Department of Electronic & Telecommunication Engineering, University of Moratuwa, Sri Lanka.

Closed Loop Stepper Motor

Group Members:

220169V	Fernando S.R.N.
220276V	Jayathissa M.P.N.V
220134K	Dissanayaka D.M.A.D.
220353F	Lakshan K.P.
220355M	Lakshan R.G.R.
220386H	Manujaya U.G.P.
220472T	Perera P.L.P.
220683P	Weerakoon W.M.B.H

Design Report
EN 2160 Electronic Design Realization

20/04/2025

1 Inclusive Concept Design Model – Abstract Description

The Inclusive Concept Design Model is a structured, iterative framework that emphasizes the importance of making informed, user-centered decisions early in the design process to create accessible and effective solutions for diverse user populations. Rooted in the principles of inclusive design, the model supports designers in answering four fundamental questions:

1. What should we do next? (Manage)
2. What are the needs? (Explore)
3. How can the needs be met? (Create)
4. How well are the needs met? (Evaluate)

These questions are addressed through four interconnected phases:

- **Manage** ensures strategic oversight by reviewing progress, refining goals, planning next steps, aligning stakeholders, and maintaining flexibility to accommodate new insights.
- **Explore** delves into stakeholder and user needs through observation, stakeholder mapping, persona development, and user journey analysis, recognizing diversity and real-world complexity.
- **Create** generates and develops design ideas, combining creativity with practicality through concept development and prototyping, all while challenging assumptions and seeking simplicity.
- **Evaluate** tests concepts against defined criteria, using both expert and user feedback, and quantifies inclusivity by estimating potential exclusion due to physical or cognitive demands.

The model encourages rapid iteration with early evaluation to enable meaningful change, avoiding costly revisions later in development. By systematically linking design activities to key questions, it provides a flexible yet rigorous path to producing inclusive, viable, and user-validated design solutions.

In the design process, we are following this model described above. We were going through 2 iterative cycles.

First Iterative Cycle

2 Stakeholder Map

2.1 Primary Stakeholders (Direct Users & Developers)

Stakeholder	**Role / Interest**	**Needs / Expectations**
Embedded System Engineer	Designs the control system and firmware	Reliable MCU, efficient motor control, testability
Hardware Designer	Designs the PCB and wiring layout	Compatibility of driver, safety, thermal control
Control Systems Engineer	Designs and tunes control algorithms (e.g., FOC)	Performance accuracy, low latency, low jitter
System Integrator	Integrates motor with broader system	Interoperability, modularity

2.2 Secondary Stakeholders (Indirect Users & Supporters)

Stakeholder	Role / Interest	Needs / Expectations
Manufacturing Technician	Assembles and tests the motor system	Easy-to-follow instructions, test interfaces
Maintenance Technician	Diagnoses and fixes issues	Diagnostics features, fault logs
Product Manager	Oversees the design roadmap and cost	Cost-efficiency, market-fit, scalability
End Customer / Operator	Uses the final machine driven by the stepper motor	Smooth performance, low noise, easy operation

2.3 External Stakeholders

Stakeholder	Role / Interest	Needs / Expectations
Suppliers (TI, Toshiba)	Provide MCU and driver ICs	Consistent specs, supply chain reliability
Compliance / Regulatory Bodies	Ensure safety and standards compliance	EMC, thermal, electrical safety certifications
OEM Clients / Partners	Embed your controller into their own machines	Documentation, customization support
Academic Supervisor / Institution	Guides and evaluates your project	Clear documentation, learning outcomes

2.4 Environmental & Social Stakeholders

Stakeholder	Role / Interest	Needs / Expectations
Environment	Affected by component choices & energy use	Efficient design, minimal waste
Future Developers	Will improve or maintain your design	Well-documented code and circuit designs

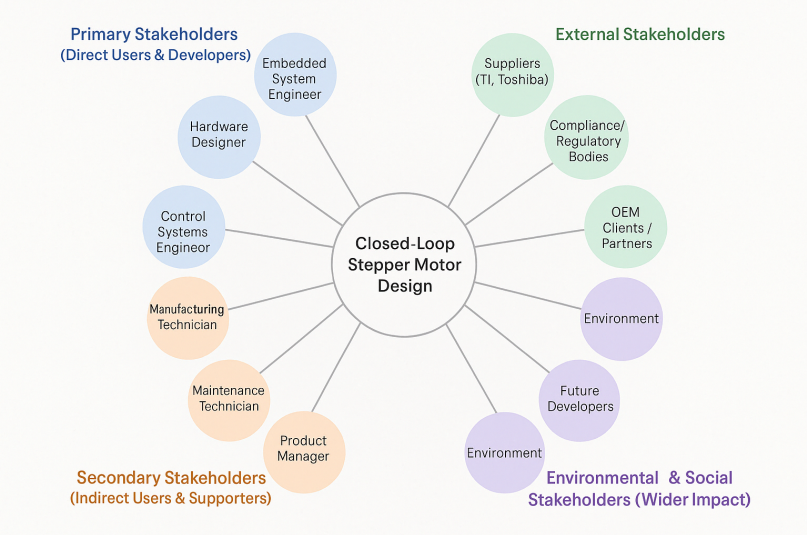


Figure 1: Stake holder map

3 User Observation Summary

To gain a deeper understanding of how users interact with closed-loop stepper motor systems, we conducted user observations through two complementary approaches. Firstly, we analyzed a selection of YouTube videos showcasing real-world applications, configurations, and troubleshooting of stepper motor controllers. This allowed us to observe a diverse range of user behaviors, workflows, and challenges across different contexts and setups.

Secondly, we directly engaged with students and hobbyists involved in robotics projects within our university. By observing their hands-on interaction with motor drivers, control systems, and tuning interfaces, we were able to identify usability issues, common workarounds, and critical user needs in a more contextualized environment. These observations have provided valuable insights into both the practical and experiential aspects of using closed-loop stepper motor systems, helping to inform our design process with a focus on inclusivity and ease of use.

4 Generate Personas

4.1 Robot Design Student (Mechanical)

- **Age:** 23
- **Background:** Second-year engineering student
- **Technical Level:** Beginner
- **Needs:** Simple wiring, minimal tuning effort, documentation with example code
- **Goals:** Use the stepper motor for an autonomous robotic arm project
- **Challenges:** Limited time and needs quick prototyping; often juggles between multiple systems
- **Quote:** "I just want it to work out of the box with minimal fiddling."

Robot Design Student (ENTC)

- **Age:** 22
- **Background:** First-year undergraduate
- **Technical Level:** Beginner
- **Needs:** Safe to use, easy setup, user-friendly UI for configuration
- **Goals:** Create robots with better accuracy and efficiency on a low budget
- **Challenges:** Lacks advanced electronics background; often needs support for troubleshooting
- **Quote:** "It should be something that can be used without a heavy understanding of the steps."

4.2 Industrial Technician

- **Age:** 38
- **Background:** Works in factory automation
- **Technical Level:** Advanced
- **Needs:** High reliability, diagnostics for closed-loop control, fine control over PID parameters

- **Goals:** Integrate the motor into a conveyor belt system with tight feedback control
- **Challenges:** Downtime is expensive, so quick diagnostics and rugged hardware are essential
- **Quote:** "I need to trust it under continuous load—no surprises."

4.3 ENTC Undergraduate

- **Age:** 25
- **Background:** Final-year engineering student
- **Technical Level:** Intermediate
- **Needs:** Simple use, better interface and libraries
- **Goals:** Use stepper motors in projects such as robot arms
- **Challenges:** Time limitation and adaptation for various occasions
- **Quote:** "Managing should be easy."

5 Capture Needs List

User Needs Report – Nirwan

- Precision and Accuracy
- Stall Detection and Recovery
- Efficiency and Power Optimization
- Smooth Motion and Vibration Control
- High-Speed Performance Without Step Loss
- Automatic Load Compensation
- Self-Tuning and Quick Calibration
- Fault Detection and Safety Mechanisms
- Cost-Effectiveness in Long-Term Operation

User Needs Report – Ravindu

- Precise position control (feedback system like an encoder)
- High efficiency (reducing power loss and heating)
- Automatic correction of position errors (closed-loop feedback)
- Smooth and quiet operation (reducing vibrations and noise)
- Easy integration with controllers (Arduino, STM32, or PLCs)
- Overload protection & fault detection

User Needs Report - Luchitha

User Need	Description	Priority
High Positional Accuracy	To ensure precise placement of components, fabric cuts, or printed labels.	High
Step Loss Prevention	Users need motors that don't skip steps during high-speed or high-load conditions.	High
Closed-Loop Feedback	Real-time position correction to improve reliability.	High
Energy Efficiency	Reduce heat generation and power wastage.	Medium
Compatibility with Existing Controllers	Easy integration with PLCs, Arduino, or industrial drivers.	High
Reduced Maintenance	Motors that require less frequent checks and reduce production downtime.	Medium
Noise Reduction	Motors that operate quietly, especially important in clean or calm environments.	Low

Table 1: Identified User Needs for 2-Phase Closed-Loop Stepper Motors

User Needs Report – Pasindu Lakshan

- High-resolution encoders for precise position feedback
- Advanced driver circuits for smooth and quiet operation
- Adaptive control algorithms for real-time speed and torque adjustments
- Easy-to-use software interface for parameter tuning
- Robust housing and thermal management for durability
- Compliance with industrial and safety standards

User Needs Report – Pasindu Manujaya

Performance Requirements

- High positioning accuracy with closed-loop feedback
- Repeatable and smooth movement control
- Minimal step loss with automatic correction
- High torque-to-efficiency ratio
- Adjustable speed control for dynamic applications

Reliability & Thermal Management

- Low heat generation for long-term reliability

Integration & Compatibility

- Compatibility with PLCs, drivers, and controllers
- Simple wiring and plug-and-play operation
- Compact design for space-constrained applications

Safety & Protection Features

- Protection against overcurrent and voltage spikes

- Emergency stop functionality
- Low noise and electromagnetic interference

Economic Considerations

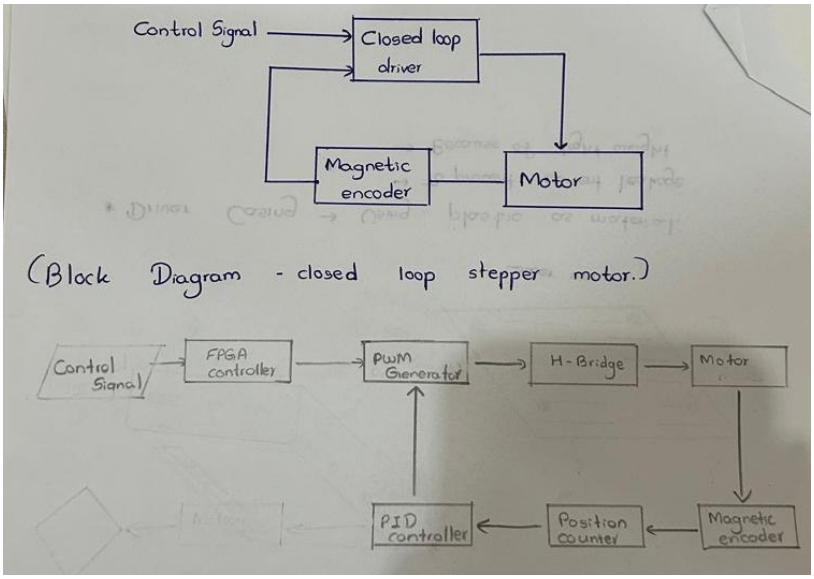
- Affordable yet high-performance solutions
- Energy efficiency to reduce operational costs
- Availability of long-term support and spare parts

User Needs Report – Hansana

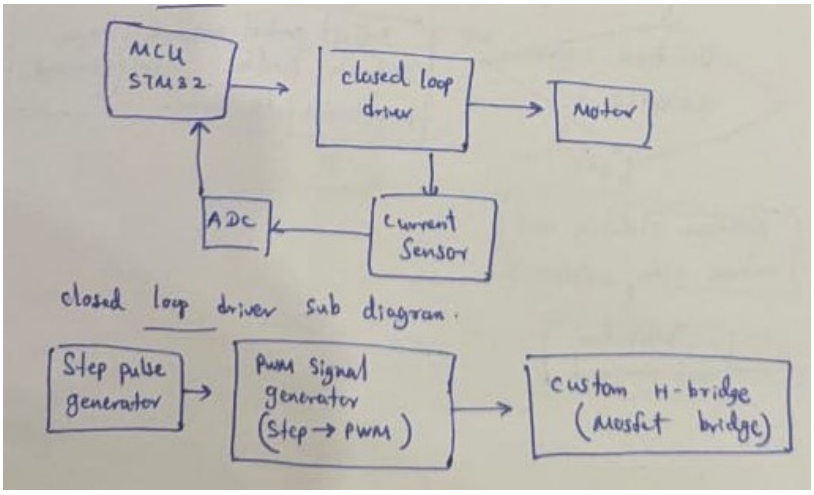
User Need	Requirement	Priority
High accuracy and precision	Position error $\leq 0.1°$	High
Real time feedback	Closed loop encoder integration	High
Smooth operation	Minimal vibration and resonance	Medium
High torque efficiency	Adaptive current control	High
Accessible interface design	Controls and displays that accommodate users with varying physical abilities	High
Clear visual indicators	High contrast displays for users with visual impairments	Medium
Auditory feedback option	Sounds alerts for critical notification	Medium
Energy efficiency	Power consumption optimization	Medium
Easy integration	Compatibility with common controllers (Arduino, Raspberry-pi)	High
User-friendly interface	Simple GUI for setup and monitoring	Medium
Fault detection	Automatic error correction and alarms	High

6 Block Diagrams

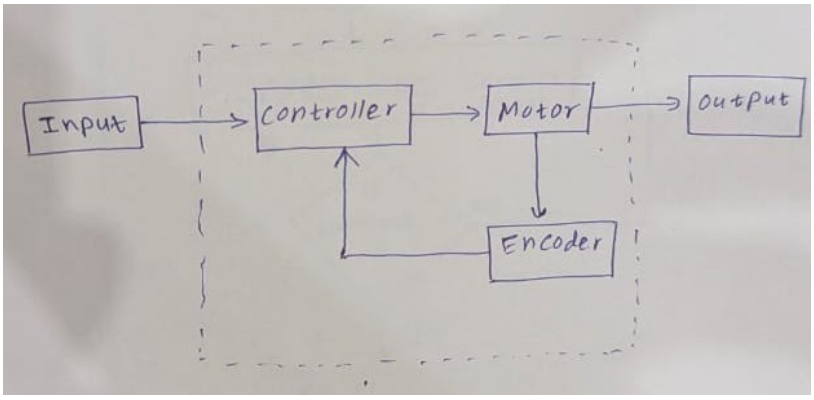
Block Diagram - Nirwan



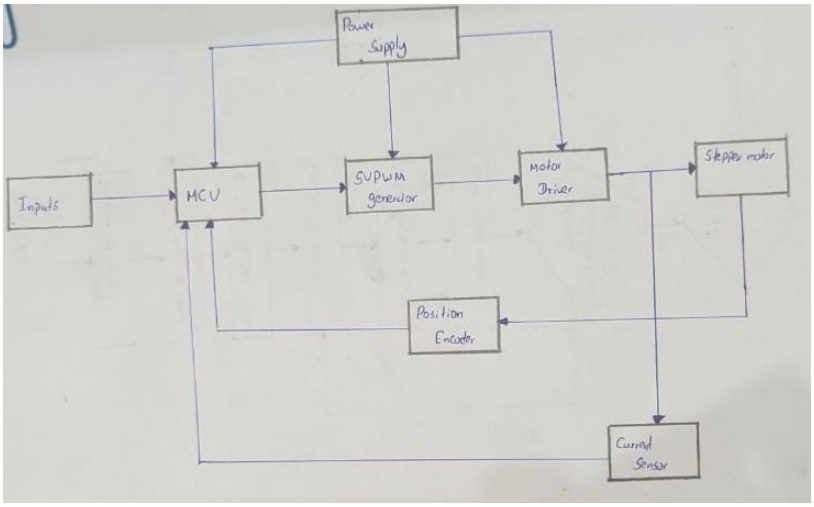
Block Diagram - Hansana



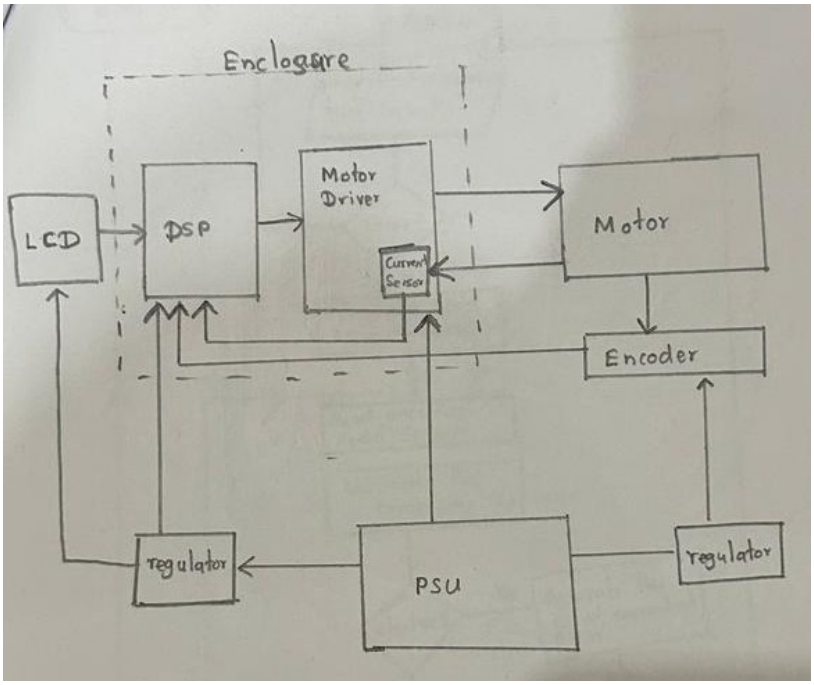
Block Diagram - Ravindu



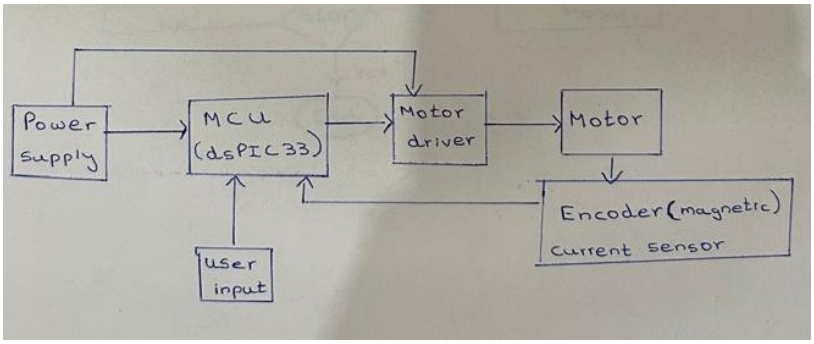
Block Diagram - Akila



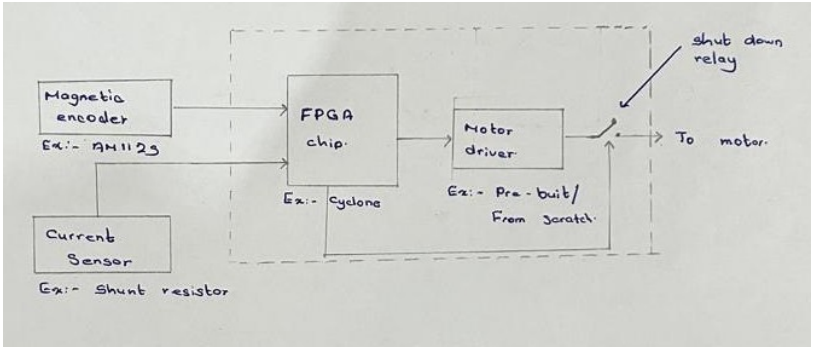
Block Diagram - Luchitha



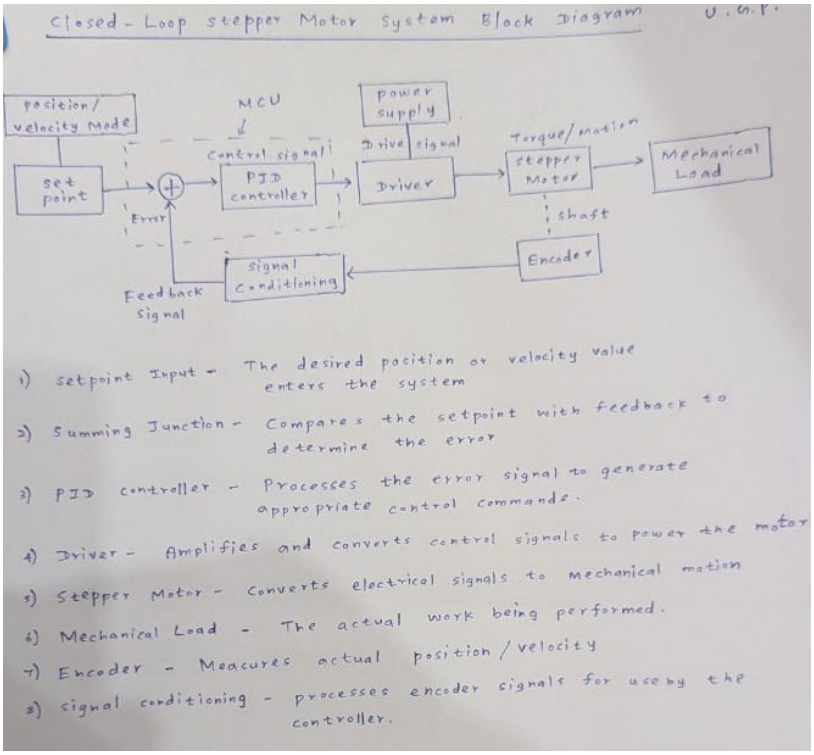
Block Diagram - Rashane



Block Diagram - Pasindu

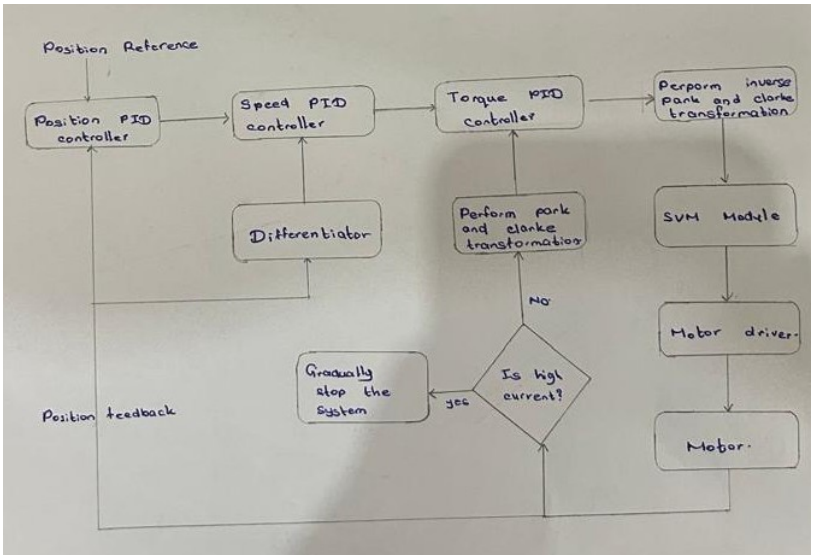


Block Diagram - Manujaya

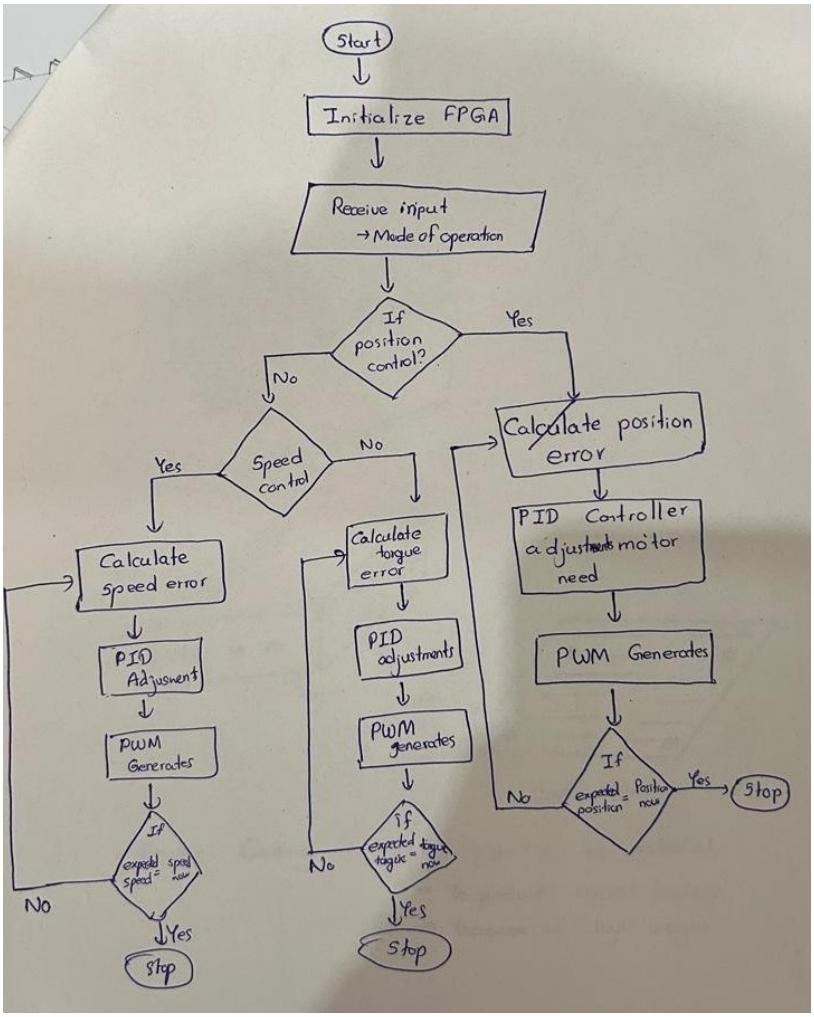


7 Flow Charts

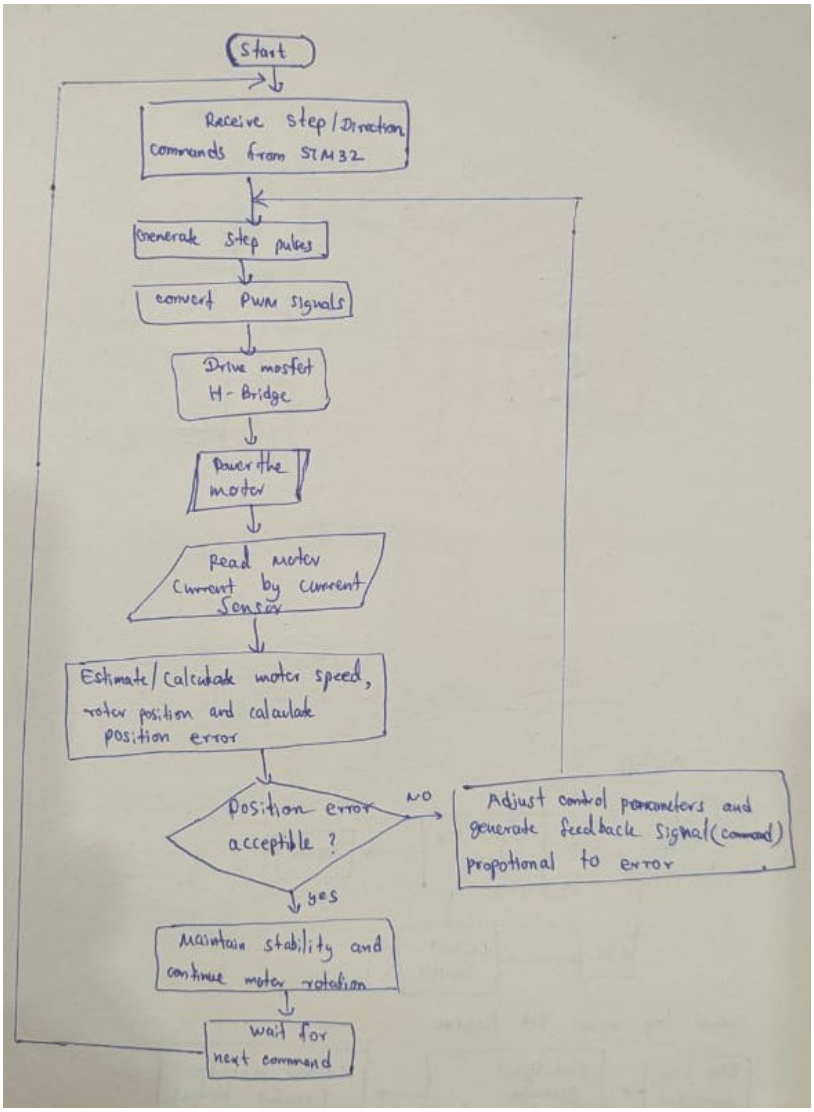
Flow Chart - Pasindu



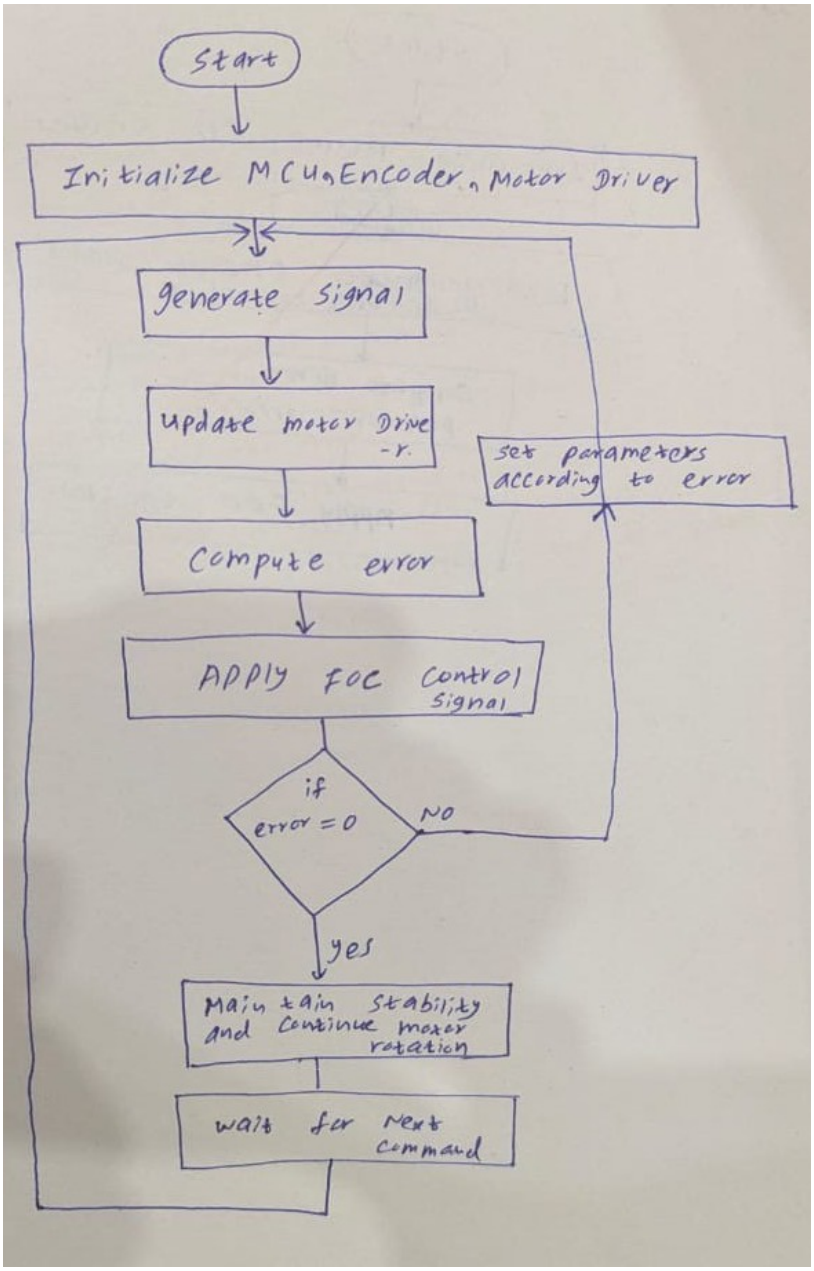
Flow Chart - Nirwan



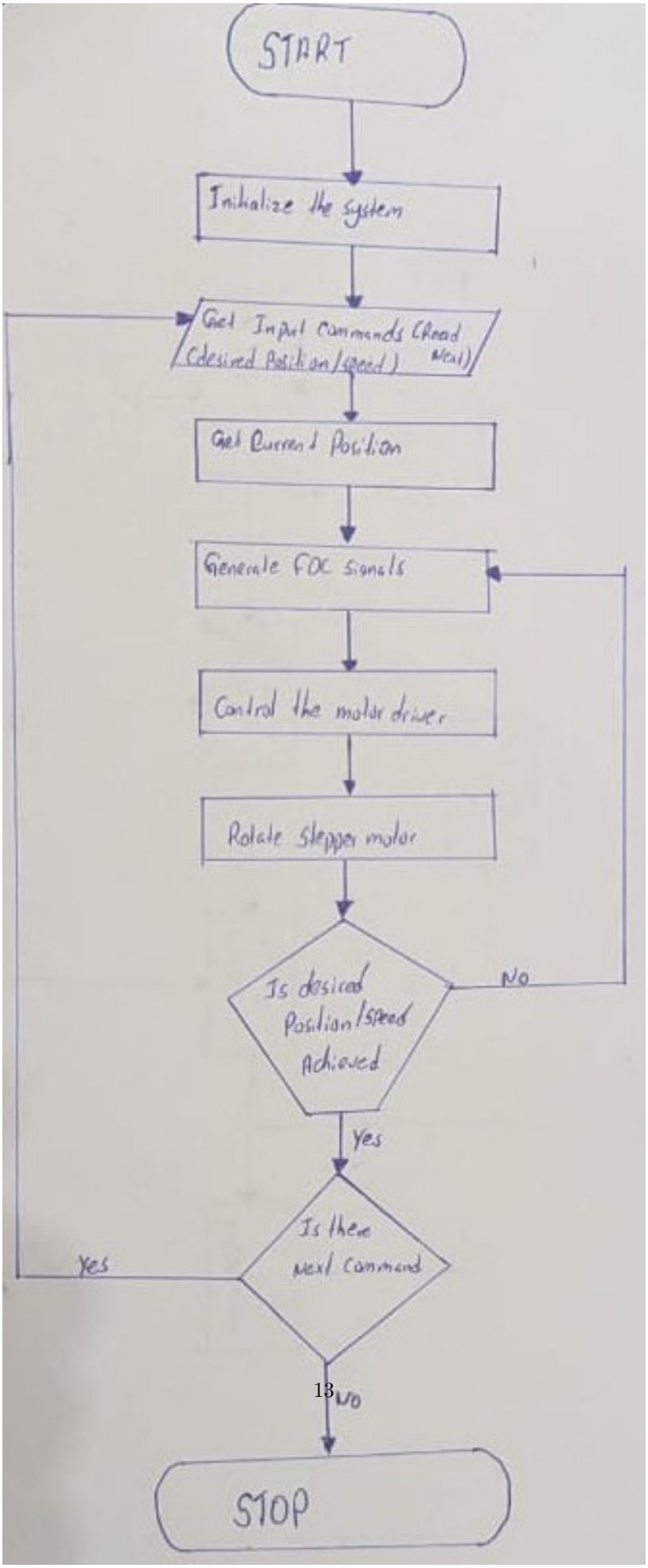
Flow Chart - Hansana



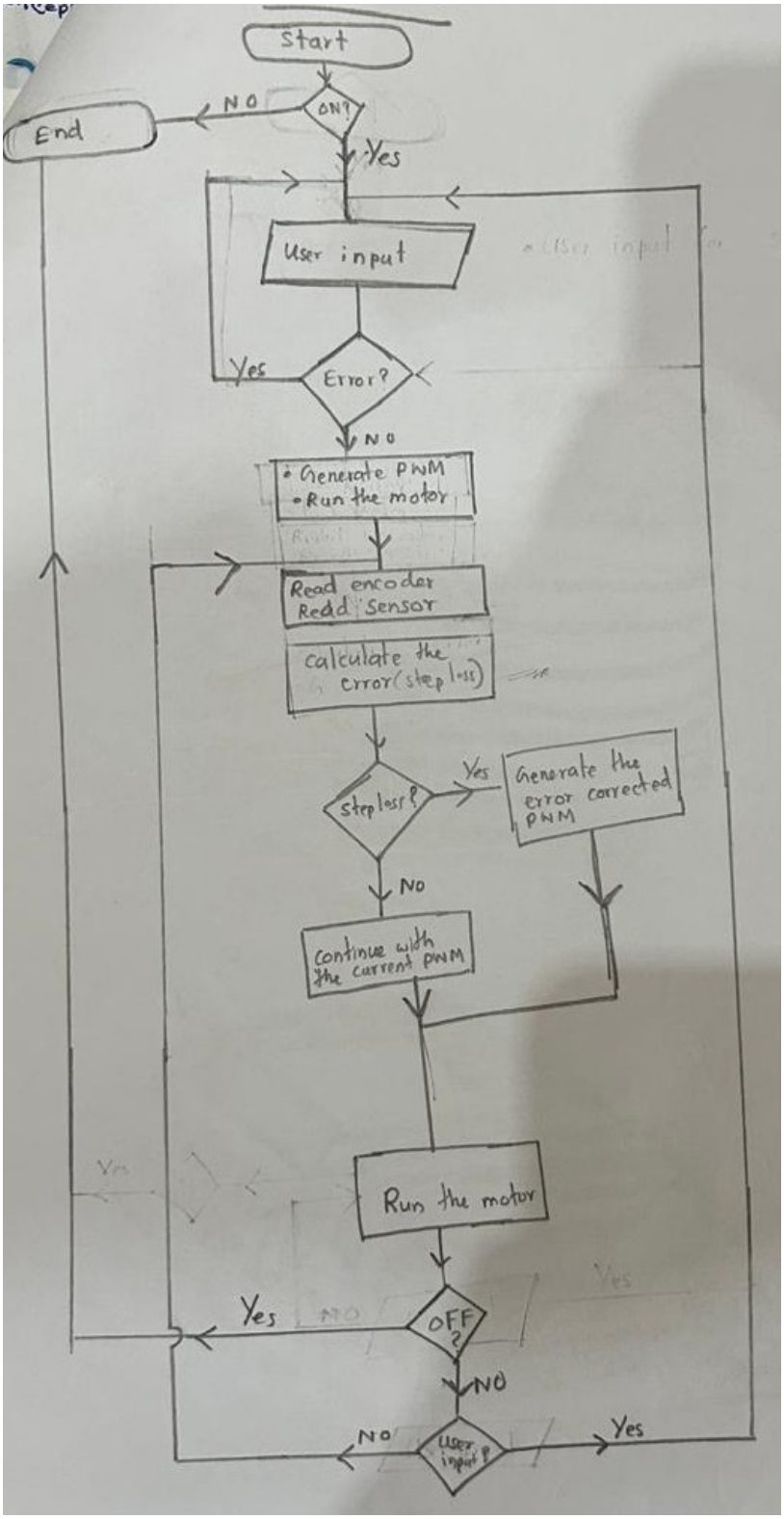
Flow Chart - Ravindu



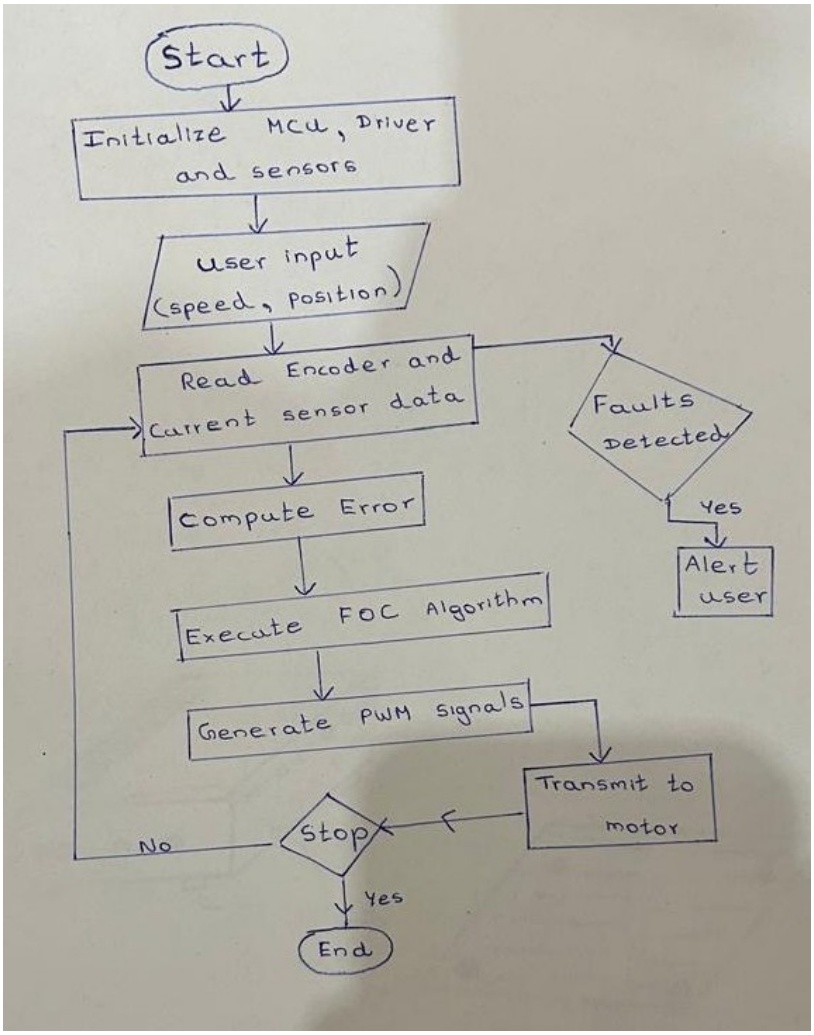
Flow Chart - Akila



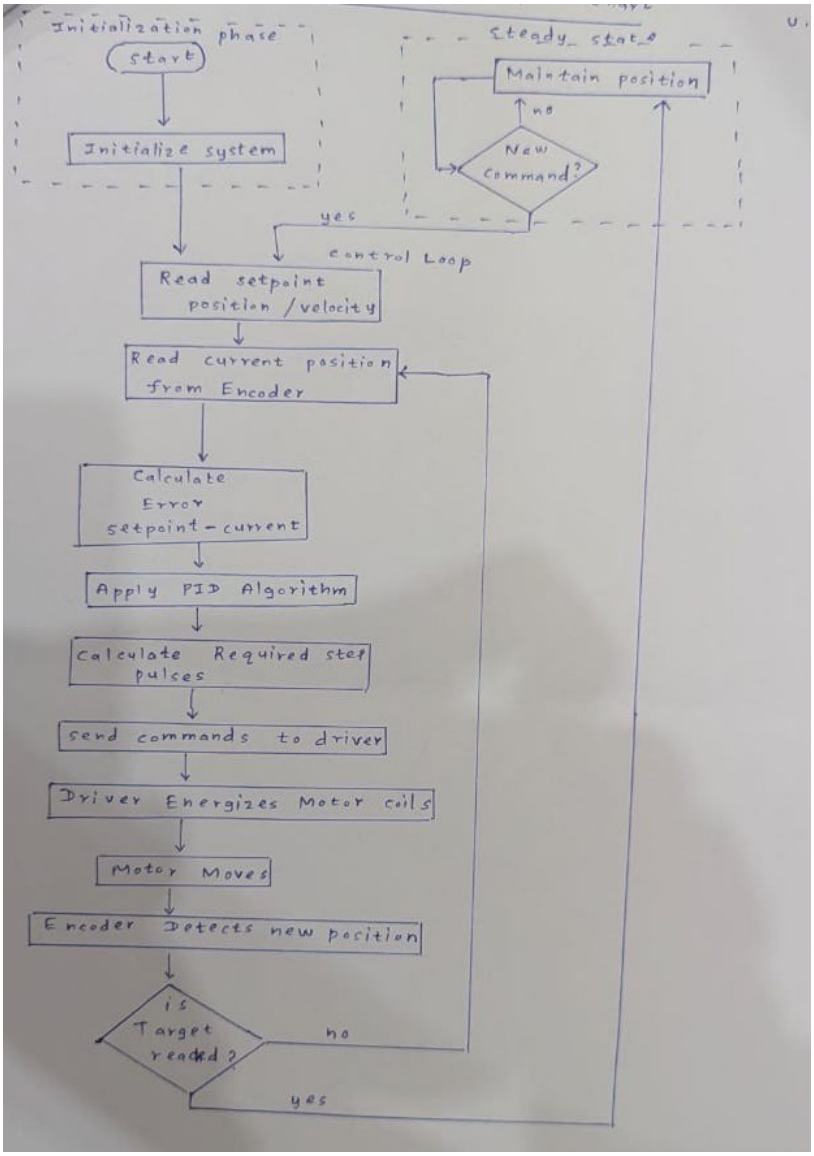
Flow Chart - Luchitha



Flow Chart - Rashane

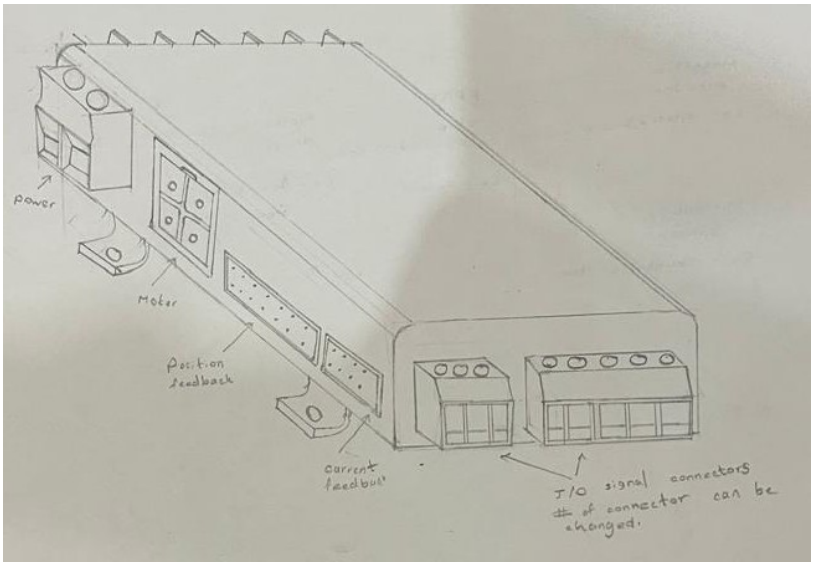


Flow Chart - Manujaya

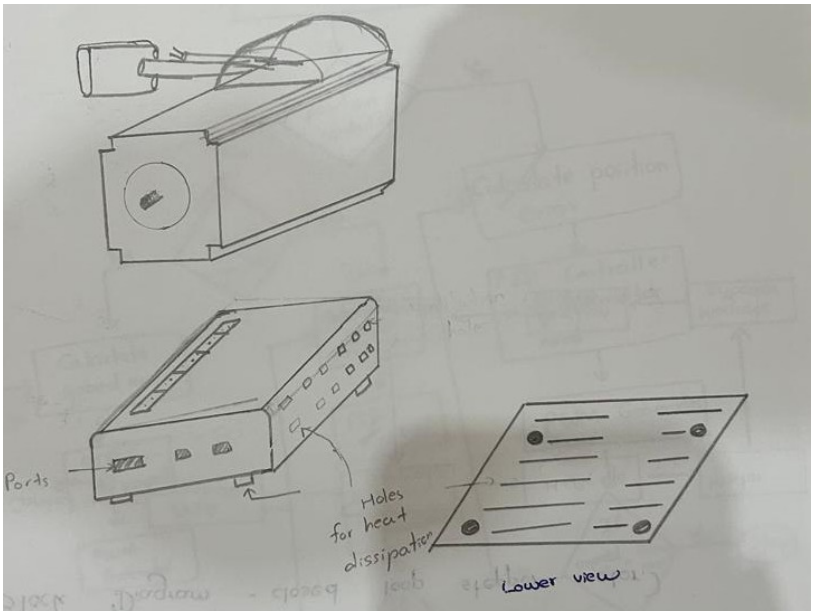


8 Conceptual Design

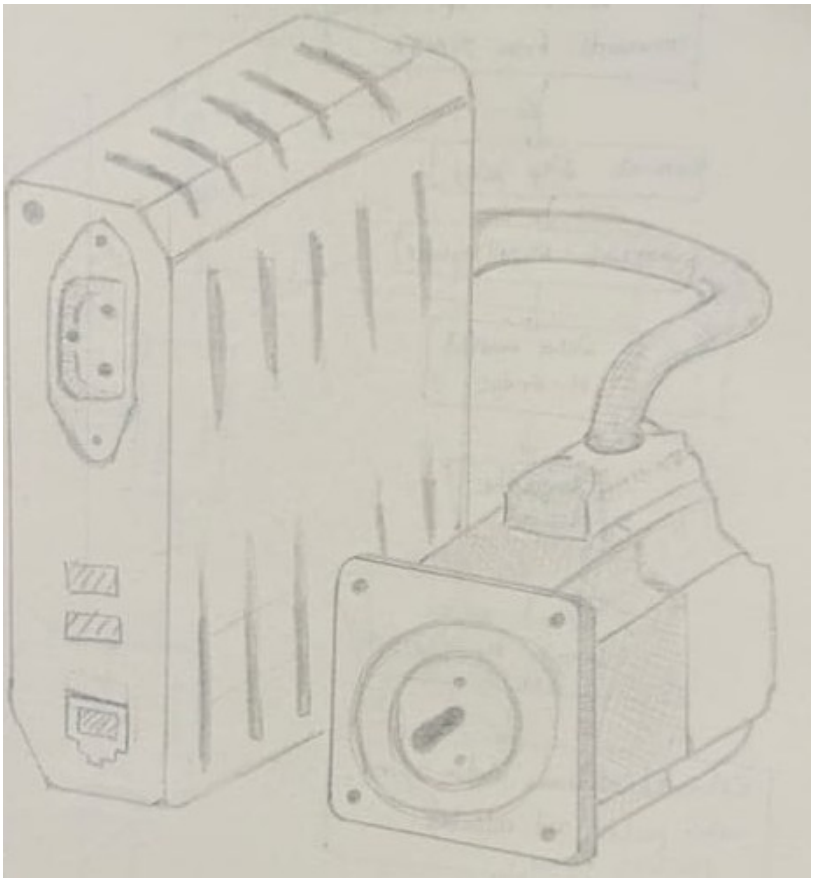
Conceptual Design - Pasindu



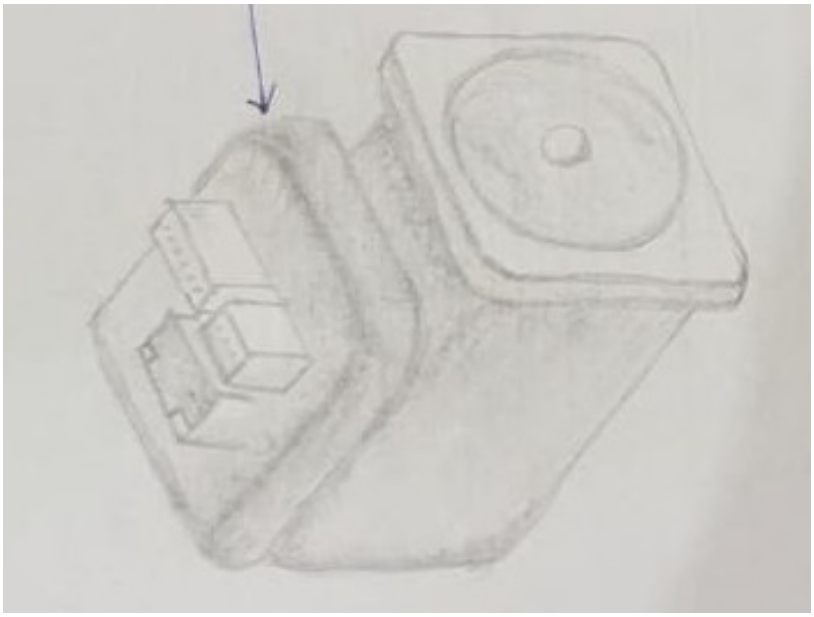
Conceptual Design - Nirwan



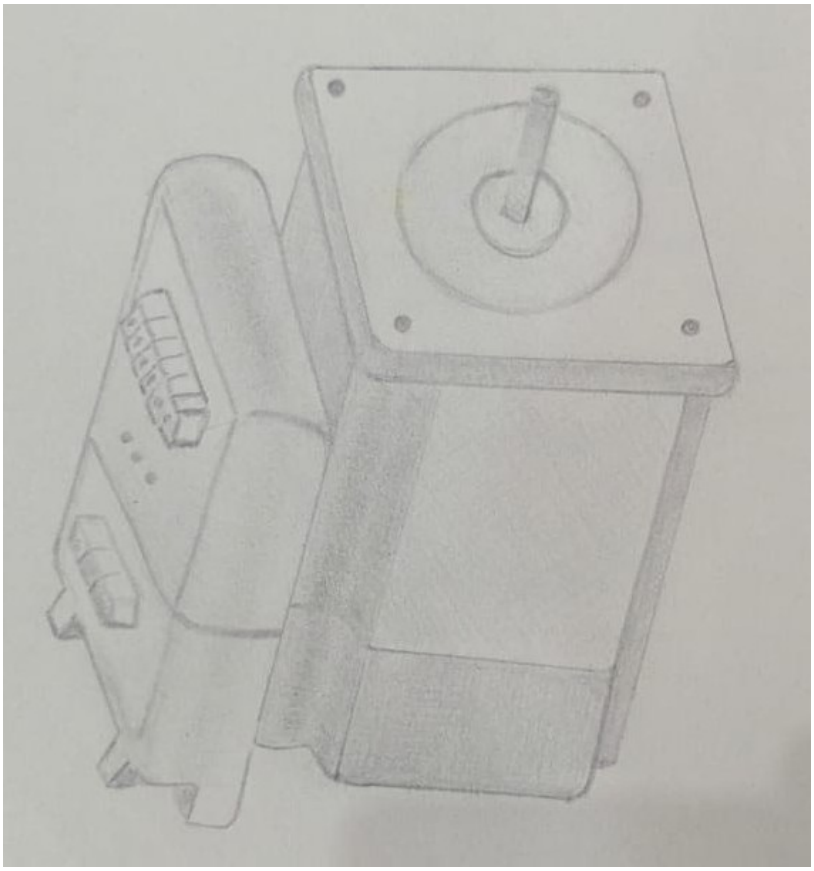
Conceptual Design - Hansana



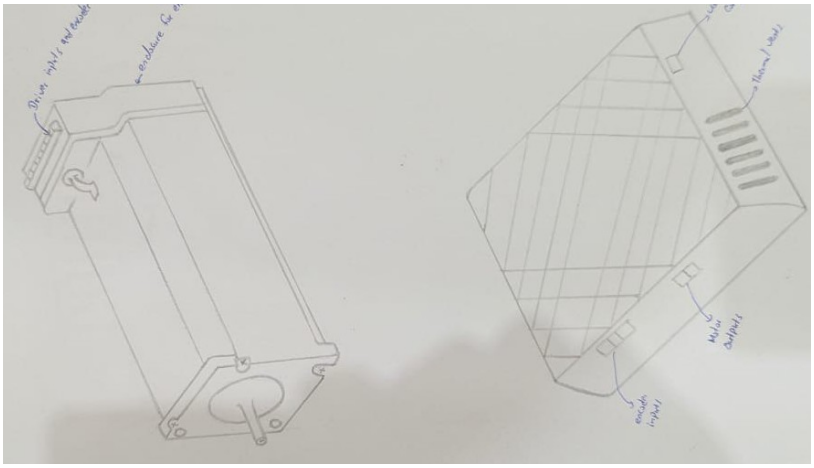
Conceptual Design - Manujaya



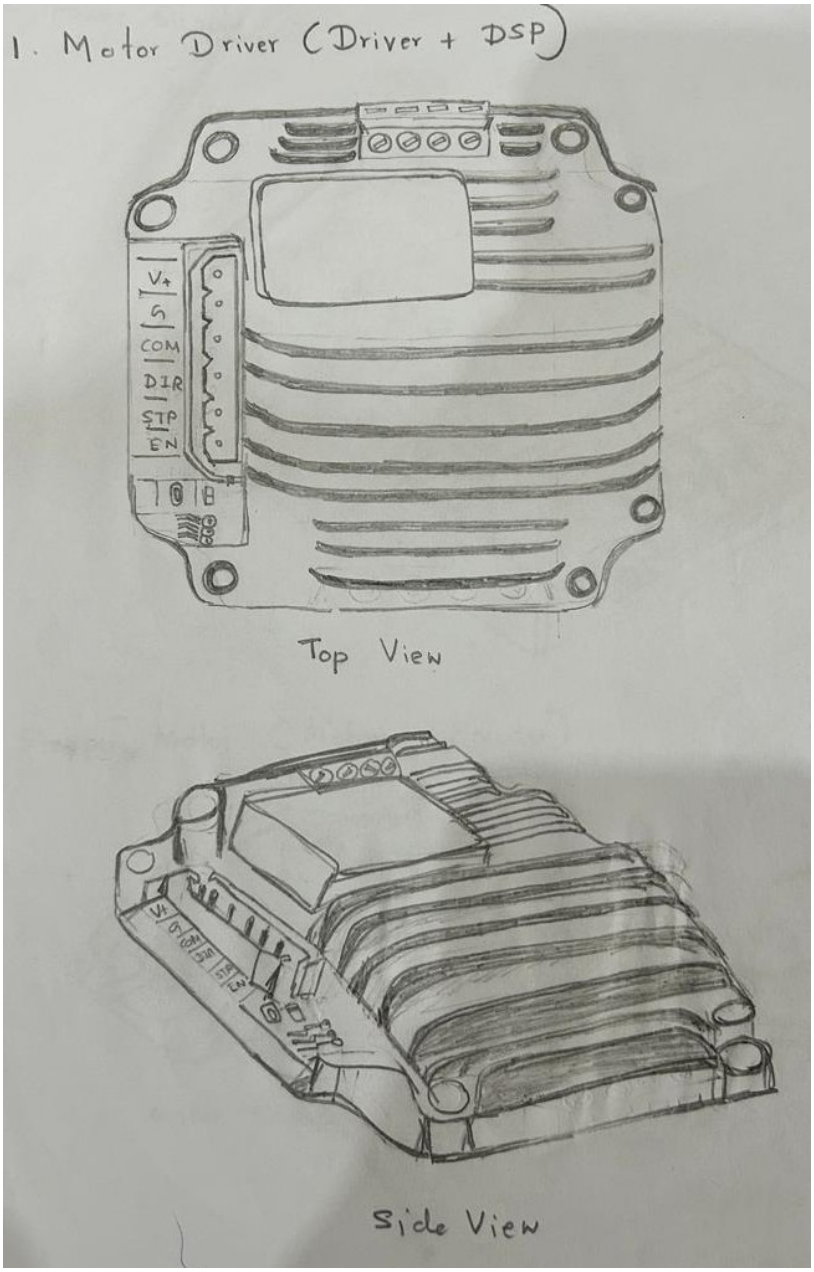
Conceptual Design - Ravindu



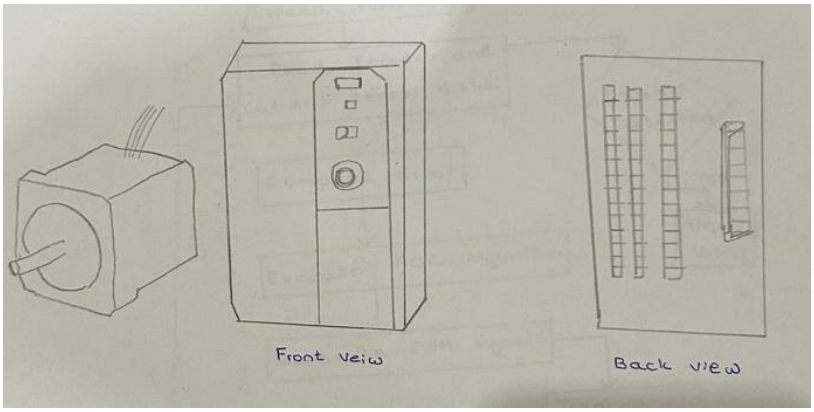
Conceptual Design - Akila



Conceptual Design - Luchitha



Conceptual Design - Rashane



9 Evaluate the designs

Design	User Need	Marks given							
		RASHANE	LUCHITHA	PASINDU	AKILA	MANUJAYA	NIRWAN	HANSANA	RAVINDU
Enclosure	Aesthetic Looks (5)	3	4	4	3	5	4	5	4
	Durability (5)	4	5	4	4	3	4	4	3
	Heat Dissipation (5)	4	5	5	2	4	5	2	2
	Compactness (5)	3	3	4	3	4	3	4	5
	Electromagnetic Interference (5)	4	4	3	4	4	4	4	2
	Safety measures (5)	4	5	4	3	5	4	4	3
	Overall (30)	22	26	24	19	25	24	23	19
Block Diagram	Fulfillment of Main functionality (10)	7	10	10	8	7	8	8	9
	Completeness of Components (5)	3	5	5	4	3	3	4	4
	Accuracy and Precision (5)	4	4	5	5	3	5	3	4
	High Speed Performance (5)	3	4	5	5	3	4	3	3
	Fault detection mechanisms (5)	3	4	5	4	2	4	2	4
	Implementation (10)	9	9	6	8	9	6	9	8
	Overall (40)	29	36	36	34	27	25	29	32
Flowchart	Logical and flow operation (5)	4	4	4	4	5	4	4	4
	Completeness of process (10)	8	9	9	8	9	8	9	9
	Operation modes (10)	6	8	8	7	8	7	8	6
	Algorithm Complexity (10)	8	9	4	4	9	6	8	8
	Overall (35)	26	30	25	23	31	25	31	27
Overall Marks for Design (100)		77	92	85	76	83	74	83	78

Figure 2: Evaluation of preliminary designs

Second Iterative Cycle

During the second development phase, we established the foundational steps of the project, including the initial setup, architecture definition, and system planning. This phase also involved creating a cross-pollinated design. A key focus at this stage was component selection, where we evaluated and chose the most suitable microcontrollers, motor drivers, sensors, and other essential components based on performance, compatibility, and scalability.

10 User Needs

10.1 Precision & Control

- **High positioning accuracy:** Users expect accurate control over motor position, especially in CNC, robotics, and medical devices.
- **Repeatability:** Consistent performance over repeated movements with minimal jitter or drift.
- **Closed-loop feedback with encoders:** Necessary to detect and correct positional errors in real time.
- **High-resolution encoders:** Improve control precision and ensure system responsiveness.

10.2 Performance & Efficiency

- **High torque-to-efficiency ratio:** Motors should deliver necessary force without excessive power consumption.
- **Energy efficiency:** Especially important for battery-powered or portable systems.
- **High-speed operation without step loss:** Required in fast-paced systems like conveyor belts or textile machines.
- **Smooth and quiet motion:** Essential for both mechanical integrity and noise-sensitive environments.
- **Adaptive speed and torque control:** Dynamic response to varying load and task conditions.

10.3 Robustness & Reliability

- **Automatic correction of step loss:** Prevent cumulative errors during operation.
- **Load compensation:** Real-time adjustments under varying mechanical loads.
- **Thermal management:** Minimize heat buildup to ensure system longevity.
- **Low vibration and EMI:** To avoid interference with sensitive equipment and extend motor life.

10.4 Safety & Protection

- **Overload protection and fault detection:** Ensure system halts or recovers safely during faults.
- **Emergency stop mechanism:** Essential in industrial and medical environments.
- **Protection against overvoltage and current spikes:** To safeguard motor and control electronics.

10.5 Usability & Integration

- **Self-tuning and quick calibration:** Allow non-expert users to configure systems easily.
- **User-friendly interface for parameter tuning:** Simple software tools for customizing motor behavior.
- **Compatibility with controllers:** Seamless integration with Arduino, STM32, PLCs, and other platforms.

- **Plug-and-play design:** Simplified setup and wiring, reducing deployment time.
- **Compact form factor:** For systems with space constraints.

10.6 Economic Considerations

- **Cost-effective and scalable:** Balance between performance and affordability for educational, industrial, and prototyping use.
- **Long-term support and spares availability:** Essential for industrial users and institutions requiring long-lasting systems.
- **Energy-efficient operation:** Reduce long-term power costs in continuous-use environments.

11 Overview of the Design

This project focuses on the design and implementation of a **Closed-Loop Stepper Motor Control System** optimized for precision motion control applications. Traditional open-loop stepper systems suffer from performance limitations such as missed steps, noise, and inefficiency under varying loads.

To address these challenges, our system integrates advanced control techniques including:

- **Field-Oriented Control (FOC)**
- **Current sensing**
- **Vector Space Modulation (VSM)**
- **Digital Signal Processing (DSP)**

The core objective of the project is to enhance the accuracy, efficiency, and dynamic response of a standard two-phase stepper motor by implementing a feedback-driven control loop. Using a TI C2000 DSP microcontroller, the system continuously monitors the rotor position and motor currents, allowing it to actively regulate torque and maintain stability even under load disturbances or rapid speed changes.

11.1 Key Features

- Rotor position feedback using encoders for closed-loop precision
- Current sensing to support real-time current regulation and fault detection
- FOC algorithm to enable smoother and more efficient motor operation
- Vector Space Modulation for optimal voltage utilization and precise PWM control
- Integration with a power driver IC (e.g., TB67H400AFTGEL) to interface the DSP with the motor

11.2 Objective

Through this project, we aim to develop a high-performance stepper control system that bridges the gap between stepper and servo systems. The final outcome will be a working prototype capable of reliable, low-noise, high-torque motor control suitable for use in applications such as:

- Robotics(Our focus is mainly for this)
- Industrial automation
- CNC systems
- Precision positioning tools

11.3 Basic Concepts Needed

11.3.1 Field oriented Control

Field-Oriented Control (FOC), or vector control, is a control technique that enables precise torque and speed control in AC machines. Although traditionally applied to Permanent Magnet Synchronous Motors (PMSM) and Brushless DC Motors (BLDC), FOC can also be effectively adapted to **2-phase closed-loop stepper motors** to enhance their dynamic performance and energy efficiency.

Why FOC for Stepper Motors?

Conventional open-loop stepper motors operate based on fixed current sequences, which can lead to step losses under high loads or speeds. By integrating FOC with a closed-loop feedback system (e.g., using encoders), the stepper motor can operate more like a servo motor with the following benefits:

- **Closed-loop position correction**: Prevents missed steps.
- **Torque control**: Adjusts current based on load, improving efficiency.
- **Smoother motion**: Reduces torque ripple and mechanical noise.
- **Lower heating**: Since current is modulated as needed.

Control Principle

1. The 2-phase currents (I_α, I_β) are measured from the motor.
2. Clarke and Park transforms convert these currents into i_d and i_q in the rotating reference frame.
3. A position sensor (e.g., encoder) provides real-time rotor angle feedback.
4. PI controllers regulate i_d and i_q to maintain the desired flux and torque.
5. Inverse Park and Clarke transforms convert the control output back into two-phase voltages.
6. A PWM inverter applies these voltages to the stepper motor windings.

FOC transforms the traditional stepper motor into a high-performance actuator with servo-like behavior, making it a practical solution for precision-driven industrial applications in Sri Lanka.

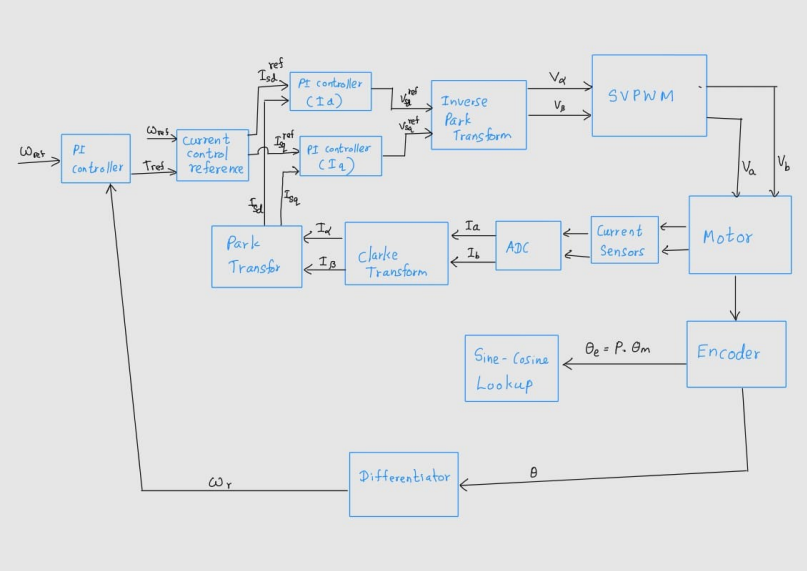


Figure 3: Block Diagram of the FOC

11.3.2 Current Sensing

Current sensing is a fundamental component in closed-loop motor control systems such as Field-Oriented Control (FOC). It provides the real-time feedback necessary to regulate motor torque, prevent overcurrent conditions, and enable precise dynamic performance.

In the context of FOC for 2-phase stepper motors, accurate current measurement is essential to compute the motor's instantaneous torque and flux-producing components (i_d and i_q). These are used in the control loop to adjust the PWM outputs accordingly.

Why Current Sensing is Crucial

- **Feedback for FOC:** Current readings are transformed into the rotating reference frame to control torque and magnetic flux independently.
- **Overcurrent Protection:** Detects anomalies and protects motor and driver circuitry.
- **Improved Efficiency:** Enables dynamic current modulation based on load, minimizing unnecessary power consumption.
- **Thermal Management:** By controlling average current, it reduces heating in the motor and driver components.

Techniques for Current Sensing

- **Shunt Resistors:** Low-cost, simple method placed in series with the motor windings, but require precise amplification.
- **Hall-effect Sensors:** Non-intrusive and capable of handling higher currents with galvanic isolation.
- **Current Sense Amplifiers (CSAs):** High-accuracy amplifiers like the **INA241B** are designed to work with low-side or high-side shunt resistors and provide precise, bidirectional current measurements.

Implementation in This Project

We selected the **INA241B2IDDFR**, a high-precision, bidirectional current sense amplifier from Texas Instruments, due to the following benefits:

- $\pm 20\,V$ Common-Mode Range: Allows flexible placement in high-side or low-side configurations.
- Integrated Filtering: Reduces noise without external components.
- Low Offset and High Accuracy: Ensures precise current readings critical for FOC loops.
- I^2C Interface (in INA241x series): Enables digital interfacing in more advanced implementations.

These current measurements are fed into the DSP (TMS320F28069), where they undergo Clarke and Park transformations before entering the PI controllers. This tightly integrated feedback loop ensures optimal performance and protection in the closed-loop stepper motor system.

11.3.3 Space Vector Pulse Width Modulation (SVPWM)

Space Vector Pulse Width Modulation (SVPWM) is an advanced modulation technique used to drive AC motors efficiently. Although stepper motors are typically driven in an open-loop fashion with simple pulse signals, closed-loop control using Field-Oriented Control (FOC) allows the stepper motor to behave more like a synchronous AC motor. This enables the use of SVPWM to optimize performance.

Concept of SVPWM

SVPWM generates motor drive signals by treating the three-phase (or in the case of 2-phase systems, equivalent α-β) voltage as a rotating vector in a 2D space. The objective is to approximate this rotating vector using a combination of six fixed inverter switching vectors and two zero vectors within a PWM cycle.

- The rotating reference voltage vector is synthesized using time-weighted application of adjacent switching states.
- This technique allows full utilization of the DC bus voltage.
- Harmonic distortion is reduced, improving efficiency and reducing torque ripple.

SVPWM Steps

1. The desired voltage vector is computed from the d-q components using inverse Park transformation.
2. This vector is transformed into the α-β frame (equivalent to 2-phase representation).
3. The sector (out of six) where the voltage vector lies is identified.
4. Switching times for the active vectors and zero vectors are calculated.
5. PWM pulses are generated accordingly to control the inverter that drives the stepper motor.

Advantages of SVPWM for Stepper Motors

- **Improved Efficiency:** Minimizes harmonic losses in the motor.
- **Lower Torque Ripple:** Results in smoother motion.
- **Better DC Bus Utilization:** Up to 15% more effective than sine PWM.
- **Silent Operation:** Reduces audible noise due to optimized switching.

SVPWM is a modern, efficient modulation technique that brings servo-like performance to closed-loop 2-phase stepper motors. Its implementation can lead to higher precision, lower noise, and better energy efficiency in automation systems.

11.3.4 Digital Signal Processing

Digital Signal Processing (DSP) plays a critical role in implementing high-performance control algorithms like Field-Oriented Control (FOC) in real-time embedded systems. For motor control, DSP enables precise computation of mathematical transformations, filtering, and control law execution at high speed and accuracy.

Importance in Motor Control

In closed-loop stepper motor control, DSP is used to:

- Perform Clarke and Park transformations to convert phase currents to a rotating reference frame.
- Execute PI controllers to regulate torque-producing and flux-producing current components (i_d, i_q).
- Apply inverse transforms and generate PWM duty cycles for the inverter using techniques like SVPWM.
- Filter noisy sensor data and ensure system stability via fast feedback loops.

DSP in This Project

This project utilizes the **TMS320F28069PFPS** MCU, which belongs to the Texas Instruments C2000 family — a line of DSP-enabled microcontrollers specifically optimized for motor control. Key DSP-related features include:

- **32-bit Floating Point CPU:** Allows efficient execution of real-time control algorithms with high accuracy.
- **Trigonometric Math Unit (TMU):** Hardware-accelerated computation of sine/cosine and other nonlinear functions used in FOC.
- **High-Speed ADC and PWM Modules:** Ensure fast and deterministic sampling and actuation.
- **CLA (Control Law Accelerator):** A dedicated co-processor that handles time-critical control loops independently from the main CPU.

With these features, the TMS320F28069 enables precise current regulation, rapid response to load changes, and efficient implementation of SVPWM and FOC algorithms — all essential for a high-performance closed-loop stepper motor system.

12 Component Selection

12.1 Motor Selection

Overview of Stepper Motors

Stepper motors are electromechanical devices that convert electrical pulses into discrete mechanical movements. Unlike conventional DC motors, which rotate continuously, stepper motors move in fixed steps, making them ideal for applications requiring precise position control without the need for complex feedback mechanisms.

Working Principle

A stepper motor consists of a rotor (rotating part) and a stator (stationary part with coils). When stator windings are energized in a specific sequence, the rotor aligns with the magnetic field and rotates incrementally. Each electrical pulse results in movement by a fixed step angle (typically 1.8°, or 200 steps per revolution).

Types of Stepper Motors

- **Permanent Magnet (PM) Stepper**
- **Variable Reluctance (VR) Stepper**
- **Hybrid Stepper** (most commonly used)

Applications

- Robotics (Main Focus)
- CNC Machines
- 3D Printers
- Camera Gimbals
- Medical Equipment
- Automated Manufacturing

Why Choose NEMA 17 for Closed-Loop Stepper Design

NEMA 17 refers to a stepper motor with a 1.7 inch × 1.7 inch (43.2 mm × 43.2 mm) faceplate. It is one of the most widely used motor sizes in hobbyist, academic, and industrial applications.

1. Standardization and Compatibility

- Universally standardized size with consistent mounting holes and shaft dimensions.
- Easy integration into mechanical frames, CNC/3D printer structures, and existing control systems.

2. Compact but Powerful

- Excellent balance between size and torque output (up to 45 N·cm).
- Suitable for space-constrained designs requiring moderate power.

3. Optimized for Closed-Loop Control

- Easily paired with encoders for feedback-based control.
- Supports advanced control strategies like Field-Oriented Control (FOC), current sensing, and stall detection.

4. High Resolution and Smooth Motion

- 1.8° step angle (200 steps/rev), microstepping support.
- High-resolution motion possible when combined with encoders and closed-loop drivers.

5. Cost-Effective and Readily Available

- More affordable than NEMA 23 or servo motors.
- Ideal for scalable designs, educational use, and rapid prototyping.

6. Strong Community and Documentation Support

- Abundant tutorials, datasheets, and open-source libraries.
- Great for reducing development time and troubleshooting complexity.

7. Efficient with Modern Drivers

- Compatible with efficient drivers (e.g., TB67H400AFTGEL, TMC2209).
- Closed-loop current regulation and Vector Space Modulation (VSM) enhance overall energy efficiency.

12.2 MCU Selection

Column1	TI TMS320F28069PFPS	NXP MPC5744P	STM32G474RET6	Infineon XMC480(	NXP i.MX RT1020
Real time performance(PWM,ADC,Encoder,FOC support)	10	8	7	6	5
Programming ease(Free IDE,libraries,Debugging)	9	7	9	8	9
Processing power(clock speed,FPU,accelerator)	8	9	9	8	10
Programming development capability	10	8	9	9	9
Price +value	10	7	9	8	9
Comparibility for other component selections	10	8	7	6	5
Total	57	47	50	45	47

Figure 4: Evaluation of MCUs Selected

The TMS320F28069PFPS, part of Texas Instruments' C2000 Piccolo family, was chosen as the core MCU for our closed-loop stepper motor controller. This MCU is designed for real-time control applications and offers a combination of high-performance DSP capabilities, motor control-optimized peripherals, and industrial reliability.

Key Reasons for Selection

- **Real-Time DSP Capabilities:** 32-bit C28x core with floating-point support enables execution of complex control algorithms like Field-Oriented Control (FOC), Clarke/Park transforms, and PID loops in real time.
- **Motor Control Peripherals:** Includes high-resolution PWM (HRPWM), eQEP, and eCAP modules—ideal for 3-phase motor topologies and closed-loop feedback systems.
- **Integrated ADCs:** Fast 12-bit ADCs allow efficient current, voltage, and temperature sensing with minimal latency, improving closed-loop performance.

- **Communication Interfaces:** Supports I^2C, SPI, UART, SCI, and CAN—essential for integration with motor drivers, sensors, and control interfaces.
- **Safety Features:** Includes watchdog timers, trip zones, and hardware-level protections for industrial-grade fault handling and system safety.
- **Development Support:** Supported by TI's ControlSuite, MotorWare, and Code Composer Studio, with example projects and libraries for rapid development.
- **Hardware Integration:** The LQFP-100 PFPS package offers compact integration with sufficient GPIOs, memory, and analog channels for multi-axis control or system scaling.
- **Availability of IDE freely:** Code Composer Studio by TI's is freely available for coding and supported platforms(development boards/ evaluation boards) are available as well.

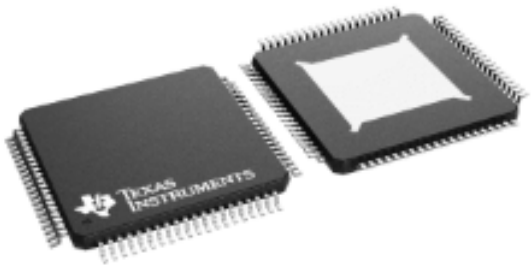


Figure 5: MCU (TMS320F28069PFPS)

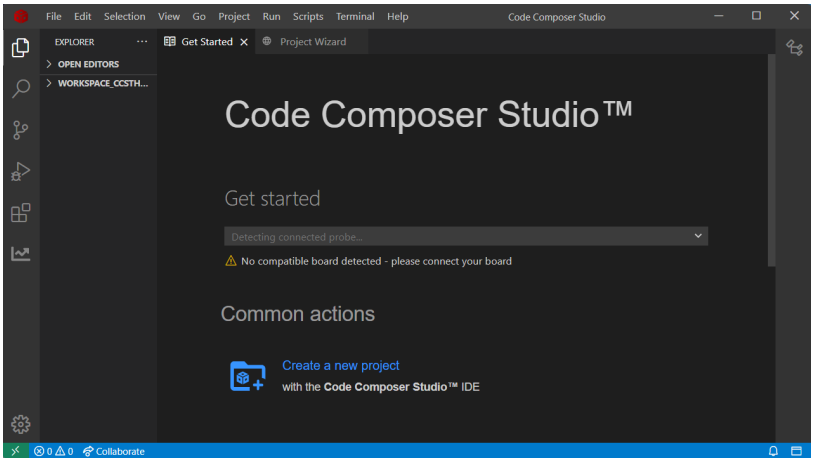


Figure 6: IDE (Code Composer Studio)

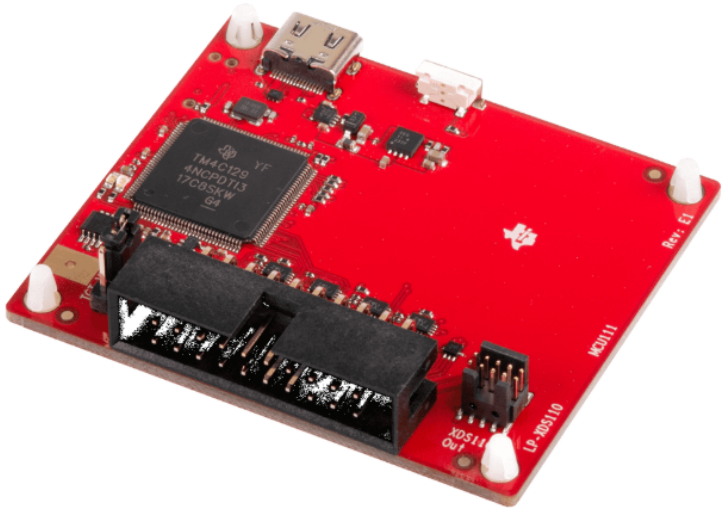


Figure 7: Development Board

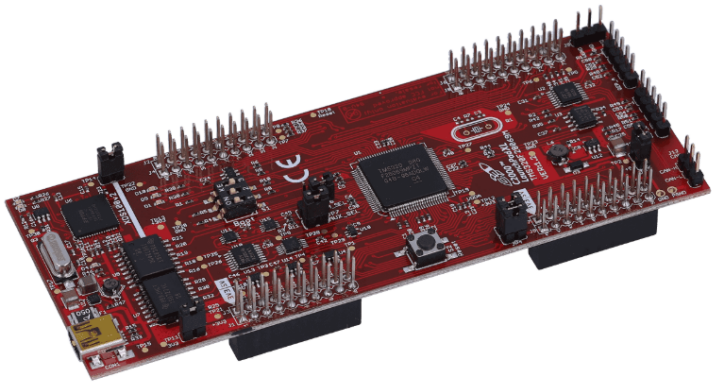


Figure 8: Evaluation board

12.3 Driver Selection

Column1	TB67H400A	DRV8462	A4989	TMC5160
PWM Compatibility	9	6	5	6
Response Time	8	9	6	8
Current regulation	7	9	5	7
FOC Compatibility	9	6	5	6
Cost Effectiveness	8	7	9	6
Total Score	41	39	30	35

Figure 9: Evaluation of Driver Selected

The TB67H400ANG from Toshiba was chosen as the primary driver IC for our stepper motor system. It is a robust, dual H-bridge motor driver suitable for medium- to high-current applications, especially in systems requiring precision and safety.

Key Reasons for Selection

- **Dual Full-Bridge Architecture:** Can drive either one bipolar stepper motor or two brushed DC motors. Simplifies wiring and board layout for dual-channel use.
- **High Voltage and Current Ratings:** Operates at up to 50 V and 4.5 A per channel. Provides enough headroom to drive NEMA 17 motors under dynamic load conditions.
- **Low On-Resistance:** Uses Toshiba's BiCD process to reduce $R_{DS(on)}$, enhancing efficiency and minimizing heat generation during high-current operation.
- **Built-in Protections:** Includes over-current, over-temperature, under-voltage, and short-circuit protection features for enhanced system reliability and safety.
- **External PWM Control Support:** Allows precise control through external DSP or MCU (e.g., TMS320F28069), enabling advanced control strategies such as Field-Oriented Control (FOC).
- **Compact Thermally Efficient Package:** Packaged in HTSSOP48 with exposed thermal pad for improved heat dissipation in compact PCB designs.

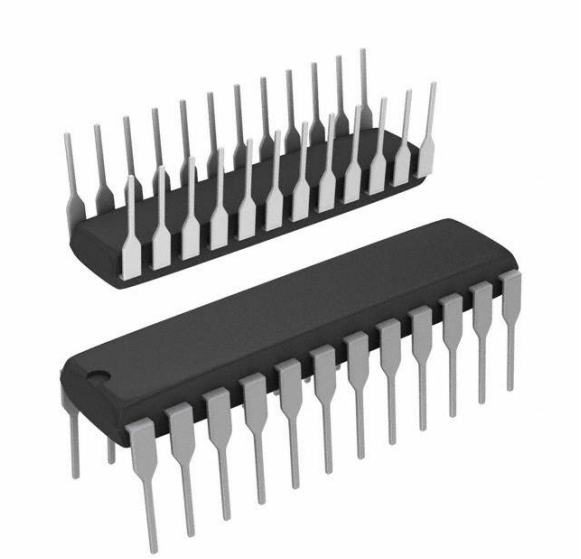


Figure 10: TB67H400ANG

12.4 ADC Selection

Column1	ADS7854IPW	MCP3208	AD7606	MAX11131
Resolution & Accuracy	9	6	8	9
Sampling Rate	9	6	8	8
Channel Count	8	8	6	6
SPI Compatibility	9	8	7	7
Cost Effectiveness	7	9	6	6
Total Score	42	37	35	36

Figure 11: Evaluation of ADCs

The ADS7854IPW from Texas Instruments was selected as the analog-to-digital converter for our closed-loop stepper motor control system. This high-speed, dual-channel ADC offers simultaneous sampling and robust performance, making it ideal for motor control applications that demand precise and synchronized current and voltage measurements.

Key Reasons for Selection

- **Simultaneous Sampling:** The two channels can sample independently but simultaneously, which is critical for phase current or voltage sensing in real-time control algorithms such as Field-Oriented Control (FOC).
- **High Throughput:** With sampling rates up to 1 MSPS per channel, the ADS7854 ensures fast signal acquisition, enabling quick updates in the control loop and improved system responsiveness.
- **SPI Communication:** The ADC communicates via a high-speed SPI interface, simplifying integration with the TMS320F28069 MCU and enabling efficient data transfer with minimal latency.
- **Robust Signal Handling:** Supports a 0–5 V input range and features low noise and offset characteristics, ensuring accurate readings even under noisy operating conditions.
- **Compact Integration:** Packaged in a 16-pin TSSOP, the ADS7854 offers a small footprint and requires minimal external circuitry, simplifying PCB layout and reducing system cost.
- **Application Compatibility:** Tailored for motor control, power systems, and automation equipment that require synchronized, real-time analog feedback.

Figure 12: ADS7854IPW

12.5 Current Sensor Selection

Column1	INA241B2IDDFR	ACS712	INA219	MAX9938
Accuracy & Offset	9	6	7	8
Bidirectional Sensing	9	8	9	6
High-Side Sensing	9	5	6	6
Bandwidth & Speed	8	5	6	7
Cost Effectiveness	7	9	8	7
Total Score	42	33	36	34

Figure 13: Evaluation of Current Sensor

The INA241B2IDDFR from Texas Instruments was selected as the primary current-sensing component in our closed-loop stepper motor control system. Designed for high-side, bidirectional current monitoring, this device offers high accuracy, fast response, and robust integration features tailored for motor control applications.

Key Reasons for Selection

- **High Accuracy and Zero-Drift:** Utilizes zero-drift architecture, providing a low offset voltage (15 μV) and minimal temperature drift, ensuring consistent measurements in varying operating environments.
- **Bidirectional High-Side Sensing:** Capable of detecting both directions of current flow while maintaining ground integrity, which is critical for safety and reliable measurements in power electronics systems.
- **Fast Response for Real-Time Control:** Offers up to 1 MHz bandwidth, making it suitable for high-frequency feedback applications such as Field-Oriented Control (FOC) and overcurrent protection.
- **Precision Gain Setting:** The B2 variant provides a gain of 20 V/V, optimized for measuring small differential voltages across shunt resistors used in phase current sensing.
- **Robust Protection and Filtering:** Integrated filtering features and a wide common-mode voltage range (up to 85 V) ensure stable performance even in noisy motor control environments.
- **Compact Footprint and Easy Integration:** Packaged in an 8-pin SOT-23 (DFN), it occupies minimal PCB space and interfaces directly with the ADS7854 ADC and TMS320F28069 MCU for closed-loop current feedback.

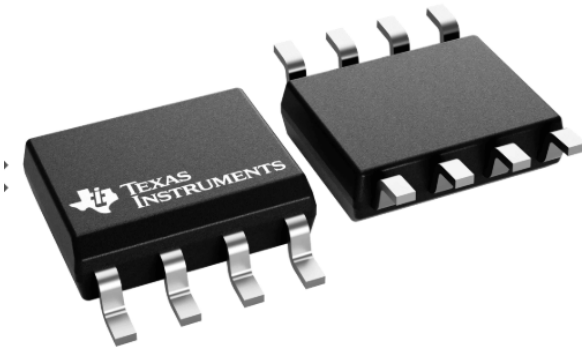


Figure 14: INA241B2IDDFR

13 Block Diagram

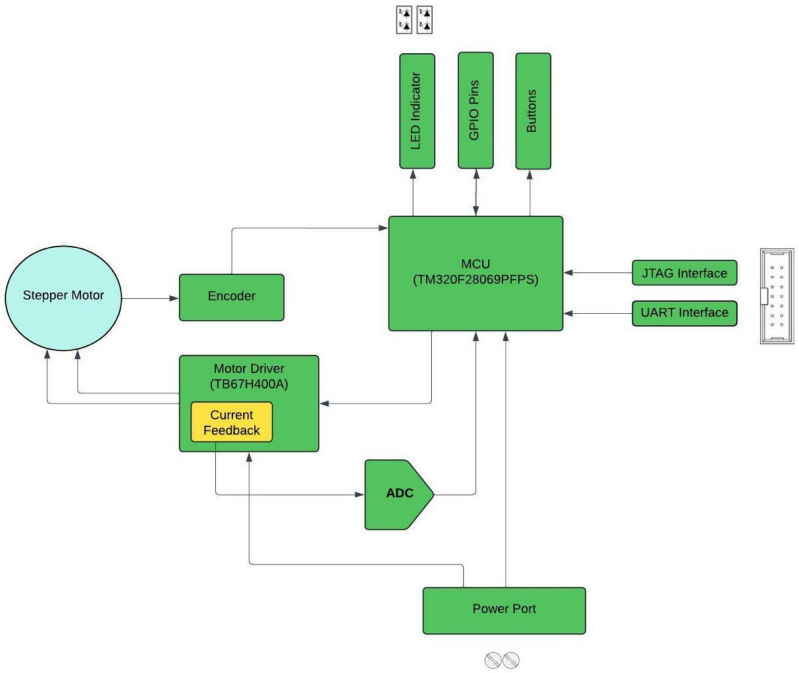


Figure 15: Block Diagram of the Closed Loop Stepper

14 Flow Chart

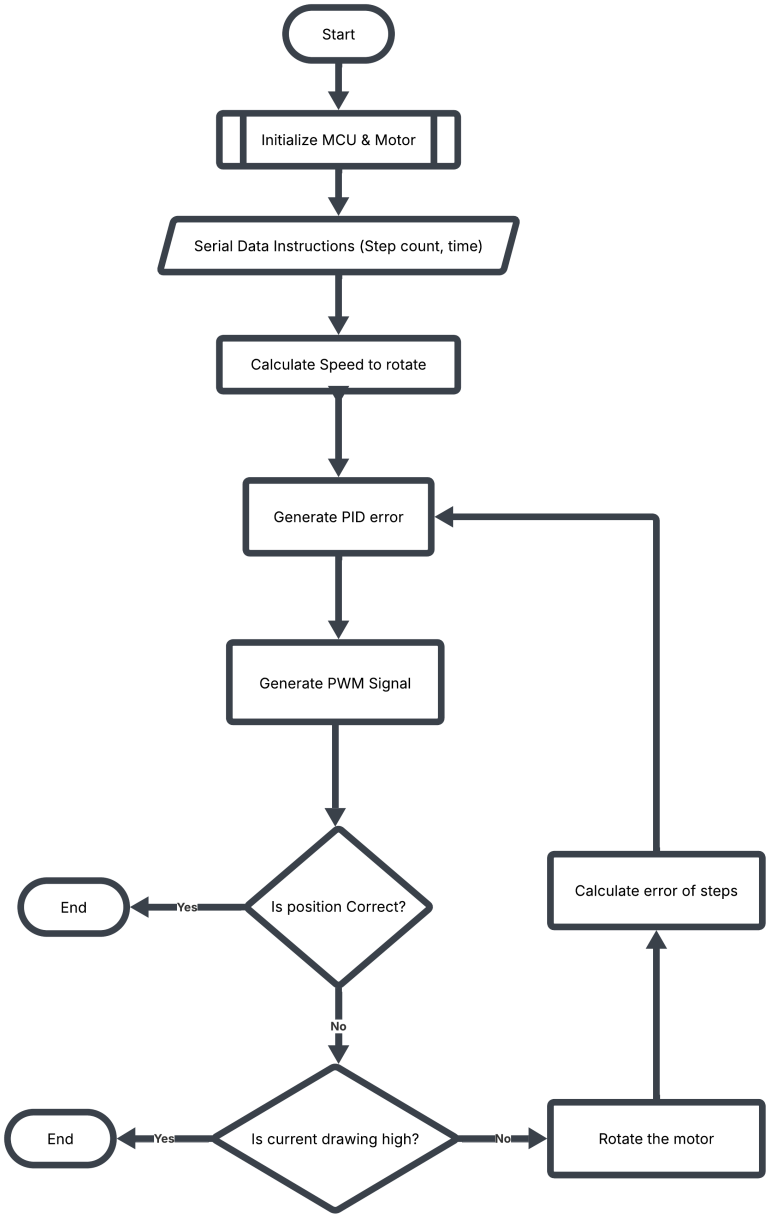


Figure 16: Flow Chart of the Closed Loop Stepper

15 SolidWorks Design

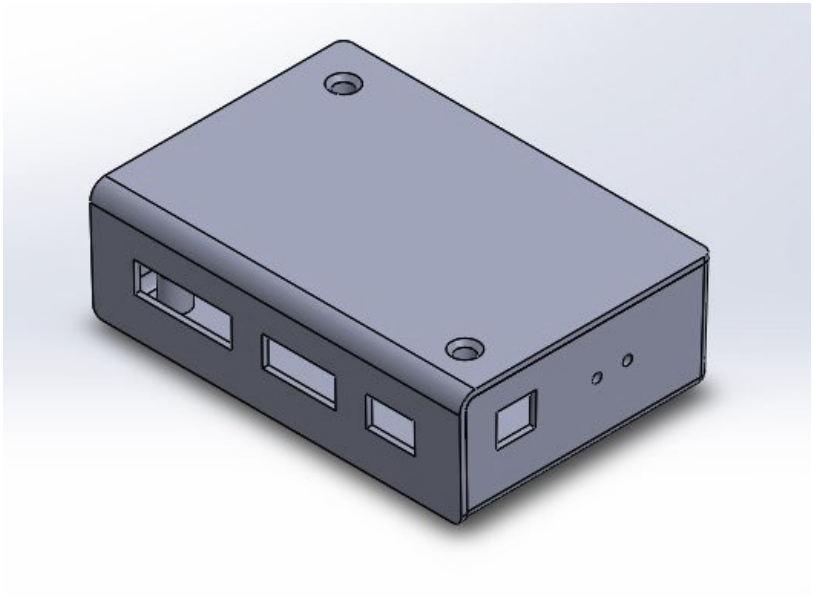


Figure 17: Conceptual Design of the Closed Loop Stepper Controller

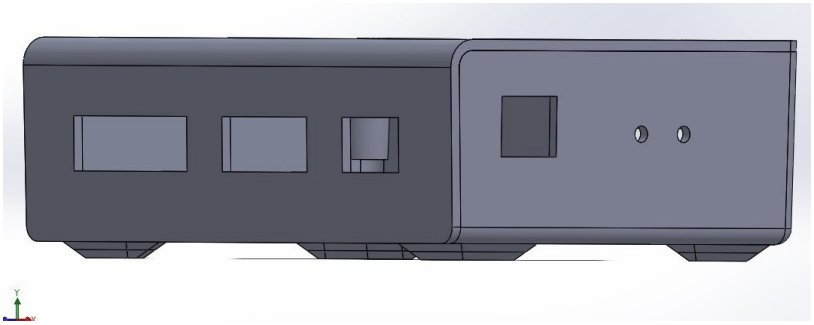


Figure 18: Conceptual Design of the Closed Loop Stepper Controller

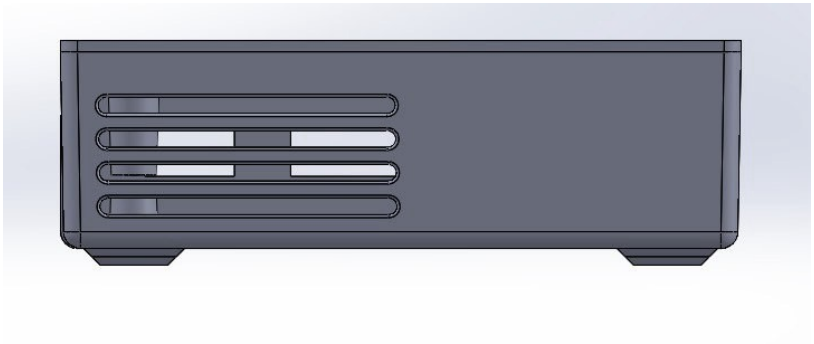


Figure 19: Conceptual Design of the Closed Loop Stepper Controller

16 Schematic Diagrams

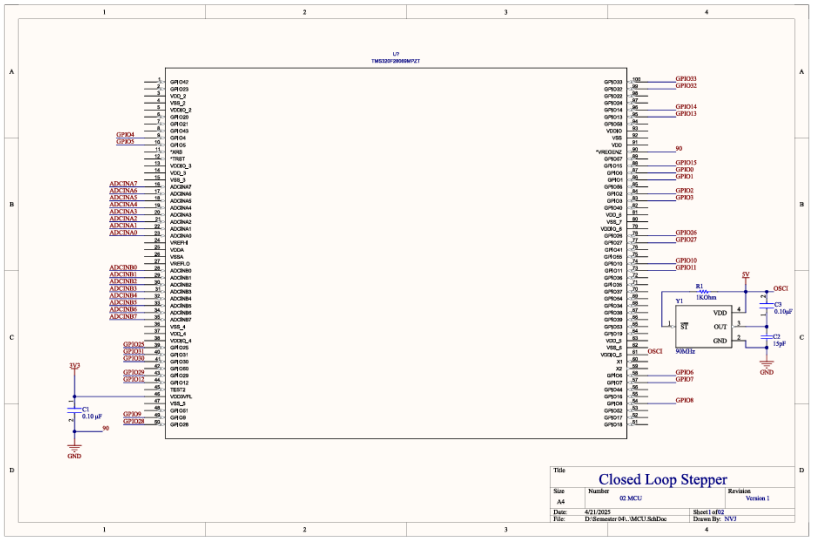


Figure 20: MCU circuit (Initial stage:Not Completed)

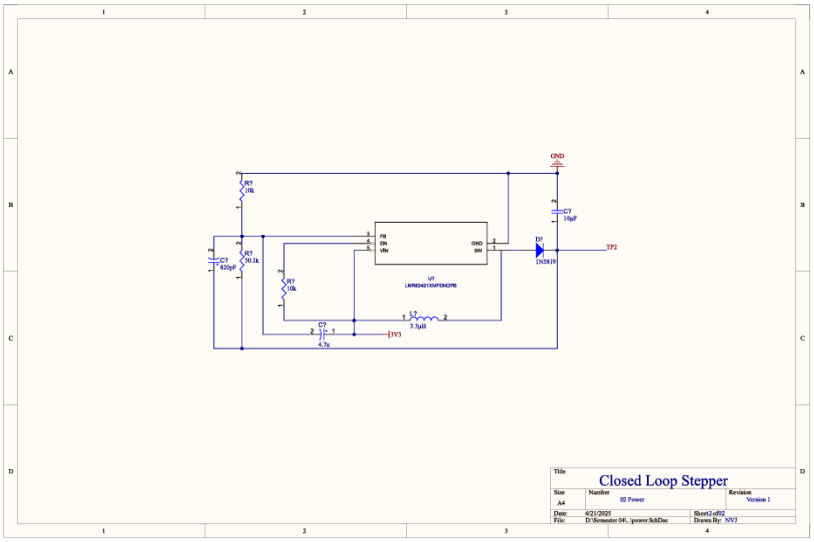


Figure 21: Power circuit (Initial stage)

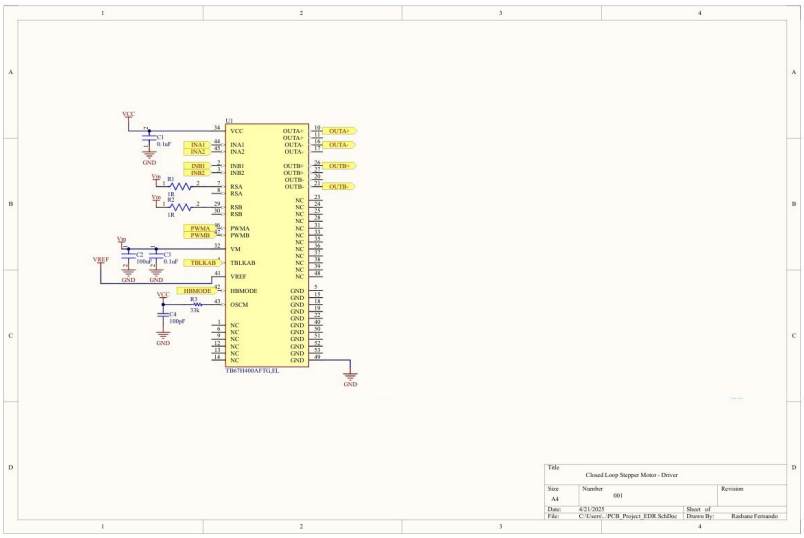


Figure 22: driver (Initial stage)

17 Estimated Budget for Components

Component	Quantity	Unit Price (USD)	Total (USD)
TMS320F28069PFPS	3	18.05	54.15
TB67H400ANG	3	6.22	18.66
ADS7854IPW	3	17.52	52.56
INA241B2IDDFR	4	2.88	11.52
LAUNCH-XLF28069M	1	46.80	46.80
LP-XDS110	1	38.56	38.56
Total			**222.25**

Table 2: Budget estimation for required components

18 Conclusion

At this stage of the closed-loop stepper motor controller project, significant progress has been achieved. The component selection phase has been successfully completed, ensuring that each element—from the high-performance **TMS320F28069** DSP microcontroller to the precise **INA241B** current sense amplifier—meets the performance, reliability, and efficiency requirements of a Field-Oriented Control (FOC) based drive system. In parallel, the mechanical structure and mounting details of the system have been designed using **SolidWorks**, allowing for proper visualization and planning of the hardware integration.

With the foundation laid, the focus of the project now shifts toward the development of detailed **electronic schematics** and **embedded software coding**. The next steps will involve:

- Creating accurate circuit schematics using design tools
- Designing and simulating the power and control PCB layout.
- Developing the FOC control algorithm code, integrating current sensing, encoder feedback, and SVPWM generation on the C2000 MCU.
- Testing and validation through simulation and real-time debugging.

Successful implementation of these stages will pave the way for a fully functional, servo-grade stepper motor driver capable of precise, quiet, and efficient operation, suited for advanced automation systems in Sri Lanka and beyond.

Implementation stage of the design - from 21st April 2025

Introduction to the Reverse Engineering Approach

Closed-loop stepper motor drivers are essential in precision motion control applications where positional accuracy, torque optimization, and reliability are critical. Unlike traditional open-loop systems, closed-loop drivers integrate real-time feedback mechanisms—typically using encoders—to dynamically correct for missed steps and optimize motor performance.

In this project, we adopt a **reverse engineering approach** by dissecting a commercially available **Leadshine closed-loop stepper motor driver**. Leadshine is a widely respected manufacturer in the motion control industry, known for robust and efficient driver designs. By analyzing this existing solution, we aim to gain a comprehensive understanding of the internal hardware architecture, signal processing methods, control strategies (such as Field-Oriented Control or PID feedback), and firmware routines.

The dissection process involves systematic examination of the driver's key components, including the power stage, microcontroller or DSP, encoder interface, current sensing circuits, and protection mechanisms. Special attention is paid to how the driver interprets feedback signals, regulates motor currents, and achieves smooth, reliable motion.

This reverse engineering effort serves two main purposes:

1. **Educational** – to deepen our understanding of industrial-grade closed-loop driver design, and
2. **Developmental** – to inform the design of our own custom closed-loop driver solution, incorporating proven techniques and identifying areas for potential improvement in efficiency, thermal performance, and adaptability.

Through this Leadshine driver dissection, we bridge the gap between theory and practice, preparing to replicate and innovate upon the core principles that make modern closed-loop stepper motor systems effective.

19 Reverse Engineering Analysis of a Closed-Loop Stepper Motor Driver(CL42T(V4.1))

Abstract This report presents a detailed reverse engineering analysis of a closed-loop stepper motor driver utilizing a dual H-bridge topology with 8 MOSFETs and an STM32 microcontroller. The study explores the hardware architecture, control principles, and implementation strategies, providing insights into the system's design for high-performance, reliable motor control applications.

19.1 Introduction

The system under analysis is a sophisticated closed-loop stepper motor driver designed to eliminate step loss through encoder feedback and advanced control algorithms. It employs a dual H-bridge configuration with 8 MOSFETs, controlled by an STM32 microcontroller, to achieve precise, reliable motion control. This report summarizes the reverse engineering process, highlighting key hardware components, control strategies, and protection mechanisms.

19.2 Hardware Architecture

19.2.1 Control Unit: STM32 Microcontroller

The central control is managed by an STM32 microcontroller, which generates PWM signals, processes encoder feedback, and executes closed-loop control algorithms. Its advanced peripherals enable precise timing and efficient processing, critical for real-time motor control.

19.2.2 Power Stage: 8 MOSFETs in Dual H-Bridge

The power stage comprises 8 MOSFETs arranged in two H-bridges, each with four devices, allowing independent control of motor phases. High-side and low-side MOSFETs are driven by dedicated driver ICs, such as IR2104, ensuring safe switching with dead-time insertion and protection features.

19.2.3 Feedback and Interface Circuits

Encoder signals are processed via differential inputs for noise immunity, providing real-time position data. Control signals for step/direction, enable, and fault monitoring are routed through dedicated connectors, facilitating integration with external controllers.

19.3 Control Principles and Implementation

19.3.1 Closed-Loop Control

The system employs encoder feedback to verify motor position, automatically adjusting PWM current control to correct errors. This approach prevents step loss and enhances accuracy, especially under load or transient conditions.

19.3.2 PWM and Microstepping

PWM techniques regulate motor current dynamically, enabling microstepping for smoother motion and higher resolution. The dual H-bridge topology supports four-quadrant operation, allowing forward and reverse current flow in each phase.

19.3.3 Protection and Fault Handling

Multiple protection layers include overcurrent, overvoltage, thermal, and position error detection. Fault conditions trigger system responses, such as disabling the driver or signaling alarms, ensuring safety and durability.

19.4 System Integration and Design Considerations

19.4.1 PCB Layout

The PCB features a clear separation between power and control subsections, with substantial copper areas for current handling and thermal management. Proper decoupling and signal routing minimize noise and improve reliability.

19.4.2 Configuration Options

Adjustable parameters include microstepping resolution, current limits, and control modes, set via DIP switches and rotary selectors. These facilitate customization for various motor types and application needs.

19.5 Conclusion

The reverse engineering analysis reveals a high-performance, feedback-based stepper driver that leverages STM32 control and a robust power stage to deliver reliable, precise motion control. Its design addresses limitations of traditional open-loop systems, making it suitable for demanding industrial and automation applications.

19.6 Mechanical specification

Figure 1: Control Unit and Microcontroller

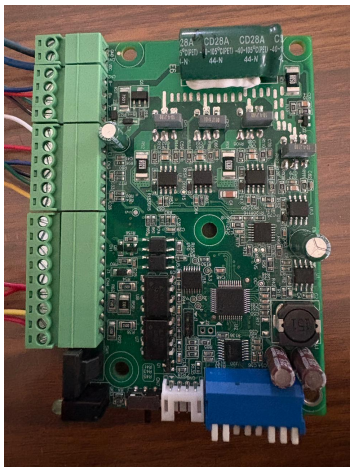


Figure 2: Dual H-Bridge MOSFET Power Stage

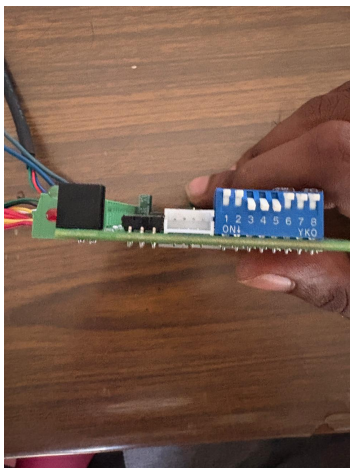
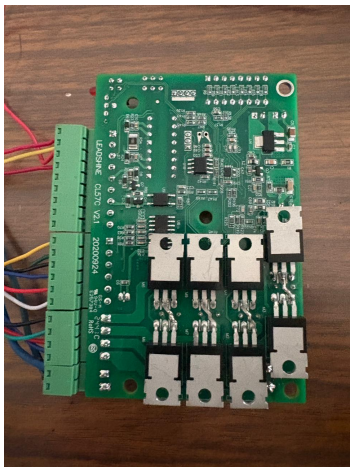


Figure 3: Feedback and Encoder Interface

Figure 4: PCB Layout and Power Components



20 Advancements in design

21 Schematic design

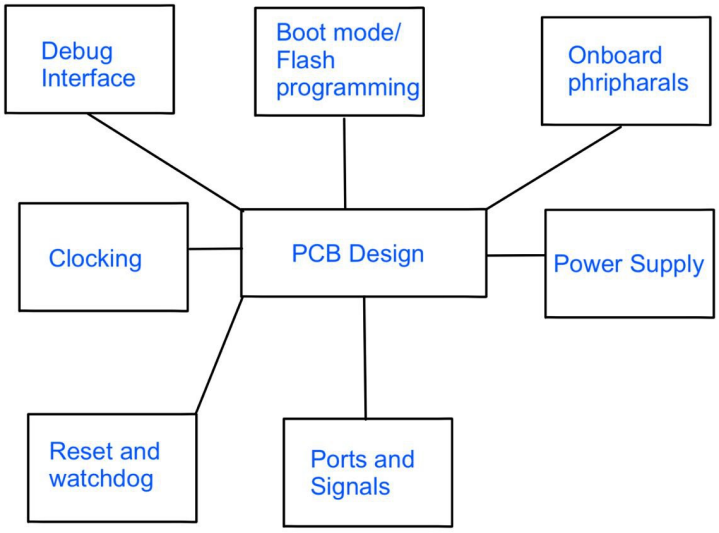


Figure 23: driver (Initial stage)

Schematic Design Using Hierarchical Design Methodology

21.0.1 Overview

To manage complexity and improve modularity in the development of our closed-loop stepper motor driver, we adopted a **hierarchical schematic design approach**. This methodology allows the system to be divided into logical functional blocks, each encapsulating a specific subsystem such as power delivery, motor control logic, feedback sensing, and protection. By organizing the design hierarchically, we facilitate easier debugging, reuse, and scalability of the overall system.

Top-Level Architecture

The top-level schematic integrates the following major subsystems:

- **Power Supply and Regulation Block**: Provides regulated DC voltages to all subsystems including logic, driver ICs, and sensing circuits.
- **Microcontroller Unit (MCU) and Motor driver Block**: Hosts the main control logic and executes the closed-loop algorithm (e.g., PID or FOC). We use a TI C2000 MCU due to its real-time control capabilities.Includes the Toshiba TB67H400AFTGEL H-Bridge driver and associated external MOSFETs to drive the two phases of the NEMA 17 stepper motor.
- **Encoder Interface Block**: Reads encoder feedback signals (e.g., quadrature signals) and converts them into position and speed information for feedback control.
- **Current Sensing Block**: Employs low-side shunt resistors and amplifiers or integrated current-sense ICs for real-time current monitoring of each motor phase.

Hierarchical Design Details

Each of the above blocks is implemented in its own schematic sheet or module, connected via hierarchical ports and buses. The hierarchical design structure allows:

- **Reusability**: Standard blocks such as voltage regulators and current sensors can be reused in future designs.
- **Simplified Debugging**: Issues can be traced to a specific subsystem quickly.
- **Collaboration**: Teams can work on different blocks independently.

Design Tools and Conventions

The schematic design was implemented using [**Tool Name, e.g., KiCad/Altium Designer/Eagle**]. Signal naming conventions, consistent net labeling, and hierarchical labels were used to ensure clarity across all sheets. All components were selected considering availability, thermal performance, and voltage/current ratings compatible with our stepper motor and driver requirements.

Block-Level Summary

- **MCU and Motor driver Block:** Includes TI C2000 series microcontroller, crystal oscillator, JTAG interface, and decoupling capacitors. TB67H400AFTGEL driver with external MOSFETs and bootstrap circuitry.
- **Power Block:** Step-down buck converter (e.g., LM2596) to supply 5V and 3.3V rails.
- **Encoder Block:** Quadrature decoder interface, ESD protection, and low-pass filtering.
- **Current Sense Block:** INA181-based amplifiers interfaced with ADC pins of the MCU.

21.1 Version 1 : Schematic designs

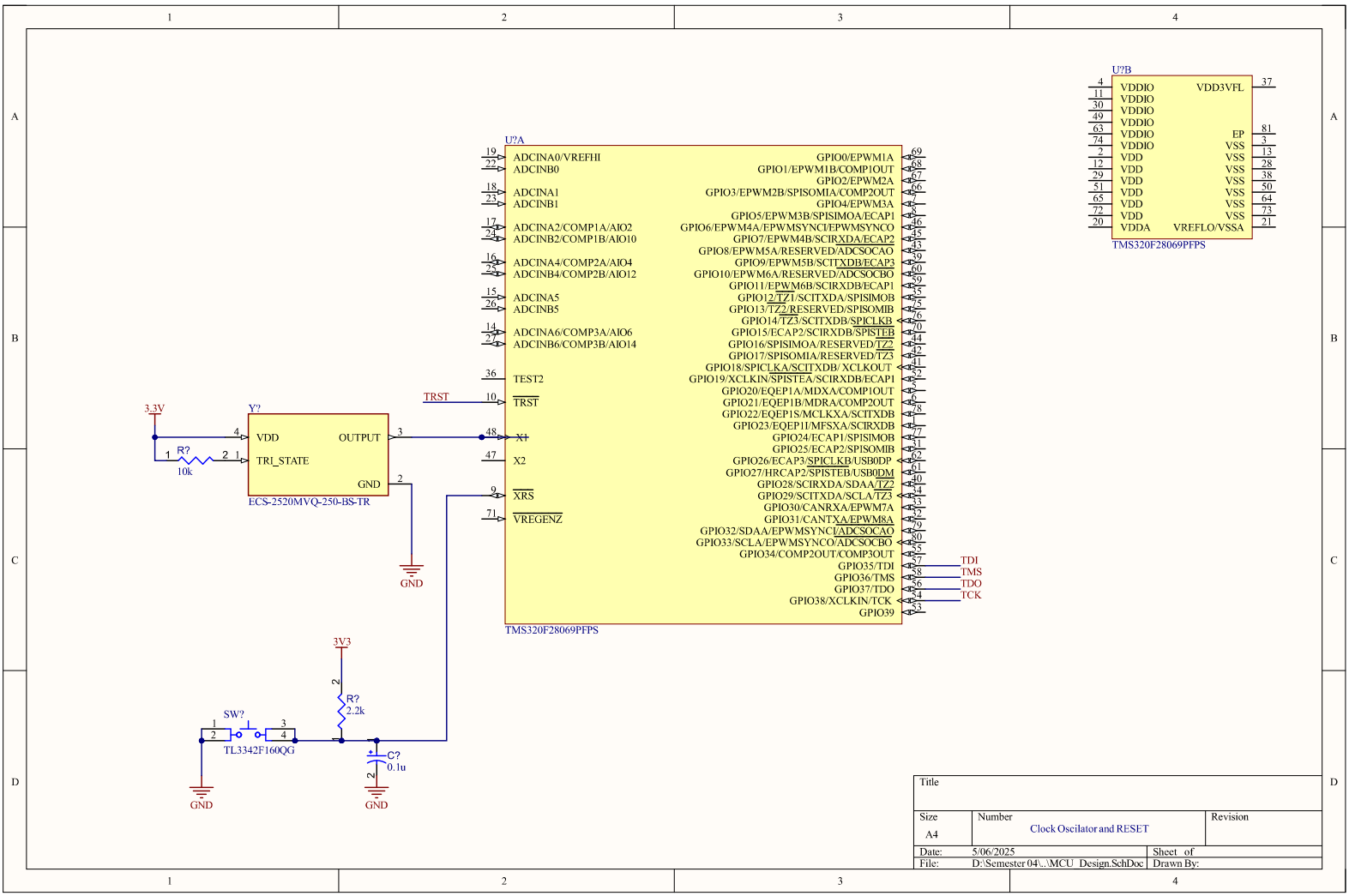
Before progressing to the full schematic design, we began with an exploratory phase aimed at understanding the functionality of key components and their interconnections. This involved carefully analyzing the Leadshine driver's internal pinouts and signal flow by studying PCB traces, datasheets, and known reference designs. The objective was to identify how critical elements—such as the motor driver IC, microcontroller, encoder interface, and current sensing components—interact within the system.

During this phase, we designed and simulated smaller, individual sections of the overall circuit. These included:

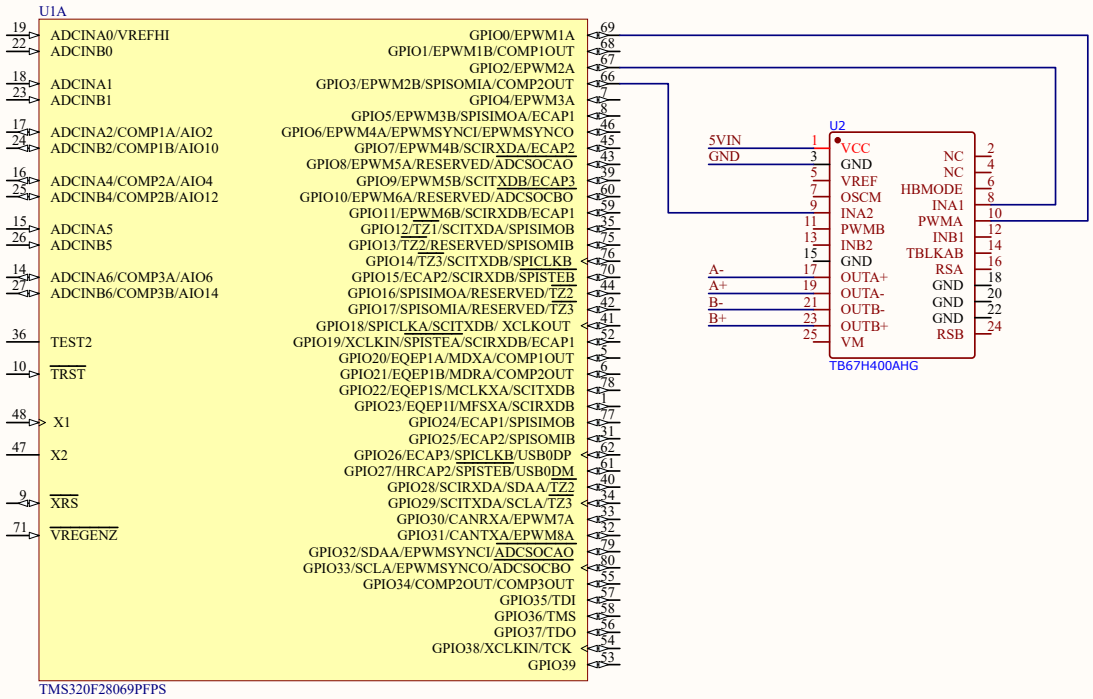
- A basic H-bridge driving stage using the Toshiba TB67H400AFTGEL driver,
- Encoder signal decoding using quadrature inputs,
- Current sensing using precision amplifiers and shunt resistors,
- Power regulation circuits for 5V and 3.3V rails.

This iterative, modular approach allowed us to validate each subsystem independently and ensured a solid understanding of component behavior before integration. It also helped us refine our component choices, confirm pin configurations, and identify required supporting circuitry (e.g., pull-up resistors, decoupling capacitors, protection diodes).

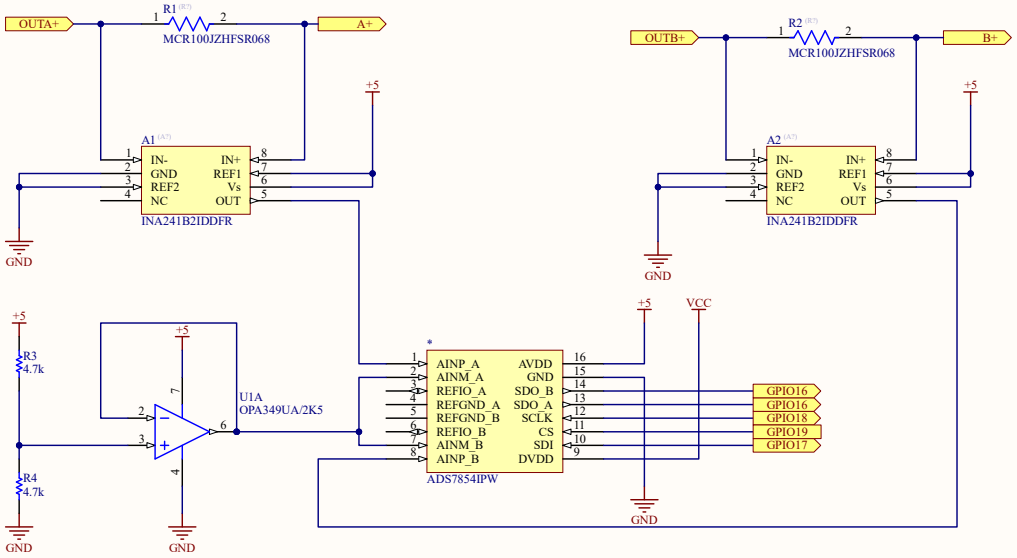
By working through these initial circuit sketches, we developed a clearer roadmap for how the various parts of the system would connect and interact in the final hierarchical schematic. This foundational stage played a crucial role in minimizing design errors and streamlining the development of the complete closed-loop stepper motor driver.



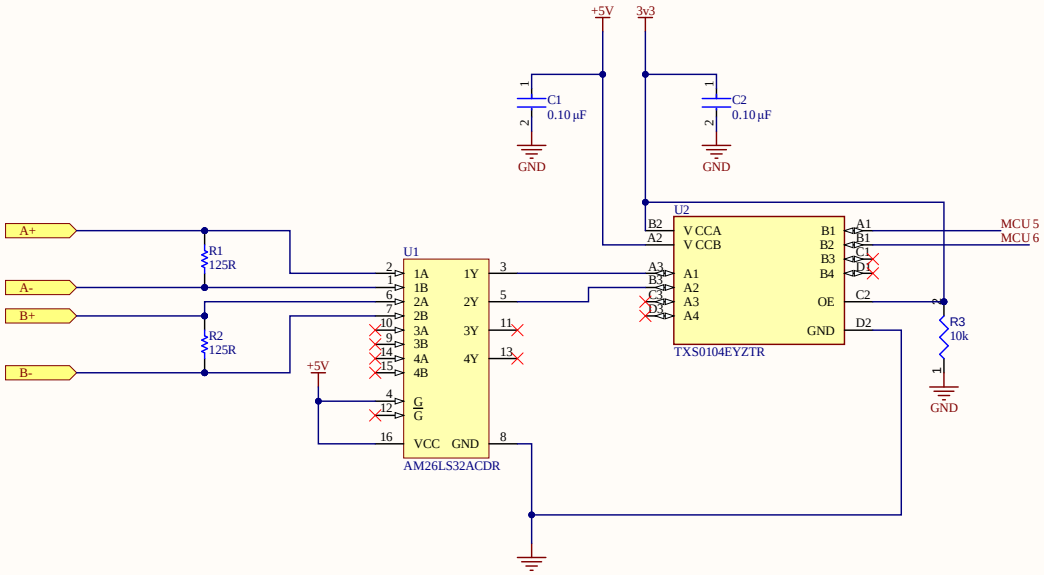
Title		
Size: A4	Number: Clock Oscilator and RESET	Revision:
Date: 5/06/2025		Sheet of
File: D:\Semester 04\...\MCU_Design.SchDoc		Drawn By:



Title		
Size: A	Number	Revision
Date: 5/08/2025		Sheet of
File: F:\altium\EDR_1\Sheet2.SchDoc		Drawn By:



Title		
Size: A4	Number	Revision
Date: 5/07/2025		Sheet of
File: C:\Users\..\EDR.SchDoc		Drawn By:



Title			
Size: Letter	Number		Revision
Date: 5/06/2025		Sheet of	
File: Sheet1.SchDoc		Drawn By:	

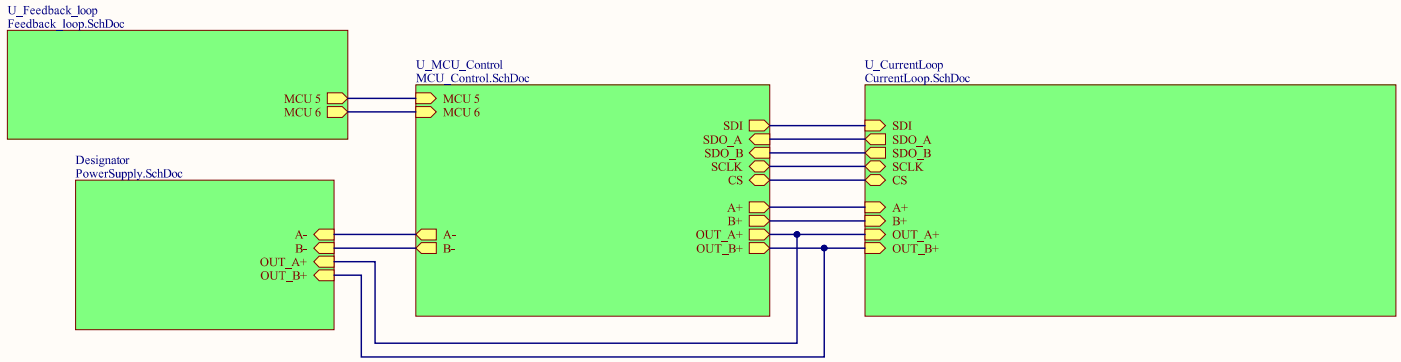
21.2 Version 2 :Structured Schematic Design Based on Hierarchical Methodology

Following the initial exploration and validation of individual circuit blocks, we proceeded to design the complete schematic using a **hierarchical design methodology**. This structured approach allowed us to break down the system into manageable and function-specific modules, promoting clarity, reusability, and ease of debugging.

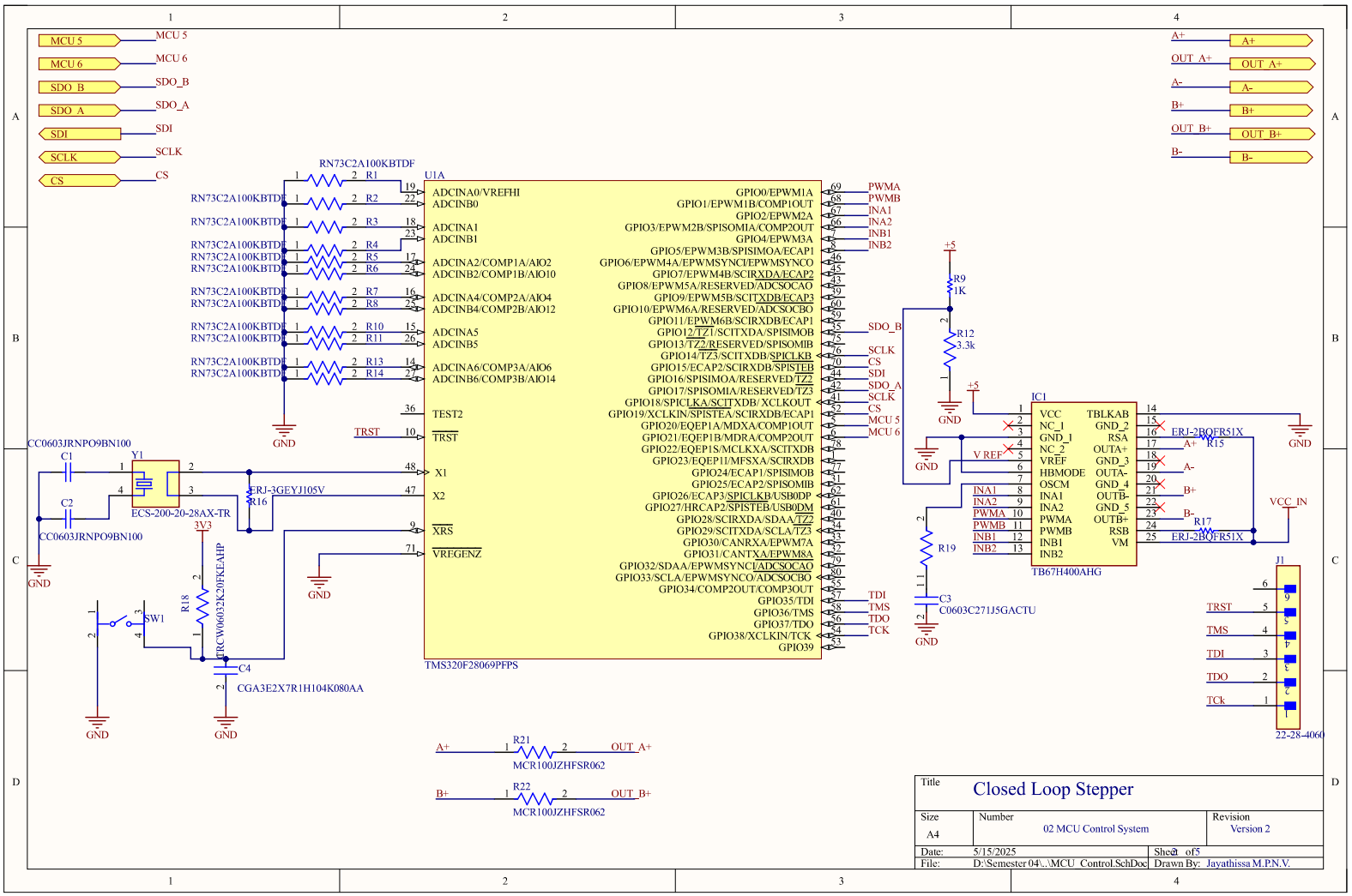
Each module—such as the power supply, motor driver stage, microcontroller unit, encoder interface, and current sensing circuitry—was designed as a separate schematic sheet, interconnected at the top level through hierarchical ports and signal buses. This modular architecture not only aligned with best practices in complex system design but also ensured scalability for future enhancements.

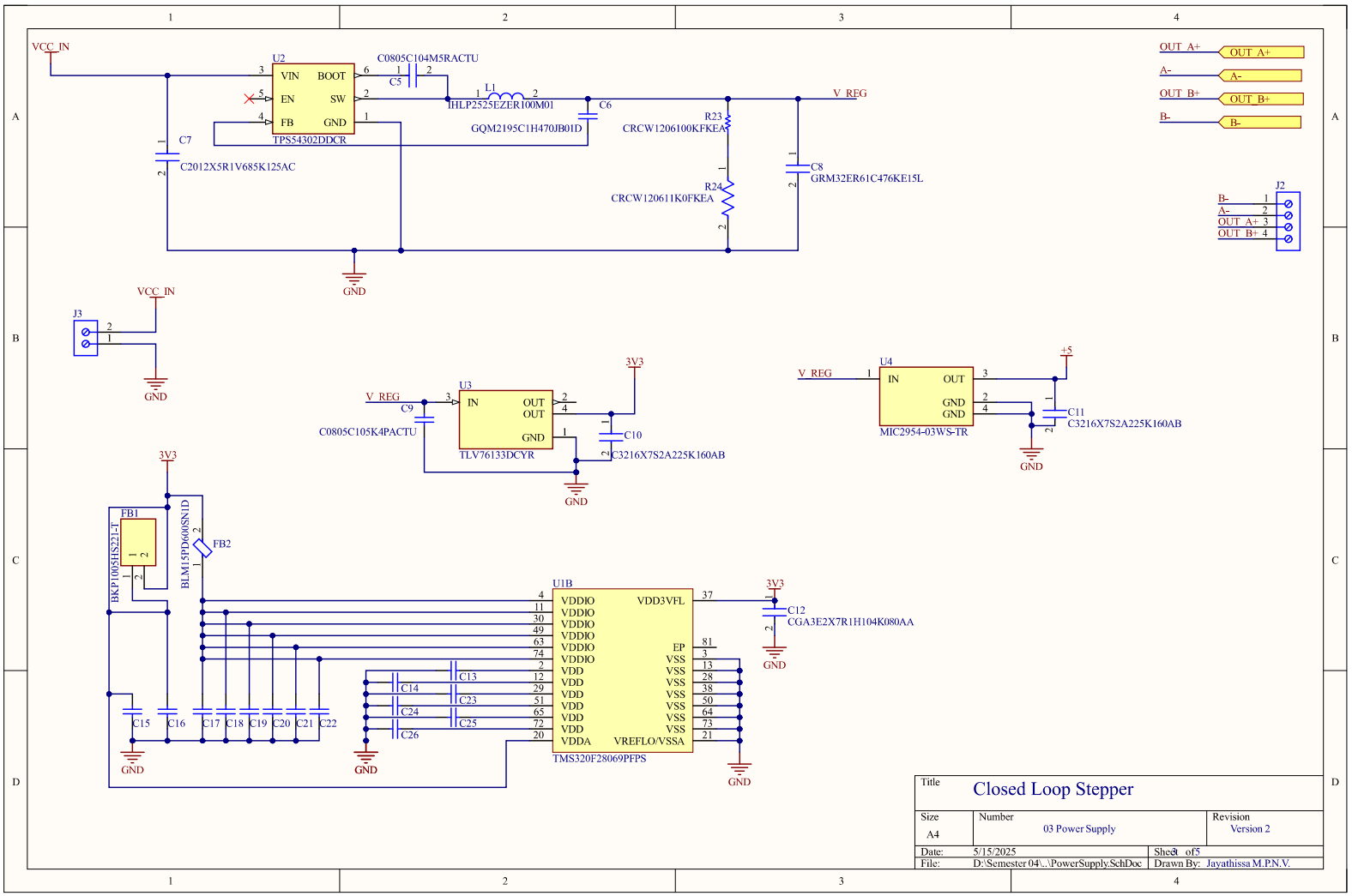
To ensure correctness and reliability, we referenced official datasheets and application notes for each key component. Additionally, we studied proven **reference designs** provided by IC manufacturers (such as Texas Instruments and Toshiba) to guide the layout, component selection, and recommended peripheral circuitry. These references were particularly helpful for configuring the TB67H400AFTGEL motor driver, selecting external components for the TI C2000 MCU, and implementing reliable encoder signal conditioning and current measurement.

This phase marked the transition from concept to implementation, where all functional blocks were accurately represented in the schematic and prepared for layout and simulation. The hierarchical schematic served as the foundation for subsequent PCB design and firmware development, with each block clearly defined and verified for integration.

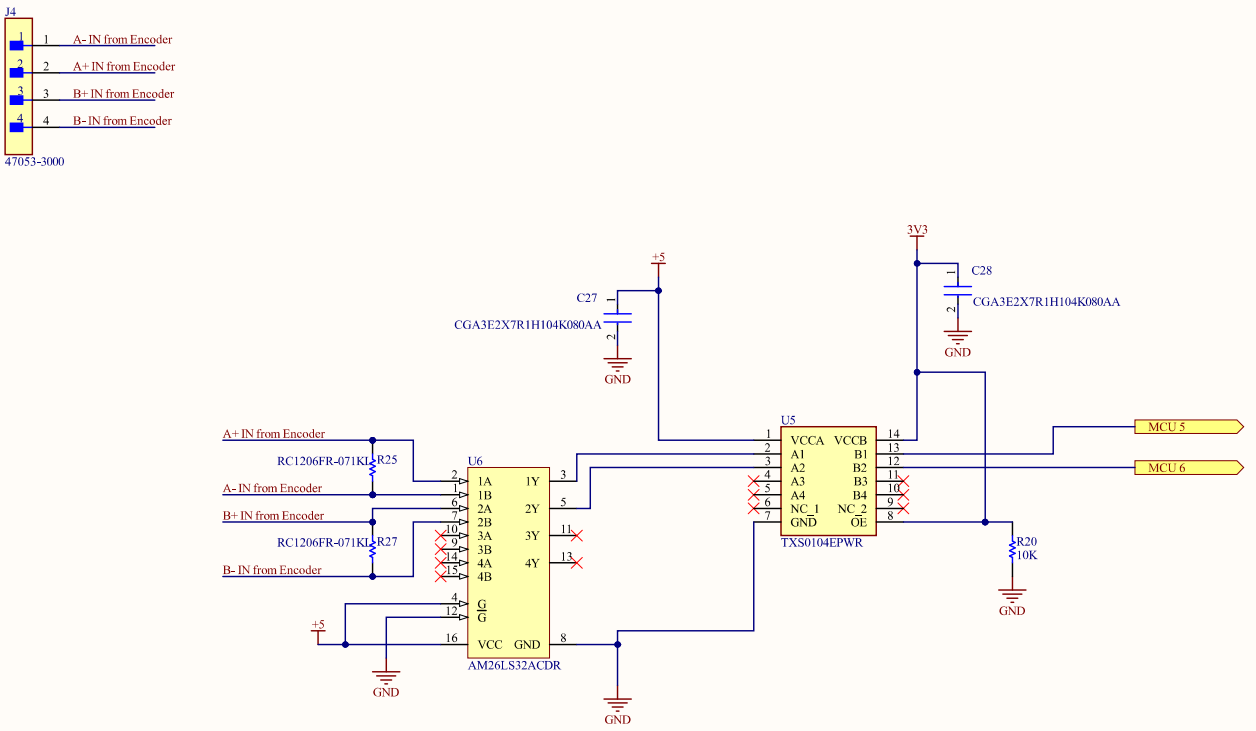


Title	Closed Loop Stepper	
Size: A4	Number: 01 Top Level Design	Revision: Version 2
Date: 5/15/2025		Sheet 1 of 5
File: D:\Semester 04\...\top_level.SchDoc		Drawn By: Jayathissa M.P.N.V.

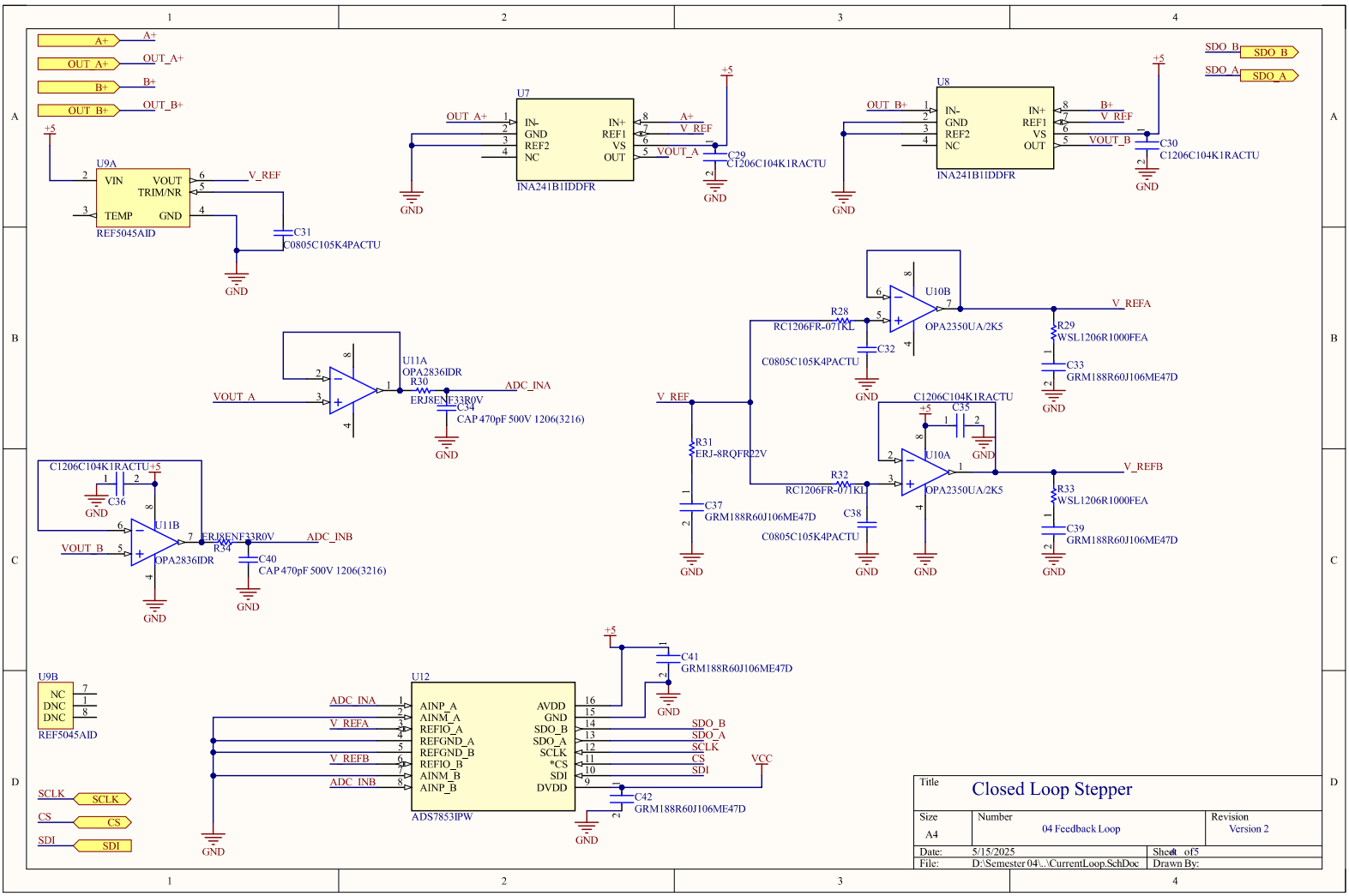

MCU 5
MCU 6
SDO_B
SDO_A
SDI
SCLK
CS
A+
OUT_A+
A-
B+
OUT_B+
B-
U1A
TMS320F28069PFPS
RN73C2A100KBTDF
TB67H400AHG
IC1
ECS-200-20-28AX-TR
CC0603JRNPO9BN100
ERJ-3GEYJ105V
ERJ-2RQFR51X
C0603C271J5GACTU
CGA3E2X7R1H104K080AA
TRCW06032K20FKEAHP
MCR100JZHFSR062
22-28-4060
3V3
GND
VCC_IN
Closed Loop Stepper
02 MCU Control System
Version 2
5/15/2025
D:\Semester 04\...\MCU_Control.SchDoc
Jayathissa M.P.N.V.



Title	Closed Loop Stepper	
Size: A4	Number: 03 Power Supply	Revision: Version 2
Date: 5/15/2025	Sheet 1 of 5	
File: D:\Semester 04\...\PowerSupply.SchDoc	Drawn By: Jayathissa M.P.N.V.	



Title	Closed Loop Stepper	
Size: A4	Number: 04 Feedback Loop	Revision: Version 2
Date: 5/15/2025	Sheet 1 of 5	
File: D:\Semester 04\...\Feedback_loop.SchDoc	Drawn By:	


A+
OUT_A+
B+
OUT_B+
U9A
REF5045AID
V_REF
C31
C0805C105K4PACTU
U7
INA241B1IDDFR
U8
C29
C1206C104K1RACTU
C30
SDO_B
SDO_A
U11A
OPA2836IDR
R30
ERJ8ENF33R0V
C34
CAP 470pF 500V 1206(3216)
ADC_INA
U11B
R34
C40
ADC_INB
U10B
OPA2350UA/2K5
R28
RC1206FR-071KL
C32
R29
WSL1206R1000FEA
C33
GRM188R60J106ME47D
C35
U10A
R31
ERJ-8RQFR22V
R32
C37
C38
R33
C39
V_REFA
V_REFB
U9B
U12
ADS7853IPW
C41
C42
SCLK
CS
SDI
VCC
Title
Closed Loop Stepper
Size
A4
Number
04 Feedback Loop
Revision
Version 2
Date:
5/15/2025
Sheet of
5
File:
D:\Semester 04\...\CurrentLoop.SchDoc
Drawn By:

21.3 Version 3 and 4 : Schematic Completion

To ensure a more efficient and error-free circuit, we undertook two additional iterations of the schematic design process. During each cycle, we carefully reviewed the existing schematic for potential issues, including logical errors, incorrect component values, and connectivity problems. Based on the observations and feedback, necessary corrections were implemented to enhance both the functionality and reliability of the circuit.

These iterative refinement cycles allowed us to improve the overall circuit design systematically. Each pass through the schematic helped us better align the design with project requirements, minimize the likelihood of hardware-level failures, and optimize component placement for clarity and ease of PCB layout in the next stages of development.

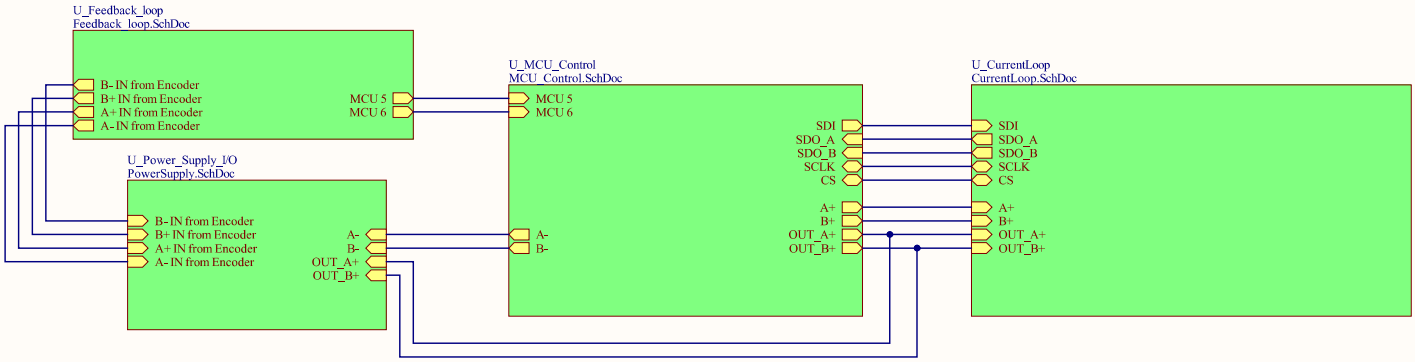
21.4 Version 5 : Final Schematic Completion and Design Considerations

With all functional blocks validated and organized hierarchically, we proceeded to complete the full schematic design of the closed-loop stepper motor driver system. During this final stage, we carefully integrated all subsystems—power regulation, control logic, motor driver stage, encoder interface, and current sensing—into a cohesive and well-structured schematic.

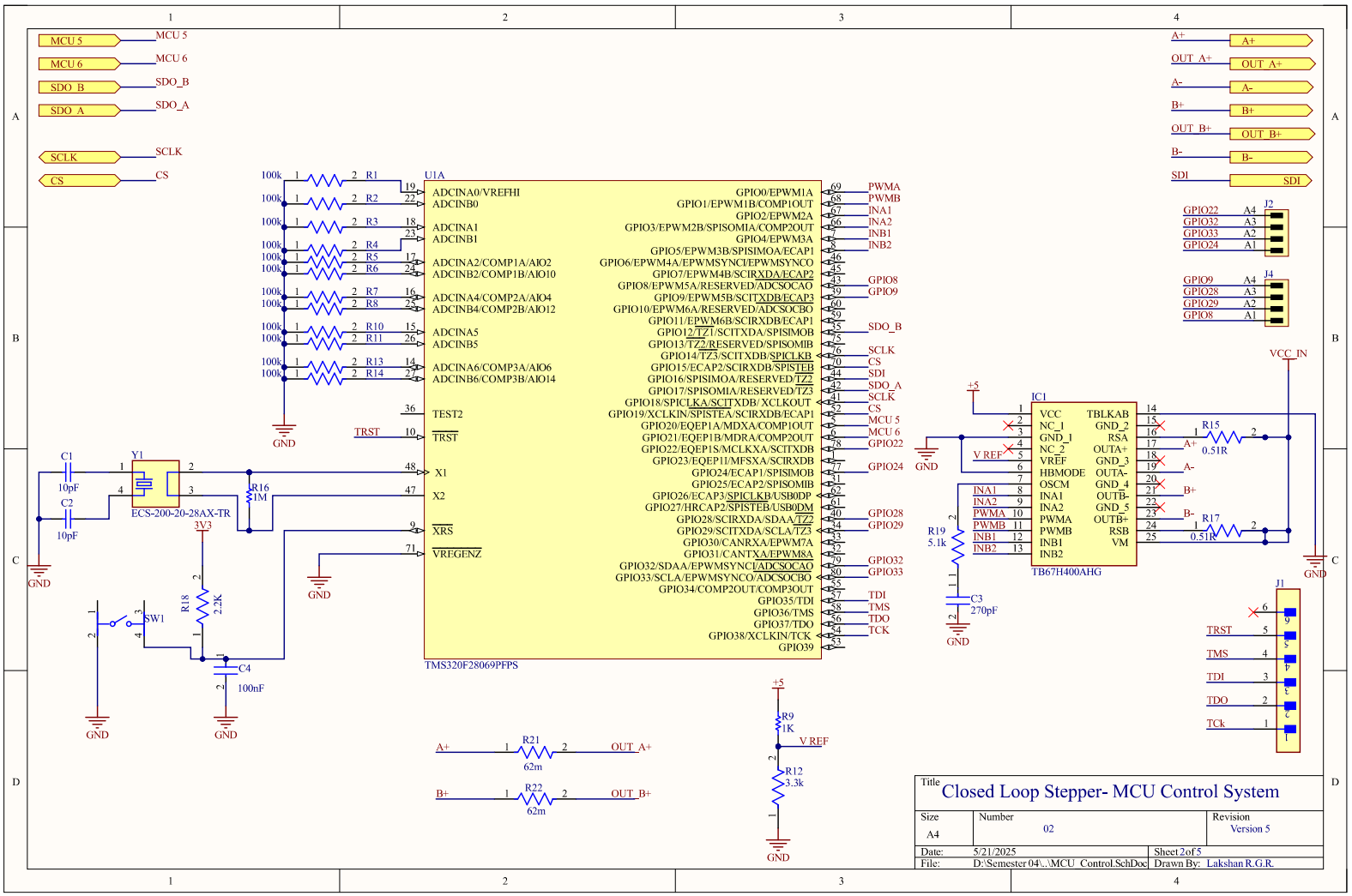
In addition to technical performance, several practical factors were taken into account to ensure the design's feasibility and manufacturability. These included:

- **Component Availability**: Preference was given to readily available and well-supported components to minimize sourcing issues and ensure supply chain stability.
- **Size Constraints**: Compact and efficient layouts were prioritized to meet physical size limitations and thermal management requirements.
- **Cost Efficiency**: Components were selected with consideration to the overall project budget without compromising performance or reliability.
- **Power and Thermal Design**: Voltage and current ratings, as well as power dissipation characteristics, were analyzed to ensure safe and efficient operation.

The result was a complete, production-ready schematic that balances performance, reliability, and practicality. This schematic forms the foundation for PCB layout and subsequent firmware development, representing the culmination of a structured, methodical design process informed by both engineering principles and real-world constraints.



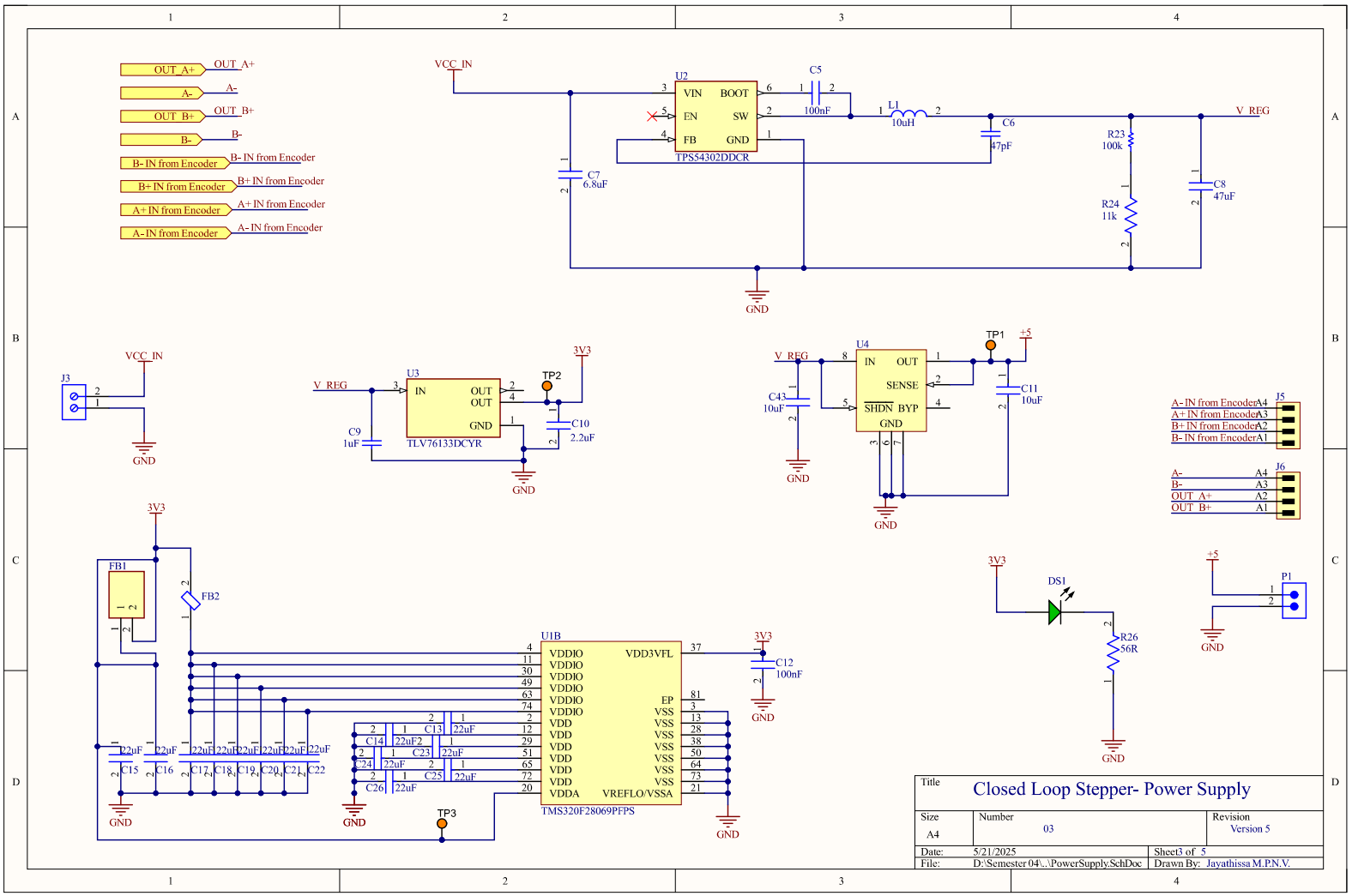
Title	Closed Loop Stepper- Top Level	
Size: A4	Number: 01	Revision: Version 5
Date: 5/21/2025		Sheet 1 of 5
File: D:\Semester 04\...\top_level.SchDoc		Drawn By: Jayathissa M.P.N.V.



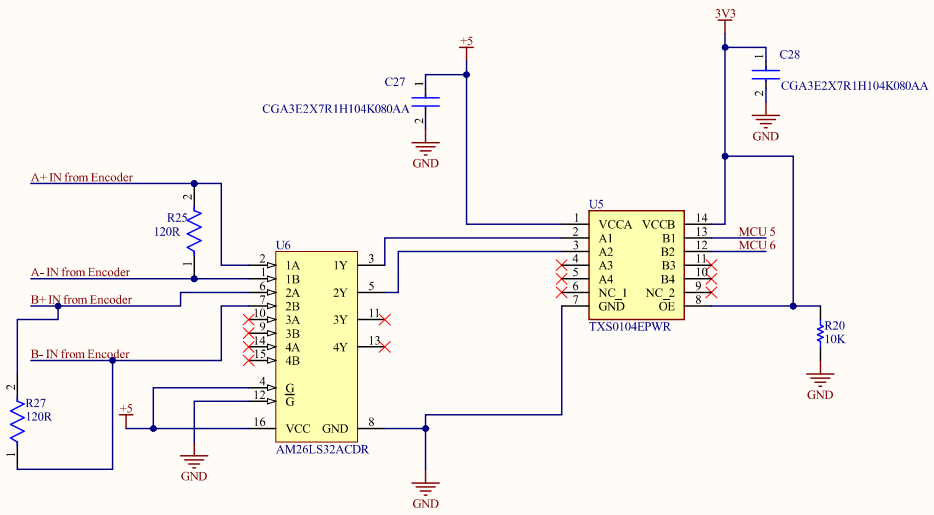
Title: Closed Loop Stepper- MCU Control System

Size	Number	Revision
A4	02	Version 5

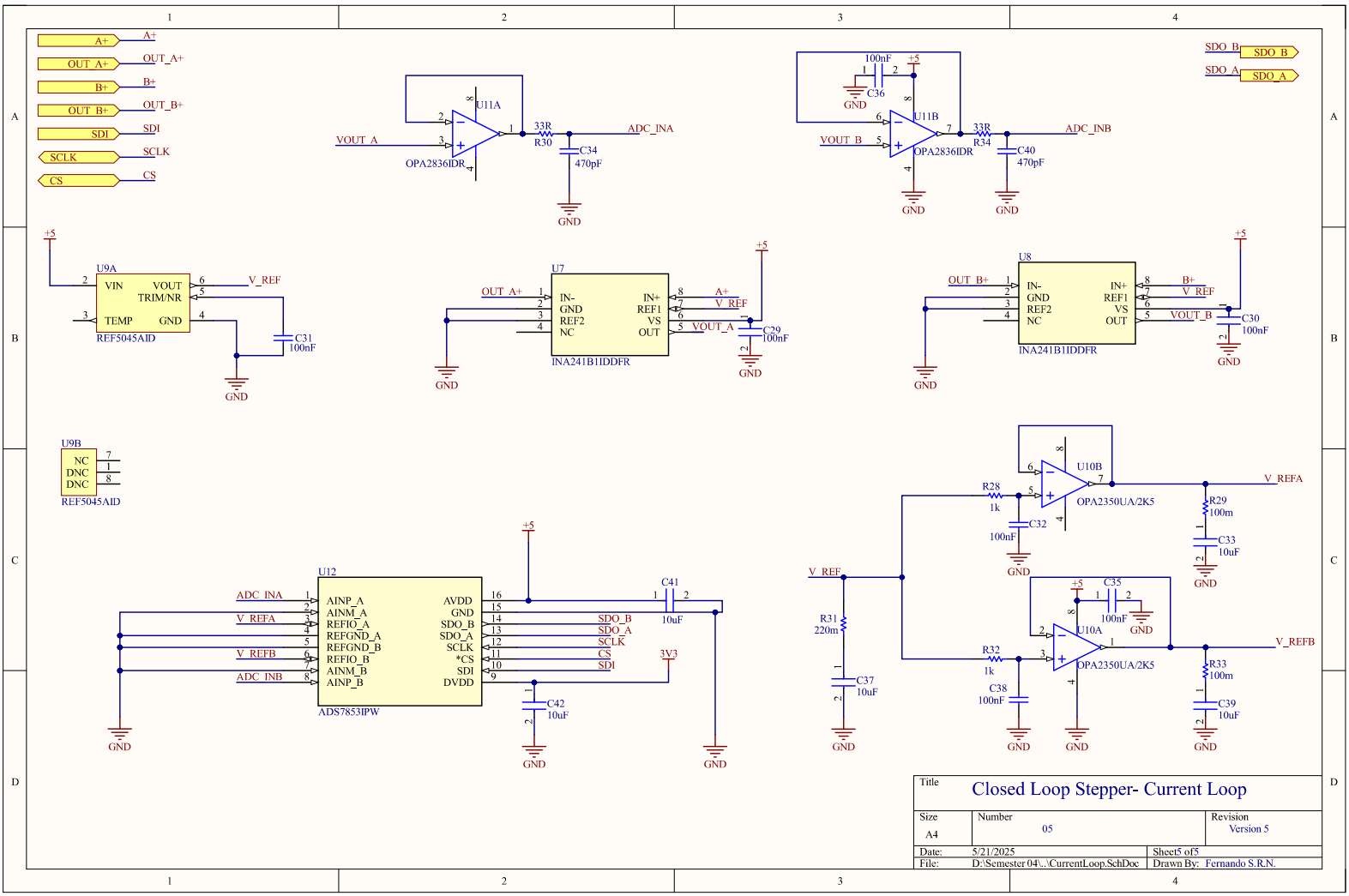
Date: 5/21/2025	Sheet 2 of 5
File: D:\Semester 04\...\MCU_Control.SchDoc	Drawn By: Lakshan R.G.R.



Title	Closed Loop Stepper- Power Supply	
Size: A4	Number: 03	Revision: Version 5
Date: 5/21/2025		Sheet3 of 5
File: D:\Semester 04\...\PowerSupply.SchDoc		Drawn By: Jayathissa M.P.N.V.



Title	Closed Loop Stepper- Feedback Loop	
Size: A4	Number: 04	Revision: Version 5
Date: 5/21/2025	Sheet 4 of 5	
File: D:\Semester 04\...\Feedback_loop.SchDoc	Drawn By: Lakshan K.P.	



Title	Closed Loop Stepper- Current Loop	
Size: A4	Number: 05	Revision: Version 5
Date: 5/21/2025		Sheet5 of5
File: D:\Semester 04\..\CurrentLoop.SchDoc		Drawn By: Fernando S.R.N.

22 PCB Design

22.1 Version 1 :PCB Design Attempt and Challenges with Two-Layer Implementation

As part of the hardware development process, we attempted to implement the closed-loop stepper motor driver on a **two-layer PCB**, with the intention of producing the board via screen printing for testing purposes. This approach was initially chosen to reduce fabrication cost and simplify the manufacturing process.

However, during the layout phase, we encountered significant challenges due to the **complexity of the circuit**. The high component density, intricate signal routing—especially for differential encoder signals, gate drive paths, and analog current sensing lines—made it difficult to maintain proper signal integrity, isolation, and power distribution on only two layers. Additionally, limitations in ground plane continuity and thermal management further complicated the design.

Despite several iterations and optimization efforts, the two-layer layout did not meet the necessary electrical and mechanical reliability standards required for high-performance motor control applications. As a result, we decided to reevaluate our PCB stack-up strategy, considering either a four-layer board or a professionally manufactured solution to accommodate the complexity of the system and ensure robust operation.

This experience underscored the importance of aligning the PCB layer count with the functional and layout demands of the design, particularly in high-current and high-speed control systems.

22.2 Version 2 :Transition to a Four-Layer PCB Design

After evaluating the limitations of our initial two-layer PCB design attempt, we transitioned to a **four-layer PCB layout** to meet the performance, reliability, and manufacturability requirements of the closed-loop stepper motor driver system.

The four-layer configuration provided the necessary routing flexibility, improved signal integrity, and enhanced thermal performance required for our complex mixed-signal design. The layer stack-up was organized as follows:

- **Top Layer**: Used primarily for component placement, high-speed signal routing (e.g., encoder lines, PWM signals), and local power traces.
- **Inner Layer 1 (Ground Plane)**: Dedicated ground plane for providing a low-impedance return path, improving EMI performance and signal integrity.
- **Inner Layer 2 (Power Plane)**: Used for distributing key power rails (e.g., 3.3V, 5V, and motor supply) to various parts of the circuit, helping to reduce voltage drop and noise.
- **Bottom Layer**: Utilized for additional routing, low-speed signal traces, and placement of passive components where needed.

This structure allowed us to effectively isolate noisy power electronics from sensitive analog and control circuitry, thereby minimizing crosstalk and ground bounce. It also enabled shorter and more direct signal paths, which is critical for high-speed control signals and ADC input accuracy.

The improved design not only met electrical performance criteria but also simplified thermal management through better copper distribution and via stitching. Careful attention was given to trace width and clearance according to IPC standards, especially in high-current paths related to motor driving and power regulation.

Ultimately, the move to a four-layer design significantly enhanced the overall robustness and functionality of the PCB, making it suitable for high-performance, closed-loop motor control applications and ready for prototyping or small-batch production.

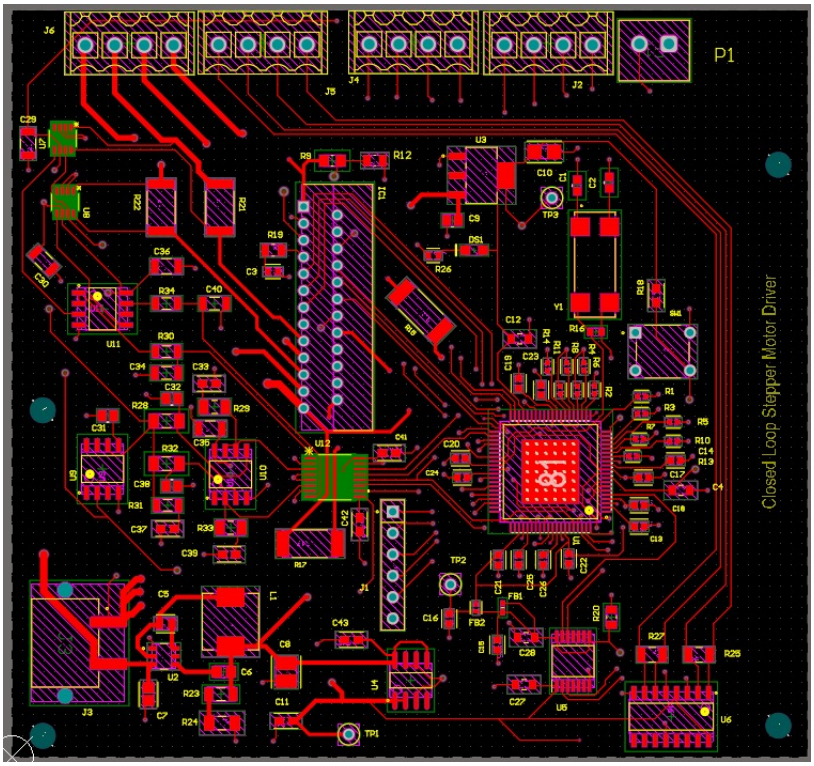


Figure 24: Top level of the PCB)

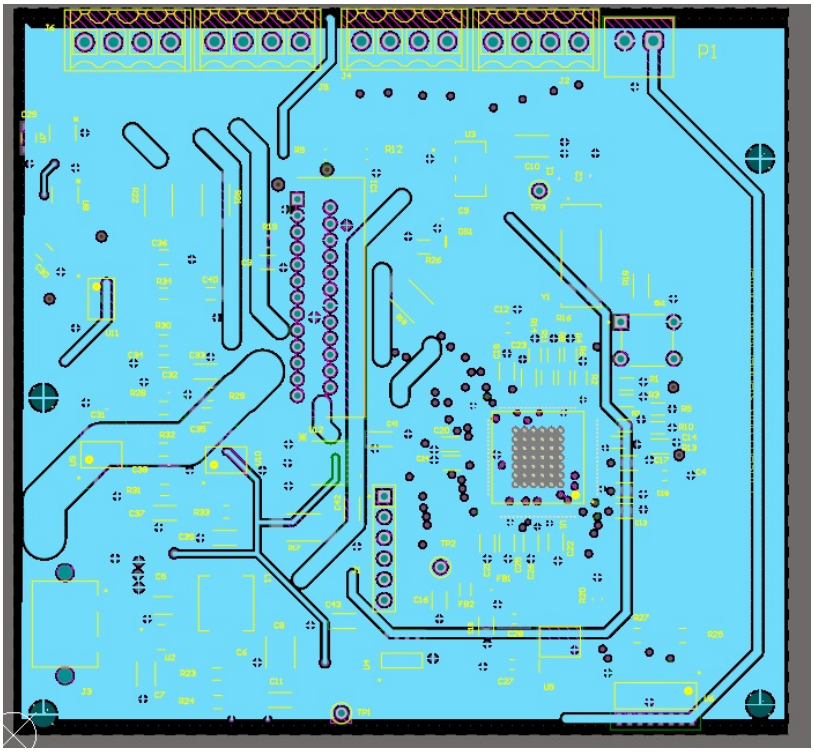


Figure 25: Second Layer of the PCB

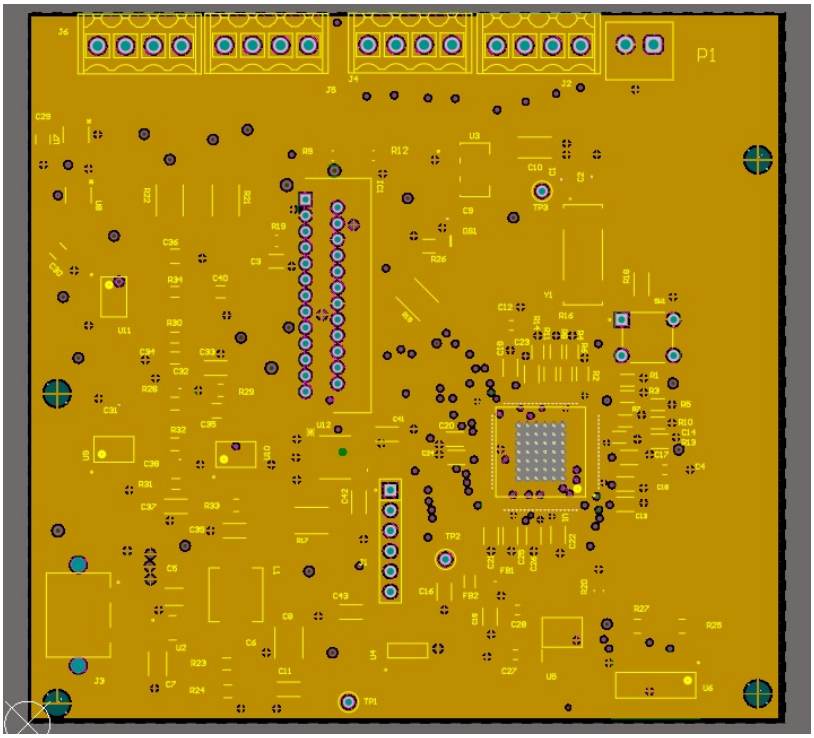


Figure 26: Third Layer of the PCB

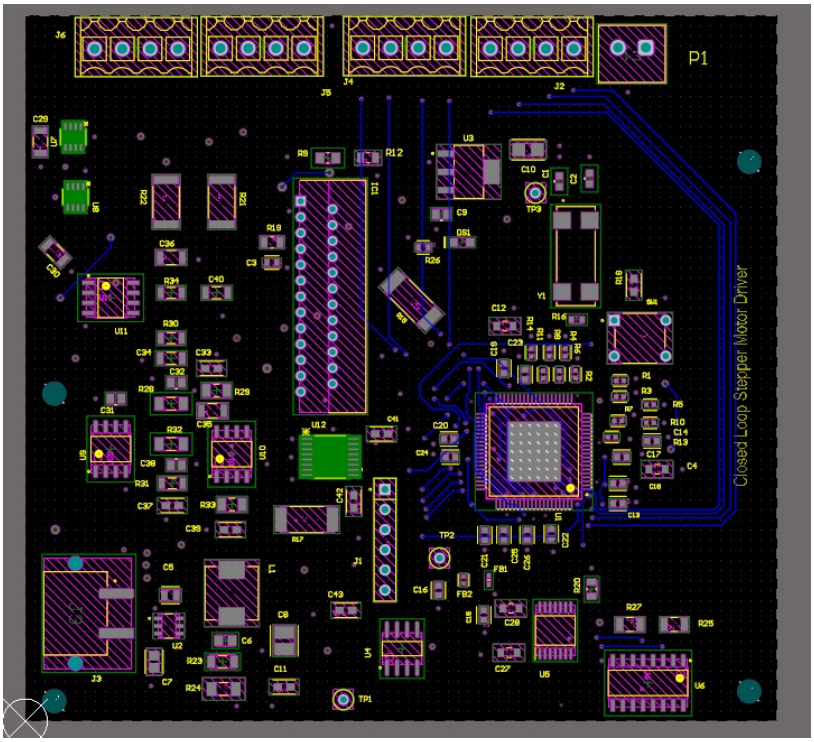


Figure 27: Bottom Layer of the PCB

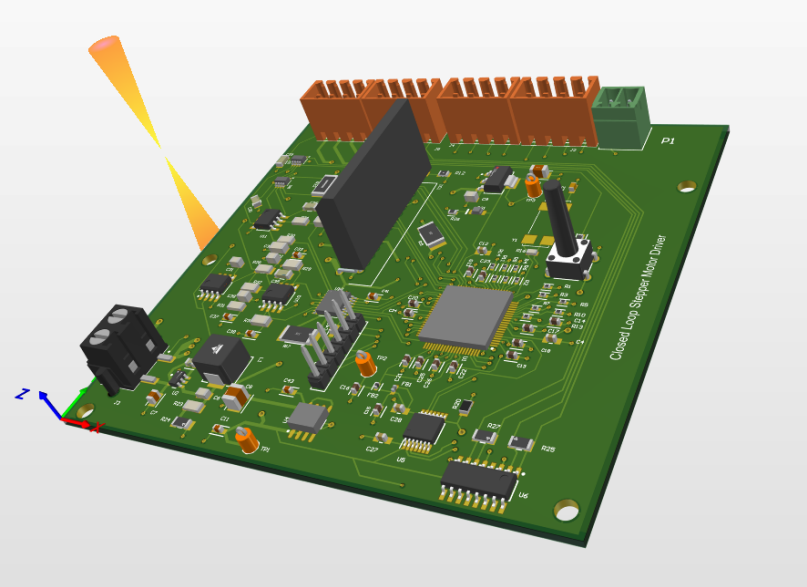


Figure 28: 3D View of the PCB

23 Enclosure Design

23.1 Enclosure Design Considerations

As part of the complete system development, we designed a custom enclosure to house the closed-loop stepper motor driver. The enclosure design process focused on balancing several key factors: **compactness**, **durability**, and the **functional requirements** of the product in its intended application environment.

Compactness was prioritized to ensure that the system could be integrated easily into a variety of industrial or prototyping setups where space is limited. This required careful internal component arrangement, efficient cable routing paths, and minimal unused volume within the enclosure.

Durability was addressed by selecting robust materials capable of withstanding mechanical stress, vibration, and moderate environmental exposure. The enclosure was designed to protect internal electronics from dust, accidental impacts, and electrostatic discharge (ESD). Mounting holes and standoff locations were included to ensure structural stability during operation.

We also took into account the **functional needs** of the product, such as:

- Proper ventilation and airflow for thermal management,
- External access to key ports and connectors (power, motor, encoder, communication interfaces),
- Mounting options for vertical or horizontal orientation,
- Maintenance access for firmware updates or part replacement.

The final enclosure design reflected a balance between aesthetic simplicity and mechanical efficiency, aligning with the real-world deployment scenarios of the driver unit. Design files were prepared using CAD software, allowing for both 3D visualization and fabrication-readiness using methods such as 3D printing or CNC machining.

23.2 Material Selection for Enclosure Fabrication

The selection of materials for the enclosure was guided by both **thermal performance** and **mechanical functionality**. To achieve an optimal balance, we adopted a hybrid construction approach that utilized different materials for different parts of the enclosure based on their specific roles.

The bottom section of the enclosure was fabricated using **aluminum**, chosen for its excellent thermal conductivity and mechanical strength. This metallic base acts as a **passive heat sink**, helping to dissipate heat generated by the power electronics—particularly the motor driver IC and voltage regulators. The flat aluminum surface also enabled direct mounting of heat-critical components using thermal pads or compound, ensuring efficient thermal transfer and enhanced reliability.

The top section of the enclosure was constructed using **PLA (Polylactic Acid)** via 3D printing. PLA was selected due to its ease of fabrication, cost-effectiveness, and suitability for low-temperature environments. The PLA top allowed for the integration of custom mounting features such as:

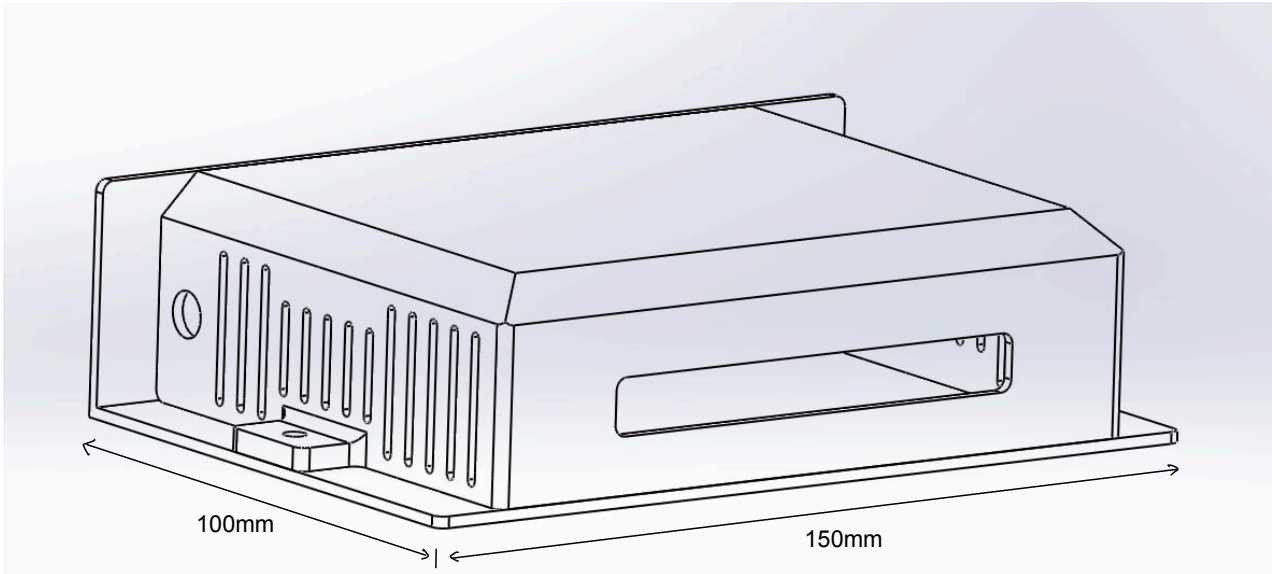
- Cutouts for connectors (power, signal),
- Screw-based mounting provisions,
- Branding or labeling as needed.

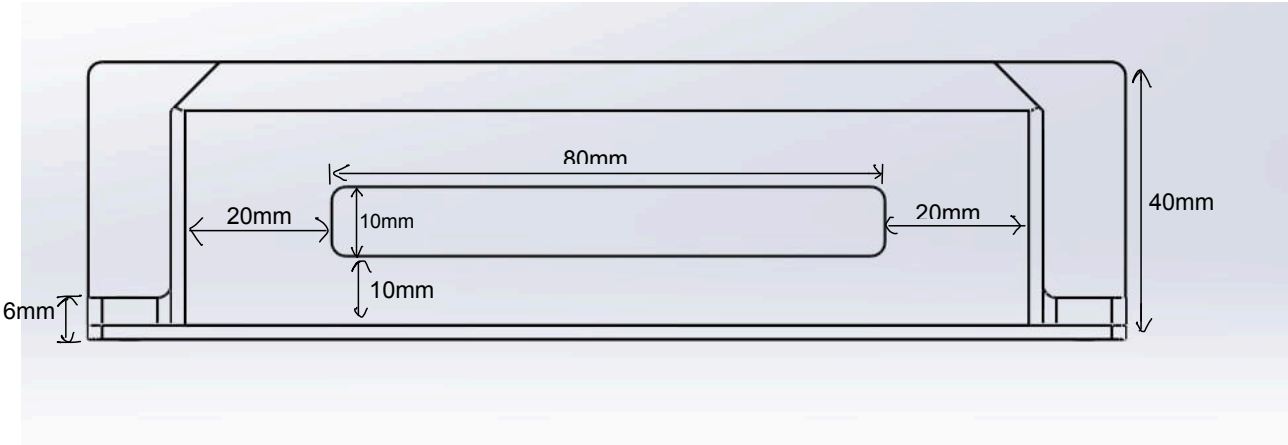
This material configuration provided a practical and efficient solution for enclosure fabrication, combining the structural and thermal benefits of metal with the flexibility and rapid prototyping advantages of 3D-printed polymer components. It also enabled faster iteration during the enclosure development phase, allowing for adjustments to port layout and component placement without the need for expensive tooling.

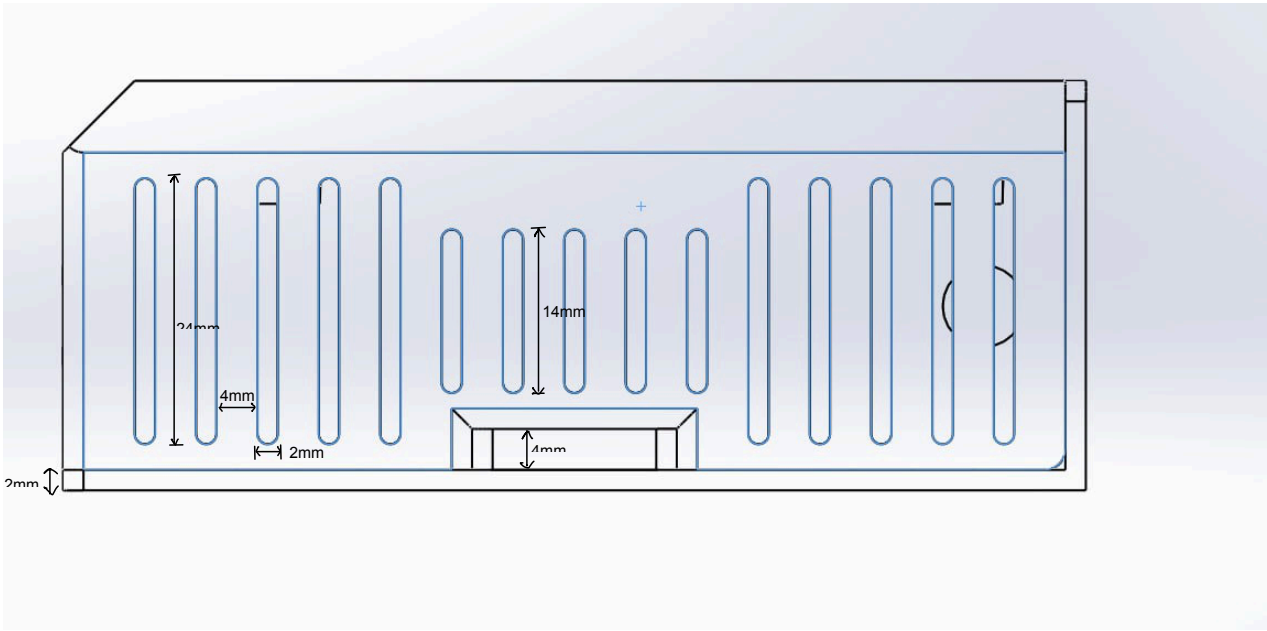
23.3 Enclosure size selection

The enclosure size was carefully determined based on the physical dimensions of all internal components, thermal management needs, and mechanical clearance for safe and efficient assembly. A compact form factor was targeted to maintain portability and integration flexibility, especially for use in embedded or space-constrained environments. Internal volume was allocated to accommodate the PCB, connectors, wiring harnesses, and passive cooling components, ensuring adequate spacing to prevent electrical interference and allow for natural convection. Additional clearance was considered for standoff mounting, future expandability, and tool access during assembly or maintenance. The final enclosure dimensions represented a trade-off between minimizing size and ensuring sufficient space for component layout, airflow, and mechanical stability, aligning well with both functional requirements and ergonomic handling.

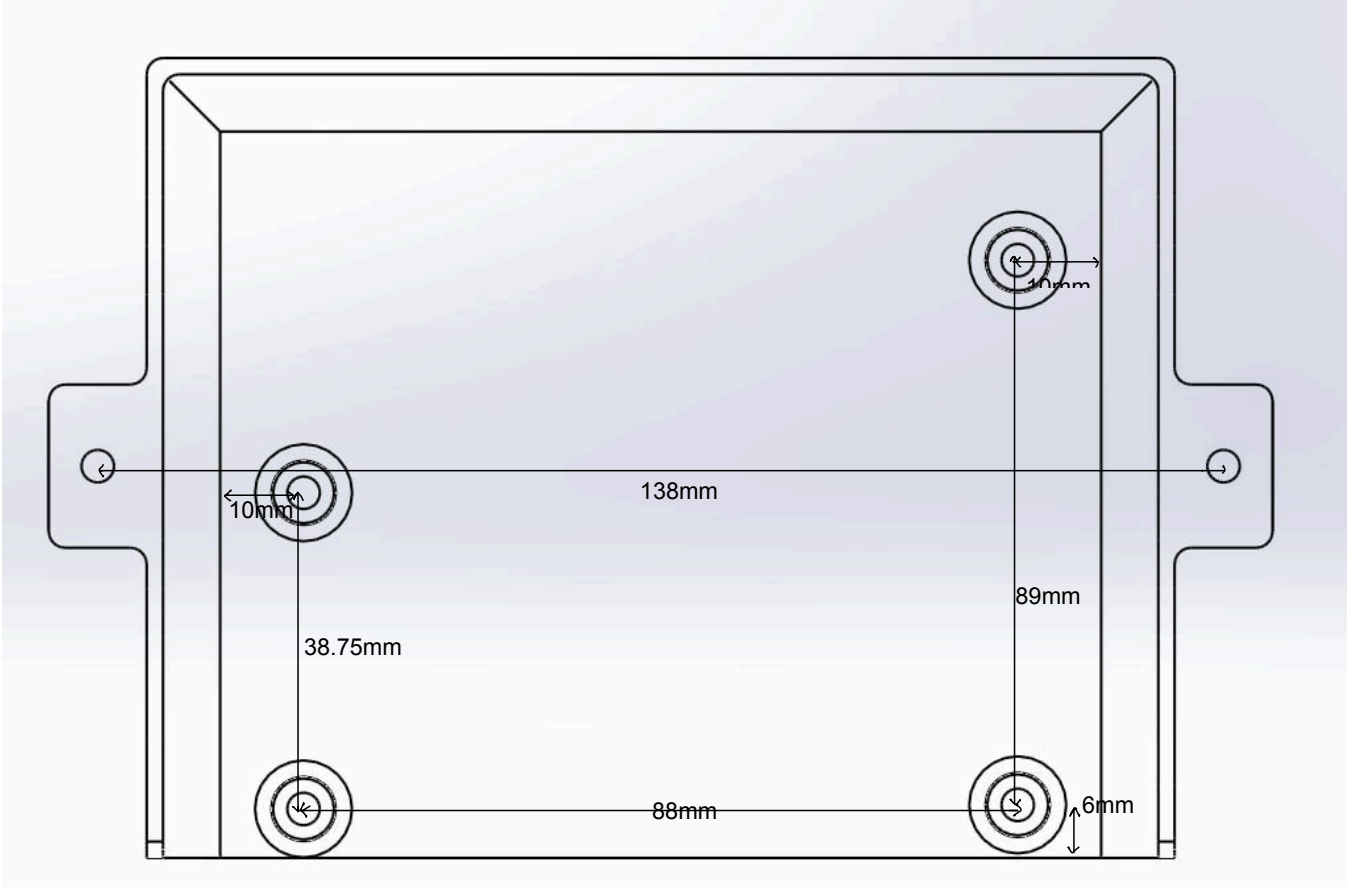
External Parameters







Internal View



23.4 Version 1: SolidWorks designs

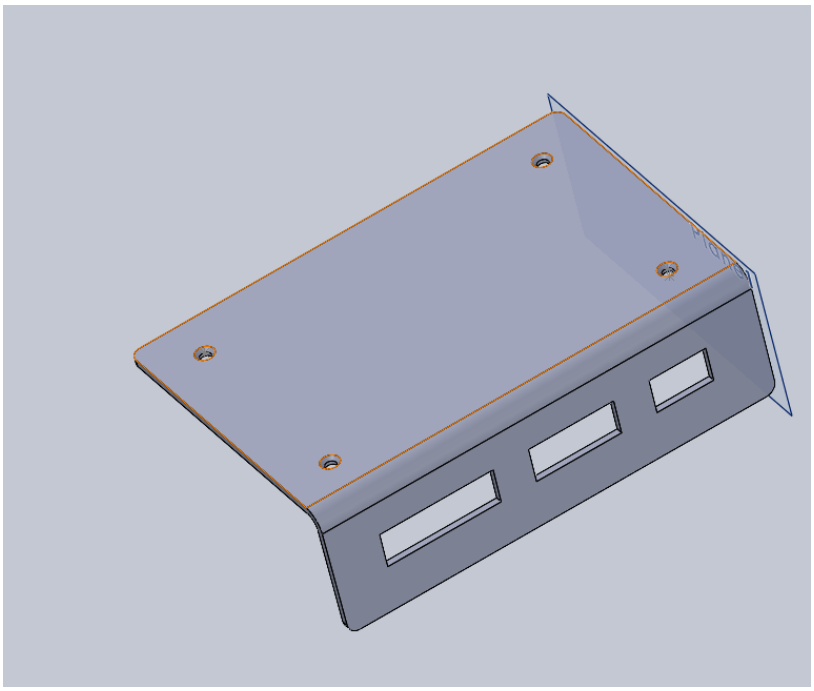


Figure 29: Top Part of the design

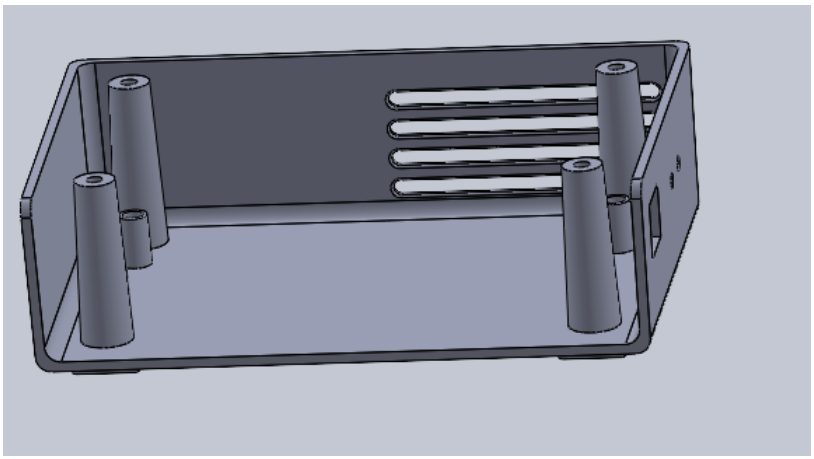


Figure 30: Bottom Part of the design

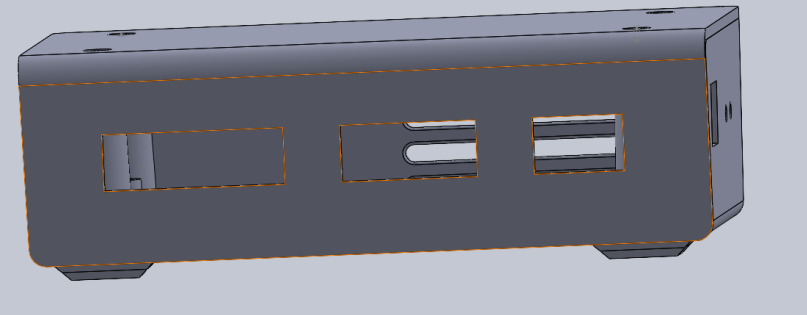


Figure 31: Assembly of the design

23.5 Enclosure Design Iterations

The enclosure design process involved **multiple iterative cycles**, during which we evaluated various design approaches to meet mechanical, thermal, and usability requirements. Each iteration focused on refining aspects such as component placement, port accessibility, airflow paths, material suitability, and overall compactness.

Initial designs were assessed for feasibility in terms of 3D printability and induction mouldability, ease of assembly, and integration with the PCB and mounting hardware. Feedback from prototyping and fit-checks guided adjustments to internal spacing, mounting hole positions, and connector alignment. Thermal simulations and real-world testing further influenced changes to vent placement and the integration of the aluminum base as a heat sink.

After addressing challenges identified in earlier versions, we finalized an enclosure design that met all key criteria—compactness, durability, ease of manufacturing, and support for proper thermal dissipation. This final design was validated through both 3D visualization and physical prototyping, ensuring readiness for deployment and use in various application scenarios.

23.6 Version 4: Final SolidWorks designs

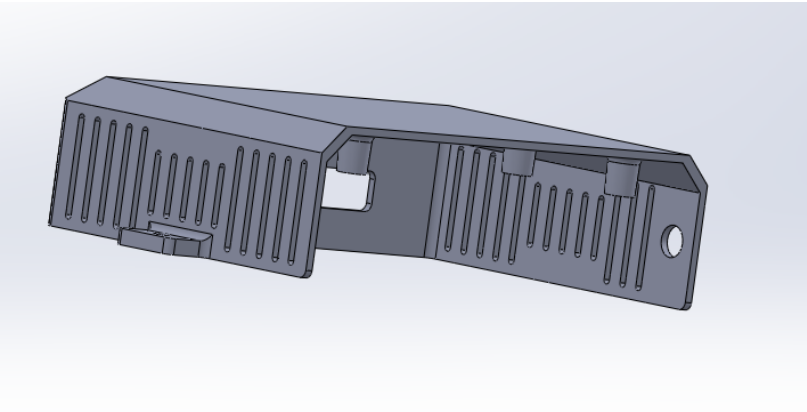


Figure 32: Top Part of the designe)

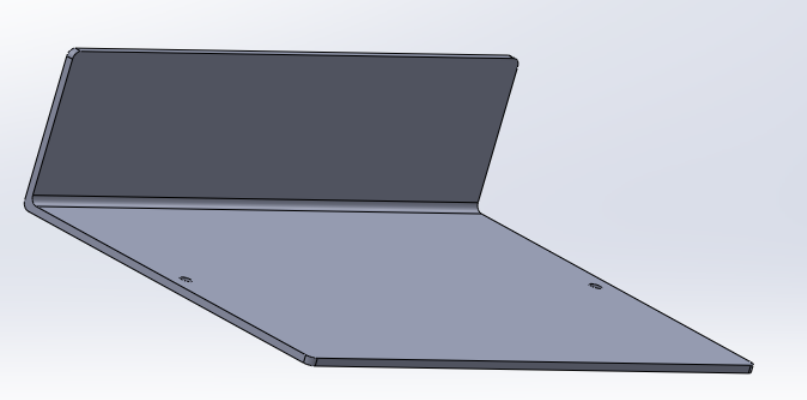


Figure 33: Bottom Part of the design

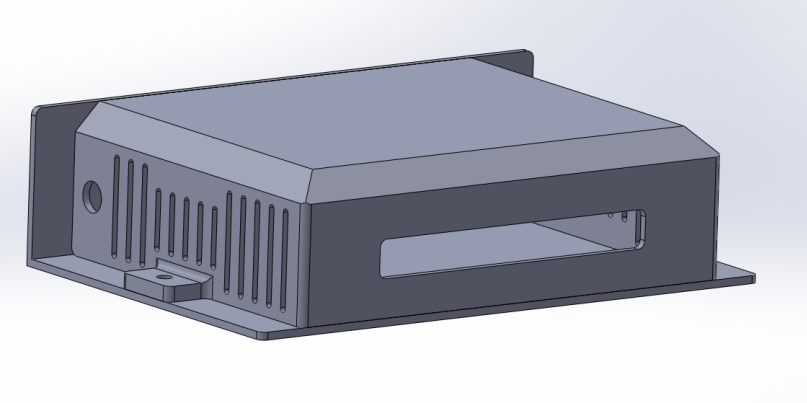


Figure 34: Assembly of the design

23.7 Final product view

24 Budget report

Component	Total (Rs)
Motor and Driver	24000
Component Ordering	120000
PCB Manufacturing	54700
Enclosure Design	4000
Total	**182700**

Table 3: Budget estimation for required components

25 Product specifications

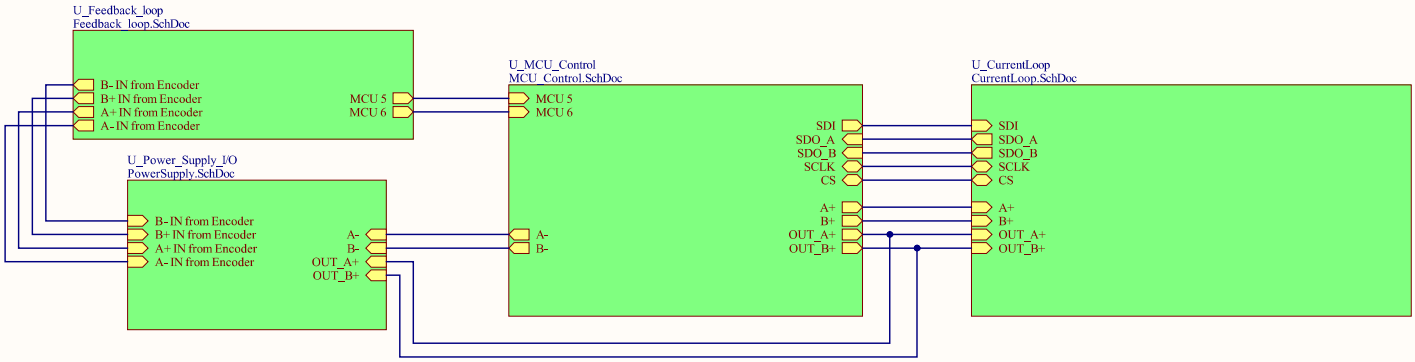
The following table summarizes the key technical specifications of the closed-loop stepper motor driver system developed in this project. The specifications reflect design decisions based on performance goals, component capabilities, and integration requirements.

Parameter	Specification
Motor Type	2-phase NEMA 17 Stepper Motor
Control Mode	Closed-loop (with encoder feedback)
Microcontroller	TI C2000 DSP-based MCU (TMS320F28069PFPS)
Motor Driver IC	Toshiba TB67H400AFTGEL
Operating Voltage	24V DC
Maximum Output Current	Up to 4.5A per phase
Feedback Input	Quadrature Encoder (A/B/Z signals)
Communication Interface	UART /JTAG(optional depending on firmware)
Control Features	Position, Speed, and Error Feedback
PCB Layers	4-Layer PCB with dedicated power and ground planes
Protection Features	Overcurrent, Overtemperature, Reverse Polarity
Firmware Capabilities	Real-time PID control, Encoder decoding, Safety monitoring
Enclosure Material	Aluminum base (heat dissipation), PLA top (customization)
Cooling Method	Passive cooling via heat-sinking base and ventilating holes
Enclosure Size	Approx. 120 mm × 100 mm × 40 mm
Mounting Options	Screw-based standoffs, PCB slot fits, optional DIN rail
Application Areas	CNC systems, 3D printers, robotic arms, precision stages

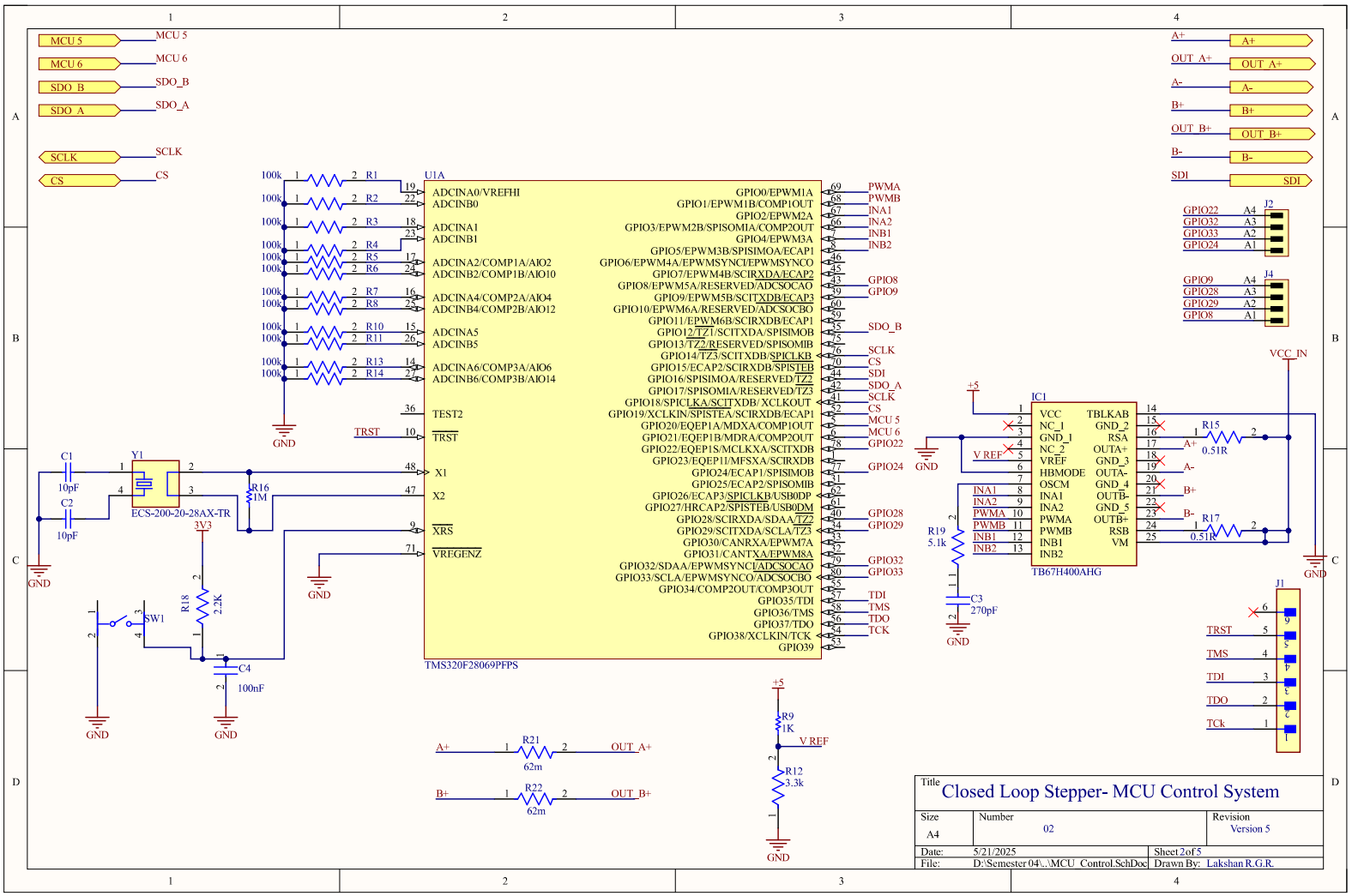
Table 4: Closed-Loop Stepper Motor Driver – Product Specifications

26 Final Product Recap

26.1 Schematic design



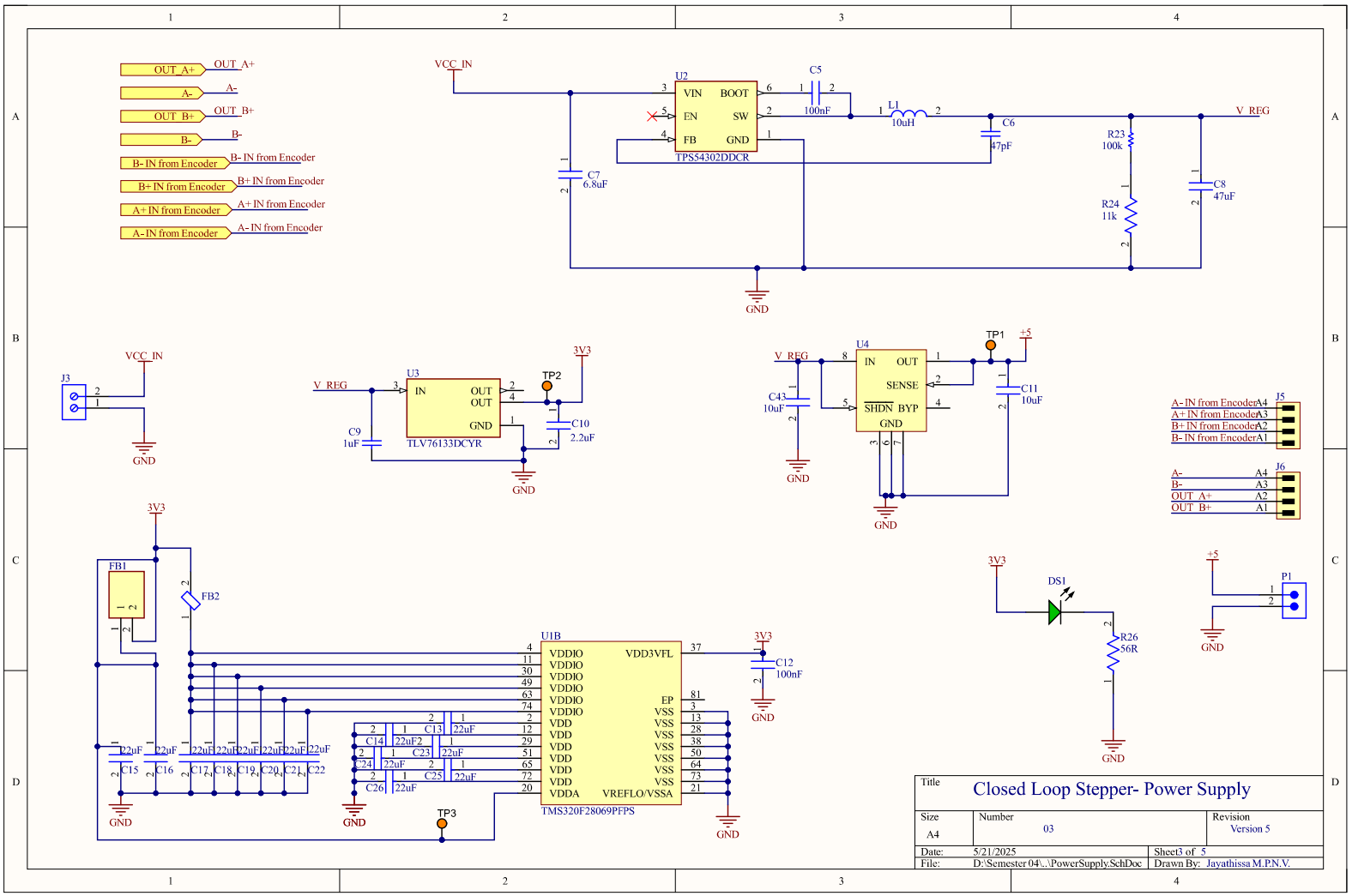
Title	Closed Loop Stepper- Top Level	
Size: A4	Number: 01	Revision: Version 5
Date: 5/21/2025		Sheet 1 of 5
File: D:\Semester 04\...\top_level.SchDoc		Drawn By: Jayathissa M.P.N.V.



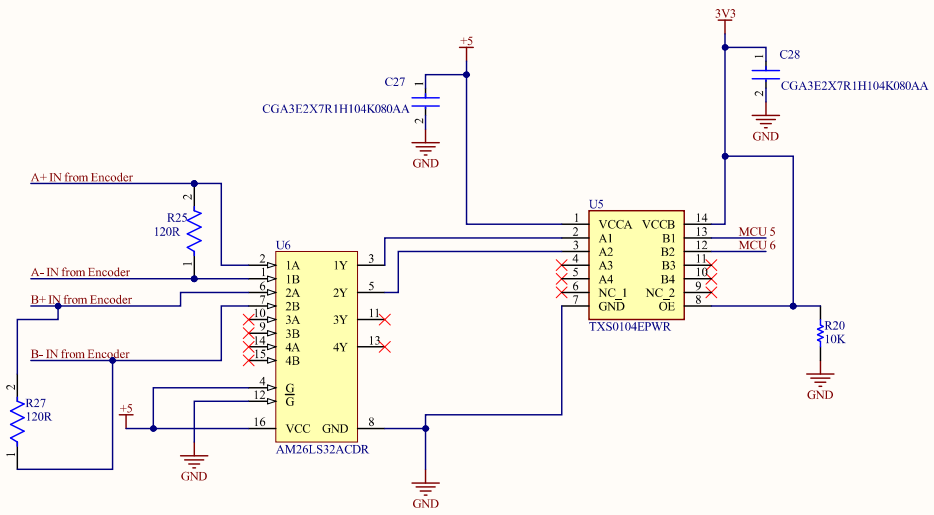
Title: Closed Loop Stepper- MCU Control System

Size	Number	Revision
A4	02	Version 5

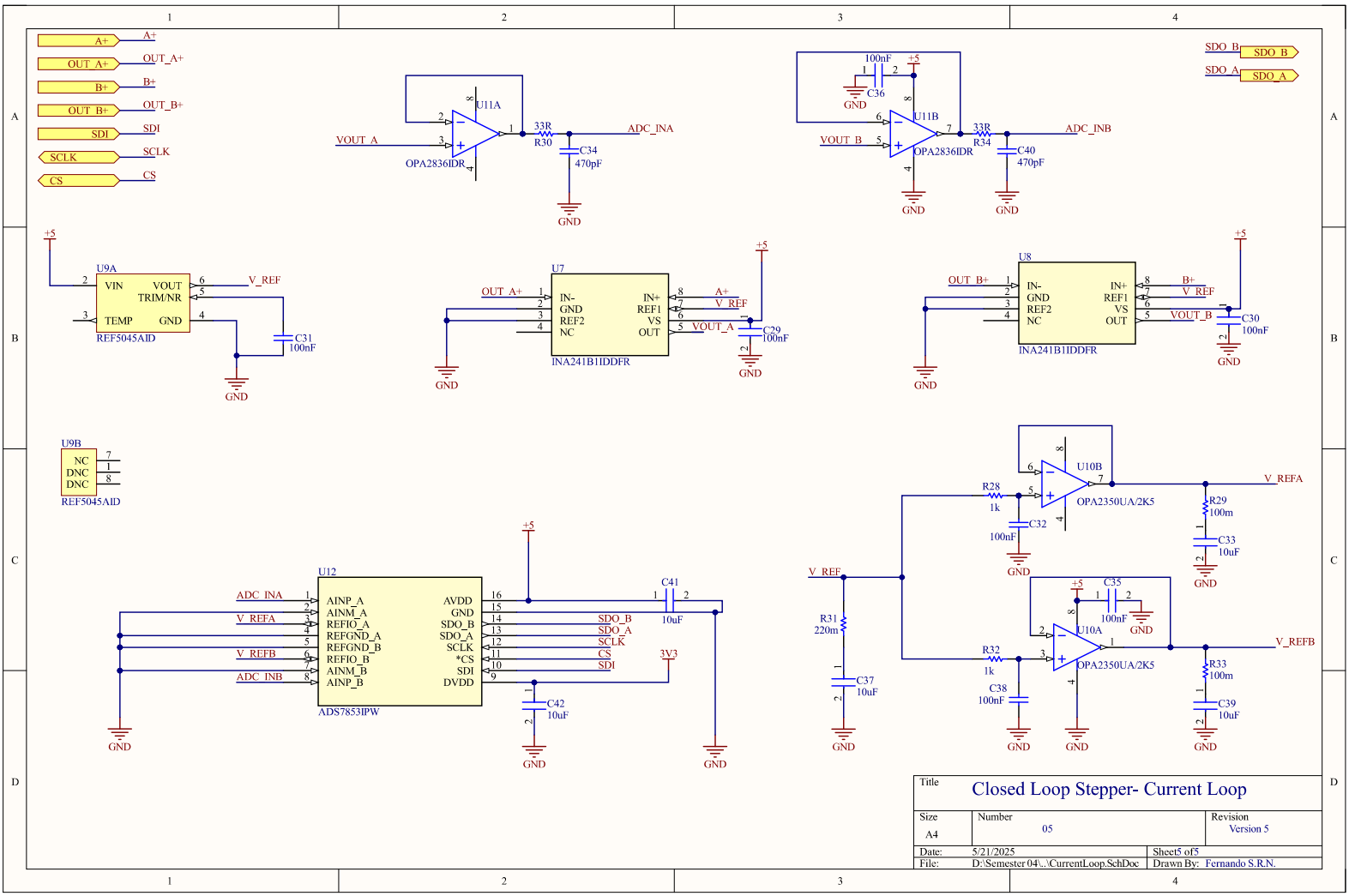
Date: 5/21/2025	Sheet 2 of 5
File: D:\Semester 04\...\MCU_Control.SchDoc	Drawn By: Lakshan R.G.R.



Title	Closed Loop Stepper- Power Supply	
Size: A4	Number: 03	Revision: Version 5
Date: 5/21/2025		Sheet3 of 5
File: D:\Semester 04\...\PowerSupply.SchDoc		Drawn By: Jayathissa M.P.N.V.



Title	Closed Loop Stepper- Feedback Loop	
Size: A4	Number: 04	Revision: Version 5
Date: 5/21/2025	Sheet 4 of 5	
File: D:\Semester 04\...\Feedback_loop.SchDoc	Drawn By: Lakshan K.P.	



Title	Closed Loop Stepper- Current Loop	
Size: A4	Number: 05	Revision: Version 5
Date: 5/21/2025		Sheet5 of5
File: D:\Semester 04\..\CurrentLoop.SchDoc		Drawn By: Fernando S.R.N.

26.2 Coding and algorithm

The closed-loop stepper motor control system relies on a sophisticated algorithm implemented in the firmware to achieve precise motor control. The core of this algorithm is **Field-Oriented Control (FOC)**, which transforms the motor's phase currents into a rotating reference frame to independently control torque and flux.

26.2.1 Flow Chart of the Code

The firmware follows a structured flow to ensure real-time control and responsiveness. The main steps are illustrated below:

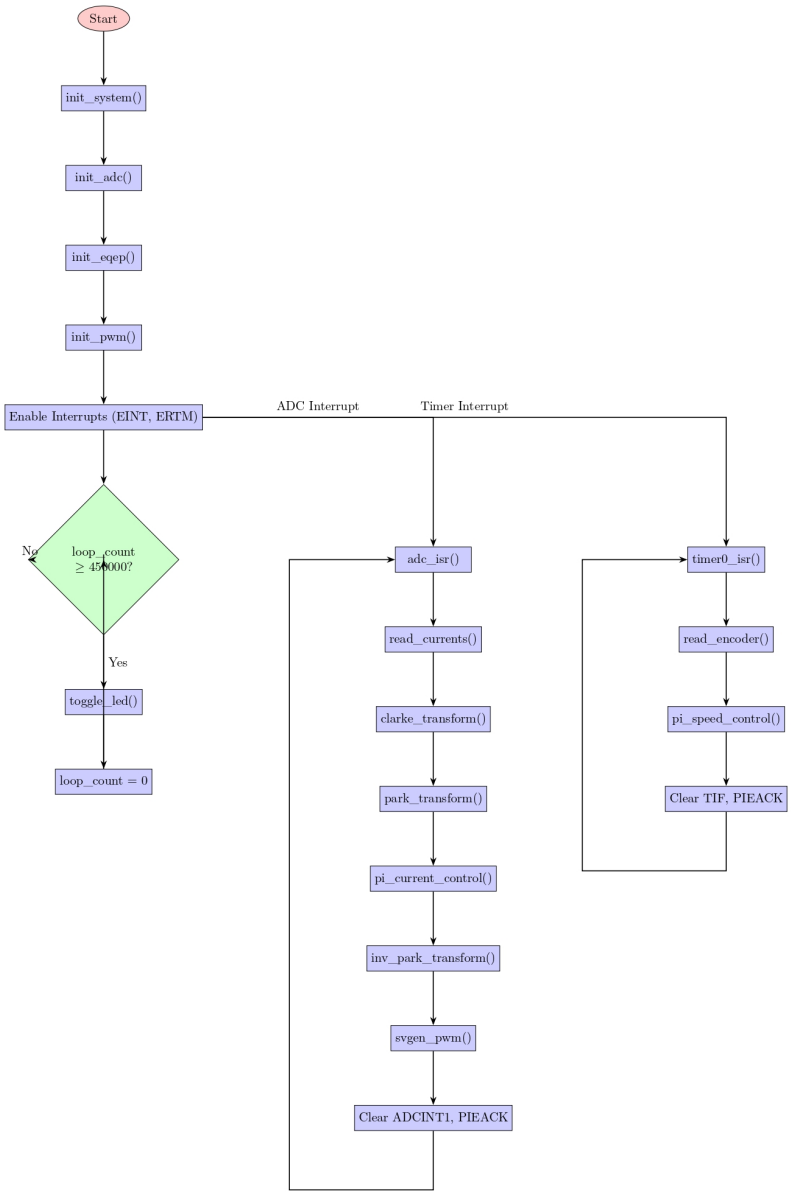


Figure 35: Flow Chart

1. **Initialization**: Configure the microcontroller, ADC, PWM, and encoder interfaces.
2. **Main Loop**: Handles non-time-critical tasks like LED blinking.
3. **ADC Interrupt Service Routine (ISR)**: Executes current control at high frequency (10 kHz).
    - Reads phase currents (`Ia`, `Ib`).
    - Performs Clarke and Park transforms.
    - Implements PI control for current regulation.
    - Generates PWM signals using Space Vector Modulation (SVPWM).
4. **Timer ISR**: Executes speed control at a lower frequency (1 kHz).
    - Reads encoder position and computes speed.
    - Implements PI control for speed regulation.

26.2.2 Algorithms

26.2.3 Clarke and Park Transforms

- Converts phase currents (Ia, Ib) into a stationary reference frame (I, I) and then into a rotating reference frame (Id, Iq).
- Enables independent control of torque (Iq) and flux (Id).
- transforms.c is responsible for Clarke,Prak and Inverse Park transforms which utilizes C2000 libraries IQmath.

```c
// Clarke transform
void clarke_transform(void) {
    Ialpha = Ia;
    Ibeta = _IQmpy(_IQ(0.57735026919), Ia) + _IQmpy(_IQ(1.15470053838), Ib);
}

// Park transform
void park_transform(void) {
    _iq cos_theta = _IQcos(theta_e);
    _iq sin_theta = _IQsin(theta_e);
    Id = _IQmpy(Ialpha, cos_theta) + _IQmpy(Ibeta, sin_theta);
    Iq = _IQmpy(Ibeta, cos_theta) - _IQmpy(Ialpha, sin_theta);
}
```

Figure 36: Clarke and Park Transforms

26.2.4 PI Controllers

- Adjusts Vd and Vq to minimize the error between reference and measured currents (Id$_ref, Iq_ref$).$AdjustsIq_ref tominim$

```
void pi_current_control(void) {
    static _iq Id_error_int = _IQ(0.0), Iq_error_int = _IQ(0.0);
    _iq Id_error, Iq_error, Vmax;

    Id_error = Id_ref - Id;
    Iq_error = Iq_ref - Iq;
    Vmax = _IQ(VDC * 0.5);

    Id_error_int += _IQmpy(KI_D, Id_error);
    if (_IQabs(Id_error_int) > Vmax) Id_error_int = _IQmpy(Vmax, _IQsgn(Id_error_int));
    Iq_error_int += _IQmpy(KI_Q, Iq_error);
    if (_IQabs(Iq_error_int) > Vmax) Iq_error_int = _IQmpy(Vmax, _IQsgn(Iq_error_int));

    Vd = _IQmpy(KP_D, Id_error) + Id_error_int;
    Vq = _IQmpy(KP_Q, Iq_error) + Iq_error_int;

    if (_IQabs(Vd) > Vmax) Vd = _IQmpy(Vmax, _IQsgn(Vd));
    if (_IQabs(Vq) > Vmax) Vq = _IQmpy(Vmax, _IQsgn(Vq));
}
```

Figure 37: PI Control for Current

```
void pi_speed_control(void) {
    static _iq speed_error_int = _IQ(0.0);
    _iq speed_error;

    speed_error = speed_ref - speed;
    speed_error_int += _IQmpy(KI_SPEED, speed_error);
    if (_IQabs(speed_error_int) > _IQ(CURRENT_MAX)) {
        speed_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(speed_error_int));
    }

    Iq_ref = _IQmpy(KP_SPEED, speed_error) + speed_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}
```

Figure 38: PI Control for Speed

```
void pi_position_control(void) {
    static _iq pos_error_int = _IQ(0.0);
    _iq pos_error;

    pos_error = pos_ref - pos;
    pos_error_int += _IQmpy(KI_POS, pos_error);
    if (_IQabs(pos_error_int) > _IQ(CURRENT_MAX)) {
        pos_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(pos_error_int));
    }

    Iq_ref = _IQmpy(KP_POS, pos_error) + pos_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}
```

Figure 39: PI Control for Position

```
void pi_torque_control(void) {
    static _iq torque_error_int = _IQ(0.0);
    _iq torque_error, torque_actual;

    torque_actual = _IQmpy(_IQ(Kt), Iq);
    torque_error = torque_ref - torque_actual;
    torque_error_int += _IQmpy(KI_TORQUE, torque_error);
    if (_IQabs(torque_error_int) > _IQ(CURRENT_MAX)) {
        torque_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(torque_error_int));
    }

    Iq_ref = _IQmpy(KP_TORQUE, torque_error) + torque_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}
```

Figure 40: PI Control for Torque

26.2.5 Space Vector PWM (SVPWM)

- Converts V and V into PWM duty cycles for the motor driver.
- Maximizes voltage utilization and reduces harmonic distortion. consider the deadtime for the motor driver which is 80 ns as given the datasheet.

```
void svgen_pwm(void) {
    _iq Va, Vb;
    Uint16 cmpa, cmpb;
    float temp;

    Va = Valpha;
    Vb = _IQmpy(_IQ(-0.5), Valpha) + _IQmpy(_IQ(0.86602540378), Vbeta);

    temp = _IQtoF(_IQmpy(_IQ(Va / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpa = (Uint16)temp;

    temp = _IQtoF(_IQmpy(_IQ(Vb / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpb = (Uint16)temp;

    EPwm1Regs.CMPA.half.CMPA = cmpa;
    EPwm2Regs.CMPA.half.CMPA = cmpb;
}
```

Figure 41: Space Vector PWM

```
//dead time initialization
void init_pwm_deadband(void) {
    EALLOW; // Enable protected register access

    // Configure Dead-Band for EPWM1 (EPWM1A and EPWM1B)
    EPwm1Regs.DBCTL.bit.OUT_MODE = 3; // Enable Dead-Band for both RED and FED
    EPwm1Regs.DBCTL.bit.POLSEL = 2;   // Active high complementary (AHC) mode
    EPwm1Regs.DBCTL.bit.IN_MODE = 0;  // EPWMxA is source for both RED and FED
    EPwm1Regs.DBRED.bit.DBRED = 8;    // Rising Edge Delay: ~80 ns (8 cycles at 90 MHz)
    EPwm1Regs.DBFED.bit.DBFED = 8;    // Falling Edge Delay: ~80 ns (8 cycles at 90 MHz)

    // Configure Dead-Band for EPWM2 (EPWM2A and EPWM2B)
    EPwm2Regs.DBCTL.bit.OUT_MODE = 3; // Enable Dead-Band for both RED and FED
    EPwm2Regs.DBCTL.bit.POLSEL = 2;   // Active high complementary (AHC) mode
    EPwm2Regs.DBCTL.bit.IN_MODE = 0;  // EPWMxA is source for both RED and FED
    EPwm2Regs.DBRED.bit.DBRED = 8;    // Rising Edge Delay: ~80 ns (8 cycles at 90 MHz)
    EPwm2Regs.DBFED.bit.DBFED = 8;    // Falling Edge Delay: ~80 ns (8 cycles at 90 MHz)

    EDIS; // Disable protected register access
```

Figure 42: Dead Time Intialization

26.2.6 Code

main.c | transforms.c | pi_controllers.c | *pwm_control.c | *1.c | system_init.h | syste

```
#include "motor_control.h"

// Main function
void main(void) {
    init_system();
    init_adc();
    init_eqep();
    init_pwm();

    // Enable interrupts
    EINT; // Global interrupts
    ERTM; // Real-time mode

    while (1) {
        // Blink LED every 500ms
        if (loop_count++ >= 450000) { // 450000 / 90 MHz = 500ms
            toggle_led();
            loop_count = 0;
        }
    }
}
```

Figure 43: Main loop and initialization code.

```c
#ifndef SYSTEM_INIT_H
#define SYSTEM_INIT_H

// Function prototypes
void init_system(void);
void init_adc(void)
void init_eqep(void);
void init_pwm(void);
void init_pwm_deadband(void);

#endif
```

Figure 44: Main loop and initialization code.

```c
// ADC initialization
void init_adc(void) {
    InitAdc(); // Initialize ADC module
    EALLOW; // Enable protected register access

    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1; // Interrupt pulse at end of conversion
    AdcRegs.ADCCTL1.bit.ADCBGPWD = 1;    // Power up band gap
    AdcRegs.ADCCTL1.bit.ADCPWDN = 1;     // Power up ADC

    AdcRegs.ADCSOC0CTL.bit.CHSEL = 0;    // SOC0: ADCINA0 (Ia)
    AdcRegs.ADCSOC1CTL.bit.CHSEL = 1;    // SOC1: ADCINA1 (Ib)
    AdcRegs.ADCSOC0CTL.bit.ACQPS = 6;    // Acquisition window: 6 cycles
    AdcRegs.ADCSOC1CTL.bit.ACQPS = 6;    // Acquisition window: 6 cycles
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 5;  // Trigger: ePWM1 SOCA
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 5;  // Trigger: ePWM1 SOCA

    AdcRegs.ADCINTSOCSEL1.bit.SOC1 = 1;  // SOC1 triggers ADCINT1
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;// Clear ADCINT1 flag
    AdcRegs.INTSEL1N2.bit.INT1E = 1;     // Enable ADCINT1
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;   // Enable PIE interrupt for ADCINT1

    EDIS; // Disable protected register access

    AdcRegs.ADCSOCFRC1.all = 0x0003;     // Force SOC0 and SOC1 to start
}
```

Figure 45: ADC ISR for high-frequency current control.

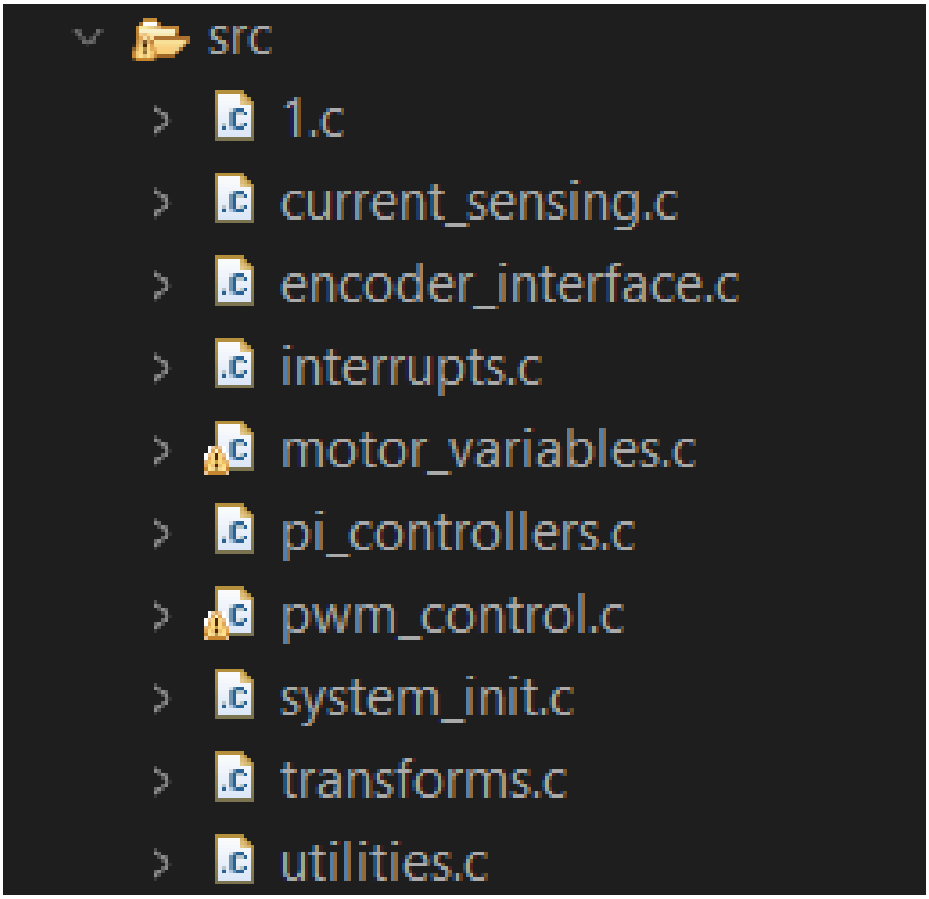
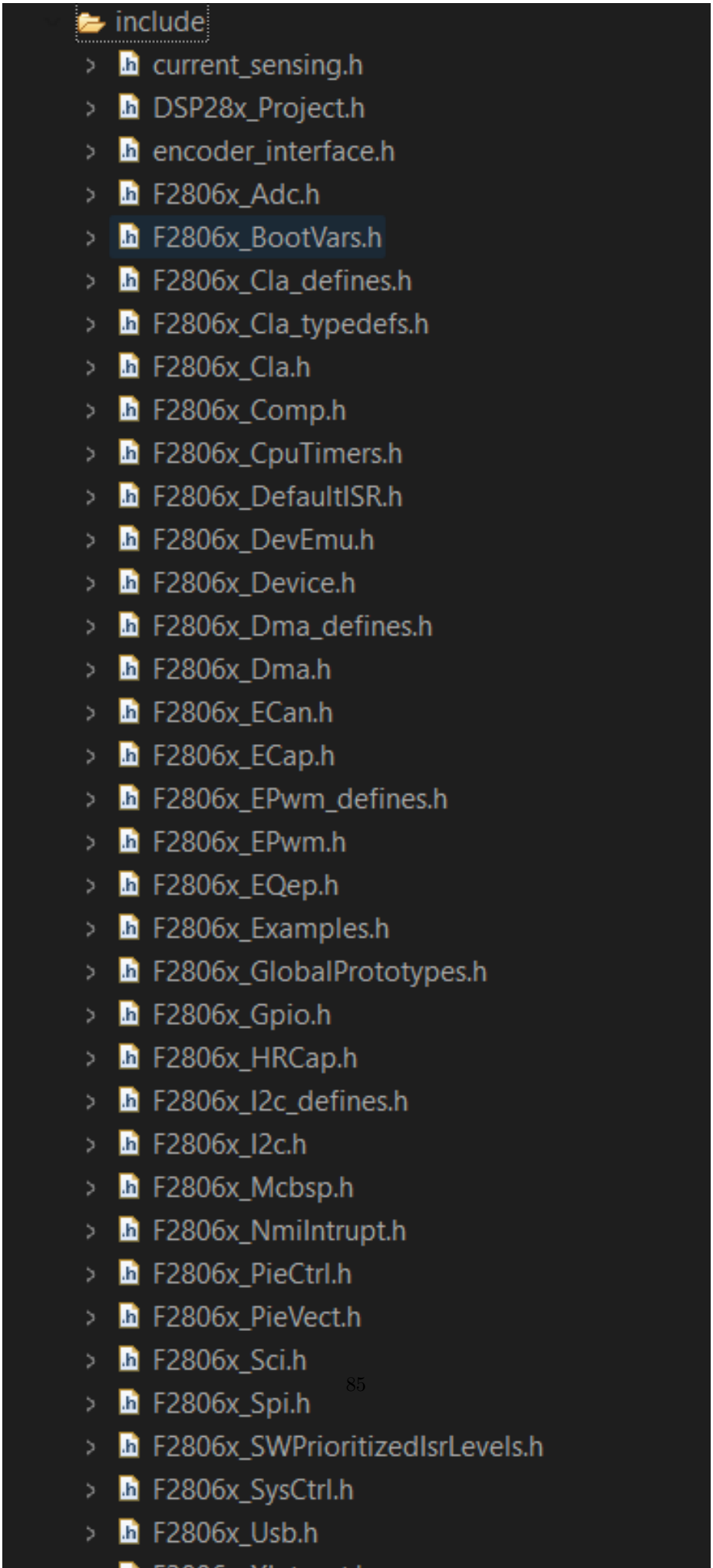


Figure 46: source Files.


include
current_sensing.h
DSP28x_Project.h
encoder_interface.h
F2806x_Adc.h
F2806x_BootVars.h
F2806x_Cla_defines.h
F2806x_Cla_typedefs.h
F2806x_Cla.h
F2806x_Comp.h
F2806x_CpuTimers.h
F2806x_DefaultISR.h
F2806x_DevEmu.h
F2806x_Device.h
F2806x_Dma_defines.h
F2806x_Dma.h
F2806x_ECan.h
F2806x_ECap.h
F2806x_EPwm_defines.h
F2806x_EPwm.h
F2806x_EQep.h
F2806x_Examples.h
F2806x_GlobalPrototypes.h
F2806x_Gpio.h
F2806x_HRCap.h
F2806x_I2c_defines.h
F2806x_I2c.h
F2806x_Mcbsp.h
F2806x_NmiIntrupt.h
F2806x_PieCtrl.h
F2806x_PieVect.h
F2806x_Sci.h
F2806x_Spi.h
F2806x_SWPrioritizedIsrLevels.h
F2806x_SysCtrl.h
F2806x_Usb.h

26.3 Enclosure Design

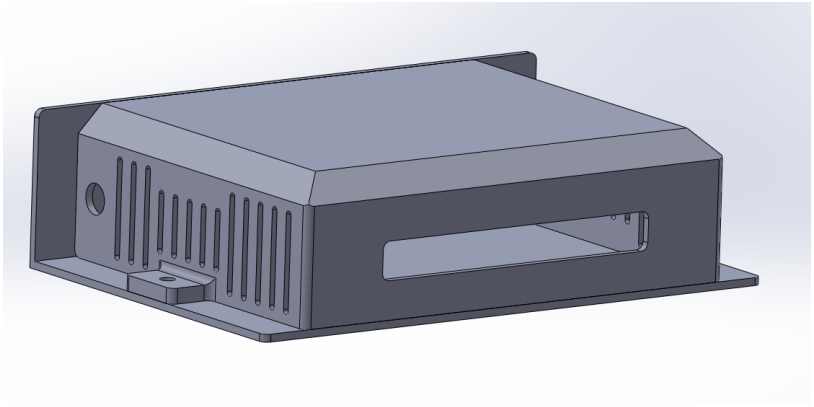


Figure 48: Assembly of the design

27 Conclusion of the product design

The development of the closed-loop stepper motor driver system was a comprehensive engineering effort that integrated multiple disciplines—including embedded programming, electronic circuit design, PCB layout, thermal management, and mechanical enclosure design. We began by reverse engineering an industrial driver to gain insights into the functional blocks and signal interactions, which informed our schematic design. The schematics were developed hierarchically and refined through multiple iterations, addressing integration and signal integrity challenges while referencing proven design standards.

To support complex routing and power distribution needs, we transitioned from an initial two-layer PCB to a robust four-layer design, optimizing for performance, EMI control, and manufacturability. In parallel, the embedded firmware was developed to implement closed-loop control logic, handle encoder feedback, and communicate with external interfaces—ensuring precise motor positioning and real-time adaptability.

The mechanical enclosure was designed with a focus on compactness, durability, and thermal efficiency. We adopted a hybrid material approach, using an aluminum base for heat dissipation and a 3D-printed PLA top for ease of customization and port accessibility. Multiple iterations were conducted to refine the enclosure size, layout, and mounting strategy, ultimately arriving at a solution that is functional, manufacturable, and suitable for real-world deployment.

Through careful planning, iterative prototyping, and cross-domain integration, the final system represents a reliable and cost-effective solution for closed-loop stepper motor control, suitable for industrial automation, robotics, or precision motion applications.

28 Final Design Report

This project focuses on the design and implementation of a closed-loop stepper motor driver utilizing a TI C2000 microcontroller(TMS320F28069PFPS) and a Toshiba TB67H400AHG driver IC to control a NEMA 17 stepper motor with enhanced precision and efficiency. Unlike traditional open-loop stepper systems, this design incorporates real-time feedback from an encoder, enabling precise control of motor position, speed, and torque under varying load conditions.

The motor control strategy is based on Field-Oriented Control (FOC), which improves the dynamic performance and energy efficiency by treating the stepper motor like a synchronous AC machine. This method allows smoother motion, reduced vibration, and minimized power losses. The microcontroller handles real-time current regulation, position tracking using encoder feedback, and PWM signal generation for the driver.

A custom hardware platform was developed, including power electronics, signal conditioning circuits, and a microcontroller interface. Software was developed in C using TI's Code Composer Studio and leverages the C2000's advanced peripherals such as ePWM, ADC, and eQEP modules.

28.1 Reverse Engineering

This section includes a detailed reverse engineering report of CL42T-V41 closed loop stepper driver.

28.1.1 Block Diagram of the Circuit

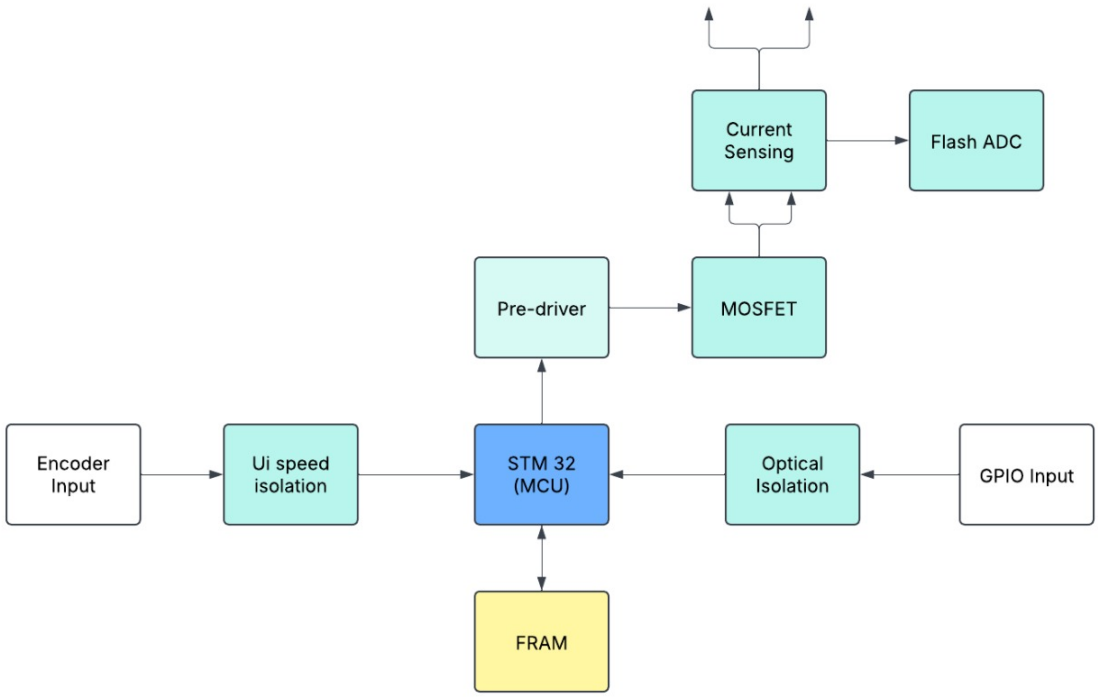


Figure 49: Block Diagram

28.1.2 MCU Circuit

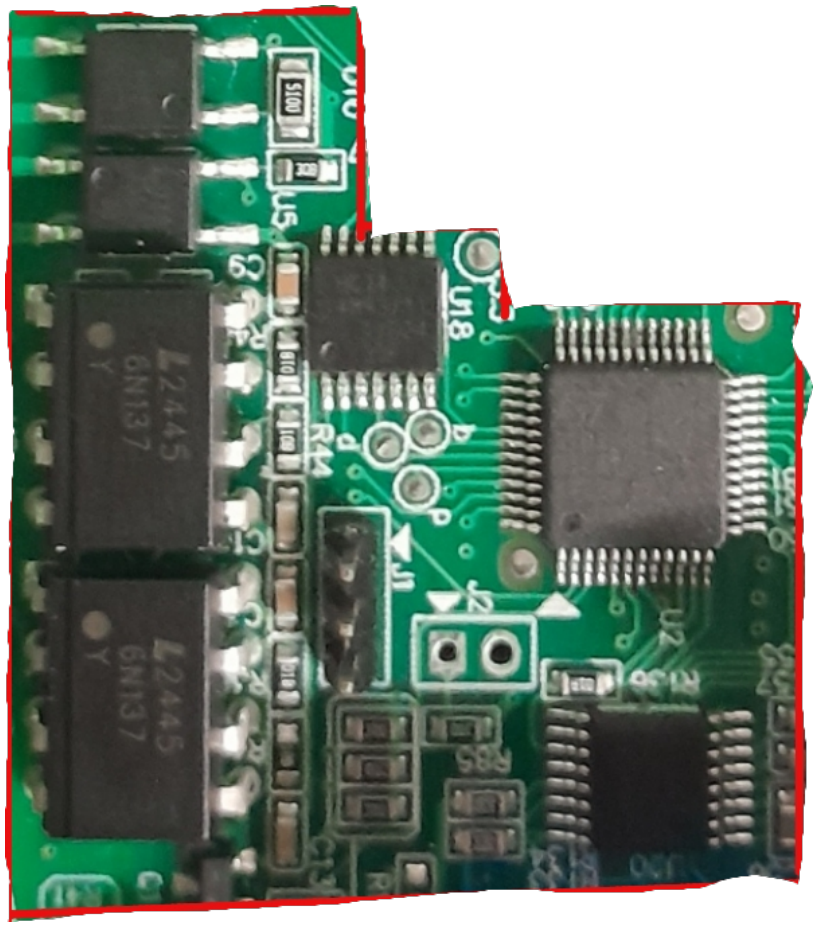


Figure 50: MCU circuit consisting of optocouplers

Main components used:

- **L2445 Optocouplers** – Provide electrical isolation between the MCU and high-voltage sections of the motor driver. They protect the MCU by transmitting control or feedback signals optically across an isolation barrier.
- **HC14 (SN74HC14)** – A hex inverter with Schmitt-trigger inputs used for signal conditioning. It cleans up noisy encoder signals or switch inputs, ensuring the MCU receives stable digital inputs for precise feedback control.
- **TPT3232E (likely MAX3232E or equivalent)** – A dual RS-232 transceiver used to interface the MCU with serial communication lines (e.g., to a PC or external controller). It converts between TTL logic levels and RS-232 voltage levels.
- **FM24C16D** – A 16-kbit I2C FRAM (Ferroelectric RAM) chip used to store parameters such as calibration data, motor profiles, or error logs. Unlike EEPROM, it offers fast writes and virtually unlimited endurance.
- **T4032 BDOe** – Likely refers to the TPT4032, a quad RS-422 differential receiver. It is possibly used to receive differential encoder signals from the stepper motor over long distances. The IC converts the high-speed RS-422 differential signals into standard TTL/CMOS logic

levels readable by the MCU. Its strong ESD protection and wide input voltage tolerance ensure reliable communication in noisy industrial motor control environments.

28.1.3 Current Sensing Circuit

Main components used:

- **20 mΩ Current Sense Resistors** – Used to measure the current flowing to the stepper motor. The small voltage drop across these low-value resistors is sensed by the op-amps or ADCs in the MCU to monitor motor current in real-time and enable closed-loop control.
- **SGM8634 Quad OpAmp** – This is a quad low-noise CMOS operational amplifier. It can be used to implement a simple **Flash ADC** (Analog-to-Digital Converter) when paired with a **resistor voltage divider** network (ladder) and a set of **comparators**.

  Flash ADC Concept with Op-Amps and Resistors:

  - A **resistor ladder** creates a set of evenly spaced reference voltages across a range (e.g., 0V to 3.3V).
  - Each of the four op-amps in the SGM8634 can be configured as a **comparator**, comparing the analog input signal to one of the reference voltages.
  - The output of each comparator (op-amp) goes high or low depending on whether the input signal is above or below the reference threshold.
  - The digital outputs of the comparators form a **thermometer code**, which can be decoded into a binary output by the MCU or a logic circuit.

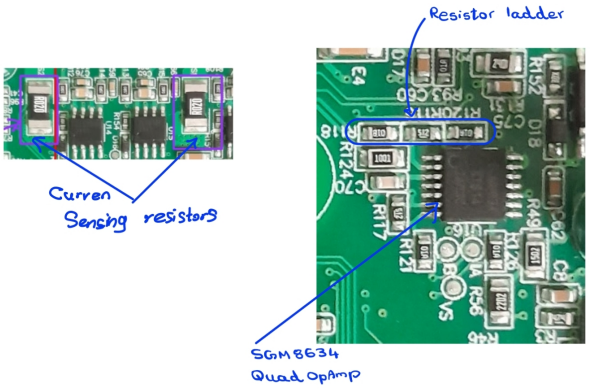


Figure 51: Current sense components and flash ADC implementation

28.1.4 Driver Circuit

Main components used:

- **SLM2004S** – This is a smart gate driver IC designed for driving MOSFETs in full H-bridge or half-bridge configurations. Each SLM2004S typically contains two high-side and two low-side gate drivers, allowing it to control a full H-bridge. It provides integrated dead-time control and other motor controls which are essential for robust and efficient motor control.

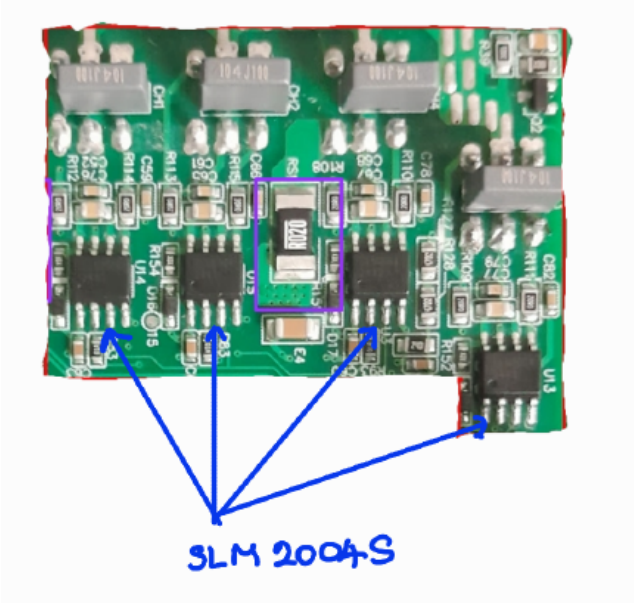


Figure 52: Components of driver circuit

- **8 MOSFETs** – These are power transistors used to switch current to the motor phases. In an H-bridge configuration, four MOSFETs are used per phase (two per leg), enabling precise control over current direction and amplitude for each winding of the stepper motor.

Working Together in a Closed-Loop Stepper Motor Driver:
To drive a 2-phase bipolar stepper motor in a closed-loop system:

- **4 SLM2004S** ICs are used. Since each phase of the motor requires an H-bridge for bidirectional current control, two SLM2004S ICs can control one phase.
- With **4 SLM2004S ICs** and **8 MOSFETs**, you can construct **two full H-bridges** – one for each motor phase.
- The stepper motor phases are driven with current controlled via the MOSFETs, under the supervision of the SLM2004S drivers.
- In a closed-loop system, feedback from a position or current sensor (e.g., via op-amp/comparator circuits) is processed by the MCU, which adjusts the gate drive signals sent to the SLM2004S ICs to maintain precise motor positioning and torque.

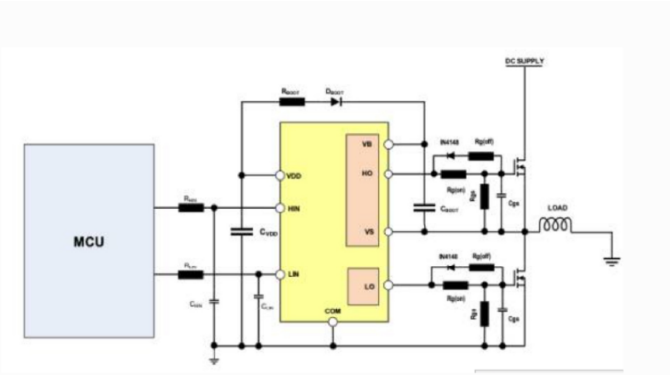


Figure 53: Working of driver

28.1.5 Power Circuit

Main components used:

- **150 μH Inductor** – This is likely part of a buck (step-down) converter circuit. In buck regulators, the inductor stores and transfers energy to step down the voltage efficiently from a higher supply voltage (e.g., 24V) to a lower voltage level.
- **LD117AG** – This is a low dropout (LDO) linear voltage regulator. It is used to provide a stable and precise output voltage (typically 3.3V) from a higher input voltage. The "G" variant generally refers to specific package or thermal characteristics

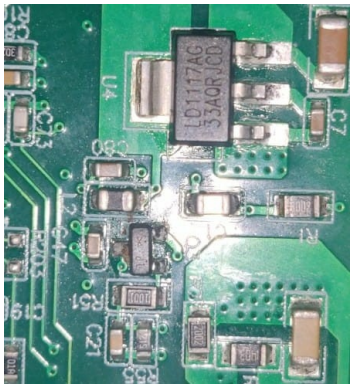


Figure 54: LD117AG

Some components on the PCB could not be identified as they are either unmarked or are custom ICs, likely proprietary to the manufacturer.

Possible working scenario: Based on the circuitry observed, it is likely that the 24V input supply is first stepped down using a buck converter, and the resulting intermediate voltage is then further regulated by the LD117AG LDO to provide a clean and stable 3.3V supply for powering the MCU and other sensitive logic components.

28.1.6 Comparison

Table 5: Comparison: Leadshine Closed-loop Driver vs Custom Driver

Feature / Component	**Leadshine Closed Loop Driver**	**Custom Driver**
Microcontroller (MCU)	**STM32G4** – ARM Cortex-M4 with motor control peripherals	**TI C2000** (TMS320 series) – Real-time control MCU
Motor Driver IC	Pre-driver + External MOSFETs	**Toshiba TB67H400** – Integrated MOSFETs in single chip
Signal Isolation	Uses **optocouplers** – For isolation and accurate signal transmission between control and power domains	**No optocouplers** – Direct interfacing between MCU and driver IC
Current Sensing	External 20 mΩ shunt resistors + Likely op-amp based **Flash ADC**	External 62 mΩ shunt resistors + **SAR ADC**
Encoder Feedback Interface	RS-422 receiver (e.g., **TPT4032**) to handle encoder feedback	Uses differential line receiver, level shifter & QEP module of MCU to receive feedback
Voltage Regulation	Buck stage followed by **LD117AG LDO** to 3.3V for powering MCU	Buck stage followed by two LDO's to convert to 3.3V & 5.5V

28.2 Prototype design

To verify and debug the closed-loop control algorithms, a compact **prototype test bench** was developed. The setup included the following key components:

- **TI C2000 LaunchPad** (e.g., F280049C or F28379D) to run the Field-Oriented Control (FOC) algorithm.
- **TB67H400AFTGEL stepper motor driver IC** mounted on a custom PCB for interfacing with the motor and microcontroller.
- **NEMA 17 stepper motor** equipped with a **rotary incremental encoder** for real-time position feedback.
- **12–24V regulated DC power supply**, depending on load requirements and motor voltage.
- **Breadboard and jumper wires** used for signal routing during early development, later replaced by a custom-designed PCB.
- **Oscilloscope and logic analyzer** to monitor PWM signals, current waveforms, and encoder pulses.
- **Serial communication** (UART or USB) enabled for logging data and tuning parameters using a PC interface or debug console.
- A **test load** attached to the motor shaft to simulate mechanical load conditions during motion profiling and step response testing.

The prototype allowed for iterative code testing, including:

- Encoder signal decoding and position tracking
- PI controller tuning for current and position loops
- Closed-loop FOC implementation and validation under static and dynamic loads

This test bench proved essential for debugging, refining the control logic, and ensuring safe operation before deploying the design on a final PCB.

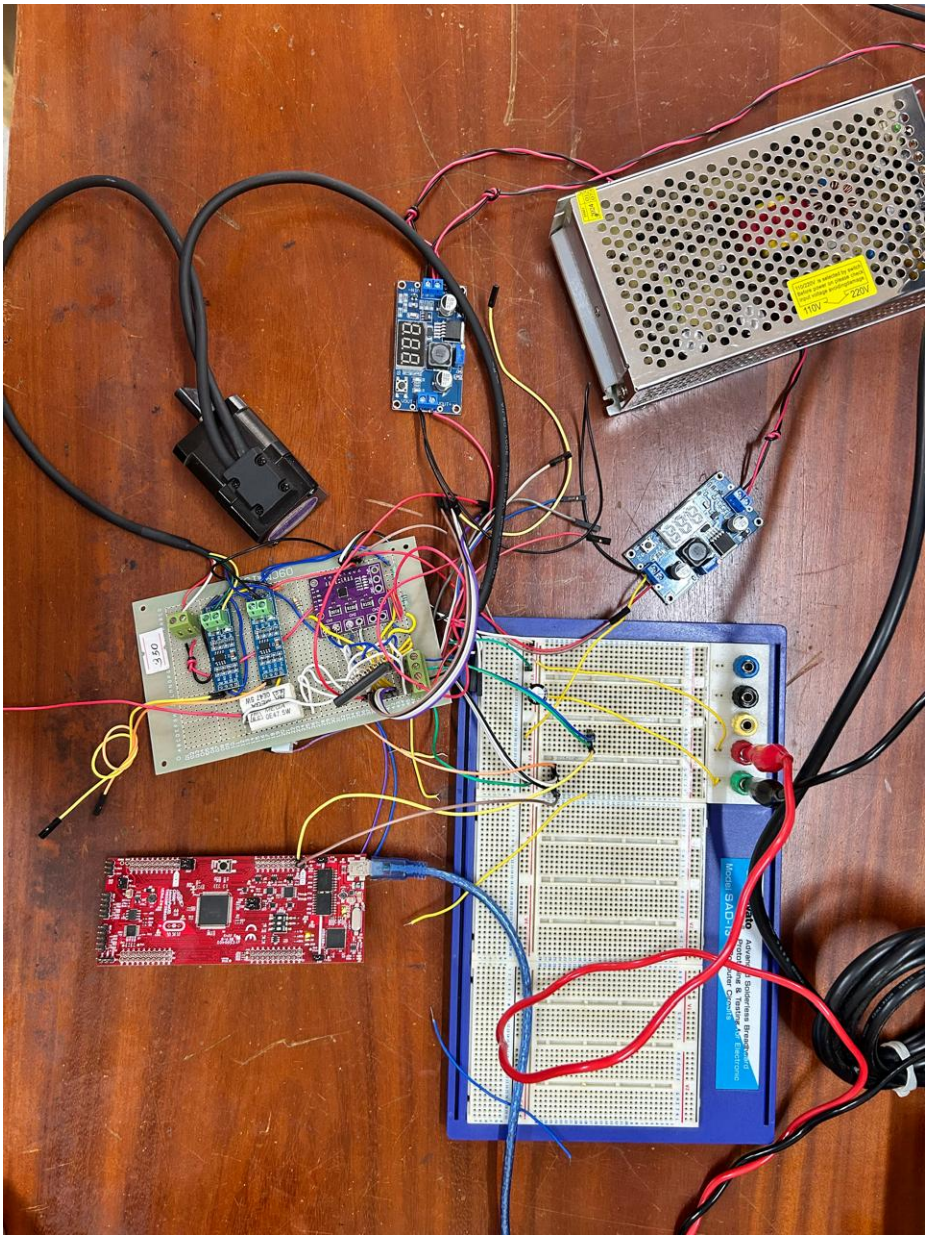


Figure 55: Prototype setup for testing the closed-loop stepper motor driver. The system includes the TI C2000 LaunchPad, TB67H400AFTGEL driver board, NEMA 17 stepper motor with encoder, and power electronics.

28.3 Circuit Design

29 Circuit and Schematic Design

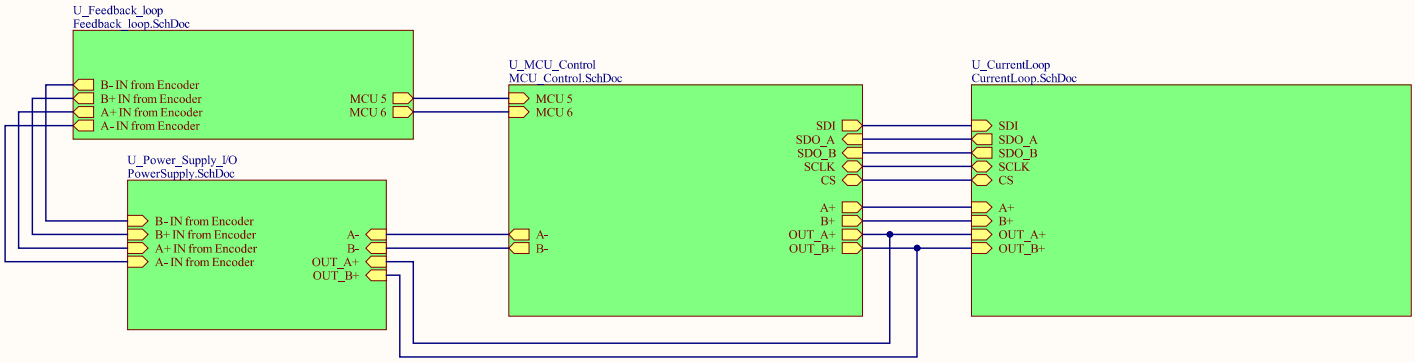
The circuit was designed to enable precise and reliable control of a NEMA 17 stepper motor using a TI C2000 microcontroller and the Toshiba TB67H400AFTGEL motor driver. The schematic integrates power stages, signal interfaces, and feedback mechanisms critical for closed-loop operation.

- **Motor Driver:** The TB67H400AFTGEL dual H-bridge driver supports high current and voltage, with built-in protections such as overcurrent, thermal shutdown, and undervoltage

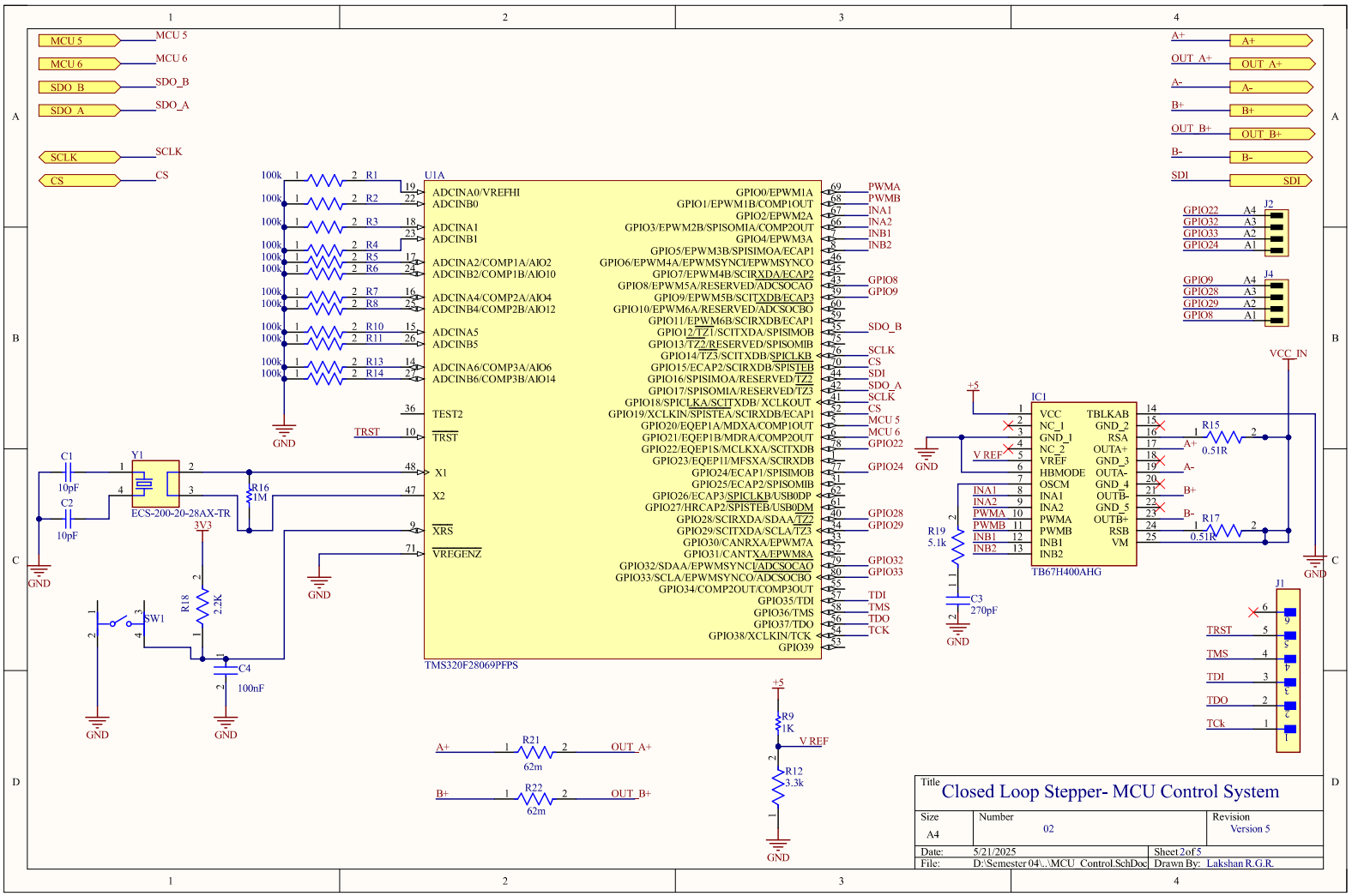
lockout. It receives PWM inputs from the microcontroller and drives the motor phases accordingly.

- **Microcontroller Interface:** PWM and GPIO signals from the TI C2000 LaunchPad control the motor driver. The ePWM module generates high-resolution PWM signals for precise control, while GPIOs handle driver enable/reset logic.
- **Feedback Path:** A rotary quadrature encoder is connected to the eQEP module of the C2000 to provide real-time position feedback, enabling accurate closed-loop control.
- **Current Sensing:** Shunt resistors with differential amplifiers may be included to monitor phase currents for FOC current loop regulation.
- **Power Supply:** A 12–24V DC supply powers the motor. Linear or switching regulators are used to generate 5V and 3.3V rails for the microcontroller and encoder.

The schematic was developed using *[Tool Name]*, and the PCB layout was optimized for compact size, minimal EMI, and proper grounding. Differential signal routing and power-ground separation techniques were applied to maintain signal integrity and reliable operation.



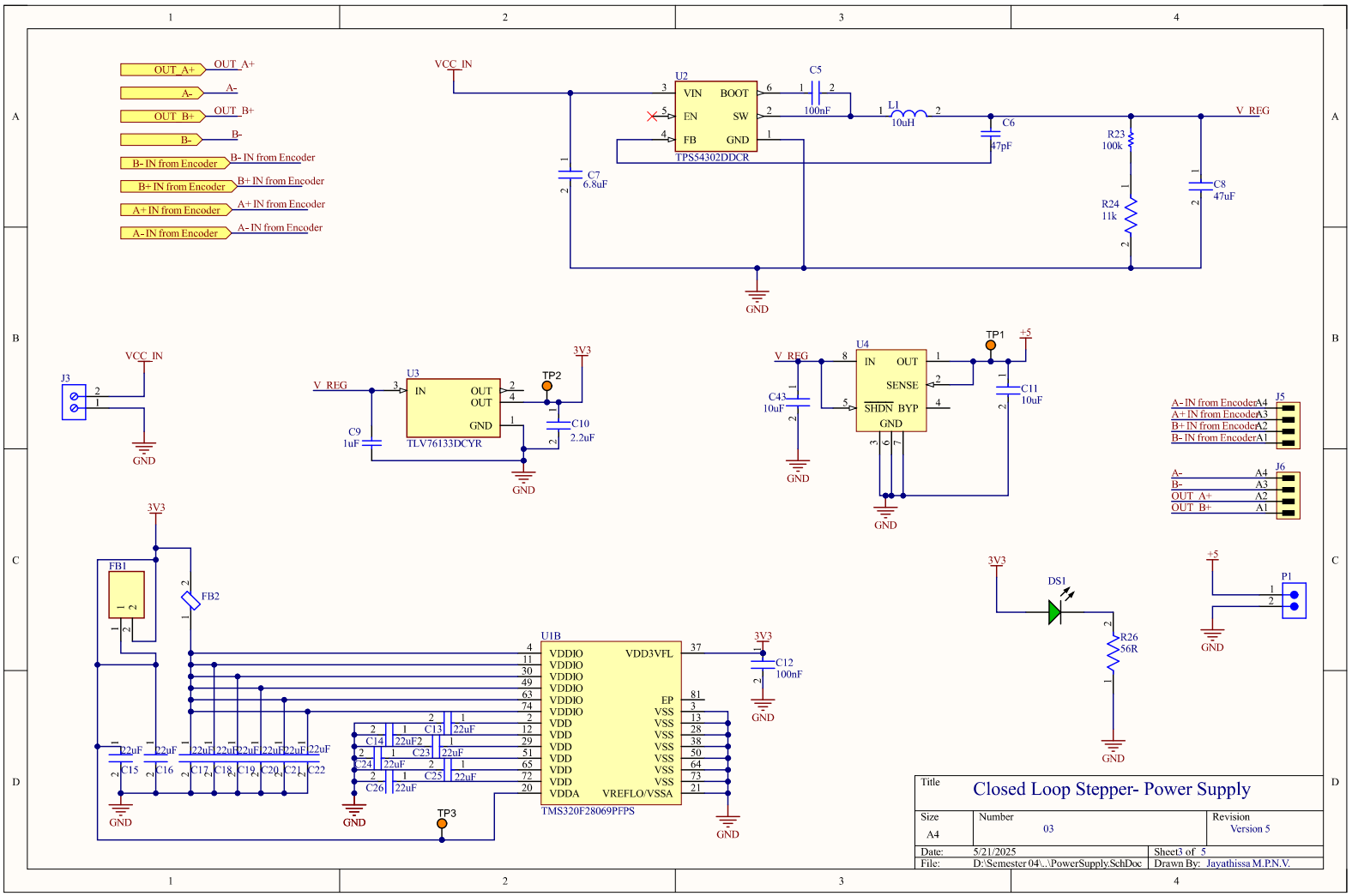
Title	Closed Loop Stepper- Top Level	
Size: A4	Number: 01	Revision: Version 5
Date: 5/21/2025		Sheet 1 of 5
File: D:\Semester 04\...\top_level.SchDoc		Drawn By: Jayathissa M.P.N.V.



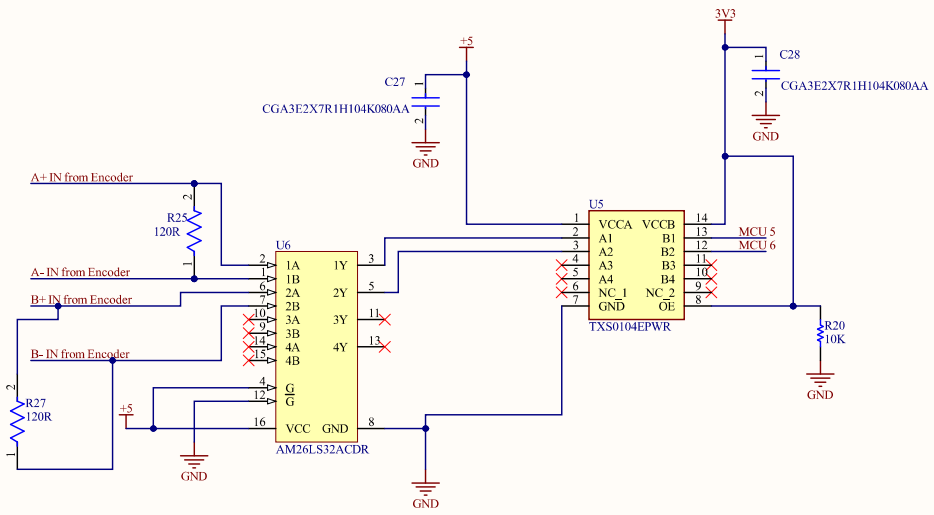
Title: Closed Loop Stepper- MCU Control System

Size	Number	Revision
A4	02	Version 5

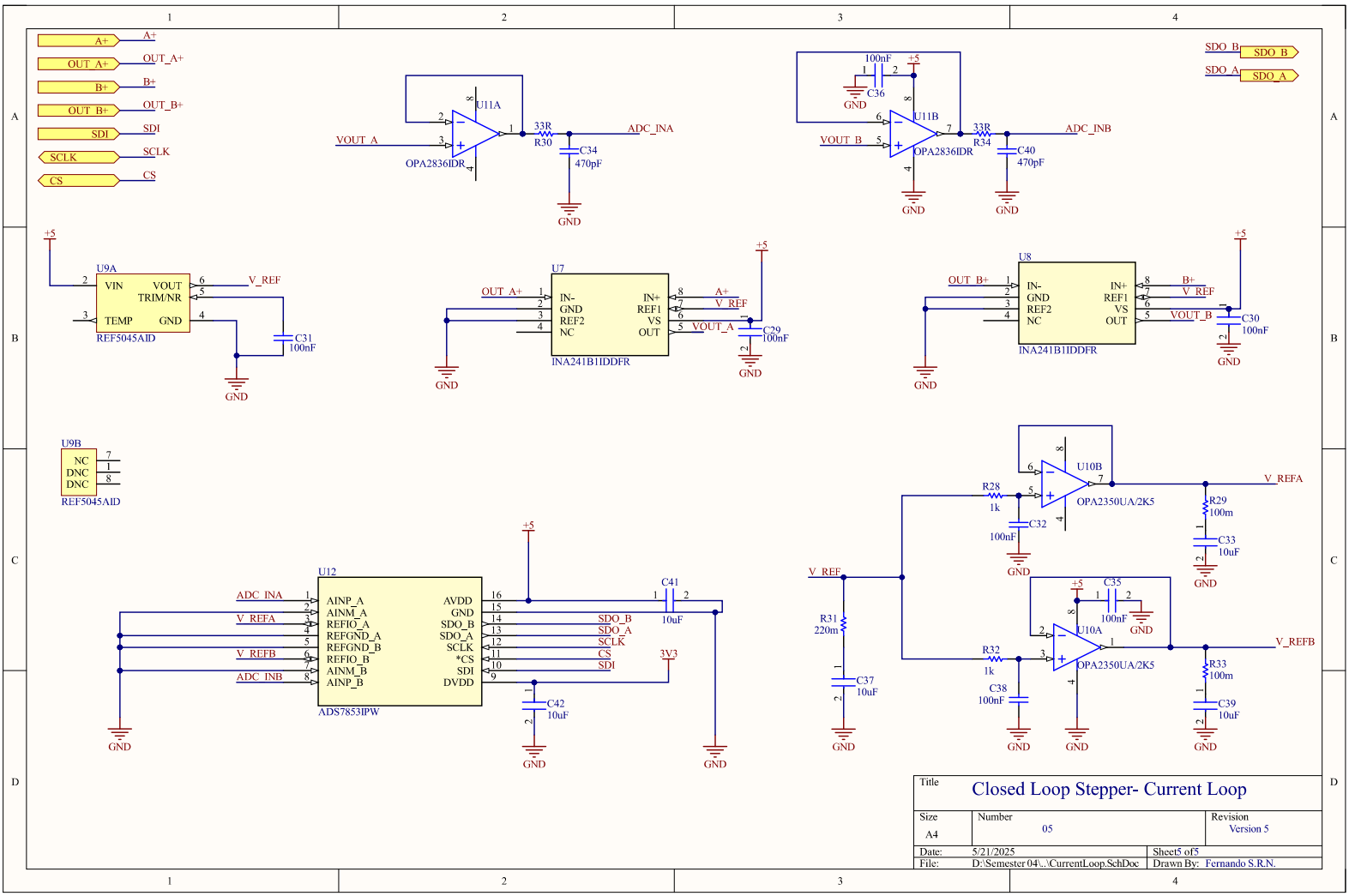
Date: 5/21/2025	Sheet 2 of 5
File: D:\Semester 04\...\MCU_Control.SchDoc	Drawn By: Lakshan R.G.R.



Title	Closed Loop Stepper- Power Supply	
Size: A4	Number: 03	Revision: Version 5
Date: 5/21/2025		Sheet3 of 5
File: D:\Semester 04\...\PowerSupply.SchDoc		Drawn By: Jayathissa M.P.N.V.



Title	Closed Loop Stepper- Feedback Loop	
Size: A4	Number: 04	Revision: Version 5
Date: 5/21/2025	Sheet 4 of 5	
File: D:\Semester 04\...\Feedback_loop.SchDoc	Drawn By: Lakshan K.P.	



Title	Closed Loop Stepper- Current Loop	
Size: A4	Number: 05	Revision: Version 5
Date: 5/21/2025		Sheet5 of5
File: D:\Semester 04\..\CurrentLoop.SchDoc		Drawn By: Fernando S.R.N.

29.1 PCB Design

30 PCB Design

The PCB was designed to integrate the motor driver circuitry, the encoder feedback interface, and the microcontroller connections into a compact and reliable layout suitable for closed-loop stepper motor control.

- **Board Stackup:** A four-layer PCB layout was used, with the top layer allocated for signal traces and components, the bottom layer for signal traces, and the other two layers featuring a solid ground plane and power returns to reduce EMI.
- **Power and Ground Routing:** Wide copper pours were used for the motor power and ground nets to support high current flow and minimize voltage drops. Analog and digital grounds were separated and connected at a single-point ground to prevent noise coupling.
- **Driver Placement:** The TB67H400AFTGEL driver IC was positioned near the motor output terminals to reduce trace length and inductive interference. High-frequency decoupling capacitors were placed close to the driver's supply pins.
- **Microcontroller Interface:** A pin header allowed seamless connection to the TI C2000 LaunchPad, enabling modular testing and easy firmware updates.
- **Encoder Input:** A dedicated connector supported the quadrature encoder signals, with careful routing of A/B lines to maintain signal integrity and timing accuracy.
- **Thermal Design:** Adequate copper area and thermal vias were provided under the driver IC to dissipate heat efficiently and ensure thermal reliability during high current operation.
- **Debugging Support:** Test points were included for PWM signals, phase currents, and encoder channels, allowing easy access for oscilloscope probes during validation.
- **Silkscreen Markings:** All connectors, components, and test points were labeled clearly to support assembly and troubleshooting.

The PCB was designed using *[Tool Name]* and adhered to good practices for power electronics, including trace width calculation, component placement, and thermal management to ensure safe and reliable performance.

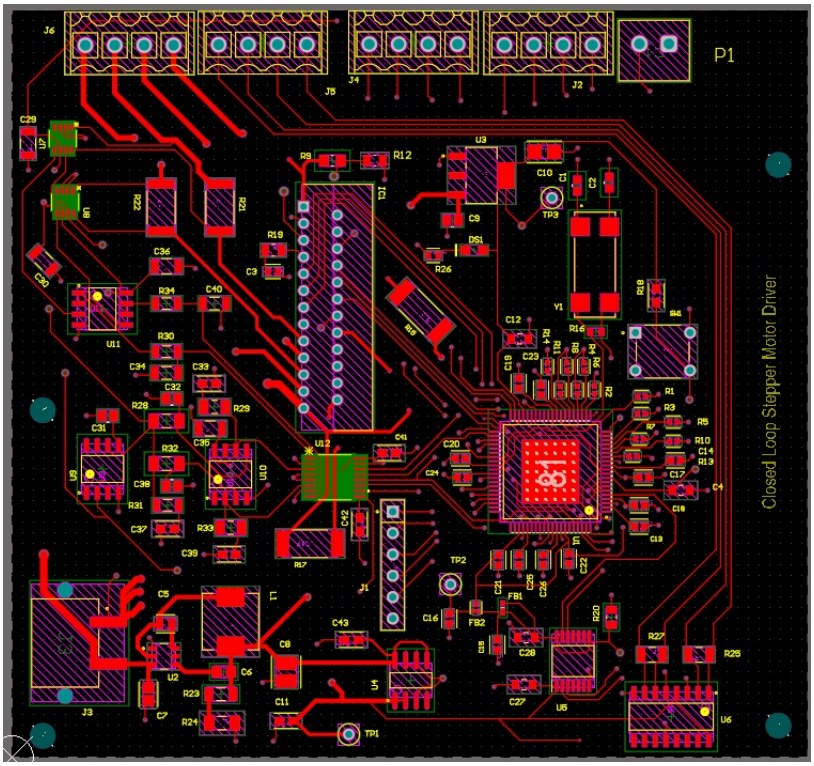


Figure 56: Top level of the PCB)

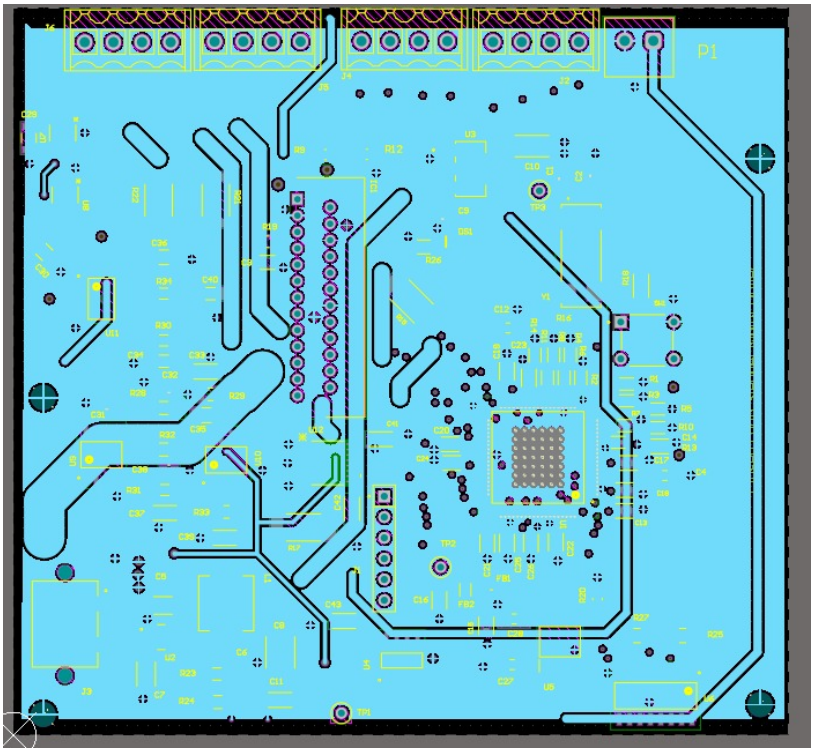


Figure 57: Second Layer of the PCB

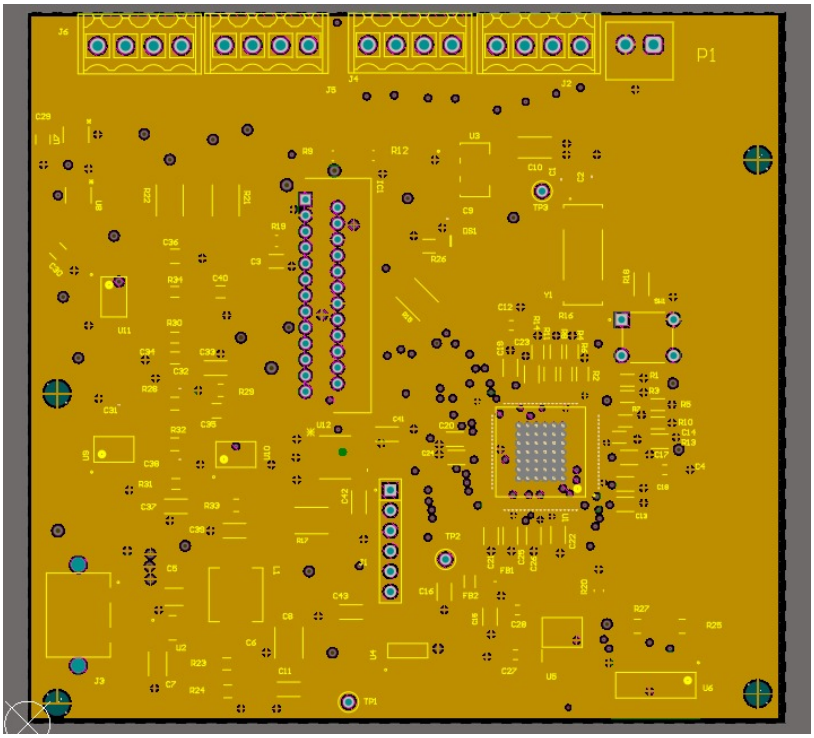


Figure 58: Third Layer of the PCB

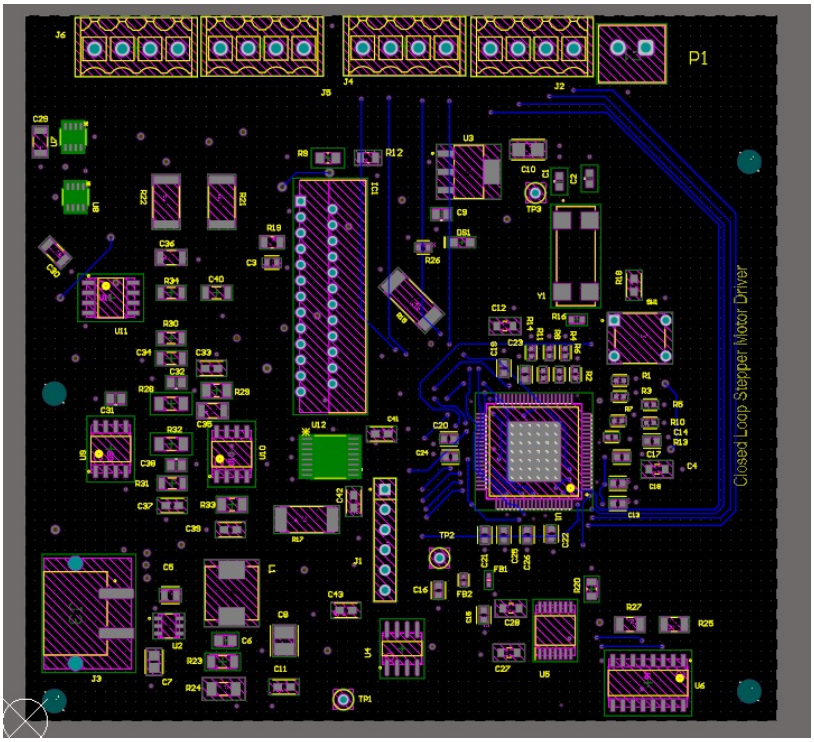


Figure 59: Bottom Layer of the PCB

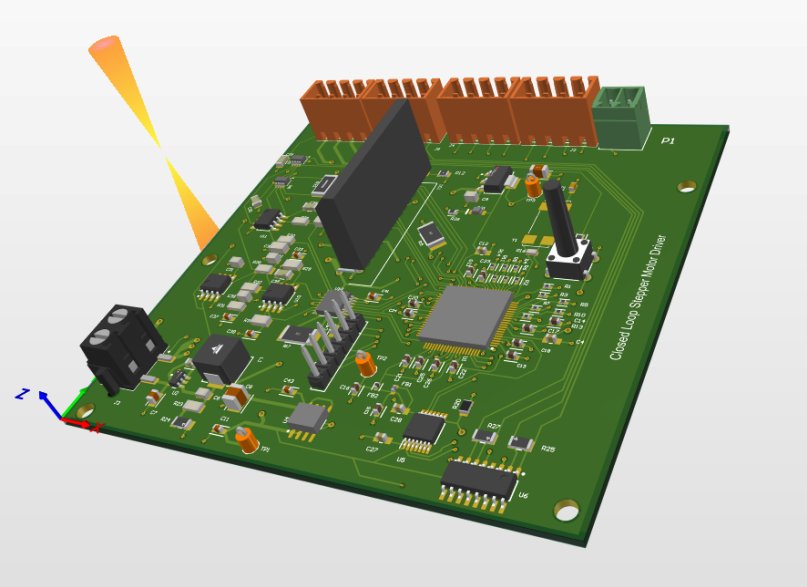


Figure 60: 3D View of the PCB

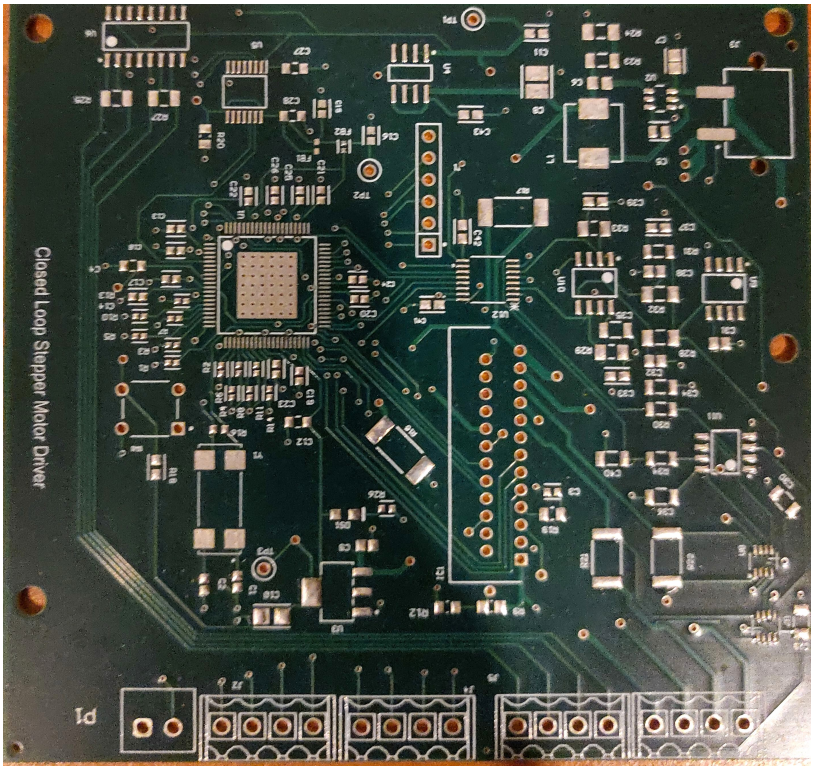


Figure 61: Actual view of PCB.

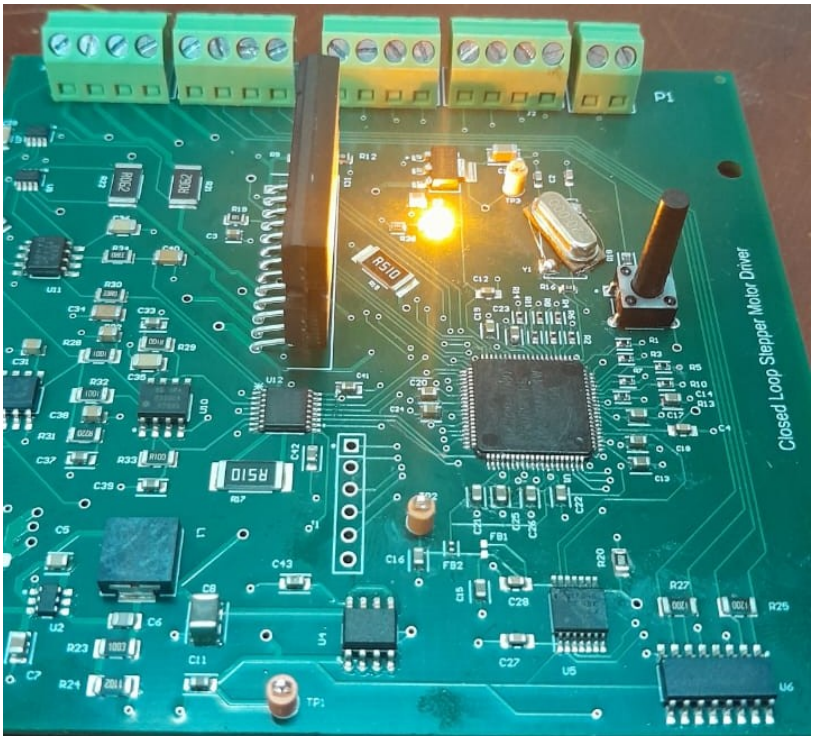


Figure 62: Actual view of PCB after mounting components.

30.1 Coding and Algorithm design

For coding, we used Code Compiler Studio IDE which comes with a very useful library collection.I t comes with device specific libraries, example codes and many other resources.

We used several source files and header files with a clear structure so debugging is convenient.

When using **Header** files we created the following Header files to support our product.

- current_sensing.h
- motor_constants.h
- motor_control.h
- motor_variables.h
- pi_controllers.h
- pwm_control.h
- system_init.h
- transforms.h
- utilities.h

The following **Source** files were used.

- current_sensing.c
- encoder_interface.c
- interrupts.h

- pi_controllers.h
- pwm_control.c
- system_init.c
- transforms.c
- utilities.c

30.1.1 Full code implementation

current_sensing.h

```
#ifndef CURRENT_SENSING_H
#define CURRENT_SENSING_H

// Function prototypes
void read_currents(void);

#endif // CURRENT_SENSING_H


}
```

motor_constants.h

```
#ifndef MOTOR_CONSTANTS_H
#define MOTOR_CONSTANTS_H

// Mathematical constants
#define M_PI 3.14159265358979323846

// Motor parameters
#define POLE_PAIRS 50           // NEMA 17: 200 steps/rev / 4 phases = 50 pole pairs
#define VDC 15.0                // DC bus voltage (V)
#define PWM_PERIOD 4500         // 10 kHz PWM (90 MHz / (2 * 4500))
#define CURRENT_MAX 2.0         // Max current (A)
#define SPEED_MAX 1000.0        // Max speed (RPM)
#define ENCODER_COUNTS_PER_REV (200.0 * POLE_PAIRS) // 10000 counts/rev
#define Kt 0.2                  // Torque constant (Nm/A)



// PI controller gains
#define KP_D 0.5                // D-axis current proportional gain
#define KI_D 0.01               // D-axis current integral gain
#define KP_Q 0.5                // Q-axis current proportional gain
#define KI_Q 0.01               // Q-axis current integral gain
#define KP_SPEED 0.1            // Speed proportional gain
#define KI_SPEED 0.001          // Speed integral gain
#define KP_POS 0.5              // Position proportional gain
#define KI_POS 0.01             // Position integral gain
#define KP_TORQUE 0.5           // Torque proportional gain
#define KI_TORQUE 0.01          // Torque integral gain

// Control modes
```

```
#define MODE_POSITION 0         // Control mode: Position
#define MODE_SPEED 1            // Control mode: Speed
#define MODE_TORQUE 2           // Control mode: Torque

#endif // MOTOR_CONSTANTS_H

}
```

motor_control.h

```
#ifndef MOTOR_CONTROL_H
#define MOTOR_CONTROL_H

#include "F2806x_Device.h"
#include "F2806x_Examples.h"
#include "F2806x_I2c.h"
#include "IQmathLib.h"
#include "F2806x_EPwm.h"
#include "F2806x_EQep.h"
#include "F2806x_Gpio.h"
#include "F2806x_Adc.h"
#include <stdint.h>

// Include all module headers
#include "motor_constants.h"
#include "motor_variables.h"
#include "system_init.h"
#include "current_sensing.h"
#include "encoder_interface.h"
#include "pwm_control.h"
#include "transforms.h"
#include "pi_controllers.h"
#include "interrupts.h"
#include "utilities.h"

#endif // MOTOR_CONTROL_H
}
```

motor_variables.h

```
#ifndef MOTOR_VARIABLES_H
#define MOTOR_VARIABLES_H

#include "IQmathLib.h"

// External variable declarations
extern _iq Id_ref;              // D-axis current reference
extern _iq Iq_ref;              // Q-axis current reference
extern _iq speed_ref;           // Speed reference (RPM)
extern _iq pos_ref;             // Position reference (radians)
extern _iq torque_ref;          // Torque reference (Nm)
extern _iq theta_e;             // Electrical angle
extern _iq pos;                 // Measured position
extern _iq speed;               // Measured speed
extern _iq Ia, Ib;              // Phase currents
```

```
extern _iq Ialpha, Ibeta;       // Clarke transform outputs
extern _iq Id, Iq;              // Park transform outputs
extern _iq Vd, Vq;              // Voltage outputs
extern _iq Valpha, Vbeta;       // Inverse Park transform outputs
extern Uint32 encoder_pos;      // Encoder position
extern Uint16 led_state;        // LED state
extern Uint32 loop_count;       // Loop counter for LED
extern int control_mode;        // Control mode
extern Uint16 i2c_buffer[2];    // I2C buffer for INA3221 reads

#endif // MOTOR_VARIABLES_H

}
```

pi_controllers.h

```
#ifndef PI_CONTROLLERS_H
#define PI_CONTROLLERS_H

// Function prototypes
void pi_current_control(void);
void pi_speed_control(void);
void pi_position_control(void);
void pi_torque_control(void);

#endif // PI_CONTROLLERS_H

}
```

pwm_control.h

```
#ifndef PWM_CONTROL_H
#define PWM_CONTROL_H

// Function prototypes
void svgen_pwm(void);

#endif // PWM_CONTROL_H
}
```

system_init.h

```
#ifndef SYSTEM_INIT_H
#define SYSTEM_INIT_H

// Function prototypes
void init_system(void);
void init_adc(void)
void init_eqep(void);
void init_pwm(void);
void init_pwm_deadband(void);

#endif
}
```

transforms.h

```
#ifndef TRANSFORMS_H
#define TRANSFORMS_H

// Function prototypes
void clarke_transform(void);
void park_transform(void);
void inv_park_transform(void);

#endif // TRANSFORMS_H

}
```

utilities.h

```
#ifndef UTILITIES_H
#define UTILITIES_H

#include "IQmathLib.h"

// Function prototypes
void toggle_led(void);
_iq _IQsgn(_iq value);

#endif // UTILITIES_H
}
```

current_sensing.c

```
#include "motor_control.h"

// Read phase currents
void read_currents(void) {
    Ia = _IQmpy(_IQ(AdcResult.ADCRESULT0 - ADC_OFFSET), _IQ(ADC_GAIN));
    Ib = _IQmpy(_IQ(AdcResult.ADCRESULT1 - ADC_OFFSET), _IQ(ADC_GAIN));
}


}
```

encoder_interface.c

```
#include "motor_control.h"

extern Uint32 last_pos; // Defined in motor_variables.c

void read_encoder(void) {
    _iq delta_pos;
    encoder_pos = EQep1Regs.QPOSCNT;
    pos = _IQmpy(_IQ(encoder_pos), _IQ(2.0 * M_PI / ENCODER_COUNTS_PER_REV));
    theta_e = _IQmpy(_IQ(encoder_pos), _IQ(2.0 * M_PI / (200.0 * POLE_PAIRS)));
    delta_pos = _IQ(encoder_pos - last_pos);
    speed = _IQmpy(delta_pos, _IQ(60.0 / (0.0001 * 200.0 * POLE_PAIRS)));
    last_pos = encoder_pos;
}

}
```

interrupts.c

```
#include "motor_control.h"

__interrupt void timer0_isr(void) {
    read_currents();
    read_encoder();
    clarke_transform();
    park_transform();
    pi_current_control();
    inv_park_transform();
    svgen_pwm();

    switch (control_mode) {
        case MODE_POSITION:
            pi_position_control();
            break;
        case MODE_SPEED:
            pi_speed_control();
            break;
        case MODE_TORQUE:
            pi_torque_control();
            break;
    }

    CpuTimer0Regs.TCR.bit.TIF = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

}
```

pi_controllers.c

```
#include "motor_control.h"

void pi_current_control(void) {
    static _iq Id_error_int = _IQ(0.0), Iq_error_int = _IQ(0.0);
    _iq Id_error, Iq_error, Vmax;

    Id_error = Id_ref - Id;
    Iq_error = Iq_ref - Iq;
    Vmax = _IQ(VDC * 0.5);

    Id_error_int += _IQmpy(KI_D, Id_error);
    if (_IQabs(Id_error_int) > Vmax) Id_error_int = _IQmpy(Vmax, _IQsgn(Id_error_int));
    Iq_error_int += _IQmpy(KI_Q, Iq_error);
    if (_IQabs(Iq_error_int) > Vmax) Iq_error_int = _IQmpy(Vmax, _IQsgn(Iq_error_int));

    Vd = _IQmpy(KP_D, Id_error) + Id_error_int;
    Vq = _IQmpy(KP_Q, Iq_error) + Iq_error_int;

    if (_IQabs(Vd) > Vmax) Vd = _IQmpy(Vmax, _IQsgn(Vd));
    if (_IQabs(Vq) > Vmax) Vq = _IQmpy(Vmax, _IQsgn(Vq));
}
```

```
void pi_position_control(void) {
    static _iq pos_error_int = _IQ(0.0);
    _iq pos_error;

    pos_error = pos_ref - pos;
    pos_error_int += _IQmpy(KI_POS, pos_error);
    if (_IQabs(pos_error_int) > _IQ(CURRENT_MAX)) {
        pos_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(pos_error_int));
    }

    Iq_ref = _IQmpy(KP_POS, pos_error) + pos_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}

void pi_speed_control(void) {
    static _iq speed_error_int = _IQ(0.0);
    _iq speed_error;

    speed_error = speed_ref - speed;
    speed_error_int += _IQmpy(KI_SPEED, speed_error);
    if (_IQabs(speed_error_int) > _IQ(CURRENT_MAX)) {
        speed_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(speed_error_int));
    }

    Iq_ref = _IQmpy(KP_SPEED, speed_error) + speed_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}

void pi_torque_control(void) {
    static _iq torque_error_int = _IQ(0.0);
    _iq torque_error, torque_actual;

    torque_actual = _IQmpy(_IQ(Kt), Iq);
    torque_error = torque_ref - torque_actual;
    torque_error_int += _IQmpy(KI_TORQUE, torque_error);
    if (_IQabs(torque_error_int) > _IQ(CURRENT_MAX)) {
        torque_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(torque_error_int));
    }

    Iq_ref = _IQmpy(KP_TORQUE, torque_error) + torque_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}
```

```
}
```

pwm_control.c

```
void svgen_pwm(void) {
    _iq Va, Vb;
    Uint16 cmpa, cmpb;
    float temp;

    Va = Valpha;
    Vb = _IQmpy(_IQ(-0.5), Valpha) + _IQmpy(_IQ(0.86602540378), Vbeta);

    temp = _IQtoF(_IQmpy(_IQ(Va / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpa = (Uint16)temp;

    temp = _IQtoF(_IQmpy(_IQ(Vb / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpb = (Uint16)temp;

    EPwm1Regs.CMPA.half.CMPA = cmpa;
    EPwm2Regs.CMPA.half.CMPA = cmpb;
}

}
```

system_init.c

```
#include "motor_control.h"

// System initialization
void init_system(void) {
    InitSysCtrl(); // Initialize system clock (90 MHz)
    DINT; // Disable interrupts

    InitPieCtrl();
    IER = 0x0000; // Clear interrupt enable register
    IFR = 0x0000; // Clear interrupt flag register

    InitPieVectTable();
    EALLOW;
    PieVectTable.TINT0 = &timer0_isr; // Map Timer0 interrupt
    EDIS;

    // Configure LED (GPIO34)
    EALLOW;
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0; // GPIO function
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1; // Output
    GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // LED off
    EDIS;

    // Enable Timer0 interrupt
```

```
    IER |= M_INT1;
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1; // Timer0 interrupt
}


// ADC initialization
void init_adc(void) {
    InitAdc(); // Initialize ADC module
    EALLOW; // Enable protected register access

    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1; // Interrupt pulse at end of conversion
    AdcRegs.ADCCTL1.bit.ADCBGPWD = 1;    // Power up band gap
    AdcRegs.ADCCTL1.bit.ADCPWDN = 1;     // Power up ADC

    AdcRegs.ADCSOC0CTL.bit.CHSEL = 0;    // SOC0: ADCINA0 (Ia)
    AdcRegs.ADCSOC1CTL.bit.CHSEL = 1;    // SOC1: ADCINA1 (Ib)
    AdcRegs.ADCSOC0CTL.bit.ACQPS = 6;    // Acquisition window: 6 cycles
    AdcRegs.ADCSOC1CTL.bit.ACQPS = 6;    // Acquisition window: 6 cycles
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 5;  // Trigger: ePWM1 SOCA
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 5;  // Trigger: ePWM1 SOCA

    AdcRegs.ADCINTSOCSEL1.bit.SOC1 = 1;  // SOC1 triggers ADCINT1
    AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;// Clear ADCINT1 flag
    AdcRegs.INTSEL1N2.bit.INT1E = 1;     // Enable ADCINT1
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;   // Enable PIE interrupt for ADCINT1

    EDIS; // Disable protected register access

    AdcRegs.ADCSOCFRC1.all = 0x0003;     // Force SOC0 and SOC1 to start
}

// eQEP initialization
void init_eqep(void) {
    InitEQep1Gpio();
    EALLOW;
    EQep1Regs.QEPCTL.bit.QPEN = 1; // Enable eQEP
    EQep1Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
    EQep1Regs.QPOSMAX = 200 * POLE_PAIRS - 1; // Max position count
    EQep1Regs.QPOSINIT = 0; // Initial position
    EQep1Regs.QEPCTL.bit.PCRM = 0; // Position reset on index
    EQep1Regs.QEINT.bit.PCE = 1; // Position counter error interrupt
    EQep1Regs.QCLR.bit.PCE = 1; // Clear position error
    EDIS;
}

// PWM initialization
void init_pwm(void) {
    InitEPwm1Gpio();
    InitEPwm2Gpio();
    EALLOW;
    EPwm1Regs.TBCTL.bit.CTRMODE = 3; // Up-down count
    EPwm1Regs.TBPRD = PWM_PERIOD; // 10 kHz PWM
    EPwm1Regs.CMPA.half.CMPA = 0; // Initial compare value
    EPwm1Regs.AQCTLA.bit.CAU = 2; // Set on CMPA up
```

```
    EPwm1Regs.AQCTLA.bit.CAD = 1; // Clear on CMPA down
    EPwm1Regs.TBCTL.bit.PHSEN = 0; // Disable phaseCube
    EPwm2Regs.TBCTL.bit.CTRMODE = 3;
    EPwm2Regs.TBPRD = PWM_PERIOD;
    EPwm2Regs.CMPA.half.CMPA = 0;
    EPwm2Regs.AQCTLA.bit.CAU = 2;
    EPwm2Regs.AQCTLA.bit.CAD = 1;
    EPwm2Regs.TBCTL.bit.PHSEN = 0;

    EPwm1Regs.ETSEL.bit.SOCAEN = 1; // Enable SOCA
    EPwm1Regs.ETSEL.bit.SOCASEL = 1; // Trigger on period match
    EPwm1Regs.ETPS.bit.SOCAPRD = 1; // Trigger every period
    EDIS;

    InitCpuTimers();
    ConfigCpuTimer(&CpuTimer0, 90, 100); // 90 MHz, 100 us period (10 kHz)
    CpuTimer0Regs.TCR.bit.TSS = 0; // Start timer
}

//dead time initialization
void init_pwm_deadband(void) {
    EALLOW; // Enable protected register access

    // Configure Dead-Band for EPWM1 (EPWM1A and EPWM1B)
    EPwm1Regs.DBCTL.bit.OUT_MODE = 3; // Enable Dead-Band for both RED and FED
    EPwm1Regs.DBCTL.bit.POLSEL = 2;   // Active high complementary (AHC) mode
    EPwm1Regs.DBCTL.bit.IN_MODE = 0;  // EPWMxA is source for both RED and FED
    EPwm1Regs.DBRED.bit.DBRED = 8;    // Rising Edge Delay: ~80 ns (8 cycles at 90 MHz)
    EPwm1Regs.DBFED.bit.DBFED = 8;    // Falling Edge Delay: ~80 ns (8 cycles at 90 MHz)

    // Configure Dead-Band for EPWM2 (EPWM2A and EPWM2B)
    EPwm2Regs.DBCTL.bit.OUT_MODE = 3; // Enable Dead-Band for both RED and FED
    EPwm2Regs.DBCTL.bit.POLSEL = 2;   // Active high complementary (AHC) mode
    EPwm2Regs.DBCTL.bit.IN_MODE = 0;  // EPWMxA is source for both RED and FED
    EPwm2Regs.DBRED.bit.DBRED = 8;    // Rising Edge Delay: ~80 ns (8 cycles at 90 MHz)
    EPwm2Regs.DBFED.bit.DBFED = 8;    // Falling Edge Delay: ~80 ns (8 cycles at 90 MHz)

    EDIS; // Disable protected register access
}

}
```

transforms.c

```
#include "motor_control.h"

void clarke_transform(void) {
    Ialpha = Ia;
    Ibeta = _IQmpy(_IQ(0.57735026919), Ia) + _IQmpy(_IQ(1.15470053838), Ib);
}

void park_transform(void) {
    _iq cos_theta = _IQcos(theta_e);
    _iq sin_theta = _IQsin(theta_e);
    Id = _IQmpy(Ialpha, cos_theta) + _IQmpy(Ibeta, sin_theta);
```

```
    Iq = _IQmpy(Ibeta, cos_theta) - _IQmpy(Ialpha, sin_theta);
}

void inv_park_transform(void) {
    _iq cos_theta = _IQcos(theta_e);
    _iq sin_theta = _IQsin(theta_e);
    Valpha = _IQmpy(Vd, cos_theta) - _IQmpy(Vq, sin_theta);
    Vbeta = _IQmpy(Vd, sin_theta) + _IQmpy(Vq, cos_theta);
}
}
```

utilities.c

```
#include "motor_control.h"

void toggle_led(void) {
    led_state = !led_state;
    GpioDataRegs.GPBSET.bit.GPIO34 = led_state;
    GpioDataRegs.GPBCLEAR.bit.GPIO34 = !led_state;
}

_iq _IQsgn(_iq value) {
    if (_IQtoF(value) > 0) return _IQ(1.0);
    if (_IQtoF(value) < 0) return _IQ(-1.0);
    return _IQ(0.0);
}

}
```

30.1.2 Breadboard Implementation Code

```
#include "F2806x_Device.h"
#include "F2806x_Examples.h"
#include "F2806x_I2c.h"
#include "IQmathLib.h"
#include <stdint.h>

// Define constants
#define M_PI 3.14159265358979323846 // Accurate position measurement
#define POLE_PAIRS 50           // NEMA 17: 200 steps/rev / 4 phases = 50 pole pairs
#define VDC 12.0                // DC bus voltage (V)
#define PWM_PERIOD 4500         // 10 kHz PWM (90 MHz / (2 * 4500))
#define CURRENT_MAX 2.0         // Max current (A)
#define SPEED_MAX 1000.0        // Max speed (RPM)
#define ADC_OFFSET 0            // No offset for INA3221
#define ADC_GAIN 1.0            // Adjust for INA3221 scaling
#define ENCODER_COUNTS_PER_REV (200.0 * POLE_PAIRS) // 10000 counts/rev
#define Kt 0.2                  // Torque constant (Nm/A)
#define INA3221_ADDR 0x40       // INA3221 I2C address (A0 = GND)
#define INA3221_CONFIG_REG 0x00 // INA3221 configuration register
#define INA3221_CURRENT_REG_CH1 0x01 // Channel 1 current (Ia)
#define INA3221_CURRENT_REG_CH2 0x03 // Channel 2 current (Ib)
#define SHUNT_RESISTANCE 0.05   // 0.05 ohm shunt (verify module specs)
#define INA3221_CURRENT_LSB (40e-6 / SHUNT_RESISTANCE) // 0.8 mA/LSB for 0.05 ohm
```

```
// PI controller gains
#define KP_D 0.5                // D-axis current proportional gain
#define KI_D 0.01               // D-axis current integral gain
#define KP_Q 0.5                // Q-axis current proportional gain
#define KI_Q 0.01               // Q-axis current integral gain
#define KP_SPEED 0.1            // Speed proportional gain
#define KI_SPEED 0.001          // Speed integral gain
#define KP_POS 0.5              // Position proportional gain
#define KI_POS 0.01             // Position integral gain
#define KP_TORQUE 0.5           // Torque proportional gain
#define KI_TORQUE 0.01          // Torque integral gain
#define MODE_POSITION 0         // Control mode: Position
#define MODE_SPEED 1            // Control mode: Speed
#define MODE_TORQUE 2           // Control mode: Torque

// Global variables
_iq Id_ref = _IQ(0.0);          // D-axis current reference
_iq Iq_ref = _IQ(0.0);          // Q-axis current reference
_iq speed_ref = _IQ(100.0);     // Speed reference (RPM)
_iq pos_ref = _IQ(0.0);         // Position reference (radians)
_iq torque_ref = _IQ(0.0);      // Torque reference (Nm)
_iq theta_e = _IQ(0.0);         // Electrical angle
_iq pos = _IQ(0.0);             // Measured position
_iq speed = _IQ(0.0);           // Measured speed
_iq Ia, Ib;                     // Phase currents
_iq Ialpha, Ibeta;              // Clarke transform outputs
_iq Id, Iq;                     // Park transform outputs
_iq Vd, Vq;                     // Voltage outputs
_iq Valpha, Vbeta;              // Inverse Park transform outputs
Uint32 encoder_pos;             // Encoder position
static Uint32 last_pos = 0;     // Previous encoder position
Uint16 led_state = 0;           // LED state
Uint32 loop_count = 0;          // Loop counter for LED
int control_mode = MODE_SPEED;  // Control mode (default: speed)
Uint16 i2c_buffer[2];           // I2C buffer for INA3221 reads

// Function prototypes
void init_system(void);
void init_i2c(void);
void init_ina3221(void);
void init_eqep(void);
void init_pwm(void);
void read_currents(void);
void read_encoder(void);
void clarke_transform(void);
void park_transform(void);
void pi_current_control(void);
void pi_speed_control(void);
void pi_position_control(void);
void pi_torque_control(void);
void inv_park_transform(void);
void svgen_pwm(void);
void toggle_led(void);
_iq _IQsgn(_iq value);
```

```
// Interrupt service routine
__interrupt void timer0_isr(void);

// Main function
void main(void) {
    init_system();
    init_i2c();
    init_ina3221();
    init_eqep();
    init_pwm();

    // Enable interrupts
    EINT; // Global interrupts
    ERTM; // Real-time mode

    while (1) {
        // Blink LED every 500ms
        if (loop_count++ >= 450000) { // 450000 / 90 MHz = 500ms
            toggle_led();
            loop_count = 0;
        }
    }
}

// System initialization
void init_system(void) {
    InitSysCtrl(); // Initialize system clock (90 MHz)
    DINT; // Disable interrupts

    InitPieCtrl();
    IER = 0x0000; // Clear interrupt enable register
    IFR = 0x0000; // Clear interrupt flag register

    InitPieVectTable();
    EALLOW;
    PieVectTable.TINT0 = &timer0_isr; // Map Timer0 interrupt
    EDIS;

    // Configure LED (GPIO34)
    EALLOW;
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0; // GPIO function
    GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1; // Output
    GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // LED off
    EDIS;

    // Enable Timer0 interrupt
    IER |= M_INT1;
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1; // Timer0 interrupt
}

// I2C initialization (J5: GPIO32/SDAA, GPIO33/SCLA)
void init_i2c(void) {
    EALLOW;
```

```
    SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 1; // Enable I2C-A clock
    GpioCtrlRegs.GPBPUD.bit.GPIO32 = 0; // Enable pull-up for SDA
    GpioCtrlRegs.GPBPUD.bit.GPIO33 = 0; // Enable pull-up for SCL
    GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 2; // GPIO32 = SDAA
    GpioCtrlRegs.GPBMUX1.bit.GPIO33 = 2; // GPIO33 = SCLA
    I2caRegs.I2CPSC.all = 8; // Prescaler: 90 MHz / (8+1) = 10 MHz
    I2caRegs.I2CCLKL = 12; // Low time for 400 kHz
    I2caRegs.I2CCLKH = 12; // High time for 400 kHz
    I2caRegs.I2CIER.all = 0x00; // Disable I2C interrupts
    I2caRegs.I2CMDR.all = 0x0020; // Master, free-run
    I2caRegs.I2CFFTX.all = 0x6000; // Enable TX FIFO
    I2caRegs.I2CFFRX.all = 0x2040; // Enable RX FIFO
    EDIS;
}

// Configure INA3221 module
void init_ina3221(void) {
    EALLOW;
    I2caRegs.I2CSAR = INA3221_ADDR; // Set I2C slave address
    I2caRegs.I2CCNT = 3; // 3 bytes: reg + 2-byte config
    I2caRegs.I2CDXR = INA3221_CONFIG_REG; // Config register
    I2caRegs.I2CDXR = 0x71; // Enable CH1 & CH2, continuous mode
    I2caRegs.I2CDXR = 0x27; // 588 us conversion, 4-sample avg
    I2caRegs.I2CMDR.all = 0x0620; // Master, TX, start
    while (I2caRegs.I2CSTR.bit.BB == 1); // Wait for bus free
    while (I2caRegs.I2CSTR.bit.XRDY == 0); // Wait for TX complete
    EDIS;
}

// eQEP initialization
void init_eqep(void) {
    InitEQep1Gpio();
    EALLOW;
    EQep1Regs.QEPCTL.bit.QPEN = 1; // Enable eQEP
    EQep1Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
    EQep1Regs.QPOSMAX = 200 * POLE_PAIRS - 1; // Max position count
    EQep1Regs.QPOSINIT = 0; // Initial position
    EQep1Regs.QEPCTL.bit.PCRM = 0; // Position reset on index
    EQep1Regs.QEINT.bit.PCE = 1; // Position counter error interrupt
    EQep1Regs.QCLR.bit.PCE = 1; // Clear position error
    EDIS;
}

// PWM initialization
void init_pwm(void) {
    InitEPwm1Gpio();
    InitEPwm2Gpio();
    EALLOW;
    EPwm1Regs.TBCTL.bit.CTRMODE = 3; // Up-down count
    EPwm1Regs.TBPRD = PWM_PERIOD; // 10 kHz PWM
    EPwm1Regs.CMPA.half.CMPA = 0; // Initial compare value
    EPwm1Regs.AQCTLA.bit.CAU = 2; // Set on CMPA up
    EPwm1Regs.AQCTLA.bit.CAD = 1; // Clear on CMPA down
    EPwm1Regs.TBCTL.bit.PHSEN = 0; // Disable phase sync
```

```
    EPwm2Regs.TBCTL.bit.CTRMODE = 3;
    EPwm2Regs.TBPRD = PWM_PERIOD;
    EPwm2Regs.CMPA.half.CMPA = 0;
    EPwm2Regs.AQCTLA.bit.CAU = 2;
    EPwm2Regs.AQCTLA.bit.CAD = 1;
    EPwm2Regs.TBCTL.bit.PHSEN = 0;

    EPwm1Regs.ETSEL.bit.SOCAEN = 1; // Enable SOCA
    EPwm1Regs.ETSEL.bit.SOCASEL = 1; // Trigger on period match
    EPwm1Regs.ETPS.bit.SOCAPRD = 1; // Trigger every period
    EDIS;

    InitCpuTimers();
    ConfigCpuTimer(&CpuTimer0, 90, 100); // 90 MHz, 100 us period (10 kHz)
    CpuTimer0Regs.TCR.bit.TSS = 0; // Start timer
}

// Read phase currents from INA3221 module
void read_currents(void) {
    int16_t raw_current;
    float current;

    // Read Channel 1 (Ia)
    EALLOW;
    I2caRegs.I2CSAR = INA3221_ADDR;
    I2caRegs.I2CCNT = 1; // Write register address
    I2caRegs.I2CDXR = INA3221_CURRENT_REG_CH1;
    I2caRegs.I2CMDR.all = 0x0620; // Master, TX, start
    while (I2caRegs.I2CSTR.bit.BB == 1); // Wait for bus free
    while (I2caRegs.I2CSTR.bit.XRDY == 0); // Wait for TX complete

    I2caRegs.I2CCNT = 2; // Read 2 bytes
    I2caRegs.I2CMDR.all = 0x0420; // Master, RX, start
    while (I2caRegs.I2CSTR.bit.RRDY == 0); // Wait for RX data
    i2c_buffer[0] = I2caRegs.I2CDRR; // MSB
    while (I2caRegs.I2CSTR.bit.RRDY == 0);
    i2c_buffer[1] = I2caRegs.I2CDRR; // LSB
    raw_current = (i2c_buffer[0] << 8) | i2c_buffer[1];
    current = (float)raw_current * INA3221_CURRENT_LSB;
    Ia = _IQmpy(_IQ(current), _IQ(ADC_GAIN));

    // Read Channel 2 (Ib)
    I2caRegs.I2CCNT = 1;
    I2caRegs.I2CDXR = INA3221_CURRENT_REG_CH2;
    I2caRegs.I2CMDR.all = 0x0620;
    while (I2caRegs.I2CSTR.bit.BB == 1);
    while (I2caRegs.I2CSTR.bit.XRDY == 0);

    I2caRegs.I2CCNT = 2;
    I2caRegs.I2CMDR.all = 0x0420;
    while (I2caRegs.I2CSTR.bit.RRDY == 0);
    i2c_buffer[0] = I2caRegs.I2CDRR;
    while (I2caRegs.I2CSTR.bit.RRDY == 0);
```

```
    i2c_buffer[1] = I2caRegs.I2CDRR;
    raw_current = (i2c_buffer[0] << 8) | i2c_buffer[1];
    current = (float)raw_current * INA3221_CURRENT_LSB;
    Ib = _IQmpy(_IQ(current), _IQ(ADC_GAIN));
    EDIS;
}

// Read encoder and compute speed
void read_encoder(void) {
    _iq delta_pos;
    encoder_pos = EQep1Regs.QPOSCNT;
    pos = _IQmpy(_IQ(encoder_pos), _IQ(2.0 * M_PI / ENCODER_COUNTS_PER_REV));
    theta_e = _IQmpy(_IQ(encoder_pos), _IQ(2.0 * M_PI / (200.0 * POLE_PAIRS)));
    delta_pos = _IQ(encoder_pos - last_pos);
    speed = _IQmpy(delta_pos, _IQ(60.0 / (0.0001 * 200.0 * POLE_PAIRS)));
    last_pos = encoder_pos;
}

// Clarke transform
void clarke_transform(void) {
    Ialpha = Ia;
    Ibeta = _IQmpy(_IQ(0.57735026919), Ia) + _IQmpy(_IQ(1.15470053838), Ib);
}

// Park transform
void park_transform(void) {
    _iq cos_theta = _IQcos(theta_e);
    _iq sin_theta = _IQsin(theta_e);
    Id = _IQmpy(Ialpha, cos_theta) + _IQmpy(Ibeta, sin_theta);
    Iq = _IQmpy(Ibeta, cos_theta) - _IQmpy(Ialpha, sin_theta);
}

// PI control for currents
void pi_current_control(void) {
    static _iq Id_error_int = _IQ(0.0), Iq_error_int = _IQ(0.0);
    _iq Id_error, Iq_error, Vmax;

    Id_error = Id_ref - Id;
    Iq_error = Iq_ref - Iq;
    Vmax = _IQ(VDC * 0.5);

    Id_error_int += _IQmpy(KI_D, Id_error);
    if (_IQabs(Id_error_int) > Vmax) Id_error_int = _IQmpy(Vmax, _IQsgn(Id_error_int));
    Iq_error_int += _IQmpy(KI_Q, Iq_error);
    if (_IQabs(Iq_error_int) > Vmax) Iq_error_int = _IQmpy(Vmax, _IQsgn(Iq_error_int));

    Vd = _IQmpy(KP_D, Id_error) + Id_error_int;
    Vq = _IQmpy(KP_Q, Iq_error) + Iq_error_int;

    if (_IQabs(Vd) > Vmax) Vd = _IQmpy(Vmax, _IQsgn(Vd));
    if (_IQabs(Vq) > Vmax) Vq = _IQmpy(Vmax, _IQsgn(Vq));
}

// PI control for position
```

```
void pi_position_control(void) {
    static _iq pos_error_int = _IQ(0.0);
    _iq pos_error;

    pos_error = pos_ref - pos;
    pos_error_int += _IQmpy(KI_POS, pos_error);
    if (_IQabs(pos_error_int) > _IQ(CURRENT_MAX)) {
        pos_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(pos_error_int));
    }

    Iq_ref = _IQmpy(KP_POS, pos_error) + pos_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}

// PI control for speed
void pi_speed_control(void) {
    static _iq speed_error_int = _IQ(0.0);
    _iq speed_error;

    speed_error = speed_ref - speed;
    speed_error_int += _IQmpy(KI_SPEED, speed_error);
    if (_IQabs(speed_error_int) > _IQ(CURRENT_MAX)) {
        speed_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(speed_error_int));
    }

    Iq_ref = _IQmpy(KP_SPEED, speed_error) + speed_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
    }
}

// PI control for torque
void pi_torque_control(void) {
    static _iq torque_error_int = _IQ(0.0);
    _iq torque_error, torque_actual;

    torque_actual = _IQmpy(_IQ(Kt), Iq);
    torque_error = torque_ref - torque_actual;
    torque_error_int += _IQmpy(KI_TORQUE, torque_error);
    if (_IQabs(torque_error_int) > _IQ(CURRENT_MAX)) {
        torque_error_int = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(torque_error_int));
    }

    Iq_ref = _IQmpy(KP_TORQUE, torque_error) + torque_error_int;
    Id_ref = _IQ(0.0);

    if (_IQabs(Iq_ref) > _IQ(CURRENT_MAX)) {
        Iq_ref = _IQmpy(_IQ(CURRENT_MAX), _IQsgn(Iq_ref));
```

```
    }
}

// Inverse Park transform
void inv_park_transform(void) {
    _iq cos_theta = _IQcos(theta_e);
    _iq sin_theta = _IQsin(theta_e);
    Valpha = _IQmpy(Vd, cos_theta) - _IQmpy(Vq, sin_theta);
    Vbeta = _IQmpy(Vd, sin_theta) + _IQmpy(Vq, cos_theta);
}

// Space vector PWM
void svgen_pwm(void) {
    _iq Va, Vb;
    Uint16 cmpa, cmpb;
    float temp;

    Va = Valpha;
    Vb = _IQmpy(_IQ(-0.5), Valpha) + _IQmpy(_IQ(0.86602540378), Vbeta);

    temp = _IQtoF(_IQmpy(_IQ(Va / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpa = (Uint16)temp;

    temp = _IQtoF(_IQmpy(_IQ(Vb / VDC), _IQ(PWM_PERIOD / 2)) + _IQ(PWM_PERIOD / 2));
    if (temp > PWM_PERIOD) temp = PWM_PERIOD;
    else if (temp < 0) temp = 0;
    cmpb = (Uint16)temp;

    EPwm1Regs.CMPA.half.CMPA = cmpa;
    EPwm2Regs.CMPA.half.CMPA = cmpb;
}

// Toggle LED
void toggle_led(void) {
    led_state = !led_state;
    GpioDataRegs.GPBSET.bit.GPIO34 = led_state;
    GpioDataRegs.GPBCLEAR.bit.GPIO34 = !led_state;
}

// Custom IQ sign function
_iq _IQsgn(_iq value) {
    if (_IQtoF(value) > 0) return _IQ(1.0);
    if (_IQtoF(value) < 0) return _IQ(-1.0);
    return _IQ(0.0);
}

// Timer ISR (10 kHz)
__interrupt void timer0_isr(void) {
    read_currents();
    read_encoder();
    clarke_transform();
    park_transform();
```

```
    pi_current_control();
    inv_park_transform();
    svgen_pwm();

    switch (control_mode) {
        case MODE_POSITION:
            pi_position_control();
            break;
        case MODE_SPEED:
            pi_speed_control();
            break;
        case MODE_TORQUE:
            pi_torque_control();
            break;
    }

    CpuTimer0Regs.TCR.bit.TIF = 1; // Clear timer interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge PIE group 1
}
```

30.2 Enclosure design

A custom enclosure was developed to house the stepper motor driver circuit, providing protection, structural support, and user accessibility. The enclosure enhances system robustness during testing and operation while improving safety and portability.

- **Material:** The enclosure was constructed using acrylic sheets or 3D-printed ABS/PLA plastic for the prototype phase. For industrial deployment, metal enclosures are recommended for improved thermal performance and electromagnetic shielding.
- **Dimensions and Fit:** The internal layout was designed to securely fit the PCB using mounting standoffs, with precisely cut openings for connectors, cables, and test points.
- **Ventilation:** Air slots were added near the motor driver section to aid passive cooling and prevent overheating during high-load operation.
- **Cable Routing:** Openings were positioned for motor wires, power input, and encoder connections. Cable glands or rubber grommets were included to protect cables from wear.
- **User Interface Access:** Cutouts were added for USB/UART ports, indicator LEDs, and optional reset or power switches, making the system more user-friendly during debugging.
- **Modularity and Maintenance:** The enclosure is assembled using screws, allowing easy removal of the top cover for internal access, inspection, or replacement of components.
- **Safety:** The enclosure prevents accidental contact with high-voltage terminals or heated components, ensuring safe use in educational or lab environments.

The enclosure design ensures the entire system is portable, safe, and mechanically stable during operation, while still allowing access for firmware updates and system expansion.

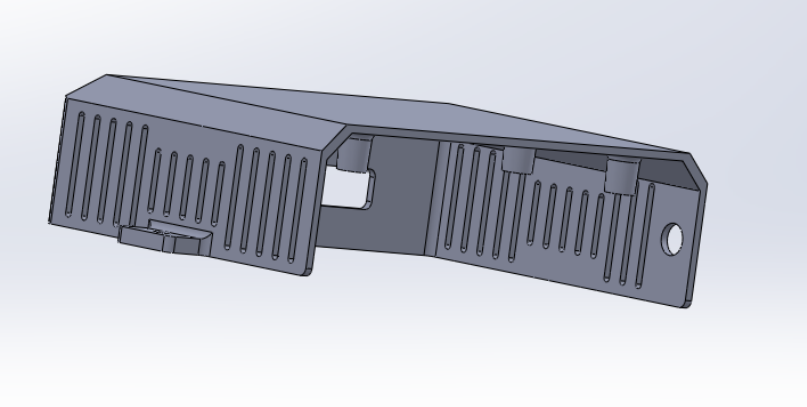


Figure 63: Top Part of the designe)

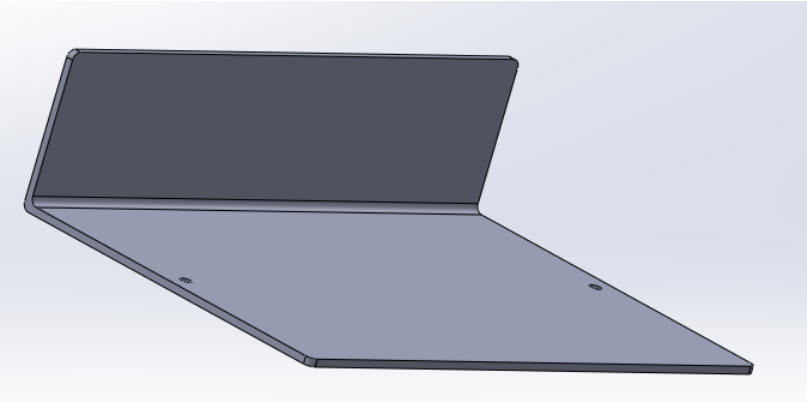


Figure 64: Bottom Part of the design

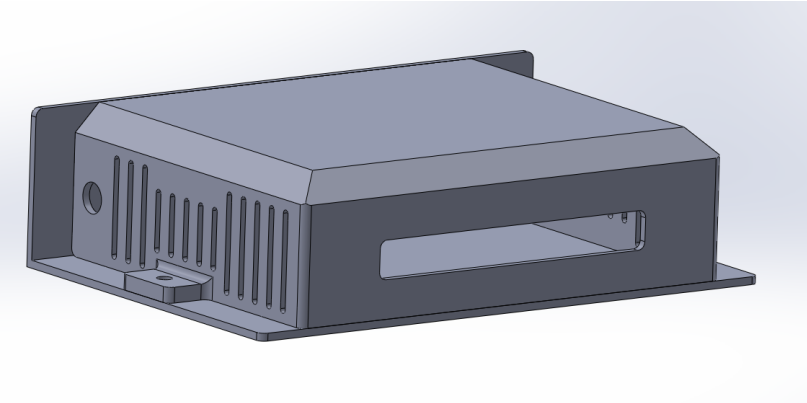


Figure 65: Assembly of the design

Figure 66: Final design top view

Figure 67: Final design bottom view

Figure 68: Final design side view

30.3 Testing Phase

After receiving the manufactured PCB, the following steps were carried out to ensure the board met the necessary design and quality requirements before assembly:

- Inspect the PCB visually for broken traces, solder mask misalignment, poor silkscreen, and damaged vias.
- Measure the board dimensions, via positions, and hole diameters and compare with the design.
- Confirm that all solder mask and silkscreen placements match the intended component layout.
- Use a multimeter to check continuity of key signal paths and ensure no shorts exist, especially between power and ground.
- Dry-fit components to verify footprint accuracy.

30.4 Final budget report

The following table outlines the detailed budget for the closed-loop stepper motor driver project, including costs for components, prototyping, and testing equipment.

Item Description	Quantity	Unit Price (LKR)	Total (LKR)
TI C2000 LaunchPad (F28069M)	1	14500	14500
XDS110 Debugger	1	12100	12100
NEMA 17 Stepper Motor	1	7100	7100
Lead Schine Stepper Motor Driver	1	17700	17700
TB67H400AFTGEL Motor Driver IC	3	1930	5790
TMS320F28069PFPS MCU	3	5600	16800
ADS7854IPW	3	5425	16275
INA241B2IDDFR	4	895	3580
ECS-2520MVQ-200-BS-TR	3	325	975
PCB Screen print			9800
Prototype cost			3300
Custom PCB Fabrication			54700
Power Supply (12V/3A)	1	1650	1650
3D Printing			1730
Metal Works			1200
Printing cost			3000
Schipping cost			23000
Tax and ware house rent paid			42000
Total Estimated Cost			**235200**

Table 6: Detailed project budget for the closed-loop stepper motor driver system

30.4.1 Cost for a single closed loop stepper motor driver

Item Description	Quantity	Unit Price (LKR)	Total (LKR)
TB67H400AFTGEL Motor Driver IC	1	1930	1930
TMS320F28069PFPS MCU	1	5600	5600
ADS7854IPW	1	5425	5425
INA241B2IDDFR	2	895	1790
ECS-2520MVQ-200-BS-TR	1	325	325
PCB and other components			8000
Total Estimated Cost			**23070**

Table 7: Detailed project budget for the closed-loop stepper motor driver system