Department of Electronic and Telecommunication Engineering

University of Moratuwa Faculty of Engineering

Electronic Design Realization EN2160

Pick and Place Robot Arm

Design Documentation

Group members:

210625H – SURENDRA SAJE
210668P – VIDMAL HVP
210490L – PRABODHA KPKA
210418C – NAYANTHRA JNP

Table of Contents

1. General

a) Marketing Need

We recognize a significant demand for an automated system capable of precise placement and rotation of metal rings in H-Bridge assemblies. This need arises from the increasing market pressure for high-quality, reliable motor control components that can be produced at scale. The system promises to reduce production costs and enhance product reliability, providing a competitive edge in the motor control market.

b) Detailed analysis of Existing Products

1. **ABB H-Bridge Systems**:
    - **Model**: ABB-HBS100
    - **Features**: High precision placement, automated assembly line integration, advanced error detection.
    - **Specifications**: ±0.01 mm placement accuracy, throughput of 500 units per hour.
2. **Siemens Automated Assembly**:
    - **Model**: SAA-HB-500
    - **Features**: Robust vision systems, flexible component handling, scalable modular design.
    - **Specifications**: 0.02-degree rotation accuracy, compatible with various H-Bridge configurations.

3. **AlphaTech Motor Control Assembly**:

    - **Model**: AT-MCA300
    - **Features**: Manual adjustment options, semi-automated system, integration with local supply chains.
    - **Specifications**: ±0.03 mm placement accuracy, manual inspection required.

c) User Identification

The primary users of the automated assembly system are manufacturers of motor control systems and electronic components, including companies that produce H-Bridges for automotive, industrial, and consumer electronics applications. These users prioritize precision, efficiency, and scalability in their production processes. User exceptions of a automated assembly product is give below,

- **Increased Production Speed and Throughput:** By automating assembly tasks, we anticipate a significant reduction in assembly time, thereby boosting production volumes and scalability.
- **Enhanced Accuracy and Consistency:** Utilizing robotics ensures precise handling and placement of components, minimizing errors and variations in product quality.
- **Reduced Labor Costs:** Automation eliminates the need for manual labor in repetitive tasks, leading to cost savings and improved operational efficiency.
- **Improved Worker Safety:** Automating hazardous or repetitive tasks reduces the risk of injuries, enhancing workplace safety and employee well-being

d) Market Segment

The market segment for this project includes:

- **Automotive Industry**: Manufacturers of electric and hybrid vehicles.
- **Industrial Automation**: Companies producing machinery and robotics that require precise motor control.

e) Final Specifications

Robotic arm manipulation:

The robotic arm, equipped with a precision gripper, is designed for the delicate handling of metal rings of various sizes. Its advanced motion control algorithms enable smooth, precise movements, ensuring efficient and accurate assembly operations while minimizing mechanical wear and tear.

Control system:

The system uses microcontroller units for centralized management of assembly operations. The intuitive user interface simplifies programming and monitoring, while data logging capabilities provide insights for quality control and process optimization, enhancing overall system performance.

Control Capabilities:

The device offers comprehensive control over connected components, allowing movement in four directions left, right, up, and down through user-defined commands. This flexibility in motion control ensures precise handling of the metal rings, facilitating their accurate placement in the assembly process.

Automatic Functionality:

Equipped with an advanced automation mode, the system can autonomously execute predefined tasks with precision and efficiency. This functionality ensures seamless operation, reduces the need for manual intervention, and maintains high standards of consistency and quality in the production process.

Safety Features:

The system includes robust safety features such as emergency stop buttons and protective enclosures, ensuring the safety of operators. It complies with industry safety standards, offering a secure operating environment and reducing the risk of workplace accidents and injuries.

2. Electronic Design

Electronic design is the art of establishing cutting-edge solutions through the integration of electronic components. It involves the careful selection, configuration, and optimization of components to achieve desired performance and functionality. From microcontrollers to actuators, each component plays an important part in shaping the final product's functions and accuracy. Through careful research and execution, electronic designers transform abstract ideas into actual realities, driving progress and innovation in various industries and applications.

a) Conceptual Block Diagram – Selected Design

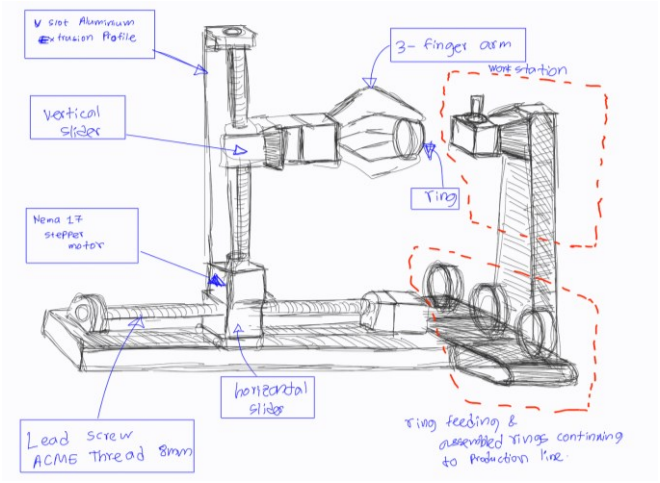


Figure 1 – Sketch Diagram

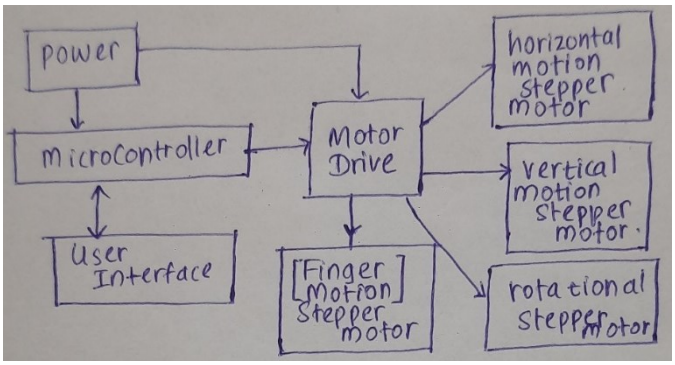


Figure 2 – Block Diagram

b) Component Selection

In the section of electronic design, component selection stands as the cornerstone of a successful project. Each component chosen plays a major role in shaping the functionality, efficiency, and reliability of the final product. In this stage, we delve into the selecting key components for our project, with the focus on stepper motors, microcontrollers, and motor drivers. We compare the chosen components with available alternatives to justify our selections.

Electronic Component	**Selected Model**
Motors as actuators	Stepper Motor – Nema 17
Microcontroller	ATmega328P
Motor Driver	A4988

01. Stepper Motors – Nema 17

Declaration:

In our findings and calculations, we evaluated both **open-loop and closed-loop** stepper motor control systems for their application in controlling sliders and velocities. Closed-loop systems, offering feedback control, emerged as the superior choice due to their ability to dynamically adjust for accurate positioning and speed control. However, due to budgetary constraints, we opted for open-loop stepper motors, specifically selecting the NEMA 17 model. The open-loop system, despite lacking feedback, was deemed feasible for our application through detailed mathematical analysis and feasibility studies, which confirmed its adequacy in meeting the required performance specifications within the cost limitations.

Specifications and Considerations:

The NEMA 17 stepper motor was selected based on several key specifications and considerations:

1. **Shaft Load:**
    - The motor's ability to withstand a shaft load of 20,000 hours at 1000 RPM axial 25 N (5.6 lbs.), push 65 N (15 lbs.), and pull radial 29 N (6.5 lbs.) at the flat center aligns well with the demands of our project. This capability ensures smooth and reliable operation, even under load.
2. **Step Angle and Resolution:**

- With a step angle of 1.8 degrees, the NEMA 17 motor provides 200 steps per revolution. This high level of resolution allows for precise control over movement, crucial for achieving the desired positioning accuracy.

3. **Voltage and Torque:**
    - The motor is rated for a voltage of 12 volts, although it can be operated at lower voltages. While lower voltages may result in reduced torque, the 12V rating ensures sufficient torque output for our application's requirements.
4. **Applications and Versatility:**
    - NEMA 17 stepper motors are commonly used in CNC machines, linear actuators, and various precision control applications. Their widespread usage and compatibility with diverse systems attest to their reliability and versatility.

Figure 3 - Nema 17 Stepper Motor

Alternative Stepper Motor Options:

While the NEMA 17 stepper motor was ultimately chosen for its suitability in our project, several alternative options were considered and evaluated. Each of these alternatives offers unique characteristics and specifications that may be suitable for different applications. However, after careful analysis, they were deemed less appropriate for our specific project requirements. Here are some alternative NEMA stepper motor models and the reasons for their rejection:

1. **NEMA 14 Stepper Motor:**
    - This smaller-sized stepper motor offers a compact form factor, making it ideal for applications where space is limited. However, its lower torque output and reduced shaft load capacity compared to the NEMA 17 make it less suitable for our project, which demands higher torque and load-bearing capabilities.
2. **NEMA 23 Stepper Motor:**

- With a larger frame size and higher torque output than the NEMA 17, the NEMA 23 stepper motor could potentially provide greater power and performance. However, its increased size and weight may not be compatible with the space constraints of our project. Additionally, the higher voltage requirement of NEMA 23 motors may introduce complexities in our system design, as we aim to operate at lower voltages for efficiency and compatibility.

3. **NEMA 34 Stepper Motor:**
    - Offering even higher torque output and robust construction, the NEMA 34 stepper motor is suitable for heavy-duty applications requiring substantial power. However, its larger size and increased power consumption make it unsuitable for our project, which prioritizes compactness, energy efficiency, and precise control over motion.
4. **NEMA 11 Stepper Motor:**
    - This smaller-sized stepper motor is similar to the NEMA 14 in terms of compactness but may offer slightly higher torque output. However, its diminutive size also limits its load-bearing capacity and overall performance compared to the NEMA 17. For our project, which requires a balance between size, torque, and precision, the NEMA 11 was not considered optimal.

For further detailed information on stepper motor specifications and support, please refer to the following datasheet links:

- NEMA 17 Stepper Motor Data Sheet
- NEMA 14 Stepper Motor Data Sheet
- NEMA 23 Stepper Motor Data Sheet
- NEMA 34 Stepper Motor Data Sheet
- NEMA 11 Stepper Motor Data Sheet

02. Microcontroller – ATMEGA328P

Reasons for Selection:

The choice of the ATMEGA328P microcontroller was the result of a comprehensive evaluation of its features and suitability for our project's specific needs, particularly given our requirement for register-level programming. Several key factors influenced this decision:

1. **Direct Register-Level Programming Capability**: The ATMEGA328P allows for low-level access and manipulation of hardware registers, which provides us with

precise control over the microcontroller's operations without relying on high-level libraries. This capability is crucial for fine-tuning performance and optimizing our application.

2. **Ample I/O Pins**: With 23 General-Purpose Input/Output (GPIO) pins, including several that support Pulse Width Modulation (PWM), the ATMEGA328P offers sufficient flexibility to control multiple stepper motors directly. It also provides room for connecting additional sensors or peripherals, catering to the project's expanding requirements.
3. **Versatile Operating Voltage Range**: Operating from 1.8V to 5.5V, the ATMEGA328P is adaptable to various power supply scenarios typically found in embedded systems. This versatility simplifies integration with diverse hardware setups and aids in robust power management.
4. **Adequate Processing Power**: Featuring an 8-bit AVR architecture and operating at a clock speed of 16MHz, the ATMEGA328P delivers adequate processing capabilities for handling the tasks required in our project. Its balance between computational efficiency and power consumption makes it well-suited for embedded applications that demand reliable performance without excessive power draw.

Comparison with Alternatives:

- **STM32F103C8T6**: While this microcontroller provides superior processing power with a 32-bit ARM Cortex-M3 core, it necessitates more complex development efforts due to its architecture and the requirement to program without high-level libraries, making it less straightforward compared to the ATMEGA328P for our project's needs.
- **PIC16F877A**: Although this microcontroller offers more processing power with its 32-bit ARM Cortex-M3 core, it requires more complex development and register-level programming, which can be time-consuming and challenging compared to the ATMEGA328P.

For further detailed information, please refer to the following datasheet links:

- ATMEGA328P Datasheet
- PIC16F877A Datasheet
- STM32F103C8T6 Datasheet

03. Motor Driver – A4988

Declaration:

All the detailed calculations and feasibility analyses supporting the selection of the A4988 motor driver, including considerations for current, and voltage requirements, are thoroughly documented in the calculation section.

Reasons for Selection: The A4988 motor driver was selected based on several key factors that align with our project's requirements:

1. **Wide Operating Voltage Range:** With an operating voltage range of 8V to 35V, the A4988 accommodates various power supply configurations commonly encountered in stepper motor control applications, ensuring compatibility and versatility.
2. **Suitable Current Handling Capability:** The A4988 offers a maximum continuous current of 1A per phase and a peak current of 2A, providing sufficient current handling capability for driving stepper motors in our project while maintaining reliable performance.
3. **Simplicity and Cost-effectiveness:** Known for its simplicity and cost-effectiveness, the A4988 motor driver offers a straightforward implementation and represents an economical solution without compromising on essential features and functionality.

Specifications:

- Operating Voltage Range: 8V to 35V
- Maximum Continuous Current: 1A per phase
- Peak Current: 2A

Applications: The A4988 motor driver finds widespread use in various applications, including:

- Stepper motor control in CNC machines and 3D printers
- Robotics projects requiring precise control over motor movements.

Comparison with Alternatives:

- **DRV8825:** While offering a higher voltage range (8.2V to 45V) and peak current (2.5A), the DRV8825 might be over-specification for our project's requirements, potentially leading to higher costs without significant benefits.
- **TB67S249-FTG:** Despite providing a higher peak current (4.5A), the TB67S249-FTG requires a higher voltage range (10V to 47V), which might not be necessary for our application, making it less suitable in terms of cost and complexity.

For further detailed information, please refer to the following datasheet links:

- A4988 Datasheet
- DRV8825 Datasheet
- TB67S249-FTG Datasheet

04. Additional Components

Emergency Stop Switch

In our project for automated metal ring placement and rotation, the inclusion of an Emergency Stop (E-stop) switch is critical to ensure safety and operational control. The E-stop switch serves as a fail-safe mechanism that allows operators to immediately halt the system in case of emergencies or unforeseen situations. Here's how we integrated and utilized the Emergency Stop switch in our system:

Purpose and Functionality

The primary purpose of the Emergency Stop switch is to provide a quick and effective means to shut down the entire system in case of emergencies such as equipment malfunction, safety hazards, or any other critical incidents. When activated, the E-stop immediately cuts power to all motors, controllers, and other active components, bringing the system to a safe and controlled stop.

1. **Placement**: The Emergency Stop switch is strategically located within easy reach of the operator, ensuring quick access in emergency situations. It is prominently positioned on the control panel or within the vicinity of the operating area.
2. **Wiring and Configuration**: The switch is wired in series with the main power supply or motor control circuitry. This configuration ensures that activating the E-stop interrupts the flow of electrical power to critical components, effectively halting all motion and operations.

Safety Considerations

- **Immediate Action**: The design ensures that pressing the Emergency Stop switch results in an immediate and complete cessation of all automated movements and operations.
- **Reset Mechanism**: After activation, the E-stop switch usually requires manual reset or acknowledgment to restore power and resume operations. This prevents accidental reactivation and ensures deliberate control over system restart.

c) Circuit Design

Designing electronic circuits is essential to make components work. Every circuit is expertly designed to convert electrical signals into mechanical actions. Every component, including motor driver shields, power management, and microcontroller schematics, is carefully designed to ensure smooth operation.

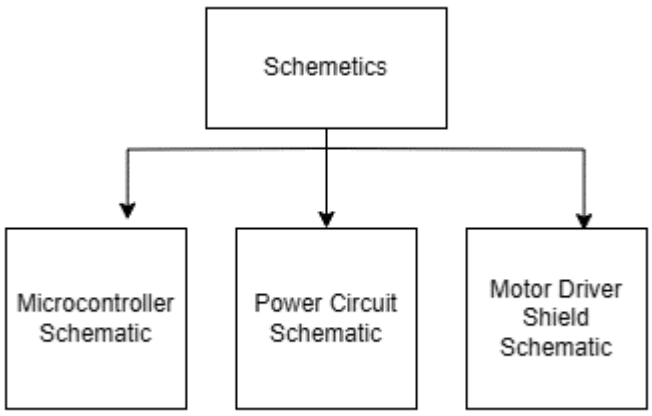
Circuit Testing

During the testing phase, the following procedures were implemented to verify the functionality and reliability of the circuits:

1. **Power Circuit Verification**: The power circuit was initially constructed, and power was introduced to it. Voltages at various points within the circuit were measured to confirm proper operation and stability.
2. **Microcontroller Circuit Testing**: After the power circuit was validated, the microcontroller section of the circuit was connected. Voltages at different nodes within the microcontroller circuit were checked to ensure correct functioning.
3. **Motor Driver, Relays, and Button JSTs Testing**: Following this, the motor drivers, relays, and button JSTs were integrated into the circuit. Voltage testing was carried out to verify that these components were receiving the correct voltage levels and operating as expected.

01. Schematics Design

In the schematics design phase, each circuit component is meticulously mapped out to ensure proper functionality and integration. This involves drafting detailed diagrams that illustrate the connections between various electronic elements, such as resistors, capacitors, transistors, and integrated circuits.

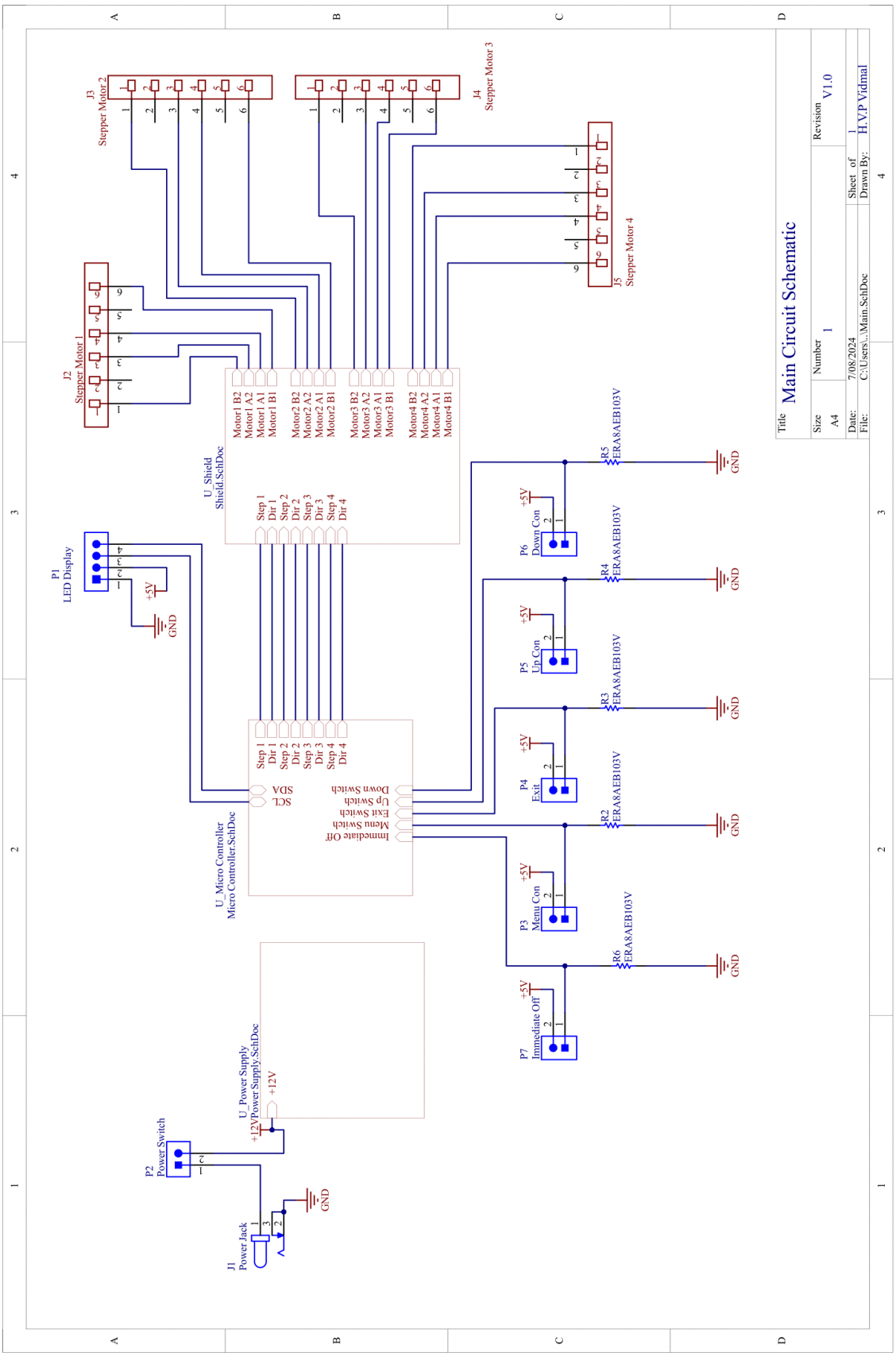


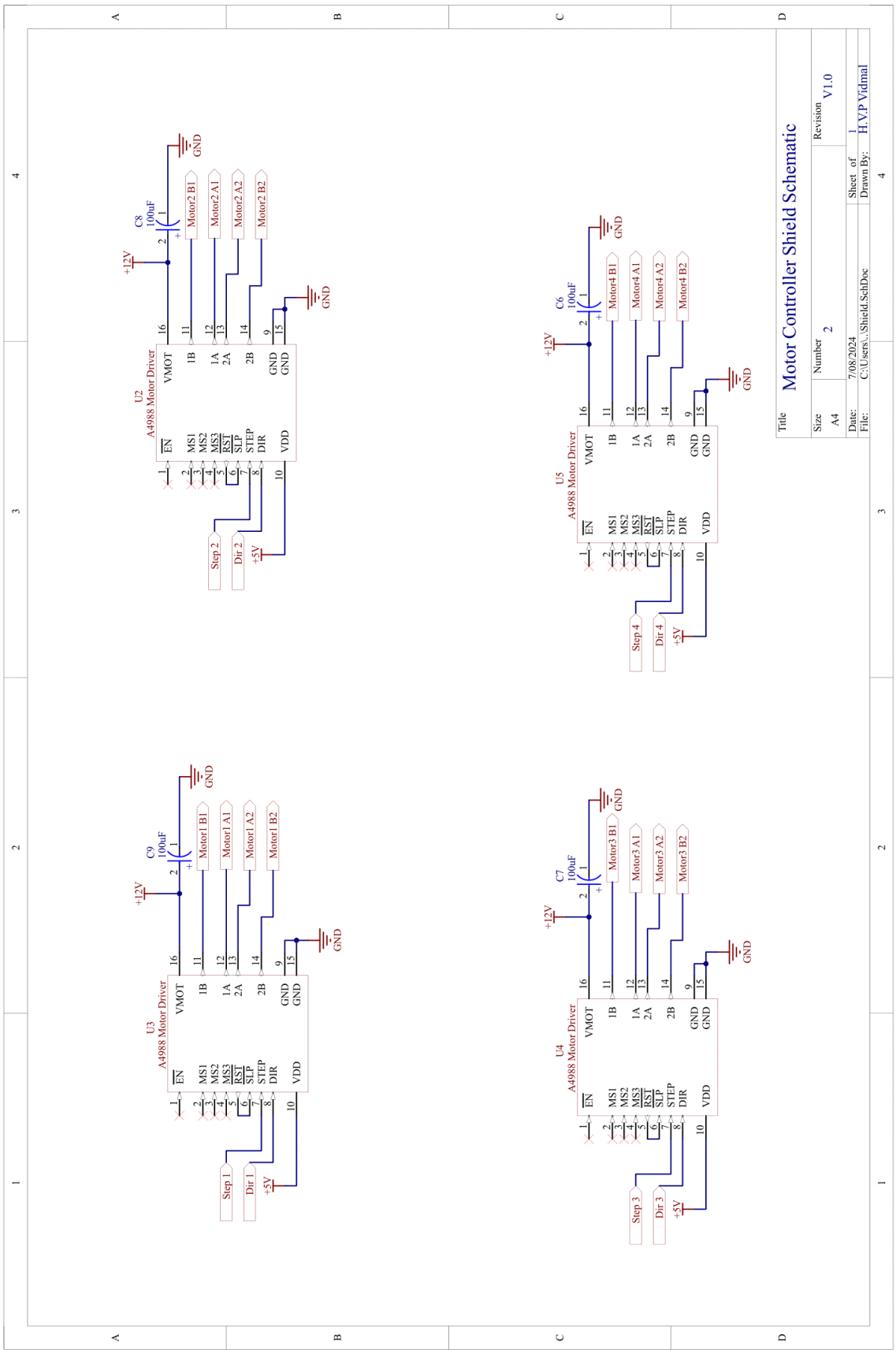
The layout must be clear and concise, with labels and annotations providing clarity on component values, pin configurations, and signal pathways. Attention to detail is paramount to avoid errors and ensure optimal performance. Additionally, considerations for power distribution, signal integrity, and thermal management are carefully incorporated into the schematics to guarantee reliable operation under varying conditions.

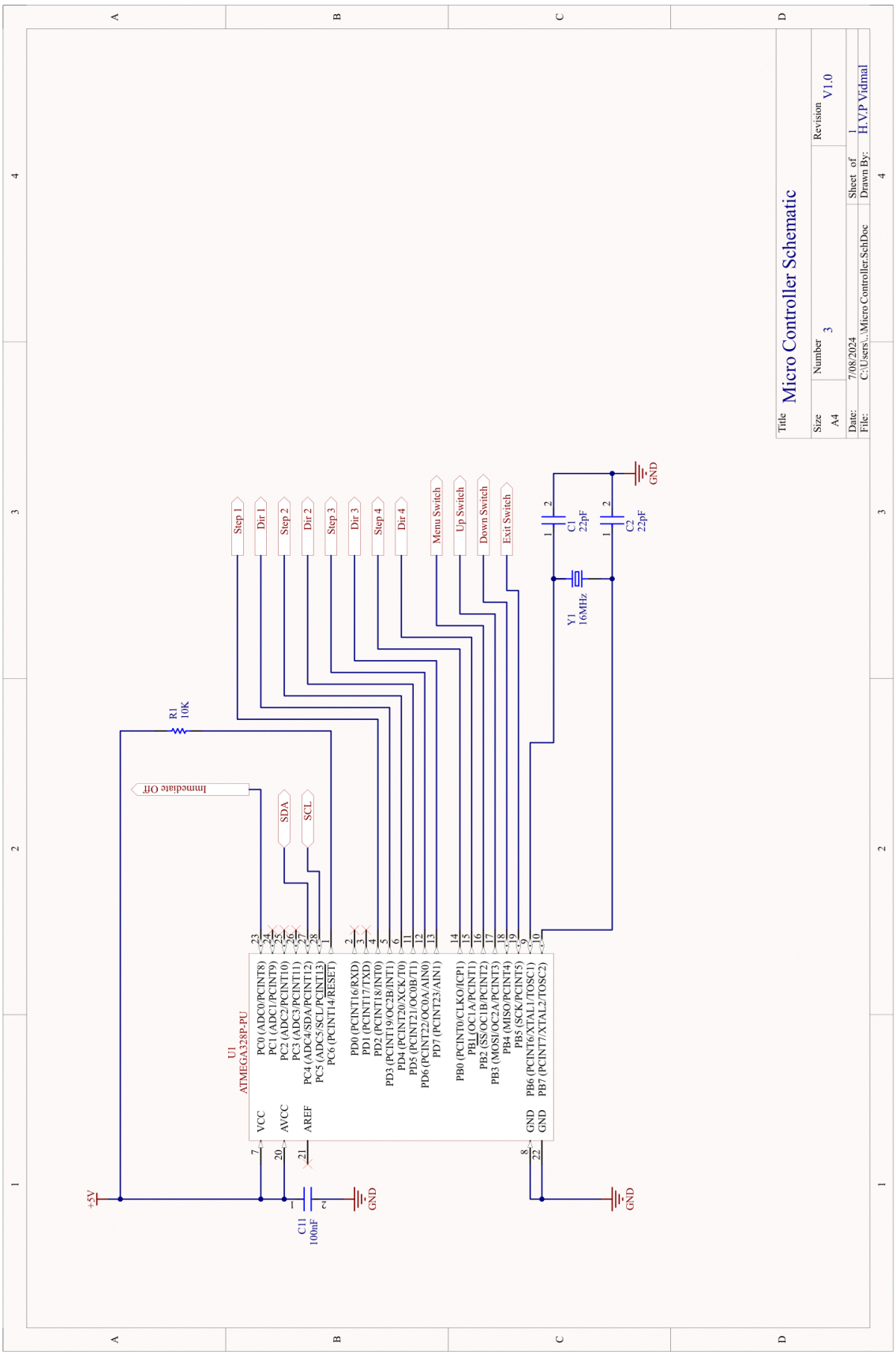
1. Microcontroller Schematic (ATmega328P): The microcontroller schematic incorporates the ATmega328P MCU along with support components such as crystal oscillator, decoupling capacitors, and programming header. It provides sufficient GPIO pins for motor control and communication interfaces such as SPI, and I2C.

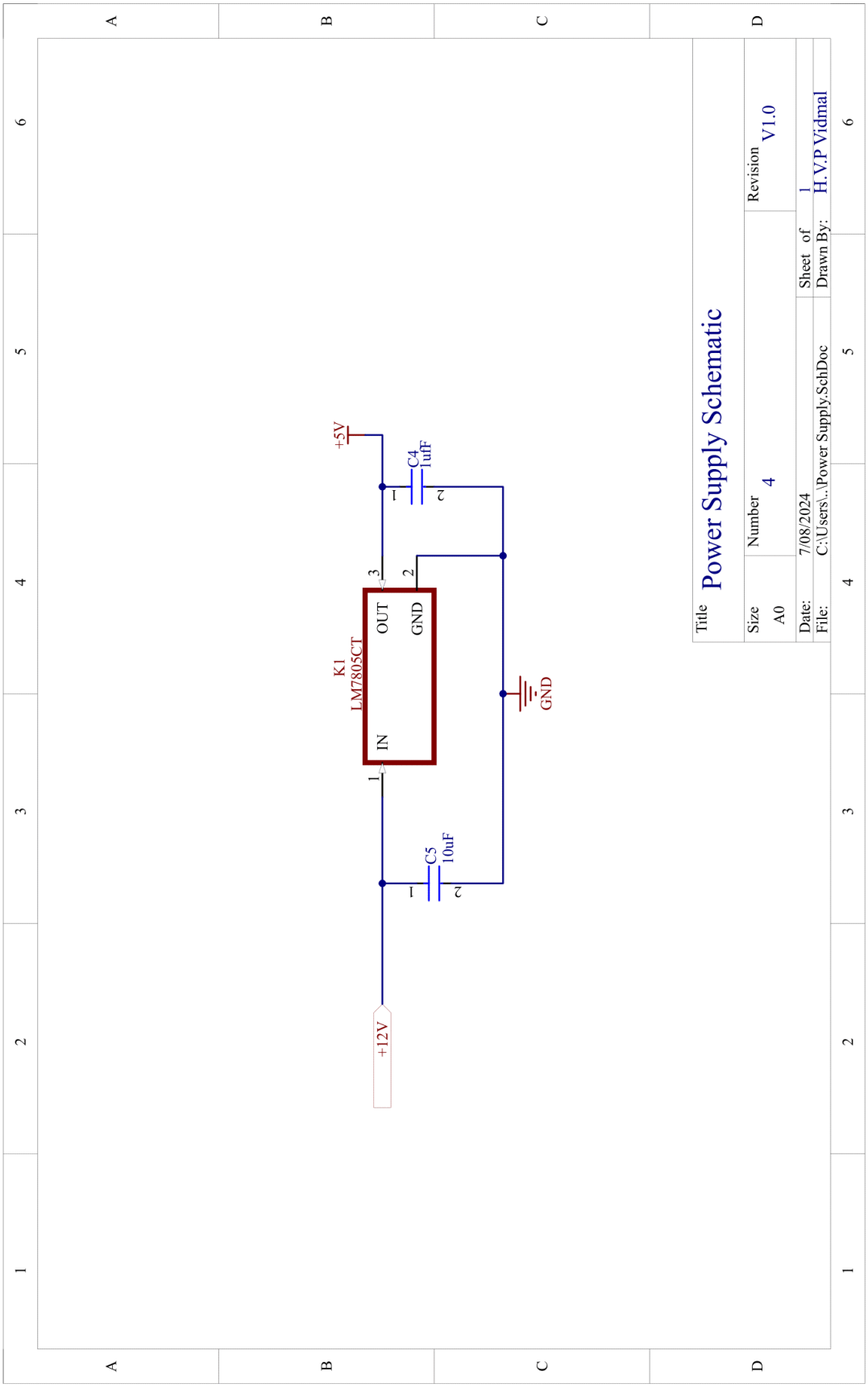
2. Power Schematic (LM7805): The power schematic features the LM7805 linear voltage regulator to provide a stable 5V supply for the MCU and other components on the PCB. It includes input and output capacitors for filtering, as well as protection diodes for input voltage reverse polarity protection.

3. Motor Driver Shield Schematic (A4988 * 4): The motor driver shield schematic integrates four A4988 stepper motor driver ICs modules, each capable of driving one stepper motor. It includes protection diodes, current sensing resistors, and other necessary components for motor control. The schematic ensures efficient and precise control of stepper motors with micro stepping capability.


Main Circuit Schematic
Title
Size
A4
Number
1
Revision
V1.0
Date:
7/08/2024
Sheet of
1
File:
C:\Users\...\Main.SchDoc
Drawn By:
H.V.P Vidmal
J1
Power Jack
P2
Power Switch
U_Power Supply
Power Supply.SchDoc
+12V
GND
P1
LED Display
+5V
J2
Stepper Motor 1
J3
Stepper Motor 2
J4
Stepper Motor 3
J5
Stepper Motor 4
U_Shield
Shield.SchDoc
Motor1 B2
Motor1 A2
Motor1 A1
Motor1 B1
Motor2 B2
Motor2 A2
Motor2 A1
Motor2 B1
Motor3 B2
Motor3 A2
Motor3 A1
Motor3 B1
Motor4 B2
Motor4 A2
Motor4 A1
Motor4 B1
Step 1
Dir 1
Step 2
Dir 2
Step 3
Dir 3
Step 4
Dir 4
U_Micro Controller
Micro Controller.SchDoc
SDA
SCL
Down Switch
Up Switch
Exit Switch
Menu Switch
Immediate Off
P3
Menu Con
P4
Exit
P5
Up Con
P6
Down Con
P7
Immediate Off
R2
ERA8AEB103V
R3
ERA8AEB103V
R4
ERA8AEB103V
R5
ERA8AEB103V
R6
ERA8AEB103V


Motor Controller Shield Schematic
Title
Size
A4
Number
2
Revision
V1.0
Date:
7/08/2024
Sheet of
1
File:
C:\Users\...\Shield.SchDoc
Drawn By:
H.Y.P Vidmal
U2
A4988 Motor Driver
U3
A4988 Motor Driver
U4
A4988 Motor Driver
U5
A4988 Motor Driver
C8
100uF
C9
100uF
C7
100uF
C6
100uF
+12V
GND
+5V
Step 1
Dir 1
Step 2
Dir 2
Step 3
Dir 3
Step 4
Dir 4
Motor1 B1
Motor1 A1
Motor1 A2
Motor1 B2
Motor2 B1
Motor2 A1
Motor2 A2
Motor2 B2
Motor3 B1
Motor3 A1
Motor3 A2
Motor3 B2
Motor4 B1
Motor4 A1
Motor4 A2
Motor4 B2


Micro Controller Schematic
Title
Size
A4
Number
3
Revision
V1.0
Date:
7/08/2024
Sheet of
1
File:
C:\Users\...\Micro Controller.SchDoc
Drawn By:
H.Y.P Vidmal
U1
ATMEGA328P-PU
PC0 (ADC0/PCINT8)
PC1 (ADC1/PCINT9)
PC2 (ADC2/PCINT10)
PC3 (ADC3/PCINT11)
PC4 (ADC4/SDA/PCINT12)
PC5 (ADC5/SCL/PCINT13)
PC6 (PCINT14/RESET)
PD0 (PCINT16/RXD)
PD1 (PCINT17/TXD)
PD2 (PCINT18/INT0)
PD3 (PCINT19/OC2B/INT1)
PD4 (PCINT20/XCK/T0)
PD5 (PCINT21/OC0B/T1)
PD6 (PCINT22/OC0A/AIN0)
PD7 (PCINT23/AIN1)
PB0 (PCINT0/CLKO/ICP1)
PB1 (OC1A/PCINT1)
PB2 (SS/OC1B/PCINT2)
PB3 (MOSI/OC2A/PCINT3)
PB4 (MISO/PCINT4)
PB5 (SCK/PCINT5)
PB6 (PCINT6/XTAL1/TOSC1)
PB7 (PCINT7/XTAL2/TOSC2)
VCC
AVCC
AREF
GND
GND
Step 1
Dir 1
Step 2
Dir 2
Step 3
Dir 3
Step 4
Dir 4
Menu Switch
Up Switch
Down Switch
Exit Switch
Immediate Off
SDA
SCL
R1
10K
C11
100nF
+5V
GND
Y1
16MHz
C1
22pF
C2
22pF
GND
GND


Power Supply Schematic
Title
Size
A0
Number
4
Revision
V1.0
Date:
7/08/2024
Sheet of
1
File:
C:\Users\...\Power Supply.SchDoc
Drawn By:
H.V.P Vidmal
+5V
C4
1uFF
K1
LM7805CT
OUT
GND
IN
GND
C5
10uF
+12V

Component List for Circuit Design

1. ATmega328P Microcontroller

- **Model**: ATmega328P-PU
- **Description**: An 8-bit AVR microcontroller with 23 GPIO pins, 32KB flash memory, and 2KB SRAM. Ideal for embedded applications requiring precise control.

2. LM7805 Voltage Regulator

- **Model**: LM7805CT
- **Description**: A linear voltage regulator that outputs a stable 5V from an input voltage between 7V and 35V. Provides up to 1.5A of current.

3. A4988 Stepper Motor Driver module (4x)

- **Model**: A4988 module
- **Description**: A micro stepping driver for controlling bipolar stepper motors, capable of up to 1/16th micro stepping. Features adjustable current limiting and thermal protection.

4. Crystal Oscillator (16MHz)

- **Model**: HC-49S 16MHz
- **Description**: Provides a stable 16MHz clock signal for the microcontroller, ensuring accurate timing and system stability.

5. Ceramic Capacitors (for Decoupling)

- **Model**: VJ1206Y104KXBAT 0.1µF 50V
- **Description**: Multilayer ceramic capacitors used for noise reduction and voltage stabilization in power and signal lines.

6. Electrolytic Capacitors [for Input/Output Filtering] (4x)

- **Model**: MAL214699103E3 100µF 25V
- **Description**: Used for filtering and smoothing power supply outputs to maintain stable voltage levels.

7. Electrolytic Capacitors [Radial Lead Type] (2x)

- **Model**: ECE-A1AN471U

- **Description**: These are capacitors designed for use in motor controllers, providing reliable filtering and energy storage.

8. Resistors (6x)

- **Model**: 1/4W 10kΩ
- **Description**: Standard 10kΩ resistors used for current limiting, voltage division, and signal pull-up/pull-down applications.

9. Diodes

- **Model**: 1N4007
- **Description**: General-purpose diodes used for rectification and protection in circuits, capable of handling up to 1A of current.

10. Power Jack

- **Model**: DCJ20010AK1K
- **Description**: The DC Power Jack is a 5A, 20VDC through-hole connector with a 2mm pin diameter and a 6.3mm outer barrel diameter for secure power connections.

11. Terminal Blocks/4-pin JPT Connectors [for Motor Connections] (4x)

- **Model**: S6B-XH-A (LF)(SN)
- **Description**: These connectors are used for secure and reliable motor connections, allowing for easy interfacing with motor controllers and other electrical components.

12. 1602 LCD Character Display

- **Model**: ACM1602E-FL-YBS
- **Description**: The 1602 LCD Character Display is a popular 16x2 alphanumeric LCD module used in electronics projects to display text and simple characters.

13. 2-Pin JST Male Jack (6x)

- **Model**: B2B-PH-TW-S (LF)(SN) 12V 5A
- **Description**: The 2-Pin JST Male Jack, Model B2B-PH-TW-S (LF)(SN), is a 12V 5A connector used for secure electrical connections in various applications.

02. PCB Design

The PCB layout integrates all three schematics onto a single board, ensuring optimal component placement and signal routing. Key considerations include:

- **Traces**: Adequate width traces are used for power paths to minimize voltage drop and support the current requirements of the components. Signal traces are routed to minimize interference and maintain signal integrity.
- **Component Placement**: Components are positioned strategically to minimize trace lengths and optimize signal paths. Motor drivers are placed close to motor terminals to reduce noise and interference.
- **Ground Planes**: Ground planes are utilized to provide low impedance return paths and minimize ground loops. Separate ground regions are designated for analog and digital sections to prevent crosstalk and interference.
- **Thermal Management**: Heat dissipation for components such as the LM7805 voltage regulator and A4988 motor drivers is ensured through proper placement and possibly the addition of heat sinks or thermal vias.

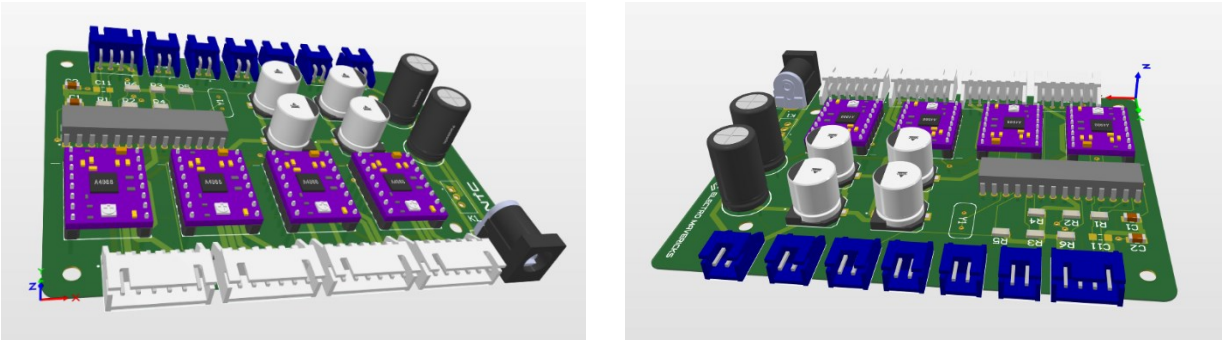


Figure 4 – PCB Design 3D view

03. Finalized PCB Dimensions

The dimension of the PCB is finalized to 73.79mm in length and 100.33mm in width.

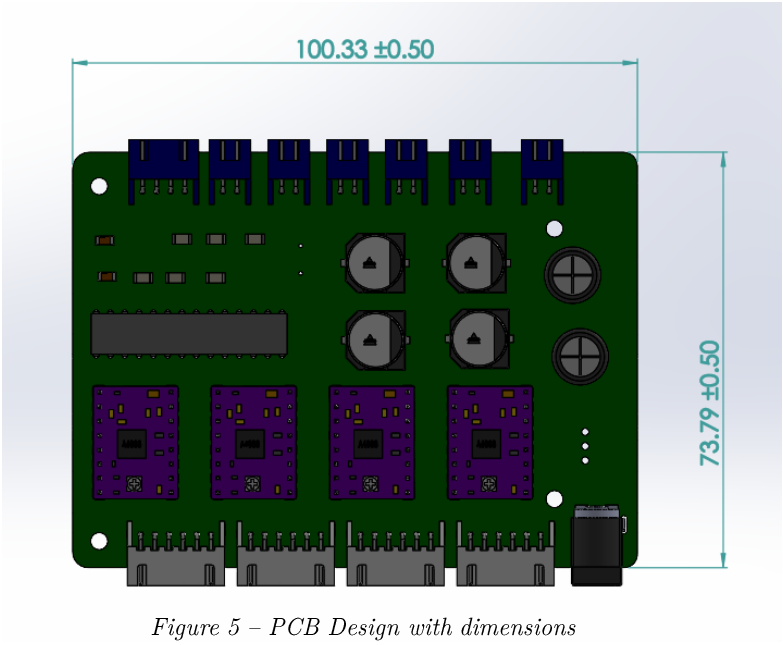


Figure 5 – PCB Design with dimensions

04. Finalized printed PCB

The final design incorporates all schematics into a single PCB layout, ensuring compatibility and proper functionality. The design meets the requirements for controlling stepper motors and interfacing with the ATmega328P MCU while adhering to best practices for electronic circuit design.

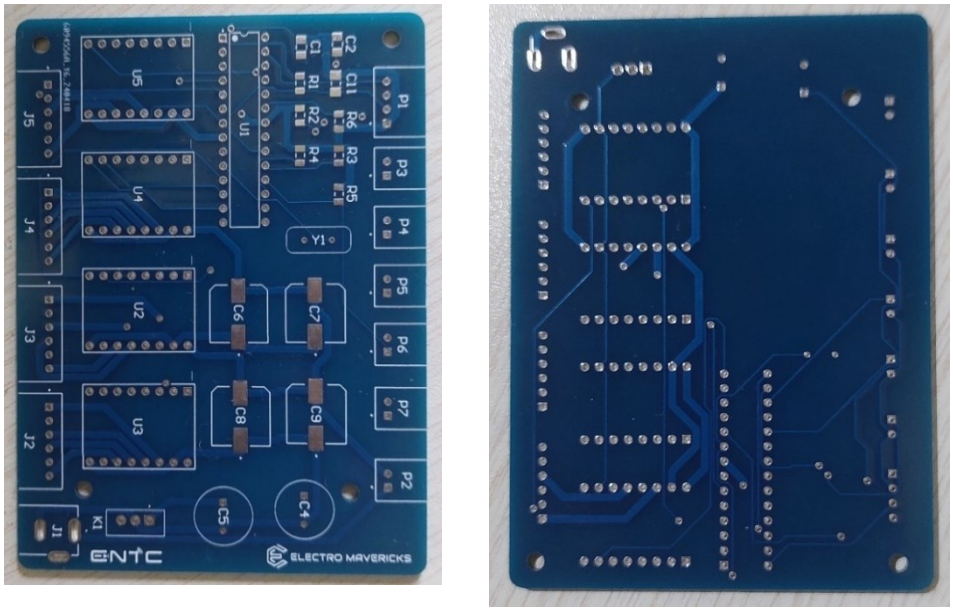


Figure 6 – Printed PCB

05. Soldered PCB

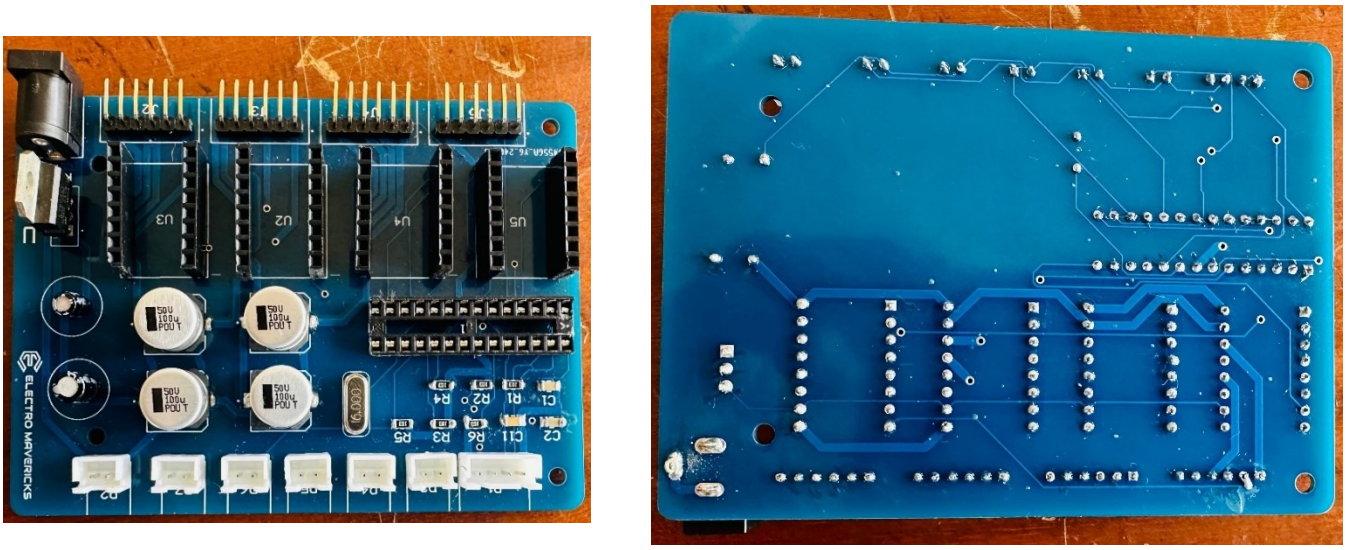


Figure 7 – PCB after soldering

06. Assembled PCB

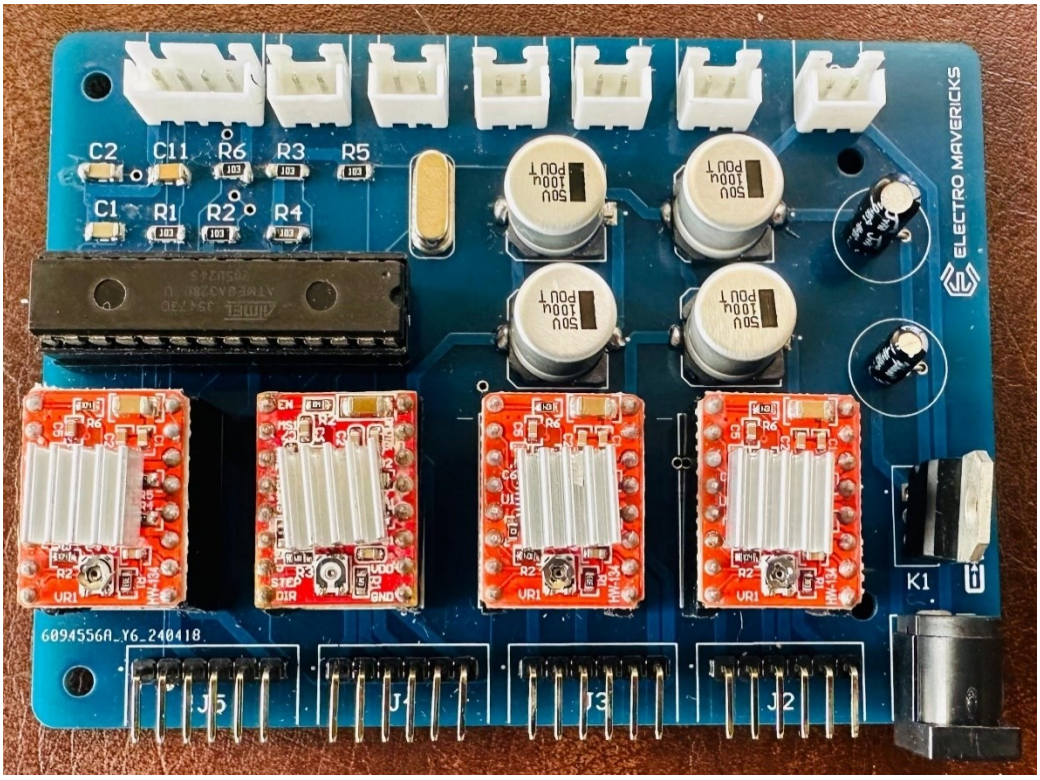


Figure 8 – Assembled PCB

3. Industrial Design

a) Sub-assembly Design

1. **Component Integration:** The mechanical design encompasses the seamless integration of several key components to ensure the efficient operation and versatility of the robot arm system. These components include:

    - **Robot Arm Structure:** The industrial design focuses on the mechanical structure of the robot arm, comprising sturdy and rigid materials to withstand operational stresses. The design prioritizes simplicity, ease of assembly, and structural integrity to ensure reliable performance in diverse environments.

    - **Slider Mechanisms:** Two sliders are incorporated for motion along the X(horizontal) and Z(vertical) axes. These sliders are meticulously designed to provide smooth and precise movement, facilitating accurate positioning of the end-effector tooling. Adjustable positioning allows for flexible placement of objects, enhancing the arm's adaptability to different tasks. In the slider mechanisms, we employ a **T8 Lead Screw with a 500mm ACME Thread** as the screw component. This choice ensures precise linear motion along the X and Z axes. The ACME thread design offers excellent mechanical advantage and self-locking properties, enhancing positional accuracy and stability during operation.

    - Additionally, the sliders are constructed using **aluminum extrusion slots**. Aluminum extrusions provide a lightweight yet durable framework for the sliders, offering exceptional strength-to-weight ratio and corrosion resistance. This choice of material ensures structural integrity while minimizing the overall weight of the slider mechanisms, contributing to improved efficiency and maneuverability of the robot arm.

    - **Rotational Mechanism:** A dedicated rotational mechanism with worm gear is integrated into the design to enable controlled rotation of the gripper assembly. This mechanism facilitates precise orientation of objects held by the gripper, ensuring accurate placement and manipulation during operations. The rotational mechanism enhances the arm's capabilities for handling tasks requiring precise angular positioning.

2. **Material Selection:** Material selection is a critical aspect of the mechanical design, influencing the arm's performance, durability, and overall functionality. Key considerations include:

    - **Structural Components:** High-strength materials such as aluminum alloys or carbon fiber composites are chosen for the robot arm structure to ensure robustness and

rigidity while minimizing weight. These materials offer excellent strength-to-weight ratios, enhancing maneuverability and energy efficiency.

- **Rotational Mechanism:** High-precision bearings and metal worm gear are utilized for the rotational mechanism to facilitate smooth and accurate rotation of the gripper assembly. The use of high-quality materials ensures precise positioning control and minimizes backlash, optimizing the arm's performance during manipulation tasks.

3. **User Interface:** The mechanical design incorporates user-friendly features to enhance the human-machine interaction and operational efficiency of the robot arm. Key elements include:
    - **Control Panel:** An intuitive control panel is integrated into the enclosure, featuring ergonomic controls and clear visual indicators for ease of operation. The user interface is designed to be user-friendly, allowing operators to interact with the arm efficiently and intuitively.
    - **HMI (Human-Machine Interface):** A graphical user interface (GUI) with LCD screen may be incorporated to setting up parameters in arm movements. The HMI allows operators to monitor and control the arm's operation, improving accessibility and productivity.
4. **Safety Features and Compliance:** Safety is a paramount concern in mechanical design, with robust features and adherence to industry safety standards. Key safety considerations include:
    - **Emergency Stop Mechanism:** An emergency stop (E-stop) button is strategically positioned on the arm structure to immediately halt all motion in case of emergency or unexpected events, ensuring operator safety.
    - **Protective Enclosures:** Transparent or opaque enclosures may be installed around main PCB. Protective enclosures enhance operator safety and compliance with safety regulations in industrial environments.

b) Rough sketches

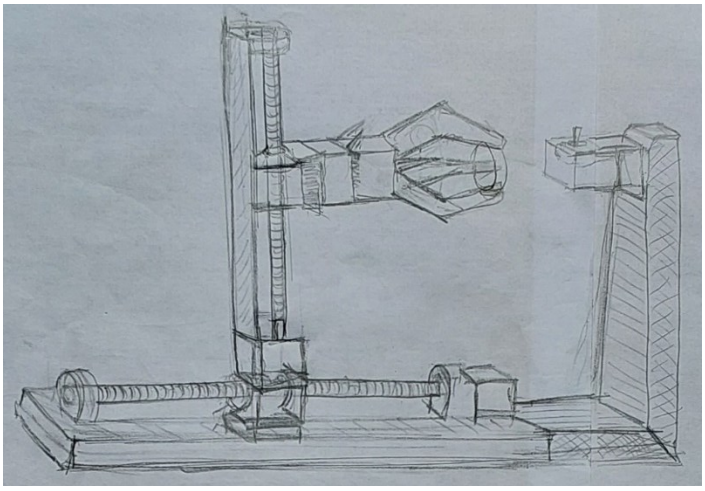


Figure 9 – Rough sketch

c) 3D Modeled Simulation

For the final selected conceptual design, we have simulated it using SolidWorks 3D for realize mechanical concepts.

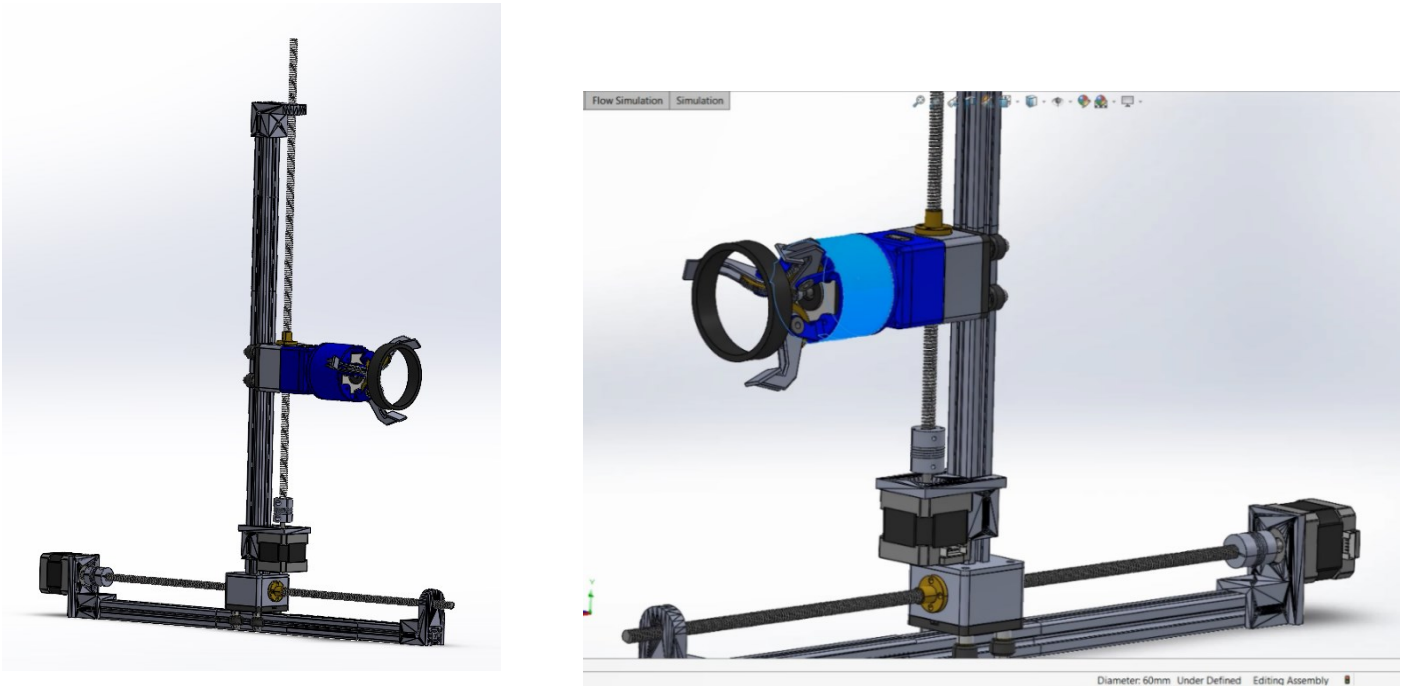


Figure 10 – Final Design 3D model

d) Design of Parts

Supporters

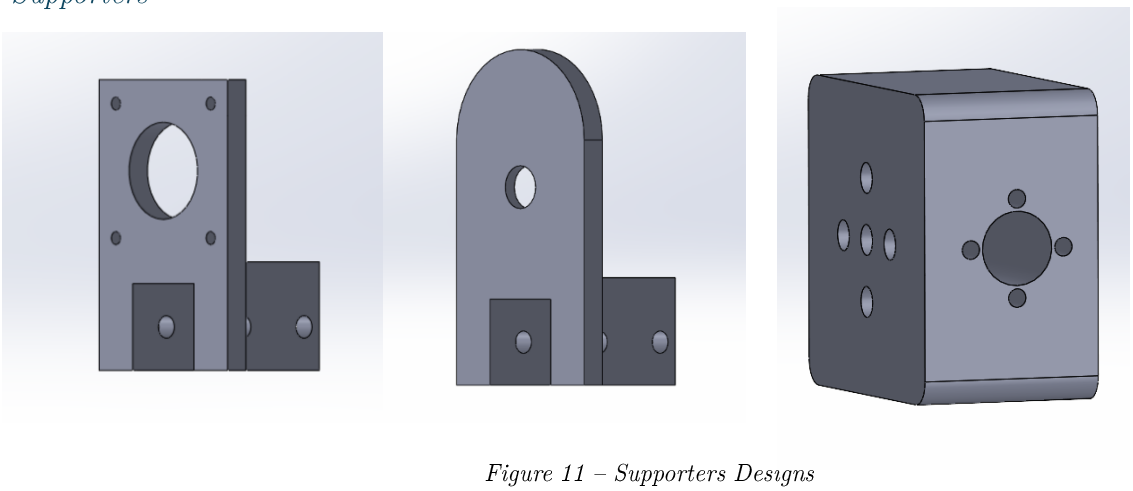


Figure 11 – Supporters Designs

Ring Picking Arm

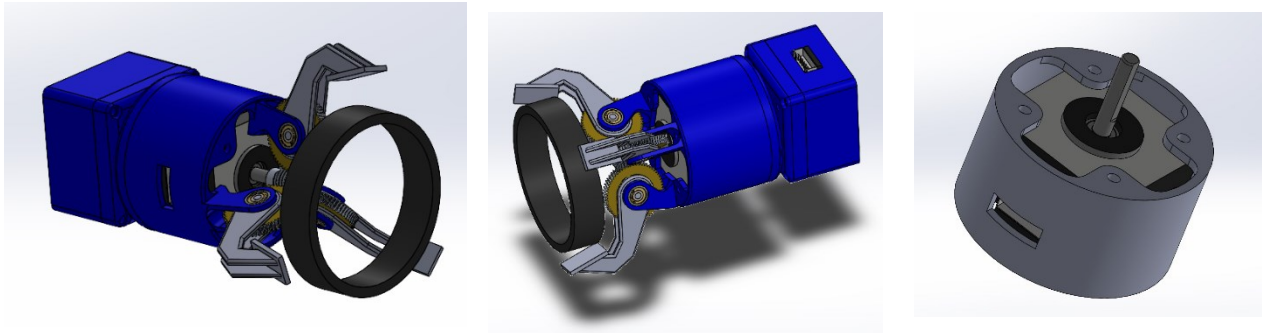


Figure 12 – Gripper Design

Enclosure

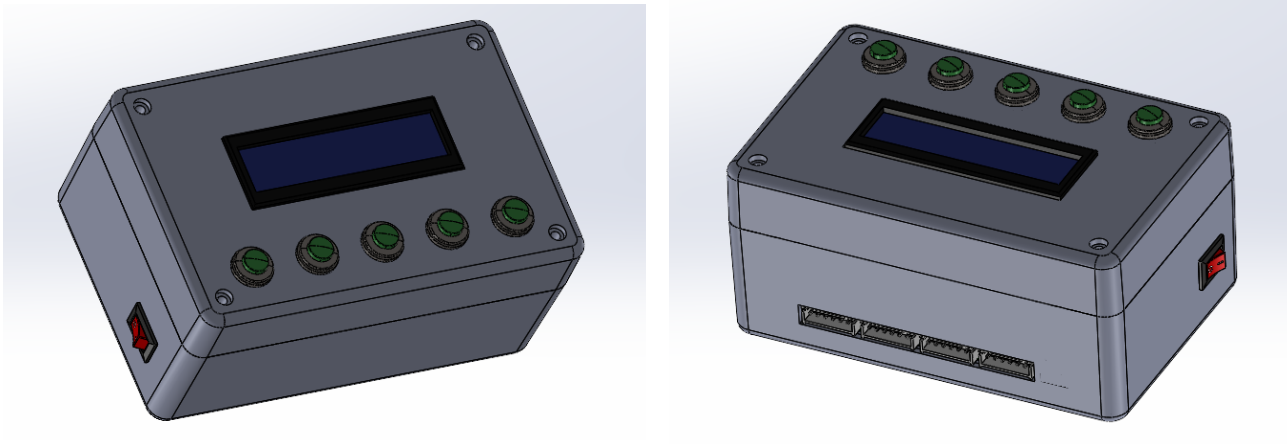


Figure 13 – Controller Design

Enclosure mold design – Top Part

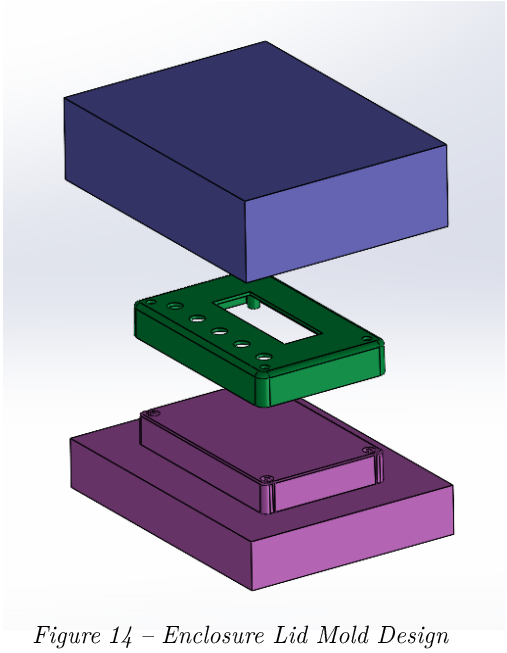


Figure 14 – Enclosure Lid Mold Design

Enclosure mold design – Bottom part

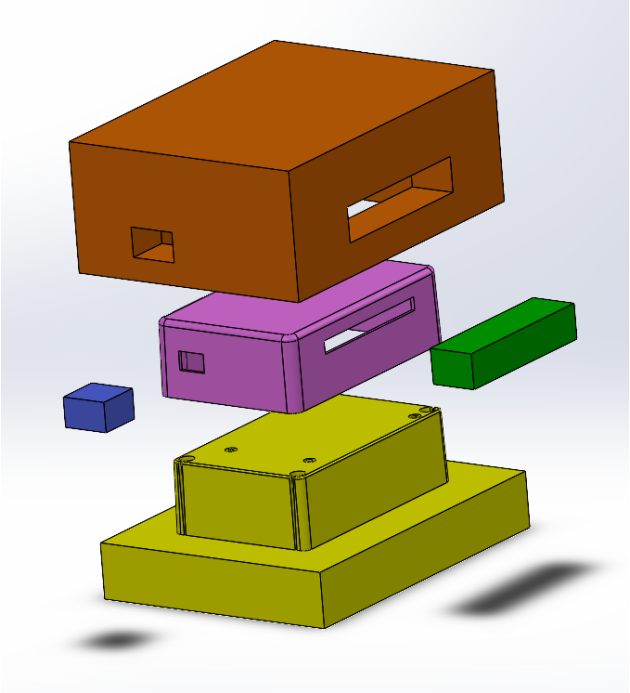


Figure 15 – Enclosure Box Mold Design

e) 3D Printed Parts

The connecting parts of the device will be created using 3D printing with 100% fill as it needs to be hard.

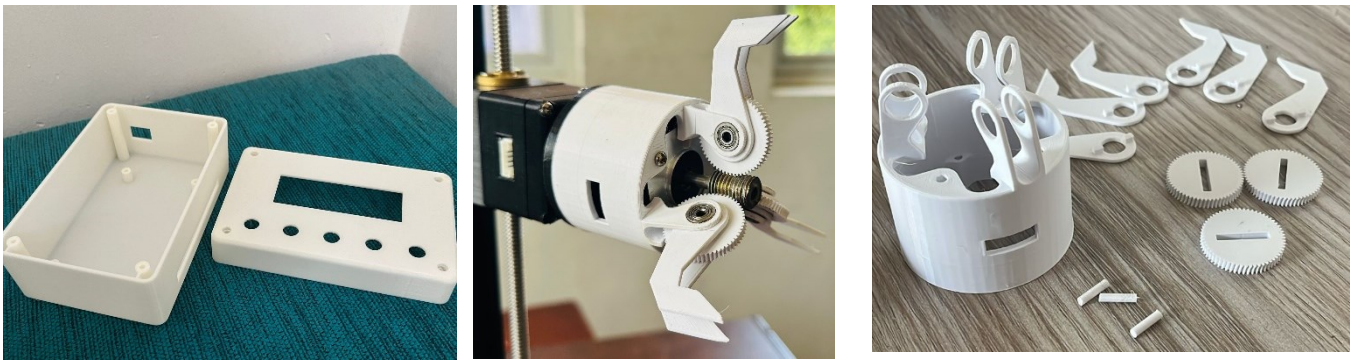


Figure 16 – 3D printed parts

f) Standards Used

Relevant industry standards for mechanical design and fabrication will be followed to ensure compliance with safety, quality, and regulatory requirements. Standards may include ISO, ANSI, ASTM, and specific industry guidelines for robotics and automation.

g) Finish

The finish of mechanical components is determined based on functional requirements, aesthetic considerations, and environmental factors. Common finishes include paint, powder coating, anodizing, plating, and surface treatments to enhance durability and appearance.

4. Mechanical Design

The mechanical design of the automated system for the precise placement and rotation of the metal ring in H-Bridge assemblies involves several key sub-assemblies and components. This section outlines the design of these sub-assemblies, their specifications, and the criteria used for selecting various mechanical parts to ensure optimal performance and reliability.

a) Design of Mechanical Sub-Assemblies

1. Robotic Arm Assembly:

The robotic arm is the central component responsible for handling and positioning the metal rings. It features a multi-axis design, allowing for precise movement in all directions. The arm is equipped with specialized grippers tailored to securely hold metal rings of varying sizes without causing deformation or damage. The design includes a robust frame to support high-speed operations and maintain stability.

2. Gripper:

The gripper is designed to provide firm but gentle handling of metal rings. It incorporates a compliant mechanism that adjusts its grip strength according to the ring size and material, reducing the risk of slippage or damage. The gripper's design allows for quick swapping to accommodate different ring sizes, enhancing the system's flexibility and efficiency.

3. Base and Frame Structure:

The system's base and frame provide structural support for all mechanical components. Designed for stability and rigidity, the frame minimizes vibrations and maintains alignment accuracy during operation. It is constructed from high-strength aluminum alloy, balancing durability with lightweight properties to facilitate ease of movement and installation.

4. Protective Enclosures:

Protective enclosures surround critical components to prevent contamination and provide safety for operators. These enclosures are designed with transparent polycarbonate panels for visibility and include quick-access doors for maintenance. The design ensures compliance with safety regulations and protects the system from environmental hazards.

b) Specifications of Mechanical Sub-Assemblies

1. Robotic Arm:

- **Range of Motion**: 360°
- **Payload Capacity**: 1 kg
- **Drive System**: Stepper motors with motor controllers

2. Gripper:

- **Grip Force**: up to 10 N
- **Jaw Opening Range**: 15 mm to 75 mm

3. Base and Frame Structure:

- **Material**: Extruded aluminum with stainless steel reinforcements
- **Dimensions**: 600 mm x 500 mm x 500 mm (L x W x H)

c) Selection Criteria for Various Parts

1. Robotic Arm:

- **Precision**: Chosen for high repeatability and precision in handling metal ring.
- **Payload Capacity**: Designed to ensure it can handle the maximum weight of the metal rings and gripper.

2. Base and Frame Structure:

- **Stability**: The frame's material and design were chosen to provide maximum stability and minimize vibrations.
- **Material Choice**: Extruded aluminum with stainless steel reinforcements was selected for its combination of strength and lightness.

T8 Lead Screw 1000mm ACME Thread:

- Length: 1000mm
- Type: ACME thread for efficient linear motion
- Precision and controlled movements for CNC and robotic applications and self-locking capability.

Figure 17 – T8 Lead Screw 1000mm ACME Thread

Openbuilds (Type A) V Gantry Plate Set:

- Compatibility with V Slot aluminum extrusions
- Durability and stability for precise alignment
- Modular design for flexibility in configurations

Figure 18 – Openbuilds (Type A) V Gantry Plate Set

V Slot Aluminum Extrusion Profile:

- Versatility in mounting linear motion components
- Strength-to-weight ratio suitable for structural integrity
- Easy assembly with T-slot design

Figure 19 – V Slot Aluminum Extrusion Profile

Aluminum Flexible Coupling 5×8mm:

- Compatibility with 5mm motor shaft and 8mm lead screw
- Damping properties for vibration reduction

Figure 20 – Aluminum Flexible Coupling 5×8mm

5. Calculations

a) Control System Selection

Open-Loop Control System

Pros:

1. **Simplicity and Cost-Effectiveness**: Open-loop systems are straightforward and typically less expensive to implement as they do not require feedback sensors or complex control algorithms.
2. **Ease of Implementation**: With fewer components, open-loop systems are easier to set up and maintain, reducing the overall development time and costs.
3. **Lower Hardware Requirements**: The absence of feedback sensors simplifies the hardware requirements, making it easier to design compact and lightweight systems.
4. **Adequate for Predictable Loads**: These systems are effective in applications where the load is consistent and predictable, ensuring reliable operation without the need for real-time adjustments.

Cons:

1. **Lack of Feedback**: Without feedback, open-loop systems cannot compensate for variations in load or external disturbances, leading to potential inaccuracies and inconsistencies in operation.
2. **Limited Precision**: The absence of real-time adjustments means that open-loop systems may struggle to achieve high precision, particularly in dynamic environments where load conditions vary.
3. **No Error Correction**: Any discrepancies or errors in movement are not detected or corrected, which can result in cumulative errors over time.
4. **Potential for Slippage**: In the case of stepper motors, there is a risk of missed steps or slippage, which can lead to a loss of synchronization and inaccurate positioning.

Closed-Loop Control System

Pros:

1. **High Precision and Accuracy**: Closed-loop systems provide real-time feedback, enabling precise control of position, speed, and torque, and ensuring accurate operation even in varying load conditions.
2. **Error Detection and Correction**: The feedback mechanism allows the system to detect and correct errors, maintaining accuracy and preventing cumulative deviations.
3. **Improved Stability**: These systems can adapt to changes and disturbances, maintaining stability and consistent performance.

4. **Adaptability**: Closed-loop systems can adjust to different load conditions and operational requirements, making them suitable for complex and dynamic environments.

Cons:

1. **Higher Cost and Complexity**: The need for feedback sensors and more complex control algorithms increases the cost and complexity of closed-loop systems.
2. **Increased Development Time**: Implementing a closed-loop system requires more time for design, testing, and tuning to achieve the desired performance.
3. **Higher Hardware Requirements**: The inclusion of sensors and additional components increases the hardware requirements, potentially leading to larger and heavier systems.
4. **Maintenance**: Closed-loop systems may require more frequent maintenance and calibration to ensure the accuracy and reliability of the feedback sensors.

In our analysis, we compared open-loop and closed-loop stepper motor control systems for controlling sliders and velocities. Despite closed-loop systems offering superior accuracy with feedback control, budget constraints led us to choose the open-loop NEMA 17 stepper motors. Our detailed calculations and feasibility studies confirmed that the open-loop system, although lacking feedback, would still meet our performance requirements within budget.

Evaluation of NEMA 17 and A4988 in an Open-Loop System

NEMA 17 Stepper Motors

Advantages:

1. **Adequate Performance**: NEMA 17 stepper motors offer sufficient torque and resolution for many applications, making them a popular choice for tasks that require reliable and consistent operation.
2. **Cost-Effective**: These motors are relatively inexpensive, making them suitable for budget-conscious projects where high precision is not critical.
3. **Ease of Use**: NEMA 17 stepper motors are widely used and supported by numerous resources, including libraries and tutorials, simplifying their integration and use in projects.
4. **No Feedback Required**: As part of an open-loop system, NEMA 17 motors do not require feedback sensors, reducing the overall system complexity and cost.

Disadvantages:

1. **Precision Limitations**: Without feedback, these motors may not achieve the precision required for tasks with tight tolerances, as they cannot compensate for missed steps or variations in load.

2. **Risk of Missteps**: The lack of feedback means there is a risk of missed steps or synchronization issues, especially under varying loads or during rapid acceleration or deceleration.

A4988 Stepper Motor Driver

Advantages:

1. **Simple Integration**: The A4988 is a compact and affordable stepper motor driver that is easy to integrate with NEMA 17 motors, providing basic control functions suitable for open-loop systems.
2. **Micro stepping Capabilities**: The A4988 supports micro stepping, which allows for smoother and more precise control of the stepper motor, improving the resolution and reducing mechanical vibrations.
3. **Cost-Effective Solution**: The driver is relatively inexpensive, making it an attractive choice for projects with budget constraints.
4. **No Feedback Requirements**: The A4988 does not require feedback for operation, aligning with the principles of open-loop control systems and simplifying the overall system design.

Disadvantages:

1. **Limited Precision**: While the A4988 provides basic control, it lacks the feedback mechanisms necessary for high-precision applications, potentially leading to cumulative errors and inaccuracies.
2. **Fixed Current Control**: The driver provides fixed current control, which may not be ideal for applications that require dynamic adjustment of motor torque or speed.
3. **No Error Correction**: The absence of feedback means that the A4988 cannot detect or correct errors, which can lead to performance issues in applications with varying loads or operating conditions.

b) Stepper Motor related calculations

The development of the automated assembly system for custom H-bridges necessitates the meticulous application of several engineering principles to ensure precise placement and rotation of components. This section delves into the calculations and methodologies used for the linear and angular motions of the robotic arm, specifically focusing on the x-axis and y-axis movements, and the angular rotation required for the metal ring placement.

Linear Motion Calculations (X and Z Axis Motion)

The linear motion of the robotic arm is critical for accurately positioning the assembled transistor in the ring in the correct position. The calculations are based on the kinematic equations of motion and the stepper motor specifications:

Steps to Distance Conversion Formula:

- Steps per Revolution (Steps/Rev) : Typically 200 steps/rev for a NEMA 17 motor.
- Distance (in centimeters) : The linear distance to move.
- Step Angle : For NEMA 17 motors, the step angle is 1.8 degrees.

The formula used for converting the desired linear distance into steps is:

$$steps = \frac{distance_{mm} \times steps_{per_{rev}}}{screw\ rod\ pitch}$$

Where:

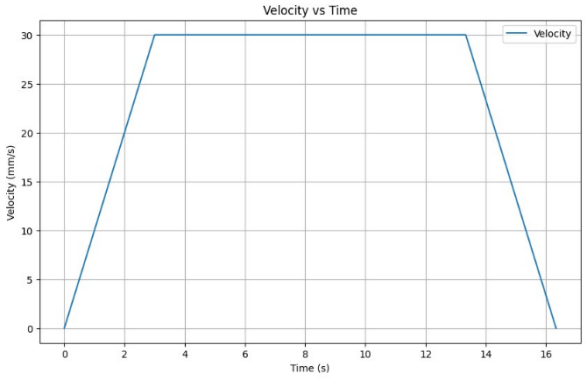
- $steps$ is the number of steps to move.
- $distance_{mm}$ is the distance to move in centimeters.
- $steps_{per_{rev}}$ is the steps per revolution(rotation) of the stepper motor.
- $screw\ rod\ pitch$ is measured as 2mm.

For 10mm linear motion in x or z axis,

$steps = \frac{10mm \times 200_{per_{rev}}}{2mm_{per_{rev}}} = 1000steps$ are needed.

Basic Parameters

- $distance\ per\ step = \frac{screw\ rod\ pitch}{steps\ per\ revolution} = \frac{2mm}{200} = 0.01mm$
- $Set\ RPM = 1000$
- $Maximum\ Velocity(v_{max}) = screwrod\ pitch \times RPM = 2mm \times 1000 = 33.33mm/s$
- $Let's\ set\ the\ acceleration\ to\ 10mm/s^2$ $(Acceptable\ for\ the\ system)$



Since the maximum velocity $\approx 33.33ms^{-1}$, let's use 30mm/s as velocity and 10mm/s acceleration and deacceleration

This calculation ensures the precise linear movement required for the placement process.

Angular Motion Calculations

The rotational movement of the metal ring is another crucial aspect of the placement process ensuring that each of the 6 holes in the ring is properly aligned with the assembled transistor. The angular motion is calculated using the angular kinematic equations and the stepper motor's step angle:

Angular Kinematic Equations

1. $\omega = \omega_o + \alpha t$

2. $\theta = \omega_o t + \frac{1}{2}\alpha t^2$

3. $\omega^2 = {\omega_o}^2 + 2\alpha\theta$

Where:

- ω is the final angular velocity.
- ω_o is the initial angular velocity.
- α is the angular acceleration.
- t is the time.
- θ is the angular displacement.

The step angle of 1.8 degrees (0.0314 radians) for the NEMA 17 motor, to achieve an angular displacement of π radians (180 degrees):

For $\theta = \pi$,

$$steps = \frac{\theta}{step\ angle} = \frac{\pi}{0.0314} \approx 100\ steps$$

- $angle\ between\ two\ holes = \frac{2\pi}{number\ of\ holes}$
- $Let\ number\ of\ holes = 6$
- $angle = \frac{2\pi}{6} = \frac{\pi}{3}$
- $for\ this\ angular\ rotation,$
- $steps = \frac{\theta}{step\ angle} = \frac{\pi/3}{0.0314} \approx 100\ steps$
- $steps = angle \times steps\ per\ angle = \frac{200}{2\pi} \times \pi/3 \approx 33\ steps$

c) Acceleration and Deacceleration Calculations

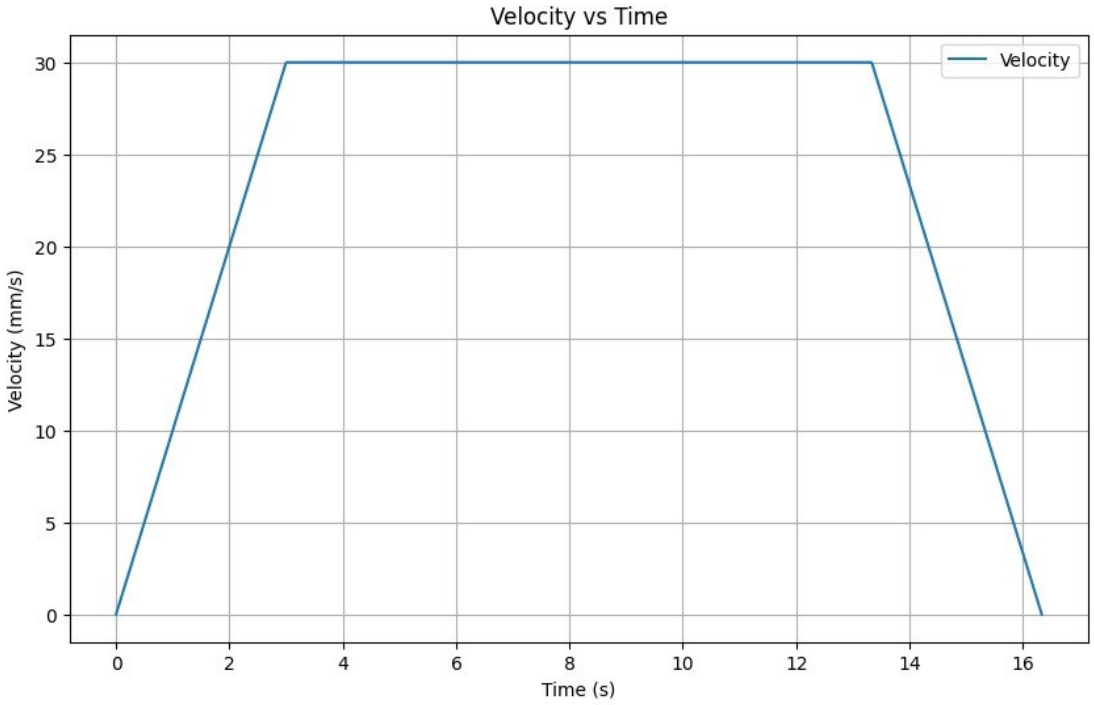
For both linear and angular motions, it is essential to incorporate acceleration and deceleration phases to ensure smooth and precise movements, thereby minimizing mechanical stress and enhancing the longevity of the system. This is typically achieved using trapezoidal motion profiles, which include phases of acceleration, constant velocity, and deceleration.

Trapezoidal Motion Profile

- Acceleration Phase: From 0 to maximum velocity (v_{max} *or* ω_{max})
- Constant Velocity Phase: Maintaining v_{max} *or* ω_{max}
- Deceleration Phase: From v_{max} *or* ω_{max} to 0.

This approach ensures that the system operates within safe mechanical limits and achieves the required precision.

Graphical Representation and Data Visualization



To further validate and optimize the motion profiles, graphs can be generated using the calculated values for displacement, velocity, and time. These graphs, along with the data from stepper motor datasheets, provide a visual representation of the system's performance and facilitate fine-tuning of the control algorithms.

By applying these engineering principles and performing the necessary calculations, we ensure that the robotic arm operates with high precision and efficiency, ultimately enhancing the overall functionality and reliability of the automated assembly system for custom H-bridges.

6. Software Details

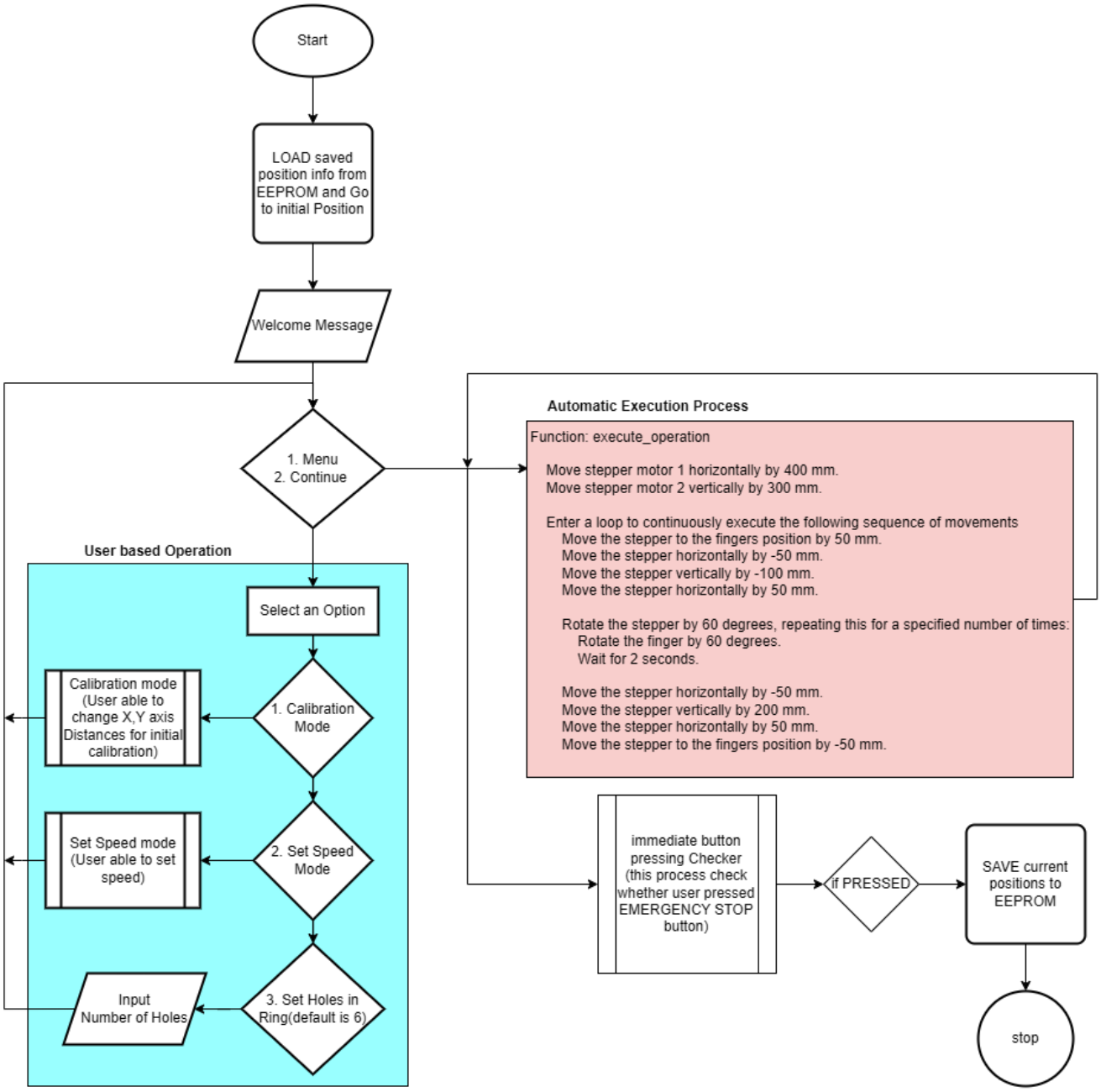
a) Software Architecture Overview

The main software components include:

- **Main Program:** Orchestrates the overall operation flow, including initializing modules, controlling motor movements, managing user interface interactions, and handling emergency stop procedures.
- **Motor Control Module:** Manages the movement and positioning of stepper motors based on input commands and programmed logic. It implements motion control algorithms, error correction strategies, and system logic to ensure precise motor movements and stable operation.
- **User Interface Module:** Facilitates user interaction through input devices and an LCD display, allowing users to control system parameters and monitor operation status.
- **Emergency Stop:** Implements immediate system halt procedures in emergencies or safety concerns and save current position to EEPROM.

```
- Main Program
  - Initializing Modules and Parameters
  - Align Ring to MOSFET
  - Move Forward to Pivot Placement
  - Wait Until Pivot Placement Completed
  - Retract Arm
  - Rotate 60 Degrees
  - Loop:
    - Go to Step 3 if Not All 6 MOSFETs Pivoted
  - Completed and Go to Placing Phase
- UI Module
  - Manual Control Interface
  - Calibration Interface
- Stepper Control Module
  - Acceleration Control
  - Deceleration Control
  - Motion Profile Management
- Emergency Stop Procedure
  - Immediate System Halt
  - Save current positions to EEPROM
```

b) Operation Flowchart



c) Source Code

The code handles the control of stepper motors for horizontal, vertical, rotational, and gripping movements along with LCD and button interfacing for user input.

Initialization

The main function initializes the motor pins, LCD, and buttons, reads motor positions from EEPROM, and enters the main loop for operation.

```
int main() {
    initializeMotorPins();   // Initialize direction and step pins as output
    setupLCD();              // Initialize LCD
    setupButtonPins();       // Initialize buttons

    initializeMotorPositions();  // Initialize motor positions from EEPROM

    loop();     // Execute the main operation

    return 0;
}
```

Necessary Headers and Imports

The following headers are included for AVR microcontroller operations, interrupts, delay functions, and EEPROM handling.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/eeprom.h>
```

Variable Assignment

Various parameters, pin definitions, and variables for motor positions, speeds, coordinates, and default values are defined.

```
#define MAX_SPEED 200
#define ACCEL_RATE 10
#define DECEL_RATE 10

#define F_CPU 16000000UL

#define DIR1_PORT PORTD
#define DIR1_PIN PD3
#define STEP1_PORT PORTD
#define STEP1_PIN PD2

#define DIR2_PORT PORTD
#define DIR2_PIN PD5
#define STEP2_PORT PORTD
#define STEP2_PIN PD4

#define DIR3_PORT PORTD
#define DIR3_PIN PD7
#define STEP3_PORT PORTD
#define STEP3_PIN PD6

#define DIR4_PORT PORTB
#define DIR4_PIN PB1
#define STEP4_PORT PORTB
#define STEP4_PIN PB0

#define UP_BUTTON_PIN PC0
#define DOWN_BUTTON_PIN PC1
#define MENU_BUTTON_PIN PC2
#define CANCEL_BUTTON_PIN PC3
#define IMMEDIATE_BUTTON_PIN PC4

uint16_t eepromAddr_stepper1 = 0;
uint16_t eepromAddr_stepper2 = 2;
uint16_t eepromAddr_stepper3 = 4;
uint16_t eepromAddr_stepper4 = 6;

int16_t savedPosition1 = 0;
```

```
int16_t savedPosition2 = 0;
int16_t savedPosition3 = 0;
int16_t savedPosition4 = 0;

int16_t initial_speed1 = 10;
int16_t initial_speed2 = 10;
int16_t initial_speed3 = 10;
int16_t initial_speed4 = 10;
int16_t initial_x_coordinate = 10;
int16_t initial_z_coordinate = 10;
int16_t default_holes = 6;

int16_t temp_speed1 = 0;
int16_t temp_speed2 = 0;
int16_t temp_speed3 = 10;
int16_t temp_speed4 = 10;
int16_t temp_x_coordinate = 0;
int16_t temp_z_coordinate = 0;
int16_t temp_holes = 0;

int16_t  runloop = 1;
```

Execution

The main loop displays the main menu and handles button presses to navigate the menu or execute operations.

```
void loop() {
    while (1) {
        displayMainMenu();
        _delay_ms(100);

        if (currentStateChange(UP_BUTTON_PIN)) {
            current_main_mode = (current_main_mode + 1) % max_main_modes;
        } else if (currentStateChange(DOWN_BUTTON_PIN)) {
            current_main_mode = (current_main_mode - 1 + max_main_modes) %
max_main_modes;
        } else if (currentStateChange(MENU_BUTTON_PIN)) {
            if (current_main_mode == 0) {
                navigateMenu();
            } else {
                execute_operation();
            }
```

```
        }
    }
}
```

Functions

LCD Functions

These functions handle the initialization, command, data sending, and string printing to the LCD.

```
void i2c_init() {
    TWSR = 0x00;
    TWBR = 0x0C;
    TWCR = (1 << TWEN);
}

void i2c_start() {
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

void i2c_stop() {
    TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
    while (TWCR & (1 << TWSTO));
}

void i2c_write(uint8_t data) {
    TWDR = data;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

void lcd_command(uint8_t cmd) {
    i2c_start();
    i2c_write(0x4E);
    i2c_write(0x00);
    i2c_write(cmd);
    i2c_stop();
}

void lcd_data(uint8_t data) {
    i2c_start();
    i2c_write(0x4E);
```

```
    i2c_write(0x40);
    i2c_write(data);
    i2c_stop();
}

void lcd_init() {
    _delay_ms(50);
    lcd_command(0x38);
    _delay_ms(5);
    lcd_command(0x0C);
    _delay_ms(5);
    lcd_command(0x01);
    _delay_ms(5);
    lcd_command(0x06);
}

void lcd_print(const char* str) {
    while (*str) {
        lcd_data(*str++);
    }
}

void setupLCD() {
    i2c_init();
    lcd_init();
    lcd_print("Welcome to Pick and Place Robot Arm");
    _delay_ms(5000);
    lcd_command(0x01);
}
```

Button Configuration

This function sets up the button pins for input and enables pull-up resistors.

```
void setupButtonPins() {
    DDRC &= ~((1 << UP_BUTTON_PIN) | (1 << DOWN_BUTTON_PIN) | (1 <<
MENU_BUTTON_PIN) | (1 << CANCEL_BUTTON_PIN) | (1 <<
IMMEDIATE_BUTTON_PIN));
    PORTC |= (1 << UP_BUTTON_PIN) | (1 << DOWN_BUTTON_PIN) | (1 <<
MENU_BUTTON_PIN) | (1 << CANCEL_BUTTON_PIN) | (1 <<
IMMEDIATE_BUTTON_PIN);
}
```

Stepper Motor Functions

These functions handle the movements of stepper motors with acceleration and deceleration phases.

Basic Motions

```
void stepper_to_horizontal(int distance_cm) {
    float distance_per_rotation_cm = 2.0;
    float steps_per_cm = 200.0;
    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;
    step_motor(&STEP1_PORT, STEP1_PIN, &DIR1_PORT, DIR1_PIN, steps, 1);
}

void stepper_to_vertical(int distance) {
    float distance_per_rotation_cm = 2.0;
    float steps_per_cm = 200.0;
    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;
    step_motor(&STEP2_PORT, STEP2_PIN, &DIR2_PORT, DIR2_PIN, steps, 1);
}

void stepper_to_rotation(int angle) {
    int steps = angle * 200 / 360;
    step_motor(&STEP3_PORT, STEP3_PIN, &DIR3_PORT, DIR3_PIN, steps, 1);
}

void stepper_to_fingers(int distance) {
    float distance_per_rotation_cm = 1.0;
    float steps_per_cm = 200.0;
    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;
    step_motor(&STEP4_PORT, STEP4_PIN, &DIR4_PORT, DIR4_PIN, steps, 1);
}
```

Acceleration and Deacceleration Handling

```
void step_motor(volatile uint8_t *step_port, uint8_t step_pin, volatile uint8_t *dir_port, uint8_t
dir_pin, uint16_t steps, uint8_t direction) {

    //check immediate action
    if (immediate()) {
```

```
        save_positions();  // Save current positions
        runloop = 0;       // Exit the loop
        break;
    }
// Set direction
if (direction) {
    *dir_port |= (1 << dir_pin);
} else {
    *dir_port &= ~(1 << dir_pin);
}

int MAX_STEP_SPEED =MAX_SPEED*200/2
int ACCEL_RATE_STEP =ACCEL_RATE*200/2
int DECEL_RATE_STEP =DECEL_RATE*200/2


// Calculate the number of steps for each phase
uint16_t accel_steps = MAX_STEP_SPEED / ACCEL_RATE_STEP;
uint16_t decel_steps = MAX_STEP_SPEED / DECEL_RATE_STEP;
uint16_t constant_steps = steps - (accel_steps + decel_steps);

// Acceleration phase
for (uint16_t i = 0; i < accel_steps; i++) {
    *step_port |= (1 << step_pin);
    _delay_us(1000000 / (ACCEL_RATE_STEP * i + 1));
    *step_port &= ~(1 << step_pin);
    _delay_us(1000000 / (ACCEL_RATE_STEP * i + 1));
}

// Constant speed phase
for (uint16_t i = 0; i < constant_steps; i++) {
    *step_port |= (1 << step_pin);
    _delay_us(1000000 / MAX_STEP_SPEED);
    *step_port &= ~(1 << step_pin);
    _delay_us(1000000 / MAX_STEP_SPEED);
}

// Deceleration phase
for (uint16_t i = decel_steps; i > 0; i--) {
    *step_port |= (1 << step_pin);
    _delay_us(1000000 / (DECEL_RATE_STEP * i + 1));
    *step_port &= ~(1 << step_pin);
    _delay_us(1000000 / (DECEL_RATE_STEP * i + 1));
```

```
    }
}
```

Menu Navigation Functions

These functions handle the display and navigation of menus.

```
void displayMainMenu() {
    lcd_command(0x01);
    lcd_print("1. Calibrate X-Axis");
    lcd_command(0xC0);
    lcd_print("2. Calibrate Z-Axis");
    lcd_command(0x94);
    lcd_print("3. Set Speed");
    lcd_command(0xD4);
    lcd_print("4. Set No. of Holes");
    lcd_command(0xC4);
    lcd_print("5. Execute Operation");
}
```

Helper Functions

These functions include utility functions for button state changes, saving and reading EEPROM, and menu navigation.

```
bool currentStateChange(uint8_t pin) {
    static uint8_t lastState[8] = {0};
    bool changed = (PINC & (1 << pin)) != lastState[pin];
    lastState[pin] = PINC & (1 << pin);
    return changed;
}

void savePositionToEEPROM(uint16_t addr, int16_t value) {
    eeprom_write_word((uint16_t*)addr, value);
}

int16_t readPositionFromEEPROM(uint16_t addr) {
    return eeprom_read_word((uint16_t*)addr);
}

void navigateMenu() {
    lcd_command(0x01);
    lcd_print("Select Option:");
    _delay_ms(2000);
```

```
    int sub_mode1 = 0;
    while (1) {
        if (currentStateChange(UP_BUTTON_PIN)) {
            sub_mode1 = (sub_mode1 + 1) % 2;
        } else if (currentStateChange(DOWN_BUTTON_PIN)) {
            sub_mode1 = (sub_mode1 - 1 + 2) % 2;
        } else if (currentStateChange(MENU_BUTTON_PIN)) {
            if (sub_mode1 == 0) {
                navigateSub1Menu(&sub_mode1);
            } else {
                navigateSub2Menu(&sub_mode1);
            }
        }
    }
}

void navigateSub1Menu(int* sub_mode1) {
    lcd_command(0x01);
    lcd_print("Calibrate X-Axis");
    _delay_ms(2000);
    calibrateXAxis();
    *sub_mode1 = 0;
}

void navigateSub2Menu(int* sub_mode2) {
    lcd_command(0x01);
    lcd_print("Calibrate Z-Axis");
    _delay_ms(2000);
    calibrateZAxis();
    *sub_mode2 = 0;
}

void calibrateXAxis() {
    lcd_command(0x01);
    lcd_print("Calibrating X-Axis");
    _delay_ms(2000);
}

void calibrateZAxis() {
    lcd_command(0x01);
    lcd_print("Calibrating Z-Axis");
    _delay_ms(2000);
}
```

```
void setSpeed(int stepper) {
    lcd_command(0x01);
    lcd_print("Set Speed for Stepper");
    _delay_ms(2000);
}

void setStepperSpeed(int stepperNumber) {
    lcd_command(0x01);
    lcd_print("Set Speed for Stepper");
    _delay_ms(2000);
}

void setNumberOfHoles() {
    lcd_command(0x01);
    lcd_print("Set Number of Holes");
    _delay_ms(2000);
}

bool immediate() {
    return PINC & (1 << IMMEDIATE_BUTTON_PIN);
}

void save_positions() {
    savePositionToEEPROM(eepromAddr_stepper1, savedPosition1);
    savePositionToEEPROM(eepromAddr_stepper2, savedPosition2);
    savePositionToEEPROM(eepromAddr_stepper3, savedPosition3);
    savePositionToEEPROM(eepromAddr_stepper4, savedPosition4);
}

void initializeMotorPins() {
    DDRD |= (1 << DIR1_PIN) | (1 << STEP1_PIN) | (1 << DIR2_PIN) | (1 << STEP2_PIN)
| (1 << DIR3_PIN) | (1 << STEP3_PIN);
    DDRB |= (1 << DIR4_PIN) | (1 << STEP4_PIN);
}

void initializeMotorPositions() {
    savedPosition1 = readPositionFromEEPROM(eepromAddr_stepper1);
    savedPosition2 = readPositionFromEEPROM(eepromAddr_stepper2);
    savedPosition3 = readPositionFromEEPROM(eepromAddr_stepper3);
    savedPosition4 = readPositionFromEEPROM(eepromAddr_stepper4);
}
```

d) Full Source Code

The full source code includes the entire software implementation for controlling an automated system for metal ring placement and rotation.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/eeprom.h>

// Define the parameters
#define MAX_SPEED 200   // Maximum speed (steps per second)
#define ACCEL_RATE 10   // Acceleration rate (steps per second^2)
#define DECEL_RATE 10   // Deceleration rate (steps per second^2)

// Define F_CPU for _delay_ms()
#define F_CPU 16000000UL

// Pin Definitions
#define DIR1_PORT PORTD
#define DIR1_PIN PD3
#define STEP1_PORT PORTD
#define STEP1_PIN PD2

#define DIR2_PORT PORTD
#define DIR2_PIN PD5
#define STEP2_PORT PORTD
#define STEP2_PIN PD4

#define DIR3_PORT PORTD
#define DIR3_PIN PD7
#define STEP3_PORT PORTD
#define STEP3_PIN PD6

#define DIR4_PORT PORTB
#define DIR4_PIN PB1
#define STEP4_PORT PORTB
#define STEP4_PIN PB0

#define UP_BUTTON_PIN PC0
#define DOWN_BUTTON_PIN PC1
#define MENU_BUTTON_PIN PC2
#define CANCEL_BUTTON_PIN PC3
```

```
#define IMMEDIATE_BUTTON_PIN PC4

// EEPROM Addresses for saving motor positions
uint16_t eepromAddr_stepper1 = 0;
uint16_t eepromAddr_stepper2 = 2;
uint16_t eepromAddr_stepper3 = 4;
uint16_t eepromAddr_stepper4 = 6;

// Variables to store saved positions
int16_t savedPosition1 = 0;
int16_t savedPosition2 = 0;
int16_t savedPosition3 = 0;
int16_t savedPosition4 = 0;

int16_t initial_speed1 = 10;
int16_t initial_speed2 = 10;
int16_t initial_speed3 = 10;
int16_t initial_speed4 = 10;
int16_t initial_x_coordinate = 10;
int16_t initial_z_coordinate = 10;
int16_t default_holes = 6;

int16_t temp_speed1 = 0;
int16_t temp_speed2 = 0;
int16_t temp_speed3 = 10;
int16_t temp_speed4 = 10;
int16_t temp_x_coordinate = 0;
int16_t temp_z_coordinate = 0;
int16_t temp_holes = 0;

int16_t  runloop = 1;
// LCD Initialization
void i2c_init() {
    // Initialize I2C (TWI) interface
    TWSR = 0x00;  // Set prescaler to 1
    TWBR = 0x0C;  // Set bit rate register (SCL frequency = F_CPU / (16 + 2 * TWBR *
prescaler))
    TWCR = (1 << TWEN);  // Enable TWI
}

void i2c_start() {
    // Send start condition
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
```

```
    while (!(TWCR & (1 << TWINT)));  // Wait for start condition to be transmitted
}

void i2c_stop() {
    // Send stop condition
    TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
    while (TWCR & (1 << TWSTO));  // Wait for stop condition to be executed
}

void i2c_write(uint8_t data) {
    // Write data to TWI data register
    TWDR = data;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));  // Wait for data to be transmitted
}

void lcd_command(uint8_t cmd) {
    // Send command to LCD
    i2c_start();
    i2c_write(0x4E);  // LCD I2C address
    i2c_write(0x00);  // Control byte: Co = 0, RS = 0
    i2c_write(cmd);
    i2c_stop();
}

void lcd_data(uint8_t data) {
    // Send data to LCD
    i2c_start();
    i2c_write(0x4E);  // LCD I2C address
    i2c_write(0x40);  // Control byte: Co = 0, RS = 1
    i2c_write(data);
    i2c_stop();
}

void lcd_init() {
    // Initialize LCD
    _delay_ms(50);
    lcd_command(0x38);  // Function set: 8-bit, 2 lines, 5x8 dots
    _delay_ms(5);
    lcd_command(0x0C);  // Display on, cursor off, blink off
    _delay_ms(5);
    lcd_command(0x01);  // Clear display
    _delay_ms(5);
```

```
    lcd_command(0x06);  // Entry mode set: increment, no shift
}

void lcd_print(const char* str) {
    // Print string to LCD
    while (*str) {
        lcd_data(*str++);
    }
}

void setupLCD() {
    i2c_init();
    lcd_init();
    lcd_print("Welcome to Pick and Place Robot Arm");
    _delay_ms(5000);
    lcd_command(0x01);  // Clear display
}

// Button Configuration
void setupButtonPins() {
    DDRC &= ~((1 << UP_BUTTON_PIN) | (1 << DOWN_BUTTON_PIN) | (1 <<
MENU_BUTTON_PIN) | (1 << CANCEL_BUTTON_PIN) | (1 <<
IMMEDIATE_BUTTON_PIN));
    PORTC |= (1 << UP_BUTTON_PIN) | (1 << DOWN_BUTTON_PIN) | (1 <<
MENU_BUTTON_PIN) | (1 << CANCEL_BUTTON_PIN) | (1 <<
IMMEDIATE_BUTTON_PIN);
}

// Main Menu and Submenus
const char* options_main[] = { "1 - Menu", "2 - Continue" };
int current_main_mode = 0;
const int max_main_modes = 2;

const char* options[] = { "1 - Calibration Mode", "2 - Set Speed", "3 - Number of the holes", "4 -
Back" };
const char* sub1_options[] = { "1 - Distance X axis", "2 - Distance Z axis", "3 - Back" };
const char* sub2_options[] = { "1 - Stepper 1", "2 - Stepper 2", "3 - Stepper 3", "4 - Stepper 4", "5 -
Back" };

void displayMainMenu() {
    lcd_command(0x01);  // Clear display
    lcd_print(options_main[current_main_mode]);
}
```

```
void loop() {
    while (1) {
        displayMainMenu();
        _delay_ms(100);

        if (currentStateChange(UP_BUTTON_PIN)) {
            current_main_mode = (current_main_mode + 1) % max_main_modes;
        } else if (currentStateChange(DOWN_BUTTON_PIN)) {
            current_main_mode = (current_main_mode - 1 + max_main_modes) %
max_main_modes;
        } else if (currentStateChange(MENU_BUTTON_PIN)) {
            if (current_main_mode == 0) {
                navigateMenu();
            } else {
                execute_operation();
            }
        }
    }
}

void stepper_to_horizontal(int distance_cm) {
    // Calculate steps based on distance (convert cm to steps)
    // 360 degrees of rotation corresponds to 2 cm of linear distance
    float distance_per_rotation_cm = 2.0;
    float steps_per_cm = 200.0;  // full step mode

    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;

    // Move the motor
    step_motor(&STEP1_PORT, STEP1_PIN, &DIR1_PORT, DIR1_PIN, steps, 1);
}

// Function to move stepper motor 2 vertically
void stepper_to_vertical(int distance) {
    /// Calculate steps based on distance (convert cm to steps)
    // 360 degrees of rotation corresponds to 2 cm of linear distance
    float distance_per_rotation_cm = 2.0;
    float steps_per_cm = 200.0;  // full step mode

    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;
    step_motor(&STEP2_PORT, STEP2_PIN, &DIR2_PORT, DIR2_PIN, steps, 1);
}
```

```
// Function to rotate stepper motor 3 by a specified angle
void stepper_to_rotation(int angle) {
    // Calculate steps based on angle (convert degrees to steps)
    int steps = angle*200/360;

    // Move the motor
    step_motor(&STEP3_PORT, STEP3_PIN, &DIR3_PORT, DIR3_PIN, steps, 1);
}

// Function to move stepper motor 4 for fingers or gripper movement
void stepper_to_fingers(int distance) {
     /// Calculate steps based on distance (convert cm to steps)
    // 360 degrees of rotation corresponds to 2 cm of linear distance
    float distance_per_rotation_cm = 1.0;
    float steps_per_cm = 200.0;  // full step mode

    int steps = distance_cm * steps_per_cm / distance_per_rotation_cm;
    step_motor(&STEP4_PORT, STEP4_PIN, &DIR4_PORT, DIR4_PIN, steps, 1);
}

void step_motor(volatile uint8_t *step_port, uint8_t step_pin, volatile uint8_t *dir_port, uint8_t
dir_pin, uint16_t steps, uint8_t direction) {

    //check immediate action
    if (immediate()) {
            save_positions();  // Save current positions
            runloop = 0;       // Exit the loop
            break;
        }
    // Set direction
    if (direction) {
        *dir_port |= (1 << dir_pin);
    } else {
        *dir_port &= ~(1 << dir_pin);
    }

    int MAX_STEP_SPEED =MAX_SPEED*200/2
    int ACCEL_RATE_STEP =ACCEL_RATE*200/2
    int DECEL_RATE_STEP =DECEL_RATE*200/2


    // Calculate the number of steps for each phase
```

```
    uint16_t accel_steps = MAX_STEP_SPEED / ACCEL_RATE_STEP;
    uint16_t decel_steps = MAX_STEP_SPEED / DECEL_RATE_STEP;
    uint16_t constant_steps = steps - (accel_steps + decel_steps);

    // Acceleration phase
    for (uint16_t i = 0; i < accel_steps; i++) {
        *step_port |= (1 << step_pin);
        _delay_us(1000000 / (ACCEL_RATE_STEP * i + 1));
        *step_port &= ~(1 << step_pin);
        _delay_us(1000000 / (ACCEL_RATE_STEP * i + 1));
    }

    // Constant speed phase
    for (uint16_t i = 0; i < constant_steps; i++) {
        *step_port |= (1 << step_pin);
        _delay_us(1000000 / MAX_STEP_SPEED);
        *step_port &= ~(1 << step_pin);
        _delay_us(1000000 / MAX_STEP_SPEED);
    }

    // Deceleration phase
    for (uint16_t i = decel_steps; i > 0; i--) {
        *step_port |= (1 << step_pin);
        _delay_us(1000000 / (DECEL_RATE_STEP * i + 1));
        *step_port &= ~(1 << step_pin);
        _delay_us(1000000 / (DECEL_RATE_STEP * i + 1));
    }
}

void execute_operation() {

    stepper_to_horizontal(40);  // Move stepper 1 horizontally by 40 cm
    stepper_to_vertical(30);    // Move stepper 2 vertically by 30cm

    while (runloop) {
        // Execute the sequence of movements
        stepper_to_fingers(5);
        _delay_ms(1000);
        stepper_to_horizontal(-5);
        _delay_ms(1000);
        stepper_to_vertical(-10);
        _delay_ms(1000);
```

```
        stepper_to_horizontal(5);
        _delay_ms(1000);
        for (int i=0,I<holes,i++){
            stepper_to_rotation(30);
            _delay_ms(2000);
        }
        stepper_to_horizontal(-5);
        _delay_ms(1000);

        stepper_to_vertical(20);
        _delay_ms(1000);
        stepper_to_horizontal(5);
        _delay_ms(1000);
        stepper_to_fingers(-5);
        _delay_ms(1000);

        stepper_to_vertical(-10);

        _delay_ms(5000);  // Wait for 5 seconds
    }
}

// Helper Functions
bool currentStateChange(uint8_t pin) {
    static uint8_t lastState[5] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
    static uint8_t counter[5] = { 0, 0, 0, 0, 0 };
    uint8_t index = 0;

    switch (pin) {
        case UP_BUTTON_PIN: index = 0; break;
        case DOWN_BUTTON_PIN: index = 1; break;
        case MENU_BUTTON_PIN: index = 2; break;
        case CANCEL_BUTTON_PIN: index = 3; break;
        case IMMEDIATE_BUTTON_PIN: index = 4; break;
    }

    uint8_t currentState = (PINC & (1 << pin)) >> pin;
    if (currentState != lastState[index]) {
        counter[index]++;
        if (counter[index] >= 4) {  // Debounce threshold
            counter[index] = 0;
            lastState[index] = currentState;
            if (currentState == 0) {
```

```
            return true;
        }
    }
} else {
    counter[index] = 0;
}

return false;
}

void savePositionToEEPROM(uint16_t addr, int16_t value) {
    eeprom_update_word((uint16_t*)addr, value);
}

int16_t readPositionFromEEPROM(uint16_t addr) {
    return eeprom_read_word((uint16_t*)addr);
}

void navigateMenu() {
    int mode = 0;
    int sub_mode1 = 0;
    int sub_mode2 = 0;

    while (1) {
        lcd_command(0x01);  // Clear display
        lcd_print(options[mode]);
        _delay_ms(100);

        if (currentStateChange(UP_BUTTON_PIN)) {
            mode = (mode + 1) % 4;
        } else if (currentStateChange(DOWN_BUTTON_PIN)) {
            mode = (mode - 1 + 4) % 4;
        } else if (currentStateChange(MENU_BUTTON_PIN)) {
            switch (mode) {
                case 0:
                    while (1) {
                        navigateSub1Menu(&sub_mode1);
                        if (sub_mode1 == 2) break;
                        if (currentStateChange(MENU_BUTTON_PIN)) {
                            switch (sub_mode1) {
                                case 0: calibrateXAxis(); break;
                                case 1: calibrateZAxis(); break;
                            }
```

```
                    }
                }
                break;
            case 1:
                while (1) {
                    navigateSub2Menu(&sub_mode2);
                    if (sub_mode2 == 4) break;
                    if (currentStateChange(MENU_BUTTON_PIN)) {
                        switch (sub_mode2) {
                            case 0: setSpeed(1); break;
                            case 1: setSpeed(2); break;
                            case 2: setSpeed(3); break;
                            case 3: setSpeed(4); break;
                        }
                    }
                }
                break;
            case 2: setNumberOfHoles(); break;
            case 3: return;
        }
    }
  }
}

void navigateSub1Menu(int* sub_mode1) {
    lcd_command(0x01);  // Clear display
    lcd_print(sub1_options[*sub_mode1]);
    _delay_ms(100);

    if (currentStateChange(UP_BUTTON_PIN)) {
        *sub_mode1 = (*sub_mode1 + 1) % 3;
    } else if (currentStateChange(DOWN_BUTTON_PIN)) {
        *sub_mode1 = (*sub_mode1 - 1 + 3) % 3;
    }
}

void navigateSub2Menu(int* sub_mode2) {
    lcd_command(0x01);  // Clear display
    lcd_print(sub2_options[*sub_mode2]);
    _delay_ms(100);

    if (currentStateChange(UP_BUTTON_PIN)) {
        *sub_mode2 = (*sub_mode2 + 1) % 5;
```

```
    } else if (currentStateChange(DOWN_BUTTON_PIN)) {
        *sub_mode2 = (*sub_mode2 - 1 + 5) % 5;
    }
}

void calibrateXAxis() {
    // Calibration for X Axis
    lcd_command(0x01);  // Clear display
    lcd_print("Calibrating X Axis");
    _delay_ms(1000);

    lcd_command(0x01); // Clear display
    lcd_print("Enter X value:");
    _delay_ms(2000);

    while (!currentStateChange(CANCEL_PIN)) {
        lcd_command(0x01); // Clear display
        char buffer[16];
        sprintf(buffer, "X value: %d", temp_x_coordinate);
        lcd_print(buffer);

        if (currentStateChange(UP_PIN)) {
            temp_x_coordinate++;
            _delay_ms(200);
        } else if (currentStateChange(DOWN_PIN)) {
            temp_x_coordinate--;
            _delay_ms(200);
        } else if (currentStateChange(MENU_PIN)) {
            lcd_command(0x01); // Clear display
            lcd_print("X coordinate done");
            initial_x_coordinate = temp_x_coordinate;
            _delay_ms(2000);
            break;
        }
    }
}


void calibrateZAxis() {
    // Calibration for Z Axis
    lcd_command(0x01);  // Clear display
    lcd_print("Calibrating Z Axis");
    _delay_ms(1000);
```

```
    lcd_command(0x01); // Clear display
    lcd_print("Enter Z value:");
    _delay_ms(2000);

    while (!currentStateChange(CANCEL_PIN)) {
        lcd_command(0x01); // Clear display
        char buffer[16];
        sprintf(buffer, "Z value: %d", temp_z_coordinate);
        lcd_print(buffer);

        if (currentStateChange(UP_PIN)) {
            temp_z_coordinate++;
            _delay_ms(200);
        } else if (currentStateChange(DOWN_PIN)) {
            temp_z_coordinate--;
            _delay_ms(200);
        } else if (currentStateChange(MENU_PIN)) {
            lcd_command(0x01); // Clear display
            lcd_print("Z coordinate done");
            initial_z_coordinate = temp_z_coordinate;
            _delay_ms(2000);
            break;
        }
    }
}

void setSpeed(int stepper) {
    // Set Speed for a stepper motor
    lcd_command(0x01);  // Clear display
    lcd_print("Set Speed");
    _delay_ms(1000);

    while (!currentStateChange(CANCEL_PIN)) {
        lcd_command(0x01); // Clear display
        lcd_print(sub2_options[current_2_submode]);

        if (currentStateChange(UP_PIN)) {
            _delay_ms(200);
            current_2_submode = (current_2_submode + 1) % max_2_submodes;
        } else if (currentStateChange(DOWN_PIN)) {
            _delay_ms(200);
```

```
            current_2_submode = (current_2_submode - 1 + max_2_submodes) %
max_2_submodes;
        } else if (currentStateChange(MENU_PIN)) {
            _delay_ms(200);
            if (current_2_submode >= 0 && current_2_submode <= 3) {
                setStepperSpeed(current_2_submode + 1);
            } else if (current_2_submode == 4) {
                break;
            }
        }
    }
}

void setStepperSpeed(int stepperNumber) {
    char buffer[16];
    int* temp_speed;
    int* initial_speed;

    switch (stepperNumber) {
        case 1:
            temp_speed = &temp_speed1;
            initial_speed = &initial_speed1;
            break;
        case 2:
            temp_speed = &temp_speed2;
            initial_speed = &initial_speed2;
            break;
        case 3:
            temp_speed = &temp_speed3;
            initial_speed = &initial_speed3;
            break;
        case 4:
            temp_speed = &temp_speed4;
            initial_speed = &initial_speed4;
            break;
    }

    lcd_command(0x01); // Clear display
    sprintf(buffer, "Enter speed%d value:", stepperNumber);
    lcd_print(buffer);
    _delay_ms(2000);

    while (!currentStateChange(CANCEL_PIN)) {
```

```
        lcd_command(0x01); // Clear display
        sprintf(buffer, "speed%d value: %d", stepperNumber, *temp_speed);
        lcd_print(buffer);

        if (currentStateChange(UP_PIN)) {
            (*temp_speed)++;
            _delay_ms(200);
        } else if (currentStateChange(DOWN_PIN)) {
            (*temp_speed)--;
            _delay_ms(200);
        } else if (currentStateChange(MENU_PIN)) {
            lcd_command(0x01); // Clear display
            sprintf(buffer, "speed%d value done", stepperNumber);
            lcd_print(buffer);
            *initial_speed = *temp_speed;
            _delay_ms(2000);
            break;
        }
    }
}

void setNumberOfHoles() {
    // Set Number of Holes
    lcd_command(0x01);  // Clear display
    lcd_print("Set Number of Holes");
    _delay_ms(1000);

    lcd_command(0x01); // Clear display
    lcd_print("Enter number of holes:");
    _delay_ms(2000);

    while (!currentStateChange(CANCEL_PIN)) {
        lcd_command(0x01); // Clear display
        char buffer[16];
        sprintf(buffer, "holes value: %d", temp_holes);
        lcd_print(buffer);

        if (currentStateChange(UP_PIN)) {
            temp_holes++;
            _delay_ms(200);
        } else if (currentStateChange(DOWN_PIN)) {
            temp_holes--;
            _delay_ms(200);
```

```
        } else if (currentStateChange(MENU_PIN)) {
            lcd_command(0x01); // Clear display
            lcd_print("holes value done");
            default_holes = temp_holes;
            _delay_ms(2000);
            break;
        }
    }
}

bool immediate() {
    return !currentStateChange(IMMEDIATE_BUTTON_PIN);
}

// Save current positions to EEPROM
void save_positions() {
    savePositionToEEPROM(eepromAddr_stepper1, savedPosition1);
    savePositionToEEPROM(eepromAddr_stepper2, savedPosition2);
    savePositionToEEPROM(eepromAddr_stepper3, savedPosition3);
    savePositionToEEPROM(eepromAddr_stepper4, savedPosition4);
}

void initializeMotorPins() {
    // Set direction and step pins as output
    DDRD |= (1 << DIR1_PIN) | (1 << STEP1_PIN) | (1 << DIR2_PIN) | (1 << STEP2_PIN)
| (1 << DIR3_PIN) | (1 << STEP3_PIN);
    DDRB |= (1 << DIR4_PIN) | (1 << STEP4_PIN);
}

void initializeMotorPositions() {
    // Initialize motor positions from EEPROM
    savedPosition1 = readPositionFromEEPROM(eepromAddr_stepper1);
    savedPosition2 = readPositionFromEEPROM(eepromAddr_stepper2);
    savedPosition3 = readPositionFromEEPROM(eepromAddr_stepper3);
    savedPosition4 = readPositionFromEEPROM(eepromAddr_stepper4);
}

int main() {
    initializeMotorPins();   // Initialize direction and step pins as output
    setupLCD();              // Initialize LCD
    setupButtonPins();       // Initialize buttons

    initializeMotorPositions();  // Initialize motor positions from EEPROM
```

```
    loop();     // Execute the main operation

    return 0;
}
```

7. Project Progression Images

a) PCB

Bare PCB

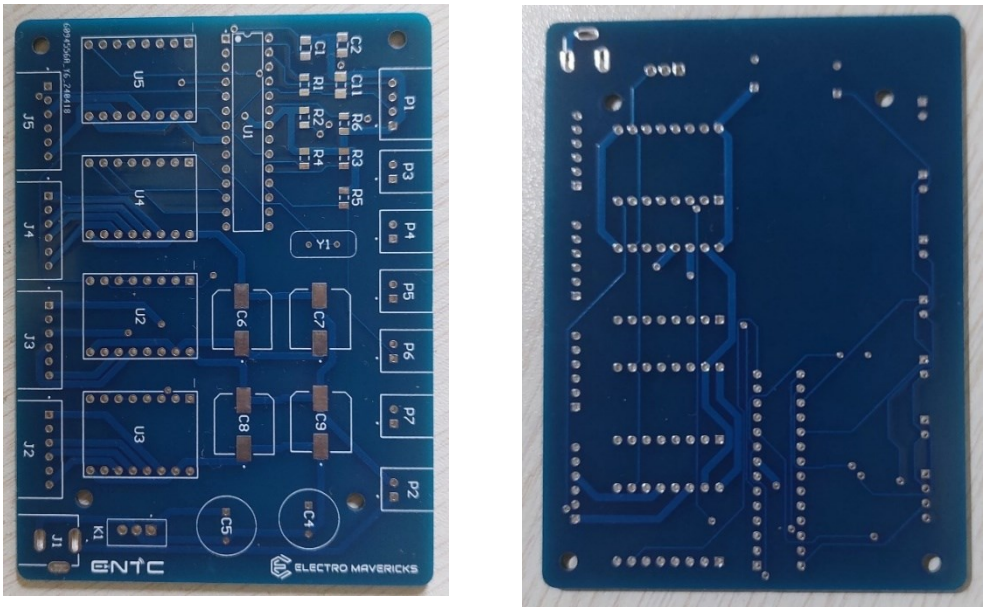


Figure 21 – Bare PCB front and back view

Soldered PCB

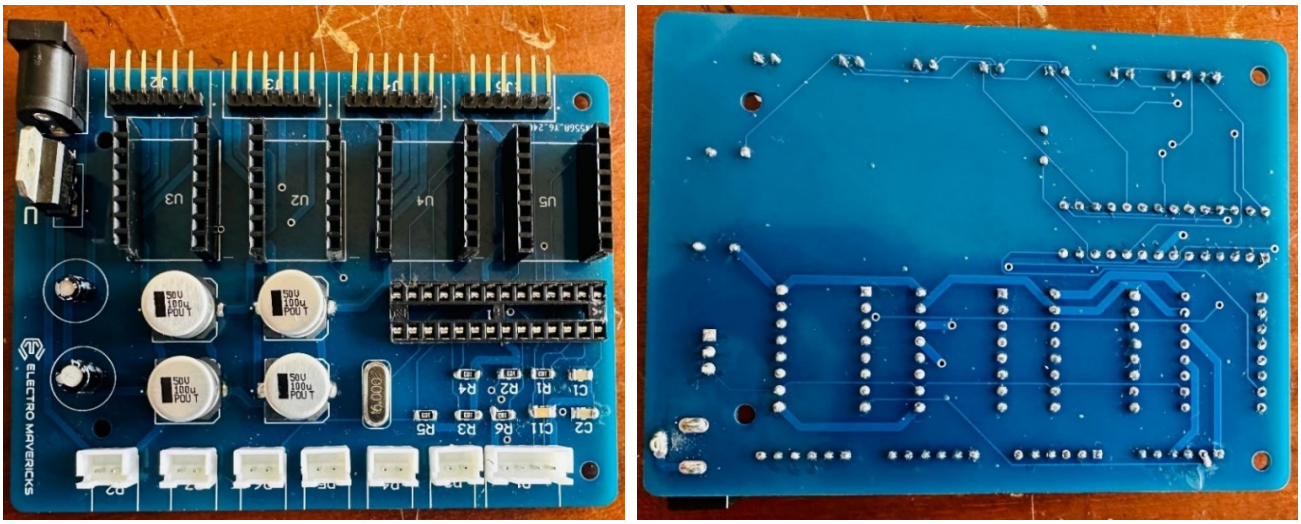


Figure 22 –Soldered PCB front and back view

PCB Testing

Figure 23 –PCB testing

b) Physical Components

Enclosure

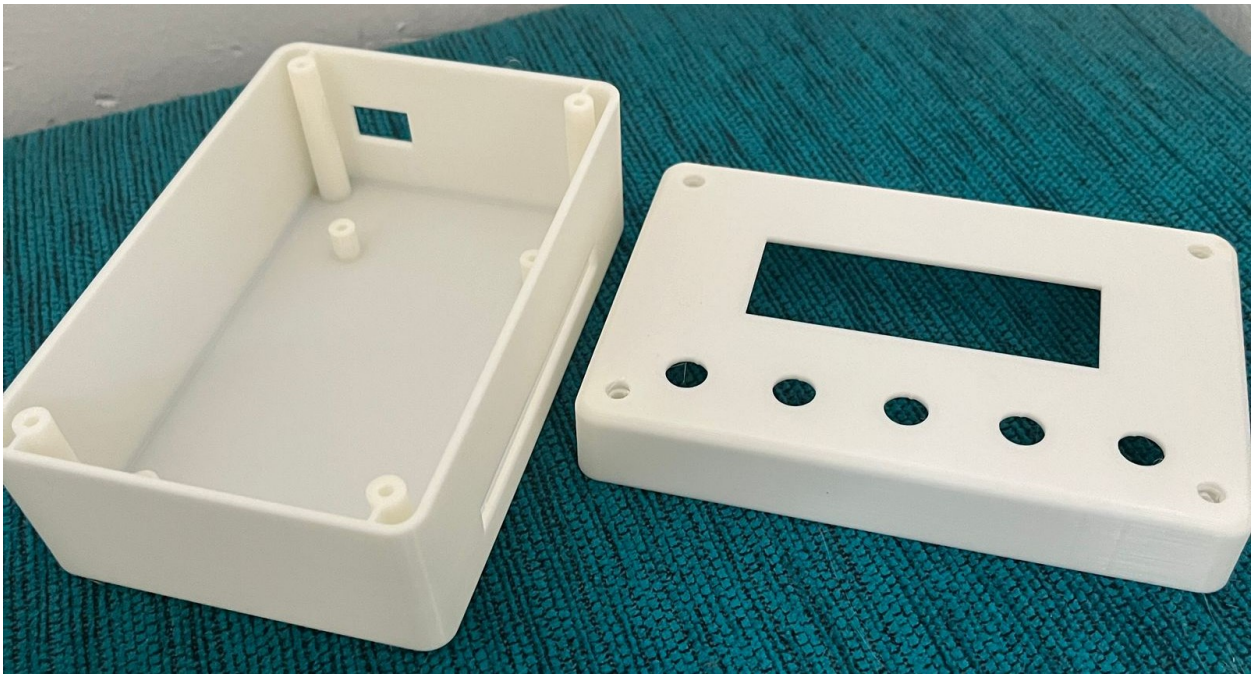


Figure 24 –Printed enclosure lid and box view

Gripper Arm Parts

Figure 25 –Printed gripper arm parts

c) Assembled Physical Preview

Enclosure

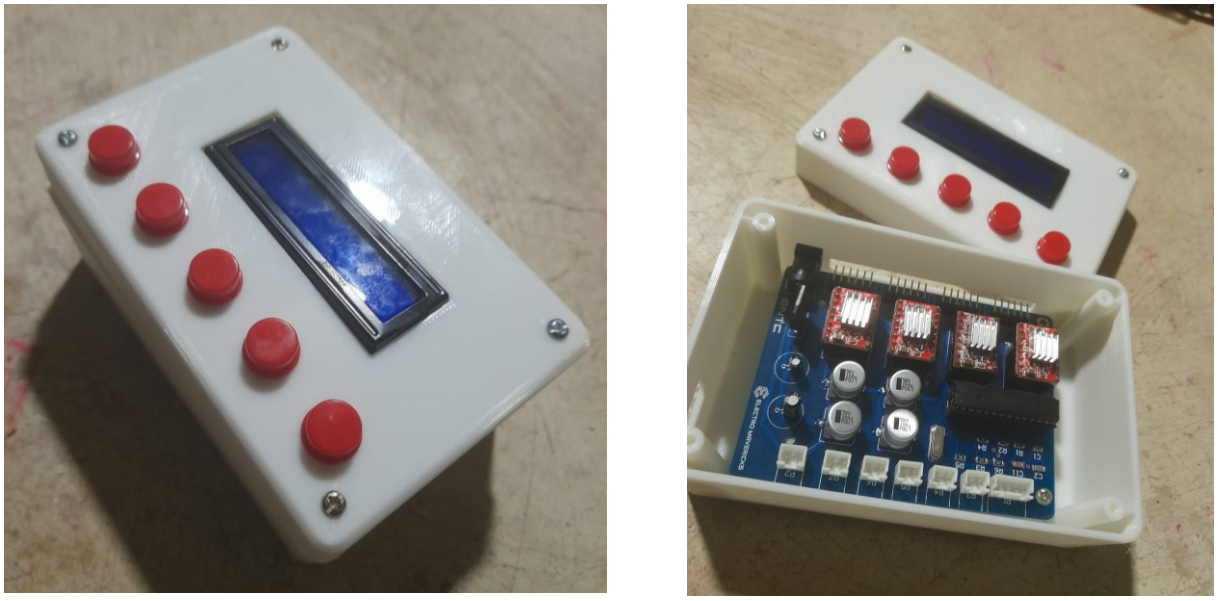


Figure 26 –Assembled enclosure close and open view

Sliding Parts with arm

Figure 27 – arm and slider assembled view

8. References

- "Automated Assembly of Power Electronics Modules Using a Pick-and-Place Robot" by P. Zhou et al. (2018)
- "Design and Development of a Flexible Pick-and-Place Robot for Electronics Assembly" by S.M.A. Hashmi et al. (2015)
- "Automated Assembly of High-Density TRANSISTOR Arrays for Motor Drives" by Y. Liu et al. (2020)
- "A Novel Automated Assembly System for High-Power TRANSISTOR Modules" by IEEE (Institute of Electrical and Electronics Engineers) Xplore
- Atmel Corporation, "8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash," Atmel Datasheets, 2016. [Online]. Available: https://www.microchip.com/design-centers/8-bit.
- Microchip Technology Inc., "AVR and ARM Toolchains & C Compilers," Microchip, 2020. [Online]. Available: https://www.microchip.com/mplab/avr-support/avr-and-arm-toolchains-c-compilers.
- Bruce R. Land, "AVR Programming Tutorials," YouTube, 2011. [Online]. Available: https://www.youtube.com/playlist?list=PLD7F7ED1F3505D8D5.
- B. Roth, "AVRDUDE User Manual," Free Software Foundation, 2018. [Online]. Available: https://www.nongnu.org/avrdude/user-manual/avrdude.html.
- Atmel Corporation, "ATMEGA328P Datasheet," Atmel, 2016. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328P-DS-DS40002061A.pdf.
- Allegro MicroSystems, "A4988 Datasheet," Allegro MicroSystems, 2016. [Online]. Available: https://www.allegromicro.com/en/Products/Motor-Driver-And-Interface-ICs/Bipolar-Stepper-Motor-Drivers/A4988.
- MinebeaMitsumi Inc., "NEMA 17 Stepper Motor Data Sheet," MinebeaMitsumi Inc., 2015. [Online]. Available: https://www.minebeamitsumi.com/english/product/rotary/download/nema17.pdf.

Appendix

1) Log Entries

A.1 26 February - 3 March

A.1.1 Research Phase

- **Conducted Research**: Investigated existing automated systems focused on metal ring placement and rotation specifically in H-Bridge assemblies.
- **Industry Standards**: Evaluated the current industry standards and the latest technologies for achieving high precision in automated assembly processes.
- **System Design Decision**: Based on insights gathered, decided to develop a system incorporating advanced robotic arms, sliders, and vision systems to ensure precision and efficiency.

A.2 4 March - 10 March

A.2.1 Conceptualization Phase

- **Brainstorming Sessions**: Held multiple brainstorming sessions to generate innovative ideas for the precise placement and rotation of metal rings within H-Bridge assemblies.
- **Concept Development**: Developed four different conceptual designs, each evaluated for its precision, scalability, cost-effectiveness, and technical complexity.
- **Concept Selection**: After thorough analysis, the most promising design was selected for further development based on an evaluation framework.

A.3 11 March - 17 March

A.3.1 Design Evaluation Phase

- **Detailed Evaluation**: Conducted a detailed evaluation of the selected conceptual designs considering technical feasibility, scalability, and alignment with the project's objectives.
- **Structured Assessment**: Created a structured assessment method to ensure an objective and comprehensive evaluation of each design.
- **Final Design Selection**: The best conceptual design was selected through a consensus among team members after an in-depth analysis.

A.4 18 March - 24 March

A.4.1 Equipment Selection Phase

- **Equipment Analysis**: Analyzed various options for robotic arms, sliders, and gripper mechanisms in terms of their efficiency, precision, and compatibility with the overall system design.
- **Equipment Finalization**: Finalized the most suitable equipment through collaborative discussions and technical evaluations, ensuring they meet the required specifications.
- **Project Plan Development**: Developed a detailed project plan outlining key milestones, tasks, and timelines to guide the project's progression.

- **Design Specification Definition**: Defined comprehensive design specifications, including technical requirements and performance benchmarks, to steer the development process.

A.5 25 March - 31 March

A.5.1 Design Phase

- **Prototype Design Initiation**: Started the design phase by creating a prototype of the metal ring placement system using SolidWorks software.
- **Feature Visualization**: Visualized and validated key features and functionalities of the conceptual design through the prototype.
- **PCB Design**: Designed a printed circuit board (PCB) for the control system and the selected robotic arm, laying the foundation for integration and operation.

A.6 1 April - 7 April

A.6.1 Prototype Assembly

- **Prototype Assembly**: Assembled the prototype, including the robotic arm and sliders, to test and refine the design.
- **Initial Experiments**: Conducted initial experiments with motors to assess movement precision and stability, crucial for the system's success.
- **PCB Testing**: Tested the soldered PCB to ensure proper functionality and seamless integration with the control system.

A.7 8 April - 14 April

A.7.1 Design Refinement

- **Iterative Refinement**: Engaged in iterative refinement of the prototype, incorporating feedback from initial tests and reviews to improve the design.
- **Feasibility Assessments**: Conducted feasibility assessments to identify and address any design limitations, ensuring the prototype met all performance requirements.
- **Performance Optimization**: Ensured that the prototype met all specified performance benchmarks and addressed any identified limitations to enhance overall design efficacy.

A.8 15 April - 21 April

A.8.1 System Enclosure Design

- **Enclosure Design**: Designed an enclosure to house the PCB and other control components, ensuring it included a user interface for operational control.
- **Accessibility and Maintenance**: Ensured the enclosure design allowed for easy access and maintenance, facilitating user interaction and system upkeep.

A.9 22 April - 28 April

A.9.1 Final Design Adjustments

- **Design Documentation Finalization**: Finalized detailed design documentation for the metal ring placement system, encompassing all aspects of the design.
- **Project Review Preparation**: Prepared for the upcoming evaluation by reviewing all project elements to ensure completeness and alignment with project goals.
- **Stakeholder Alignment**: Ensured that all design elements met stakeholder requirements and were aligned with the project's objectives and goals.

A.10 29 April - 5 May

A.10.1 Review and Testing

- **Prototype Assembly Completion**: Completed the assembly of the prototype, ensuring all components were correctly integrated.
- **Internal Reviews**: Conducted internal reviews of the design documentation and the prototype to identify any areas for improvement.
- **Peer Review Sessions**: Engaged in peer review sessions to gather additional feedback and make necessary adjustments to the design.
- **System Testing**: Tested the metal ring placement system comprehensively to ensure it met all operational specifications and performance benchmarks.

A.11 6 May - 12 May

A.11.1 System Enclosure Finalization

- **Enclosure Design Finalization**: Finalized the design of the system enclosure using SolidWorks, ensuring it met all design and operational requirements.
- **3D Printing**: 3D printed the enclosure and integrated it with the user interface, PCB, and control system to complete the housing assembly.

A.12 13 May - 19 May

A.12.1 Final Report Preparation

- **Documentation Compilation**: Compiled all project documentation, including detailed design reports, test results, and feedback from prototype demonstrations.
- **Project Report Finalization**: Finalized the project report, ensuring it provided comprehensive details about the design process, challenges encountered, and solutions implemented throughout the project.

A.13 30 June - 08 July

A.13.1 Studied AVR Programming

- Reviewed instructional videos by the professor and other AVR programming resources.
- Implemented several AVR programs for practice.

A.13.2 Refined and Rewrote Code

- Improved and revised the code based on feedback and project requirements.
- Modularized the existing codebase in C++.
- Finalized the code for the microcontroller unit (MCU).

A.13.3 Updated Design Calculations

- Enhanced the accuracy of design-related calculations.

A.13.4 Prepared Final Reports with Revisions

- Completed final reports with necessary revisions.
- Integrated feedback provided by the professor.
- Included progress visuals and relevant calculations.

2). Declaration

In our previous submission, we have incorporated Arduino libraries and utilized Arduino-style coding to implement various functionalities within the project.

In previous report we included below code for check stepper motors functionality.

```
#include <AccelStepper.h>

// Define pins numbers
const int stepPin = 8;
const int dirPin = 9;

// Create a new instance of the AccelStepper class
AccelStepper stepper(AccelStepper::DRIVER, stepPin, dirPin);

void setup() {
  // Set the maximum speed and acceleration in steps per second and steps per
second^2
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(500);
}

void loop() {
  // Move the stepper motor 200 steps in the forward direction
  stepper.moveTo(200);
  stepper.runToPosition();

  // Wait for 1 second
  delay(1000);

  // Move the stepper motor 200 steps in the reverse direction
  stepper.moveTo(0);
  stepper.runToPosition();

  // Wait for 1 second
  delay(1000);
}
```

3). Document Reviews

Date	Index	Name with Initials	Signature
07/07/2023	210179R	Gammune D.J.P.	
07/07/2023	210285M	Kavishan G.P	
07/07/2023	210079K	Charles. J	
07/07/2023	210054F	Atapattu A.M.L.R	

4). Datasheets

Stepper Motor NEMA 17

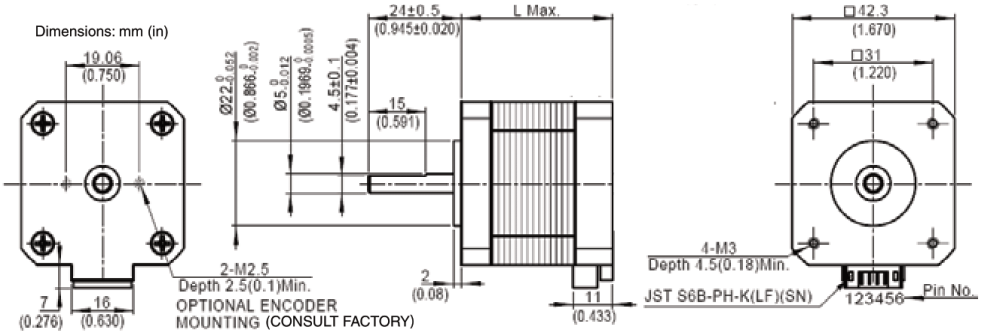
This document describes mechanical and electrical specifications for PBC Linear stepper motors; including standard, hollow, and extended shaft variations.

Standard shaft motor shown.

Phases	2
Steps/Revolution	200
Step Accuracy	±5%
Shaft Load	20,000 Hours at 1000 RPM
Axial	25 N (5.6 lbs.) Push
	65 N (15 lbs.) Pull
Radial	29 N (6.5 lbs.) At Flat Center
IP Rating	40
Approvals	RoHS
Operating Temp	-20° C to +40° C
Insulation Class	B, 130° C
Insulation Resistance	100 MegOhms

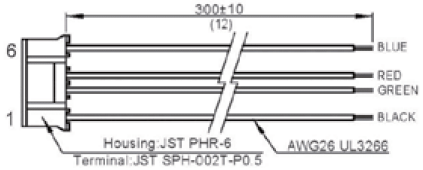
Description	Length	Mounted Rated Current	Mounted Holding Torque		Winding Ohms mH		Detent Torque		Rotor Inertia		Motor Weight	
(Stack)	"L" Max	Amps	Nm Typ.	oz-in Typ.	±10% @ 20°C	Typ.	mNm	oz-in	g cm2	oz-in2	kg	lbs
Single	39.8 mm (1.57 in)	2	0.48	68	1.04	2.2	15	2.1	57	0.31	0.28	0.62
Double	48.3 mm (1.90 in)	2	0.63	89	1.3	2.9	25	3.5	82	0.45	0.36	0.79
Triple	62.8 mm (2.47 in)	2	0.83	120	1.49	3.8	30	4.2	123	0.67	0.6	1.3

*All standard motors have plug connector. Consult factory for other options.



Standard shaft dimensions shown. All other dimensions apply to hollow and extended shaft options.

Dimensions: mm (in)
4 Lead Connector, PBC Part#6200490
(Consult factory for optional motor connectors)

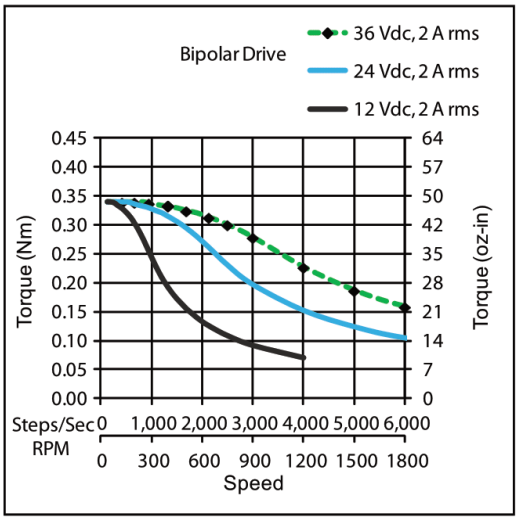


PBC Linear

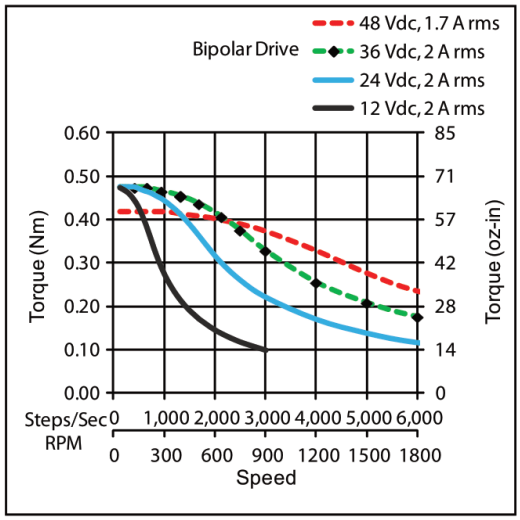
Engineering Your Linear Motion Solutions • pbclinear.com

NEMA 17 **Stepper Motor**

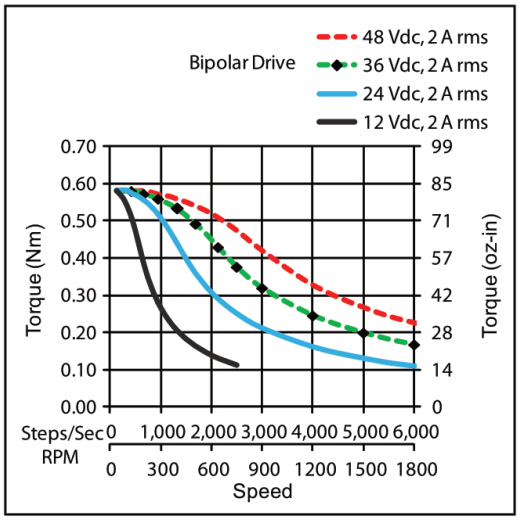
*Performance curves apply to continuous duty cycles.
Consult factory for intermittent cycles or other voltages.



Single Stack



Double Stack



Triple Stack

PBC Linear

Engineering Your Linear Motion Solutions • pbclinear.com

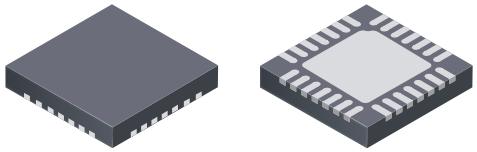
DMOS Microstepping Driver with Translator And Overcurrent Protection

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $^1/_2$, $^1/_4$, $^1/_8$, and $^1/_{16}$

Package:

28-contact QFN
with exposed thermal pad
5 mm × 5 mm × 0.90 mm
(ET package)



Approximate size

Description

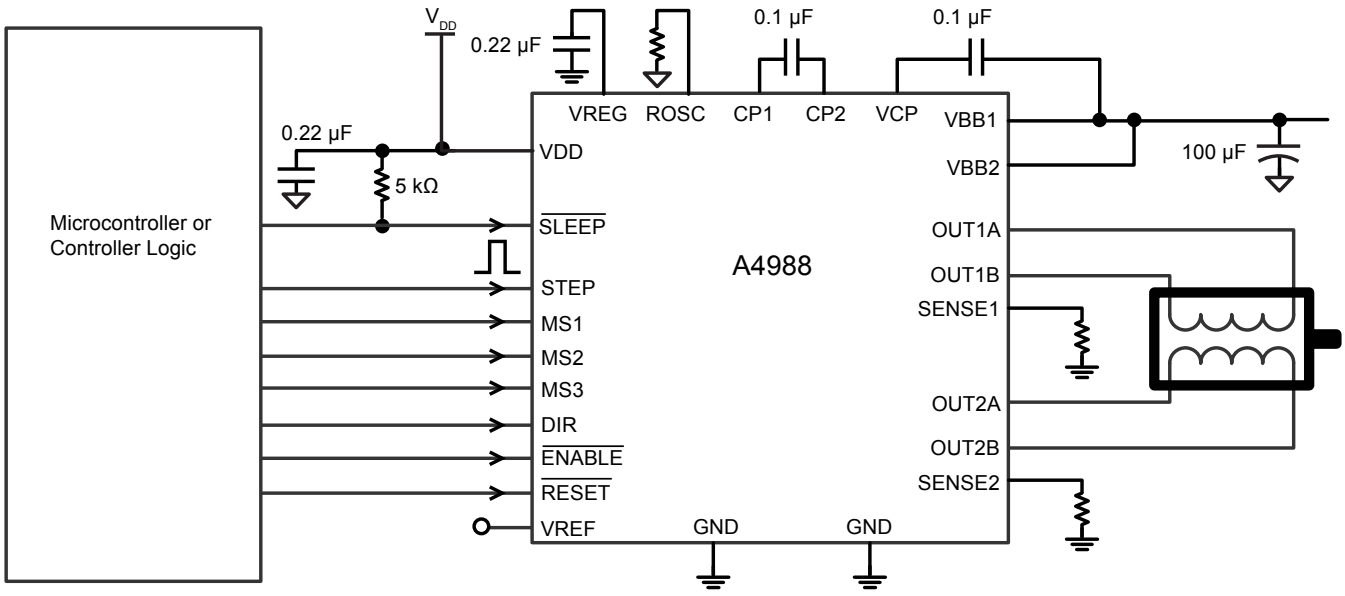
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ±2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

Continued on the next page…

Typical Application Diagram



Description (continued)

Internal synchronous rectification control circuitry is provided to improve power dissipation during PWM operation. Internal circuit protection includes: thermal shutdown with hysteresis, undervoltage lockout (UVLO), and crossover-current protection. Special power-on sequencing is not required.

The A4988 is supplied in a surface mount QFN package (ES), 5 mm × 5 mm, with a nominal overall package height of 0.90 mm and an exposed pad for enhanced thermal dissipation. It is lead (Pb) free (suffix –T), with 100% matte tin plated leadframes.

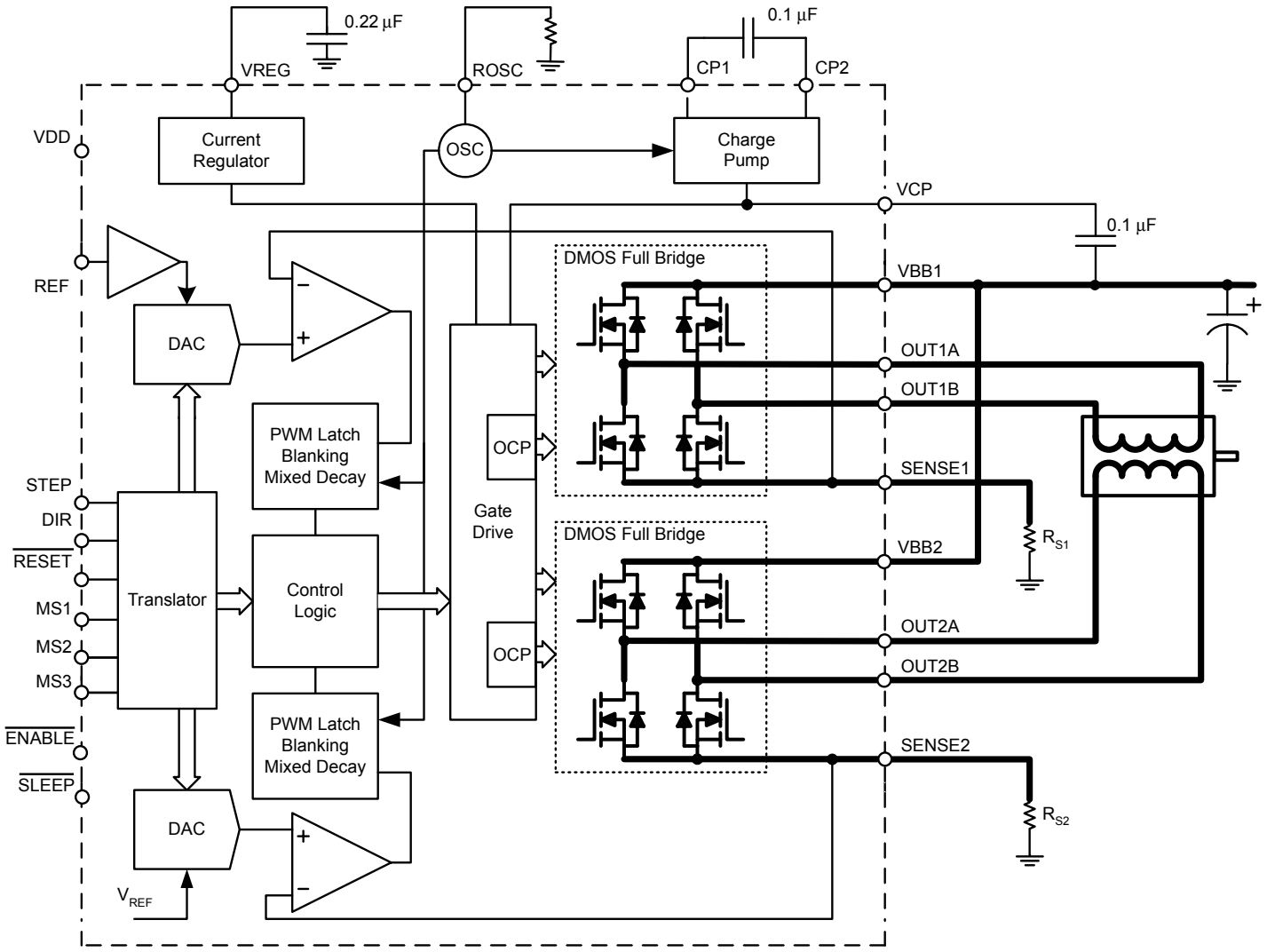
Selection Guide

Part Number	Package	Packing
A4988SETTR-T	28-contact QFN with exposed thermal pad	1500 pieces per 7-in. reel

Absolute Maximum Ratings

Characteristic	Symbol	Notes	Rating	Units
Load Supply Voltage	V_{BB}		35	V
Output Current	I_{OUT}		±2	A
Logic Input Voltage	V_{IN}		–0.3 to 5.5	V
Logic Supply Voltage	V_{DD}		–0.3 to 5.5	V
Motor Outputs Voltage			–2.0 to 37	V
Sense Voltage	V_{SENSE}		–0.5 to 0.5	V
Reference Voltage	V_{REF}		5.5	V
Operating Ambient Temperature	T_A	Range S	–20 to 85	ºC
Maximum Junction	T_J(max)		150	ºC
Storage Temperature	T_{stg}		–55 to 150	ºC

Functional Block Diagram



ELECTRICAL CHARACTERISTICS[1] at T_A = 25°C, V_{BB} = 35 V (unless otherwise noted)

Characteristics	Symbol	Test Conditions	Min.	Typ.[2]	Max.	Units
Output Drivers						
Load Supply Voltage Range	V_{BB}	Operating	8	–	35	V
Logic Supply Voltage Range	V_{DD}	Operating	3.0	–	5.5	V
Output On Resistance	R_{DSON}	Source Driver, I_{OUT} = –1.5 A	–	320	430	mΩ
		Sink Driver, I_{OUT} = 1.5 A	–	320	430	mΩ
Body Diode Forward Voltage	V_F	Source Diode, I_F = –1.5 A	–	–	1.2	V
		Sink Diode, I_F = 1.5 A	–	–	1.2	V
Motor Supply Current	I_{BB}	f_{PWM} < 50 kHz	–	–	4	mA
		Operating, outputs disabled	–	–	2	mA
Logic Supply Current	I_{DD}	f_{PWM} < 50 kHz	–	–	8	mA
		Outputs off	–	–	5	mA
Control Logic						
Logic Input Voltage	$V_{IN(1)}$		V_{DD}×0.7	–	–	V
	$V_{IN(0)}$		–	–	V_{DD}×0.3	V
Logic Input Current	$I_{IN(1)}$	V_{IN} = V_{DD}×0.7	–20	<1.0	20	µA
	$I_{IN(0)}$	V_{IN} = V_{DD}×0.3	–20	<1.0	20	µA
Microstep Select	R_{MS1}	MS1 pin	–	100	–	kΩ
	R_{MS2}	MS2 pin	–	50	–	kΩ
	R_{MS3}	MS3 pin	–	100	–	kΩ
Logic Input Hysteresis	$V_{HYS(IN)}$	As a % of V_{DD}	5	11	19	%
Blank Time	t_{BLANK}		0.7	1	1.3	µs
Fixed Off-Time	t_{OFF}	OSC = VDD or GND	20	30	40	µs
		R_{OSC} = 25 kΩ	23	30	37	µs
Reference Input Voltage Range	V_{REF}		0	–	4	V
Reference Input Current	I_{REF}		–3	0	3	µA
Current Trip-Level Error[3]	err_I	V_{REF} = 2 V, %$I_{TripMAX}$ = 38.27%	–	–	±15	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 70.71%	–	–	±5	%
		V_{REF} = 2 V, %$I_{TripMAX}$ = 100.00%	–	–	±5	%
Crossover Dead Time	t_{DT}		100	475	800	ns
Protection						
Overcurrent Protection Threshold[4]	I_{OCPST}		2.1	–	–	A
Thermal Shutdown Temperature	T_{TSD}		–	165	–	°C
Thermal Shutdown Hysteresis	T_{TSDHYS}		–	15	–	°C
VDD Undervoltage Lockout	V_{DDUVLO}	V_{DD} rising	2.7	2.8	2.9	V
VDD Undervoltage Hysteresis	$V_{DDUVLOHYS}$		–	90	–	mV

[1]For input and output current specifications, negative current is defined as coming out of (sourcing) the specified device pin.

[2]Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits.

[3]V_{ERR} = [(V_{REF}/8) – V_{SENSE}] / (V_{REF}/8).

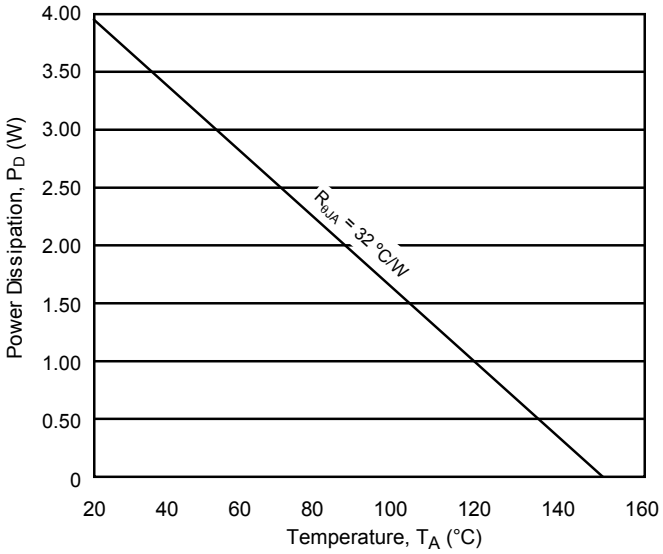
[4]Overcurrent protection (OCP) is tested at T_A = 25°C in a restricted range and guaranteed by characterization.

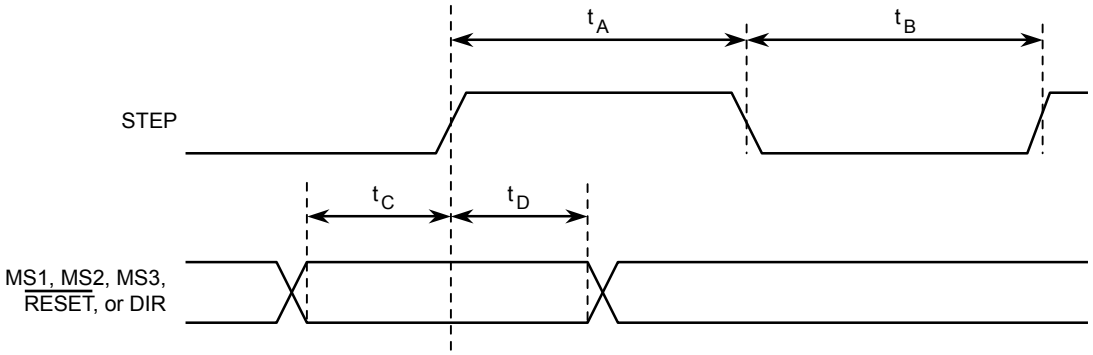
THERMAL CHARACTERISTICS

Characteristic	Symbol	Test Conditions*	Value	Units
Package Thermal Resistance	$R_{\theta JA}$	Four-layer PCB, based on JEDEC standard	32	ºC/W

*Additional thermal information available on Allegro Web site.

Power Dissipation versus Ambient Temperature





Time Duration	Symbol	Typ.	Unit
STEP minimum, HIGH pulse width	t_A	1	µs
STEP minimum, LOW pulse width	t_B	1	µs
Setup time, input change to STEP	t_C	200	ns
Hold time, input change to STEP	t_D	200	ns

Figure 1: Logic Interface Timing Diagram

Table 1: Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Functional Description

Device Operation. The A4988 is a complete microstepping motor driver with a built-in translator for easy operation with minimal control lines. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth, and sixteenth-step modes. The currents in each of the two output full-bridges and all of the N-channel DMOS FETs are regulated with fixed off-time PWM (pulse width modulated) control circuitry. At each step, the current for each full-bridge is set by the value of its external current-sense resistor (R_{S1} and R_{S2}), a reference voltage (V_{REF}), and the output voltage of its DAC (which in turn is controlled by the output of the translator).

At power-on or reset, the translator sets the DACs and the phase current polarity to the initial Home state (shown in Figures 9 through 13), and the current regulator to Mixed Decay Mode for both phases. When a step command signal occurs on the STEP input, the translator automatically sequences the DACs to the next level and current polarity. (See Table 2 for the current-level sequence.) The microstep resolution is set by the combined effect of the MSx inputs, as shown in Table 1.

When stepping, if the new output levels of the DACs are lower than their previous output levels, then the decay mode for the active full-bridge is set to Mixed. If the new output levels of the DACs are higher than or equal to their previous levels, then the decay mode for the active full-bridge is set to Slow. This automatic current decay selection improves microstepping performance by reducing the distortion of the current waveform that results from the back EMF of the motor.

Microstep Select (MSx). The microstep resolution is set by the voltage on logic inputs MSx, as shown in Table 1. The MS1 and MS3 pins have a 100 kΩ pull-down resistance, and the MS2 pin has a 50 kΩ pull-down resistance. When changing the step mode the change does not take effect until the next STEP rising edge.

If the step mode is changed without a translator reset, and absolute position must be maintained, it is important to change the step mode at a step position that is common to both step modes in order to avoid missing steps. When the device is powered down, or reset due to TSD or an over current event the translator is set to the home position which is by default common to all step modes.

Mixed Decay Operation. The bridge operates in Mixed decay mode, at power-on and reset, and during normal running according to the ROSC configuration and the step sequence, as shown in Figures 9 through 13. During Mixed decay, when the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF}. After that, it switches to Slow decay mode for the remainder of t_{OFF}. A timing diagram for this feature appears on the next page.

Typically, mixed decay is only necessary when the current in the winding is going from a higher value to a lower value as determined by the state of the translator. For most loads automatically-selected mixed decay is convenient because it minimizes ripple when the current is rising and prevents missed steps when the current is falling. For some applications where microstepping at very low speeds is necessary, the lack of back EMF in the winding causes the current to increase in the load quickly, resulting in missed steps. This is shown in Figure 2. By pulling the ROSC pin to ground, mixed decay is set to be active 100% of the time, for both rising and falling currents, and prevents missed steps as shown in Figure 3. If this is not an issue, it is recommended that automatically-selected mixed decay be used, because it will produce reduced ripple currents. Refer to the Fixed Off-Time section for details.

Low Current Microstepping. Intended for applications where the minimum on-time prevents the output current from regulating to the programmed current level at low current steps. To prevent this, the device can be set to operate in Mixed decay mode on both rising and falling portions of the current waveform. This feature is implemented by shorting the ROSC pin to ground. In this state, the off-time is internally set to 30 µs.

Reset Input ($\overline{\text{RESET}}$). The $\overline{\text{RESET}}$ input sets the translator to a predefined Home state (shown in Figures 9 through 13), and turns off all of the FET outputs. All STEP inputs are ignored until the $\overline{\text{RESET}}$ input is set to high.

Step Input (STEP). A low-to-high transition on the STEP

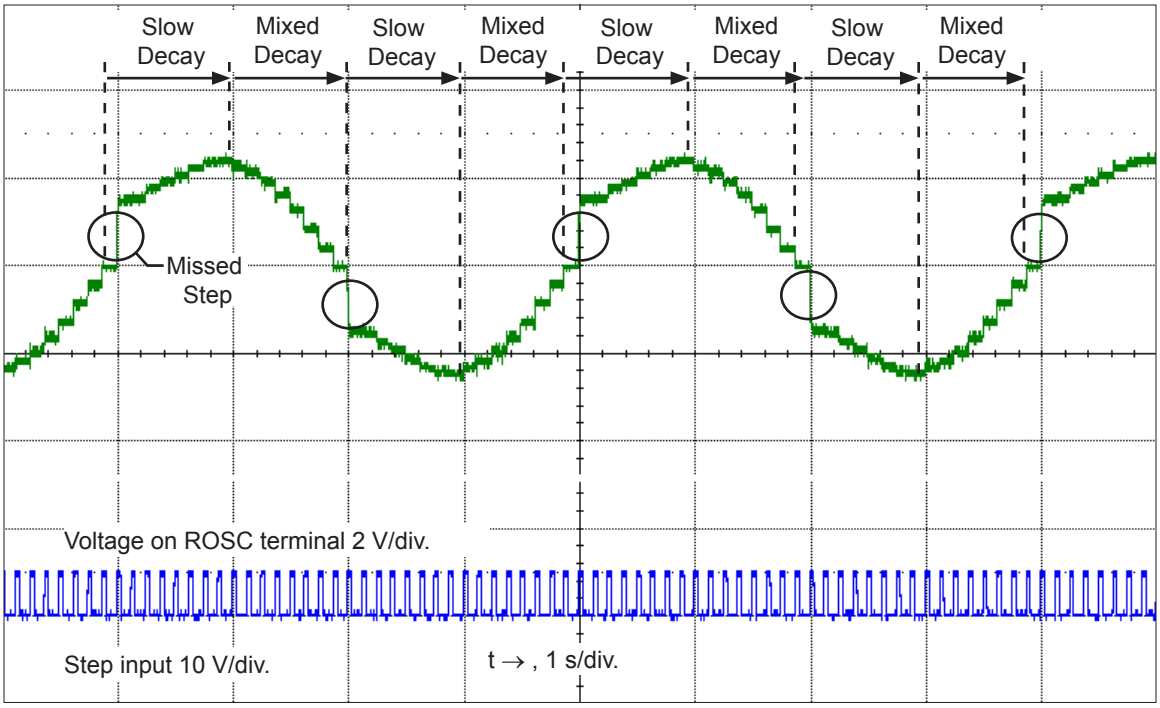


Figure 2: Missed Steps in Low-Speed Microstepping

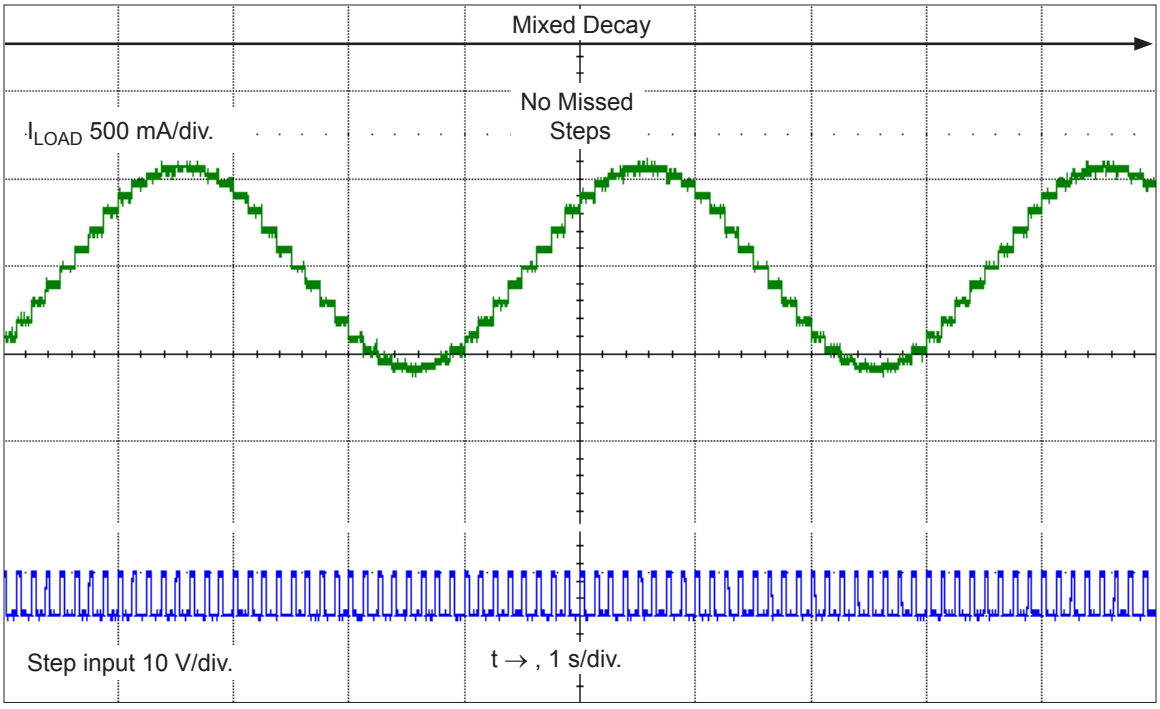


Figure 3: Continuous Stepping Using Automatically-Selected Mixed Stepping (ROSC pin grounded)

input sequences the translator and advances the motor one increment. The translator controls the input to the DACs and the direction of current flow in each winding. The size of the increment is determined by the combined state of the MSx inputs.

Direction Input (DIR). This determines the direction of rotation of the motor. Changes to this input do not take effect until the next STEP rising edge.

Internal PWM Current Control. Each full-bridge is controlled by a fixed off-time PWM current control circuit that limits the load current to a desired value, I_{TRIP}. Initially, a diagonal pair of source and sink FET outputs are enabled and current flows through the motor winding and the current sense resistor, R_{Sx}. When the voltage across R_{Sx} equals the DAC output voltage, the current sense comparator resets the PWM latch. The latch then turns off the appropriate source driver and initiates a fixed off time decay mode

The maximum value of current limiting is set by the selection of R_{Sx} and the voltage at the VREF pin. The transconductance function is approximated by the maximum value of current limiting, $I_{TripMAX}$ (A), which is set by

$$I_{TripMAX} = V_{REF} / (8 \times R_S)$$

where R_S is the resistance of the sense resistor (Ω) and V_{REF} is the input voltage on the REF pin (V).

The DAC output reduces the V_{REF} output to the current sense comparator in precise steps, such that

$$I_{trip} = (\%I_{TripMAX} / 100) \times I_{TripMAX}$$

(See Table 2 for $\%I_{TripMAX}$ at each step.)

It is critical that the maximum rating (0.5 V) on the SENSE1 and SENSE2 pins is not exceeded.

Fixed Off-Time. The internal PWM current control circuitry uses a one-shot circuit to control the duration of time that the DMOS FETs remain off. The off-time, t_{OFF}, is determined by the ROSC terminal. The ROSC terminal has three settings:

- ROSC tied to VDD — off-time internally set to 30 µs, decay mode is automatic Mixed decay except when in full step where decay mode is set to Slow decay
- ROSC tied directly to ground — off-time internally set to 30 µs, current decay is set to Mixed decay for both increasing and decreasing currents for all step modes.
- ROSC through a resistor to ground — off-time is determined by the following formula, the decay mode is automatic Mixed decay for all step modes except full step whic is set to slow decay.

$$t_{OFF} \approx R_{OSC} / 825$$

Where t_{OFF} is in µs.

Blanking. This function blanks the output of the current sense comparators when the outputs are switched by the internal current control circuitry. The comparator outputs are blanked to prevent false overcurrent detection due to reverse recovery currents of the clamp diodes, and switching transients related to the capacitance of the load. The blank time, t_{BLANK} (µs), is approximately

$$t_{BLANK} \approx 1\ \mu s$$

Shorted-Load and Short-to-Ground Protection. If the motor leads are shorted together, or if one of the leads is shorted to ground, the driver will protect itself by sensing the overcurrent event and disabling the driver that is shorted, protecting the device from damage. In the case of a short-to-ground, the device will remain disabled (latched) until the $\overline{\text{SLEEP}}$ input goes high or VDD power is removed. A short-to-ground overcurrent event is shown in Figure 4.

When the two outputs are shorted together, the current path is through the sense resistor. After the blanking time (≈1 µs) expires, the sense resistor voltage is exceeding its trip value, due to the overcurrent condition that exists. This causes the driver to go into a fixed off-time cycle. After the fixed off-time expires the driver turns on again and the process repeats. In this condition the driver is completely protected against overcurrent events, but the short is repetitive with a period equal to the fixed off-time of the driver. This condition is shown in Figure 5.

During a shorted load event it is normal to observe both a positive and negative current spike as shown in Figure 3, due to the direction change implemented by the Mixed decay feature. This is shown in Figure 6. In both instances the overcurrent circuitry is protecting the driver and prevents damage to the device.

Charge Pump (CP1 and CP2). The charge pump is used to generate a gate supply greater than that of VBB for driving the source-side FET gates. A 0.1 µF ceramic capacitor, should be connected between CP1 and CP2. In addition, a 0.1 µF ceramic capacitor is required between VCP and VBB, to act as a reservoir for operating the high-side FET gates.

Capacitor values should be Class 2 dielectric ±15% maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

V_{REG} (VREG). This internally-generated voltage is used to operate the sink-side FET outputs. The nominal output voltage of the VREG terminal is 7 V. The VREG pin must be decoupled with a 0.22 µF ceramic capacitor to ground. V_{REG} is internally monitored. In the case of a fault condition, the FET outputs of the A4988 are disabled.

Capacitor values should be Class 2 dielectric ±15% maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

Enable Input ($\overline{\text{ENABLE}}$). This input turns on or off all of the FET outputs. When set to a logic high, the outputs are disabled. When set to a logic low, the internal control enables the outputs as required. The translator inputs STEP, DIR, and MSx, as well as the internal sequencing logic, all remain active, independent of the $\overline{\text{ENABLE}}$ input state.

Shutdown. In the event of a fault, overtemperature (excess T_J) or an undervoltage (on VCP), the FET outputs of the A4988 are disabled until the fault condition is removed. At power-on, the UVLO (undervoltage lockout) circuit disables the FET outputs and resets the translator to the Home state.

Sleep Mode ($\overline{\text{SLEEP}}$). To minimize power consumption when the motor is not in use, this input disables much of the internal circuitry including the output FETs, current regulator, and charge pump. A logic low on the $\overline{\text{SLEEP}}$ pin puts the A4988 into Sleep mode. A logic high allows normal operation, as well as start-up (at which time the A4988 drives the motor to the Home microstep position). When emerging from Sleep mode, in order to allow the charge pump to stabilize, provide a delay of 1 ms before issuing a Step command.

Mixed Decay Operation. The bridge operates in Mixed Decay mode, depending on the step sequence, as shown in Figures 9 through 13. As the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF}. After that, it switches to Slow Decay mode for the remainder of t_{OFF}. A timing diagram for this feature appears in Figure 7.

Synchronous Rectification. When a PWM-off cycle is triggered by an internal fixed-off time cycle, load current recirculates according to the decay mode selected by the control logic. This synchronous rectification feature turns on the appropriate FETs during current decay, and effectively shorts out the body diodes with the low FET $R_{DS(ON)}$. This reduces power dissipation significantly, and can eliminate the need for external Schottky diodes in many applications. Synchronous rectification turns off when the load current approaches zero (0 A), preventing reversal of the load current.

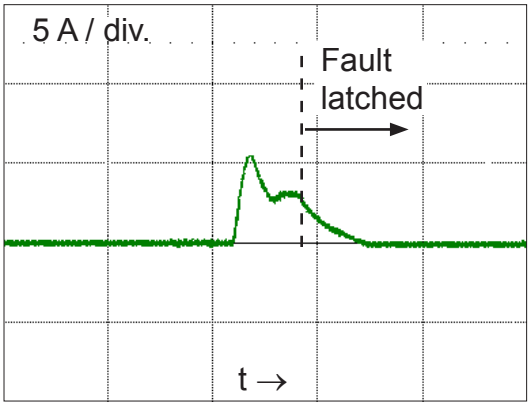


Figure 4: Short-to-Ground Event

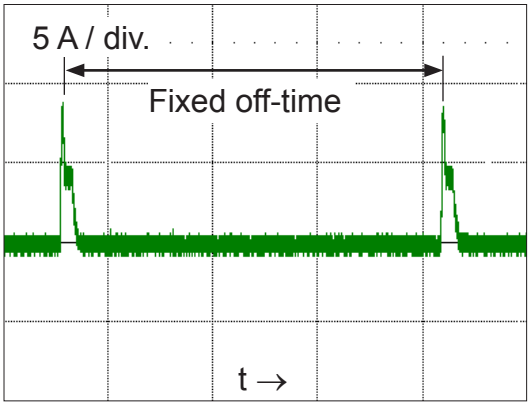


Figure 5. Shorted Load (OUTxA → OUTxB) in Slow Decay Mode

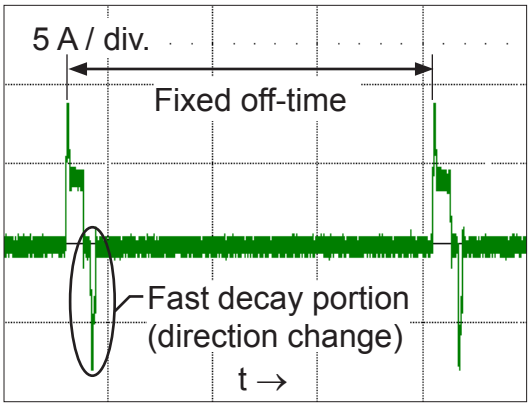
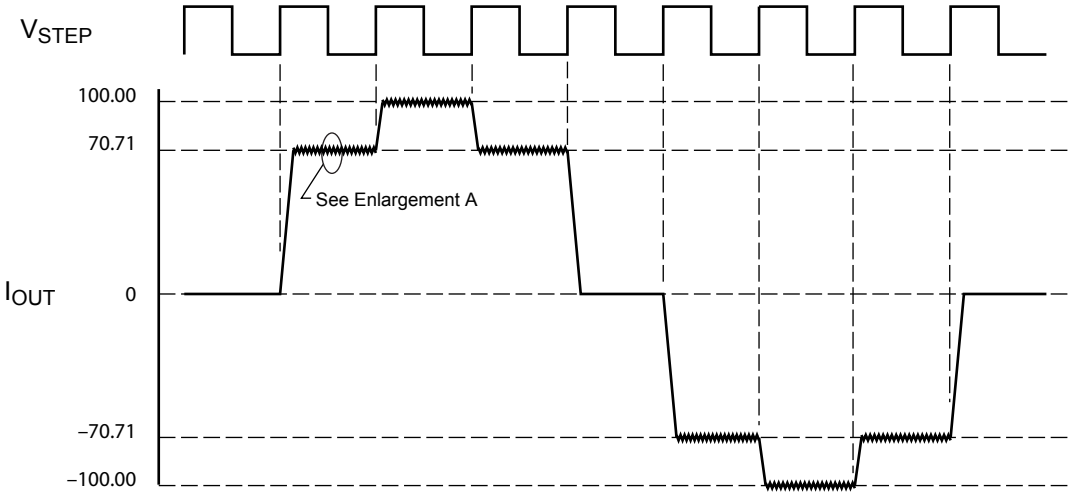
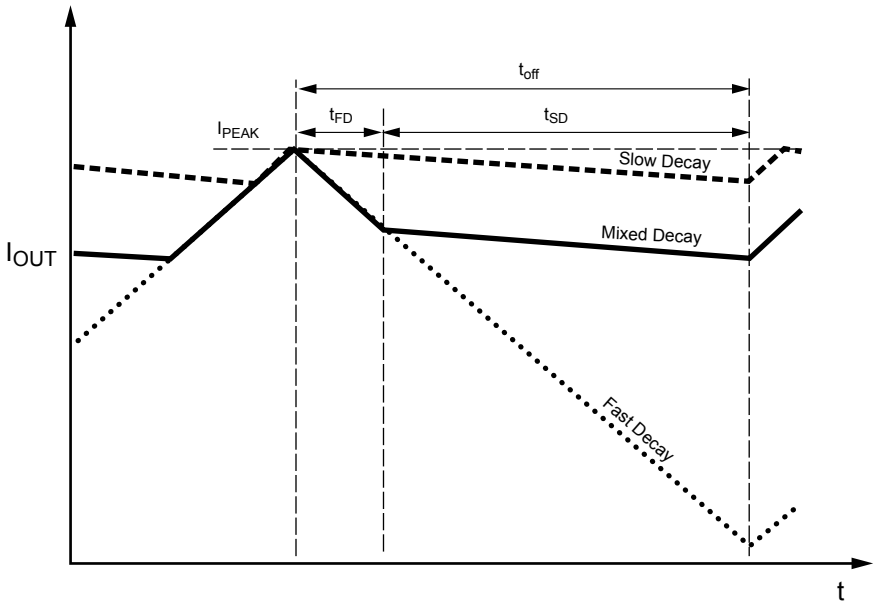


Figure 6: Shorted Load (OUTxA → OUTxB) in Mixed Decay Mode



Enlargement A



Symbol	Characteristic
t_{off}	Device fixed off-time
I_{PEAK}	Maximum output current
t_{SD}	Slow decay interval
t_{FD}	Fast decay interval
I_{OUT}	Device output current

Figure 7: Current Decay Modes Timing Chart

Application Layout

Layout. The printed circuit board should use a heavy ground-plane. For optimum electrical and thermal performance, the A4988 must be soldered directly onto the board. Pins 3 and 18 are internally fused, which provides a path for enhanced thermal dissipation. Theses pins should be soldered directly to an exposed surface on the PCB that connects to thermal vias are used to transfer heat to other layers of the PCB.

In order to minimize the effects of ground bounce and offset issues, it is important to have a low impedance single-point ground, known as a *star ground*, located very close to the device. By making the connection between the pad and the ground plane directly under the A4988, that area becomes an ideal location for a star ground point. A low impedance ground will prevent ground bounce during high current operation and ensure that the supply voltage remains stable at the input terminal.

The two input capacitors should be placed in parallel, and as close to the device supply pins as possible. The ceramic capacitor (CIN1) should be closer to the pins than the bulk capacitor (CIN2). This is necessary because the ceramic capacitor will be responsible for delivering the high frequency current components. The sense resistors, RSx , should have a very low impedance path to ground, because they must carry a large current while supporting very accurate voltage measurements by the current sense comparators. Long ground traces will cause additional voltage drops, adversely affecting the ability of the comparators to accurately measure the current in the windings. The SENSEx pins have very short traces to the RSx resistors and very thick, low impedance traces directly to the star ground underneath the device. If possible, there should be no other components on the sense circuits.

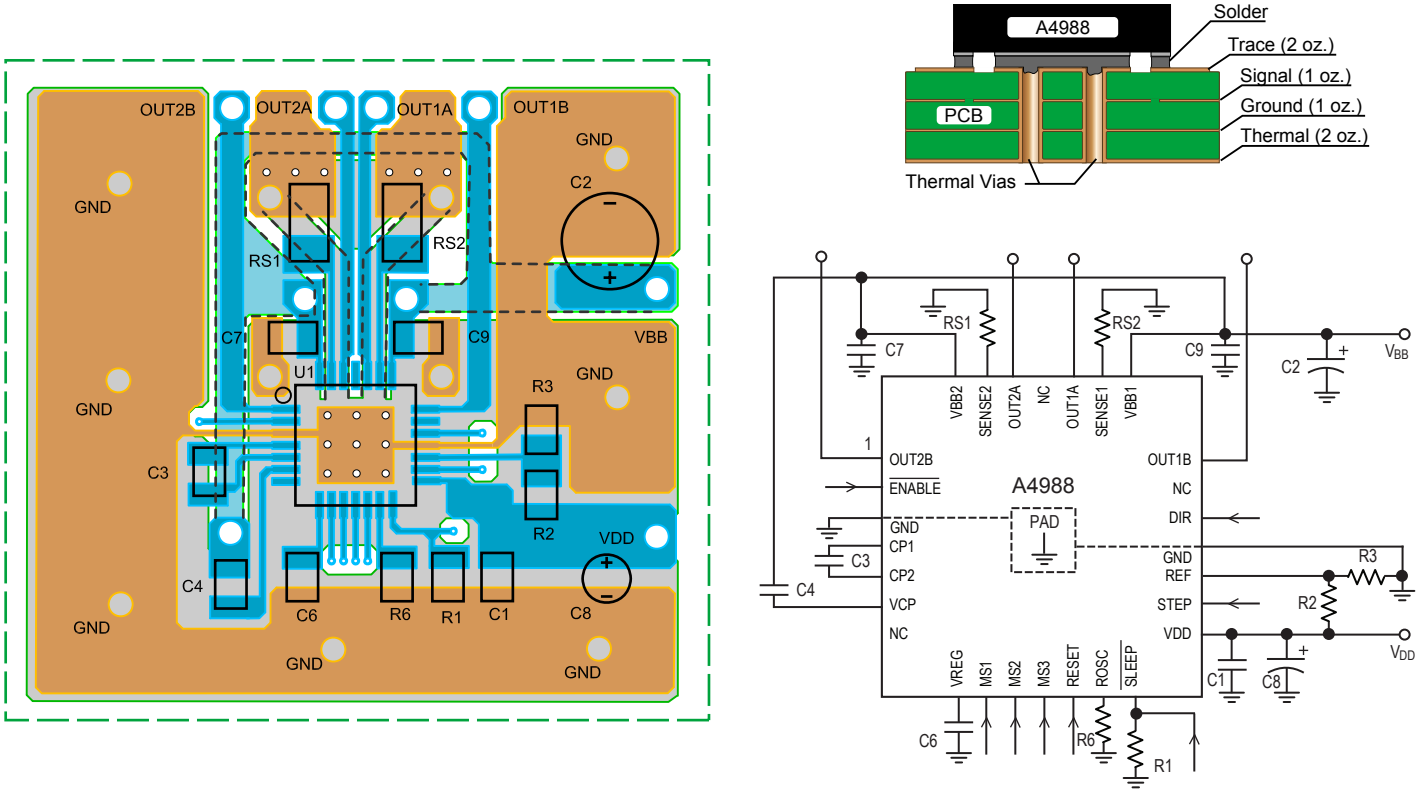
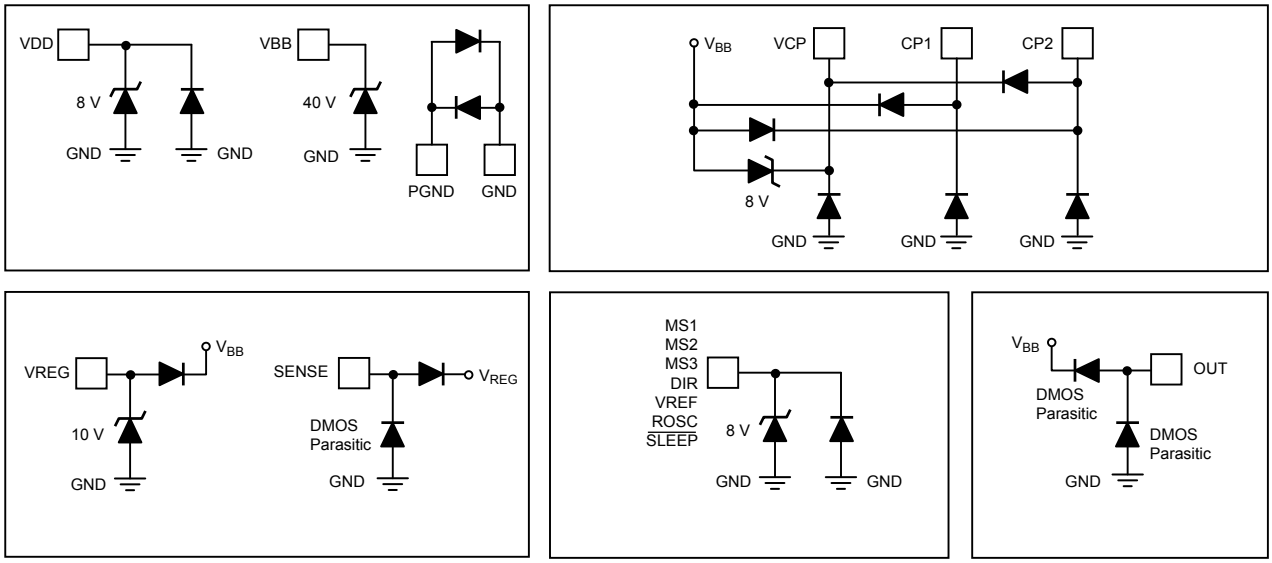
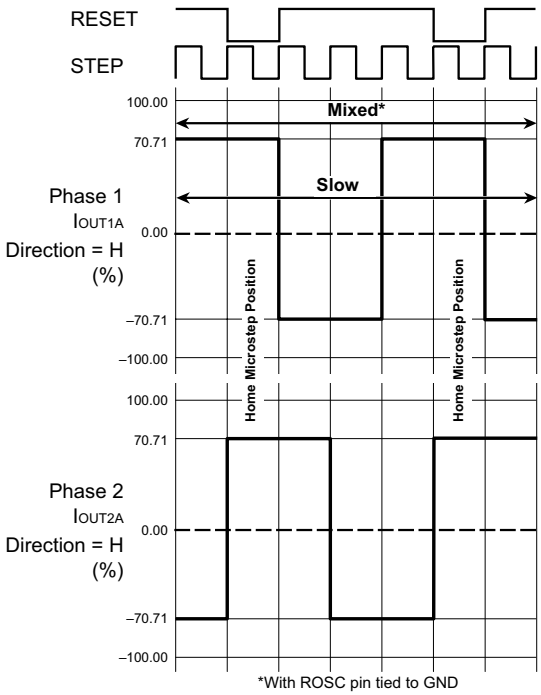


Figure 8: Typical Application and Circuit Layout

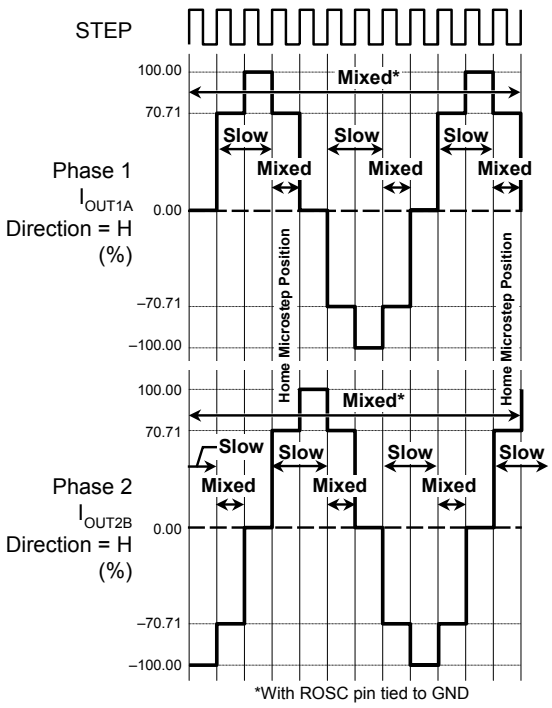
Pin Circuit Diagrams





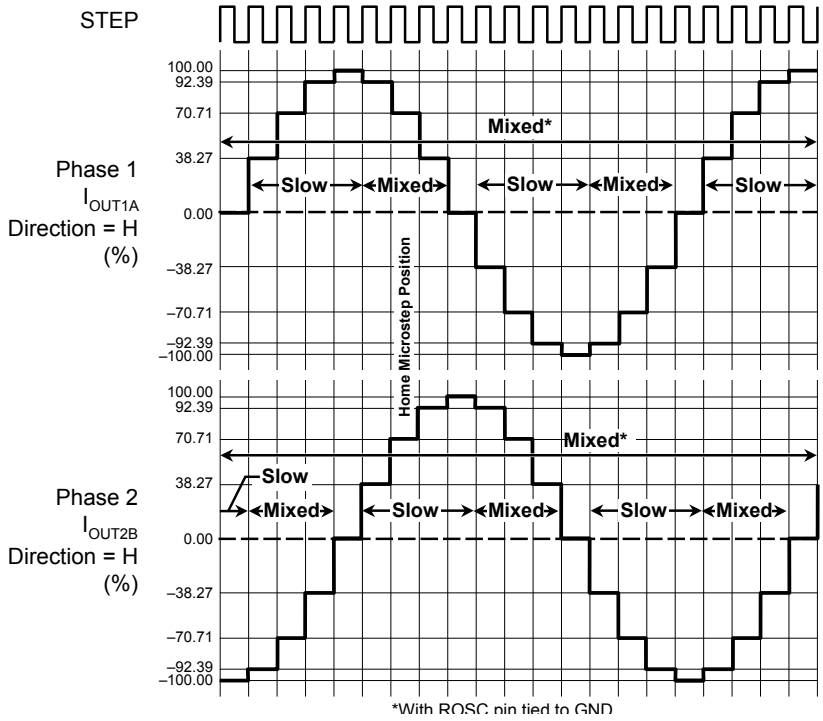
DIR= H

Figure 9: Decay Mode for Full-Step Increments



DIR= H

Figure 10: Decay Modes for Half-Step Increments



DIR= H

Figure 11: Decay Modes for Quarter-Step Increments

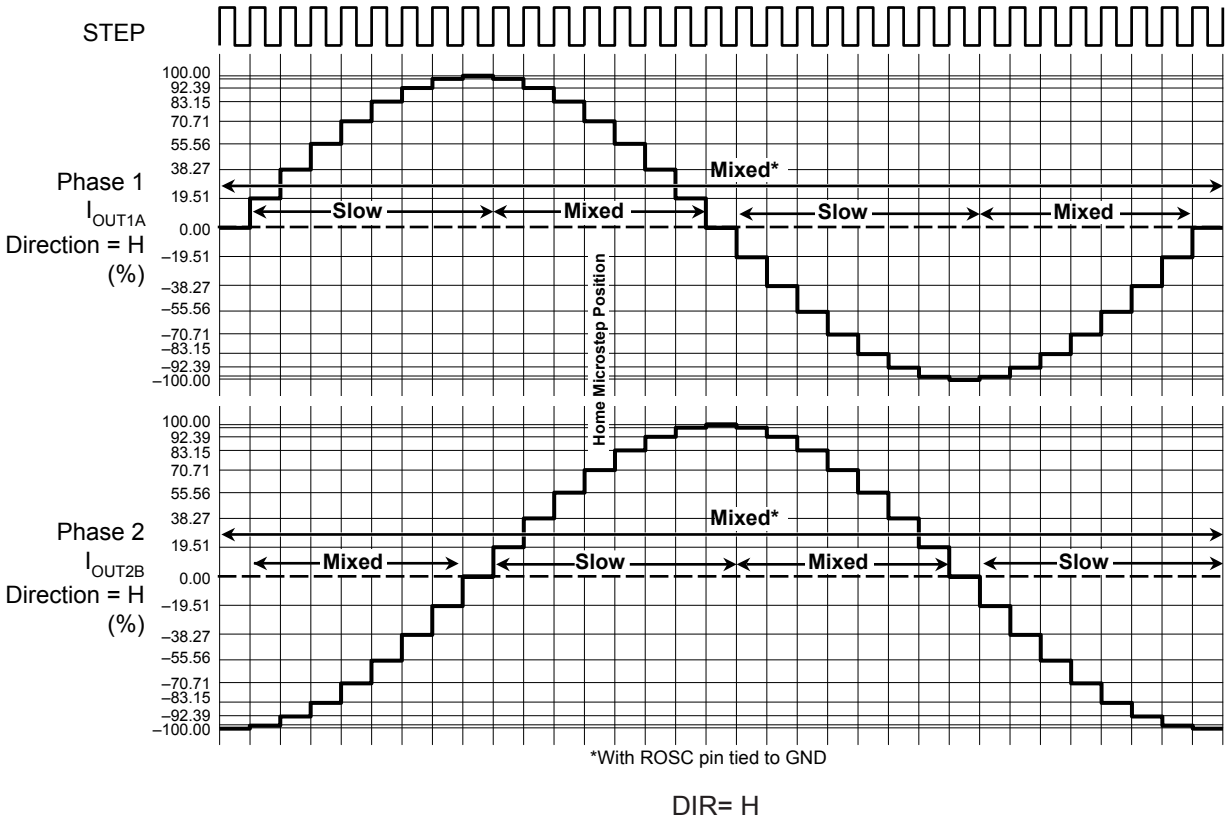


Figure 12: Decay Modes for Eighth-Step Increments

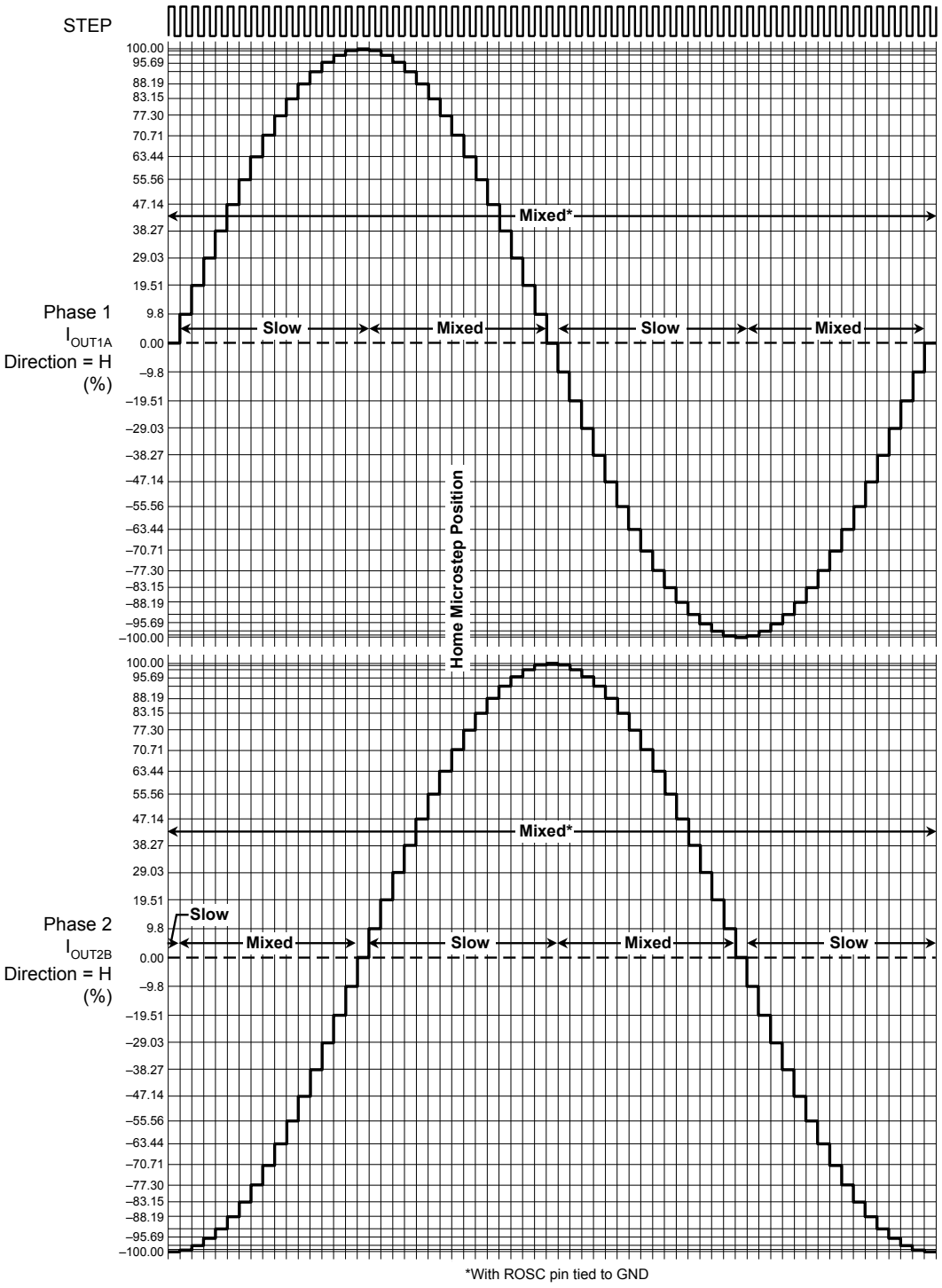


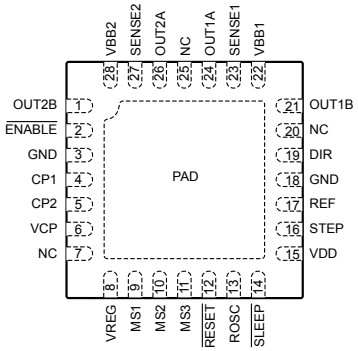
Figure 13: Decay Modes for Sixteenth-Step Increments

Table 2: Step Sequencing Settings

Home microstep position at Step Angle 45º; DIR = H

Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% $I_{tripMax}$] (%)	Phase 2 Current [% $I_{tripMax}$] (%)	Step Angle (º)	Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% $I_{tripMax}$] (%)	Phase 2 Current [% $I_{tripMax}$] (%)	Step Angle (º)
	1	1	1	1	100.00	0.00	0.0		5	9	17	33	–100.00	0.00	180.0
				2	99.52	9.80	5.6					34	–99.52	–9.80	185.6
			2	3	98.08	19.51	11.3				18	35	–98.08	–19.51	191.3
				4	95.69	29.03	16.9					36	–95.69	–29.03	196.9
		2	3	5	92.39	38.27	22.5			10	19	37	–92.39	–38.27	202.5
				6	88.19	47.14	28.1					38	–88.19	–47.14	208.1
			4	7	83.15	55.56	33.8				20	39	–83.15	–55.56	213.8
				8	77.30	63.44	39.4					40	–77.30	–63.44	219.4
1	2	3	5	9	70.71	70.71	45.0	3	6	11	21	41	–70.71	–70.71	225.0
				10	63.44	77.30	50.6					42	–63.44	–77.30	230.6
			6	11	55.56	83.15	56.3				22	43	–55.56	–83.15	236.3
				12	47.14	88.19	61.9					44	–47.14	–88.19	241.9
		4	7	13	38.27	92.39	67.5			12	23	45	–38.27	–92.39	247.5
				14	29.03	95.69	73.1					46	–29.03	–95.69	253.1
			8	15	19.51	98.08	78.8				24	47	–19.51	–98.08	258.8
				16	9.80	99.52	84.4					48	–9.80	–99.52	264.4
	3	5	9	17	0.00	100.00	90.0		7	13	25	49	0.00	–100.00	270.0
				18	–9.80	99.52	95.6					50	9.80	–99.52	275.6
			10	19	–19.51	98.08	101.3				26	51	19.51	–98.08	281.3
				20	–29.03	95.69	106.9					52	29.03	–95.69	286.9
		6	11	21	–38.27	92.39	112.5			14	27	53	38.27	–92.39	292.5
				22	–47.14	88.19	118.1					54	47.14	–88.19	298.1
			12	23	–55.56	83.15	123.8				28	55	55.56	–83.15	303.8
				24	–63.44	77.30	129.4					56	63.44	–77.30	309.4
2	4	7	13	25	–70.71	70.71	135.0	4	8	15	29	57	70.71	–70.71	315.0
				26	–77.30	63.44	140.6					58	77.30	–63.44	320.6
			14	27	–83.15	55.56	146.3				30	59	83.15	–55.56	326.3
				28	–88.19	47.14	151.9					60	88.19	–47.14	331.9
		8	15	29	–92.39	38.27	157.5			16	31	61	92.39	–38.27	337.5
				30	–95.69	29.03	163.1					62	95.69	–29.03	343.1
			16	31	–98.08	19.51	168.8				32	63	98.08	–19.51	348.8
				32	–99.52	9.80	174.4					64	99.52	–9.80	354.4

Pin-out Diagram

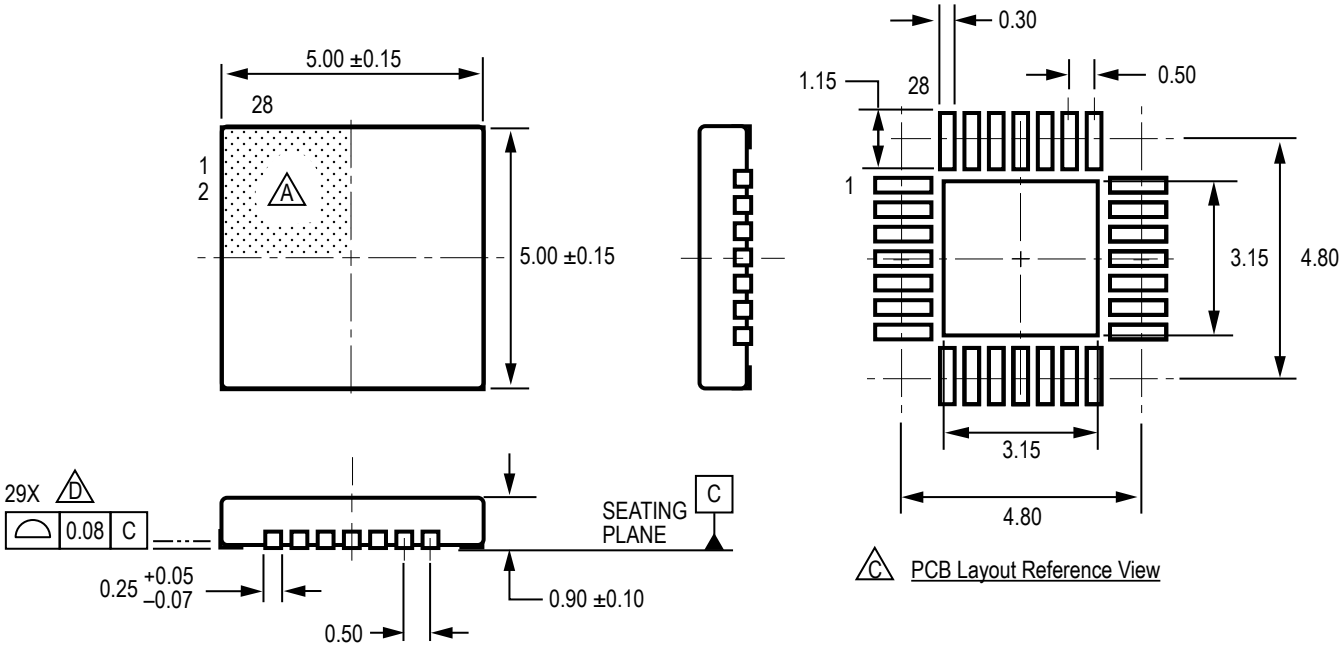


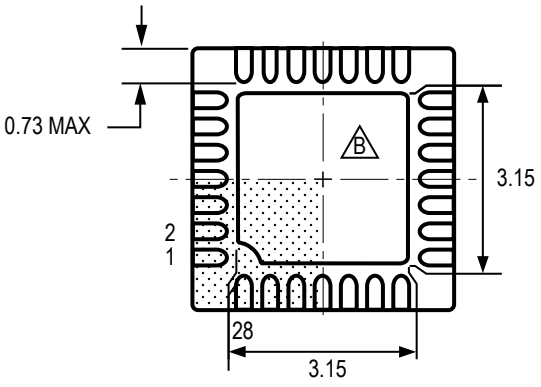
Terminal List Table

Name	Number	Description
CP1	4	Charge pump capacitor terminal
CP2	5	Charge pump capacitor terminal
VCP	6	Reservoir capacitor terminal
VREG	8	Regulator decoupling terminal
MS1	9	Logic input
MS2	10	Logic input
MS3	11	Logic input
$\overline{\text{RESET}}$	12	Logic input
ROSC	13	Timing set
$\overline{\text{SLEEP}}$	14	Logic input
VDD	15	Logic supply
STEP	16	Logic input
REF	17	G_m reference voltage input
GND	3, 18	Ground*
DIR	19	Logic input
OUT1B	21	DMOS Full Bridge 1 Output B
VBB1	22	Load supply
SENSE1	23	Sense resistor terminal for Bridge 1
OUT1A	24	DMOS Full Bridge 1 Output A
OUT2A	26	DMOS Full Bridge 2 Output A
SENSE2	27	Sense resistor terminal for Bridge 2
VBB2	28	Load supply
OUT2B	1	DMOS Full Bridge 2 Output B
$\overline{\text{ENABLE}}$	2	Logic input
NC	7, 20, 25	No connection
PAD	–	Exposed pad for enhanced thermal dissipation*

*The GND pins must be tied together externally by connecting to the PAD ground plane under the device.

ET Package, 28-Pin QFN with Exposed Thermal Pad





For Reference Only; not for tooling use
(reference JEDEC MO-220VHHD-1)
Dimensions in millimeters
Exact case and lead configuration at supplier discretion within limits shown

A Terminal #1 mark area

B Exposed thermal pad (reference only, terminal #1 identifier appearance at supplier discretion)

C Reference land pattern layout (reference IPC7351 QFN50P500X500X100-29V1M);
All pads a minimum of 0.20 mm from all adjacent pads; adjust as necessary to meet application process requirements and PCB layout tolerances; when mounting on a multilayer PCB, thermal vias at the exposed thermal pad land can improve thermal dissipation (reference EIA/JEDEC Standard JESD51-5)

D Coplanarity includes exposed thermal pad and terminals

Revision History

Revision	Revision Date	Description of Revision
4	January 27, 2012	Update I_{OCPST}
5	May 7, 2014	Revised text on pg. 9; revised Figure 8 and Table 2

For the latest version of this document, visit our website:
www.allegromicro.com