής παλμών. Το CAM παράγει τον G-code από το σχέδιο και υπολογίζει τους παλμούς ανά άξονα. Το FPGA λαμβάνει τα δεδομένα κίνησης, σε δυαδική μορφή, και αναλαμβάνει την ακριβή χρονική εκτέλεσή τους.

Η επιλογή αυτή μειώνει σημαντικά την πολυπλοκότητα του ελεγκτή. Αντί το FPGA να περιέχει parser για ASCII εντολές, μετατροπές αριθμών και υπολογισμούς κινητής υποδιαστολής, η επικοινωνία περιορίζεται σε πακέτα σταθερού μήκους [7]. Με τον τρόπο αυτό το κύκλωμα μπορεί να είναι μικρότερο, πιο προβλέψιμο και πιο εύκολο να ελεγχθεί, ενώ ο υπολογιστής αξιοποιείται για τις εργασίες που ταιριάζουν περισσότερο σε λογισμικό γενικού σκοπού.

Η φυσική σύνδεση υλοποιείται με μετατροπέα CP2102 USB-to-UART. Από την πλευρά του υπολογιστή η σύνδεση εμφανίζεται ως σειριακή θύρα COM, ενώ από την πλευρά της πλακέτας DE1-SoC τα σήματα οδηγούνται στις γραμμές GPIO_0[0] και GPIO_0[1]. Η επικοινωνία είναι ασύγχρονη UART με ρυθμό 115200 baud, 8 bits δεδομένων, χωρίς bit ισοτιμίας (parity) και με ένα stop bit (8N1). Η επιλογή 8N1 είναι απλή, υποστηρίζεται άμεσα από το CP2102 και αντιστοιχεί στα modules uart_rx.v και uart_tx.v που υλοποιούνται στο FPGA.



Σχήμα 4.1*: Αρχιτεκτονική επικοινωνίας μεταξύ host και FPGA.*

Στο Σχήμα 4.1 φαίνεται η βασική ακολουθία της επικοινωνίας. Το CAM λογισμικό παράγει το αρχείο κινήσεων. Το Python script αναλαμβάνει τη σειριακή μετάδοση, μετατρέποντας κάθε γραμμή του αρχείου σε δυαδικό πακέτο. Στο FPGA, το uart_rx ανασυνθέτει τα εισερχόμενα bytes και το packet_decoder τα ομαδοποιεί σε πεδία ελέγχου. Όταν ένα πακέτο κίνησης ολοκληρωθεί, το σήμα pkt_valid ενεργοποιεί τη γεννήτρια παλμών. Μετά την ολοκλήρωση της κίνησης, το FPGA επιστρέφει στον host το byte επιβεβαίωσης 0xAA μέσω του uart_tx.

Πίνακας 4.1*: Ρόλοι των βαθμίδων στην αρχιτεκτονική επικοινωνίας.*

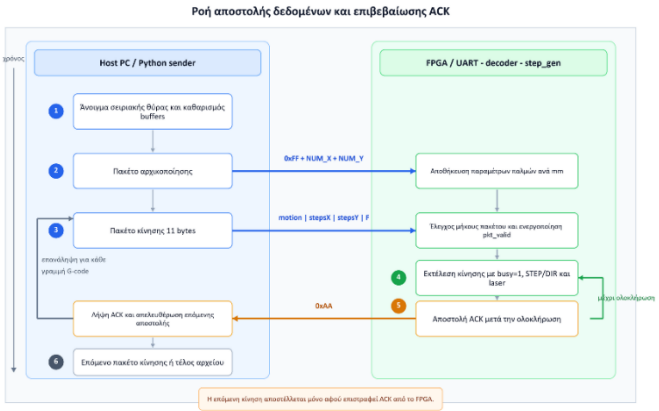
Βαθμίδα	**Ρόλος**
AutoLISP	Επιλογή σχεδίου, παραμέτρων μηχανής και εκκίνηση της διαδικασίας αποστολής.
Python sender	Ανάγνωση αρχείου παλμών, δημιουργία δυαδικών πακέτων και αναμονή επιβεβαίωσης.

CP2102	Μετατροπή USB σε TTL UART και φυσική σύνδεση host-FPGA.
UART receiver	Δειγματοληψία της σειριακής γραμμής και παραγωγή έγκυρων bytes.
Packet decoder	Συναρμολόγηση bytes σε πακέτο αρχικοποίησης ή πακέτο κίνησης.
Ελεγκτής κίνησης	Χρήση των πεδίων του πακέτου για παραγωγή σημάτων STEP/DIR και έλεγχο laser.
UART transmitter	Αποστολή επιβεβαίωσης στον host μετά την ολοκλήρωση κάθε κίνησης.

Η επικοινωνία μπορεί να θεωρηθεί ότι οργανώνεται σε τρία επίπεδα. Το πρώτο είναι το φυσικό επίπεδο, όπου καθορίζονται η καλωδίωση, οι στάθμες τάσης και οι βασικές παράμετροι UART. Το δεύτερο είναι το επίπεδο πλαισίωσης, όπου τα bytes οργανώνονται σε πακέτα σταθερού μήκους. Το τρίτο είναι το επίπεδο εφαρμογής, όπου τα πεδία του πακέτου αποκτούν σημασία για την κίνηση της μηχανής: τύπος κίνησης, αριθμός παλμών ανά άξονα και πρόωση.

Στην αρχή της διαδικασίας αποστέλλεται πακέτο αρχικοποίησης, με το αναγνωριστικό 0xFF, το οποίο μεταφέρει τις παραμέτρους NUMERATOR_X και NUMERATOR_Y. Οι παράμετροι αυτοί επιτρέπουν στην πλευρά του FPGA να υπολογίζει την περίοδο παλμών με βάση την πραγματική μηχανική ανάλυση της διάταξης. Στη συνέχεια αποστέλλονται πακέτα κίνησης των 11 bytes, ένα για κάθε γεωμετρικό τμήμα που έχει παραχθεί από το προηγούμενο στάδιο επεξεργασίας.

Η σειρά μετάδοσης είναι ελεγχόμενη. Ο host αποστέλλει ένα πακέτο κίνησης και στη συνέχεια αναμένει την απάντηση 0xAA από το FPGA. Η απάντηση παράγεται όταν το σήμα busy της γεννήτριας παλμών επιστρέψει σε λογικό μηδέν, δηλαδή όταν η αντίστοιχη κίνηση έχει ολοκληρωθεί. Μόνο τότε αποστέλλεται το επόμενο πακέτο. Με αυτόν τον μηχανισμό αποφεύγεται η ανάγκη για μεγάλο FIFO στο FPGA και εξασφαλίζεται ότι δεν θα χαθούν εντολές σε περίπτωση που μια κίνηση διαρκεί περισσότερο από την επόμενη μετάδοση του host.



Σχήμα 4.2*: Συνοπτική ροή αποστολής δεδομένων.*

Η χρήση σταθερού μήκους πακέτων απλοποιεί την αποκωδικοποίηση σε hardware. Ο packet_decoder δεν χρειάζεται να αναζητά χαρακτήρες τέλους γραμμής ούτε να μετατρέπει ASCII αριθμούς. Μετρά τα εισερχόμενα bytes, αποθηκεύει προσωρινά τα πεδία και, όταν συμπληρωθεί το αναμενόμενο μήκος, παράγει έναν παλμό εγκυρότητας. Για προστασία από ελλιπή ή λανθασμένη μετάδοση υπάρχει ένας watchdog timer: αν η λήψη ενός πακέτου διακοπεί για περίπου 100 ms, ο εσωτερικός μετρητής bytes μηδενίζεται και το κύκλωμα επιστρέφει σε κατάσταση αναμονής.

Η αρχιτεκτονική αυτή διατηρεί καθαρό διαχωρισμό ευθυνών. Ο host αποφασίζει τι πρέπει να εκτελεστεί και σε ποια σειρά, ενώ το FPGA αποφασίζει πότε ολοκληρώθηκε η αντίστοιχη κίνηση [8]. Η επικοινωνία δεν επιδιώκει τη μέγιστη θεωρητική ταχύτητα μετάδοσης, αλλά την ασφαλή εκτέλεση των εντολών κίνησης.

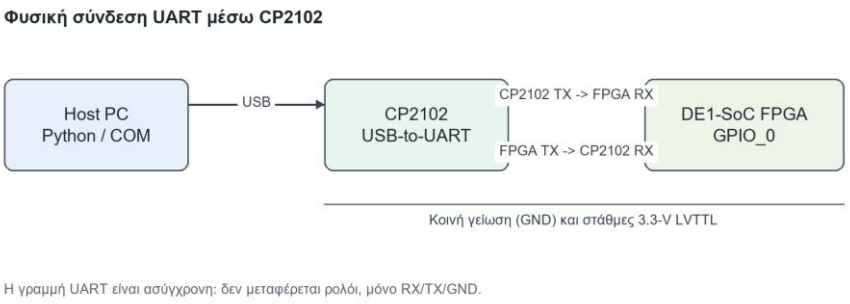
Συνολικά, η αρχιτεκτονική της επικοινωνίας μετατρέπει τη σύνθετη πληροφορία του CAM σε μικρά, σαφώς ορισμένα δυαδικά μηνύματα. Με αυτόν τον τρόπο το FPGA δεν λειτουργεί ως γενικός υπολογιστής που ερμηνεύει εντολές, αλλά ως εξειδικευμένος ελεγκτής πραγματικού χρόνου που εκτελεί κάθε ληφθέν πακέτο με σταθερό χρονισμό και επιστρέφει επιβεβαίωση μόνο όταν η μηχανή είναι έτοιμη για την επόμενη κίνηση.

4.2 Υλοποίηση UART — CP2102 bridge

Η επικοινωνία του ελεγκτή FPGA με τον υπολογιστή πραγματοποιείται μέσω σειριακής ασύγχρονης σύνδεσης UART. Επειδή ο υπολογιστής δεν διαθέτει απευθείας θύρα TTL UART, χρησιμοποιείται ο μετατροπέας CP2102 USB-to-UART. Ο CP2102 αναλαμβάνει τη μετατροπή της σύνδεσης USB σε δύο απλές σειριακές γραμμές, RX και TX, οι οποίες συνδέονται στις ακίδες GPIO της πλακέτας DE1-SoC.

Από την πλευρά του host, ο CP2102 εμφανίζεται στο λειτουργικό σύστημα ως σειριακή θύρα COM. Το Python script ανοίγει αυτή τη θύρα με τις ίδιες παραμέτρους που έχουν υλοποιηθεί στο FPGA και αποστέλλει τα bytes των πακέτων. Από την πλευρά του FPGA, η θύρα αντιμετωπίζεται ως κλασική ασύγχρονη UART γραμμή: σε κατάσταση ηρεμίας η γραμμή βρίσκεται σε λογικό 1, η μετάδοση κάθε byte ξεκινά με start bit στο λογικό 0 και ολοκληρώνεται με stop bit στο λογικό 1.

Η σύνδεση δεν χρησιμοποιεί επίπεδα τάσης RS-232, αλλά λογικές στάθμες 3.3 V. Στο αρχείο αντιστοίχισης ακροδεκτών του Quartus τα σήματα rx και tx έχουν οριστεί ως 3.3-V LVTTL, ενώ η γραμμή rx αντιστοιχίζεται στο PIN_AC18 και η γραμμή tx στο PIN_Y17.



Σχήμα 4.3*: Φυσική σύνδεση host, CP2102 και FPGA.*

Πίνακας 4.2*: Αντιστοίχιση σημάτων UART στη φυσική σύνδεση.*

Σήμα	Ακίδα FPGA	Σύνδεση	Περιγραφή
rx	PIN_AC18 / GPIO_0[0]	CP2102 TX	Είσοδος δεδομένων προς το FPGA.
tx	PIN_Y17 / GPIO_0[1]	CP2102 RX	Έξοδος δεδομένων από το FPGA προς τον host.
GND	Κοινή γείωση	GND CP2102	Αναφορά τάσης για τα σήματα UART.
I/O standard	3.3-V LVTTL	TTL UART	Λογικές στάθμες συμβατές με τη DE1-SoC.

Η ασύγχρονη φύση του UART σημαίνει ότι δεν υπάρχει ξεχωριστό σήμα ρολογιού μεταξύ host και FPGA. Ο δέκτης πρέπει να αναγνωρίσει την πτώση της γραμμής από το λογικό 1 στο λογικό 0, να επιβεβαιώσει ότι πρόκειται για start bit και στη συνέχεια να δειγματοληπτήσει τα bits δεδομένων σε κατάλληλες χρονικές στιγμές. Για τον λόγο αυτό τα modules uart_rx.v και uart_tx.v βασίζονται στο ρολόι των 50 MHz της πλακέτας.

Ο ρυθμός μετάδοσης έχει επιλεγεί στα 115200 baud. Με ρολόι 50 MHz, η διάρκεια ενός bit αντιστοιχεί σε περίπου 434 κύκλους ρολογιού. Στον δέκτη χρησιμοποιούνται δύο σταθερές: FULL_BIT = 434 για τη διάρκεια ενός bit και HALF_BIT = 217 για τη δειγματοληψία στο μέσο του start bit. Η προσέγγιση αυτή είναι επαρκής για τη συγκεκριμένη εφαρμογή, επειδή τα πακέτα είναι μικρά και η ταχύτητα 115200 baud παρέχει αρκετό περιθώριο ως προς τον χρόνο εκτέλεσης των μηχανικών κινήσεων.

Σχήμα 4.4*: Χρονική μορφή ενός UART frame 8N1.*

Πίνακας 4.3*: Παράμετροι υλοποίησης UART.*

Παράμετρος	Τιμή	Σχόλιο
Ρολόι FPGA	50 MHz	Κοινό ρολόι των UART modules και του υπόλοιπου ελεγκτή.
Baud rate	115200 baud	Ίδια τιμή σε Python και Verilog.

Μορφή frame	8N1	8 data bits, no parity, 1 stop bit.
FULL_BIT	434 κύκλοι	50.000.000 / 115.200 ≈ 434.
HALF_BIT	217 κύκλοι	Δειγματοληψία στο μέσο του start bit.
Σειρά bits	LSB first	Το λιγότερο σημαντικό bit αποστέλλεται πρώτο.
Idle κατάσταση	λογικό 1	Η γραμμή παραμένει υψηλή όταν δεν μεταδίδεται byte.

4.3 Υλοποίηση δέκτη UART

Ο δέκτης uart_rx.v μετατρέπει τη σειριακή γραμμή rx σε παράλληλο byte. Επειδή το σήμα rx προέρχεται από εξωτερικό κύκλωμα και δεν είναι συγχρονισμένο με το ρολόι του FPGA, περνά πρώτα από δύο καταχωρητές συγχρονισμού, rx_s1 και rx_s2. Με αυτόν τον τρόπο μειώνεται η πιθανότητα μετασταθερότητας πριν το σήμα χρησιμοποιηθεί από τη μηχανή καταστάσεων.

Η μηχανή καταστάσεων του δέκτη περιλαμβάνει τέσσερις καταστάσεις: S_IDLE, S_START, S_RECEIVE και S_STOP. Στην κατάσταση S_IDLE η γραμμή παρακολουθείται μέχρι να ανιχνευθεί μετάβαση στο λογικό 0. Στη συνέχεια, στην κατάσταση S_START, το κύκλωμα περιμένει 217 κύκλους και ελέγχει αν η γραμμή εξακολουθεί να είναι χαμηλή. Αν ναι, η λήψη θεωρείται έγκυρη και ξεκινά η δειγματοληψία των 8 bits δεδομένων.

Στην κατάσταση S_RECEIVE ο δέκτης δειγματοληπτεί τη γραμμή ανά 434 κύκλους και αποθηκεύει κάθε bit στον καταχωρητή rx_shift. Μετά τη λήψη του όγδοου bit περνά στην κατάσταση S_STOP. Αν το stop bit είναι υψηλό, το byte αντιγράφεται στην έξοδο rx_data και το σήμα rx_valid ενεργοποιείται για έναν κύκλο ρολογιού. Το rx_valid χρησιμοποιείται από το packet_decoder ως ένδειξη ότι υπάρχει νέο έγκυρο byte προς συναρμολόγηση.

Πίνακας 4.4*: Καταστάσεις του δέκτη uart_rx.*

Κατάσταση	**Λειτουργία**	**Έξοδος / μετάβαση**
S_IDLE	Αναμονή με τη γραμμή σε λογικό 1.	Μετάβαση σε S_START όταν ανιχνευθεί λογικό 0.
S_START	Έλεγχος του start bit στο μέσο της διάρκειάς του.	Αν παραμένει χαμηλό, αρχίζει η λήψη δεδομένων.
S_RECEIVE	Δειγματοληψία των 8 data bits ανά FULL_BIT.	Τα bits αποθηκεύονται στον rx_shift.
S_STOP	Έλεγχος stop bit.	Αν είναι υψηλό, παράγεται rx_valid για έναν κύκλο.

4.4 Υλοποίηση πομπού UART

Ο πομπός uart_tx.v εκτελεί την αντίστροφη διαδικασία. Λαμβάνει ένα παράλληλο byte στην είσοδο tx_data και, όταν ενεργοποιηθεί το tx_start, το μεταδίδει σειριακά στη γραμμή tx. Όσο διαρκεί η μετάδοση, το σήμα tx_busy παραμένει ενεργό ώστε το κύκλωμα που ζητά τη μετάδοση να μην ξεκινήσει νέο byte πριν ολοκληρωθεί το προηγούμενο.

Η γραμμή tx βρίσκεται σε λογικό 1 στην κατάσταση IDLE. Με την ενεργοποίηση του tx_start, το byte αποθηκεύεται στον εσωτερικό καταχωρητή tx_shift και η μηχανή καταστάσεων περνά στην κατάσταση START, όπου παράγεται το start bit. Ακολουθεί η κατάσταση DATA, στην οποία αποστέλλονται τα 8 bits με σειρά από το λιγότερο προς το περισσότερο σημαντικό, και τέλος η κατάσταση STOP, όπου η γραμμή επιστρέφει σε λογικό 1 για ένα bit time.

Στην παρούσα εφαρμογή ο πομπός χρησιμοποιείται κυρίως για την αποστολή της επιβεβαίωσης ολοκλήρωσης κίνησης. Στο top.v το tx_data είναι σταθερά συνδεδεμένο στην τιμή 8'hAA. Όταν το σήμα busy του step_gen μεταβεί από 1 σε 0, δηλαδή όταν ολοκληρωθεί η τρέχουσα κίνηση, ενεργοποιείται το tx_start και αποστέλλεται το ACK προς το Python script.

Πίνακας 4.5*: Καταστάσεις του πομπού uart_tx.*

Κατάσταση	Λειτουργία	Χρονισμός
IDLE	Η γραμμή tx παραμένει σε λογικό 1.	Αναμονή για tx_start.
START	Μετάδοση start bit στο λογικό 0.	Διάρκεια FULL_BIT.
DATA	Μετάδοση των 8 bits του tx_shift.	Ένα bit ανά FULL_BIT, LSB first.
STOP	Μετάδοση stop bit στο λογικό 1.	Μετά την ολοκλήρωση, tx_busy=0.

Η συγκεκριμένη σχεδίαση του UART είναι σκόπιμα απλή. Δεν χρησιμοποιείται parity bit, FIFO ή μηχανισμός αυτόματης επαναμετάδοσης. Η αξιοπιστία της επικοινωνίας βασίζεται στη μικρή απόσταση της σύνδεσης, στη σωστή επιλογή στάθμης τάσης, στη σταθερότητα του ρυθμού μετάδοσης και στον ανώτερο μηχανισμό handshake που περιγράφεται στο επόμενο υποκεφάλαιο. Αν το FPGA λάβει πλήρες και έγκυρο byte, το προωθεί στο packet_decoder· αν η λήψη διακοπεί, ο packet_decoder διαθέτει δικό του timeout ώστε να απορρίπτεται η μερική ακολουθία bytes.

Με αυτόν τον τρόπο ο CP2102 λειτουργεί ως γέφυρα φυσικού επιπέδου, ενώ όλη η λογική ερμηνείας των δεδομένων παραμένει καθαρά χωρισμένη: η Python χειρίζεται τη θύρα COM και τη δημιουργία των bytes, το uart_rx/uart_tx χειρίζεται τη σειριακή μετάδοση και το packet_decoder αναλαμβάνει τη σημασιολογική αποκωδικοποίηση των πακέτων. Ο διαχωρισμός αυτός διευκολύνει τη δοκιμή κάθε βαθμίδας ξεχωριστά και επιτρέπει τη μελλοντική αντικατάσταση του φυσικού μέσου επικοινωνίας χωρίς αλλαγή στη λογική παραγωγής παλμών

4.5 Δομή πακέτων

Στο προηγούμενο στάδιο η πληροφορία της διαδρομής έχει ήδη μετατραπεί σε αριθμούς: τύπο κίνησης, παλμούς στον άξονα X, παλμούς στον άξονα Y και πρόωση. Το FPGA δεν λαμβάνει την εντολή G-code ως κείμενο, για παράδειγμα G1 X10 Y5 F1200. Αντίθετα, λαμβάνει μια μικρή σειρά από bytes, σε γνωστή και σταθερή σειρά.

Αυτή η επιλογή απλοποιεί την υλοποίηση στο FPGA. Το κύκλωμα δεν χρειάζεται να διαβάζει χαρακτήρες, να ψάχνει κενά ή να αναγνωρίζει λέξεις όπως X, Y και F. Απλώς μετράει τα bytes που έρχονται και γνωρίζει από πριν τι σημαίνει κάθε θέση μέσα στο πακέτο.

Στην παρούσα υλοποίηση χρησιμοποιούνται δύο είδη πακέτων. Το πακέτο αρχικοποίησης στέλνει παραμέτρους της μηχανής, ενώ το πακέτο κίνησης περιγράφει μία συγκεκριμένη μετακίνηση. Η αναγνώριση γίνεται από το πρώτο byte: αν είναι 0xFF, το πακέτο θεωρείται πακέτο αρχικοποίησης· διαφορετικά θεωρείται πακέτο κίνησης.

Τα αριθμητικά πεδία που έχουν περισσότερα από ένα byte στέλνονται σε μορφή big-endian. Αυτό σημαίνει ότι αποστέλλεται πρώτα το πιο σημαντικό byte του αριθμού. Για παράδειγμα, ο αριθμός 320 σε πεδίο 32 bit γράφεται ως 00 00 01 40. Στο FPGA τα bytes ενώνονται ξανά με την ίδια σειρά ώστε να σχηματιστεί ο αρχικός αριθμός.

Πίνακας 4.7: Τύποι πακέτων του πρωτοκόλλου.

Τύπος	Πρώτο byte	Μήκος	Σκοπός
Αρχικοποίησης	0xFF	9 bytes	Μεταφορά των συντελεστών NUMERATOR_X και NUMERATOR_Y.
Κίνησης	0x00-0x03	11 bytes	Μεταφορά τύπου κίνησης, παλμών X/Y και πρόωσης F.

4.6 Πακέτο αρχικοποίησης

Το πακέτο αρχικοποίησης αποστέλλεται πριν από τις κινήσεις και δεν αντιστοιχεί σε μετατόπιση της κεφαλής. Ο ρόλος του είναι να ενημερώσει το FPGA για τους συντελεστές που χρησιμοποιούνται στον υπολογισμό της χρονικής απόστασης των παλμών.

Οι συντελεστές αυτοί είναι οι NUMERATOR_X και NUMERATOR_Y. Με απλά λόγια, συνδέουν το ρολόι των 50 MHz, τη μηχανική ανάλυση του άξονα και την τιμή πρόωσης. Έτσι η ίδια λογική Verilog μπορεί να χρησιμοποιηθεί ακόμη και αν αλλάξουν οι ρυθμίσεις των αξόνων, αρκεί να σταλούν νέες τιμές αρχικοποίησης.

Το πρώτο byte είναι πάντα 0xFF και λειτουργεί ως marker. Όταν το packet_decoder δει αυτή την τιμή ως πρώτο byte, περιμένει συνολικά 9 bytes. Όταν ολοκληρωθεί η λήψη, σχηματίζει τα δύο αριθμητικά πεδία και ενεργοποιεί για έναν κύκλο ρολογιού το σήμα cfg_valid.

Σχήμα 4.4: Δομή πακέτου αρχικοποίησης.

Πίνακας 4.8: Πεδία πακέτου αρχικοποίησης.

Byte(s)	Πεδίο	Τύπος	Περιγραφή
0	Marker	uint8	Σταθερή τιμή 0xFF που δηλώνει πακέτο αρχικοποίησης.

1-4	NUMERATOR_X	uint32	Συντελεστής υπολογισμού παλμών για τον άξονα X.
5-8	NUMERATOR_Y	uint32	Συντελεστής υπολογισμού παλμών για τον άξονα Y.

4.7 Πακέτο κίνησης

Το πακέτο κίνησης είναι το βασικό μήνυμα που στέλνει ο host στο FPGA. Κάθε τέτοιο πακέτο περιγράφει την επόμενη κίνηση που πρέπει να γίνει. Το μήκος του είναι πάντα 11 bytes, ανεξάρτητα από το αν η μετατόπιση είναι μικρή ή μεγάλη.

Το πρώτο byte δηλώνει τον τύπο της κίνησης. Τα επόμενα οκτώ bytes δηλώνουν τους παλμούς στους δύο άξονες, ενώ τα δύο τελευταία bytes μεταφέρουν την πρόωση F. Η σταθερή θέση κάθε πεδίου κάνει την αποκωδικοποίηση απλή: το FPGA δεν αναζητά τέλος γραμμής, αλλά περιμένει να συμπληρωθούν ακριβώς 11 bytes.

Τα StepsX και StepsY είναι προσημασμένοι αριθμοί. Το πρόσημο δείχνει τη φορά της κίνησης στον αντίστοιχο άξονα, ενώ η απόλυτη τιμή δείχνει πόσοι παλμοί πρέπει να παραχθούν. Με αυτόν τον τρόπο δεν χρειάζεται ξεχωριστό πεδίο κατεύθυνσης μέσα στο πακέτο.

Σχήμα 4.5*: Δομή πακέτου κίνησης.*

Πίνακας 4.6*: Πεδία πακέτου κίνησης*

Byte(s)	Πεδίο	Τύπος	Περιγραφή
0	Motion	uint8	Κωδικός που δηλώνει αν η κίνηση είναι G0, G1, G2 ή G3.
1-4	StepsX	int32	Παλμοί και φορά κίνησης στον άξονα X.
5-8	StepsY	int32	Παλμοί και φορά κίνησης στον άξονα Y.
9-10	FeedRate	uint16	Τιμή πρόωσης F που χρησιμοποιείται στον χρονισμό της κίνησης.

Πίνακας 4.7*: Κωδικοποίηση πεδίου Motion*

Τιμή	Εντολή	Λειτουργία	Κατάσταση laser
0	G0	Ταχεία μετακίνηση χωρίς χάραξη.	OFF

1	G1	Γραμμική κίνηση χάραξης.	ON κατά την κίνηση
2	G2	Κυκλικό τόξο δεξιόστροφα.	ON κατά την κίνηση
3	G3	Κυκλικό τόξο αριστερόστροφα.	ON κατά την κίνηση

4.8 Αναπαράσταση αριθμών

Στο πακέτο υπάρχουν διαφορετικοί τύποι αριθμών, επειδή δεν έχουν όλα τα πεδία την ίδια ανάγκη. Οι παλμοί των αξόνων χρειάζονται πρόσημο, γιατί η κεφαλή μπορεί να κινηθεί και προς τις δύο κατευθύνσεις. Αντίθετα, η πρόωση και οι συντελεστές αρχικοποίησης είναι θετικές τιμές.Το FPGA λαμβάνει τα bytes και το step_gen χρησιμοποιεί το πρόσημο για να ορίσει τη γραμμή DIR.

Πίνακας 4.8*: Αριθμητικά πεδία και εύρη τιμών*

Πεδίο	Τύπος	Εύρος	Χρήση
NUMERATOR_X/Y	uint32	1 έως 4.294.967.295	Παράμετροι υπολογισμού παλμών ανά άξονα.
StepsX/Y	int32	-2.147.483.648 έως 2.147.483.647	Πλήθος παλμών και κατεύθυνση κίνησης.
FeedRate	uint16	0 έως 65.535	Πρόωση F που χρησιμοποιείται από το step_gen.
Motion	uint8	0 έως 3 στην πράξη	Επιλογή τύπου κίνησης και κατάστασης laser.

4.9 Παράδειγμα κωδικοποίησης πακέτου

Αν ο host θέλει να στείλει πακέτο αρχικοποίησης με NUMERATOR_X = NUMERATOR_Y = 18.750.000, η Python δημιουργεί τα bytes με τη μορφή >BII. Το σύμβολο > δηλώνει big-endian, το B δηλώνει ένα byte και το I δηλώνει ακέραιο 32 bit χωρίς πρόσημο. Η σειρά που μεταδίδεται είναι: FF 01 1E 1A 30 01 1E 1A 30.

Στη συνέχεια, έστω ότι πρέπει να σταλεί μία γραμμική κίνηση G1 με StepsX = 320, StepsY = -160 και FeedRate = 1200. Η Python δημιουργεί το πακέτο με τη μορφή >BiiH. Το τελικό πακέτο έχει 11 bytes και αποστέλλεται ως μία ενιαία ακολουθία.

Πίνακας 4.9 *Παράδειγμα πακέτου κίνησης G1.*

Πεδίο	Τιμή	Bytes σε hex
Motion	1	01
StepsX	320	00 00 01 40

StepsY	-160	FF FF FF 60
FeedRate	1200	04 B0
Συνολικό πακέτο	-	01 00 00 01 40 FF FF FF 60 04 B0

4.10 Αποκωδικοποίηση στο FPGA

Στην πλευρά του FPGA, το uart_rx παραδίδει ένα byte κάθε φορά στο packet_decoder. Κάθε φορά που ολοκληρώνεται σωστά η λήψη ενός byte, ενεργοποιείται το σήμα rx_valid και το byte βρίσκεται στην έξοδο rx_data.

Το packet_decoder αποθηκεύει προσωρινά τα bytes και αυξάνει έναν μετρητή. Το πρώτο byte καθορίζει τον τύπο του πακέτου. Αν είναι 0xFF, το κύκλωμα περιμένει 9 bytes συνολικά. Αν δεν είναι 0xFF, περιμένει 11 bytes και το αντιμετωπίζει ως πακέτο κίνησης.

Όταν συμπληρωθεί το σωστό πλήθος bytes, το κύκλωμα ενώνει τα αντίστοιχα bytes και παράγει τα σήματα εξόδου. Για πακέτο αρχικοποίησης ενεργοποιείται το cfg_valid. Για πακέτο κίνησης ενεργοποιείται το pkt_valid, το οποίο ενημερώνει το step_gen ότι υπάρχει νέα κίνηση προς εκτέλεση.

Υπάρχει επίσης μετρητής timeout. Αν ξεκινήσει η λήψη ενός πακέτου αλλά σταματήσουν να έρχονται bytes για περίπου 100 ms, το packet_decoder μηδενίζει τον εσωτερικό μετρητή και επιστρέφει σε αναμονή. Έτσι ένα μισό ή λανθασμένο πακέτο δεν οδηγεί σε λανθασμένη κίνηση της μηχανής.

Πίνακας 4.10*: Έξοδοι του packet_decoder.*

Σήμα	**Πότε παράγεται**	**Χρήση**
motion	Με την ολοκλήρωση πακέτου κίνησης	Δηλώνει G0/G1/G2/G3 και επηρεάζει το laser.
steps_x	Από τα bytes 1-4 του πακέτου κίνησης	Παλμοί και φορά για τον άξονα X.
steps_y	Από τα bytes 5-8 του πακέτου κίνησης	Παλμοί και φορά για τον άξονα Y.
feed_rate	Από τα bytes 9-10 του πακέτου κίνησης	Χρησιμοποιείται στον χρονισμό των παλμών.
pkt_valid	Όταν ολοκληρωθεί πακέτο κίνησης	Ενημερώνει το step_gen ότι υπάρχει νέα κίνηση.
numerator_x/y	Όταν ολοκληρωθεί πακέτο αρχικοποίησης	Παράμετροι μηχανικής ανάλυσης.
cfg_valid	Όταν ολοκληρωθεί πακέτο αρχικοποίησης	Ενημερώνει το step_gen για νέους συντελεστές.

Συνοπτικά, η δομή των πακέτων λειτουργεί σαν ένας συμφωνημένος χάρτης θέσεων. Ο host και το FPGA γνωρίζουν ότι κάθε byte έχει συγκεκριμένη σημασία. Έτσι η επικοινωνία παραμένει μικρή, γρήγορη και κατάλληλη για FPGA.

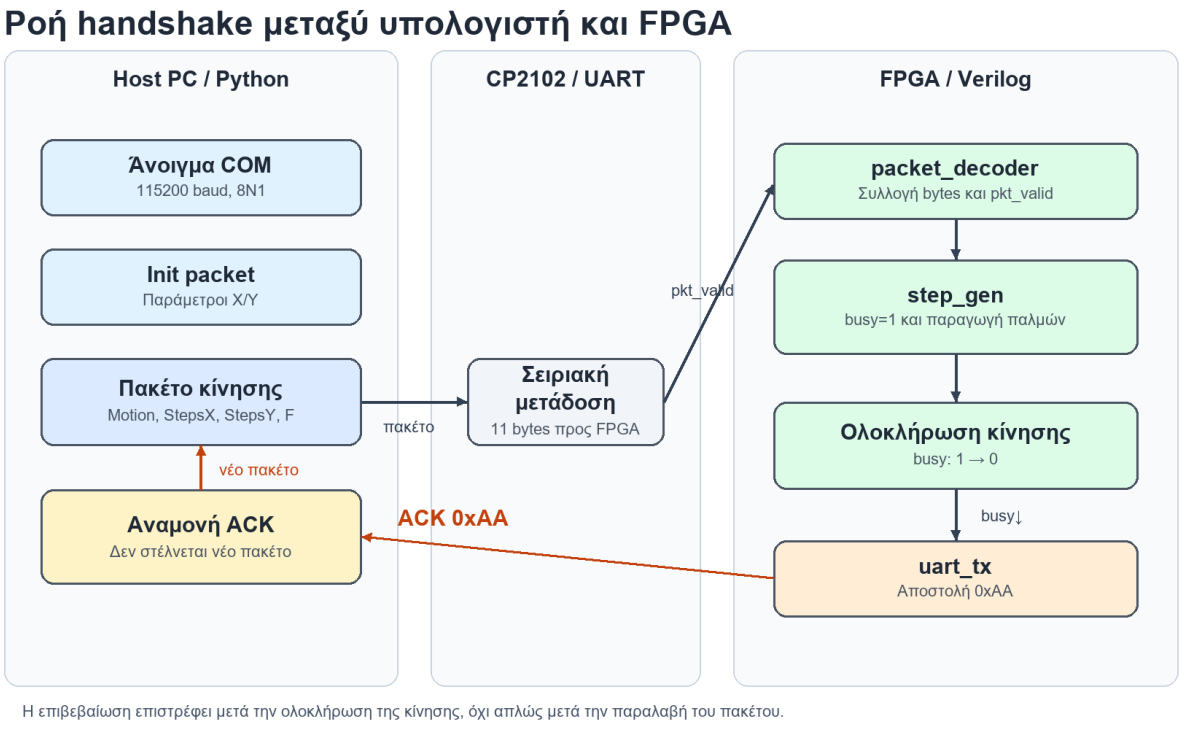
4.11 Μηχανισμός handshake και ροή δεδομένων

Αφού έχει οριστεί η μορφή των πακέτων, χρειάζεται να οριστεί και ο τρόπος με τον οποίο αυτά στέλνονται. Το βασικό ερώτημα είναι, πότε επιτρέπεται στον υπολογιστή να στείλει το επόμενο πακέτο κίνησης στο FPGA.

Στην παρούσα υλοποίηση χρησιμοποιείται λογική stop-and-wait. Δηλαδή ο υπολογιστής στέλνει ένα πακέτο κίνησης, σταματά προσωρινά την αποστολή και περιμένει απάντηση από το FPGA. Μόνο όταν λάβει την απάντηση αυτή, συνεχίζει με το επόμενο πακέτο.

Η απάντηση του FPGA είναι ένα μόνο byte με τιμή 0xAA. Το byte αυτό ονομάζεται ACK, από το acknowledgement. Στη συγκεκριμένη εφαρμογή το ACK δεν σημαίνει απλώς ότι το πακέτο έφτασε. Σημαίνει ότι η κίνηση που αντιστοιχούσε στο πακέτο ολοκληρώθηκε.

Αυτό φαίνεται από τον τρόπο παραγωγής του ACK στο Verilog. Η μονάδα step_gen κρατά το σήμα busy σε λογικό 1 όσο εκτελείται κίνηση. Όταν η κίνηση τελειώσει, το busy πέφτει σε 0. Το top ανιχνεύει αυτή την πτώση και τότε ζητά από το uart_tx να στείλει το 0xAA προς τον υπολογιστή.



Σχήμα 4.6: *Ροή δεδομένων και επιβεβαίωσης μεταξύ υπολογιστή, UART και FPGA.*

Πίνακας 4.11*: Βασικά βήματα του μηχανισμού handshake.*

Βήμα	Πλευρά	Ενέργεια	Απλή ερμηνεία
1	Host PC	Άνοιγμα σειριακής θύρας	Η Python συνδέεται με τη θύρα COM του CP2102.
2	Host PC	Καθαρισμός buffers	Απομακρύνονται παλιά ή άσχετα bytes από προηγούμενη χρήση.
3	Host PC → FPGA	Αποστολή πακέτου κίνησης	Στέλνονται τα 11 bytes της επόμενης κίνησης.
4	FPGA	Αποκωδικοποίηση πακέτου	Το packet_decoder σχηματίζει motion, steps_x, steps_y και feed_rate.
5	FPGA	Εκτέλεση κίνησης	Το step_gen παράγει STEP/DIR και κρατά busy=1.
6	FPGA → Host PC	Αποστολή ACK 0xAA	Το ACK στέλνεται όταν η κίνηση ολοκληρωθεί.
7	Host PC	Συνέχιση αποστολής	Η Python λαμβάνει το ACK και στέλνει την επόμενη κίνηση.

4.12 Ροή στην πλευρά του υπολογιστή

Στην πλευρά του υπολογιστή, το πρόγραμμα Python διαβάζει το αρχείο βημάτων που έχει παραχθεί από το προηγούμενο στάδιο. Για κάθε γραμμή του αρχείου δημιουργεί ένα δυαδικό πακέτο κίνησης και το στέλνει στη σειριακή θύρα. Αμέσως μετά την αποστολή, η Python δεν περνά στην επόμενη γραμμή. Καλεί τη διαδικασία αναμονής ACK και ελέγχει τα bytes που επιστρέφουν από το FPGA. Αν λάβει το byte 0xAA, θεωρεί ότι η κίνηση ολοκληρώθηκε και επιτρέπεται να σταλεί η επόμενη.

Αν φτάσει διαφορετικό byte, εμφανίζεται προειδοποίηση και η αναμονή συνεχίζεται. Αν δεν φτάσει ACK μέσα στον προκαθορισμένο χρόνο, για παράδειγμα μέσα σε 10 s στη βασική έκδοση αποστολής, η μετάδοση θεωρείται προβληματική. Έτσι ο υπολογιστής δεν συνεχίζει την αποστολή κινήσεων.

Πίνακας 4.12*: Ρόλος των βασικών σημάτων στη ροή ελέγχου.*

Σήμα / μεταβλητή	Πού βρίσκεται	Ρόλος
ACK_BYTE = 0xAA	Python	Η τιμή που περιμένει ο υπολογιστής ως επιβεβαίωση ολοκλήρωσης.
ACK_TIMEOUT_SEC	Python	Μέγιστος χρόνος αναμονής για ACK πριν θεωρηθεί ότι υπάρχει πρόβλημα.
rx_valid	uart_rx	Δείχνει ότι μόλις παραλήφθηκε ένα έγκυρο byte.
pkt_valid	packet_decoder	Δείχνει ότι ολοκληρώθηκε πλήρες πακέτο κίνησης.
busy	step_gen	Είναι 1 όσο εκτελείται κίνηση και 0 όταν η μονάδα είναι ελεύθερη.

busy_prev	top	Κρατά την προηγούμενη τιμή του busy για να εντοπιστεί η πτώση 1→0.
tx_start	top → uart_tx	Ξεκινά την αποστολή του ACK προς τον υπολογιστή.
tx_busy	uart_tx	Δείχνει ότι ο πομπός UART είναι ακόμη απασχολημένος.

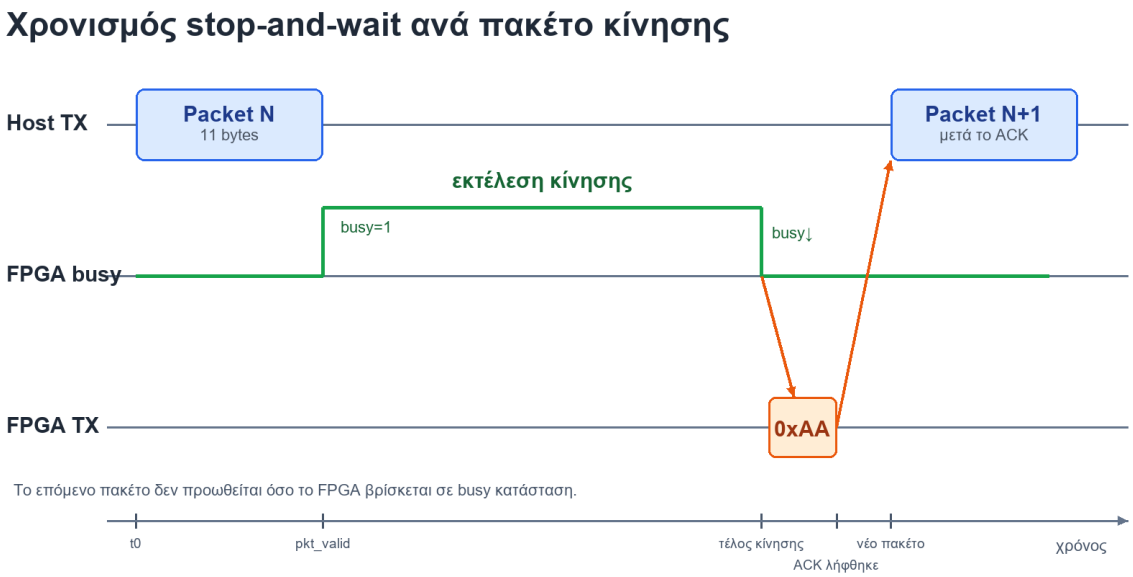
4.13 Ροή στην πλευρά του FPGA

Στην πλευρά του FPGA, η ροή ξεκινά από το uart_rx. Το uart_rx μετατρέπει το σειριακό σήμα UART σε bytes. Κάθε φορά που ολοκληρώνεται ένα byte, τοποθετείται στην έξοδο rx_data και ενεργοποιείται το rx_valid.

Το packet_decoder παίρνει αυτά τα bytes, τα μετρά και τα τοποθετεί στις σωστές θέσεις του πακέτου. Όταν συμπληρωθούν τα 11 bytes ενός πακέτου κίνησης, σχηματίζονται τα πεδία motion, steps_x, steps_y και feed_rate. Ταυτόχρονα παράγεται το pkt_valid για έναν κύκλο ρολογιού.

Το step_gen χρησιμοποιεί τα πεδία αυτά για να εκτελέσει την κίνηση. Όσο παράγονται οι παλμοί των κινητήρων, το busy είναι 1. Όταν ολοκληρωθούν οι παλμοί και για τους δύο άξονες, το busy γίνεται 0.

Το ACK παράγεται στο top επίπεδο. Εκεί συγκρίνεται η προηγούμενη τιμή του busy με την τρέχουσα. Όταν η προηγούμενη τιμή ήταν 1 και η τρέχουσα είναι 0, έχει τελειώσει η κίνηση. Τότε, αν το uart_tx είναι ελεύθερο, δίνεται παλμός tx_start και μεταδίδεται το byte 0xAA.



Σχήμα 4.7*: Χρονική ακολουθία stop-and-wait για δύο διαδοχικά πακέτα κίνησης.*

4.14 Χρονισμός και επίδραση στην αποστολή

Στο UART κάθε byte μεταδίδεται με 10 bits: ένα start bit, 8 bits δεδομένων και ένα stop bit. Με ρυθμό 115200 baud, ένα πακέτο κίνησης 11 bytes χρειάζεται περίπου 0,96 ms για να μεταδοθεί. Το ACK, που είναι ένα μόνο byte, χρειάζεται περίπου 0,087 ms.

Οι χρόνοι αυτοί είναι πολύ μικροί σε σχέση με τον χρόνο μιας πραγματικής μηχανικής κίνησης. Επομένως, η αναμονή για ACK δεν επιβαρύνει ουσιαστικά την ταχύτητα της μηχανής. Αντίθετα, προσφέρει σαφή συγχρονισμό ανάμεσα στον υπολογιστή και στο FPGA.

Το σημαντικό πλεονέκτημα είναι ότι αποφεύγεται η αποστολή νέας κίνησης όσο η προηγούμενη εκτελείται. Επειδή το σύστημα δεν χρησιμοποιεί ουρά πολλών πακέτων, το stop-and-wait εξασφαλίζει ότι κάθε πακέτο εκτελείται με τη σειρά του και δεν χάνεται επειδή η step_gen είναι ακόμη απασχολημένη.

Πίνακας 4.13*: Μηχανισμοί προστασίας και χειρισμού σφαλμάτων.*

Περίπτωση	Πώς ανιχνεύεται	Τι γίνεται
Μερικώς ληφθέν πακέτο	Ξεκίνησε πακέτο αλλά δεν φτάνουν νέα bytes.	Μετά από περίπου 100 ms το packet_decoder μηδενίζει τον μετρητή του.
Μη αναμενόμενο byte	Η Python λαμβάνει byte διαφορετικό από 0xAA.	Εμφανίζει προειδοποίηση και συνεχίζει να περιμένει σωστό ACK.
Δεν λαμβάνεται ACK	Περνά ο χρόνος ACK_TIMEOUT_SEC.	Η μετάδοση θεωρείται προβληματική και διακόπτεται ή ζητείται απόφαση από τον χρήστη.
Ολοκλήρωση κίνησης	Το busy πέφτει από 1 σε 0.	Το top στέλνει το ACK 0xAA προς τον υπολογιστή.

4.15 Πρακτικό παράδειγμα συναλλαγής

Έστω ότι πρέπει να σταλεί ένα πακέτο κίνησης G1 με steps_x = 320, steps_y = -160 και feed_rate = 1200. Στο επίπεδο των δεδομένων, το πακέτο είναι η ακολουθία 11 bytes που έχει περιγραφεί στην ενότητα 4.3. Στο επίπεδο του handshake, όμως, το βασικό σημείο είναι η σειρά των ενεργειών.

Πρώτα η Python στέλνει τα 11 bytes. Μετά σταματά και περιμένει. Το FPGA λαμβάνει το πακέτο, το αποκωδικοποιεί και εκτελεί την κίνηση. Μόλις η κίνηση τελειώσει, το FPGA στέλνει 0xAA. Μόνο τότε η Python συνεχίζει με την επόμενη γραμμή του αρχείου.

Πίνακας 4.14*: Παράδειγμα πλήρους συναλλαγής πακέτου κίνησης.*

Στάδιο	Δεδομένο / σήμα	Ερμηνεία
1	01 00 00 01 40 FF FF FF 60 04 B0	Η Python στέλνει πακέτο κίνησης 11 bytes.
2	rx_valid	Το uart_rx ενημερώνει το packet_decoder για κάθε byte.

3	pkt_valid = 1	Το packet_decoder έχει πλήρες πακέτο κίνησης.
4	busy = 1	Το step_gen εκτελεί την κίνηση.
5	busy: 1 → 0	Η κίνηση ολοκληρώνεται.
6	0xAA	Το uart_tx στέλνει ACK προς την Python.
7	Επόμενο πακέτο	Η Python συνεχίζει την αποστολή.

Ο μηχανισμός handshake είναι απλός, αλλά πολύ σημαντικός για τη σωστή λειτουργία της μηχανής. Ο υπολογιστής δεν στέλνει συνεχώς εντολές χωρίς έλεγχο. Στέλνει μία κίνηση, περιμένει να ολοκληρωθεί πραγματικά στο FPGA και μετά συνεχίζει.

Με αυτόν τον τρόπο η σειρά των κινήσεων παραμένει ελεγχόμενη, δεν απαιτείται FIFO πολλών πακέτων στο FPGA και η Python έχει άμεση ένδειξη ότι η προηγούμενη κίνηση τελείωσε. Άρα το ACK δεν είναι απλώς ένα μήνυμα επικοινωνίας, αλλά ο βασικός μηχανισμός συγχρονισμού ανάμεσα στο λογισμικό αποστολής και την πραγματική κατάσταση του ελεγκτή στο FPGA.

Κεφάλαιο 5ο: Σχεδίαση και υλοποίηση ελεγκτή FPGA

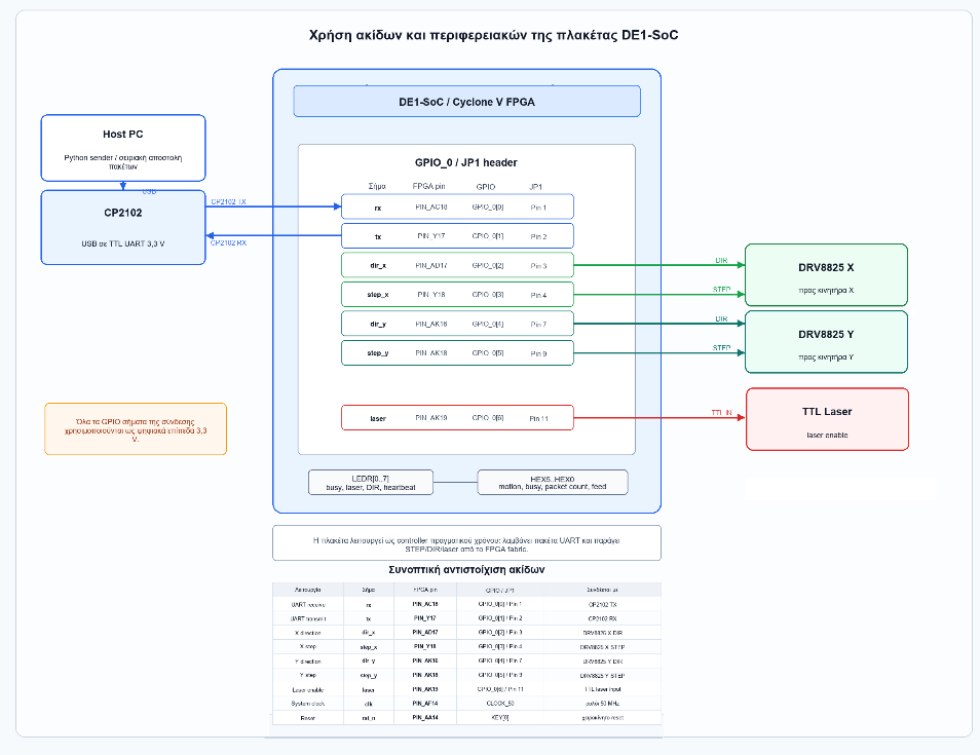
5.1 Πλατφόρμα ανάπτυξης (DE1-SoC, Cyclone V, Quartus II)

Η υλοποίηση του ελεγκτή CNC βασίστηκε στην αναπτυξιακή πλακέτα DE1-SoC της Terasic. Η πλακέτα ενσωματώνει FPGA της οικογένειας Cyclone V SoC, κύκλωμα προγραμματισμού USB-Blaster II και αρκετές διαθέσιμες θύρες εισόδου/εξόδου. Για την παρούσα εργασία χρησιμοποιείται το προγραμματιζόμενο λογικό τμήμα του FPGA, στο οποίο υλοποιούνται οι μονάδες UART, αποκωδικοποίησης πακέτων, παραγωγής παλμών και ελέγχου κεφαλής.

Η παραγωγή παλμών STEP/DIR γίνεται μέσα στο FPGA, χωρίς να εξαρτάται από λειτουργικό σύστημα ή από καθυστερήσεις λογισμικού στον υπολογιστή. Ο υπολογιστής αναλαμβάνει τη δημιουργία και αποστολή των πακέτων, ενώ το FPGA αναλαμβάνει την εκτέλεση των κινήσεων σε πραγματικό χρόνο.

5.1.1 Χρήση ακίδων και περιφερειακών της πλακέτας

Η σύνδεση της πλακέτας με το υπόλοιπο σύστημα γίνεται μέσω συγκεκριμένων σημάτων του top-level module. Το σήμα clk συνδέεται στο CLOCK_50 της πλακέτας. Τα σήματα rx και tx χρησιμοποιούνται για τη σειριακή επικοινωνία με τον CP2102, ενώ τα σήματα dir_x, step_x, dir_y και step_y οδηγούν τους drivers των βηματικών κινητήρων. Επιπλέον, το σήμα laser χρησιμοποιείται ως ψηφιακή έξοδος ενεργοποίησης του TTL laser.



Σχήμα 5.1: Χρήση pin και περιφερειακών της πλακέτας DE1-SoC για τη σύνδεση UART, οδηγών DRV8825 και TTL laser

5.1.2 Περιβάλλον Quartus II

Η ανάπτυξη και σύνθεση της Verilog υλοποίησης πραγματοποιείται στο Quartus II 13.1 Web Edition. Στο αρχείο CNC_Controller.qpf ορίζεται το project revision, ενώ στο CNC_Controller.qsf δηλώνονται η συσκευή στόχος, το top-level entity, τα Verilog αρχεία και οι αναθέσεις pin.

Μετά τη σύνθεση και το fitting, το Quartus παράγει το αρχείο .sof. Το αρχείο αυτό φορτώνεται στο FPGA μέσω του Programmer και του on-board USB-Blaster II. Μετά τον προγραμματισμό, η λογική λειτουργεί απευθείας στο FPGA και μπορεί να δεχθεί πακέτα μέσω της σειριακής σύνδεσης.

Πίνακας 5.1*: Ρόλος βασικών αρχείων του Quartus project*

Αρχείο / στοιχείο	**Ρόλος**	**Σχέση με την υλοποίηση**
CNC_Controller.qpf	Αρχείο project	Ορίζει το Quartus project και το ενεργό revision.
CNC_Controller.qsf	Αρχείο ρυθμίσεων	Περιέχει device, top entity, source files και pin assignments.
top.v	Top-level module	Συνδέει UART, packet decoder, step generator, laser και ενδείξεις.
uart_rx.v / uart_tx.v	Σειριακή επικοινωνία	Υλοποιούν τη λήψη πακέτων και την αποστολή ACK.
packet_decoder.v	Αποκωδικοποίηση	Μετατρέπει τα εισερχόμενα bytes σε πεδία κίνησης.
step_gen.v	Παραγωγή παλμών	Παράγει STEP/DIR με βάση τα steps και το feed rate.
laser_ctrl.v	Έλεγχος laser	Ενεργοποιεί ή απενεργοποιεί το laser ανάλογα με τον τύπο κίνησης.
output_files/*.sof	Bitstream	Αρχείο προγραμματισμού που φορτώνεται στο FPGA.

Με βάση αυτή την αρχιτεκτονική, να διαχωρίζεται ο ρόλος του κάθε συστήματος: ο υπολογιστής δημιουργεί και αποστέλλει τα πακέτα, ενώ το FPGA εκτελεί με σταθερό χρονισμό τις κινήσεις της μηχανής. Αυτός ο διαχωρισμός αποτελεί τη βάση για την πειραματική υλοποίηση και τον έλεγχο που παρουσιάζονται στα επόμενα κεφάλαια.

5.2 Σχεδίαση modules σε Verilog

5.2.1 Δέκτης UART (uart_rx)

Η μονάδα uart_rx αποτελεί το πρώτο στάδιο της σειριακής επικοινωνίας από τον υπολογιστή προς το FPGA. Η είσοδος rx συνδέεται με την έξοδο TX του CP2102 και μεταφέρει τα bytes που στέλνει το πρόγραμμα Python. Ο ρόλος της μονάδας είναι να μετατρέψει το σειριακό σήμα UART σε παράλληλη λέξη των 8 bit, ώστε στη συνέχεια να μπορεί να την επεξεργαστεί η μονάδα αποκωδικοποίησης πακέτων.

Η υλοποίηση χρησιμοποιεί τυπική μορφή UART 8N1, δηλαδή ένα start bit, οκτώ bit δεδομένων και ένα stop bit, χωρίς bit ισοτιμίας. Σε κατάσταση ηρεμίας η γραμμή rx βρίσκεται σε λογικό 1. Η μετάβαση της γραμμής σε λογικό 0 δηλώνει την έναρξη ενός νέου byte. Τα δεδομένα λαμβάνονται με σειρά LSB first, όπως προβλέπει η τυπική λειτουργία UART.

Το ρολόι του FPGA είναι 50 MHz και η ταχύτητα επικοινωνίας έχει οριστεί στα 115200 baud. Επομένως, η διάρκεια ενός bit αντιστοιχεί περίπου σε 50.000.000 / 115200 = 434 κύκλους ρολογιού. Για τον λόγο αυτό στον κώδικα χρησιμοποιείται η σταθερά FULL_BIT = 434, ενώ η σταθερά HALF_BIT = 217 χρησιμοποιείται για τη δειγματοληψία στο μέσο του start bit. Η δειγματοληψία στο μέσο του bit μειώνει την πιθανότητα να διαβαστεί λάθος τιμή λόγω της μετάβασης του σήματος.

5.2.1.1 Συγχρονισμός εισόδου και χρονισμός

Το σήμα rx προέρχεται από εξωτερικό κύκλωμα και δεν είναι συγχρονισμένο με το ρολόι του FPGA. Για τον λόγο αυτό περνά πρώτα από δύο καταχωρητές, τους rx_s1 και rx_s2. Η τεχνική αυτή χρησιμοποιείται για να μειωθεί ο κίνδυνος μετασταθερότητας [2] ετσι ώστε η μηχανή καταστάσεων να επεξεργάζεται μία συγχρονισμένη εκδοχή της εισόδου.

Ο μετρητής count χρησιμοποιείται για τη μέτρηση των κύκλων ρολογιού που αντιστοιχούν σε κάθε bit. Στο start bit η μονάδα περιμένει μισό χρόνο bit και ελέγχει αν η γραμμή παραμένει στο 0. Αν η γραμμή έχει επιστρέψει στο 1, η μετάβαση θεωρείται θόρυβος ή ψευδής εκκίνηση και ο δέκτης επιστρέφει στην κατάσταση αναμονής.

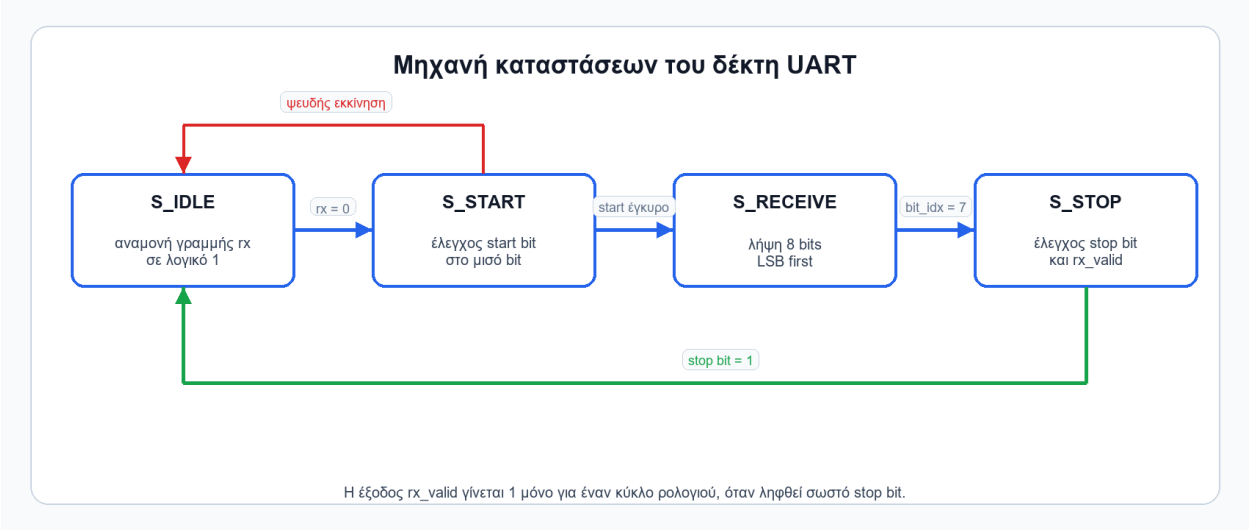
5.2.1.2 Σήματα ελέγχου καταστάσεων

Η λήψη κάθε byte ελέγχεται από τέσσερα σήματα. Η μηχανή ξεκινά από την κατάσταση S_IDLE, ανιχνεύει το start bit, λαμβάνει τα οκτώ bit δεδομένων και στο τέλος ελέγχει το stop bit. Μόνο όταν το stop bit είναι έγκυρο παράγεται παλμός rx_valid.

Πίνακας 5.2: Σήματα ελέγχου δέκτη

Κατάσταση	Ρόλος	Μετάβαση
S_IDLE	Αναμονή για νέο byte. Η γραμμή rx πρέπει να βρίσκεται κανονικά σε λογικό 1.	Όταν rx_s2 = 0, θεωρείται πιθανό start bit και η FSM περνά στην S_START.
S_START	Επιβεβαίωση του start bit στο μέσο της διάρκειάς του.	Αν το rx_s2 παραμένει 0, αρχίζει η λήψη δεδομένων. Αν είναι 1, επιστροφή σε S_IDLE.
S_RECEIVE	Δειγματοληψία των 8 bit δεδομένων ανά FULL_BIT κύκλους.	Μετά τη λήψη του bit 7, η FSM περνά στην S_STOP.

S_STOP	Έλεγχος του stop bit και ολοκλήρωση του byte.	Αν το stop bit είναι 1, το rx_shift αντιγράφεται στο rx_data και το rx_valid γίνεται 1 για έναν κύκλο.



Σχήμα 5.2: *Μηχανή καταστάσεων του δέκτη UART.*

5.2.1.3 Παραγωγή του rx_valid

Η έξοδος rx_valid δεν παραμένει μόνιμα ενεργή. Γίνεται 1 μόνο για έναν κύκλο του ρολογιού, ακριβώς όταν έχει ληφθεί σωστά ένα byte. Με αυτόν τον τρόπο η επόμενη μονάδα, δηλαδή ο packet_decoder, μπορεί να αναγνωρίσει με σαφήνεια πότε υπάρχει νέο διαθέσιμο byte στην έξοδο rx_data. Στον επόμενο κύκλο το rx_valid μηδενίζεται αυτόματα, ώστε να μην διαβαστεί το ίδιο byte περισσότερες από μία φορές.

Αν το stop bit δεν είναι λογικό 1, τότε το byte δεν θεωρείται έγκυρο και δεν παράγεται rx_valid. Η επιλογή αυτή λειτουργεί ως βασικός έλεγχος σφάλματος του πλαισίου UART, χωρίς να απαιτείται πιο σύνθετος μηχανισμός ανίχνευσης σφαλμάτων.

5.2.1.4 Βασικό απόσπασμα κώδικα

Το παρακάτω απόσπασμα δείχνει τις βασικές σταθερές χρονισμού και τις καταστάσεις που χρησιμοποιεί ο δέκτης. Οι τιμές αυτές συνδέουν άμεσα τη συχνότητα ρολογιού των 50 MHz με την επιλεγμένη ταχύτητα UART των 115200 baud.

```
localparam FULL_BIT = 10'd434;
localparam HALF_BIT = 10'd217;

localparam S_IDLE    = 2'd0;
localparam S_START   = 2'd1;
localparam S_RECEIVE = 2'd2;
localparam S_STOP    = 2'd3;

reg [1:0] state;
reg [9:0] count;
reg [2:0] bit_idx;
reg [7:0] rx_shift;
reg       rx_s1, rx_s2;
```

5.2.1.5 Σύνδεση με την υπόλοιπη υλοποίηση

Στο συνολικό σύστημα, ο uart_rx δεν γνωρίζει τη δομή των πακέτων που στέλνει ο υπολογιστής. Η ευθύνη του περιορίζεται στη σωστή λήψη ανεξάρτητων bytes. Κάθε φορά που παράγεται rx_valid, ο packet_decoder διαβάζει το rx_data και τοποθετεί το byte στη σωστή θέση του πακέτου. Έτσι διαχωρίζεται καθαρά η φυσική λήψη UART από την ερμηνεία των εντολών κίνησης.

Η συγκεκριμένη σχεδίαση είναι κατάλληλη για την εφαρμογή, επειδή η επικοινωνία μέσω CP2102 στα 115200 baud έχει αρκετά χαμηλή ταχύτητα σε σχέση με το ρολόι του FPGA. Το FPGA έχει έτσι αρκετούς κύκλους ρολογιού για να εντοπίσει την αρχή του byte, να δειγματοληπτήσει τα δεδομένα και να ενημερώσει αξιόπιστα την επόμενη μονάδα της αλυσίδας επικοινωνίας.

5.2.2 Πομπός UART (uart_tx)

Η μονάδα uart_tx υλοποιεί τον πομπό της σειριακής επικοινωνίας UART. Σε αντίθεση με τον uart_rx, ο οποίος λαμβάνει bytes από τον υπολογιστή, ο uart_tx μετατρέπει μία παράλληλη λέξη 8 bit σε σειριακό σήμα εξόδου tx. Το σήμα αυτό οδηγείται στην είσοδο RX του CP2102, ώστε ο υπολογιστής να μπορεί να λάβει απάντηση από το FPGA.

Στην παρούσα υλοποίηση ο πομπός χρησιμοποιείται κυρίως για την αποστολή του byte επιβεβαίωσης ACK. Το byte αυτό έχει τιμή 0xAA και αποστέλλεται όταν ολοκληρωθεί μία κίνηση. Με αυτόν τον τρόπο το πρόγραμμα Python δεν στέλνει ανεξέλεγκτα όλα τα πακέτα, αλλά περιμένει να ενημερωθεί ότι το FPGA έχει τελειώσει την προηγούμενη εντολή.

Όπως και ο δέκτης, ο πομπός λειτουργεί με ρολόι 50 MHz και ταχύτητα 115200 baud. Η διάρκεια ενός bit προκύπτει από τη σχέση 50.000.000 / 115.200 και αντιστοιχεί σε 434 κύκλους ρολογιού. Για κάθε bit που αποστέλλεται, η έξοδος tx διατηρείται σταθερή για έναν πλήρη χρόνο bit.

Σχήμα 5.3: *Μορφή πλαισίου αποστολής UART 8N1.*

5.2.2.1 Σήματα εισόδου και εξόδου

Η μονάδα δέχεται το byte προς αποστολή μέσω του tx_data και ξεκινά τη μετάδοση όταν το tx_start γίνει 1 για έναν κύκλο ρολογιού. Κατά τη διάρκεια της μετάδοσης το tx_busy παραμένει ενεργό, ώστε η ανώτερη λογική να γνωρίζει ότι ο πομπός δεν είναι διαθέσιμος για νέο byte. Η έξοδος tx είναι η πραγματική σειριακή γραμμή UART.

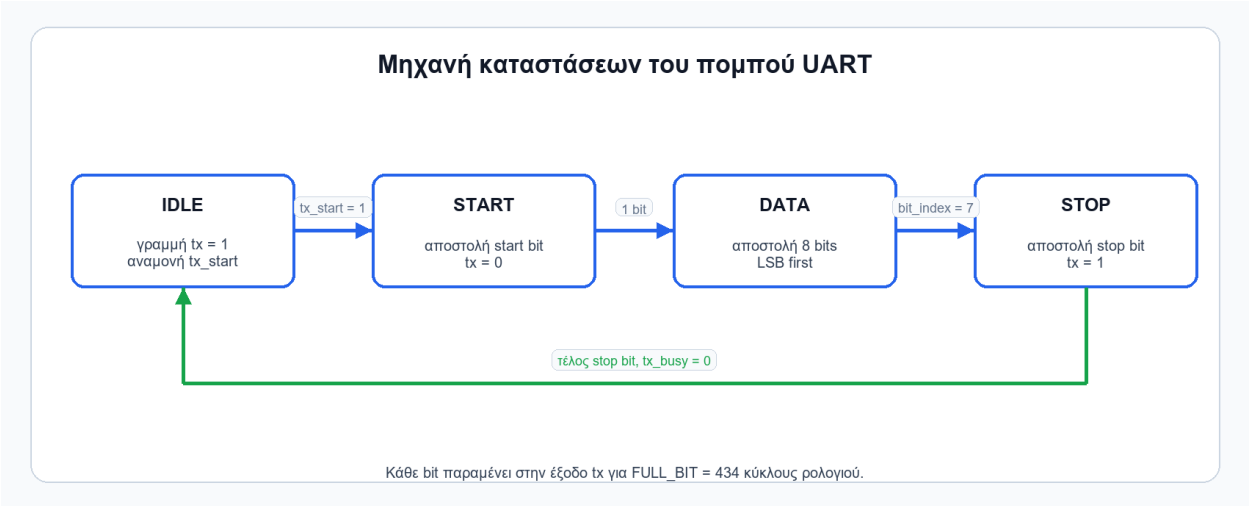
Πίνακας 5.3: Σήματα ελέγχου πομπού

Σήμα	Κατεύθυνση	Περιγραφή
clk	είσοδος	Ρολόι συστήματος 50 MHz.
rst	είσοδος	Σύγχρονο reset, ενεργό σε λογικό 1.
tx_data[7:0]	είσοδος	Το byte που πρόκειται να αποσταλεί σειριακά.
tx_start	είσοδος	Παλμός ενός κύκλου που ξεκινά νέα μετάδοση.
tx_busy	έξοδος	Δείχνει ότι ο πομπός βρίσκεται σε εξέλιξη μετάδοσης.
tx	έξοδος	Σειριακό σήμα UART προς το CP2102 RX.

5.2.2.2 Σήματα καταστάσεων

Η λειτουργία του πομπού οργανώνεται σε τέσσερις καταστάσεις: IDLE, START, DATA και STOP. Η κατάσταση IDLE κρατά τη γραμμή tx σε λογικό 1, που είναι η κανονική κατάσταση ηρεμίας του UART. Όταν εμφανιστεί παλμός tx_start, το byte αποθηκεύεται στον εσωτερικό καταχωρητή tx_shift και η μονάδα ξεκινά την αποστολή.

Κατάσταση	Ρόλος	Μετάβαση
IDLE	Η γραμμή tx βρίσκεται σε λογικό 1 και το tx_busy είναι 0.	Με παλμό tx_start αποθηκεύεται το tx_data και η FSM περνά στην START.
START	Αποστέλλεται το start bit, δηλαδή λογικό 0, για έναν χρόνο bit.	Μετά από FULL_BIT κύκλους η FSM περνά στην DATA.
DATA	Αποστέλλονται τα οκτώ bit του tx_shift με σειρά LSB first.	Μετά το bit 7 η FSM περνά στην STOP.
STOP	Αποστέλλεται stop bit, δηλαδή λογικό 1, για έναν χρόνο bit.	Μετά το stop bit το tx_busy γίνεται 0 και η FSM επιστρέφει στην IDLE.



Σχήμα 5.4: *Μηχανή καταστάσεων του πομπού UART.*

5.2.2.3 Χρονισμός μετάδοσης

Ο μετρητής clk_count μετρά τους κύκλους του ρολογιού για κάθε bit. Όταν φτάσει στην τιμή FULL_BIT - 1, ολοκληρώνεται η αποστολή του τρέχοντος bit και ο μετρητής μηδενίζεται. Στην κατάσταση DATA χρησιμοποιείται επίσης ο μετρητής bit_index, ο οποίος επιλέγει ποιο bit του tx_shift οδηγείται στην έξοδο tx.

Η αποστολή γίνεται με σειρά LSB first. Αυτό σημαίνει ότι πρώτο αποστέλλεται το λιγότερο σημαντικό bit του byte και τελευταίο το περισσότερο σημαντικό. Η επιλογή αυτή ακολουθεί την τυπική μορφή UART και είναι συμβατή με τον τρόπο που ο υπολογιστής και το CP2102 λαμβάνουν το σειριακό byte.

5.2.2.4 Σύνδεση με τη λογική ACK

Στο top module, η είσοδος tx_data του uart_tx συνδέεται σταθερά με την τιμή 8'hAA. Η τιμή αυτή είναι το ACK byte που περιμένει το πρόγραμμα Python. Το tx_start ενεργοποιείται όταν ανιχνευθεί πτώση του σήματος busy, δηλαδή όταν η μονάδα παραγωγής παλμών έχει ολοκληρώσει την τρέχουσα κίνηση. Αν ο πομπός δεν είναι ήδη απασχολημένος, ξεκινά η αποστολή του ACK προς τον υπολογιστή.

```
uart_tx uart_tx_inst (
    .clk      (clk),
    .rst      (rst),
    .tx_data  (8'hAA),
    .tx_start (tx_start),
    .tx_busy  (tx_busy),
    .tx       (tx)
);

// Send ACK when motion is complete
if (busy_prev && !busy) begin
    if (!tx_busy)
        tx_start <= 1'b1;
end
```

5.2.2.5 Βασικό απόσπασμα κώδικα

Οι βασικές σταθερές και οι καταχωρητές του πομπού φαίνονται στο ακόλουθο απόσπασμα. Το FULL_BIT καθορίζει τη διάρκεια κάθε bit, ενώ το tx_shift κρατά το byte που βρίσκεται σε εξέλιξη μετάδοσης.

```
localparam CLK_FREQ  = 50_000_000;
localparam BAUD_RATE = 115_200;
localparam FULL_BIT  = CLK_FREQ / BAUD_RATE;  // 434 cycles

localparam IDLE  = 2'd0;
localparam START = 2'd1;
localparam DATA  = 2'd2;
localparam STOP  = 2'd3;

reg [1:0] state;
reg [8:0] clk_count;
reg [2:0] bit_index;
reg [7:0] tx_shift;
```

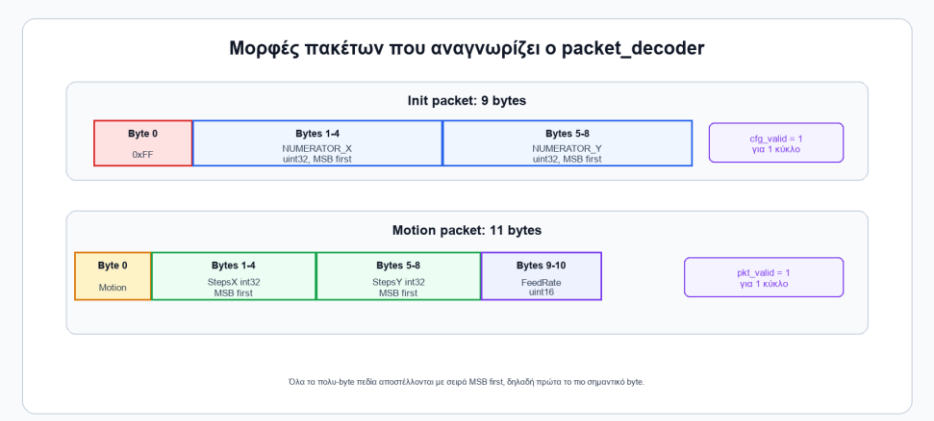
Το uart_tx δεν συμμετέχει στην αποκωδικοποίηση των πακέτων κίνησης. Η λειτουργία του είναι να παρέχει έναν απλό και αξιόπιστο μηχανισμό επιστροφής πληροφορίας προς τον υπολογιστή. Με την αποστολή του ACK μετά από κάθε ολοκληρωμένη κίνηση, η επικοινωνία γίνεται συγχρονισμένη: ο υπολογιστής στέλνει νέο πακέτο μόνο όταν λάβει επιβεβαίωση ότι το FPGA είναι έτοιμο.

5.2.3 Αποκωδικοποιητής πακέτων (packet_decoder)

Η μονάδα packet_decoder βρίσκεται αμέσως μετά τον δέκτη UART. Ο uart_rx παραδίδει κάθε byte σε παράλληλη μορφή μέσω του rx_data και ενημερώνει τη μονάδα με έναν παλμό rx_valid. Ο αποκωδικοποιητής δεν ασχολείται με τη φυσική λήψη της σειριακής γραμμής, αλλά με την ομαδοποίηση των bytes σε ολοκληρωμένα πακέτα εντολών.

Η ανάγκη για ξεχωριστή μονάδα αποκωδικοποίησης προκύπτει επειδή η Python δεν στέλνει απλό κείμενο G-code προς το FPGA. Αντίθετα, στέλνει δυαδικά πακέτα σταθερής μορφής. Με αυτόν τον τρόπο μειώνεται ο όγκος των δεδομένων και το FPGA λαμβάνει απευθείας τις αριθμητικές τιμές που χρειάζεται για την εκτέλεση της κίνησης.

Ο αποκωδικοποιητής αναγνωρίζει δύο τύπους πακέτων. Το πρώτο είναι το πακέτο αρχικοποίησης, το οποίο ξεκινά με το byte 0xFF και μεταφέρει τις τιμές NUMERATOR_X και NUMERATOR_Y. Το δεύτερο είναι το πακέτο κίνησης, το οποίο μεταφέρει τον τύπο κίνησης, τα βήματα των αξόνων X και Y και την τιμή πρόωσης.



Σχήμα 5.5: *Μορφές πακέτων που αποκωδικοποιεί η μονάδα packet_decoder.*

5.2.3.1 Σήματα εισόδου και εξόδου

Τα βασικά σήματα εισόδου είναι το rx_data και το rx_valid. Το rx_valid λειτουργεί ως ένδειξη ότι το rx_data περιέχει νέο έγκυρο byte. Στην έξοδο, η μονάδα παράγει είτε σήματα κίνησης, όταν ολοκληρωθεί motion packet, είτε σήματα αρχικοποίησης, όταν ολοκληρωθεί init packet.

Πίνακας 5.4: Σήματα εισόδου εξόδου αποκωδικοποιητή πακέτων

Σήμα	Κατεύθυνση	Περιγραφή
rx_data[7:0]	είσοδος	Το byte που παραδόθηκε από τον δέκτη UART.
rx_valid	είσοδος	Παλμός ενός κύκλου που δείχνει ότι υπάρχει νέο byte.
motion[1:0]	έξοδος	Κωδικός κίνησης, όπως G0, G1, G2 ή G3.
steps_x[31:0]	έξοδος	Προσημασμένος αριθμός βημάτων για τον άξονα X.
steps_y[31:0]	έξοδος	Προσημασμένος αριθμός βημάτων για τον άξονα Y.
feed_rate[15:0]	έξοδος	Τιμή πρόωσης που χρησιμοποιείται από τη μονάδα παραγωγής παλμών.
pkt_valid	έξοδος	Παλμός ενός κύκλου όταν ολοκληρωθεί motion packet.
numerator_x/y	έξοδος	Τιμές χρονισμού αξόνων που προκύπτουν από init packet.
cfg_valid	έξοδος	Παλμός ενός κύκλου όταν ολοκληρωθεί init packet.

5.2.3.2 Πακέτο αρχικοποίησης

Το πακέτο αρχικοποίησης έχει μήκος 9 bytes. Το πρώτο byte είναι πάντα 0xFF και χρησιμοποιείται ως δείκτης ότι το πακέτο δεν είναι εντολή κίνησης αλλά πακέτο παραμετροποίησης. Τα επόμενα τέσσερα bytes σχηματίζουν το NUMERATOR_X και τα τελευταία τέσσερα bytes σχηματίζουν το NUMERATOR_Y.

Οι τιμές αυτές στέλνονται με σειρά MSB first. Αυτό σημαίνει ότι πρώτα φτάνει το πιο σημαντικό byte και τελευταίο το λιγότερο σημαντικό byte. Όταν ληφθούν και τα 9 bytes, ο αποκωδικοποιητής ενημερώνει τις εξόδους numerator_x και numerator_y και ενεργοποιεί το cfg_valid για έναν κύκλο ρολογιού.

5.2.3.3 Πακέτο κίνησης

Το πακέτο κίνησης έχει μήκος 11 bytes. Το πρώτο byte περιέχει τον τύπο κίνησης. Στον κώδικα χρησιμοποιούνται τα δύο λιγότερο σημαντικά bit του πρώτου byte (b0[1:0]) για την έξοδο motion. Τα επόμενα τέσσερα bytes σχηματίζουν το steps_x, τα επόμενα τέσσερα το steps_y και τα δύο τελευταία το feed_rate.

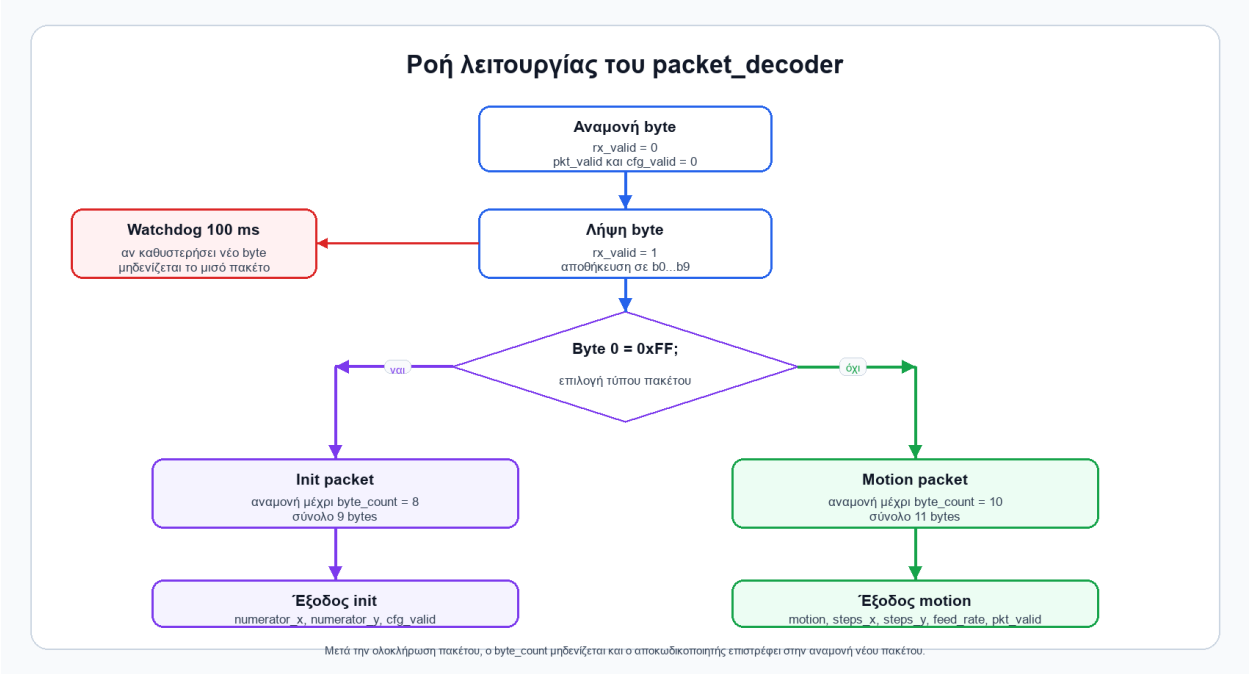
Τα πεδία steps_x και steps_y είναι 32-bit προσημασμένες τιμές. Το πρόσημο δηλώνει την κατεύθυνση κίνησης κάθε άξονα, ενώ η απόλυτη τιμή δηλώνει το πλήθος των παλμών. Το feed_rate είναι 16-bit μη προσημασμένη τιμή και χρησιμοποιείται από τη μονάδα step_gen για τον υπολογισμό του χρονισμού των παλμών.

Πίνακας 5.5: Δύο τύποι πακέτων

Τύπος	Μήκος	Byte 0	Κύρια πεδία	Παλμός ολοκλήρωσης
Init	9 bytes	0xFF	NUMERATOR_X, NUMERATOR_Y	cfg_valid
Motion	11 bytes	motion	steps_x, steps_y, feed_rate	pkt_valid

5.2.3.4 Μέτρηση bytes και επιλογή τύπου πακέτου

Η θέση του κάθε byte μέσα στο πακέτο παρακολουθείται από τον καταχωρητή byte_count. Κάθε φορά που λαμβάνεται νέο byte, αυτό αποθηκεύεται σε έναν από τους καταχωρητές b0 έως b9. Στο πρώτο byte του πακέτου ο αποκωδικοποιητής αποφασίζει τον τύπο του πακέτου. Αν το πρώτο byte είναι ίσο με 0xFF, τότε το is_cfg_pkt γίνεται 1 και η μονάδα περιμένει συνολικά 9 bytes. Αν το πρώτο byte είναι διαφορετικό από 0xFF, τότε το πακέτο θεωρείται motion packet και η μονάδα περιμένει συνολικά 11 bytes.



Σχήμα 5.6: *Ροή λειτουργίας του αποκωδικοποιητή πακέτων.*

5.2.3.5 Watchdog χρονικού ορίου

Για λόγους αξιοπιστίας, ο αποκωδικοποιητής περιλαμβάνει watchdog χρονικού ορίου. Όταν ξεκινήσει η λήψη ενός πακέτου, ενεργοποιείται ο μετρητής timeout_cnt. Αν για περίπου 100 ms δεν ληφθεί νέο byte, ο αποκωδικοποιητής θεωρεί ότι το πακέτο έμεινε ημιτελές ή ότι προέκυψε σφάλμα επικοινωνίας. Τότε μηδενίζεται το byte_count και η μονάδα επιστρέφει σε κατάσταση αναμονής νέου πακέτου.

Η τιμή TIMEOUT_CYCLES είναι 5.000.000 κύκλοι. Με ρολόι 50 MHz, αυτό αντιστοιχεί σε 5.000.000 / 50.000.000 = 0,1 s, δηλαδή 100 ms. Το χρονικό αυτό όριο είναι αρκετά μεγαλύτερο από τον χρόνο μετάδοσης ενός byte στα 115200 baud, αλλά βοηθά στην ανάκαμψη σε περίπτωση που χαθεί κάποιο byte.

5.2.3.6 Βασικό απόσπασμα κώδικα

Το παρακάτω απόσπασμα δείχνει τις βασικές σταθερές και τους καταχωρητές που χρησιμοποιούνται για τη συγκέντρωση των bytes, την αναγνώριση του init packet και τον watchdog μηχανισμό.

```
localparam TIMEOUT_CYCLES = 23'd5_000_000;
localparam CFG_MARKER     = 8'hFF;

reg [3:0]  byte_count;
reg [22:0] timeout_cnt;
reg        timeout_active;
reg        is_cfg_pkt;

reg [7:0] b0,b1,b2,b3,b4,b5,b6,b7,b8,b9;
```

Η ολοκλήρωση των δύο πακέτων γίνεται σε διαφορετικό πλήθος bytes. Στο init packet η ολοκλήρωση γίνεται όταν byte_count = 8, ενώ στο motion packet όταν byte_count = 10.

```
if (is_cfg_pkt && byte_count == 4'd8) begin
    numerator_x <= { b1, b2, b3, b4 };
    numerator_y <= { b5, b6, b7, rx_data };
    cfg_valid   <= 1'b1;
    byte_count  <= 4'd0;
end
else if (!is_cfg_pkt && byte_count == 4'd10) begin
    motion    <= b0[1:0];
    steps_x   <= { b1, b2, b3, b4 };
    steps_y   <= { b5, b6, b7, b8 };
    feed_rate <= { b9, rx_data };
    pkt_valid <= 1'b1;
    byte_count <= 4'd0;
end
```

5.2.3.7 Σύνδεση με την υπόλοιπη αρχιτεκτονική

Στη συνολική αρχιτεκτονική, τα πεδία steps_x, steps_y, feed_rate και pkt_valid χρησιμοποιούνται από τη μονάδα παραγωγής παλμών. Το motion χρησιμοποιείται επίσης από τη μονάδα ελέγχου του laser, ώστε η ενεργοποίηση του laser να εξαρτάται από τον τύπο της κίνησης. Τα σήματα numerator_x, numerator_y και cfg_valid προορίζονται για τη μεταφορά των παραμέτρων χρονισμού προς τη μονάδα παραγωγής παλμών.

Με αυτόν τον διαχωρισμό, η επικοινωνία γίνεται καθαρή και επεκτάσιμη. Ο uart_rx αναλαμβάνει μόνο τη λήψη bytes, ο packet_decoder μετατρέπει τα bytes σε πεδία πακέτου και οι επόμενες μονάδες χρησιμοποιούν πλέον αριθμητικά σήματα έτοιμα για εκτέλεση κίνησης.

5.2.4 Γεννήτρια παλμών (step_gen)

Η μονάδα step_gen αποτελεί το τμήμα της υλοποίησης που μετατρέπει τα αριθμητικά δεδομένα ενός motion packet σε πραγματικά σήματα οδήγησης για τους οδηγούς των βηματικών κινητήρων. Δέχεται τον αριθμό βημάτων για τους άξονες X και Y, την τιμή πρόωσης και τον παλμό εγκυρότητας pkt_valid, και παράγει τα σήματα DIR και STEP για κάθε άξονα.

Η λειτουργία της μονάδας είναι χρονικά κρίσιμη, επειδή τα σήματα STEP πρέπει να παράγονται με σταθερή περίοδο και με ελάχιστο πλάτος παλμού που ικανοποιεί τις απαιτήσεις του οδηγού DRV8825. Για τον λόγο αυτό η παραγωγή των παλμών γίνεται στο FPGA και όχι στον υπολογιστή, ώστε ο χρονισμός να είναι ντετερμινιστικός.

Η μονάδα υποστηρίζει συντονισμένη κίνηση δύο αξόνων. Αυτό σημαίνει ότι οι άξονες X και Y μπορούν να κινούνται ταυτόχρονα, αλλά με διαφορετική συχνότητα παλμών, ώστε να ολοκληρώνουν την κίνηση περίπου την ίδια χρονική στιγμή. Έτσι η κεφαλή ακολουθεί τη ζητούμενη γεωμετρική διαδρομή αντί να κινείται πρώτα στον έναν άξονα και μετά στον άλλον.

Πίνακας 5.6: Σήματα εισόδου εξόδου

Σήμα	Κατεύθυνση	Περιγραφή
steps_x, steps_y	είσοδος	Προσημασμένος αριθμός βημάτων για τους άξονες X και Y.
feed_rate	είσοδος	Συνιστώσα πρόωσης που έχει υπολογιστεί ώστε η συνισταμένη ταχύτητα να αντιστοιχεί στο G-code F.
pkt_valid	είσοδος	Παλμός ενός κύκλου που δηλώνει ότι υπάρχει νέο motion packet.
numerator_x/y	είσοδος	Παράμετροι μετατροπής πρόωσης σε κύκλους ρολογιού ανά βήμα.
cfg_valid	είσοδος	Ενημερώνει τα numerator_x/y όταν η μονάδα είναι σε IDLE.
dir_x, dir_y	έξοδος	Κατεύθυνση περιστροφής κάθε βηματικού κινητήρα.
step_x, step_y	έξοδος	Παλμοί STEP προς τους οδηγούς DRV8825.
busy	έξοδος	Δείχνει ότι βρίσκεται σε εξέλιξη κίνηση.

5.2.4.1 Παράμετροι χρονισμού

Η διάρκεια του παλμού STEP έχει οριστεί σε 100 κύκλους ρολογιού. Με ρολόι 50 MHz, ένας κύκλος διαρκεί 20 ns, άρα 100 κύκλοι αντιστοιχούν σε 2 μs. Η τιμή αυτή καλύπτει την ελάχιστη απαίτηση του DRV8825, η οποία είναι περίπου 1,9 μs. Επιπλέον, μετά την αλλαγή των σημάτων DIR εφαρμόζεται χρόνος αναμονής 200 κύκλων, δηλαδή περίπου 4 μs, πριν παραχθεί ο πρώτος παλμός STEP[4].

Οι παράμετροι numerator_x και numerator_y επιτρέπουν στη μονάδα να υπολογίζει την περίοδο των παλμών με βάση τα πραγματικά βήματα ανά χιλιοστό του κάθε άξονα [3]. Η γενική σχέση είναι NUMERATOR = CLK_HZ · 60 / steps_per_mm. Αν δεν έχει ληφθεί init packet, χρησιμοποιείται η προεπιλεγμένη τιμή 18.750.000, που αντιστοιχεί σε 160 steps/mm.

Σχήμα 5.7: *Χρονισμός σημάτων DIR και STEP στη γεννήτρια παλμών.*

5.2.4.2 Σήματα καταστάσεων

Η λειτουργία της μονάδας οργανώνεται σε πέντε καταστάσεις: IDLE, SETUP, DIR_WAIT, RUN και DONE. Στην IDLE η μονάδα περιμένει νέο pkt_valid. Στην SETUP υπολογίζεται η κατεύθυνση και η απόλυτη τιμή των βημάτων. Στην DIR_WAIT εφαρμόζεται χρόνος σταθεροποίησης της κατεύθυνσης και υπολογίζονται οι περίοδοι των παλμών. Στην RUN παράγονται οι παλμοί STEP και στην DONE ολοκληρώνεται η κίνηση.

Πίνακας 5.7: Σήματα καταστάσεων γεννήτριας παλμών

Κατάσταση	Ρόλος	Μετάβαση
IDLE	Αναμονή για νέο motion packet. Το busy είναι 0.	Με pkt_valid γίνεται busy = 1 και μετάβαση σε SETUP.
SETUP	Υπολογισμός DIR και απόλυτων τιμών abs_x, abs_y.	Μετάβαση σε DIR_WAIT.
DIR_WAIT	Αναμονή DIR_SETTLE και υπολογισμός period_x/period_y.	Αν δεν υπάρχουν βήματα, μετάβαση σε DONE, αλλιώς σε RUN.
RUN	Παραγωγή παλμών STEP μέχρι να μηδενιστούν rem_x και rem_y.	Όταν ολοκληρωθούν και οι δύο άξονες, μετάβαση σε DONE.
DONE	Μηδενισμός busy και STEP εξόδων.	Επιστροφή σε IDLE.

Σχήμα 5.8: *Μηχανή καταστάσεων της μονάδας step_gen.*

5.2.4.3 Υπολογισμός κατεύθυνσης και πλήθους βημάτων

Τα steps_x και steps_y είναι προσημασμένες τιμές. Το πρόσημο χρησιμοποιείται για τον καθορισμό της κατεύθυνσης, ενώ η απόλυτη τιμή χρησιμοποιείται ως πλήθος παλμών που πρέπει να παραχθούν. Στον κώδικα, αν το πιο σημαντικό bit είναι 0, η κατεύθυνση θεωρείται θετική. Αν είναι 1, η τιμή είναι αρνητική.

```
dir_x <= (steps_x[31] == 1'b0) ? 1'b1 : 1'b0;
dir_y <= (steps_y[31] == 1'b0) ? 1'b1 : 1'b0;

abs_x <= steps_x[31] ? (~steps_x + 32'd1) : steps_x;
abs_y <= steps_y[31] ? (~steps_y + 32'd1) : steps_y;
```

5.2.4.4 Υπολογισμός περιόδου παλμών

Στην κατάσταση DIR_WAIT υπολογίζονται οι περίοδοι period_x και period_y. Η περίοδος εκφράζει πόσοι κύκλοι ρολογιού μεσολαβούν ανάμεσα σε δύο διαδοχικούς παλμούς STEP. Μικρότερη περίοδος σημαίνει μεγαλύτερη ταχύτητα άξονα, ενώ μεγαλύτερη περίοδος σημαίνει μικρότερη ταχύτητα.

Σημαντικό είναι ότι το feed_rate που φτάνει στο FPGA δεν είναι απλώς το αρχικό F του G-code. Στο στάδιο μετατροπής G-code σε βήματα, το F χρησιμοποιείται ως συνισταμένη ταχύτητα της κεφαλής. Έτσι, όταν ο step_gen προσαρμόζει την ταχύτητα των αξόνων, και η τελική κίνηση αντιστοιχεί στη ζητούμενη συνισταμένη ταχύτητα.

```
if (abs_x >= abs_y) begin
    period_x <= num_x / {16'd0, feed_rate};
    if (abs_y == 32'd0)
        period_y <= 32'd0;
    else
        period_y <= (num_x / {16'd0, feed_rate}) * abs_x / abs_y;
end
else begin
    period_y <= num_y / {16'd0, feed_rate};
    if (abs_x == 32'd0)
        period_x <= 32'd0;
    else
        period_x <= (num_y / {16'd0, feed_rate}) * abs_y / abs_x;
end
```

5.2.4.5 Παραγωγή παλμών STEP

Στην κατάσταση RUN κάθε άξονας διαθέτει δικό του μετρητή χρόνου, cnt_x και cnt_y. Όταν ο μετρητής βρίσκεται στην αρχή της περιόδου, το αντίστοιχο STEP γίνεται 1. Μετά από PULSE_HIGH κύκλους, το STEP μηδενίζεται. Όταν ο μετρητής φτάσει στο τέλος της περιόδου, μηδενίζεται και αφαιρείται ένα βήμα από το rem_x ή rem_y.

Η διαδικασία αυτή συνεχίζεται μέχρι να μηδενιστούν τα εναπομένοντα βήματα και στους δύο άξονες. Τότε η μονάδα περνά στην κατάσταση DONE, μηδενίζει τα STEP σήματα και κατεβάζει το busy. Η πτώση του busy χρησιμοποιείται από το top module για να σταλεί ACK προς τον υπολογιστή μέσω του uart_tx.

5.2.4.6 Βασικές σταθερές και καταχωρητές

```
localparam PULSE_HIGH = 32'd100;   // 2 us HIGH
localparam DIR_SETTLE = 9'd200;    // 4 us before first STEP
localparam NUMERATOR_DEFAULT = 32'd18_750_000;

localparam IDLE     = 3'd0;
localparam SETUP    = 3'd1;
localparam DIR_WAIT = 3'd2;
localparam RUN      = 3'd3;
localparam DONE     = 3'd4;
```

Η step_gen είναι η μονάδα που μετατρέπει την αριθμητική περιγραφή της κίνησης σε πραγματικό ηλεκτρικό χρονισμό. Οι προηγούμενες μονάδες της αλυσίδας λαμβάνουν και αποκωδικοποιούν τα πακέτα, όμως η συγκεκριμένη μονάδα είναι αυτή που καθορίζει πότε θα παραχθεί κάθε παλμός STEP και ποια κατεύθυνση θα έχει κάθε άξονας.

Με τη χρήση ανεξάρτητων μετρητών για τους δύο άξονες και με προσαρμογή της περιόδου του μη κυρίαρχου άξονα, η υλοποίηση μπορεί να εκτελέσει συντονισμένες κινήσεις X/Y χωρίς τη συμμετοχή επεξεργαστή. Αυτό είναι ένα από τα βασικά πλεονεκτήματα της υλοποίησης σε FPGA.

5.2.5 Ελεγκτής laser (laser_ctrl)

Η μονάδα laser_ctrl ελέγχει την έξοδο ενεργοποίησης του laser. Η έξοδος laser οδηγεί το TTL enable του laser και είναι ενεργή σε λογικό 1. Η συγκεκριμένη μονάδα δεν ρυθμίζει την ισχύ του laser και δεν παράγει PWM. Ο ρόλος της είναι να αποφασίζει αν το laser πρέπει να είναι αναμμένο ή σβηστό κατά την εκτέλεση κάθε κίνησης.

Η απόφαση βασίζεται σε δύο πληροφορίες. Η πρώτη είναι ο τύπος της κίνησης, δηλαδή το motion που προέρχεται από τον packet_decoder. Η δεύτερη είναι το σήμα busy που προέρχεται από τη γεννήτρια παλμών step_gen και δείχνει ότι η κίνηση βρίσκεται σε εξέλιξη. Το laser ανάβει μόνο όταν η κίνηση είναι κίνηση χάραξης.

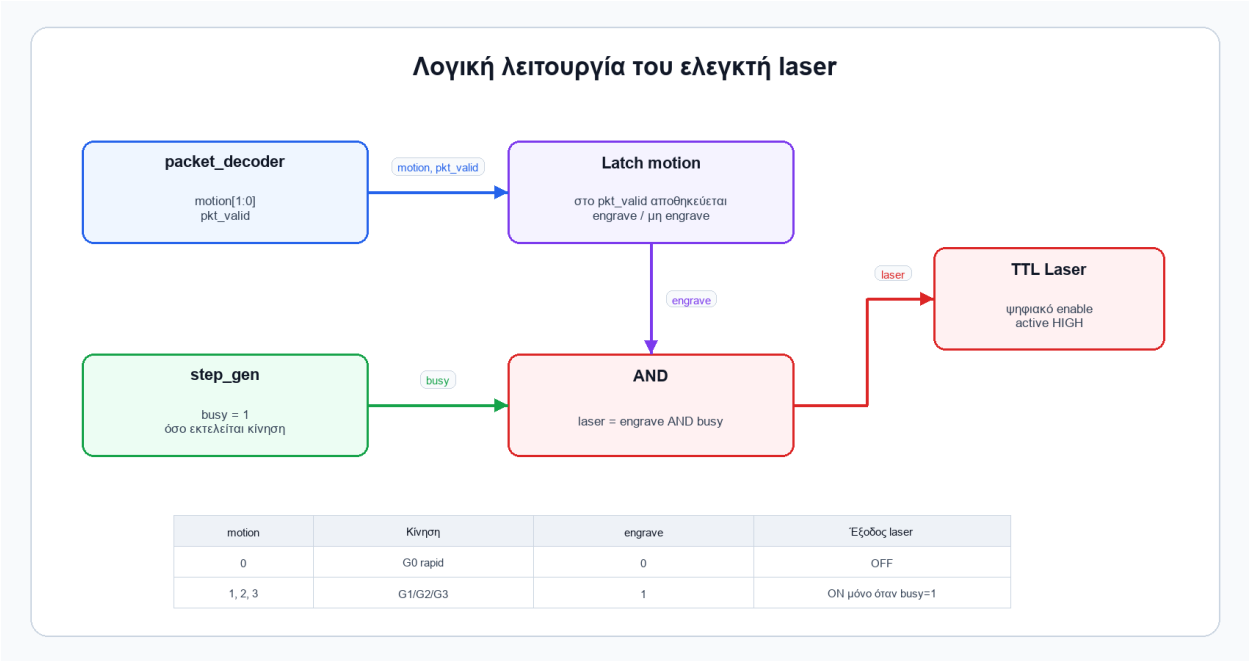
Για κινήσεις G0, οι οποίες χρησιμοποιούνται ως γρήγορες κινήσεις τοποθέτησης, το laser παραμένει σβηστό. Για κινήσεις G1, G2 και G3, οι οποίες αντιστοιχούν σε γραμμική ή καμπύλη χάραξη, η μονάδα επιτρέπει την ενεργοποίηση του laser όσο διαρκεί η κίνηση.

Πίνακας 5.8: Σήματα εισόδου εξόδου μονάδας κεφαλής (Lazer)

Σήμα	Κατεύθυνση	Περιγραφή
clk	είσοδος	Ρολόι συστήματος 50 MHz.
rst	είσοδος	Σύγχρονο reset, ενεργό σε λογικό 1.
motion[1:0]	είσοδος	Κωδικός κίνησης από τον packet_decoder.
pkt_valid	είσοδος	Παλμός ενός κύκλου όταν έχει ληφθεί νέο motion packet.
busy	είσοδος	Σήμα από τη step_gen που είναι 1 όσο εκτελείται κίνηση.
laser	έξοδος	Ψηφιακό TTL enable προς το laser, active HIGH.

5.2.5.1 Κανόνες ενεργοποίησης

Η μονάδα χρησιμοποιεί έναν εσωτερικό καταχωρητή engrave. Ο καταχωρητής αυτός κρατά την πληροφορία αν η τρέχουσα κίνηση είναι κίνηση χάραξης. Η τιμή του ενημερώνεται μόνο όταν εμφανιστεί pkt_valid, δηλαδή τη στιγμή που έχει ολοκληρωθεί η λήψη νέου motion packet.



Σχήμα 5.9: *Λογική λειτουργία της μονάδας laser_ctrl.*

Ο τύπος κίνησης αποθηκεύεται τη στιγμή που το pkt_valid γίνεται 1. Αυτό είναι σημαντικό, επειδή την ίδια χρονική στιγμή η step_gen ξεκινά να χρησιμοποιεί τα δεδομένα του νέου πακέτου. Αν το laser_ctrl διάβαζε απευθείας το motion χωρίς να το αποθηκεύσει, θα μπορούσε να εμφανιστεί χρονική ασάφεια ανάμεσα στην έναρξη της κίνησης και στην ενεργοποίηση του laser.

Με την αποθήκευση στον καταχωρητή engrave, η μονάδα κρατά σταθερή την απόφαση για όλη τη διάρκεια της κίνησης. Έτσι, ακόμη και αν το motion αλλάξει αργότερα λόγω νέου πακέτου, η έξοδος laser εξαρτάται από την πληροφορία που αντιστοιχεί στην τρέχουσα κίνηση.

Η τελική έξοδος της μονάδας προκύπτει από τη λογική πράξη laser = engrave AND busy. Αν το engrave είναι 0, το laser παραμένει σβηστό ανεξάρτητα από το busy. Αν το engrave είναι 1, το laser ανάβει μόνο όσο το busy είναι 1. Όταν η κίνηση ολοκληρωθεί και το busy επιστρέψει στο 0, το laser σβήνει αυτόματα.

Η επιλογή αυτή αποτρέπει την παραμονή του laser αναμμένου σε κατάσταση αναμονής. Ακόμη και για κίνηση χάραξης, το laser δεν ενεργοποιείται απλώς επειδή λήφθηκε εντολή G1/G2/G3. Ενεργοποιείται μόνο όταν η step_gen έχει ξεκινήσει την πραγματική κίνηση.

Σχήμα 5.10: *Χρονισμός ενεργοποίησης του laser σε κίνηση χάραξης.*

Κατά το reset, τόσο η έξοδος laser όσο και ο εσωτερικός καταχωρητής engrave μηδενίζονται. Αυτό σημαίνει ότι μετά από επανεκκίνηση ή αρχικοποίηση του FPGA το laser βρίσκεται σε ασφαλή κατάσταση OFF. Η ενεργοποίηση μπορεί να συμβεί μόνο μετά τη λήψη νέου motion packet που αντιστοιχεί σε κίνηση χάραξης και αφού το busy γίνει 1.

5.2.5.2 Βασικό απόσπασμα κώδικα

Το βασικό μέρος της υλοποίησης είναι μικρό και φαίνεται στο παρακάτω απόσπασμα. Ο καταχωρητής engrave ενημερώνεται στο pkt_valid, ενώ η έξοδος laser ακολουθεί τη λογική engrave AND busy.

```
reg engrave;   // 1 = this move should fire laser

always @(posedge clk) begin
    if (rst) begin
        laser   <= 1'b0;
        engrave <= 1'b0;
    end
    else begin
        if (pkt_valid) begin
            engrave <= (motion == 2'd1 ||
                        motion == 2'd2 ||
                        motion == 2'd3) ? 1'b1 : 1'b0;
        end

        laser <= engrave & busy;
    end
end
```

Η μονάδα laser_ctrl συμπληρώνει τη λειτουργία της step_gen. Η step_gen καθορίζει πότε κινείται η κεφαλή και παράγει τους παλμούς προς τους κινητήρες, ενώ η laser_ctrl καθορίζει αν κατά την ίδια κίνηση πρέπει να είναι ενεργό το laser. Με τον διαχωρισμό αυτό, η λογική κίνησης και η λογική ενεργοποίησης του laser παραμένουν απλές και ευανάγνωστες.

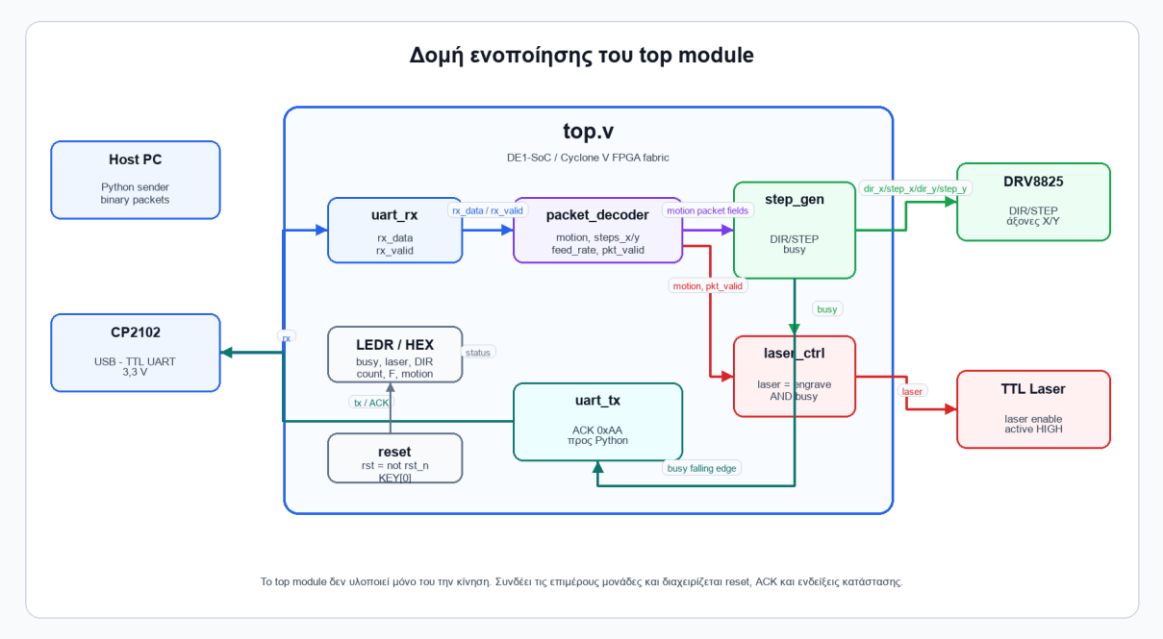
Στην πράξη, η λειτουργία είναι η εξής: το FPGA λαμβάνει ένα motion packet, ο packet_decoder δίνει τον τύπο κίνησης, η step_gen εκτελεί την κίνηση και η laser_ctrl ενεργοποιεί το laser μόνο αν η κίνηση είναι G1, G2 ή G3. Με αυτόν τον τρόπο οι κινήσεις γρήγορης μετακίνησης G0 δεν χαράζουν το υλικό.

5.2.6 Module ενοποίησης (top)

Το module top αποτελεί το ανώτερο επίπεδο της Verilog υλοποίησης. Δεν υλοποιεί από μόνο του έναν νέο αλγόριθμο κίνησης, αλλά ενοποιεί τις επιμέρους μονάδες του συστήματος και καθορίζει πώς μεταφέρονται τα σήματα από τη μία μονάδα στην επόμενη. Με αυτόν τον τρόπο το FPGA λειτουργεί ως ενιαία μονάδα ελέγχου της μηχανής CNC.

Στο top συνδέονται ο δέκτης UART, ο αποκωδικοποιητής πακέτων, η γεννήτρια παλμών, ο ελεγκτής laser και ο πομπός UART. Η βασική ροή είναι η εξής: ο υπολογιστής στέλνει bytes μέσω CP2102, ο uart_rx τα μετατρέπει σε παράλληλα δεδομένα, ο packet_decoder σχηματίζει πεδία κίνησης, η step_gen παράγει STEP/DIR και η laser_ctrl αποφασίζει αν πρέπει να ενεργοποιηθεί το laser.

Επιπλέον, το top διαχειρίζεται το reset, την αποστολή ACK προς τον υπολογιστή μετά την ολοκλήρωση κάθε κίνησης, καθώς και τις ενδείξεις LED και HEX που χρησιμοποιούνται για παρακολούθηση της λειτουργίας.



Σχήμα 5.11: *Δομή ενοποίησης των υπομονάδων στο top module.*

5.2.6.1 Εσωτερική διασύνδεση μονάδων

Η πρώτη σύνδεση αφορά τον uart_rx και τον packet_decoder. Ο uart_rx δίνει στην έξοδο το rx_data και έναν παλμό rx_valid κάθε φορά που έχει ληφθεί σωστά ένα byte. Ο packet_decoder χρησιμοποιεί αυτά τα bytes για να σχηματίσει είτε init packet είτε motion packet.

Όταν ολοκληρωθεί motion packet, τα πεδία motion, steps_x, steps_y και feed_rate γίνονται διαθέσιμα. Το pkt_valid ενημερώνει τις επόμενες μονάδες ότι τα πεδία αυτά είναι έγκυρα. Η step_gen χρησιμοποιεί τα steps_x, steps_y και feed_rate για την παραγωγή των παλμών, ενώ η laser_ctrl χρησιμοποιεί το motion και το busy για την ενεργοποίηση του laser.

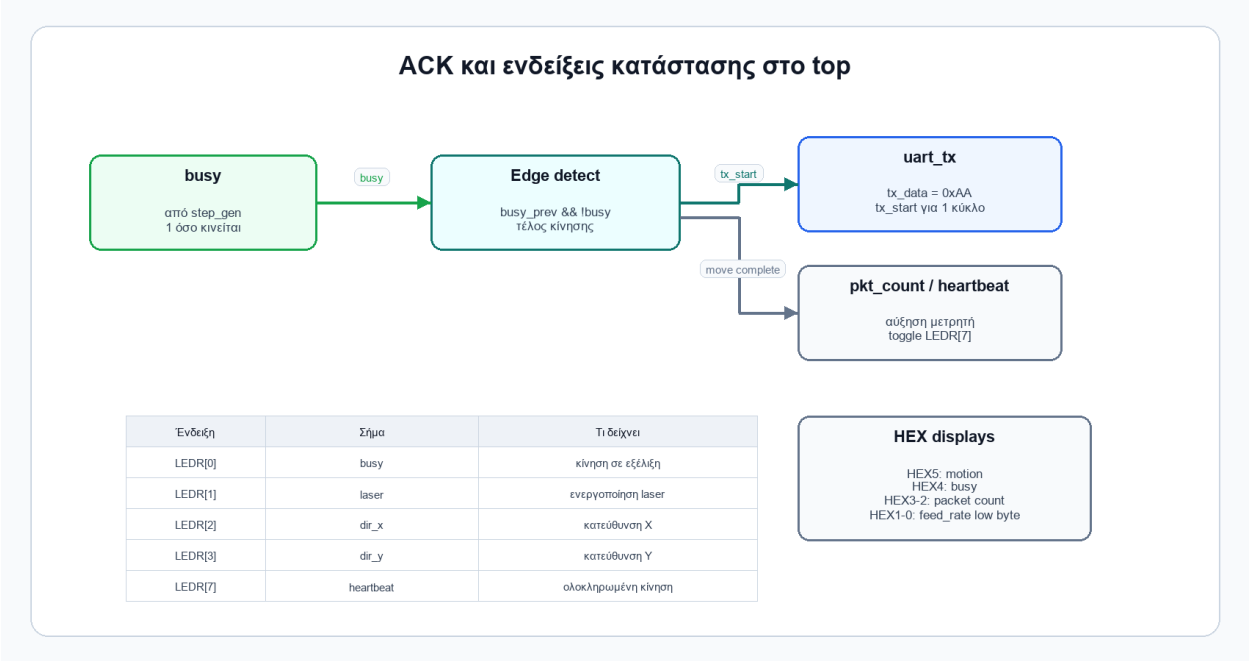
Στην τελική διασύνδεση, τα σήματα numerator_x, numerator_y και cfg_valid του packet_decoder πρέπει επίσης να οδηγούνται στη step_gen, ώστε η γεννήτρια παλμών να μπορεί να ενημερώνει δυναμικά τις παραμέτρους χρονισμού των αξόνων από το init packet.

Πίνακας 5.9: Σήματα μονάδας top.v

Μονάδα	Κύρια είσοδος	Κύρια έξοδος
uart_rx	rx	rx_data, rx_valid
packet_decoder	rx_data, rx_valid	motion, steps_x/y, feed_rate, pkt_valid, cfg_valid
step_gen	steps_x/y, feed_rate, pkt_valid	dir_x/y, step_x/y, busy
laser_ctrl	motion, pkt_valid, busy	laser
uart_tx	tx_data = 0xAA, tx_start	tx

Η επικοινωνία με τον υπολογιστή βασίζεται σε απλό handshake. Η Python στέλνει ένα πακέτο κίνησης και στη συνέχεια περιμένει από το FPGA ένα byte ACK. Το ACK έχει τιμή 0xAA και στέλνεται από τον uart_tx όταν ολοκληρωθεί η κίνηση.

Η ολοκλήρωση της κίνησης ανιχνεύεται από την πτώση του σήματος busy. Το top κρατά την προηγούμενη τιμή του busy στον καταχωρητή busy_prev. Όταν ισχύει busy_prev = 1 και busy = 0, σημαίνει ότι η step_gen μόλις πέρασε από κατάσταση κίνησης σε κατάσταση αναμονής. Τότε, αν ο uart_tx δεν είναι απασχολημένος, το tx_start γίνεται 1 για έναν κύκλο ρολογιού και αποστέλλεται το 0xAA.



Σχήμα 5.12: *Λογική ACK και ενδείξεων κατάστασης στο top module.*

```
tx_start <= 1'b0;      // default: no transmit
busy_prev <= busy;

if (busy_prev && !busy) begin
    if (!tx_busy)
        tx_start <= 1'b1;  // send ACK
    pkt_count <= pkt_count + 8'd1;
    LEDR[7]   <= ~LEDR[7];
end
```

Το top module είναι το σημείο όπου η επιμέρους λογική μετατρέπεται σε πλήρες σύστημα. Χωρίς το top, οι μονάδες uart_rx, packet_decoder, step_gen, laser_ctrl και uart_tx παραμένουν ανεξάρτητα κυκλώματα. Με τη διασύνδεσή τους, σχηματίζεται η συνολική αλυσίδα επικοινωνίας και ελέγχου της μηχανής.

Η αρχιτεκτονική αυτή διατηρεί καθαρά τα όρια των υπομονάδων: η UART λήψη ασχολείται μόνο με bytes, ο αποκωδικοποιητής με πακέτα, η γεννήτρια παλμών με χρονισμό STEP/DIR, ο ελεγκτής laser με το enable του laser και ο πομπός UART με το ACK. Το top συντονίζει αυτές τις λειτουργίες και τις συνδέει με τις πραγματικές ακίδες της πλακέτας.

Κεφάλαιο 6ο: Δοκιμές και αξιολόγηση

6.1 Πλατφόρμα και μεθοδολογία δοκιμών

Η πλατφόρμα και η μεθοδολογία δοκιμών εξετάζονται ενιαία, επειδή η διάταξη ελέγχου καθορίζει και τον τρόπο με τον οποίο επαληθεύεται η λειτουργία της υλοποίησης. Στόχος είναι να επιβεβαιωθεί η πλήρης ροή από το αρχείο βημάτων στον υπολογιστή έως τα σήματα STEP/DIR, την ενεργοποίηση του laser και την επιστροφή ACK από το FPGA.

Η διαδικασία είναι σταδιακή. Πρώτα ελέγχεται η ορθότητα του αρχείου εισόδου και η σειριακή επικοινωνία, στη συνέχεια η αποκωδικοποίηση στο FPGA και τέλος η πραγματική κίνηση των αξόνων. Το laser ενεργοποιείται μόνο αφού έχει επιβεβαιωθεί ότι η κίνηση χωρίς χάραξη είναι προβλέψιμη και ασφαλής.

Σχήμα 6.1: — *Συνοπτική διάταξη της πλατφόρμας δοκιμών.*

6.1.1 Διάταξη δοκιμών

Η διάταξη χωρίζεται σε τρία επίπεδα: το λογισμικό του υπολογιστή, την επικοινωνία UART μέσω CP2102 και την προγραμματιζόμενη λογική της πλακέτας DE1-SoC. Η έξοδος του FPGA οδηγεί τους DRV8825 για τους δύο άξονες και την TTL είσοδο του laser.

Πίνακας 6.1: *Βασικά στοιχεία της πλατφόρμας δοκιμών.*

Τμήμα	Στοιχείο	Ρόλος
Υπολογιστής	AutoCAD / AutoLISP / steps file	Παραγωγή γεωμετρικής διαδρομής και βημάτων.
Υπολογιστής	Python 3 / pyserial	Ανάγνωση αρχείου, αποστολή πακέτων και αναμονή ACK.
Επικοινωνία	CP2102 USB-TTL UART	Σύνδεση USB με UART 3.3 V προς το FPGA.
FPGA	DE1-SoC / Cyclone V	UART, packet decoding, step generation και laser control.
Μηχανή	DRV8825, βηματικοί κινητήρες, TTL laser	Παραγωγή κίνησης X/Y και έλεγχος χάραξης.

Πίνακας 6.2: — *Κύριες συνδέσεις και παράμετροι της πλατφόρμας.*

Παράμετρος	Τιμή / σύνδεση	Σχόλιο
Ρολόι FPGA	50 MHz	Βασικός χρονισμός των RTL counters.
UART	115200 baud, 8N1	Ίδια ρύθμιση στο Python script και στο FPGA.
RX / TX	GPIO_0[0] / GPIO_0[1]	Σύνδεση με TX/RX του CP2102.
STEP/DIR	GPIO_0[2..5]	Έξοδοι προς τους οδηγούς X και Y.
Laser / ACK	GPIO_0[6] / 0xAA	TTL laser enable και επιβεβαίωση ολοκλήρωσης κίνησης.

Πίνακας 6.3: *Ενδείξεις που χρησιμοποιούνται κατά τις δοκιμές.*

Ένδειξη	Πληροφορία	Χρήση
LEDR[0]	busy	Δείχνει ενεργή κίνηση από το step_gen.
LEDR[1]	laser	Επιβεβαιώνει πότε ενεργοποιείται η έξοδος laser.
LEDR[2], LEDR[3]	dir_x, dir_y	Έλεγχος φοράς κίνησης στους δύο άξονες.
LEDR[7]	heartbeat	Αλλάζει κατάσταση μετά από ολοκληρωμένη κίνηση.
HEX displays	motion, busy, packet count, feed	Γρήγορη παρακολούθηση της κατάστασης χωρίς όργανο.

6.1.2 Μεθοδολογία ελέγχου

Η μεθοδολογία ακολουθεί τη φυσική ροή των δεδομένων. Κάθε στάδιο θεωρείται αποδεκτό μόνο όταν έχει επιβεβαιωθεί το αναμενόμενο αποτέλεσμα. Αν παρουσιαστεί απόκλιση, ο έλεγχος επιστρέφει στο προηγούμενο επίπεδο μέχρι να εντοπιστεί η αιτία.

Σχήμα 6.2: *Συμπυκνωμένη ροή της μεθοδολογίας δοκιμών.*

Πίνακας 6.4: *Στάδια της μεθοδολογίας δοκιμών.*

Στάδιο	Αντικείμενο ελέγχου	Κριτήριο επιτυχίας
1	Προετοιμασία διάταξης	Σωστές συνδέσεις, κοινή γείωση και ασφαλής κατάσταση laser.
2	Αρχείο εισόδου	Έγκυρες εγγραφές motion, steps_x, steps_y και feed_rate.
3	Προγραμματισμός FPGA	Το .sof φορτώνεται στην DE1-SoC χωρίς σφάλμα.

4	UART / ACK	Το Python script ανοίγει τη COM θύρα και λαμβάνει ACK 0xAA.
5	Κίνηση χωρίς laser	Οι άξονες κινούνται σωστά και το heartbeat αλλάζει.
6	Laser και πλήρης διαδρομή	Το laser λειτουργεί μόνο σε χάραξη και όλα τα πακέτα ολοκληρώνονται.

Πίνακας 6.5: *Ενδεικτικά σενάρια δοκιμών.*

Σενάριο	Σκοπός	Παρατήρηση
Μικρή ή μηδενική κίνηση	Έλεγχος ότι δεν παράγονται ανεπιθύμητοι παλμοί.	Το busy δεν πρέπει να μένει ενεργό χωρίς λόγο.
Κίνηση μόνο στον X ή Y	Έλεγχος STEP/DIR κάθε άξονα ξεχωριστά.	Ο άλλος άξονας πρέπει να παραμένει ακίνητος.
Διαγώνια X-Y	Έλεγχος συντονισμένης κίνησης.	Οι άξονες ολοκληρώνουν την κίνηση συγχρονισμένα.
G0 μετακίνηση	Έλεγχος γρήγορης κίνησης χωρίς χάραξη.	Το laser παραμένει απενεργοποιημένο.
G1/G2/G3 ή μικρό σχέδιο	Έλεγχος χάραξης και ολοκληρωμένης ροής.	Κάθε κίνηση επιστρέφει ACK πριν σταλεί η επόμενη.

6.1.3 Καταγραφή και απομόνωση σφαλμάτων

Κάθε δοκιμή συνοδεύεται από σύντομη καταγραφή των βασικών παραμέτρων και των παρατηρούμενων ενδείξεων. Η απομόνωση σφάλματος γίνεται με βάση το πρώτο επίπεδο στο οποίο η συμπεριφορά αποκλίνει από το αναμενόμενο.

Πίνακας 6.6: *Στοιχεία που καταγράφονται σε κάθε δοκιμή.*

Πεδίο	Τι καταγράφεται	Χρήση
Αρχείο εισόδου	Όνομα αρχείου ή σεναρίου.	Επιτρέπει επανάληψη της ίδιας δοκιμής.
Σειριακή θύρα	COM port και baud rate.	Επιβεβαιώνει τη σωστή ρύθμιση επικοινωνίας.
Πακέτα / ACK	Πλήθος πακέτων και λήψη ACK ή timeout.	Ελέγχει τον συγχρονισμό host-FPGA.
LEDR / HEX	busy, laser, DIR, heartbeat, motion, feed.	Βοηθά στον γρήγορο εντοπισμό αποκλίσεων.
Κίνηση / laser	Φορά, ομαλότητα, ολοκλήρωση και κατάσταση laser.	Επιβεβαιώνει step_gen, οδηγούς και laser_ctrl.

Πίνακας 6.7: *Οδηγός απομόνωσης σφαλμάτων κατά τις δοκιμές.*

Σύμπτωμα	Πιθανό σημείο ελέγχου	Ενέργεια
Δεν ανοίγει η COM θύρα	CP2102, driver, επιλογή θύρας	Έλεγχος Device Manager και COM_PORT.
Δεν λαμβάνεται ACK	RX/TX, packet_decoder, busy	Έλεγχος καλωδίωσης, baud rate και heartbeat.

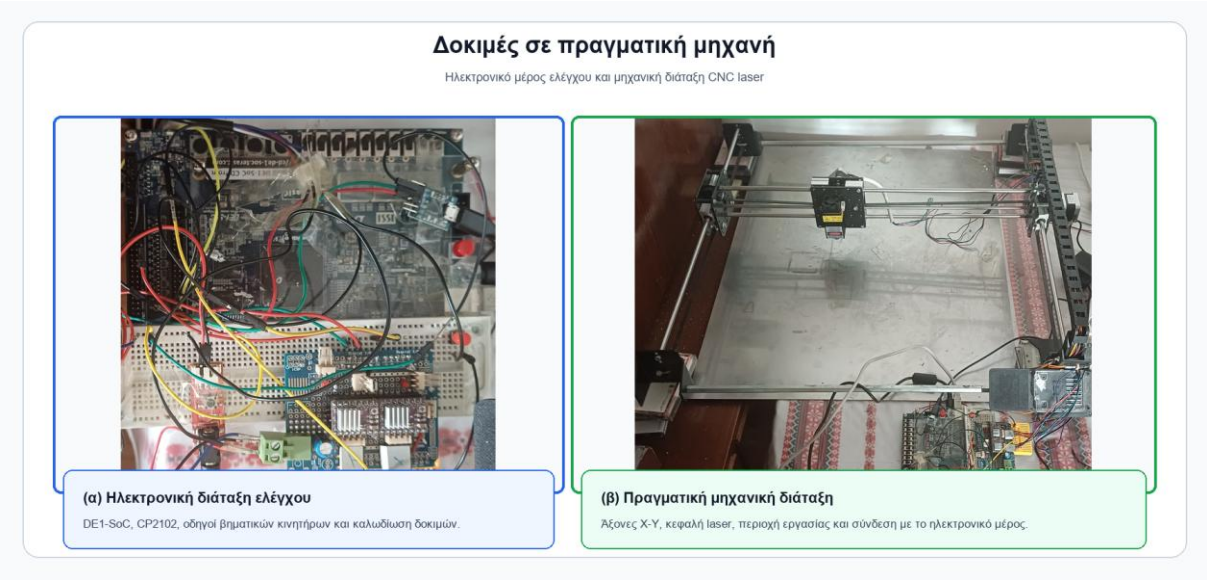
Timeout σε κίνηση	step_gen ή πλήθος βημάτων	Έλεγχος steps_x, steps_y και feed_rate.
Λάθος φορά ή άξονας	DIR/STEP ή σύνδεση οδηγού	Έλεγχος σημάτων και πιθανή αντιστροφή φοράς.
Laser ενεργό σε G0	motion field ή laser_ctrl	Έλεγχος motion στο πακέτο και λογικής laser.

Η δοκιμή θεωρείται ολοκληρωμένη όταν όλα τα πακέτα έχουν σταλεί, έχει ληφθεί ACK για κάθε κίνηση και οι ενδείξεις συμφωνούν με την αναμενόμενη λειτουργία. Σε περίπτωση απόκλισης, το αποτέλεσμα χρησιμοποιείται για διόρθωση στο αντίστοιχο επίπεδο και η δοκιμή επαναλαμβάνεται.

6.2 Δοκιμές σε πραγματική μηχανή

Μετά την επαλήθευση της επικοινωνίας, των πακέτων και της γεννήτριας παλμών, η υλοποίηση δοκιμάστηκε πάνω στην πραγματική μηχανή CNC laser. Στο στάδιο αυτό δεν ελέγχεται μόνο η ορθότητα της RTL λογικής, αλλά και η συνεργασία της με το ηλεκτρονικό και μηχανικό μέρος: τη σύνδεση CP2102, τους οδηγούς των βηματικών κινητήρων, την καλωδίωση STEP/DIR, την κεφαλή laser και την πραγματική περιοχή εργασίας.

Οι δοκιμές οργανώθηκαν με προοδευτική αύξηση της πολυπλοκότητας. Αρχικά ελέγχθηκε η ηλεκτρονική διάταξη και η κίνηση χωρίς ενεργό laser. Στη συνέχεια εκτελέστηκαν μικρές κινήσεις σε κάθε άξονα, συνδυασμένη κίνηση X-Y και απλές διαδρομές χάραξης, ώστε να επιβεβαιωθεί ότι η κεφαλή ακολουθεί τη ζητούμενη διαδρομή και ότι το laser ενεργοποιείται μόνο όταν η κίνηση αντιστοιχεί σε χάραξη.



Σχήμα 6.3: *Πραγματική διάταξη δοκιμών: (α) ηλεκτρονικό μέρος ελέγχου με DE1-SoC, CP2102 και οδηγούς, (β) μηχανική διάταξη CNC laser με άξονες X-Y και κεφαλή laser.*

6.2.1 Προετοιμασία διάταξης

Πριν από την εκτέλεση κινήσεων ελέγχθηκαν οι βασικές συνδέσεις της διάταξης. Η εικόνα του ηλεκτρονικού μέρους δείχνει ότι το FPGA συνδέεται μέσω CP2102 με τον υπολογιστή και μέσω GPIO με την πλακέτα οδηγών. Επειδή οι έξοδοι GPIO της DE1-SoC είναι στάθμης 3.3 V, αποφεύγεται η απευθείας σύνδεση με σήματα 5 V και επιβεβαιώνεται κοινή γείωση μεταξύ FPGA, CP2102, οδηγών και laser.

Πίνακας 6.8: *Έλεγχοι προετοιμασίας πριν από τις δοκιμές στην πραγματική μηχανή.*

Έλεγχος	Σκοπός	Αποδοχή
Τροφοδοσία και γείωση	Σταθερή αναφορά τάσης για όλα τα επιμέρους κυκλώματα.	Κοινή γείωση μεταξύ FPGA, CP2102, οδηγών και laser.
RX/TX CP2102	Σωστή σειριακή επικοινωνία με τον υπολογιστή.	Η θύρα ανοίγει στα 115200 baud και επιστρέφεται ACK.
STEP/DIR οδηγοί	Σωστή αντιστοίχιση σημάτων X και Y.	Κάθε άξονας κινείται μόνο όταν λαμβάνει τους αντίστοιχους παλμούς.
Laser enable	Έλεγχος της TTL εισόδου laser.	Το laser παραμένει ανενεργό σε G0 και ενεργοποιείται μόνο σε χάραξη.
Μηχανική διάταξη	Ελεύθερη κίνηση κεφαλής στην περιοχή εργασίας.	Δεν υπάρχει εμπλοκή καλωδίων ή μηχανικών μερών στη διαδρομή.

6.2.2 Δοκιμές κίνησης αξόνων

Οι πρώτες κινήσεις έγιναν με απενεργοποιημένο laser. Με τον τρόπο αυτό ελέγχθηκε αν οι παλμοί STEP μετατρέπονται σε πραγματική κίνηση και αν η φορά που δίνει το DIR αντιστοιχεί στην αναμενόμενη μετατόπιση. Οι κινήσεις μικρής απόστασης είναι ιδιαίτερα χρήσιμες, επειδή επιτρέπουν γρήγορο έλεγχο χωρίς κίνδυνο πρόσκρουσης στα όρια της μηχανής.

Πίνακας 6.9: *Δοκιμές κίνησης των αξόνων στην πραγματική μηχανή.*

Δοκιμή	Τι ελέγχεται	Αναμενόμενη συμπεριφορά
Κίνηση X+	STEP/DIR άξονα X προς θετική φορά.	Η κεφαλή μετακινείται μόνο στον X.
Κίνηση X-	Αντιστροφή φοράς του άξονα X.	Η κεφαλή επιστρέφει χωρίς κίνηση στον Y.
Κίνηση Y+	STEP/DIR άξονα Y προς θετική φορά.	Η γέφυρα/κεφαλή μετακινείται στον Y.
Κίνηση Y-	Αντιστροφή φοράς του άξονα Y.	Η κίνηση γίνεται στην αντίθετη κατεύθυνση.
Διαγώνια X-Y	Συνδυασμένη κίνηση δύο αξόνων.	Οι άξονες ολοκληρώνουν συγχρονισμένα την εντολή.

6.2.3 Δοκιμές laser και γεωμετρικής διαδρομής

Αφού επιβεβαιώθηκε η βασική κίνηση, ελέγχθηκε η έξοδος laser. Η λογική της υλοποίησης διαχωρίζει τις γρήγορες μετακινήσεις από τις κινήσεις χάραξης: στις κινήσεις G0 η έξοδος laser πρέπει να παραμένει ανενεργή, ενώ στις κινήσεις G1/G2/G3 μπορεί να ενεργοποιείται όσο το FPGA βρίσκεται σε κατάσταση busy. Έτσι αποφεύγεται ανεπιθύμητη χάραξη κατά τη μετακίνηση από ένα σημείο σε άλλο.

Πίνακας 6.10: *Έλεγχος ενεργοποίησης laser στην πραγματική μηχανή.*

Περίπτωση	Σήμα / κατάσταση	Κριτήριο
G0 μετακίνηση	motion=0, busy ενεργό	Laser OFF σε όλη τη διάρκεια της μετακίνησης.
G1 ευθεία	motion=1, busy ενεργό	Laser ON μόνο όσο εκτελείται η χάραξη.
G2/G3 τόξο	motion=2/3 ή αντίστοιχη κωδικοποίηση	Laser ON κατά την εκτέλεση των κινήσεων χάραξης.
Τέλος κίνησης	busy ανενεργό και ACK	Laser OFF και αποστολή ACK στον υπολογιστή.

Πίνακας 6.11: *Απλές γεωμετρικές διαδρομές για δοκιμή σε πραγματική μηχανή.*

Διαδρομή	Σκοπός	Παρατήρηση
Ευθεία X ή Y	Έλεγχος ενός άξονα με ενεργή χάραξη.	Το ίχνος πρέπει να είναι συνεχές και ευθύ.
Διαγώνια γραμμή	Έλεγχος συνδυασμένης κίνησης X-Y.	Η κεφαλή πρέπει να κινείται ομαλά χωρίς εμφανή διακοπή.
Ορθογώνιο	Έλεγχος αλλαγών κατεύθυνσης και επιστροφής.	Οι πλευρές και οι γωνίες πρέπει να είναι ευδιάκριτες.
Μικρό τόξο	Έλεγχος ακολουθίας μικρών κινήσεων χάραξης.	Το ίχνος πρέπει να προσεγγίζει ομαλή καμπύλη.

6.2.4 Παρακολούθηση και αξιολόγηση

Κατά την εκτέλεση παρακολουθούνται ταυτόχρονα η φυσική κίνηση, οι ενδείξεις της πλακέτας και η έξοδος του προγράμματος Python. Το ACK είναι κρίσιμο, επειδή επιβεβαιώνει ότι ο υπολογιστής δεν στέλνει το επόμενο πακέτο πριν ολοκληρωθεί η τρέχουσα κίνηση. Παράλληλα, οι ενδείξεις busy, laser και heartbeat βοηθούν στον γρήγορο εντοπισμό σφαλμάτων.

Πίνακας 6.12: *Σημεία παρακολούθησης κατά τις δοκιμές σε πραγματική μηχανή.*

Σημείο παρακολούθησης	**Τι δείχνει**	**Χρήση**
Κίνηση κεφαλής	Πραγματική μετατόπιση στους άξονες X/Y.	Επιβεβαίωση φοράς, ομαλότητας και ολοκλήρωσης.
LEDR / HEX	busy, laser, direction, packet count και feed.	Γρήγορη σύγκριση πραγματικής κατάστασης με εντολή.
Python sender	Αποστολή πακέτων και λήψη ACK.	Εντοπισμός timeout ή απώλειας συγχρονισμού.
Ίχνος χάραξης	Οπτικό αποτέλεσμα πάνω στο δοκίμιο.	Έλεγχος συνέχειας, θέσης και ενεργοποίησης laser.

Πίνακας 6.13: *Κριτήρια αποδοχής των δοκιμών στην πραγματική μηχανή.*

Κριτήριο	**Αποδεκτή συμπεριφορά**	**Ενέργεια σε απόκλιση**
Συγχρονισμός πακέτων	ACK μετά από κάθε ολοκληρωμένη κίνηση.	Έλεγχος UART, packet_decoder και busy.
Κίνηση αξόνων	Η κεφαλή κινείται στη ζητούμενη φορά και απόσταση.	Έλεγχος STEP/DIR, οδηγών και καλωδίωσης.
Laser	Ενεργό μόνο σε κινήσεις χάραξης και όχι σε G0.	Έλεγχος motion field και laser_ctrl.
Μηχανική ασφάλεια	Δεν υπάρχει εμπλοκή ή πρόσκρουση στη διαδρομή.	Διακοπή δοκιμής και έλεγχος ορίων/καλωδίων.
Οπτικό αποτέλεσμα	Το ίχνος συμφωνεί με την απλή γεωμετρία δοκιμής.	Έλεγχος steps/mm, φοράς αξόνων και προώσεων.

Οι δοκιμές στην πραγματική μηχανή επιβεβαιώνουν τη συνεργασία των επιμέρους βαθμίδων της υλοποίησης: παραγωγή πακέτων στον υπολογιστή, λήψη και αποκωδικοποίηση στο FPGA, παραγωγή παλμών STEP/DIR, οδήγηση των αξόνων και έλεγχος laser. Η χρήση της πραγματικής διάταξης είναι απαραίτητη, επειδή φανερώνει θέματα που δεν εμφανίζονται σε απομονωμένο έλεγχο RTL, όπως καλωδίωση, φορά αξόνων, μηχανικές αντιστάσεις και συμπεριφορά της κεφαλής πάνω στο δοκίμιο.

6.3 Μετρήσεις ακρίβειας και jitter

Το κεφάλαιο αυτό αξιολογεί τη συμπεριφορά της μηχανής μετά την ολοκλήρωση της επικοινωνίας υπολογιστή-FPGA και της παραγωγής παλμών STEP/DIR. Η αξιολόγηση χωρίζεται σε δύο επίπεδα: στη γεωμετρική ακρίβεια της παραγόμενης χάραξης και στη χρονική σταθερότητα των παλμών που οδηγούν τους βηματικούς κινητήρες.

Η γεωμετρική ακρίβεια ελέγχεται από το τελικό ίχνος πάνω στο δοκίμιο, δηλαδή από το αν οι ευθείες, οι διαγώνιες και οι καμπύλες ακολουθούν τη μορφή που ζητήθηκε από το G-code. Το jitter, αντίθετα, είναι χρονικό μέγεθος και αφορά τη διακύμανση της περιόδου μεταξύ διαδοχικών παλμών STEP. Για τον λόγο αυτό οι φωτογραφίες της χάραξης μπορούν να δείξουν ενδείξεις αστάθειας ή απώλειας βημάτων, αλλά δεν αρκούν για αριθμητική μέτρηση jitter χωρίς παλμογράφο ή λογικό αναλυτή.

Ενδεικτικά (για τις δοκιμές που έγιναν) είναι τα Σχήμα 6.4, Σχήμα 6.6 και η έξοδος του Python sender που φαίνεται στο Σχήμα 6.5.

6.3.1 Πειραματική ροή αποστολής

Πριν από την αξιολόγηση της χάραξης ελέγχθηκε ότι η εφαρμογή Python στέλνει διαδοχικά τα πακέτα κίνησης προς το FPGA και λαμβάνει επιβεβαίωση ολοκλήρωσης. Η έξοδος του προγράμματος δείχνει ότι μεταδόθηκαν έξι κινήσεις, ότι όλες στάλθηκαν επιτυχώς και ότι δεν αναφέρθηκε σφάλμα μετάδοσης. Η ένδειξη αυτή δεν αποτελεί μέτρηση ακρίβειας, αλλά τεκμηριώνει ότι η δοκιμή εκτελέστηκε ως πλήρης ακολουθία κινήσεων.

```
Starting transmission...

✓ Seg 1: G0-Rapid  X=4257  Y=15889  F=1200
✓ Seg 2: G1-Linear  X=0  Y=8000  F=800=  +4257  Y= +15889  F=1200  ETA:  26.2s
✓ Seg 3: G0-Rapid  X=2317  Y=-5034  F=1200
✓ Seg 4: G1-Linear  X=4132  Y=3864  F=800
✓ Seg 5: G0-Rapid  X=-3807  Y=-6829  F=1200
✓ Seg 6: G1-Linear  X=6000  Y=0  F=800
Seg     6/6  [100.0%]  G1-Linear     X=  +6000  Y=     +0  F= 800  ETA:   0.0s

============================================================
Transmission complete
============================================================
Total segments : 6
Sent OK        : 6
Errors         : 0
Time elapsed   : 18.11s
Avg rate       : 0.3 segments/sec
============================================================
```

Σχήμα 6.6 *Έξοδος του Python script κατά την αποστολή έξι κινήσεων προς το FPGA.*

Πίνακας 6.14 Πειραματικά τεκμήρια για την αξιολόγηση του κεφαλαίου 6.5.

Τεκμήριο	Ρόλος στη δοκιμή	Τι αξιολογείται
Σχήμα 6.4	Έξοδος της εφαρμογής Python κατά την αποστολή της δοκιμαστικής διαδρομής.	Σωστή ακολουθία segment, επιβεβαίωση αποστολής, απουσία σφαλμάτων επικοινωνίας.
Σχήμα 6.7	Φωτογραφία ευθύγραμμων κινήσεων μετά την αφαίρεση βοηθητικών σημάνσεων.	Ευθύτητα, συνέχεια γραμμής, συντονισμός των αξόνων σε διαγώνια κίνηση.
Σχήμα 6.8	Δεύτερη φωτογραφία ευθύγραμμων κινήσεων, επίσης χωρίς βοηθητικό περίγραμμα.	Επαναληψιμότητα της μορφής, ομοιομορφία ίχνους, πιθανές τοπικές ασυνέχειες.
Σχήμα 6.6	Κλειστή καμπύλη γεωμετρία πάνω στο δοκίμιο.	Συνέχεια περιγράμματος, ομαλότητα καμπυλών και συμπεριφορά στις μεταβάσεις.

6.3.2 Έλεγχος γραμμικών κινήσεων

Οι γραμμικές δοκιμές περιλαμβάνουν οριζόντιες, κατακόρυφες και διαγώνιες κινήσεις. Οι οριζόντιες και κατακόρυφες γραμμές ελέγχουν κυρίως τη συμπεριφορά ενός άξονα κάθε φορά, ενώ η διαγώνια γραμμή ελέγχει τη συνδυασμένη λειτουργία των δύο αξόνων. Εάν υπήρχε σοβαρό πρόβλημα στον συγχρονισμό των παλμών, θα ήταν πιθανό να εμφανιστούν απότομα σπασίματα, ασυνέχειες ή έντονες αλλαγές κλίσης στη διαγώνια τροχιά.

Σχήμα 6.9 *Δοκιμές ευθύγραμμων κινήσεων.*

Στις φωτογραφίες παρατηρείται ότι οι γραμμές παράγονται ως συνεχόμενα ίχνη χωρίς εμφανή διακοπή της κίνησης. Η διαγώνια χάραξη δείχνει ότι οι δύο άξονες μπορούν να κινηθούν ταυτόχρονα, ενώ οι κατακόρυφες και οριζόντιες γραμμές επιβεβαιώνουν τη βασική λειτουργία κάθε άξονα χωριστά. Οι μικρές μεταβολές στο πάχος ή στην ένταση του ίχνους δεν μπορούν να αποδοθούν απευθείας στο FPGA, επειδή επηρεάζονται επίσης από το υλικό του δοκιμίου, την εστίαση του laser, την ταχύτητα, την ισχύ και την τοπική απορρόφηση της επιφάνειας.

6.3.3 Έλεγχος κλειστής και καμπύλης γεωμετρίας

Το Σχήμα 6.10 χρησιμοποιείται ως δοκιμή πιο σύνθετης γεωμετρίας. Σε σχέση με μια απλή ευθεία, μια κλειστή καμπύλη μορφή είναι πιο χρήσιμη για τον εντοπισμό σωρευτικών αποκλίσεων, επειδή περιλαμβάνει συνεχείς αλλαγές κατεύθυνσης και μεταβάσεις μεταξύ τμημάτων της τροχιάς. Η επιτυχής χάραξη μιας τέτοιας μορφής δείχνει ότι η ακολουθία πακέτων, η αποκωδικοποίηση και η παραγωγή παλμών συνεργάζονται σωστά σε διαδοχικές κινήσεις.

Σχήμα 6.11 *Κλειστή καμπύλη δοκιμής για αξιολόγηση συνέχειας τροχιάς και ομαλότητας της χάραξης.*

Το περίγραμμα παρουσιάζει συνεχή διαδρομή χωρίς εμφανή απώλεια segment. Η τραχύτητα στα όρια της γραμμής είναι αναμενόμενη σε πραγματική χάραξη πάνω σε ξύλινη ή ινώδη επιφάνεια και δεν ταυτίζεται υποχρεωτικά με σφάλμα θέσης. Για αριθμητική αξιολόγηση θα πρέπει η γεωμετρία να μετρηθεί με βαθμονομημένη φωτογραφία, παχύμετρο ή άλλο όργανο μέτρησης, ώστε να συγκριθούν οι πραγματικές διαστάσεις με τις ονομαστικές διαστάσεις του σχεδίου.

Πίνακας 6.15 Παρατηρήσεις από τις πραγματικές χαράξεις.

Δοκιμή	Παρατήρηση	Ερμηνεία
Αποστολή 6 segment	Η έξοδος Python εμφανίζει Sent OK = 6 και Errors = 0.	Η επικοινωνία ολοκληρώθηκε χωρίς αναφορά σφάλματος και η ακολουθία κινήσεων εκτελέστηκε πλήρως.
Οριζόντια/κατακόρυφη γραμμή	Το ίχνος παραμένει συνεχές κατά μήκος της κίνησης.	Δεν φαίνεται εμφανής απώλεια παλμών ή διακοπή της κίνησης σε έναν άξονα.
Διαγώνια γραμμή	Η γραμμή σχηματίζεται ως ενιαία τροχιά με σταθερή γενική κλίση.	Η ταυτόχρονη κίνηση X-Y λειτουργεί επαρκώς για τη συγκεκριμένη δοκιμή.
Κλειστή καμπύλη	Το περίγραμμα χαράσσεται ως συνεχής μορφή.	Η διαδοχή των κινήσεων και οι μεταβάσεις μεταξύ τμημάτων δεν εμφανίζουν ορατή ασυνέχεια.
Μεταβολή πάχους ίχνους	Υπάρχουν τοπικές διαφοροποιήσεις στο πλάτος και στην ένταση της γραμμής.	Πιθανές αιτίες είναι το υλικό, η εστίαση, η ισχύς laser και η ταχύτητα. Δεν τεκμηριώνεται από μόνη της χρονικό jitter.

6.3.4 Θεωρητική εκτίμηση jitter

Στη συγκεκριμένη υλοποίηση, μετά τη λήψη ενός έγκυρου πακέτου, η παραγωγή των παλμών STEP γίνεται μέσα στο FPGA από τη μονάδα step_gen. Αυτό σημαίνει ότι, κατά τη διάρκεια μιας κίνησης, ο χρονισμός των παλμών δεν εξαρτάται από τον υπολογιστή, το λειτουργικό σύστημα ή τη σειριακή επικοινωνία [2][6]. Τα στοιχεία αυτά επηρεάζουν κυρίως τον χρόνο ανάμεσα σε δύο διαδοχικά segment, επειδή το επόμενο πακέτο αποστέλλεται μετά την ολοκλήρωση της προηγούμενης κίνησης και την αποστολή ACK.

Το FPGA λειτουργεί με ρολόι 50 MHz, δηλαδή με περίοδο 20 ns. Επομένως η χρονική θέση κάθε ακμής STEP μπορεί να μεταβληθεί μόνο κατά ακέραια πολλαπλάσια των 20 ns. Η περίοδος βήματος υπολογίζεται από τη σχέση cycles_per_step = NUMERATOR / feed_rate. Επειδή το αποτέλεσμα τελικά υλοποιείται ως ακέραιος αριθμός κύκλων ρολογιού, το αριθμητικό σφάλμα χρονισμού από τη στρογγυλοποίηση περιορίζεται σε λιγότερο από έναν κύκλο ρολογιού ανά περίοδο.

Πίνακας 6.16 Όρια και τρόπος αξιολόγησης του jitter στην υλοποίηση FPGA.

Μέγεθος	Τιμή / τρόπος ελέγχου	Σχόλιο
Συχνότητα ρολογιού	50 MHz	Βασικός χρονισμός όλων των μετρητών της RTL υλοποίησης.
Περίοδος ρολογιού	20 ns	Ελάχιστη χρονική ανάλυση με την οποία μπορούν να τοποθετηθούν οι ακμές των σημάτων.
STEP high	100 κύκλοι = 2 μs	Συμβατό με τις ελάχιστες απαιτήσεις οδηγών τύπου DRV8825.

DIR settle	200 κύκλοι = 4 μs	Χρόνος αναμονής μετά την αλλαγή κατεύθυνσης πριν από τον πρώτο παλμό.
Jitter κατά την κίνηση	Μετράται με παλμογράφο ή λογικό αναλυτή στις ακμές STEP.	Δεν μπορεί να υπολογιστεί αξιόπιστα μόνο από φωτογραφία του ίχνους.
Καθυστέρηση μεταξύ segment	Επηρεάζεται από ACK, Python και UART.	Δεν είναι jitter των STEP παλμών, αλλά νεκρός χρόνος ανάμεσα σε κινήσεις.

6.3.5 Σχέση με τις δοκιμές της προγενέστερης υλοποίησης

Η λογική των δοκιμών ακολουθεί το ίδιο πνεύμα με το Appendix C, Operation Tests, του προγενέστερου project “Two Axis CNC - Hardware and Software Development”, όπου ελέγχθηκαν βασικές κινήσεις G00, G01 σε κατακόρυφη, οριζόντια και κεκλιμένη γραμμή, καθώς και κινήσεις G02/G03. Στην παρούσα εργασία οι ίδιες κατηγορίες γεωμετρικών δοκιμών παραμένουν χρήσιμες, αλλά αξιολογούνται πλέον μέσα από διαφορετική αρχιτεκτονική: η ακολουθία παράγεται στον υπολογιστή, μετατρέπεται σε δυαδικά πακέτα, αποστέλλεται μέσω UART και εκτελείται από FPGA.

Η διαφορά αυτή είναι σημαντική για την ερμηνεία των αποτελεσμάτων. Στην παλαιότερη υλοποίηση η έμφαση δινόταν κυρίως στην επιβεβαίωση της λειτουργίας της μηχανής και των κινήσεων. Στη νέα υλοποίηση πρέπει επιπλέον να διαχωρίζεται η αξιοπιστία της επικοινωνίας από τη χρονική σταθερότητα της παλμογένεσης. Η έξοδος Python τεκμηριώνει την αποστολή και το ACK, ενώ οι φωτογραφίες τεκμηριώνουν την πραγματική γεωμετρική συμπεριφορά της μηχανής.

6.3.6 Συμπέρασμα μετρήσεων

Από τα διαθέσιμα πειραματικά τεκμήρια προκύπτει ότι η μηχανή εκτελεί συνεχείς ευθύγραμμες και καμπύλες κινήσεις χωρίς εμφανή απώλεια segment ή πλήρη διακοπή της τροχιάς. Η έξοδος του Python sender δείχνει επιτυχή ολοκλήρωση της αποστολής, ενώ οι φωτογραφίες επιβεβαιώνουν ότι οι παλμοί του FPGA οδηγούν τους άξονες και το laser με λειτουργικό τρόπο.

Η αξιολόγηση της γεωμετρικής ακρίβειας παραμένει κυρίως ποιοτική, επειδή οι φωτογραφίες δεν συνοδεύονται από βαθμονομημένη κλίμακα μέτρησης. Για πλήρη ποσοτική αξιολόγηση απαιτείται μέτρηση διαστάσεων πάνω στο δοκίμιο και καταγραφή παλμών STEP με παλμογράφο ή λογικό αναλυτή. Παρόλα αυτά, τα διαθέσιμα αποτελέσματα είναι επαρκή για να τεκμηριώσουν ότι η νέα υλοποίηση FPGA μπορεί να εκτελέσει πραγματικές χαράξεις με σταθερή ακολουθία εντολών και χωρίς εμφανή λειτουργική αστοχία.

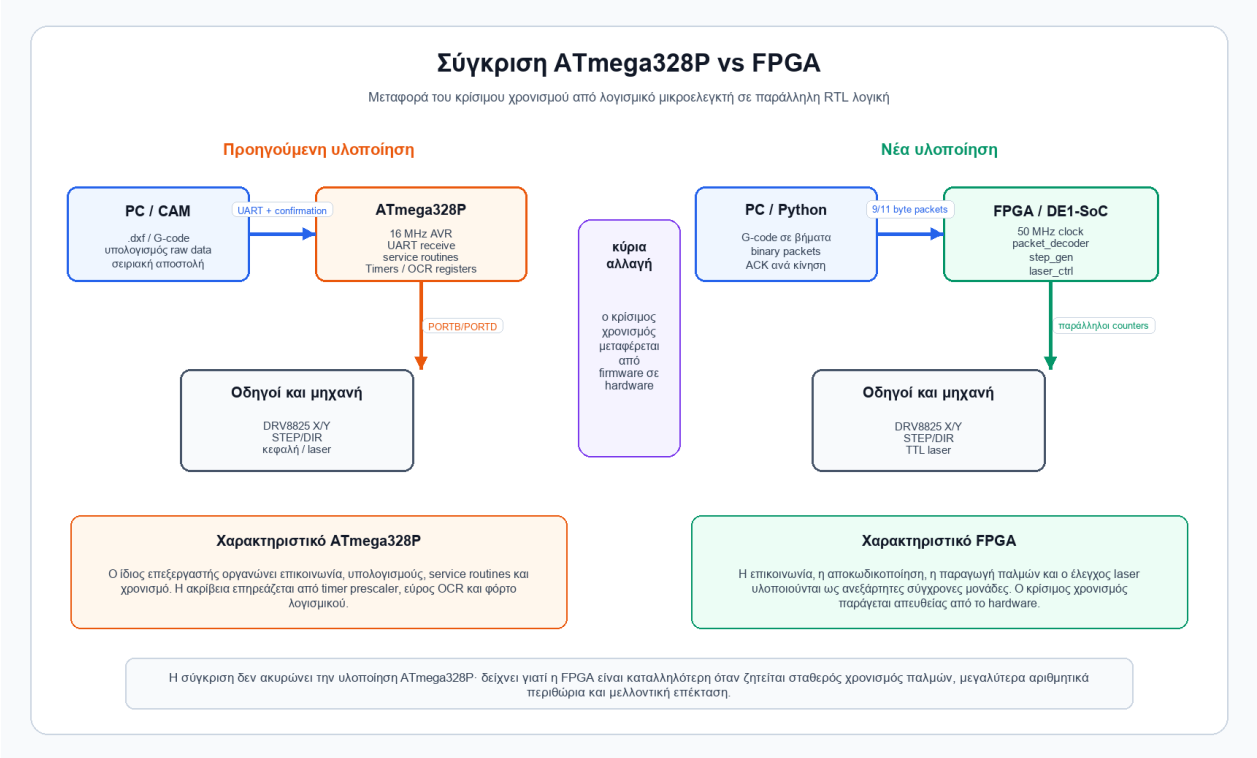
6.4 Σύγκριση ATmega328P vs FPGA

Η παρούσα ενότητα συγκρίνει την προηγούμενη προσέγγιση ελέγχου δύο αξόνων με μικροελεγκτή ATmega328P με τη νέα υλοποίηση που βασίζεται σε FPGA. Η σύγκριση δεν γίνεται για να αναιρεθεί η αξία της αρχικής λύσης, αλλά για να φανεί ποια τεχνικά σημεία βελτιώνονται όταν ο κρίσιμος χρονισμός των παλμών STEP μεταφέρεται από λογισμικό μικροελεγκτή σε σύγχρονη λογική υλοποιημένη σε hardware.

Στην υλοποίηση με ATmega328P ο μικροελεγκτής εκτελεί τις ρουτίνες κίνησης, χειρίζεται την επικοινωνία και χρησιμοποιεί τους εσωτερικούς timers για την παραγωγή παλμών. Στη νέα υλοποίηση, ο υπολογιστής εξακολουθεί να προετοιμάζει τα δεδομένα κίνησης, όμως η λήψη πακέτων, η

αποκωδικοποίηση, ο χρονισμός των παλμών και ο έλεγχος του laser εκτελούνται από ξεχωριστά RTL modules μέσα στο FPGA.

Η διαφορά αυτή είναι ουσιαστική για μια CNC εφαρμογή, επειδή η ποιότητα της κίνησης δεν εξαρτάται μόνο από το αν οι σωστοί παλμοί τελικά παράγονται, αλλά και από το πόσο σταθερά παράγονται χρονικά και πόσο εύκολα μπορεί να επεκταθεί η αρχιτεκτονική χωρίς να επιβαρυνθεί ο ίδιος επεξεργαστικός πυρήνας.



Σχήμα 6.12 *Αρχιτεκτονική σύγκριση της υλοποίησης με ATmega328P και της νέας υλοποίησης με FPGA.*

6.4.1 Βάση σύγκρισης

Στην προηγούμενη εργασία, ο ATmega328P ήταν το κεντρικό στοιχείο ελέγχου. Η πλακέτα Arduino UNO χρησιμοποιήθηκε ως αναπτυξιακή βάση, ενώ ειδικό shield συνέδεε τους οδηγούς των βηματικών κινητήρων και τα βοηθητικά σήματα. Η λογική εξόδου στηριζόταν στην άμεση χρήση των θυρών του μικροελεγκτή, όπως το PORTB για σήματα κατεύθυνσης, ενεργοποίησης και κεφαλής, και στη χρήση των timers και των compare registers για τη συχνότητα των STEP παλμών.

Στη νέα εργασία, το αντίστοιχο κέντρο ελέγχου είναι η πλακέτα DE1-SoC με FPGA Cyclone V. Η επικοινωνία γίνεται μέσω CP2102 και UART στα 115200 baud, ενώ τα δεδομένα δεν μεταφέρονται ως γραμμές κειμένου αλλά ως δυαδικά πακέτα. Το init packet μεταφέρει τους παράγοντες NUMERATOR_X και NUMERATOR_Y, ενώ κάθε motion packet μεταφέρει τύπο κίνησης, βήματα X/Y και πρόωση.

Επομένως, η σύγκριση αφορά κυρίως τη θέση όπου υλοποιείται ο χρονισμός. Στον ATmega328P ο χρονισμός είναι αποτέλεσμα προγραμματισμού timers, interrupt routines και χειρισμού καταχωρητών. Στο FPGA ο χρονισμός παράγεται από counters που λειτουργούν απευθείας στο ρολόι των 50 MHz, με ξεχωριστή μηχανή καταστάσεων για την εκτέλεση της κίνησης.

Πίνακας 6.17 *Συνοπτική αρχιτεκτονική σύγκριση ATmega328P και FPGA.*

Κριτήριο	ATmega328P	FPGA
Κεντρική λογική	Ένας 8-bit μικροελεγκτής εκτελεί πρόγραμμα, ρουτίνες και interrupts.	Πολλά RTL modules λειτουργούν παράλληλα μέσα στο FPGA.
Ρολόι / χρονισμός	16 MHz ρολόι AVR και timers με επιλογή prescaler.	50 MHz ρολόι συστήματος, δηλαδή χρονικό πλέγμα 20 ns.
Παραγωγή STEP	Οι παλμοί εξαρτώνται από timers, compare registers και service routines.	Οι παλμοί παράγονται από το step_gen με counters ανά άξονα.
Επικοινωνία	UART με raw data και επιβεβαίωση λήψης.	Binary protocol με init packet, motion packet και ACK 0xAA μετά την κίνηση.
Έλεγχος αξόνων	Ο μικροελεγκτής χειρίζεται οδηγούς μέσω ports και bit masks.	Το FPGA οδηγεί STEP/DIR για X/Y και TTL laser από GPIO_0.
Υλοποίηση τόξων	Τα τόξα προσεγγίζονται με μικρά γραμμικά τμήματα και ρουτίνες G02/G03.	Η προεπεξεργασία γίνεται στον host και το FPGA εκτελεί διαδοχικές κινήσεις βημάτων.
Επέκταση	Περιορίζεται από timers, διαθέσιμα pins και χρόνο CPU.	Μπορούν να προστεθούν νέα modules, όπως PWM laser ή πρόσθετη λογική ελέγχου.

6.4.2 Χρονισμός και αριθμητικά περιθώρια

Ένα από τα σημαντικότερα σημεία της προηγούμενης υλοποίησης ήταν ο περιορισμός των timers και των καταχωρητών. Στον ATmega328P η επιλογή prescaler επηρεάζει άμεσα την ανάλυση της ταχύτητας. Με μεγάλο prescaler μειώνεται η απαίτηση σε αριθμητικό εύρος, αλλά δίνει την δυνατότητα εκτέλεσης μικρών ταχύτητων. Με μικρό prescaler βελτιώνεται η ανάλυση, αλλά οι τιμές των compare registers μπορεί να ξεπεράσουν το διαθέσιμο εύρος.

Επιπλέον, η προηγούμενη ανάλυση έδειχνε ότι με 16-bit διαθέσιμο αριθμητικό χώρο για τη μετατόπιση και ρύθμιση 1/32 microstep, η μέγιστη μετατόπιση χωρίς overflow ήταν περίπου 409,6 mm. Η τιμή αυτή είναι κοντά στις διαστάσεις μιας μικρής μηχανής και δείχνει γιατί η αύξηση της ανάλυσης σε βήματα/mm δημιουργεί παράλληλα μεγαλύτερες απαιτήσεις στους καταχωρητές.

Στη νέα υλοποίηση τα steps_x και steps_y μεταφέρονται ως signed 32-bit τιμές. Αυτό δεν σημαίνει ότι η μηχανική ακρίβεια γίνεται αυτόματα καλύτερη, αλλά αφαιρεί έναν σημαντικό περιορισμό αριθμητικού εύρους από τον ελεγκτή. Παράλληλα, η χρήση ρολογιού 50 MHz δίνει χρονικό βήμα 20 ns για τον εσωτερικό χρονισμό των παλμών.

Η FPGA υλοποίηση χρησιμοποιεί ακόμη 100 κύκλους για το υψηλό επίπεδο του STEP, δηλαδή 2 μs, και 200 κύκλους αναμονής μετά την αλλαγή DIR, δηλαδή 4 μs. Οι τιμές αυτές παράγονται ντετερμινιστικά από το hardware και δεν εξαρτώνται από το αν ο host υπολογιστής καθυστερήσει να στείλει το επόμενο πακέτο.

Πίνακας 6.18 *Σύγκριση χρονισμού, αριθμητικού εύρους και διαθέσιμων πόρων.*

Παράμετρος	ATmega328P	FPGA	Σχόλιο
Βασικό ρολόι	16 MHz	50 MHz	Το FPGA παρέχει μικρότερο χρονικό βήμα για τον εσωτερικό χρονισμό.
Χρονική ανάλυση	Εξαρτάται από prescaler και timer.	20 ns ανά κύκλο ρολογιού.	Η τελική αξιοπιστία προϋποθέτει σωστό timing closure.
Εύρος μετατόπισης	Στην προηγούμενη λύση χρησιμοποιήθηκαν 16-bit καταχωρητές.	32-bit signed steps ανά άξονα.	Μειώνεται ο κίνδυνος overflow σε μεγάλες διαδρομές.
STEP high	Παράγεται μέσω λογισμικού/timer.	100 κύκλοι = 2 μs.	Σταθερή τιμή στην RTL υλοποίηση.
DIR settle	Ρυθμίζεται από τη ρουτίνα ελέγχου.	200 κύκλοι = 4 μs.	Εξασφαλίζεται πριν την πρώτη ακμή STEP.
Χρήση πόρων	Περιορισμένοι timers και pins στον μικροελεγκτή.	Σύνθεση: 2.172 / 32.070 ALMs (7%), 570 registers, 4 DSP blocks.	Η τρέχουσα FPGA υλοποίηση αφήνει σημαντικό περιθώριο επέκτασης.

6.4.3 Επίδραση στην κίνηση και στις δοκιμές

Στον ATmega328P η εκτέλεση των ρουτινών κίνησης γίνεται από τον ίδιο επεξεργαστή που πρέπει να χειρίζεται την κατάσταση της επικοινωνίας, τα flags των timers και τη λογική τερματισμού της κίνησης. Αυτό είναι πρακτικό για μια απλή και χαμηλού κόστους κατασκευή, αλλά όσο αυξάνονται οι απαιτήσεις σε ταχύτητα, ακρίβεια ή ταυτόχρονη λειτουργία, η σχεδίαση γίνεται πιο ευαίσθητη σε περιορισμούς λογισμικού.

Στο FPGA, η σειριακή επικοινωνία δεν παράγει άμεσα τους παλμούς STEP. Το packet_decoder συγκεντρώνει τα bytes, το step_gen εκτελεί την κίνηση και το laser_ctrl καθορίζει την έξοδο του laser. Μετά την έναρξη της κίνησης, οι counters του step_gen λειτουργούν αυτόνομα. Έτσι η καθυστέρηση του υπολογιστή ή της UART επηρεάζει κυρίως το κενό μεταξύ διαδοχικών κινήσεων και όχι τη χρονική σταθερότητα των παλμών μέσα στην ίδια κίνηση.

Για τις δοκιμές αυτό έχει πρακτική συνέπεια. Στην υλοποίηση ATmega328P η αξιολόγηση πρέπει να εξετάζει μαζί το λογισμικό, τους timers και την εκτέλεση των ρουτινών. Στην υλοποίηση FPGA η αξιολόγηση μπορεί να διαχωριστεί πιο καθαρά: επικοινωνία και ACK από τη μία πλευρά, χρονισμός STEP/DIR από την άλλη. Αυτό κάνει ευκολότερο τον εντοπισμό σφαλμάτων.

Πίνακας 6.19 *Επιπτώσεις της αρχιτεκτονικής επιλογής στις δοκιμές και στη συμπεριφορά της κίνησης.*

Περιοχή αξιολόγησης	ATmega328P	FPGA
Jitter παλμών	Μπορεί να επηρεαστεί από interrupt latency, flags και φόρτο ρουτίνας.	Προκύπτει κυρίως από το ρολόι, τους counters και το timing closure της σύνθεσης.
Ακρίβεια ταχύτητας	Επηρεάζεται έντονα από την εύρος των OCR τιμών.	Εξαρτάται από τη διαίρεση NUMERATOR / feed_rate και το χρονικό πλέγμα των 20 ns.
Συνδυασμένη κίνηση X-Y	Οι ταχύτητες υπολογίζονται ώστε η συνισταμένη να ακολουθεί την πρόωση, με περιορισμούς timer.	Το feed_rate προσαρμόζεται πριν την αποστολή και το step_gen συγχρονίζει τους δύο άξονες.
Καθυστέρηση επικοινωνίας	Μπορεί να μπλέκεται με την εκτέλεση ρουτίνας, ειδικά σε σύνθετη κίνηση.	Περιορίζεται κυρίως ανάμεσα σε διαδοχικά πακέτα, λόγω ACK.
Διαγνωστικός έλεγχος	Πιο εύκολη κατανόηση σε επίπεδο firmware, δυσκολότερη παρατήρηση εσωτερικού χρόνου.	Δυνατότητα παρατήρησης σημάτων STEP/DIR, busy, ACK και RTL καταστάσεων.

6.4.4 Πλεονεκτήματα και περιορισμοί

Το βασικό πλεονέκτημα του ATmega328P είναι η απλότητα. Η πλατφόρμα Arduino είναι οικονομική, εύκολη στην τροφοδοσία και κατάλληλη για γρήγορη ανάπτυξη. Η λογική γράφεται ως firmware και μπορεί να τροποποιηθεί χωρίς να απαιτείται γνώση HDL ή εργαλείων σύνθεσης. Για μια εκπαιδευτική ή πειραματική μηχανή χαμηλών απαιτήσεων, αυτό είναι σημαντικό πλεονέκτημα.

Το βασικό πλεονέκτημα του FPGA είναι ότι ο χρονισμός δεν ανταγωνίζεται τον υπόλοιπο κώδικα[4][6]. Η παραγωγή παλμών, η αποκωδικοποίηση πακέτων και ο έλεγχος εξόδων μπορούν να λειτουργούν παράλληλα. Επίσης, η αρχιτεκτονική μπορεί να επεκταθεί με πρόσθετα modules χωρίς να αυξηθεί αναλογικά ο φόρτος ενός κεντρικού επεξεργαστή.

Από την άλλη πλευρά, το FPGA απαιτεί αυστηρότερο σχεδιασμό. Η υλοποίηση πρέπει να ελέγχεται με simulation, μετρήσεις σε πραγματικά pins και έλεγχο timing από το Quartus/TimeQuest. Επομένως η FPGA λύση είναι ισχυρότερη ως αρχιτεκτονική, αλλά έχει μεγαλύτερη δυσκολία ανάπτυξης και αποσφαλμάτωσης.

Πίνακας 6.20 *Συνοπτική αξιολόγηση των δύο επιλογών ελέγχου*

Επιλογή	Πλεονεκτήματα	Περιορισμοί	Καταλληλότερη χρήση
ATmega328P	Χαμηλό κόστος, απλή ανάπτυξη, εύκολες αλλαγές firmware.	Περιορισμένοι timers, 8-bit/16-bit αριθμητικά όρια, μικρότερη δυνατότητα παράλληλης εκτέλεσης.	Απλές εκπαιδευτικές CNC διατάξεις και χαμηλές ταχύτητες.

FPGA	Παράλληλος χρονισμός, 32-bit βήματα, καθαρό πρωτόκολλο πακέτων, δυνατότητα επέκτασης.	Μεγαλύτερη πολυπλοκότητα, ανάγκη HDL, simulation και timing analysis.	Συστήματα όπου η σταθερότητα παλμών και η επεκτασιμότητα είναι κεντρικές απαιτήσεις.

6.4.5 Συμπέρασμα σύγκρισης

Η μετάβαση από ATmega328P σε FPGA μεταβάλλει τον χαρακτήρα του ελεγκτή. Η προηγούμενη λύση ήταν περισσότερο firmware-based: ο μικροελεγκτής εκτελούσε διαδοχικά τις λειτουργίες επικοινωνίας, υπολογισμού και ελέγχου παλμών. Η νέα λύση είναι hardware-based: κάθε κρίσιμη λειτουργία περιγράφεται ως ξεχωριστή λογική μονάδα και εκτελείται παράλληλα.

Για τη συγκεκριμένη εργασία, η επιλογή FPGA είναι τεχνικά δικαιολογημένη επειδή δίνει καθαρότερο χρονισμό STEP/DIR, μεγαλύτερο αριθμητικό περιθώριο για τα βήματα, καλύτερη διάκριση ανάμεσα σε επικοινωνία και εκτέλεση κίνησης και δυνατότητα μελλοντικής επέκτασης. Η πραγματική γεωμετρική ακρίβεια εξακολουθεί να εξαρτάται από τη μηχανική κατασκευή, τους ιμάντες, τους οδηγούς και τη ρύθμιση steps/mm, όμως ο ελεγκτής FPGA μειώνει τους περιορισμούς που προέρχονται από τον ίδιο τον ψηφιακό χρονισμό.

Κεφάλαιο 7ο: Συμπεράσματα και μελλοντικές επεκτάσεις

7.1 Συμπεράσματα

Η παρούσα εργασία είχε ως αντικείμενο την ανάπτυξη ενός ελεγκτή μηχανής CNC δύο αξόνων με χρήση FPGA, με στόχο την εκτέλεση κινήσεων που παράγονται από σχέδιο CAD και μετατρέπονται σε παλμούς για βηματικούς κινητήρες και TTL laser. Το τελικό αποτέλεσμα δεν περιορίζεται στην παραγωγή ενός μεμονωμένου κυκλώματος, αλλά περιλαμβάνει ολόκληρη ροή εργασίας: σχεδίαση στο AutoCAD, παραγωγή και επεξεργασία G-code, μετατροπή σε βήματα, σειριακή αποστολή πακέτων και εκτέλεση της κίνησης από το FPGA.

Η εργασία μπορεί να θεωρηθεί εξέλιξη της προγενέστερης εργασίας “Two Axis CNC - Hardware and Software Development”, στην οποία η κίνηση υλοποιούνταν με μικροελεγκτή ATmega328P. Η προηγούμενη υλοποίηση απέδειξε ότι είναι δυνατή η κατασκευή και ο έλεγχος μιας απλής CNC διάταξης δύο αξόνων, αλλά ανέδειξε και σημαντικούς περιορισμούς: περιορισμένο εύρος καταχωρητών για τη μετατόπιση, συμβιβασμούς μεταξύ prescaler, ανάλυσης ταχύτητας και microstep, καθώς και αυξημένη πολυπλοκότητα λογισμικού για τον ακριβή χειρισμό των χρονισμών.

Με βάση αυτά τα συμπεράσματα, η νέα υλοποίηση μεταφέρει τις χρονικά κρίσιμες λειτουργίες από το λογισμικό μικροελεγκτή σε ψηφιακή λογική FPGA. Η παραγωγή STEP, DIR και laser γίνεται με κοινό ρολόι 50 MHz και με καταστάσεις υλικού, ενώ ο υπολογιστής αναλαμβάνει τις εργασίες που είναι πιο κατάλληλες για λογισμικό, όπως η γεωμετρική επεξεργασία, η οργάνωση της διαδρομής και ο υπολογισμός των αριθμητικών παραμέτρων.

7.1.1 Κύρια τεχνικά συμπεράσματα

Το βασικό συμπέρασμα είναι ότι η χρήση FPGA είναι κατάλληλη για τον συγκεκριμένο τύπο ελεγκτή, επειδή παρέχει ντετερμινιστικό χρονισμό χωρίς εξάρτηση από interrupts, polling βρόχους ή καθυστερήσεις εκτέλεσης εντολών. Στην παλαιότερη ATmega328P προσέγγιση, η ακρίβεια της ταχύτητας συνδεόταν άμεσα με τους διαθέσιμους timer registers και τις επιλογές prescaler. Στην παρούσα προσέγγιση, η περίοδος των παλμών προκύπτει από μετρητές μεγαλύτερου εύρους μέσα στο FPGA, άρα η σχεδίαση δεν περιορίζεται με τον ίδιο τρόπο από 8-bit compare registers.

Επίσης, η χρήση signed 32-bit πεδίων για τα βήματα X και Y και 32-bit αριθμητικών παραμέτρων NUMERATOR_X και NUMERATOR_Y μειώνει σημαντικά τον κίνδυνο υπερχείλισης που εμφανιζόταν στην προγενέστερη εργασία, όπου το μέγιστο μήκος μετατόπισης εξαρτιόταν άμεσα από το διαθέσιμο πλήθος bits του displacement register και από το επιλεγμένο microstep. Έτσι, η νέα αρχιτεκτονική είναι πιο άνετη αριθμητικά και μπορεί να προσαρμοστεί ευκολότερα σε διαφορετικές μηχανικές ρυθμίσεις.

Η απόφαση να μη μεταφερθεί αυτούσιος ASCII G-code στο FPGA αποδείχθηκε επίσης σωστή. Η αποκωδικοποίηση κειμένου, η ανάλυση αριθμών και η γεωμετρική επεξεργασία θα αύξαναν σημαντικά την πολυπλοκότητα του hardware. Αντίθετα, η αποστολή δυαδικών πακέτων σταθερού μήκους επιτρέπει στο FPGA να λειτουργεί ως καθαρός ελεγκτής εκτέλεσης: λαμβάνει αριθμητικά δεδομένα κίνησης, τα αποκωδικοποιεί και παράγει παλμούς.

Πίνακας 7.1 Βασικά αποτελέσματα της παρούσας υλοποίησης.

Στόχος	Υλοποίηση	Συμπέρασμα
Παραγωγή διαδρομής από CAD	Ανάπτυξη εργαλείων AutoLISP/OpenDCL για επιλογή γεωμετρίας, παραγωγή G-code και οργάνωση διαδρομής.	Το AutoCAD μπορεί να λειτουργήσει ως CAM περιβάλλον για τη συγκεκριμένη κατασκευή.
Μετατροπή σε παλμούς	Υπολογισμός signed βημάτων X/Y, πρόωσης και παραμέτρων NUMERATOR πριν από την αποστολή στο FPGA.	Η σύνθετη γεωμετρική επεξεργασία γίνεται στον host, όπου είναι ευκολότερη η ανάπτυξη και ο έλεγχος.
Επικοινωνία	Δυαδικά πακέτα αρχικοποίησης και κίνησης μέσω UART/CP2102 με ACK 0xAA.	Η ροή δεδομένων είναι απλή, ελέγξιμη και κατάλληλη για εργαστηριακή διάταξη.
Παραγωγή STEP/DIR	Υλοποίηση step_gen με καταστάσεις IDLE, SETUP, DIR_WAIT, RUN και DONE.	Οι παλμοί παράγονται με χρονισμό υλικού και όχι με καθυστερήσεις λογισμικού.
Έλεγχος laser	Ενεργοποίηση laser με βάση τον τύπο κίνησης και το busy της εκτέλεσης.	Το laser συνδέεται λογικά με την πραγματική κίνηση κατεργασίας.

7.1.2 Σύγκριση με την προγενέστερη εργασία

Η προγενέστερη εργασία είχε ιδιαίτερη αξία επειδή ανέδειξε πρακτικά προβλήματα που δεν φαίνονται πάντα σε θεωρητική περιγραφή CNC ελέγχου. Για παράδειγμα, στα συμπεράσματά της αναφέρεται ότι με microstep 1/32 και 16-bit displacement register το μέγιστο υποστηριζόμενο μήκος είναι 409,6 mm, ενώ η μηχανική διαδρομή της κατασκευής ήταν 460 mm. Αυτό οδηγούσε σε πιθανότητα υπερχείλισης και σε ανάγκη αύξησης των καταχωρητών ή της πολυπλοκότητας του κώδικα.

Αντίστοιχα, στο τμήμα για την ακρίβεια ταχύτητας, η προηγούμενη εργασία έδειχνε ότι η επιλογή timer prescaler επηρεάζει άμεσα την απόκλιση της πραγματικής ταχύτητας από την επιθυμητή. Με μεγάλο prescaler, η τιμή του compare register μπορεί να γίνει πολύ μικρή και η αποκοπή του δεκαδικού μέρους να προκαλέσει σημαντικό σφάλμα. Με μικρότερο prescaler η ακρίβεια βελτιώνεται, αλλά απαιτούνται μεγαλύτερα registers και πιο προσεκτικός χειρισμός.

Η παρούσα FPGA υλοποίηση αντιμετωπίζει αυτά τα σημεία όχι με επιμέρους διορθώσεις στον κώδικα μικροελεγκτή, αλλά με διαφορετική αρχιτεκτονική. Η παραγωγή παλμών γίνεται από μετρητές υλικού, τα πεδία κίνησης έχουν μεγαλύτερο εύρος και οι κρίσιμες λειτουργίες δεν εξαρτώνται από τη διαδοχική εκτέλεση assembly ρουτινών. Με άλλα λόγια, η FPGA προσέγγιση μειώνει την ανάγκη για σύνθετα τεχνάσματα λογισμικού στον χρονισμό[2][6].

Πίνακας 7.2 Σύγκριση προγενέστερης ATmega328P υλοποίησης και παρούσας FPGA υλοποίησης.

Θέμα	Προγενέστερη εργασία ATmega328P	Παρούσα εργασία FPGA
Χρονισμός παλμών	Βασισμένος σε timers, compare registers και prescaler.	Βασισμένος σε μετρητές υλικού με ρολόι 50 MHz.
Εύρος μετατόπισης	Περιορισμός από το διαθέσιμο πλήθος bits του displacement register.	Χρήση 32-bit signed βημάτων για κάθε άξονα.
Ακρίβεια ταχύτητας	Επηρεάζεται έντονα από την επιλογή prescaler και το μέγεθος των OCR registers.	Η περίοδος STEP υλοποιείται ως ακέραιος αριθμός κύκλων FPGA με μεγαλύτερη αριθμητική ευχέρεια.
Υπολογιστική πολυπλοκότητα	Μεγάλο μέρος του ελέγχου απαιτεί προσεκτικές ρουτίνες λογισμικού/assembly.	Οι χρονικά κρίσιμες λειτουργίες υλοποιούνται ως παράλληλη ψηφιακή λογική.
Ροή δεδομένων	Πιο άμεση σύνδεση λογισμικού μικροελεγκτή με την κίνηση.	Διαχωρισμός host υπολογισμών, UART πακέτων και FPGA εκτέλεσης.
Επεκτασιμότητα	Απαιτεί αλλαγές σε timers, registers, PCB ή assembly κώδικα.	Μπορεί να επεκταθεί με νέα Verilog modules, buffers ή επιπλέον λογική ελέγχου.

7.1.3 Συμπεράσματα ανά υποσύστημα

Στο επίπεδο του λογισμικού, η εργασία έδειξε ότι το AutoLISP μπορεί να χρησιμοποιηθεί αποτελεσματικά για την ανάπτυξη εξειδικευμένων CAM λειτουργιών μέσα στο AutoCAD. Ο χρήστης παραμένει στο περιβάλλον σχεδίασης, επιλέγει γεωμετρικά στοιχεία, ορίζει παραμέτρους και παράγει την ακολουθία κίνησης χωρίς να απαιτείται ανεξάρτητο CAM πρόγραμμα.

Στο επίπεδο της επικοινωνίας, η χρήση CP2102 και UART στα 115200 baud αποδείχθηκε επαρκής για τη μετάδοση των μικρών δυαδικών πακέτων. Ο μηχανισμός ACK μετά την ολοκλήρωση της κίνησης απλοποιεί τη ροή και εξασφαλίζει ότι ο υπολογιστής δεν προωθεί νέα εντολή πριν ολοκληρωθεί η προηγούμενη. Η επιλογή αυτή είναι συντηρητική, αλλά ταιριάζει στον εργαστηριακό χαρακτήρα του συστήματος.

Στο επίπεδο του FPGA, τα modules uart_rx, uart_tx, packet_decoder, step_gen, laser_ctrl και top σχηματίζουν μια καθαρή αλυσίδα επεξεργασίας. Κάθε module έχει συγκεκριμένο ρόλο και περιορισμένη διεπαφή. Αυτό κάνει τη σχεδίαση πιο κατανοητή και ευκολότερη στην επαλήθευση, σε σύγκριση με μια μονολιθική υλοποίηση όπου η λήψη, η ερμηνεία και η εκτέλεση της κίνησης συγχωνεύονται.

Στο επίπεδο της μηχανής, οι πραγματικές δοκιμές έδειξαν ότι η αλυσίδα από το σχέδιο μέχρι τη χάραξη μπορεί να λειτουργήσει πρακτικά. Παρόλα αυτά, η γεωμετρική ακρίβεια δεν εξαρτάται αποκλειστικά από το FPGA. Παράγοντες όπως η τάνυση των ιμάντων, η ευθυγράμμιση των αξόνων, η συμπεριφορά των οδηγών, η εστίαση του laser και το υλικό του δοκιμίου παραμένουν κρίσιμοι.

7.1.4 Περιορισμοί της παρούσας υλοποίησης

Παρότι η FPGA υλοποίηση αντιμετωπίζει σημαντικούς περιορισμούς της προηγούμενης μικροελεγκτικής προσέγγισης, δεν αποτελεί πλήρες βιομηχανικό CNC controller. Η ροή δεδομένων είναι σειριακή και βασίζεται σε stop-and-wait ACK, επομένως δεν υπάρχει συνεχές streaming με εσωτερικό buffer πολλών κινήσεων. Η επιλογή αυτή απλοποιεί τον έλεγχο, αλλά μπορεί να δημιουργεί μικρό νεκρό χρόνο ανάμεσα σε διαδοχικά segment.

Επιπλέον, η εργασία δεν υλοποιεί πλήρη σχεδιασμό επιτάχυνσης και επιβράδυνσης. Οι κινήσεις εκτελούνται με απλοποιημένη λογική πρόωσης, κάτι που είναι επαρκές για τις δοκιμές και τη βασική χάραξη, αλλά θα μπορούσε να βελτιωθεί σε εφαρμογές όπου απαιτείται υψηλότερη ταχύτητα, μικρότερη καταπόνηση της μηχανής ή καλύτερη ποιότητα στις γωνίες.

Τέλος, οι μετρήσεις jitter παραμένουν κυρίως θεωρητικές, με βάση το ρολόι των 50 MHz και τη λογική παραγωγής παλμών. Για πλήρη πειραματική τεκμηρίωση απαιτείται καταγραφή των σημάτων STEP με παλμογράφο ή λογικό αναλυτή και συσχέτισή τους με τις πραγματικές γεωμετρικές αποκλίσεις πάνω στο δοκίμιο.

Πίνακας 7.3 Τελική αποτίμηση δυνατοτήτων και περιορισμών.

Πεδίο	Τι επιτεύχθηκε	Τι παραμένει για βελτίωση
Αρχιτεκτονική	Καθαρός διαχωρισμός host υπολογισμών και FPGA εκτέλεσης.	Πιθανή προσθήκη buffer και streaming πολλών πακέτων.
Χρονισμός	Ντετερμινιστική παραγωγή STEP/DIR σε λογική υλικού.	Πειραματική μέτρηση jitter με όργανα.
Κίνηση	Εκτέλεση γραμμικών και καμπύλων διαδρομών σε πραγματική μηχανή.	Προσθήκη επιτάχυνσης, επιβράδυνσης και πιο ομαλής μετάβασης στις γωνίες.

7.1.5 Γενικό συμπέρασμα

Συνολικά, η εργασία επιβεβαιώνει ότι η μετάβαση από μια μικροελεγκτική προσέγγιση σε μια FPGA προσέγγιση είναι ουσιαστική και όχι μόνο αλλαγή πλατφόρμας. Το FPGA δεν χρησιμοποιείται απλώς ως διαφορετικός επεξεργαστής, αλλά ως ψηφιακό κύκλωμα που αναλαμβάνει ακριβώς το μέρος του CNC ελέγχου όπου απαιτείται σταθερός χρονισμός. Με αυτόν τον τρόπο μειώνονται οι περιορισμοί που σχετίζονται με μικρούς timer registers, prescalers και σειριακή εκτέλεση κώδικα.

Η προγενέστερη εργασία έθεσε τη βάση: απέδειξε τη λειτουργικότητα της μηχανής, ανέδειξε τα προβλήματα ακρίβειας και πρότεινε κατευθύνσεις βελτίωσης. Η παρούσα εργασία αξιοποίησε αυτή τη βάση και μετέφερε την κρίσιμη παραγωγή παλμών σε FPGA, διατηρώντας τον υπολογιστή ως επίπεδο γεωμετρικής επεξεργασίας. Έτσι προκύπτει μια πιο καθαρή και επεκτάσιμη αρχιτεκτονική.

Το τελικό σύστημα αποτελεί λειτουργικό εκπαιδευτικό παράδειγμα συνδυασμού CAD/CAM, σειριακής επικοινωνίας, ψηφιακού σχεδιασμού και μηχανικού ελέγχου. Παρότι υπάρχουν περιθώρια εξέλιξης, η εργασία δείχνει ότι ένας FPGA controller μπορεί να αποτελέσει αξιόπιστη βάση για CNC laser εφαρμογές δύο αξόνων, ιδίως όταν ο σχεδιασμός διαχωρίζει σωστά τις εργασίες λογισμικού από τις εργασίες πραγματικού χρόνου.

7.2 Μελλοντικές επεκτάσεις

Η παρούσα υλοποίηση απέδειξε ότι ένας ελεγκτής CNC δύο αξόνων μπορεί να βασιστεί σε FPGA για την παραγωγή παλμών STEP/DIR και τον έλεγχο του laser. Παρόλα αυτά, η εργασία μπορεί να αποτελέσει βάση για αρκετές επεκτάσεις, τόσο στο λογισμικό όσο και στο hardware. Οι επεκτάσεις αυτές δεν έχουν όλες την ίδια προτεραιότητα: ορισμένες βελτιώνουν άμεσα την ποιότητα της κίνησης, άλλες αυξάνουν την ασφάλεια και άλλες στοχεύουν στη μείωση κόστους και μεγέθους του τελικού ελεγκτή.

Ένα σημαντικό συμπέρασμα από την προγενέστερη εργασία “Two Axis CNC - Hardware and Software Development” είναι ότι η ακρίβεια και η αξιοπιστία του ελέγχου δεν εξαρτώνται μόνο από τον αλγόριθμο, αλλά και από την πλατφόρμα υλοποίησης. Στην υλοποίηση με ATmega328P, οι timers, οι prescalers και το διαθέσιμο εύρος καταχωρητών δημιουργούσαν συγκεκριμένους περιορισμούς. Η μετάβαση σε FPGA αντιμετωπίζει μέρος αυτών των περιορισμών, αλλά ανοίγει και νέες κατευθύνσεις βελτίωσης.

7.2.1 Βελτίωση της κίνησης

Η σημαντικότερη τεχνική επέκταση είναι η προσθήκη προφίλ επιτάχυνσης και επιβράδυνσης. Στην παρούσα μορφή η κίνηση εκτελείται με απλοποιημένη λογική σταθερής πρόωσης ανά segment. Για μικρές εργαστηριακές δοκιμές αυτό είναι επαρκές, όμως σε μεγαλύτερες ταχύτητες μπορεί να προκαλέσει απότομες μεταβολές, απώλεια βημάτων ή αυξημένη καταπόνηση στους ιμάντες και στους κινητήρες.

Μια μελλοντική υλοποίηση θα μπορούσε να χρησιμοποιήσει τραπεζοειδές προφίλ ταχύτητας ή S-curve προφίλ. Με αυτόν τον τρόπο η συχνότητα των παλμών STEP δεν θα αλλάζει απότομα, αλλά θα αυξάνεται και θα μειώνεται σταδιακά. Η αλλαγή αυτή θα μπορούσε να υλοποιηθεί είτε στον

υπολογιστή, με διάσπαση κάθε κίνησης σε περισσότερα μικρότερα τμήματα, είτε μέσα στο FPGA, με δυναμική μεταβολή της περιόδου των παλμών.

Μια δεύτερη επέκταση αφορά την ομαλότερη ένωση διαδοχικών segment. Σήμερα η ροή είναι σειριακή και κάθε κίνηση ολοκληρώνεται πριν σταλεί η επόμενη. Μελλοντικά μπορεί να προστεθεί μικρή ουρά εντολών στο FPGA, ώστε να μειωθεί ο νεκρός χρόνος ανάμεσα σε συνεχόμενα τμήματα και να γίνει εφικτή πιο ομαλή χάραξη σύνθετων καμπυλών.

7.2.2 Επέκταση επικοινωνίας και πρωτοκόλλου

Ο μηχανισμός ACK 0xAA είναι απλός και αξιόπιστος, όμως περιορίζει τη ροή σε stop-and-wait εκτέλεση. Μια μελλοντική επέκταση θα μπορούσε να εισάγει FIFO buffer πακέτων στο FPGA. Σε αυτή την περίπτωση ο υπολογιστής θα μπορούσε να στέλνει περισσότερες εντολές εκ των προτέρων, ενώ το FPGA θα τις εκτελεί με τη σειρά. Η προσέγγιση αυτή θα μείωνε τις καθυστερήσεις μεταξύ segment, αλλά θα απαιτούσε πιο σύνθετη διαχείριση πληρότητας buffer, σφαλμάτων και επανεκκίνησης μετά από διακοπή.

Επίσης, το πρωτόκολλο μπορεί να επεκταθεί με πεδίο checksum ή CRC, ώστε να ανιχνεύονται αλλοιώσεις πακέτων. Στην παρούσα εργαστηριακή εφαρμογή η σειριακή σύνδεση είναι μικρού μήκους και ελεγχόμενη, επομένως η απλή μορφή του πακέτου είναι επαρκής. Σε πιο απαιτητικό περιβάλλον, όμως, ένας έλεγχος ακεραιότητας θα έκανε την επικοινωνία πιο ασφαλή.

7.2.3 Βελτίωση γεωμετρικής ταύτισης και βαθμονόμησης

Η σημερινή γεωμετρική ταύτιση βασίζεται κυρίως στη μεταφορά της αρχής του σχεδίου στο σημείο που δηλώνει ο χρήστης με τη λειτουργία SetOrigin. Μελλοντικά θα μπορούσε να προστεθεί πιο πλήρης βαθμονόμηση, με χρήση δύο ή περισσότερων σημείων αναφοράς πάνω στο δοκίμιο. Έτσι θα ήταν δυνατή η εκτίμηση όχι μόνο της μετατόπισης, αλλά και μικρής περιστροφής ή κλίμακας του τοποθετημένου τεμαχίου.

Παράλληλα, θα ήταν χρήσιμη η ανάπτυξη διαδικασίας αυτόματης βαθμονόμησης steps/mm. Η μηχανή θα μπορούσε να εκτελεί γνωστές μετατοπίσεις, ο χρήστης να εισάγει τη μετρούμενη απόσταση και το λογισμικό να υπολογίζει διορθωμένες τιμές για τους άξονες X και Y. Οι διορθωμένες τιμές θα αποστέλλονται στο FPGA μέσω του init packet, αξιοποιώντας τους ήδη υπάρχοντες NUMERATOR_X και NUMERATOR_Y.

7.2.4 Ασφάλεια και πραγματική χρήση laser

Για χρήση πέρα από εργαστηριακή δοκιμή, η ασφάλεια του laser πρέπει να ενισχυθεί. Μια μελλοντική πλακέτα ελέγχου θα πρέπει να διαθέτει είσοδο emergency stop, είσοδο interlock για κάλυμμα ή θάλαμο προστασίας, και ανεξάρτητο κύκλωμα ενεργοποίησης laser. Έτσι, ακόμη και αν το FPGA ή το λογισμικό παρουσιάσει σφάλμα, το laser θα μπορεί να απενεργοποιηθεί από εξωτερική ασφάλεια.

Επίσης, μπορεί να προστεθεί PWM έλεγχος ισχύος laser. Η τρέχουσα βασική υλοποίηση χειρίζεται το laser ως ON/OFF έξοδο. Με PWM θα ήταν δυνατή η ρύθμιση έντασης ανάλογα με την ταχύτητα, το υλικό ή το είδος χάραξης. Η δυνατότητα αυτή θα ήταν ιδιαίτερα χρήσιμη για raster engraving ή για χάραξη σε υλικά με διαφορετική απορρόφηση.

Πίνακας 7.4 Προτεινόμενες μελλοντικές επεκτάσεις ανά περιοχή του συστήματος.

Περιοχή	Προτεινόμενη επέκταση	Αναμενόμενο όφελος
Κίνηση	Προσθήκη επιτάχυνσης/επιβράδυνσης.	Πιο ομαλή κίνηση, μικρότερη πιθανότητα απώλειας βημάτων.
Ροή εντολών	FIFO buffer στο FPGA.	Μικρότερος νεκρός χρόνος μεταξύ διαδοχικών segment.
Επικοινωνία	Checksum ή CRC στα πακέτα.	Ανίχνευση αλλοίωσης δεδομένων και ασφαλέστερη μετάδοση.
Γεωμετρία	Ταύτιση με περισσότερα σημεία αναφοράς.	Διόρθωση μετατόπισης και πιθανής μικρής περιστροφής δοκιμίου.
Βαθμονόμηση	Αυτόματος υπολογισμός steps/mm.	Μικρότερη γεωμετρική απόκλιση ανά άξονα.
Laser	PWM έλεγχος ισχύος.	Καλύτερος έλεγχος έντασης χάραξης και δυνατότητα grayscale.
Ασφάλεια	Emergency stop, interlock και ανεξάρτητο laser enable.	Ασφαλέστερη λειτουργία σε πραγματικές συνθήκες.

7.2.5 Εναλλακτική χαμηλού κόστους FPGA πλατφόρμα

Η DE1-SoC είναι πολύ χρήσιμη εκπαιδευτική πλατφόρμα, επειδή προσφέρει άνετους πόρους, πολλά περιφερειακά και εύκολη πρόσβαση στα GPIO. Για τελική όμως κατασκευή CNC controller είναι υπερβολικά μεγάλη και ακριβή. Μια πρακτική μελλοντική επέκταση είναι η μεταφορά της υλοποίησης σε μικρότερο και οικονομικότερο FPGA chip.

Μια κατάλληλη πρόταση είναι το Gowin GW1N-9 ή η πολύ κοντινή παραλλαγή GW1NR-9, η οποία χρησιμοποιείται σε οικονομικές πλακέτες όπως η Tang Nano 9K. Το chip διαθέτει 8.640 LUT4, 6.480 flip-flops, 468 Kbit block SRAM, 20 πολλαπλασιαστές 18x18 και δύο PLL. Οι πόροι αυτοί είναι πολύ περισσότεροι από όσους απαιτούνται για UART, packet decoder, step generator δύο αξόνων, laser control και βασικά debug σήματα.

Η επιλογή αυτή δεν σημαίνει ότι το υπάρχον project μεταφέρεται χωρίς καμία αλλαγή. Θα απαιτηθεί νέα χαρτογράφηση ακίδων, προσαρμογή των constraints, έλεγχος τάσεων I/O, αλλαγή των παραμέτρων χρονισμού αν το ρολόι δεν είναι 50 MHz, και σύνθεση με το εργαλείο Gowin EDA αντί για Quartus. Παρόλα αυτά, επειδή η υλοποίηση είναι γραμμένη σε απλή Verilog και δεν βασίζεται σε ειδικό IP της Intel/Altera, η μεταφορά είναι ρεαλιστική.

Για ακόμη χαμηλότερο ή πιο ανοικτό οικοσύστημα μπορεί να εξεταστεί και η οικογένεια Lattice iCE40 UltraPlus, όπως το iCE40UP5K. Διαθέτει λιγότερους λογικούς πόρους από το GW1N-9, αλλά μπορεί να είναι επαρκές αν απλοποιηθεί η αριθμητική στο FPGA ή αν περισσότερος υπολογισμός περιόδων γίνει στον υπολογιστή. Για την παρούσα υλοποίηση, όμως, το GW1N-9/GW1NR-9 είναι πιο άνετη πρόταση, επειδή παρέχει μεγαλύτερο περιθώριο πόρων με πολύ μικρό κόστος πλακέτας ανάπτυξης.

Πίνακας 7.5 Αξιολόγηση φθηνότερης FPGA πλατφόρμας για μελλοντική υλοποίηση.

Κριτήριο	DE1-SoC / Cyclone V	Gowin GW1N-9 / GW1NR-9
Ρόλος	Εκπαιδευτική και πλούσια πλατφόρμα ανάπτυξης.	Μικρό και φθηνό FPGA για ειδικό CNC controller.
Λογικοί πόροι	Πολύ περισσότεροι από όσους απαιτεί η εφαρμογή.	8.640 LUT4 και 6.480 FF, επαρκή για UART, decoder, step_gen και laser.
Μνήμη	Μεγάλο περιθώριο για πειραματισμό.	468 Kbit block SRAM, αρκετή για μικρούς buffers ή FIFO.
Χρονισμός	50 MHz ρολόι στην υπάρχουσα υλοποίηση.	Απαιτεί προσαρμογή CLK_HZ και σταθερών αν χρησιμοποιηθεί διαφορετικό ρολόι.

Κόστος/μέγεθος	Μεγάλη και ακριβή πλακέτα για τελική κατασκευή.	Πολύ πιο μικρή και οικονομική λύση, κατάλληλη για αυτόνομο controller.
Μεταφορά project	Υπάρχουσα Quartus υλοποίηση.	Απαιτεί port σε Gowin EDA, νέα pin constraints και έλεγχο timing.

Ως εναλλακτική χαμηλού κόστους συμβατή με Quartus μπορεί να χρησιμοποιηθεί μικρή πλακέτα Intel/Altera Cyclone IV, όπως EP4CE6 ή EP4CE10, η οποία διαθέτει επαρκείς λογικούς πόρους για την παρούσα εφαρμογή. Εναλλακτικά, μπορεί να χρησιμοποιηθεί πλακέτα MAX 10, όπως η 10M08 Evaluation Kit, με την προϋπόθεση χρήσης νεότερης έκδοσης Quartus Prime Lite.

Ως οικονομική εναλλακτική για τελική ενσωματωμένη υλοποίηση μπορεί να χρησιμοποιηθεί FPGA της οικογένειας Intel MAX 10, όπως το 10M02. Το συγκεκριμένο chip διαθέτει περίπου 2000 logic elements, εσωτερική μνήμη διαμόρφωσης και επαρκή αριθμό ακίδων I/O για UART, σήματα STEP/DIR δύο αξόνων και έλεγχο TTL laser. Ειδικά η έκδοση 10M02SCE144C8G σε πακέτο EQFP είναι κατάλληλη για custom πλακέτα ελεγκτή, καθώς αποφεύγει την πολυπλοκότητα ενός BGA πακέτου και δεν απαιτεί ξεχωριστή configuration flash.

7.2.6 Συνολική κατεύθυνση εξέλιξης

Η βασική κατεύθυνση για μελλοντική εξέλιξη είναι να παραμείνει ο διαχωρισμός ρόλων που αποδείχθηκε λειτουργικός: ο υπολογιστής να αναλαμβάνει τη γεωμετρική επεξεργασία και το FPGA να αναλαμβάνει τον ακριβή χρονισμό. Οι επεκτάσεις πρέπει να ενισχύουν αυτό το μοντέλο και όχι να μεταφέρουν άσκοπη πολυπλοκότητα στο hardware.

Με αυτή τη λογική, οι πιο άμεσες βελτιώσεις είναι η προσθήκη επιτάχυνσης, η καλύτερη πειραματική μέτρηση jitter και η ενσωμάτωση περισσότερων μηχανισμών ασφάλειας. Σε επόμενο στάδιο, η μεταφορά σε φθηνότερο FPGA όπως το GW1N-9/GW1NR-9 θα μπορούσε να μετατρέψει την παρούσα εργαστηριακή υλοποίηση σε μικρό, οικονομικό και πιο πρακτικό controller για πραγματική χρήση.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] P. K. Iakovou, “Two Axis CNC - Hardware and Software Development Using ATmega328P Microcontroller,” Academia.edu. Accessed: May 5, 2026. [Online]. Available: https://www.academia.edu/127042145.

[2] F. Santos, J. Ferreira, and M. Silva, “FPGA-based motion control system for CNC machines,” in Proc. 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Porto, Portugal, 2018, pp. 388-395, doi: 10.5220/0006974403880395.

[3] Y. Zhang, X. Li, and H. Wang, “Design of CNC system based on FPGA technology,” in Proc. International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2015), Jinan, China, 2015, pp. 920-925, doi: 10.2991/amcce-15.2015.168.

[4] D. Carrica, M. A. Funes, and S. A. Gonzalez, “Novel stepper motor controller based on FPGA hardware implementation,” IEEE/ASME Transactions on Mechatronics, vol. 8, no. 1, pp. 120-124, Mar. 2003, doi: 10.1109/TMECH.2003.809160.

[5] J. W. Jeon and Y. G. Kim, “FPGA based acceleration and deceleration circuit for industrial robots and CNC machine tools,” Mechatronics, vol. 12, no. 4, pp. 635-642, May 2002, doi: 10.1016/S0957-4158(01)00012-5.

[6] J. U. Cho, Q. N. Le, and J. W. Jeon, “An FPGA-Based Multiple-Axis Motion Control Chip,” IEEE Transactions on Industrial Electronics, vol. 56, no. 3, pp. 856-870, Mar. 2009, doi: 10.1109/TIE.2008.2004671.

[7] M. A. Saifee and U. S. Mehta, “Design and implementation of FPGA based G code compatible CNC lathe controller,” International Journal of Electronics and Communication Engineering & Technology, vol. 7, no. 1, pp. 87-100, 2016.

[8] J. Fei, R. Deng, Z. Zhang, and M. Zhou, “Research on embedded CNC device based on ARM and FPGA,” Procedia Engineering, vol. 16, pp. 818-824, 2011, doi: 10.1016/j.proeng.2011.08.1160.

[9] G. Yang, Z. Ye, Y. Pan, and Z. Ma, “The Implementation of S-curve Acceleration and Deceleration Using FPGA,” TELKOMNIKA Indonesian Journal of Electrical Engineering, vol. 11, no. 1, pp. 279-286, Jan. 2013, doi: 10.11591/telkomnika.v11i1.1897.

ΠΑΡΑΡΤΗΜΑ Α : Κώδικας

Κώδικας Α.1 (AutoLISP) - Στοίχιση γεωμετρικών οντοτήτων.

```
;;; Παράρτημα Α: Στοίχιση γεωμετρικών οντοτήτων
;;; δημιουργίας συνεχών διαδρομών πριν από την παραγωγή G-code.

(setq *gc-join-tol* 0.001)

(defun gc:dist2 (a b / dx dy)
  (setq dx (- (car b) (car a))
        dy (- (cadr b) (cadr a)))
  (sqrt (+ (* dx dx) (* dy dy)))
)

(defun gc:pt= (a b)
  (<= (gc:dist2 a b) *gc-join-tol*)
)

(defun gc:pt< (a b)
  (or (< (car a) (car b))
      (and (equal (car a) (car b) 1e-12)
           (< (cadr a) (cadr b))))
)

(defun gc:round (x)
  (if (>= x 0.0)
    (fix (+ x 0.5))
    (- (fix (+ (- x) 0.5)))
  )
)

(defun gc:point-id (pt / ix iy)
  ;; endpoints που διαφέρουν μόνο αριθμητικά μέσα στην ανοχή join-tol
  ;; αντιμετωπίζονται ως το ίδιο σημείο σύνδεσης.
  (setq ix (gc:round (/ (car pt) *gc-join-tol*))
        iy (gc:round (/ (cadr pt) *gc-join-tol*)))
  (strcat (itoa ix) "," (itoa iy))
)

(defun gc:entity-start (entity)
  (cdr (assoc 'start entity))
)

(defun gc:entity-end (entity)
  (cdr (assoc 'end entity))
)

(defun gc:connection-add (table point-id idx / row)
  ;; Ο πίνακας κοινών συντεταγμένων κρατά για κάθε σημείο
  ;; τις οντότητες που αρχίζουν ή τελειώνουν σε αυτό.
  (if (setq row (assoc point-id table))
    (subst (cons point-id (cons idx (cdr row))) row table)
    (cons (list point-id idx) table)
  )
)

(defun gc:build-connection-table (entities / table i entity)
  (setq table '()
        i 0)
  (while (< i (length entities))
    (setq entity (nth i entities))
    (setq table
      (gc:connection-add table
        (gc:point-id (gc:entity-start entity))
        i))
    (setq table
      (gc:connection-add table
        (gc:point-id (gc:entity-end entity))
        i))
```

```
    (setq i (1+ i))
  )
  table
)

(defun gc:unvisited-init (n / i result)
  (setq i 0
        result '())
  (while (< i n)
    (setq result (cons i result))
    (setq i (1+ i))
  )
  (reverse result)
)

(defun gc:find-connected-entity
       (entities unvisited table pt / point-id candidates idx entity found)
  (setq point-id (gc:point-id pt)
        candidates (cdr (assoc point-id table))
        found nil)
  (while (and candidates (not found))
    (setq idx (car candidates))
    (if (member idx unvisited)
      (progn
        (setq entity (nth idx entities))
        (cond
          ((gc:pt= (gc:entity-start entity) pt)
           (setq found (list idx nil)))
          ((gc:pt= (gc:entity-end entity) pt)
           (setq found (list idx T)))
        )
      )
    )
    (setq candidates (cdr candidates))
  )
  found
)

(defun gc:entity-min-point (entity / s e)
  (setq s (gc:entity-start entity)
        e (gc:entity-end entity))
  (if (gc:pt< s e) s e)
)

(defun gc:pick-route-start
       (entities unvisited / best best-idx reverse idx entity s e minpt)
  ;; Η πρώτη οντότητα επιλέγεται από το πλησιέστερο σημείο στο (0,0),
  ;; ώστε το αποτέλεσμα να είναι σταθερό και ανεξάρτητο από τη σειρά επιλογής.
  (setq best nil
        best-idx nil
        reverse nil)
  (foreach idx unvisited
    (setq entity (nth idx entities)
          s (gc:entity-start entity)
          e (gc:entity-end entity)
          minpt (if (gc:pt< s e) s e))
    (if (or (not best) (gc:pt< minpt best))
      (setq best minpt
            best-idx idx
            reverse (not (gc:pt< s e)))
    )
  )
  (list best-idx reverse)
)

(defun gc:route-startpt (entity reverse)
  (if reverse (gc:entity-end entity) (gc:entity-start entity))
)

(defun gc:route-endpt (entity reverse)
  (if reverse (gc:entity-start entity) (gc:entity-end entity))
)
```

```
(defun gc:align-entities
       (entities / table unvisited routes start idx reverse entity
                   route startpt endpt hit changed)
  ;; Επιστρέφει λίστα συνεχών διαδρομών.
  ;; Κάθε στοιχείο μιας διαδρομής έχει μορφή: (δείκτης-οντότητας αναστροφή;).
  (setq table (gc:build-connection-table entities)
        unvisited (gc:unvisited-init (length entities))
        routes '())

  (while unvisited
    (setq start (gc:pick-route-start entities unvisited)
          idx (car start)
          reverse (cadr start)
          entity (nth idx entities)
          route (list (list idx reverse))
          unvisited (vl-remove idx unvisited)
          startpt (gc:route-startpt entity reverse)
          endpt (gc:route-endpt entity reverse))

    (setq changed T)
    (while changed
      (setq changed nil)

      ;; Επέκταση από το τέλος της τρέχουσας διαδρομής.
      (setq hit (gc:find-connected-entity entities unvisited table endpt))
      (if hit
        (progn
          (setq idx (car hit)
                reverse (cadr hit)
                entity (nth idx entities)
                route (append route (list (list idx reverse)))
                unvisited (vl-remove idx unvisited)
                endpt (gc:route-endpt entity reverse)
                changed T)
        )
      )

      ;; Επέκταση από την αρχή της τρέχουσας διαδρομής.
      (setq hit (gc:find-connected-entity entities unvisited table startpt))
      (if hit
        (progn
          (setq idx (car hit)
                reverse (not (cadr hit))
                entity (nth idx entities)
                route (cons (list idx reverse) route)
                unvisited (vl-remove idx unvisited)
                startpt (gc:route-startpt entity reverse)
                changed T)
        )
      )
    )
    (setq routes (append routes (list route)))
  )
  routes
)

(defun gc:emit-aligned-routes
       (f entities routes / route first idx reverse entity startpt pair)
  ;; Η εκπομπή μετακινείται με G0 στην αρχή κάθε διαδρομής
  ;; και στη συνέχεια γράφει G1/G2/G3 ανάλογα με τον τύπο οντότητας.
  (foreach route routes
    (setq first (car route)
          idx (car first)
          reverse (cadr first)
          entity (nth idx entities)
          startpt (gc:route-startpt entity reverse))
    (gc:emit-path-start f startpt)
    (foreach pair route
      (setq idx (car pair)
            reverse (cadr pair)
            entity (nth idx entities))
```

```
        (gc:emit-entity f entity reverse)
      )
      (gc:emit-path-end f)
    )
)
```

Κώδικας Α.2 (Python) - Αποστολή δεδομένων κίνησης

Το Python προγράμματα που χρησιμοποιείτε για την επικοινωνία του υπολογιστή με την πλακέτα DE1-SoC μέσω του CP2102 USB-UART bridge. Τα πρόγραμμα στέλνει δυαδικά πακέτα προς το FPGA και, περιμένει για το byte επιβεβαίωσης ACK.

```
"""
send_steps.py
=============
Reads a steps CSV file generated by GCODESTEPS.LSP and sends
motion packets to the DE1-SoC FPGA through a CP2102 USB-UART bridge.

Usage:
    python send_steps.py "path\\to\\Drawing1_xy_steps.txt"

Motion packet format, 11 bytes:
    Byte  0    : motion   (uint8)  0=G0, 1=G1, 2=G2, 3=G3
    Bytes 1-4  : steps_x  (int32, big-endian, signed)
    Bytes 5-8  : steps_y  (int32, big-endian, signed)
    Bytes 9-10 : feedrate (uint16, big-endian, mm/min)

Initialization packet format, 9 bytes:
    Byte  0   : 0xFF
    Bytes 1-4 : numerator_x (uint32, big-endian)
    Bytes 5-8 : numerator_y (uint32, big-endian)

Handshake:
    PC   -> FPGA : one 11-byte motion packet
    FPGA -> PC   : one ACK byte, 0xAA, when the motion is complete

Requirements:
    pip install pyserial
"""

import os
import serial
import serial.tools.list_ports
import struct
import sys
import time



# User settings
COM_PORT = "COM3"
BAUD_RATE = 115200
ACK_BYTE = 0xAA
ACK_TIMEOUT_SEC = 10.0
INTER_PKT_DELAY = 0.0
NUMERATOR_X = 18_750_000
NUMERATOR_Y = 18_750_000

PACKET_SIZE = 11
MAX_STEPS = 2_147_483_647
MAX_FEEDRATE = 65535



def list_com_ports():
    ports = serial.tools.list_ports.comports()
    if ports:
        print("\nAvailable COM ports:")
```

```
        for port in ports:
            print(f"  {port.device:8s}  {port.description}")
    else:
        print("\nNo COM ports found. Check CP2102 connection.")


def build_packet(motion: int, steps_x: int, steps_y: int, feed_rate: int) -> bytes:
    """
    Pack one motion command as:
    [motion][steps_x][steps_y][feed_rate].
    """
    return struct.pack(">BiiH", motion, steps_x, steps_y, feed_rate)


def build_init_packet(numerator_x: int, numerator_y: int) -> bytes:
    """
    Pack the initialization/configuration command:
    [0xFF][numerator_x][numerator_y].
    """
    return struct.pack(">BII", 0xFF, numerator_x, numerator_y)


def read_steps_file(filepath: str):
    """
    Parse the CSV/text file produced by the G-code step conversion.
    Expected useful columns:
        0: segment number
        2: motion code
        5: X-axis steps
        6: Y-axis steps
        8: feed rate
    """
    segments = []

    with open(filepath, "r", encoding="utf-8", errors="replace") as file:
        for line_no, line in enumerate(file, start=1):
            line = line.strip()
            if not line or line_no == 1:
                continue

            parts = line.split(",")
            if len(parts) < 9:
                print(f"  [WARN] Line {line_no}: not enough columns, skipping.")
                continue

            try:
                seg = int(parts[0].strip())
                motion = int(parts[2].strip())
                steps_x = int(parts[5].strip())
                steps_y = int(parts[6].strip())
                feed_rate = int(round(float(parts[8].strip())))
            except ValueError as err:
                print(f"  [WARN] Line {line_no}: parse error ({err}), skipping.")
                continue

            feed_rate = max(0, min(feed_rate, MAX_FEEDRATE))

            if abs(steps_x) > MAX_STEPS or abs(steps_y) > MAX_STEPS:
                print(f"  [WARN] Segment {seg}: step count exceeds int32, clamping.")
                steps_x = max(-MAX_STEPS, min(steps_x, MAX_STEPS))
                steps_y = max(-MAX_STEPS, min(steps_y, MAX_STEPS))

            segments.append({
                "seg": seg,
                "motion": motion,
                "steps_x": steps_x,
                "steps_y": steps_y,
                "feed_rate": feed_rate,
            })

    return segments
```

```
def wait_for_ack(ser: serial.Serial, seg: int, timeout: float) -> bool:
    deadline = time.time() + timeout

    while time.time() < deadline:
        if ser.in_waiting > 0:
            byte = ser.read(1)
            if byte and byte[0] == ACK_BYTE:
                return True

            if byte:
                print(f"  [WARN] Segment {seg}: unexpected byte 0x{byte[0]:02X}")

        time.sleep(0.001)

    return False


def send_steps(filepath: str):
    if not os.path.isfile(filepath):
        print(f"\n[ERROR] File not found:\n  {filepath}")
        sys.exit(1)

    print("\n" + "=" * 60)
    print("  send_steps.py - FPGA Stepper Motor Controller")
    print("=" * 60)
    print(f"\nReading: {filepath}")

    segments = read_steps_file(filepath)
    if not segments:
        print("[ERROR] No valid segments found in file.")
        sys.exit(1)

    total = len(segments)
    print(f"  Found {total} segments to send.")

    print(f"\nOpening {COM_PORT} @ {BAUD_RATE} baud ...")
    try:
        ser = serial.Serial(
            port=COM_PORT,
            baudrate=BAUD_RATE,
            bytesize=serial.EIGHTBITS,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            timeout=0.1,
        )
    except serial.SerialException as err:
        print(f"\n[ERROR] Cannot open {COM_PORT}:\n  {err}")
        list_com_ports()
        sys.exit(1)

    print("  Port open. Flushing buffers...")
    time.sleep(0.3)
    ser.reset_input_buffer()
    ser.reset_output_buffer()

    print(f"  Sending init packet: NX={NUMERATOR_X}, NY={NUMERATOR_Y}")
    ser.write(build_init_packet(NUMERATOR_X, NUMERATOR_Y))
    time.sleep(0.05)
    ser.reset_input_buffer()

    start_time = time.time()
    sent = 0
    errors = 0

    try:
        for index, item in enumerate(segments):
            seg = item["seg"]
            motion = item["motion"]
            steps_x = item["steps_x"]
            steps_y = item["steps_y"]
```

```
            feed_rate = item["feed_rate"]

            packet = build_packet(motion, steps_x, steps_y, feed_rate)
            ser.write(packet)

            if wait_for_ack(ser, seg, ACK_TIMEOUT_SEC):
                sent += 1
            else:
                errors += 1
                print(f"\n  [ERROR] Segment {seg}: ACK timeout.")
                print(f"          motion={motion}, X={steps_x}, Y={steps_y}, F={feed_rate}")
                answer = input("  Continue anyway? [y/N]: ").strip().lower()
                if answer != "y":
                    print("\n  Transmission aborted by user.")
                    break

            if index % 10 == 0 or index == total - 1:
                percent = (index + 1) / total * 100.0
                elapsed = time.time() - start_time
                rate = (index + 1) / elapsed if elapsed > 0 else 0.0
                eta = (total - index - 1) / rate if rate > 0 else 0.0
                motion_name = {
                    0: "G0",
                    1: "G1",
                    2: "G2",
                    3: "G3",
                }.get(motion, "?")

                print(
                    f"  Segment {seg:5d}/{total} [{percent:5.1f}%] "
                    f"{motion_name:2s} X={steps_x:+7d} Y={steps_y:+7d} "
                    f"F={feed_rate:4d} ETA={eta:5.1f}s",
                    end="\r",
                )

            if INTER_PKT_DELAY > 0:
                time.sleep(INTER_PKT_DELAY)

    except KeyboardInterrupt:
        print("\n\n[!] Interrupted by user.")

    finally:
        ser.close()

    elapsed = time.time() - start_time
    print("\n\n" + "=" * 60)
    print("  Transmission complete")
    print("=" * 60)
    print(f"  Total segments : {total}")
    print(f"  Sent OK        : {sent}")
    print(f"  Errors         : {errors}")
    print(f"  Time elapsed   : {elapsed:.2f} s")
    if sent > 0 and elapsed > 0:
        print(f"  Average rate   : {sent / elapsed:.1f} segments/s")

    sys.exit(1 if errors > 0 else 0)


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("\nUsage: python send_steps.py <path_to_steps_file.txt>")
        print('Example: python send_steps.py "C:\\FPGA\\Drawing1_xy_steps.txt"')
        list_com_ports()
        sys.exit(1)

    send_steps(sys.argv[1])
```

Κώδικας Α.3 (Verilog) - Αποστολή δεδομένων κίνησης

Στο παρόν παράρτημα παρατίθενται τα βασικά Verilog αρχεία της κύριας υλοποίησης του ελεγκτή CNC στο FPGA. Περιλαμβάνονται τα synthesizable αρχεία που δηλώνονται στο project VerilogGCode/CNC_Controller.qsf και χρησιμοποιούνται για τη λήψη πακέτων UART, την παραγωγή παλμών STEP/DIR, τον έλεγχο του laser και την ενοποίηση των επιμέρους modules στο top-level.

Πίνακας Π.0.1: *Αρχεία Verilog που περιλαμβάνονται στο παράρτημα.*

Α/Α	Αρχείο	Γραμμές	Ρόλος
1	top.v	203	Module ενοποίησης της υλοποίησης και αντιστοίχιση σημάτων στο top-level.
2	uart_rx.v	110	Δέκτης UART 8N1 για λήψη bytes από τον CP2102.
3	uart_tx.v	130	Πομπός UART για αποστολή ACK προς τον υπολογιστή.
4	packet_decoder.v	143	Αποκωδικοποιητής init και motion πακέτων.
5	step_gen.v	263	Γεννήτρια παλμών STEP/DIR για τους άξονες X και Y.
6	laser_ctrl.v	53	Έλεγχος ενεργοποίησης laser με βάση τον τύπο κίνησης και το busy.

Π.1 top.v - ενοποίησης της υλοποίησης και αντιστοίχιση σημάτων στο top-level.

```
0001  // ============================================================
0002  // top.v
0003  // Top-level module for DE1-SoC CNC Laser Engraver Controller
0004  //
0005  // Connects all modules:
0006  //   uart_rx → packet_decoder → step_gen → motor drivers
0007  //                                      → laser_ctrl → laser
0008  //   step_gen busy falling edge → uart_tx → ACK to Python
0009  //
0010  // Status display:
0011  //   LEDR[0] = busy       (ON while motors moving)
0012  //   LEDR[1] = laser      (ON while laser firing)
0013  //   LEDR[2] = dir_x      (ON = positive/forward)
0014  //   LEDR[3] = dir_y      (ON = positive/forward)
0015  //   LEDR[7] = heartbeat  (toggles each completed move)
0016  //
0017  //   HEX5    = Motion type    (0=G0 Rapid, 1=G1 Linear, 3=G3 Arc)
0018  //   HEX4    = Busy indicator (b=busy, 0=idle)
0019  //   HEX3-2  = Packet count   (00-FF hex)
0020  //   HEX1-0  = FeedRate low byte (e.g. 20=800mm/min, B8=3000mm/min)
0021  //
0022  // Pin assignments:
0023  //   clk      → PIN_AF14   CLOCK_50
0024  //   rst_n    → PIN_AA14   KEY[0]
0025  //   rx       → PIN_AC18   GPIO_0[0]  JP1 Pin 1  ← CP2102 TX
0026  //   tx       → PIN_Y17    GPIO_0[1]  JP1 Pin 2  ← CP2102 RX
0027  //   dir_x    → PIN_AD17   GPIO_0[2]  JP1 Pin 3  ← A4988 X DIR
0028  //   step_x   → PIN_Y18    GPIO_0[3]  JP1 Pin 4  ← A4988 X STEP
0029  //   dir_y    → PIN_AK16   GPIO_0[4]  JP1 Pin 7  ← A4988 Y DIR
0030  //   step_y   → PIN_AK18   GPIO_0[5]  JP1 Pin 9  ← A4988 Y STEP
0031  //   laser    → PIN_AK19   GPIO_0[6]  JP1 Pin 11 ← Laser enable
0032  //   LEDR[0]  → PIN_V16
0033  //   LEDR[1]  → PIN_W16
0034  //   LEDR[2]  → PIN_V17
0035  //   LEDR[3]  → PIN_V18
0036  //   LEDR[7]  → PIN_W20
0037  //   HEX0[6:0]→ PIN_AH28..PIN_AE26
0038  //   HEX1[6:0]→ PIN_AD27..PIN_AJ29
0039  //   HEX2[6:0]→ PIN_AC30..PIN_AB23
0040  //   HEX3[6:0]→ PIN_AB22..PIN_AD26
0041  //   HEX4[6:0]→ PIN_W25..PIN_AA24
0042  //   HEX5[6:0]→ PIN_AA25..PIN_V25
0043  // ============================================================
0044
0045  module top (
0046      input  wire        clk,       // 50 MHz CLOCK_50
0047      input  wire        rst_n,     // KEY[0] active LOW
0048      input  wire        rx,        // UART RX from CP2102
0049
0050      output wire        tx,        // UART TX to CP2102 (ACK)
0051      output wire        dir_x,     // Motor X direction → A4988
0052      output wire        step_x,    // Motor X step pulse → A4988
0053      output wire        dir_y,     // Motor Y direction → A4988
0054      output wire        step_y,    // Motor Y step pulse → A4988
0055      output wire        laser,     // Laser enable (active HIGH)
0056
0057      output reg  [7:0]  LEDR,      // Status LEDs
0058      output reg  [6:0]  HEX0,      // 7-segment displays
0059      output reg  [6:0]  HEX1,
0060      output reg  [6:0]  HEX2,
0061      output reg  [6:0]  HEX3,
0062      output reg  [6:0]  HEX4,
0063      output reg  [6:0]  HEX5
0064  );
0065
0066  // ── Reset ──────────────────────────────────────────────────
0067  wire rst;
0068  assign rst = ~rst_n;
0069
0070  // ── Internal signals ───────────────────────────────────────
0071  wire [7:0]  rx_data;
0072  wire        rx_valid;
0073  wire [1:0]  motion;
0074  wire [31:0] steps_x, steps_y;
0075  wire [15:0] feed_rate;
0076  wire        pkt_valid;
0077  wire        busy;
```

```
reg         tx_start;
wire        tx_busy;
reg         busy_prev;
reg [7:0]   pkt_count;

// ── Module instances ──────────────────────────────────────
uart_rx uart_rx_inst (
    .clk      (clk),
    .rst      (rst),
    .rx       (rx),
    .rx_data  (rx_data),
    .rx_valid (rx_valid)
);

packet_decoder pkt_dec_inst (
    .clk       (clk),
    .rst       (rst),
    .rx_data   (rx_data),
    .rx_valid  (rx_valid),
    .motion    (motion),
    .steps_x   (steps_x),
    .steps_y   (steps_y),
    .feed_rate (feed_rate),
    .pkt_valid (pkt_valid)
);

step_gen step_gen_inst (
    .clk       (clk),
    .rst       (rst),
    .steps_x   (steps_x),
    .steps_y   (steps_y),
    .feed_rate (feed_rate),
    .pkt_valid (pkt_valid),
    .dir_x     (dir_x),
    .step_x    (step_x),
    .dir_y     (dir_y),
    .step_y    (step_y),
    .busy      (busy)
);

laser_ctrl laser_ctrl_inst (
    .clk       (clk),
    .rst       (rst),
    .motion    (motion),
    .pkt_valid (pkt_valid),
    .busy      (busy),
    .laser     (laser)
);

uart_tx uart_tx_inst (
    .clk      (clk),
    .rst      (rst),
    .tx_data  (8'hAA),
    .tx_start (tx_start),
    .tx_busy  (tx_busy),
    .tx       (tx)
);

// ── ACK logic – send 0xAA when busy falls LOW ─────────────
always @(posedge clk) begin
    if (rst) begin
        busy_prev <= 1'b0;
        tx_start  <= 1'b0;
        pkt_count <= 8'd0;
        LEDR      <= 8'h00;
    end
    else begin
        tx_start  <= 1'b0;    // default: no transmit
        busy_prev <= busy;

        // Detect falling edge of busy (move complete)
        if (busy_prev && !busy) begin
            if (!tx_busy)
                tx_start <= 1'b1;    // send ACK
            pkt_count <= pkt_count + 8'd1;
            LEDR[7]   <= ~LEDR[7];   // heartbeat
        end

```

```
0157          // Live status LEDs
0158          LEDR[0] <= busy;
0159          LEDR[1] <= laser;
0160          LEDR[2] <= dir_x;
0161          LEDR[3] <= dir_y;
0162      end
0163  end
0164
0165  // ── 7-segment display ──────────────────────────────────────
0166  // Shows: [Motion][Busy][PktCountHi][PktCountLo][FRateHi][FRateLo]
0167
0168  function [6:0] h2s;
0169      input [3:0] h;
0170      case (h)
0171          4'h0: h2s=7'b1000000; 4'h1: h2s=7'b1111001;
0172          4'h2: h2s=7'b0100100; 4'h3: h2s=7'b0110000;
0173          4'h4: h2s=7'b0011001; 4'h5: h2s=7'b0010010;
0174          4'h6: h2s=7'b0000010; 4'h7: h2s=7'b1111000;
0175          4'h8: h2s=7'b0000000; 4'h9: h2s=7'b0010000;
0176          4'hA: h2s=7'b0001000; 4'hB: h2s=7'b0000011;
0177          4'hC: h2s=7'b1000110; 4'hD: h2s=7'b0100001;
0178          4'hE: h2s=7'b0000110; 4'hF: h2s=7'b0001110;
0179          default: h2s=7'b1111111;
0180      endcase
0181  endfunction
0182
0183  // SEG_b = lowercase 'b' = busy indicator
0184  localparam SEG_b    = 7'b0000011;  // b
0185  localparam SEG_IDLE = 7'b1000000;  // 0 = idle
0186
0187  always @(posedge clk) begin
0188      // HEX5 = motion type (0, 1, or 3)
0189      HEX5 <= h2s({2'b00, motion});
0190
0191      // HEX4 = busy indicator (b=busy, 0=idle)
0192      HEX4 <= busy ? SEG_b : SEG_IDLE;
0193
0194      // HEX3-2 = packet count in hex
0195      HEX3 <= h2s(pkt_count[7:4]);
0196      HEX2 <= h2s(pkt_count[3:0]);
0197
0198      // HEX1-0 = feed_rate low byte in hex
0199      HEX1 <= h2s(feed_rate[7:4]);
0200      HEX0 <= h2s(feed_rate[3:0]);
0201  end
0202
    203  endmodule
```

Π.2 uart_rx.v - Δέκτης UART 8N1 για λήψη bytes από τον CP2102.

```
0001  // ============================================================
0002  // uart_rx.v  (v1 — original, proven to work for full packets)
0003  //
0004  // Clock : 50 MHz   Baud : 115200   8N1
0005  // FULL_BIT = 434 cycles
0006  // HALF_BIT = 217 cycles
0007  // ============================================================
0008
0009  module uart_rx (
0010      input  wire       clk,
0011      input  wire       rst,
0012      input  wire       rx,
0013      output reg  [7:0] rx_data,
0014      output reg        rx_valid
0015  );
0016
0017  localparam FULL_BIT = 10'd434;
0018  localparam HALF_BIT = 10'd217;
0019
0020  localparam S_IDLE    = 2'd0;
0021  localparam S_START   = 2'd1;
0022  localparam S_RECEIVE = 2'd2;
0023  localparam S_STOP    = 2'd3;
0024
0025  reg [1:0]  state;
0026  reg [9:0]  count;
0027  reg [2:0]  bit_idx;
0028  reg [7:0]  rx_shift;
0029  reg        rx_s1, rx_s2;
0030
0031  always @(posedge clk) begin
0032      if (rst) begin
0033          state    <= S_IDLE;
0034          count    <= 10'd0;
0035          bit_idx  <= 3'd0;
0036          rx_shift <= 8'h00;
0037          rx_data  <= 8'h00;
0038          rx_valid <= 1'b0;
0039          rx_s1    <= 1'b1;
0040          rx_s2    <= 1'b1;
0041      end
0042      else begin
0043          rx_s1    <= rx;
0044          rx_s2    <= rx_s1;
0045          rx_valid <= 1'b0;
0046
0047          case (state)
0048
0049              S_IDLE: begin
0050                  count   <= 10'd0;
0051                  bit_idx <= 3'd0;
0052                  if (rx_s2 == 1'b0) begin
0053                      state <= S_START;
0054                      count <= 10'd1;
0055                  end
0056              end
0057
0058              S_START: begin
0059                  if (count == HALF_BIT) begin
0060                      if (rx_s2 == 1'b0) begin
0061                          count <= 10'd1;
0062                          state <= S_RECEIVE;
0063                      end
0064                      else begin
0065                          state <= S_IDLE;
0066                          count <= 10'd0;
0067                      end
0068                  end
0069                  else
0070                      count <= count + 10'd1;
0071              end
0072
0073              S_RECEIVE: begin
0074                  if (count == FULL_BIT) begin
0075                      count             <= 10'd1;
0076                      rx_shift[bit_idx] <= rx_s2;
0077                      if (bit_idx == 3'd7) begin
```

```
0078                        bit_idx <= 3'd0;
0079                        state   <= S_STOP;
0080                    end
0081                    else
0082                        bit_idx <= bit_idx + 3'd1;
0083                end
0084                else
0085                    count <= count + 10'd1;
0086            end
0087
0088            S_STOP: begin
0089                if (count == FULL_BIT) begin
0090                    count <= 10'd0;
0091                    state <= S_IDLE;
0092                    if (rx_s2 == 1'b1) begin
0093                        rx_data  <= rx_shift;
0094                        rx_valid <= 1'b1;
0095                    end
0096                end
0097                else
0098                    count <= count + 10'd1;
0099            end
0100
0101            default: begin
0102                state <= S_IDLE;
0103                count <= 10'd0;
0104            end
0105
0106        endcase
0107    end
0108 end
0109
    110 endmodule
```

Π.3 uart_tx.v - Πομπός UART για αποστολή ACK προς τον υπολογιστή.

```
0001  // ============================================================
0002  // uart_tx.v
0003  // UART Transmitter for DE1-SoC
0004  //
0005  // Settings:
0006  //   Clock     : 50 MHz
0007  //   Baud rate : 115200
0008  //   Data bits : 8
0009  //   Parity    : None
0010  //   Stop bits : 1
0011  //
0012  // Interface:
0013  //   clk       : 50 MHz system clock
0014  //   rst       : synchronous reset, active HIGH
0015  //   tx_data   : 8-bit byte to send
0016  //   tx_start  : pulse HIGH for 1 clock to begin transmission
0017  //   tx_busy   : HIGH while transmitting (do not send new byte)
0018  //   tx        : serial output to CP2102 RX pin
0019  // ============================================================
0020
0021  module uart_tx (
0022      input  wire       clk,
0023      input  wire       rst,
0024      input  wire [7:0] tx_data,
0025      input  wire       tx_start,
0026      output reg        tx_busy,
0027      output reg        tx
0028  );
0029
0030  // ── Timing parameters ──────────────────────────────────────
0031  localparam CLK_FREQ  = 50_000_000;
0032  localparam BAUD_RATE = 115_200;
0033  localparam FULL_BIT  = CLK_FREQ / BAUD_RATE;    // 434 cycles
0034
0035  // ── State encoding ─────────────────────────────────────────
0036  localparam IDLE  = 2'd0;
0037  localparam START = 2'd1;
0038  localparam DATA  = 2'd2;
0039  localparam STOP  = 2'd3;
0040
0041  // ── Internal registers ─────────────────────────────────────
0042  reg [1:0]  state;
0043  reg [8:0]  clk_count;
0044  reg [2:0]  bit_index;
0045  reg [7:0]  tx_shift;     // holds byte being transmitted
0046
0047  // ── State machine ──────────────────────────────────────────
0048  always @(posedge clk) begin
0049      if (rst) begin
0050          state     <= IDLE;
0051          clk_count <= 9'd0;
0052          bit_index <= 3'd0;
0053          tx_shift  <= 8'h00;
0054          tx_busy   <= 1'b0;
0055          tx        <= 1'b1;   // idle HIGH
0056      end
0057      else begin
0058          case (state)
0059
0060              // ─────────────────────────────────────────────
0061              // IDLE: wait for tx_start pulse
0062              // ─────────────────────────────────────────────
0063              IDLE: begin
0064                  tx      <= 1'b1;    // line idle HIGH
0065                  tx_busy <= 1'b0;
0066                  if (tx_start) begin
0067                      tx_shift  <= tx_data;
0068                      tx_busy   <= 1'b1;
0069                      clk_count <= 9'd0;
0070                      state     <= START;
0071                  end
0072              end
0073
0074              // ─────────────────────────────────────────────
0075              // START: send START bit (LOW) for one bit period
0076              // ─────────────────────────────────────────────
0077              START: begin
```

```
                tx <= 1'b0;     // START bit
                if (clk_count == FULL_BIT - 1) begin
                    clk_count <= 9'd0;
                    bit_index <= 3'd0;
                    state     <= DATA;
                end
                else begin
                    clk_count <= clk_count + 9'd1;
                end
            end

            // ---------------------------------------------------
            // DATA: send 8 bits LSB first
            // ---------------------------------------------------
            DATA: begin
                tx <= tx_shift[bit_index];   // LSB first
                if (clk_count == FULL_BIT - 1) begin
                    clk_count <= 9'd0;
                    if (bit_index == 3'd7) begin
                        bit_index <= 3'd0;
                        state     <= STOP;
                    end
                    else begin
                        bit_index <= bit_index + 3'd1;
                    end
                end
                else begin
                    clk_count <= clk_count + 9'd1;
                end
            end

            // ---------------------------------------------------
            // STOP: send STOP bit (HIGH) for one bit period
            // ---------------------------------------------------
            STOP: begin
                tx <= 1'b1;     // STOP bit
                if (clk_count == FULL_BIT - 1) begin
                    clk_count <= 9'd0;
                    tx_busy   <= 1'b0;
                    state     <= IDLE;
                end
                else begin
                    clk_count <= clk_count + 9'd1;
                end
            end

            default: state <= IDLE;

        endcase
    end
end

endmodule
```

Π.4 packet_decoder.v - Αποκωδικοποιητής init και motion πακέτων.

```
0001  // ============================================================
0002  // packet_decoder.v
0003  // Assembles incoming UART bytes into motor control fields.
0004  //
0005  // Two packet types (distinguished by first byte):
0006  //
0007  // [A] Init packet (9 bytes) — sent once before motion:
0008  //   Byte  0      : 0xFF        config marker
0009  //   Bytes 1-4    : NUMERATOR_X (uint32) MSB first
0010  //   Bytes 5-8    : NUMERATOR_Y (uint32) MSB first
0011  //   → asserts cfg_valid for one clock on completion
0012  //
0013  // [B] Motion packet (11 bytes) — one per segment:
0014  //   Byte  0      : Motion      (uint8)  0=G0, 1=G1, 3=G3
0015  //   Bytes 1-4    : StepsX      (int32)  signed, MSB first
0016  //   Bytes 5-8    : StepsY      (int32)  signed, MSB first
0017  //   Bytes 9-10   : FeedRate    (uint16) unsigned, MSB first
0018  //   → asserts pkt_valid for one clock on completion
0019  // ============================================================
0020
0021  module packet_decoder (
0022      input  wire        clk,
0023      input  wire        rst,
0024      input  wire [7:0]  rx_data,
0025      input  wire        rx_valid,
0026
0027      // Motion packet outputs
0028      output reg  [1:0]  motion,
0029      output reg  [31:0] steps_x,
0030      output reg  [31:0] steps_y,
0031      output reg  [15:0] feed_rate,
0032      output reg         pkt_valid,
0033
0034      // Init packet outputs
0035      output reg  [31:0] numerator_x,
0036      output reg  [31:0] numerator_y,
0037      output reg         cfg_valid
0038  );
0039
0040  // 100ms timeout at 50MHz = 5,000,000 cycles (23 bits)
0041  localparam TIMEOUT_CYCLES = 23'd5_000_000;
0042  localparam CFG_MARKER     = 8'hFF;   // init packet marker
0043
0044  reg [3:0]  byte_count;
0045  reg [22:0] timeout_cnt;
0046  reg        timeout_active;
0047  reg        is_cfg_pkt;       // 1 = current packet is init, 0 = motion
0048
0049  // Individual byte registers (no array — Quartus 13.1 safe)
0050  reg [7:0] b0,b1,b2,b3,b4,b5,b6,b7,b8,b9;
0051
0052  always @(posedge clk) begin
0053      if (rst) begin
0054          byte_count     <= 4'd0;
0055          pkt_valid      <= 1'b0;
0056          cfg_valid      <= 1'b0;
0057          timeout_cnt    <= 23'd0;
0058          timeout_active <= 1'b0;
0059          is_cfg_pkt     <= 1'b0;
0060          motion         <= 2'd0;
0061          steps_x        <= 32'd0;
0062          steps_y        <= 32'd0;
0063          feed_rate      <= 16'd0;
0064          numerator_x    <= 32'd18_750_000;  // safe default (1/32, GT2, 20T)
0065          numerator_y    <= 32'd18_750_000;
0066          b0<=8'h00; b1<=8'h00; b2<=8'h00; b3<=8'h00;
0067          b4<=8'h00; b5<=8'h00; b6<=8'h00; b7<=8'h00;
0068          b8<=8'h00; b9<=8'h00;
0069      end
0070      else begin
0071          pkt_valid <= 1'b0;
0072          cfg_valid <= 1'b0;
0073
0074          // ── Timeout watchdog ──────────────────────────────────
0075          if (timeout_active) begin
0076              if (rx_valid)
0077                  timeout_cnt <= 23'd0;
```

```
0078              else begin
0079                  timeout_cnt <= timeout_cnt + 23'd1;
0080                  if (timeout_cnt >= TIMEOUT_CYCLES - 23'd1) begin
0081                      byte_count     <= 4'd0;
0082                      timeout_cnt    <= 23'd0;
0083                      timeout_active <= 1'b0;
0084                      is_cfg_pkt     <= 1'b0;
0085                  end
0086              end
0087          end
0088
0089          // ── Byte collection ──────────────────────────────────────────
0090          if (rx_valid) begin
0091              timeout_cnt <= 23'd0;
0092
0093              // Store byte into register bank (shared for both packet types)
0094              case (byte_count)
0095                  4'd0:  b0 <= rx_data;
0096                  4'd1:  b1 <= rx_data;
0097                  4'd2:  b2 <= rx_data;
0098                  4'd3:  b3 <= rx_data;
0099                  4'd4:  b4 <= rx_data;
0100                  4'd5:  b5 <= rx_data;
0101                  4'd6:  b6 <= rx_data;
0102                  4'd7:  b7 <= rx_data;
0103                  4'd8:  b8 <= rx_data;
0104                  4'd9:  b9 <= rx_data;
0105                  default: ;
0106              endcase
0107
0108              // On first byte — start timeout and detect packet type
0109              if (byte_count == 4'd0) begin
0110                  timeout_active <= 1'b1;
0111                  is_cfg_pkt     <= (rx_data == CFG_MARKER) ? 1'b1 : 1'b0;
0112              end
0113
0114              // ── Init packet complete at byte 8 (9 bytes total) ──────────
0115              if (is_cfg_pkt && byte_count == 4'd8) begin
0116                  // b1-b4 = NUMERATOR_X, b5-b8 = NUMERATOR_Y (last byte = rx_data)
0117                  // byte_count==8 means we stored b0..b7, rx_data is b8
0118                  numerator_x    <= { b1, b2, b3, b4 };
0119                  numerator_y    <= { b5, b6, b7, rx_data };
0120                  cfg_valid      <= 1'b1;
0121                  byte_count     <= 4'd0;
0122                  timeout_active <= 1'b0;
0123                  timeout_cnt    <= 23'd0;
0124                  is_cfg_pkt     <= 1'b0;
0125              end
0126              // ── Motion packet complete at byte 10 (11 bytes total) ──────
0127              else if (!is_cfg_pkt && byte_count == 4'd10) begin
0128                  motion    <= b0[1:0];
0129                  steps_x   <= { b1, b2, b3, b4 };
0130                  steps_y   <= { b5, b6, b7, b8 };
0131                  feed_rate <= { b9, rx_data };
0132                  pkt_valid      <= 1'b1;
0133                  byte_count     <= 4'd0;
0134                  timeout_active <= 1'b0;
0135                  timeout_cnt    <= 23'd0;
0136              end
0137              else
0138                  byte_count <= byte_count + 4'd1;
0139          end
0140      end
0141  end
0142
    143  endmodule
```

Π.5 step_gen.v - Γεννήτρια παλμών STEP/DIR για τους άξονες X και Y.

```
0001  // ============================================================
0002  // step_gen.v
0003  // Stepper motor STEP/DIR pulse generator for DE1-SoC
0004  //
0005  // Generates STEP and DIR signals for two DRV8825 motor drivers.
0006  // Supports linear interpolation so X and Y finish together.
0007  //
0008  // Settings:
0009  //   Clock        : 50 MHz
0010  //   STEP pulse   : 100 cycles HIGH (2µs) — DRV8825 min 1.9µs
0011  //   DIR settle   : 200 cycles (4µs before first STEP)
0012  //
0013  // NUMERATOR_X / NUMERATOR_Y are no longer hardcoded.
0014  // They are received dynamically via the init packet from the PC:
0015  //   NUMERATOR = CLK_HZ * 60 / steps_per_mm
0016  //
0017  // Until the first init packet arrives the safe reset defaults
0018  // (18,750,000 = 50MHz * 60 / 160) keep behaviour identical
0019  // to the previous hardcoded version.
0020  //
0021  // cycles_per_step = NUMERATOR / feed_rate
0022  // Note: feed_rate is already the dominant-axis velocity component computed
0023  // by GCODESTEPS.LSP from the requested resultant path feed F.
0024  //
0025  // States: IDLE → SETUP → DIR_WAIT → RUN → DONE → IDLE
0026  // ============================================================
0027
0028  module step_gen (
0029      input  wire        clk,
0030      input  wire        rst,
0031
0032      // From packet_decoder — motion packet
0033      input  wire [31:0] steps_x,
0034      input  wire [31:0] steps_y,
0035      input  wire [15:0] feed_rate,
0036      input  wire        pkt_valid,
0037
0038      // From packet_decoder — init packet
0039      input  wire [31:0] numerator_x,
0040      input  wire [31:0] numerator_y,
0041      input  wire        cfg_valid,
0042
0043      // To DRV8825 drivers
0044      output reg         dir_x,
0045      output reg         step_x,
0046      output reg         dir_y,
0047      output reg         step_y,
0048      output reg         busy,
0049
0050      // For display / debug
0051      output wire [31:0] vel_x,    // current period_x (cycles between X steps)
0052      output wire [31:0] vel_y     // current period_y (cycles between Y steps)
0053  );
0054
0055  // ── Timing constants ──────────────────────────────────────
0056  localparam PULSE_HIGH = 32'd100;   // 2µs HIGH — DRV8825 requires min 1.9µs
0057  localparam DIR_SETTLE = 9'd200;    // 4µs DIR setup time
0058
0059  // ── Default NUMERATOR (1/32 microstep, GT2, 20-tooth = 160 steps/mm) ──
0060  localparam NUMERATOR_DEFAULT = 32'd18_750_000;
0061
0062  // ── States ────────────────────────────────────────────────
0063  localparam IDLE     = 3'd0;
0064  localparam SETUP    = 3'd1;
0065  localparam DIR_WAIT = 3'd2;
0066  localparam RUN      = 3'd3;
0067  localparam DONE     = 3'd4;
0068
0069  reg [2:0]  state;
0070
0071  // ── Dynamic NUMERATOR registers (latched from init packet) ────
0072  reg [31:0] num_x;   // latched NUMERATOR_X
0073  reg [31:0] num_y;   // latched NUMERATOR_Y
0074
0075  // ── Working registers ─────────────────────────────────────
0076  reg [31:0] abs_x;       // |steps_x|
0077  reg [31:0] abs_y;       // |steps_y|
```

```
0078  reg [31:0] rem_x;       // remaining X steps
0079  reg [31:0] rem_y;       // remaining Y steps
0080  reg [31:0] period_x;    // cycles between X steps
0081  reg [31:0] period_y;    // cycles between Y steps
0082  reg [31:0] cnt_x;       // X step timer
0083  reg [31:0] cnt_y;       // Y step timer
0084  reg [8:0]  dir_cnt;     // DIR settle counter
0085
0086  // Expose period registers for display
0087  assign vel_x = period_x;
0088  assign vel_y = period_y;
0089
0090  // ── Main state machine ──────────────────────────────────
0091  always @(posedge clk) begin
0092      if (rst) begin
0093          state    <= IDLE;
0094          dir_x    <= 1'b0; step_x  <= 1'b0;
0095          dir_y    <= 1'b0; step_y  <= 1'b0;
0096          busy     <= 1'b0;
0097          abs_x    <= 32'd0; abs_y   <= 32'd0;
0098          rem_x    <= 32'd0; rem_y   <= 32'd0;
0099          period_x <= 32'd0; period_y <= 32'd0;
0100          cnt_x    <= 32'd0; cnt_y   <= 32'd0;
0101          dir_cnt  <= 9'd0;
0102          num_x    <= NUMERATOR_DEFAULT;
0103          num_y    <= NUMERATOR_DEFAULT;
0104      end
0105      else begin
0106
0107          // ── Latch NUMERATOR_X/Y whenever init packet arrives ──────────────
0108          // Allowed only when IDLE — never mid-motion
0109          if (cfg_valid && state == IDLE) begin
0110              num_x <= numerator_x;
0111              num_y <= numerator_y;
0112          end
0113
0114          case (state)
0115
0116              // ──────────────────────────────────────────────────
0117              // IDLE — wait for new motion packet
0118              // ──────────────────────────────────────────────────
0119              IDLE: begin
0120                  step_x <= 1'b0;
0121                  step_y <= 1'b0;
0122                  busy   <= 1'b0;
0123
0124                  if (pkt_valid) begin
0125                      busy  <= 1'b1;
0126                      state <= SETUP;
0127                  end
0128              end
0129
0130              // ──────────────────────────────────────────────────
0131              // SETUP — latch direction and absolute step counts
0132              // ──────────────────────────────────────────────────
0133              SETUP: begin
0134                  dir_x <= (steps_x[31] == 1'b0) ? 1'b1 : 1'b0;
0135                  dir_y <= (steps_y[31] == 1'b0) ? 1'b1 : 1'b0;
0136
0137                  abs_x <= steps_x[31] ? (~steps_x + 32'd1) : steps_x;
0138                  abs_y <= steps_y[31] ? (~steps_y + 32'd1) : steps_y;
0139
0140                  dir_cnt <= 9'd0;
0141                  state   <= DIR_WAIT;
0142              end
0143
0144              // ──────────────────────────────────────────────────
0145              // DIR_WAIT — settle DIR and compute step periods
0146              // Uses num_x / num_y (dynamic, latched from init packet)
0147              // ──────────────────────────────────────────────────
0148              DIR_WAIT: begin
0149
0150                  if (dir_cnt == 9'd0) begin
0151                      rem_x <= abs_x;
0152                      rem_y <= abs_y;
0153                      cnt_x <= 32'd0;
0154                      cnt_y <= 32'd0;
0155
0156                      if (feed_rate == 16'd0) begin
```

```
                        period_x <= 32'd62_500;
                        period_y <= 32'd62_500;
                    end
                    else if (abs_x == 32'd0 && abs_y == 32'd0) begin
                        period_x <= 32'd0;
                        period_y <= 32'd0;
                    end
                    else if (abs_x >= abs_y) begin
                        // X is dominant axis. feed_rate is the X component
                        // required to make resultant path speed equal G-code F.
                        period_x <= num_x / {16'd0, feed_rate};
                        if (abs_y == 32'd0)
                            period_y <= 32'd0;
                        else
                            period_y <= (num_x / {16'd0, feed_rate})
                                        * abs_x / abs_y;
                    end
                    else begin
                        // Y is dominant axis. feed_rate is the Y component
                        // required to make resultant path speed equal G-code F.
                        period_y <= num_y / {16'd0, feed_rate};
                        if (abs_x == 32'd0)
                            period_x <= 32'd0;
                        else
                            period_x <= (num_y / {16'd0, feed_rate})
                                        * abs_y / abs_x;
                    end
                end

                if (dir_cnt >= DIR_SETTLE - 9'd1) begin
                    dir_cnt <= 9'd0;
                    if (abs_x == 32'd0 && abs_y == 32'd0)
                        state <= DONE;
                    else
                        state <= RUN;
                end
                else
                    dir_cnt <= dir_cnt + 9'd1;
            end

            // ──────────────────────────────────────────────
            // RUN — generate STEP pulses for both axes
            // ──────────────────────────────────────────────
            RUN: begin

                // ── X axis ──────────────────────────────
                if (rem_x > 32'd0 && period_x > 32'd0) begin
                    if (cnt_x == 32'd0)
                        step_x <= 1'b1;
                    else if (cnt_x == PULSE_HIGH)
                        step_x <= 1'b0;

                    if (cnt_x >= period_x - 32'd1) begin
                        cnt_x <= 32'd0;
                        rem_x <= rem_x - 32'd1;
                    end
                    else
                        cnt_x <= cnt_x + 32'd1;
                end
                else
                    step_x <= 1'b0;

                // ── Y axis ──────────────────────────────
                if (rem_y > 32'd0 && period_y > 32'd0) begin
                    if (cnt_y == 32'd0)
                        step_y <= 1'b1;
                    else if (cnt_y == PULSE_HIGH)
                        step_y <= 1'b0;

                    if (cnt_y >= period_y - 32'd1) begin
                        cnt_y <= 32'd0;
                        rem_y <= rem_y - 32'd1;
                    end
                    else
                        cnt_y <= cnt_y + 32'd1;
                end
                else
                    step_y <= 1'b0;

```

```
                // ── Both axes done? ──────────────────────────
                if ((rem_x == 32'd1 && cnt_x == period_x - 32'd1 ||
                     rem_x == 32'd0 || period_x == 32'd0) &&
                    (rem_y == 32'd1 && cnt_y == period_y - 32'd1 ||
                     rem_y == 32'd0 || period_y == 32'd0)) begin
                    step_x <= 1'b0;
                    step_y <= 1'b0;
                    state  <= DONE;
                end
            end

            // ──────────────────────────────────────────────
            // DONE — signal completion
            // ──────────────────────────────────────────────
            DONE: begin
                busy   <= 1'b0;
                step_x <= 1'b0;
                step_y <= 1'b0;
                state  <= IDLE;
            end

            default: state <= IDLE;

        endcase
    end
end

endmodule
```

Π.6 laser_ctrl.v - Έλεγχος ενεργοποίησης laser με βάση τον τύπο κίνησης και το busy.

```
0001  // ============================================================
0002  // laser_ctrl.v  — fixed version
0003  // Controls laser ON/OFF based on motion type and busy signal.
0004  //
0005  // FIX: motion is latched when pkt_valid fires (same clock edge
0006  // that step_gen latches its inputs). This avoids the race where
0007  // busy goes HIGH before laser_ctrl sees the new motion value.
0008  //
0009  // Rules:
0010  //   G0 Rapid   (motion=0) → laser OFF  (positioning move)
0011  //   G1 Linear  (motion=1) → laser ON   (engraving move)
0012  //   G2 Arc CW  (motion=2) → laser ON   (engraving move)
0013  //   G3 Arc CCW (motion=3) → laser ON   (engraving move)
0014  //
0015  // Interface:
0016  //   clk       : 50 MHz system clock
0017  //   rst       : synchronous reset, active HIGH
0018  //   motion    : [1:0] from packet_decoder
0019  //   pkt_valid : pulse from packet_decoder when new packet ready
0020  //   busy      : from step_gen (HIGH while moving)
0021  //   laser     : output to laser enable pin (active HIGH)
0022  // ============================================================
0023
0024  module laser_ctrl (
0025      input  wire       clk,
0026      input  wire       rst,
0027      input  wire [1:0] motion,
0028      input  wire       pkt_valid,
0029      input  wire       busy,
0030      output reg        laser
0031  );
0032
0033  reg engrave;   // latched: 1 = this move should fire laser
0034
0035  always @(posedge clk) begin
0036      if (rst) begin
0037          laser   <= 1'b0;
0038          engrave <= 1'b0;
0039      end
0040      else begin
0041          // Latch motion type when packet arrives
0042          // (same cycle step_gen sees pkt_valid)
0043          if (pkt_valid) begin
0044              engrave <= (motion == 2'd1 || motion == 2'd2 || motion == 2'd3) ? 1'b1 : 1'b0;
0045          end
0046
0047          // Laser follows: engrave AND busy
0048          // Turns off immediately when move finishes (busy=0)
0049          laser <= engrave & busy;
0050      end
0051  end
0052
0053  endmodule
```