Department of Electronics and Telecommunication Engineering
University of Moratuwa

EN2160 Electronic Design Realization

Closed Loop Stepper Motor Driver

Design Document

Group No: 32

210152E	Dulnath W.H.R.
210599E	Shamal G.B.A.
210600D	Shamika K.A.M.
210642G	Thennakoon T.M.K.R.

Content

Introduction

Brief Overview of the Project

The stepper motor driver project involves the design and development of a sophisticated motor control system with a built-in feedback mechanism. This closed-loop stepper motor driver aims to provide precise control over the motor's rotation, ensuring it accurately reaches and maintains the desired position. Unlike traditional open-loop systems, our driver continuously monitors the motor's position and makes real-time adjustments to correct any deviations caused by external loads or other factors. Basically, we are approaching to implement the stepper motor driver which will gain control inputs and encoder inputs and precise

Objectives and Goals

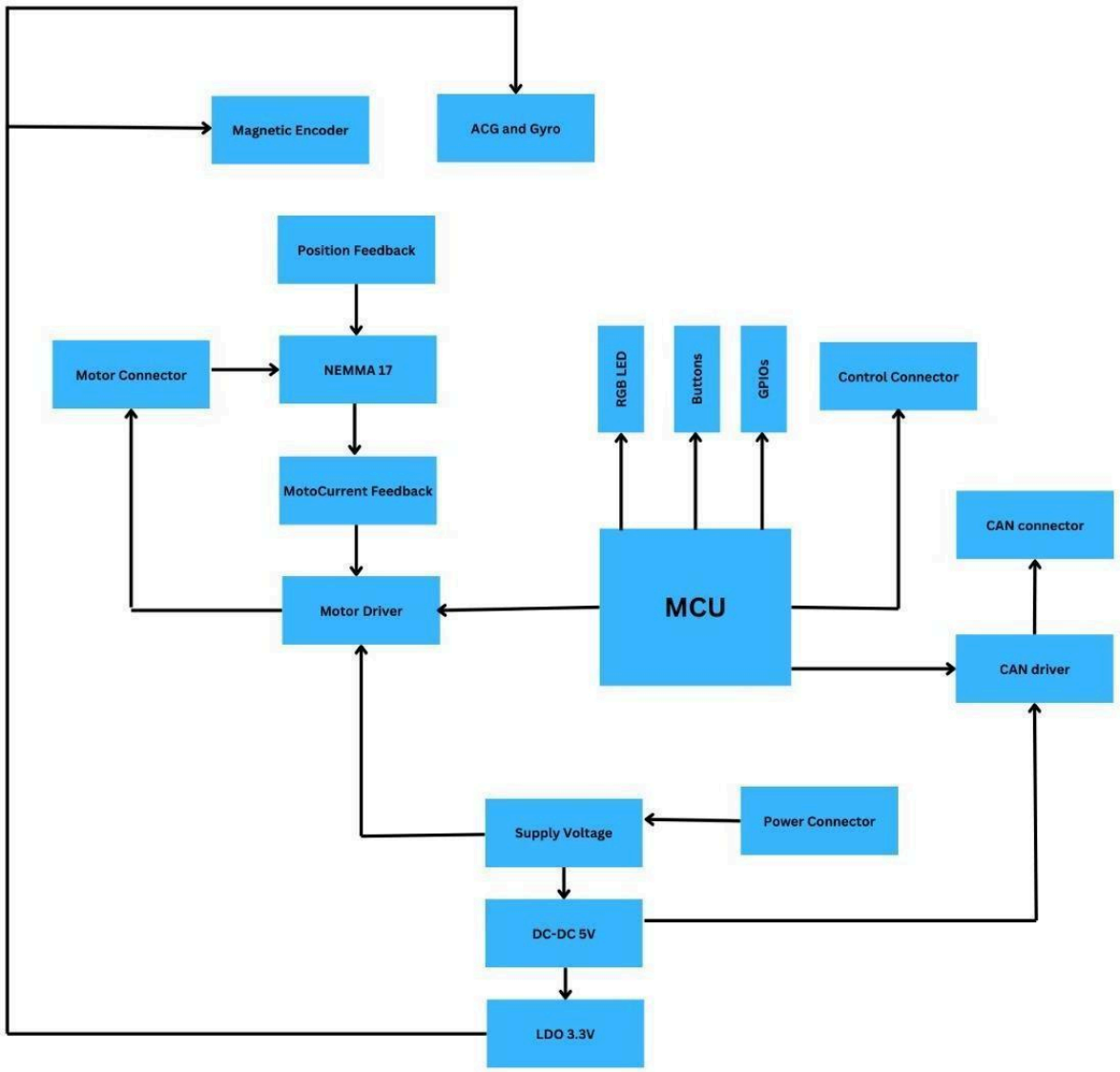
1. **Precise Motor Control:**
   - Develop a driver that ensures the stepper motor reaches the exact target position with high precision.
   - Implement a feedback mechanism to continuously monitor the motor's position and make necessary adjustments.
2. **Error Detection and Correction:**
   - Integrate sensors to detect positional errors caused by external loads or other disturbances.
   - Design a control algorithm that uses feedback data to correct these errors and return the motor to the reference position.
3. **Robust and Reliable Operation:**
   - Ensure the stepper motor driver operates reliably under various load conditions.
   - Minimize the impact of external disturbances on the motor's performance through efficient error correction.
4. **User-Friendly Interface:**
   - Provide a simple and intuitive interface for users to set target positions and monitor the motor's status.
   - Include diagnostic features to help users identify and troubleshoot issues.
5. **Scalability and Flexibility:**
   - Design the driver to be adaptable to different types and sizes of stepper motors.
   - Ensure the system can be easily scaled or modified for various applications.

By achieving these objectives, our closed-loop stepper motor driver aims to deliver precise, reliable, and adaptable motor control for a wide range of industrial and commercial applications.

System Architecture

Block Diagram

The provided block diagram illustrates the overall architecture of your closed-loop stepper motor driver system. Here's a detailed explanation of each component and its role in the system:



Block diagram explanation

1. **Microcontroller Unit:**
   - This central processing unit of the system that controls and coordinates all other components.
   - Here we are using STM32F103CBT6 microcontroller of STMicroelectronics as it suits with our meeting requirements to implement the feedback system and motor controller part.
   - It processes feedback data and generates control signals for the motor driver.
2. **Motor Connector:**
   - Connects the stepper motor (Nema 17) to the system.
   - Allows the motor to receive driving signals from the motor driver.
3. **Motor Driver:**
   - Here we are using two TB67H450FNG PWM Chopper Type DC Brushed Motor Drivers.
   - Receives control signals from the MCU and drives the stepper motor.
   - Provides the necessary current and voltage to operate the motor.
4. **Magnetic Encoder:**
   - Here we are using TZT AS5600 magnetic encoder which should get connected with the stepper motor.
   - Provides precise position feedback to the MCU.
   - Enhances the accuracy of the motor's position control.
5. **RGB LED:**
   - Indicates the status of the system through different colors.
   - Helps in visual diagnostics and monitoring.
6. **Buttons:**
   - User interface for manual control and input.
   - Allows the user to set target positions or initiate specific actions.
7. **GPIO (General Purpose Input/Output):**
   - Provides additional input and output pins for custom functionalities.
   - Allows for expansion and customization of the system.
8. **Control Connector:**
   - Interface for external control signals.
   - Enables integration with other control systems or devices.
9. **CAN Connector:**
   - Interface for CAN (Controller Area Network) communication.
   - Allows the system to communicate with other devices on a CAN bus.
10. **CAN Driver:**
    - SN65HVD23x 3.3-V CAN Bus Transceiver is used.
    - Transceiver that handles communication over the CAN bus.
    - Converts data between the MCU and the CAN network.

11. **Supply Voltage:**
- Main power input for the system.
- 12V input voltage is expected.
- Provides the necessary voltage for all components.

12. **Power Connector:**
- Interface for connecting the power supply to the system.
- Ensures all components receive stable power.

13. **DCDC 5V:**
- DC-DC converter that steps down the input voltage to 5V.
- Switching Voltage Regulators 12-76V 1Ch Buck Conv w/ Integrated FET is used here.

14. **LDO 3.3V:**
- Low Dropout Regulator that steps down the voltage to 3.3V.
- LDO Voltage Regulators Micropower 250mA Lo- Noise Ultra LDO Reg is used.

System Operation

1. **Initialization:**
- Upon power-up, the MCU initializes all components and sensors.
- LED one starts to blink with system power up.
- The system checks the status of the motor and sets the initial position.

2. **Position Control:**
- The user sets a target position using external control signals.
- Here SPI communication is used to give desired control signals.
- The MCU processes the input and calculates the necessary motor movements.

3. **Feedback Loop:**
- The motor driver receives control signals from the MCU and drives the motor.
- Position feedback from the magnetic encoder and current feedback are continuously sent to the MCU.
- The MCU adjusts the control signals in real-time to correct any deviations from the target position.

4. **Error Correction:**
- If an error is detected (e.g., due to an external load), the MCU recalculates the control signals.
- The feedback loop ensures the motor is driven back to the reference position accurately.

5. **Communication:**
- The CAN driver allows the system to communicate with other devices on a CAN network.
- Enables integration with larger systems or networks for coordinated control.
- So user can gain the ability to control multiple stepper motors simultaneously.

By incorporating these components and their interactions, your closed-loop stepper motor driver system achieves precise and reliable motor control, capable of correcting errors and maintaining the desired position even under varying load conditions.

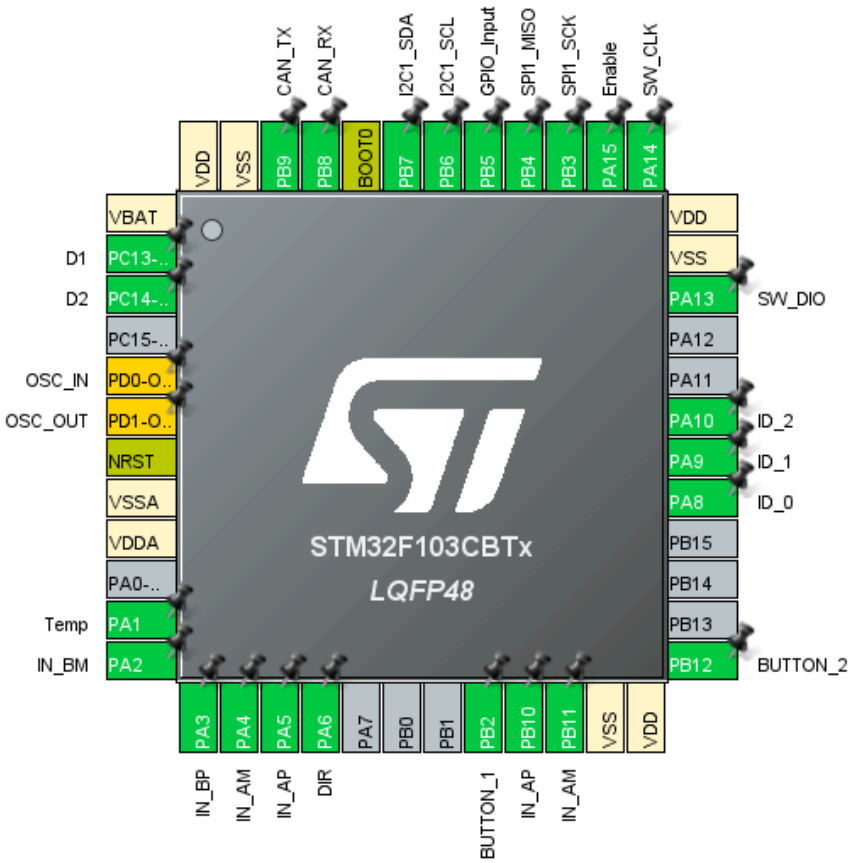
Component Selection

1. **STM Microcontroller:**

- **Justification:** The STM32F103CBT6 microcontroller from STMicroelectronics was selected for its robust processing capabilities, low power consumption, and extensive I/O options, making it ideal for a closed-loop stepper motor driver. Based on the 32-bit Arm Cortex-M3 CPU operating at 72 MHz, this MCU offers up to 128 KB of Flash memory and 20 KB of SRAM. These features provide the necessary computational power to handle real-time feedback processing and control algorithms essential for precise motor control. Its integrated peripherals, including USB, CAN, seven timers, and two ADCs, enhance its utility in motor control applications. Additionally, the MCU supports multiple communication interfaces (I2C, USART, SPI), facilitating extensive connectivity and efficient data exchange. The low-power modes and DMA controller further contribute to its efficiency and performance, ensuring reliable and precise stepper motor operation.

- **Datasheet:** https://www.mouser.com/datasheet/2/389/stm32f103c8-1851025.pdf

- **Pin configuration:**



2. **12MHz Crystal Oscillator:**

- **Justification:** The 12MHz crystal oscillators ensure a stable and accurate clock signal for the microcontroller, which is crucial for maintaining timing accuracy in control tasks. Precise timing is essential for generating accurate PWM signals and processing feedback data.

3. **3.3V CAN Interfaces:**

- **Justification:** The 3.3V CAN interfaces facilitate communication with other devices and systems on a Controller Area Network (CAN) bus. This allows for reliable data exchange and integration into larger control systems, enhancing the system's versatility and scalability.
- **Datasheet:** https://www.ti.com/lit/ds/symlink/sn65hvd232.pdf?ts=1718954842524&ref_url=https%2 53A%252F%252Fwww.mouser.com%252F

4. **PWM Chopper Type DC Brushed Motor Driver:**

- **Justification:**The TB67H450FNG by Toshiba, a PWM chopper type DC brushed motor driver, was chosen for its efficiency in controlling motor speed and torque. Featuring BiCD process integration, it supports forward, reverse, brake, and stop modes, delivering up to 50V and 3.5A through its low-resistance MOSFETs. This IC provides precise control of stepper motors by varying the PWM signal's duty cycle, making it ideal for high-precision applications. It includes essential protections like thermal shutdown, overcurrent detection, and undervoltage lockout, ensuring robust and reliable motor management. Packaged in HSOP8, it supports both constant current and direct PWM drive capabilities.

- **Datasheet:** https://www.mouser.com/datasheet/2/408/TB67H450FNG_datasheet_en_20201126-1604947.pdf

5. **Resistors and Capacitors:**

- **Justification:** Resistors and capacitors are fundamental components used for various purposes, including filtering, signal conditioning, and voltage regulation. They ensure stable operation of the circuit by smoothing out voltage fluctuations and noise.

By carefully selecting these components, the design ensures a reliable, efficient, and precise stepper motor driver system capable of performing under various conditions while maintaining high accuracy and stability.

6. Switching Voltage Regulators (12-76V 1Ch Buck Converter with Integrated FET, RoHS Compliant):

- **Justification:** The switching voltage regulators are used to step down the input voltage to 5V. The integrated FET design offers high efficiency and low power loss, which is essential for maintaining the system's overall efficiency and reliability.
- **Datasheet :** https://www.mouser.com/datasheet/2/348/bd9g341aefj_lb_e-1874278.pdf

7. LDO Voltage Regulators (Micropower 250mA Low-Noise Ultra LDO Regulator, RoHS Compliant):

- **Justification:** The LDO (Low Dropout) voltage regulators provide a stable 3.3V supply with low noise characteristics, which is critical for powering up the MCU and for sensitive analog and digital circuits. Their micropower operation helps in minimizing power consumption, making them suitable for low-power applications.
- **Datasheet:** https://www.ti.com/lit/ds/symlink/lp2992.pdf?ts=1718903067976&ref_url=http%253A%252F%252Fpavouk.org%252FCircuit Design

8. **Encoder**

- **Justification:**The TZT AS5600 magnetic encoder is a crucial component in closed-loop stepper motor control systems, providing precise positional feedback essential for accurate motor control. This encoder utilizes Hall effect technology to detect the magnetic field angle, offering high resolution and reliability in various operating conditions. Its digital output interface ensures compatibility with microcontrollers and other digital circuitry, simplifying integration into stepper motor driver systems. By continuously monitoring the rotor position, the AS5600 enables real-time adjustments to motor drive signals, enhancing stability, reducing resonance, and improving overall system performance in applications requiring precise motion control. This encoder use I2C communication.Its compact design and low power consumption make it an ideal choice for applications demanding both accuracy and efficiency.

- **data sheet :** https://files.seeedstudio.com/wiki/Grove-12-bit-Magnetic-Rotary-Position-Sensor-AS5600/res/Magnetic%20Rotary%20Position%20Sensor%20AS5600%20Datasheet.pdf

Circuit Parts

The circuit design for the stepper motor driver consists of three main parts: the power circuit, the motor controller part, and the communication part. Each of these parts plays a crucial role in ensuring the overall functionality and performance of the system.

1. Power Circuit:

- **Components Involved:**
  - Switching Voltage Regulators (12-76V 1Ch Buck Converter with Integrated FET)
  - LDO Voltage Regulators (Micropower 250mA Low-Noise Ultra LDO Regulator)
- **Functionality:**
  - The power circuit is responsible for converting the input voltage to stable 3.3V and 5V outputs required by various components of the system.
  - The switching voltage regulator efficiently steps down the input voltage to 5V. It uses a buck converter design with an integrated FET to provide high efficiency and low power loss.
  - The LDO voltage regulator further steps down the 5V to a stable 3.3V, ensuring low noise and stable operation for sensitive components such as the microcontroller and communication interfaces.
  - By providing stable and clean power supplies, the power circuit ensures the reliable operation of the entire system, preventing voltage fluctuations and noise from affecting performance.

2. Motor Controller Part:

- **Components Involved:**
  - STM Microcontroller
  - Two PWM Chopper Type DC Brushed Motor Drivers
  - Resistors and Capacitors (for filtering and signal conditioning)
  - Magnetic Encoder (for position feedback)
  - Nema 17 Stepper Motor

- **Functionality:**
    - The motor controller part is the core of the system, handling the control and driving of the stepper motor.
    - The STM microcontroller processes input signals, feedback data, and control algorithms to generate precise PWM signals for the motor driver.
    - The PWM chopper type DC brushed motor driver receives PWM signals from the microcontroller and controls the current flowing through the stepper motor coils, enabling precise control of the motor's speed and position.
  - The magnetic encoder provides real-time position feedback to the microcontroller, allowing it to adjust the motor's position accurately.
  - Resistors and capacitors are used for filtering and signal conditioning, ensuring clean and stable signals for the motor driver and feedback sensors.

3. **Communication Part:**

- **Components Involved:**
  - 3.3V CAN Interfaces
  - CAN Driver
- **Functionality:**
  - The communication part enables the stepper motor driver to connect and communicate with other devices and systems on a CAN (Controller Area Network) bus.
  - The 3.3V CAN interfaces facilitate reliable communication by converting the microcontroller's digital signals into CAN-compatible signals.
  - The CAN driver acts as a transceiver, handling the transmission and reception of data on the CAN bus.
  - This communication capability allows multiple motor drivers to be connected and controlled simultaneously, enabling coordinated operation and synchronization of multiple motors in larger systems.
  - By enabling communication with other devices, the CAN interfaces enhance the scalability and flexibility of the system, making it suitable for complex and distributed applications.

Feedback Mechanism

Description of the Feedback Mechanism Used

The feedback mechanism in our stepper motor driver system is designed to ensure precise control and accurate positioning of the motor. This is achieved by using a magnetic encoder to monitor the motor's position and provide real-time feedback to the control system.

Components Involved:

- **Magnetic Encoder:** The primary sensor used for capturing the position of the stepper motor
- **Microcontroller :** Processes the feedback signals from the magnetic encoder.
- **Motor Driver:** Adjusts the motor's operation based on the feedback processed by the MCU.

Functionality:

1. **Position Sensing:**
   - **Magnetic Encoder:** The magnetic encoder is mounted on the stepper motor and continuously monitors its rotational position. It generates signals corresponding to the motor's position, which are then fed into the microcontroller.

2. **Signal Processing:**
   - **Microcontroller:** The MCU receives the position signals from the magnetic encoder. It processes these signals to determine the current position of the motor and compares it with the desired position. The MCU executes control algorithms to calculate the necessary adjustments needed to correct any deviation from the desired position.

3. **Error Detection and Correction:**
   - **Error Identification:** If there is a discrepancy between the actual position (as reported by the encoder) and the target position, the MCU identifies this as an error.
   - **Feedback Control:** The MCU uses a feedback control loop to correct this error. It sends control signals to the motor driver, instructing it to adjust the motor's operation (e.g., speed, direction) to bring the motor back to the desired position.

4. **Motor Control:**
   - **Motor Driver:** The motor driver receives control signals from the MCU and adjusts the motor's current and voltage accordingly. This ensures that the motor follows the correct path and reaches the target position accurately.

Benefits of the Feedback Mechanism

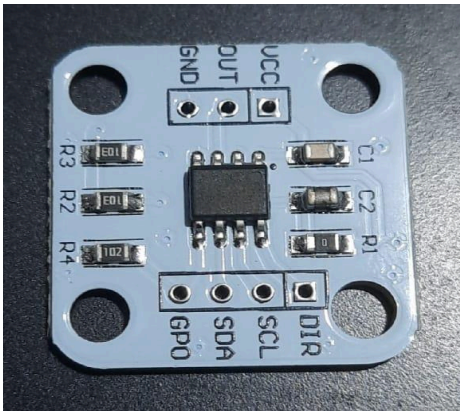
- **Precision:** The feedback mechanism allows for precise control of the stepper motor, ensuring it reaches the exact desired position.

- **Error Correction:** Real-time feedback enables immediate detection and correction of errors, improving the overall accuracy and reliability of the system.

- **Stability:** Continuous monitoring and adjustment of the motor's operation lead to stable performance, even under varying load conditions.

- **Efficiency:** By optimizing the motor's performance based on real-time feedback, the system operates more efficiently, reducing power consumption and wear on components.

In summary, the feedback mechanism involving a magnetic encoder and a microcontroller is crucial for achieving high precision and reliability in our stepper motor driver system. It ensures that the motor operates accurately, correcting any positional errors in real-time, thus maintaining the desired performance and stability.



AS5600 magnetic encoder

Overall Functionality

The closed-loop stepper motor driver system operates as follows:

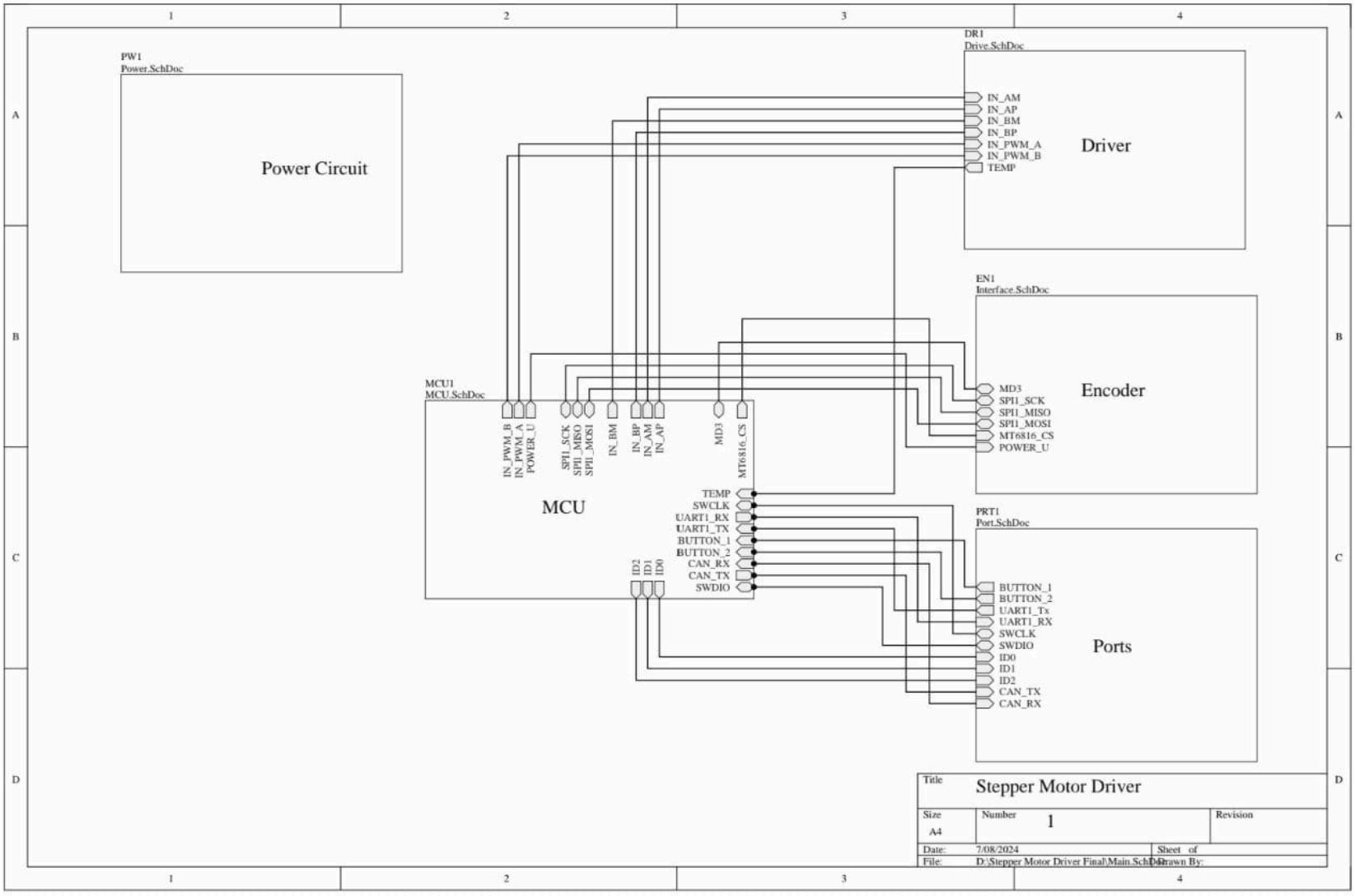
- **Microcontroller Processing:** The STM32F103CBT6 microcontroller reads the encoder data, compares it with the desired position setpoint, and computes the necessary adjustments using a PID (Proportional-Integral-Derivative) control algorithm.
- **Encoder Feedback:** The AS5600 continuously measures the shaft position of the stepper motor.
- **Motor Control:** Based on the computed control signal, the microcontroller sends PWM signals to the TB67H450FNG motor drivers. These drivers adjust the current flowing through the stepper motor windings, thereby controlling its rotation speed and direction.
- **Power Management:** The switching voltage regulators and LDO regulators ensure that all components receive the appropriate voltage levels, optimizing performance and reliability.

By integrating these functions, the stepper motor driver system achieves the following:

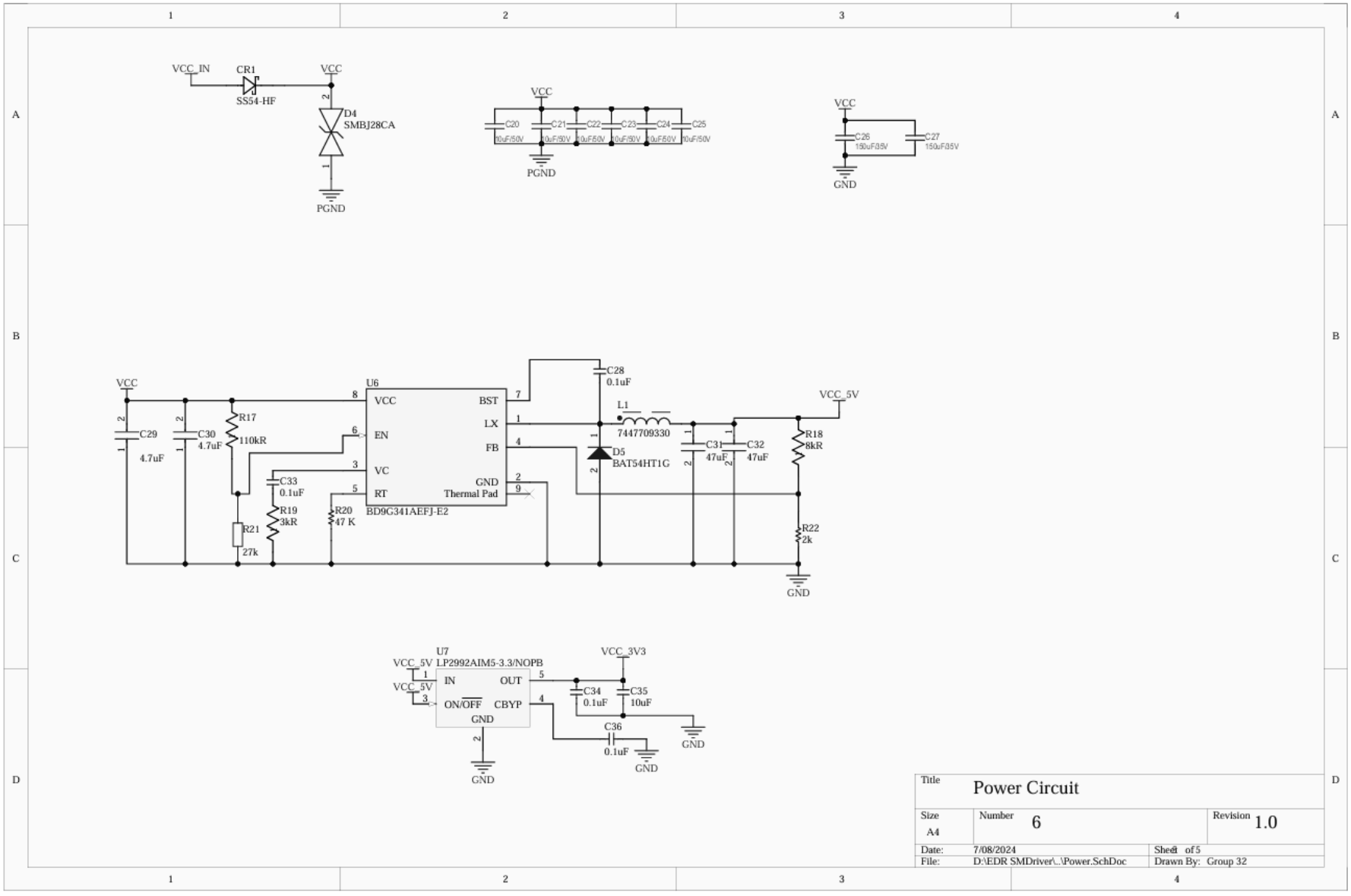
- **Precise and Reliable Motor Control:** The motor controller part ensures accurate positioning and smooth operation of the stepper motor, with real-time feedback and error correction.
- **Stable and Efficient Power Supply:** The power circuit provides clean and stable voltages necessary for the reliable operation of all components.
- **Scalable and Coordinated Operation:** The communication part allows multiple motor drivers to be connected and controlled over a CAN bus, enabling complex and coordinated motor control applications.

This comprehensive design ensures that the stepper motor driver system is robust, efficient, and capable of delivering high precision and reliability in various applications.This system is crucial in applications requiring accurate positioning, torque control, and reliability in various industrial automation, robotics, and CNC machinery environments.

Overview of the Schematic Design



1) Power Circuit

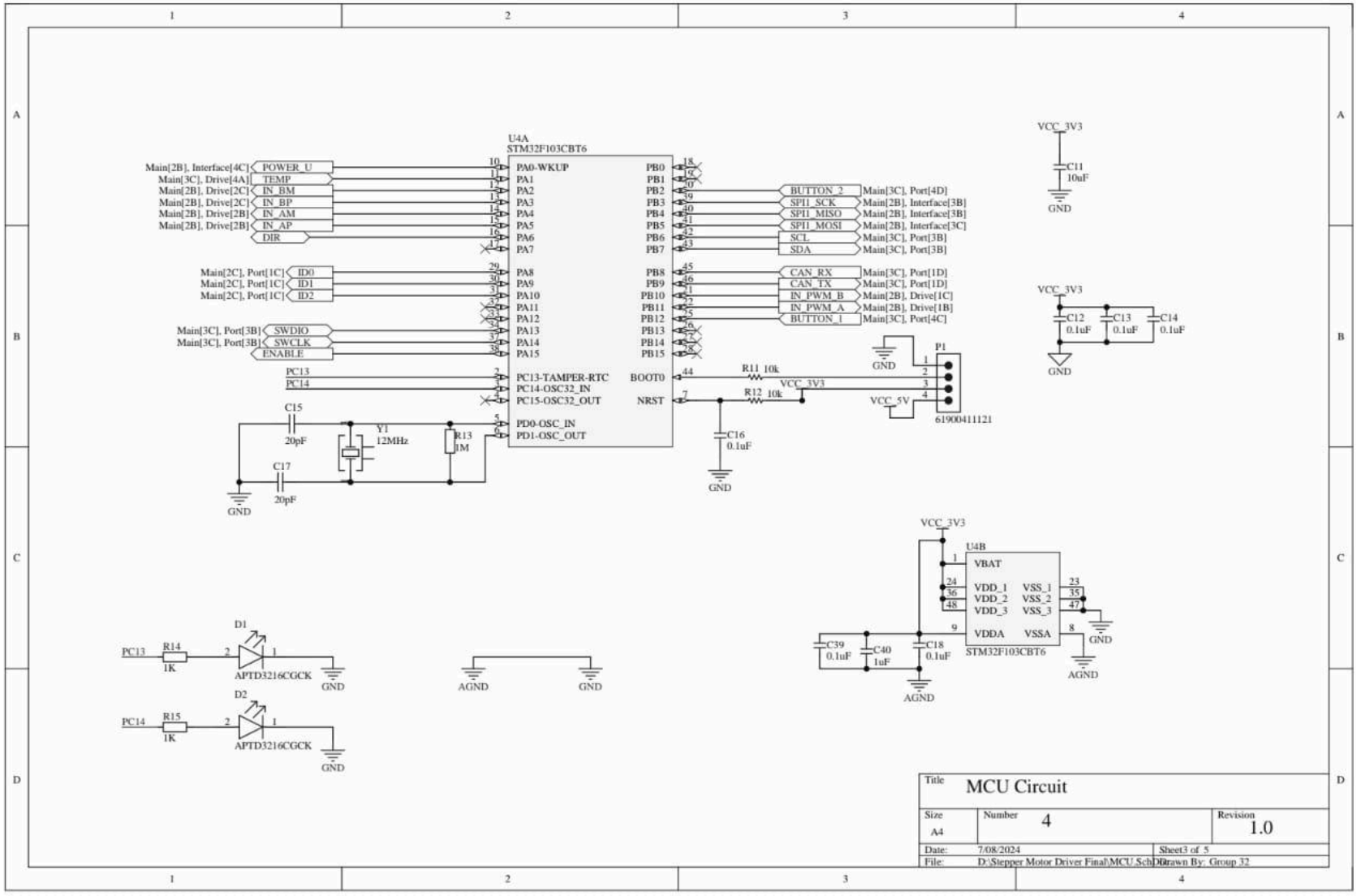


The power circuit of this stepper motor driver is designed with three primary sections: input filtering and protection, the main DC-DC converter, and the secondary voltage regulator. Starting with the input filtering and protection, the circuit incorporates a Schottky diode (SS54-HF) for reverse polarity protection, ensuring that incorrect power supply connections do not damage the circuit. Additionally, a TVS (Transient Voltage Suppression) diode (SMBJ26CA) is employed to protect against voltage spikes and transients. A series of decoupling capacitors (C20, C21, C22, C23, C24, C25, C26, C27) are used to filter out noise and stabilize the VCC input voltage.

The heart of the power circuit is the main DC-DC converter, centered around the BD9G341AEFJ-E2 buck converter IC. This IC steps down the input voltage (VCC) to a regulated 5V output (VCC_5V). The inductor (L1) in the circuit stores energy and helps smooth the current, while a Schottky diode (BAT54HT1G) allows current flow during the off cycles of the switching regulator. Capacitors C28, C31, and C32 are critical in filtering and stabilizing the 5V output. Resistors R18 and R22 form a voltage divider network for feedback to the buck converter IC, ensuring precise regulation of the output voltage. Additionally, components like C29, C30, C33, R17, R19, and R21 are part of the compensation network, maintaining stable operation of the converter.

Lastly, the secondary voltage regulator is implemented using the LP2992AIM5-3.3/NOPB low dropout (LDO) regulator. This regulator steps down the 5V output from the buck converter to 3.3V (VCC_3V3), catering to components requiring a lower voltage. Capacitors C34, C35, and C36 are employed to filter and stabilize the 3.3V output. This structured combination of input protection, DC-DC conversion, and voltage regulation ensures a stable and protected power supply for the stepper motor driver circuit, enabling reliable and efficient operation.

2) Micro Controller Circuit



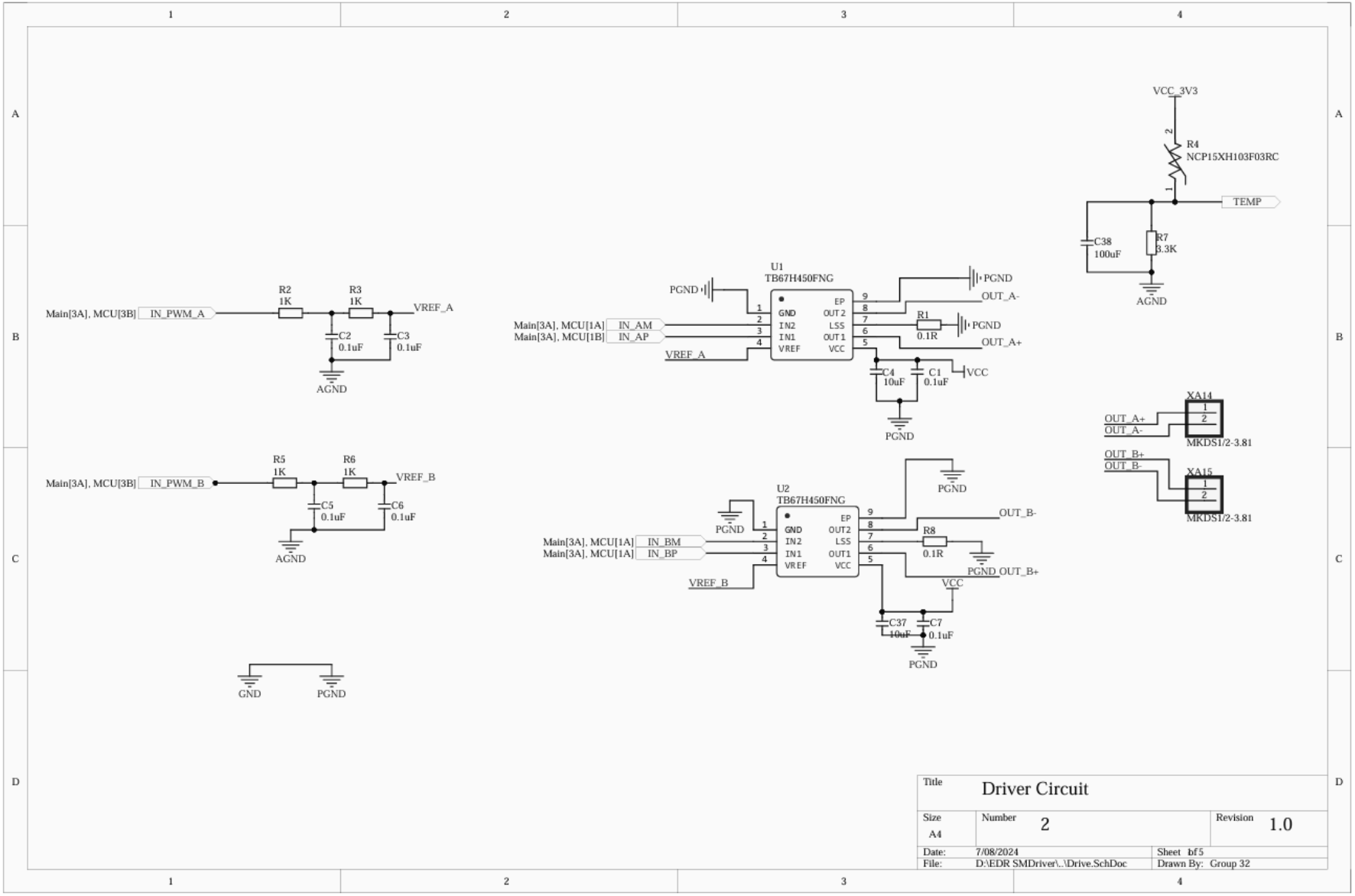
The microcontroller unit (MCU) circuit is centered around the STM32F103CBT6 microcontroller (U4A) and includes various components to support its operation, interfacing, and power management. The STM32F103CBT6, a 32-bit ARM Cortex-M3 microcontroller from STMicroelectronics, serves as the main processing unit for the stepper motor driver, handling all control and feedback tasks. A 12 MHz crystal oscillator (Y1), along with 20 pF capacitors (C15, C17) and a resistor (R13), forms the necessary oscillator circuit for accurate timing and operation, providing the clock signal for the MCU.

To ensure reliable operation, several decoupling capacitors (C11, C12, C13, C14, C16, C18, C19, C39, C40) are used to filter and stabilize the 3.3V power supply to the MCU. Pull-up resistors (R11, R12) for the reset (NRST) and boot (BOOT0) pins ensure the MCU starts up correctly, and a header (P1) provides access to the MCU's programming and debugging interface.

The MCU connects to various peripherals and interfaces through its GPIO pins, configured for input and output functions. Pins PA0-WKUP to PA15, PB0 to PB11, and PC13 to PC15 are used for these functions. The SWDIO and SWCLK pins are designated for programming and debugging the MCU, while CAN RX and CAN TX pins are used for CAN bus communication. SPI1 SCK, SPI1 MOSI, and SPI1 MISO pins are used for SPI communication.

The circuit also includes LEDs (D1, D2) for status indication, with current-limiting resistors (R14, R15) to protect the LEDs. Additional decoupling capacitors (C39, C40) are included for the analog supply (VDDA) to the MCU, ensuring stable operation. Overall, the microcontroller circuit is designed to manage the control logic for the stepper motor driver, providing reliable and efficient control through various interfaces and communication protocols. The setup ensures the MCU operates correctly and interfaces seamlessly with other components in the stepper motor driver system.

3) Driver Circuit

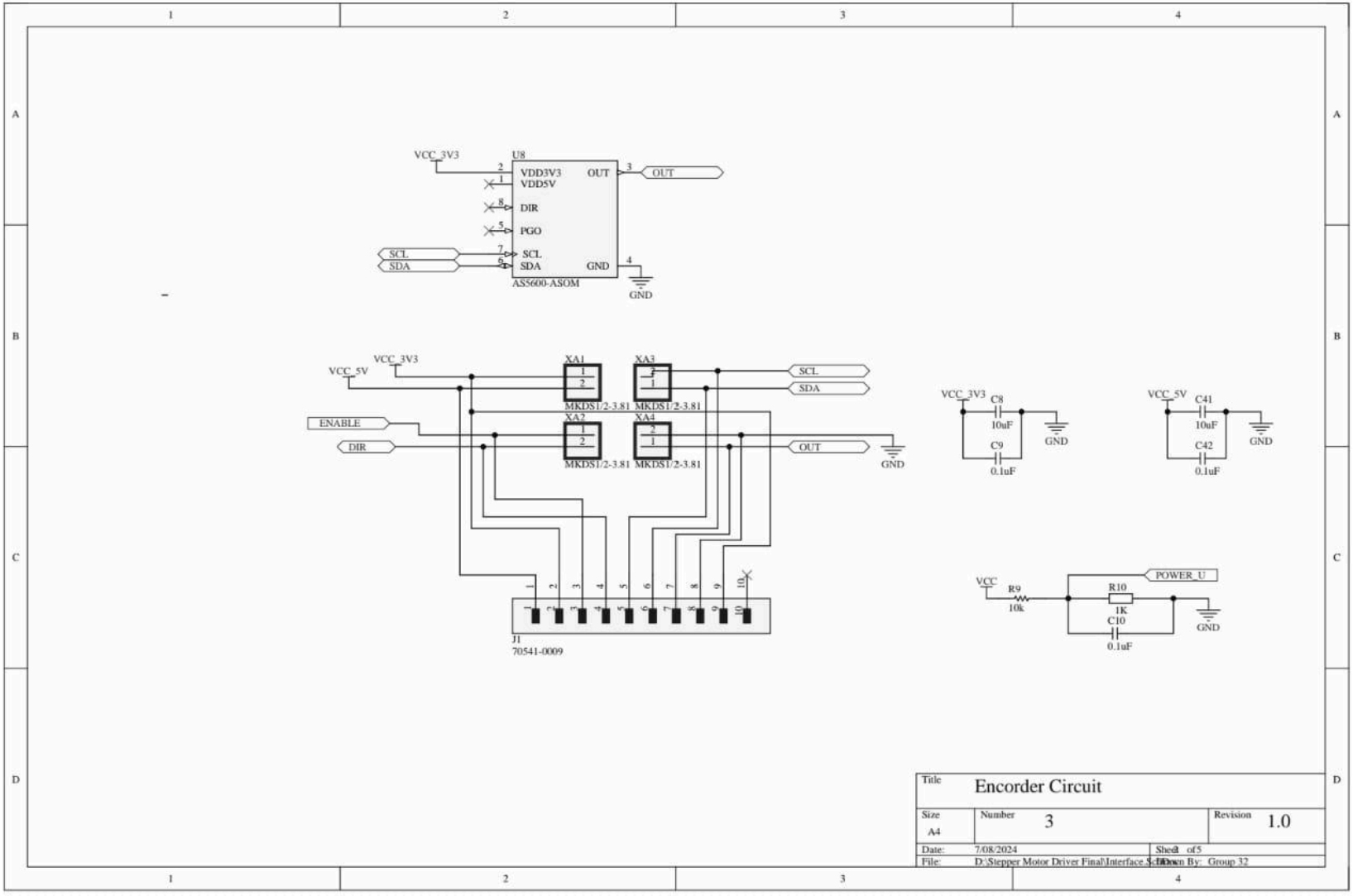


The driver circuit is responsible for controlling the stepper motor based on signals received from the microcontroller. This schematic includes key components such as PWM signal inputs, reference voltage generation, and H-bridge motor driver ICs for motor control.

1. **PWM Signal Inputs and Filtering:**
    - **PWM Inputs:** The circuit receives two PWM signals from the microcontroller (MCU), labeled as IN_PWM_A and IN_PWM_B.
    - **Filtering and Protection:** Each PWM signal is filtered using a resistor-capacitor (RC) network. For IN_PWM_A, R2 (1kΩ) and C2 (0.1µF) are used, and for IN_PWM_B, R5 (1kΩ) and C5 (0.1µF) are used. These RC networks filter noise from the PWM signals before they are fed into the motor driver ICs.
2. **Reference Voltage Generation:**
    - **VREF_A and VREF_B:** Two reference voltages, VREF_A and VREF_B, are generated using RC filters. For VREF_A, the RC filter consists of R3 (1kΩ) and C3 (0.1µF) for VREF_A, and R6 (1kΩ) and C6 (0.1µF) for VREF_B. These reference voltages are used by the motor driver ICs to control the current through the motor windings.
3. **Motor Driver ICs:**
    - **H-Bridge Driver ICs:** The core of the driver circuit consists of two H-bridge motor driver ICs, TB67H450FNG (U1 and U2). These ICs receive the filtered PWM signals and reference voltages to control the stepper motor's coils.
    - **Inputs to Driver ICs:** U1 receives inputs IN_AM and IN_AP, which are derived from the filtered PWM_A signal and VREF_A. Similarly, U2 receives inputs IN_BM and IN_BP, derived from the filtered PWM_B signal and VREF_B.
    - **Output Filtering and Current Sensing:** Each driver IC has output filtering capacitors (C1, C4, C37) and current sense resistors (R1, R8, R10) to ensure stable and precise motor control. The outputs OUT_A+ and OUT_A- from U1, and OUT_B+ and OUT_B- from U2, connect to the stepper motor windings.
4. **Temperature Monitoring:**
    - **Temperature Sensor:** A temperature sensor (R4 - NCP15XH103F03RC) is connected to the microcontroller to monitor the temperature of the driver circuit. The sensor's output is connected to the MCU via the TEMP line.
5. **Connectors and Interfaces:**
    - **Output Connectors:** The outputs of the motor driver ICs are connected to output connectors (XA14, XA15), which interface with the stepper motor. Each connector provides connections for the respective motor windings (OUT_A+, OUT_A-, OUT_B+, OUT_B-).

This driver circuit is designed to convert the PWM signals from the microcontroller into precise motor control signals, ensuring efficient and accurate operation of the stepper motor. The inclusion of filtering, reference voltage generation, and temperature monitoring further enhances the reliability and stability of the driver circuit.

4) Encoder Circuit



The schematic represents a magnetic encoder module based on the AS5600 sensor, designed for precise position sensing using I2C communication. Below is a detailed description of the circuit components and their functions:

Magnetic Encoder IC:

AS5600 (U8): The AS5600 is a contactless magnetic rotary position sensor that provides high-resolution angle measurements. It communicates with the microcontroller via the I2C interface.

Power Supply: The AS5600 is powered by a 3.3V supply (VCC_3V3), connected to its VDD3V3 pin (pin 1). It also has a VDD5V pin (pin 2) connected to a 5V supply (VCC_5V).

I2C Communication: The SDA (pin 7) and SCL (pin 6) pins of the AS5600 are used for I2C communication with the microcontroller. These lines are connected to the corresponding I2C lines on the PCB.

Decoupling Capacitors:

Capacitors C8, C9, C41, and C42: These capacitors (10µF and 0.1µF) are used for power supply decoupling to filter out noise and stabilize the supply voltage. C8 and C9 are connected to the 3.3V supply, while C41 and C42 are connected to the 5V supply.

Signal Conditioning:

Resistors and Capacitors for Signal Integrity:

R9 and R10: Resistors (10kΩ and 1kΩ) are used in the circuit to pull up the I2C lines and for signal conditioning.

C10: A capacitor (0.1µF) is used for additional filtering of the power supply.

Connector and Interface:

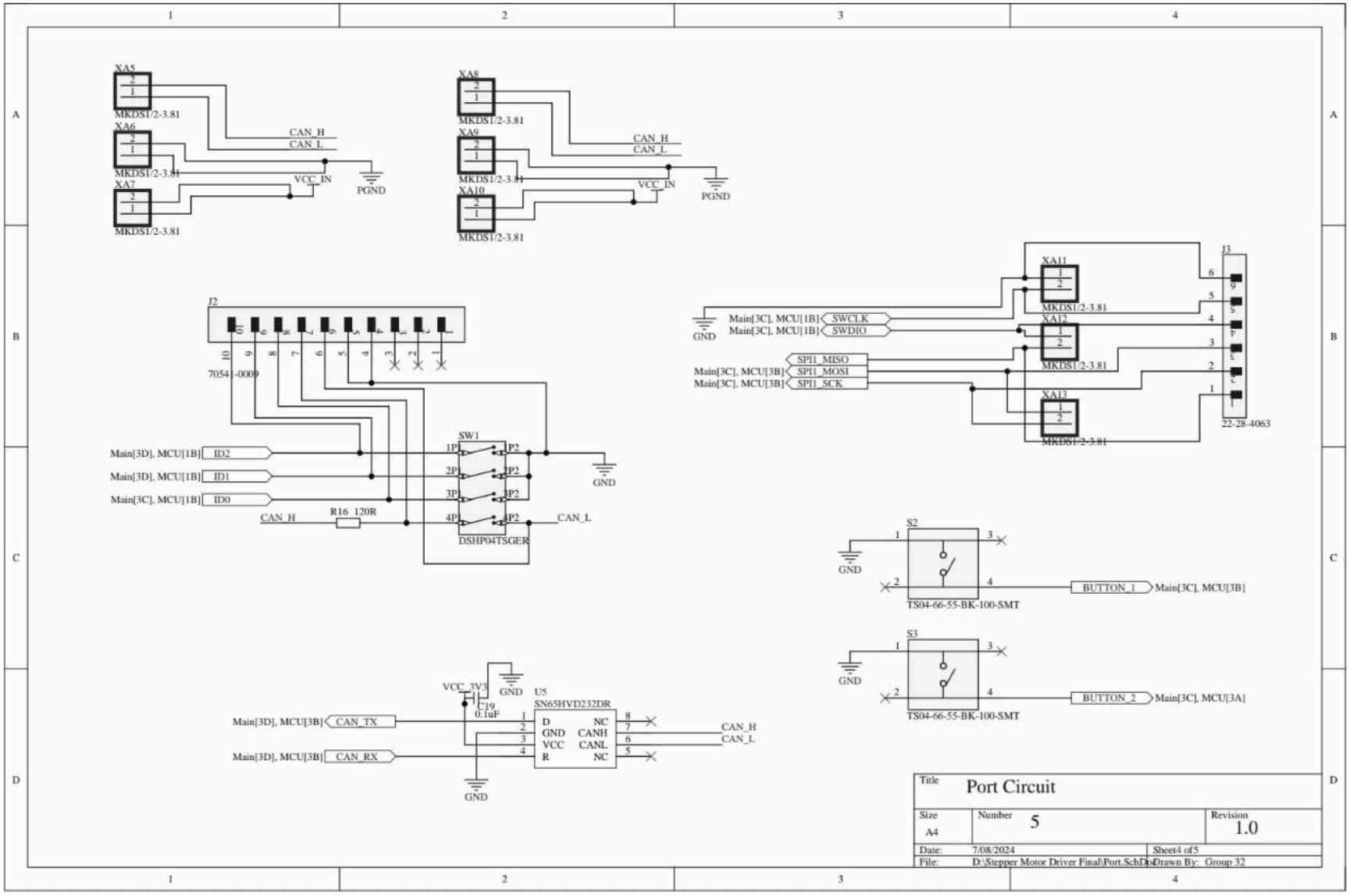
Connector J1 (70541-0009): This connector provides an interface to connect the AS5600 module to the PCB. It includes pins for the power supply, I2C lines, direction control, and output signal.

Circuit Connections:

Power Supply Connections: The 3.3V and 5V power supplies are connected to the AS5600 through the connectors and decoupling capacitors.

I2C Lines: The I2C lines (SCL and SDA) from the AS5600 are routed through connectors to interface with the microcontroller on the PCB.

5) Port Circuit



The schematic diagram showcases various port connections required for the stepper motor driver system, including power supply ports, control switches, debugging interfaces, and input buttons. Below is a detailed description of each section of the circuit:

1. **Power Supply Ports (Top Left):**
    - **Connectors XA1 and XA2:** These connectors (MKDS1/2-3.81) are used to supply power to the system. They provide connections for the power supply voltage (VCC) and ground (PGND).
2. **Microstepping Switch (Middle Left):**
    - **SW1 (DIP Switch):** This switch (DSH9HTC6S) is used for configuring the microstepping settings of the stepper motor driver. It connects to various input pins on the MCU to set the microstepping mode. The switch settings allow for different combinations of inputs to achieve the desired microstepping resolution.
    - but since we were unable to find an SPI encoder we had to reduce resolution ,therefor micro stepping is not included here.
3. **Programming and Debugging Connectors (Top Right):**
    - **Connector XA11:** This connector (MKDS1/2-3.81) is used for programming and debugging the microcontroller. It provides connections for the Serial Wire Debug (SWD) interface, including SWDIO, SWCLK, and GND.
    - **SPI Connector:** The SPI interface for control inputs is also provided through this connector, with pins for SPI_MISO, SPI_MOSI, and SPI_SCK.
4. **CAN Bus Interface (Bottom Right):**
    - **Connector J2 (70541-0009):** This connector provides an interface for the CAN bus, which is used for communication with other devices. It includes connections for CAN_H, CAN_L, and GND.
    - **Termination Resistor R16:** A 120Ω resistor is used for terminating the CAN bus to ensure proper communication.
5. **Control Buttons (Bottom Left):**
    - **Buttons S1 and S2 (TS-02-BS-128-10B-SMT):** These buttons are used for user inputs to the MCU. They are connected to the MCU's GPIO pins for triggering various control functions or commands.

PCB Layout

Design Considerations

1. Trace Width:

- **Range:** The trace widths in the design range from 0.25mm to 1mm, with larger widths used for power tracing.
- **Justification:**
  - **Current Capacity:** Wider traces (up to 1mm) are used for power lines to handle the higher current flow (nearly 4 Amperes) without excessive heating or voltage drop.
  - **Signal Integrity:** Narrower traces (0.25mm) are used for signal lines where the current is much lower, ensuring minimal interference and maintaining signal integrity.

2. Grounding:

- **Ground Net:** A dedicated ground net is used to provide a common reference point for all components, ensuring stable operation.
- **Purpose:**
  - **Noise Reduction:** A solid ground plane helps reduce electromagnetic interference (EMI) and provides a low impedance path for return currents, improving overall signal quality.
  - **Current Return Path:** Ensures that all return currents have a low-resistance path, reducing ground loops and potential noise issues.

3. Coupling Capacitors:

- **Techniques Used:** Coupling capacitors are strategically placed near the ICs.
- **Purpose:**
  - **Protection Against Current Spikes:** These capacitors help filter out transient spikes and noise from the power supply, protecting sensitive components.
  - **Power Supply Stability:** They stabilize the power supply voltage by providing local energy storage and smoothing out voltage fluctuations.

Layers Used and Their Purposes

2. Top Layer:

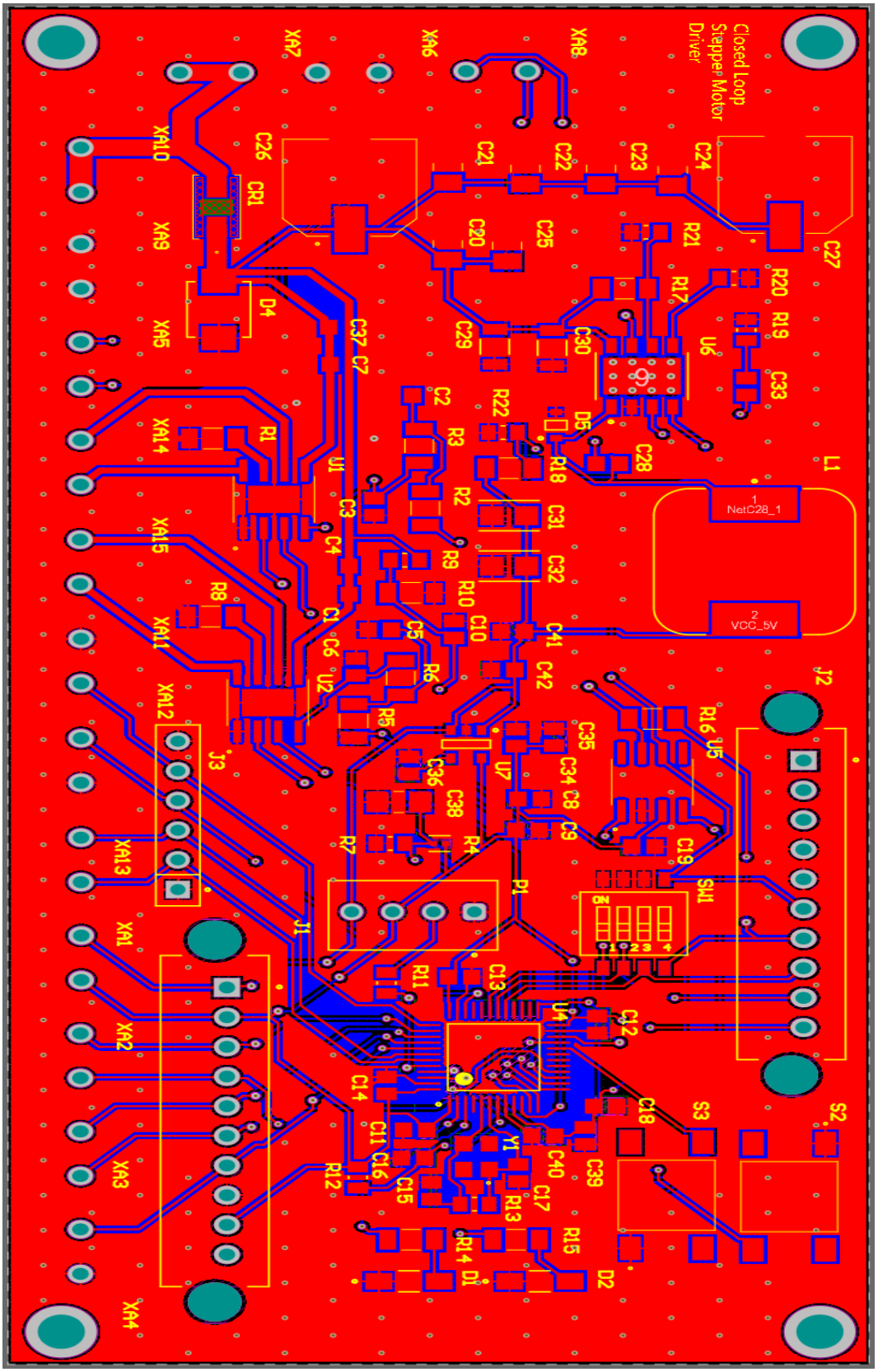
- **Purpose:**
  - **Component Placement:** The top layer is used for placing most of the components, including the microcontroller, motor driver IC, sensors, and connectors.
  - **Signal Routing:** Primary signal traces are routed on the top layer to minimize the complexity of the layout and maintain short signal paths for critical signals.

3. Ground Layer:

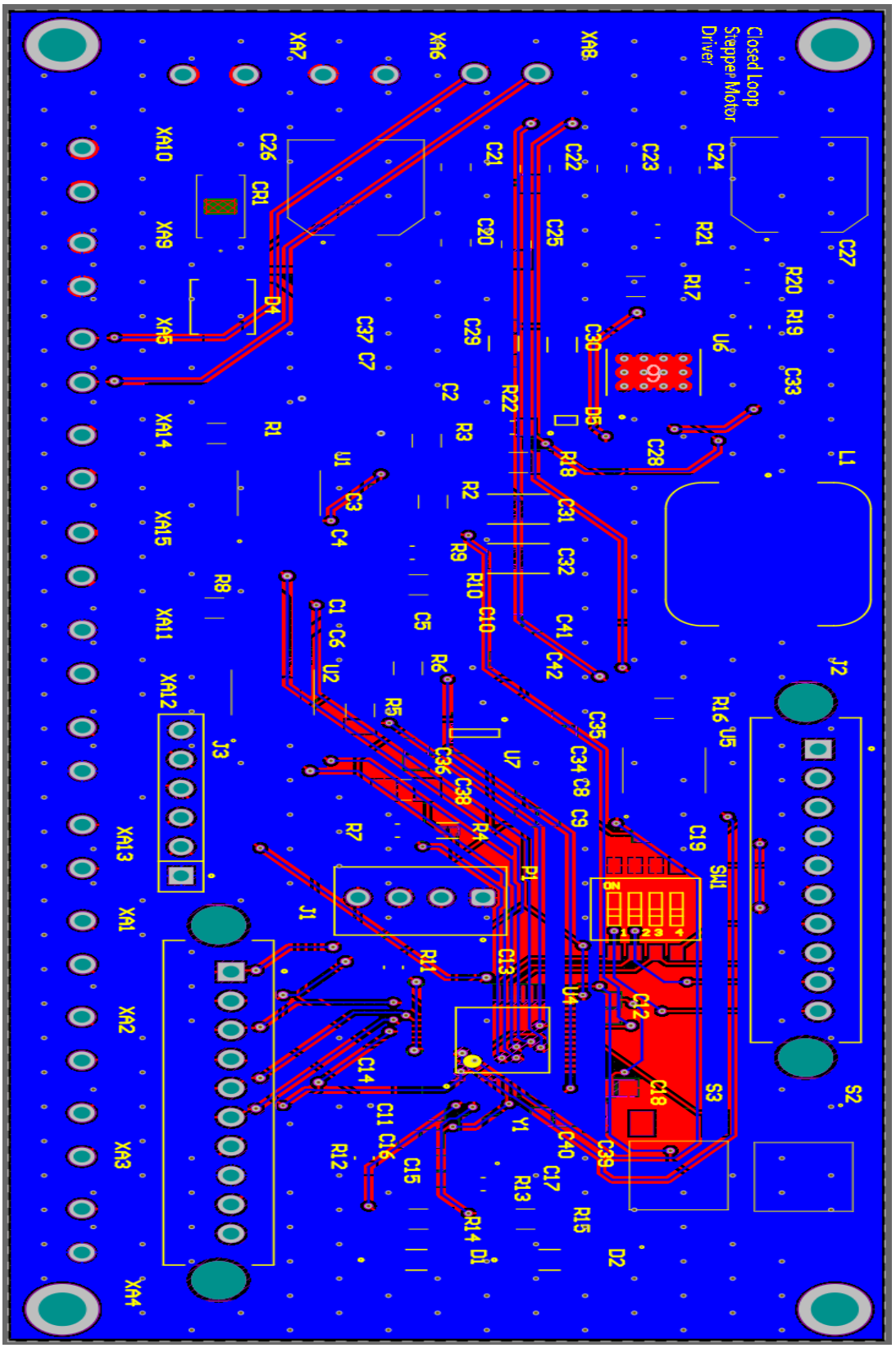
- **Purpose:**
  - **Ground Plane:** The entire layer is dedicated to the ground plane, providing a continuous and low-impedance ground reference for the entire circuit.
  - **Noise Reduction:** The ground plane helps in shielding and reducing electromagnetic interference (EMI) by providing a uniform return path for signals and power supply currents.

The design considerations ensure that the stepper motor driver is both robust and efficient. By carefully selecting trace widths, grounding techniques, and using coupling capacitors, the design addresses key electrical and thermal challenges. The use of dedicated layers for component placement and grounding further enhances the reliability and performance of the circuit. This meticulous approach ensures that the stepper motor driver can handle high currents, maintain signal integrity, and operate reliably in various conditions.

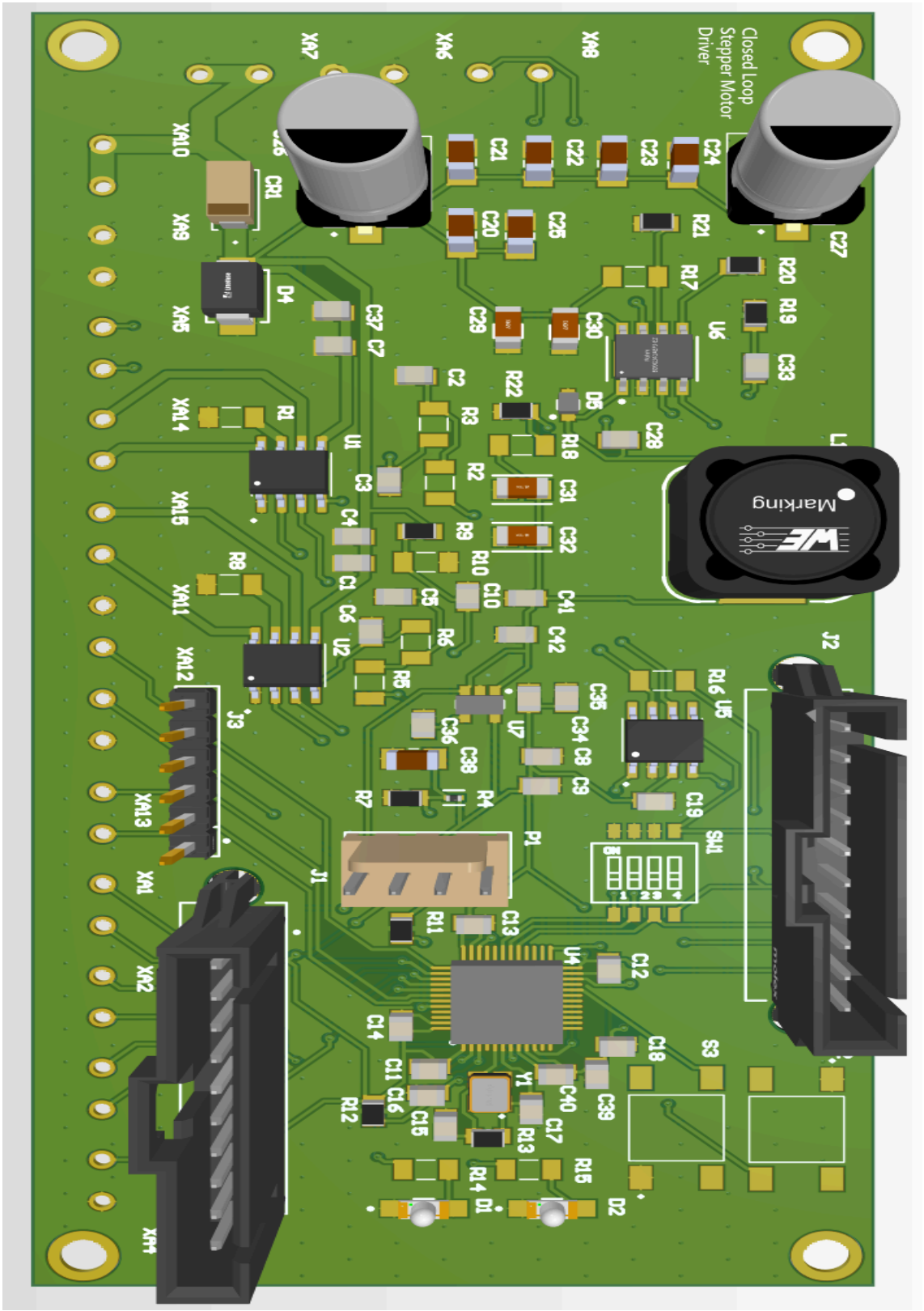
Top Layer



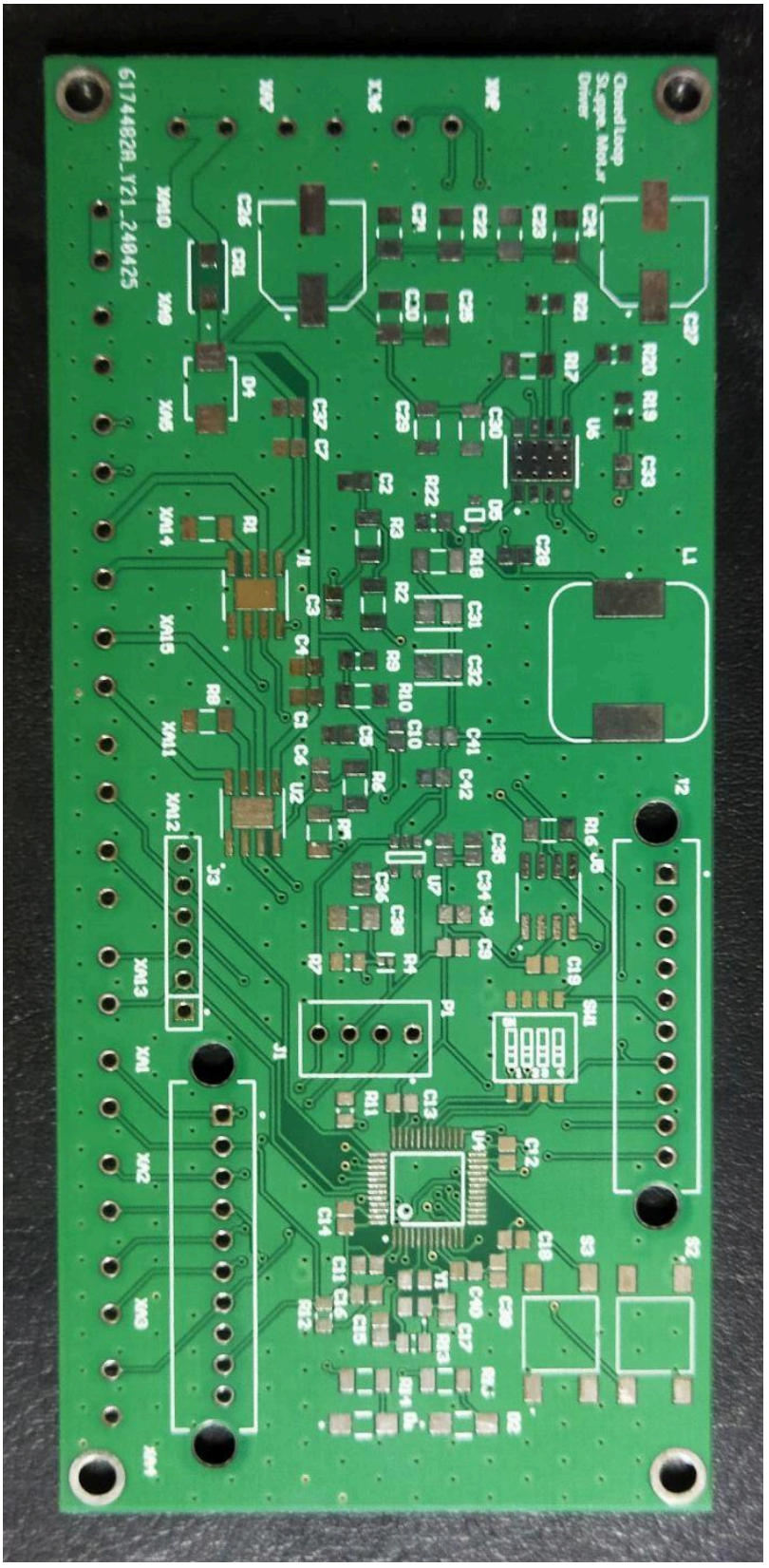
Bottom Layer

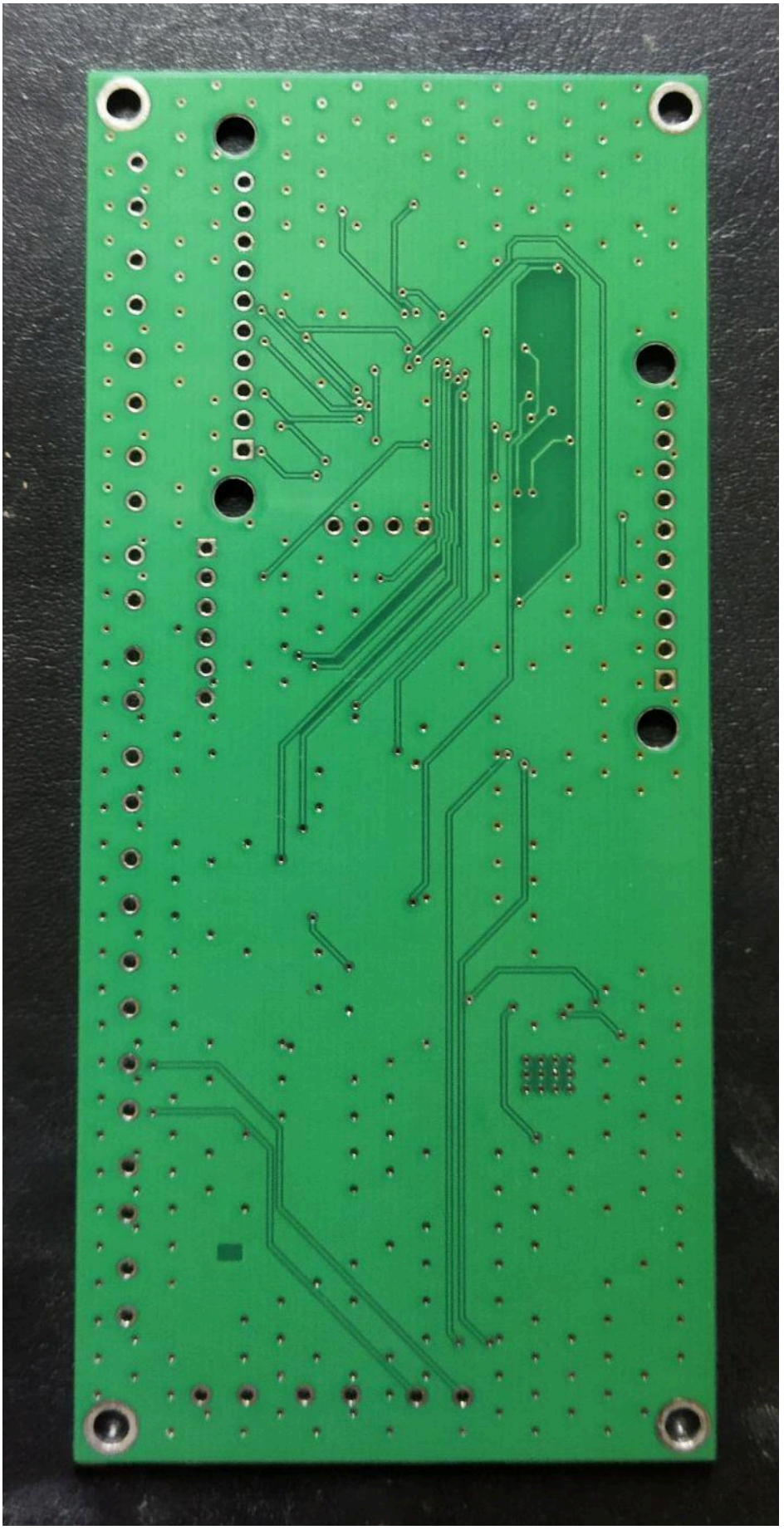


3D View of the PCB

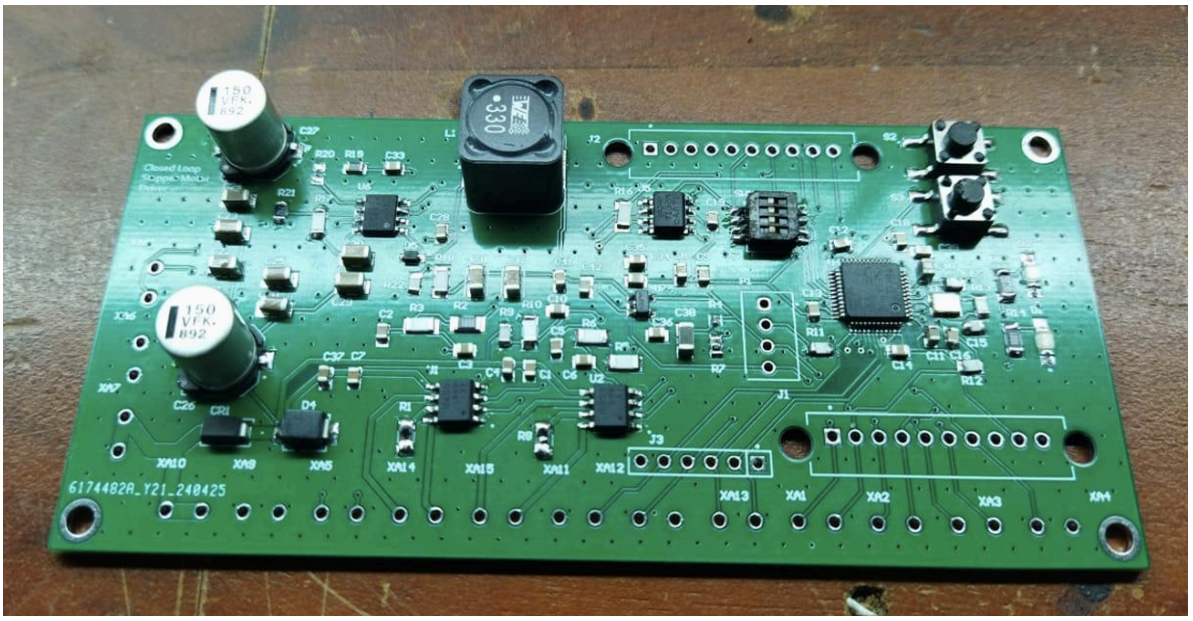


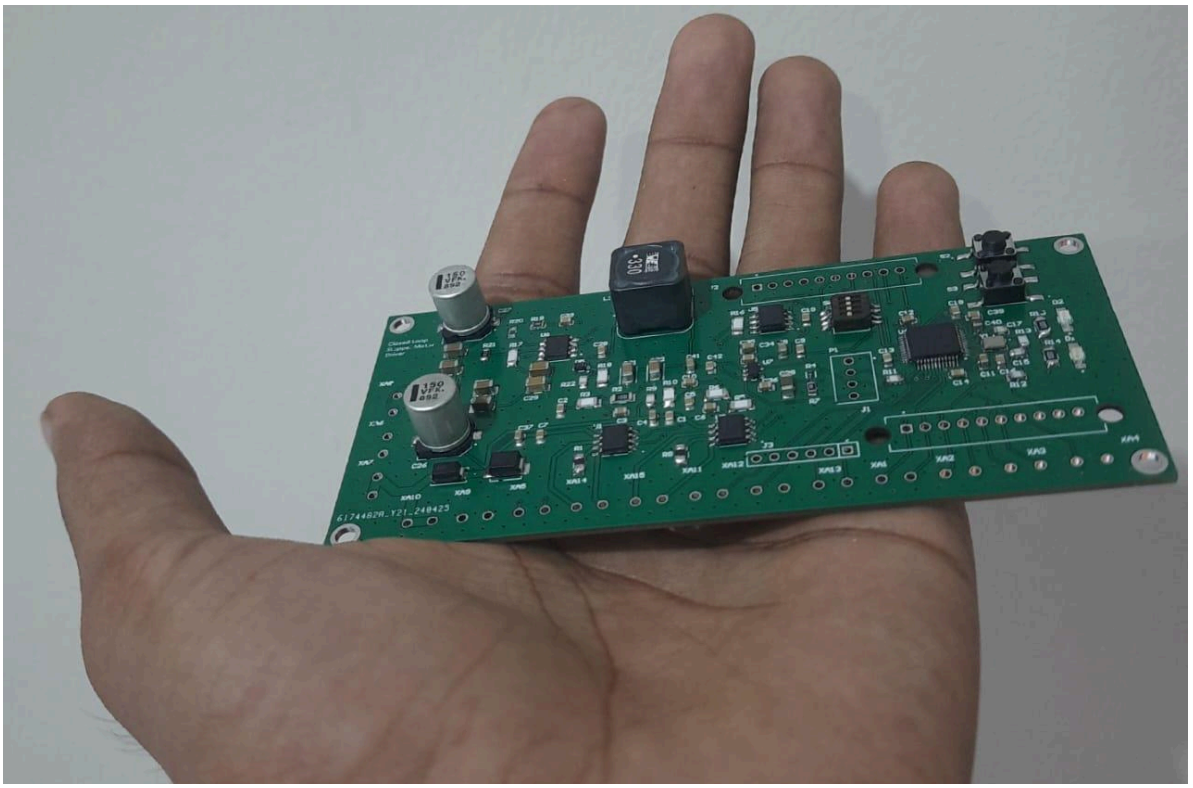
Photographs of bare PCB





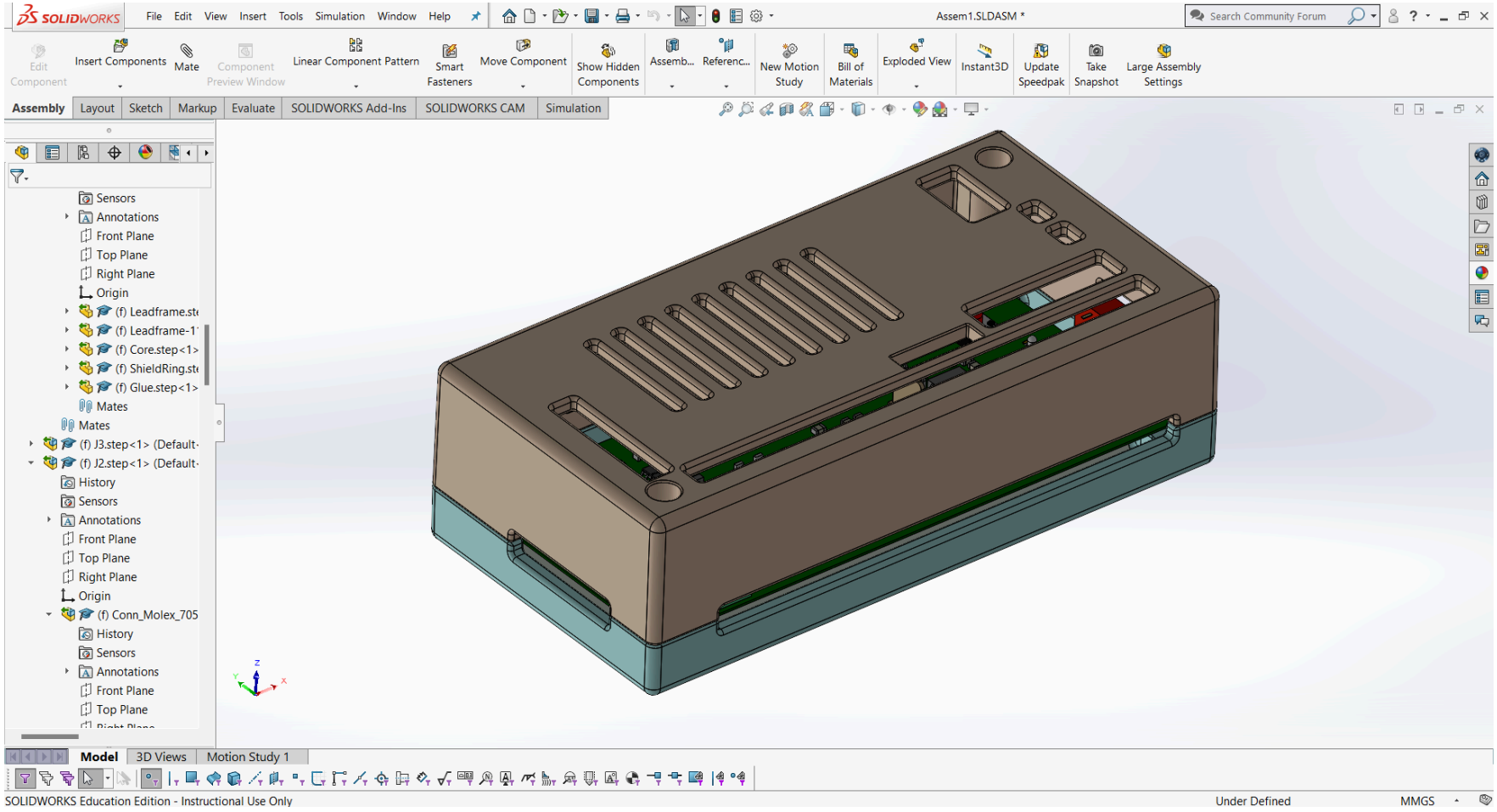
Photograph of soldered PCB

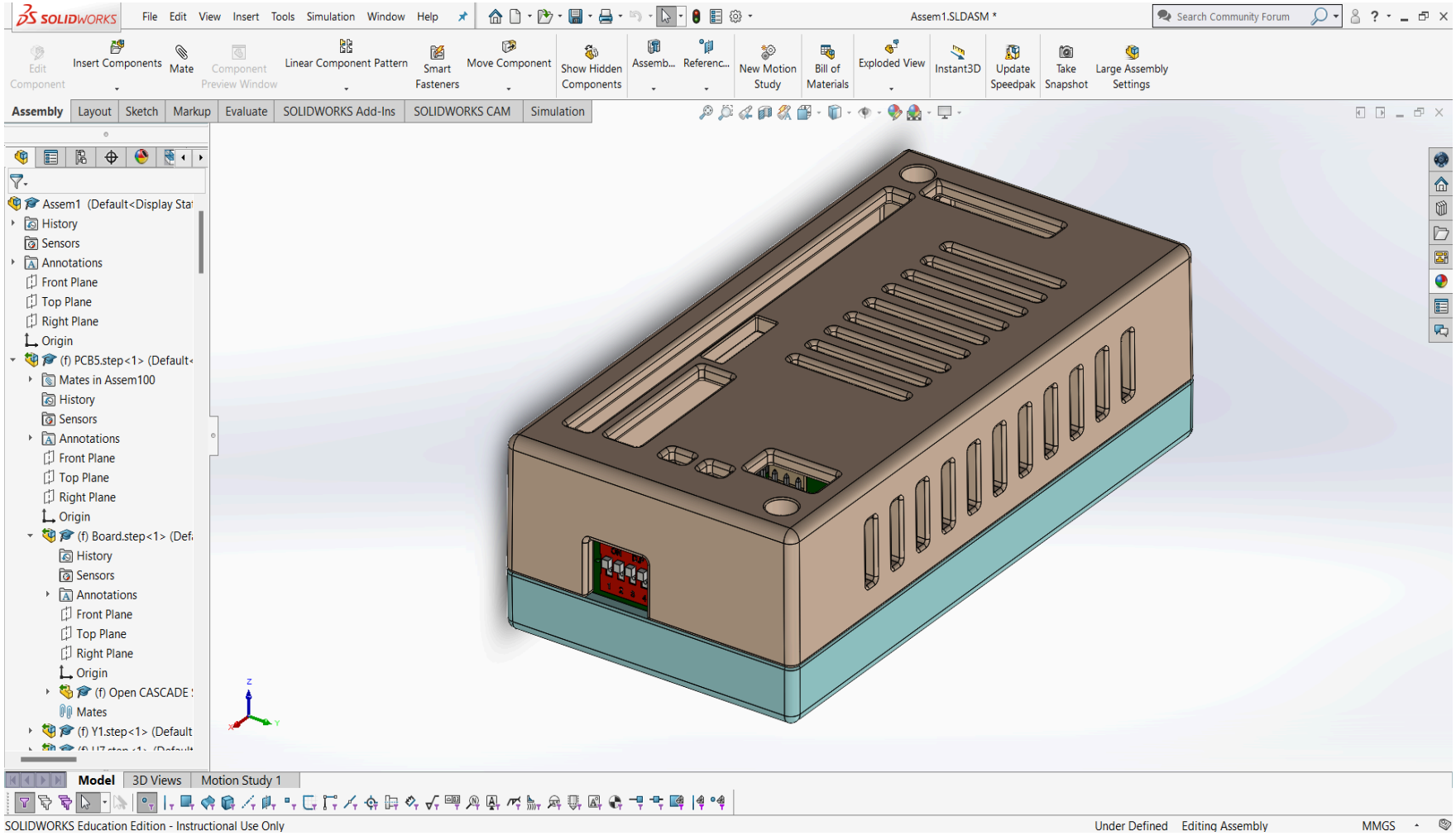


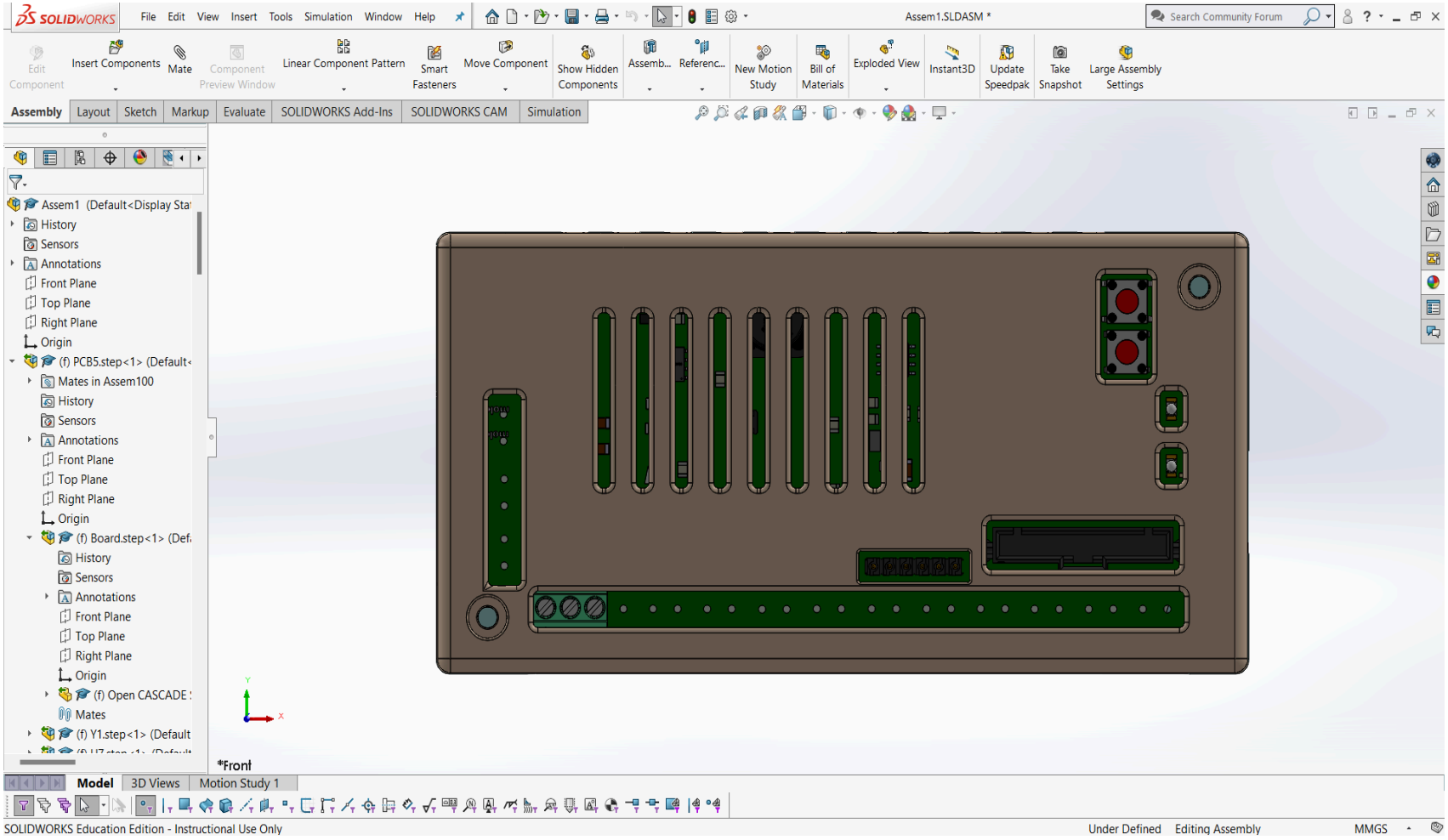


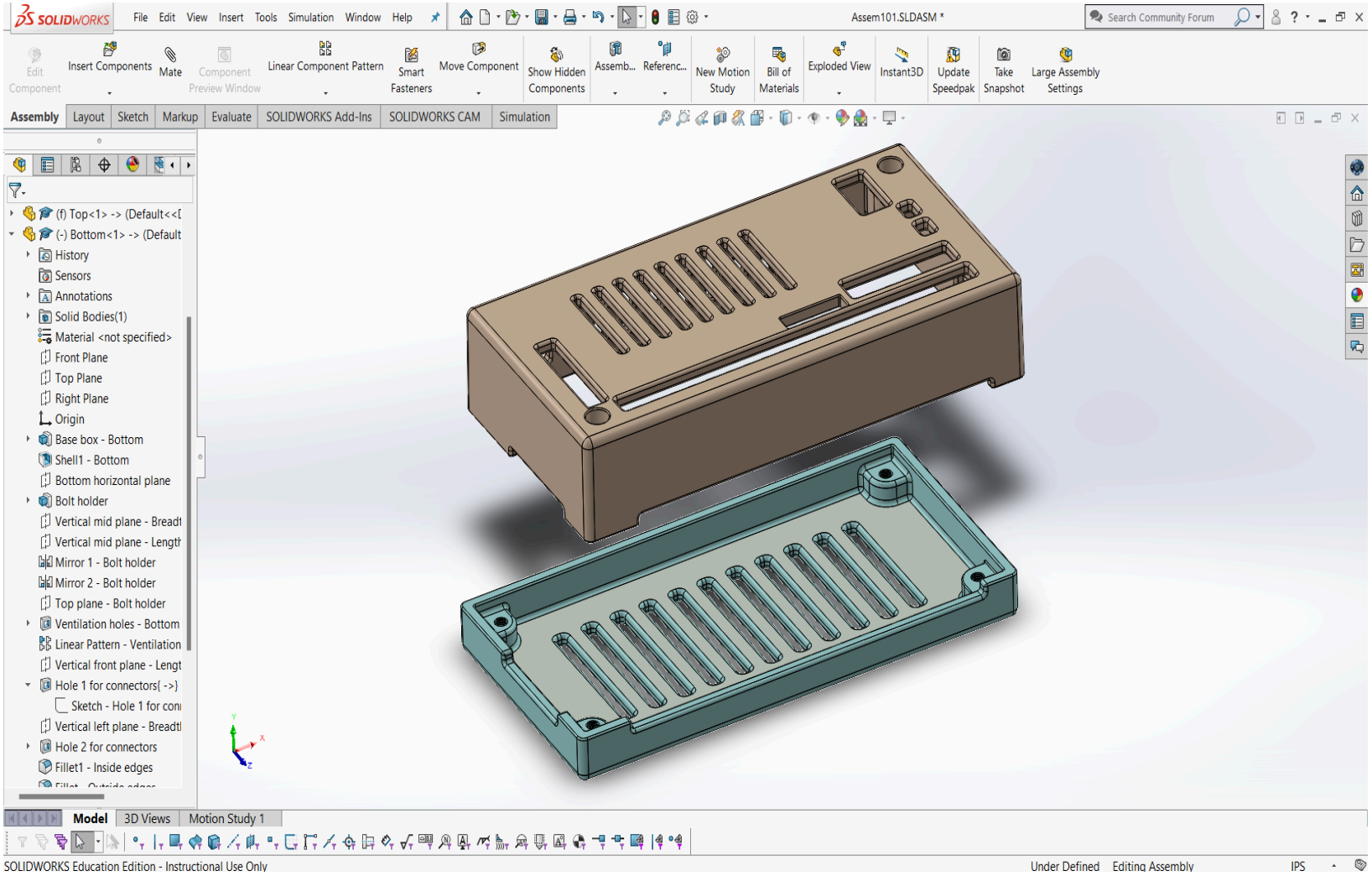
Enclosure design

Enclosure Assembly

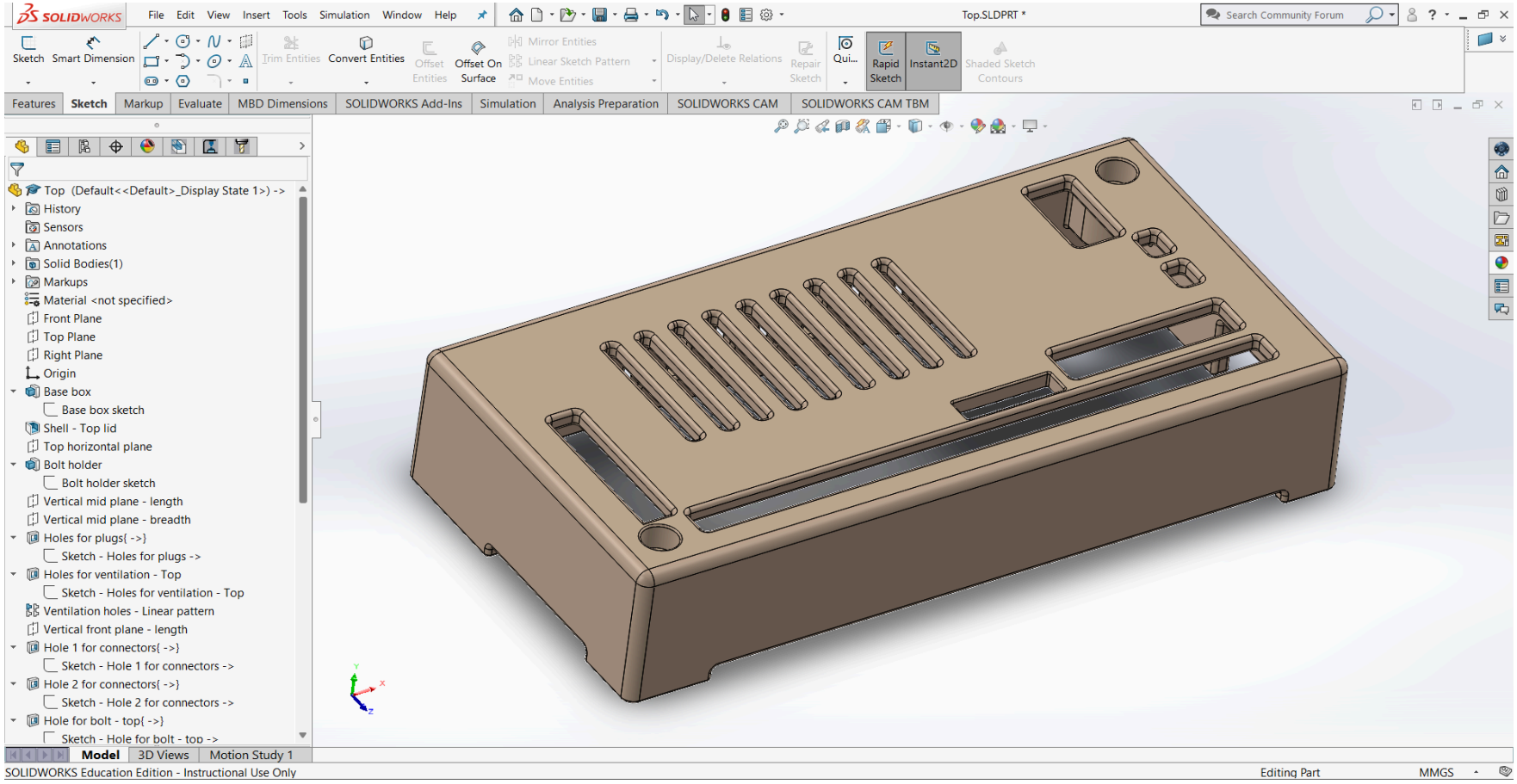


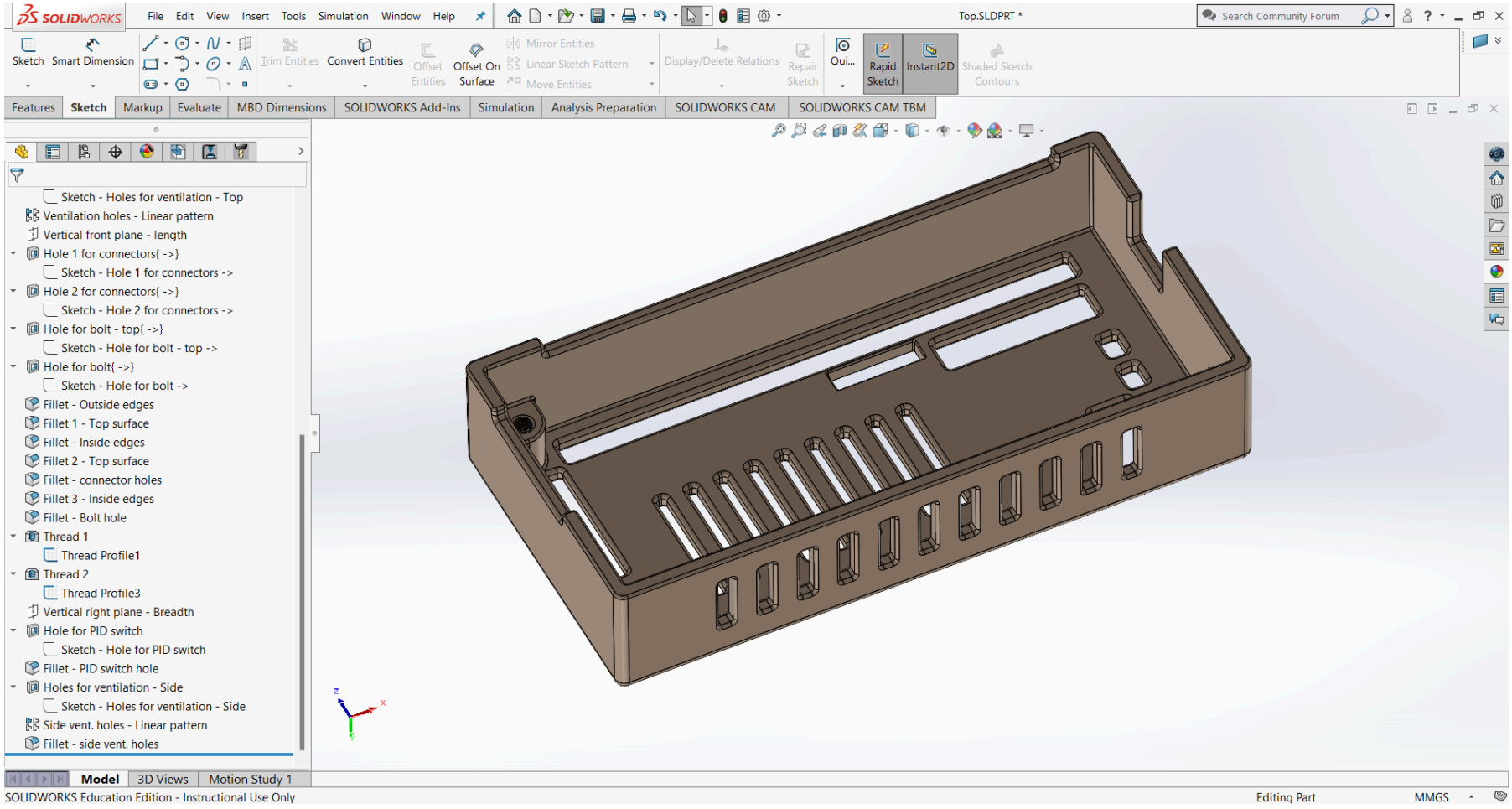

SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Assem1.SLDASM *
Search Community Forum
Edit Component
Insert Components
Mate
Component Preview Window
Linear Component Pattern
Smart Fasteners
Move Component
Show Hidden Components
Assemb...
Referenc...
New Motion Study
Bill of Materials
Exploded View
Instant3D
Update Speedpak
Take Snapshot
Large Assembly Settings
Assembly Layout Sketch Markup Evaluate SOLIDWORKS Add-Ins SOLIDWORKS CAM Simulation
Assem1 (Default<Display Sta
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) PCB5.step<1> (Default
Mates in Assem100
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Board.step<1> (Def
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Open CASCADE
Mates
(f) Y1.step<1> (Default
Model
3D Views
Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Under Defined
Editing Assembly
MMGS

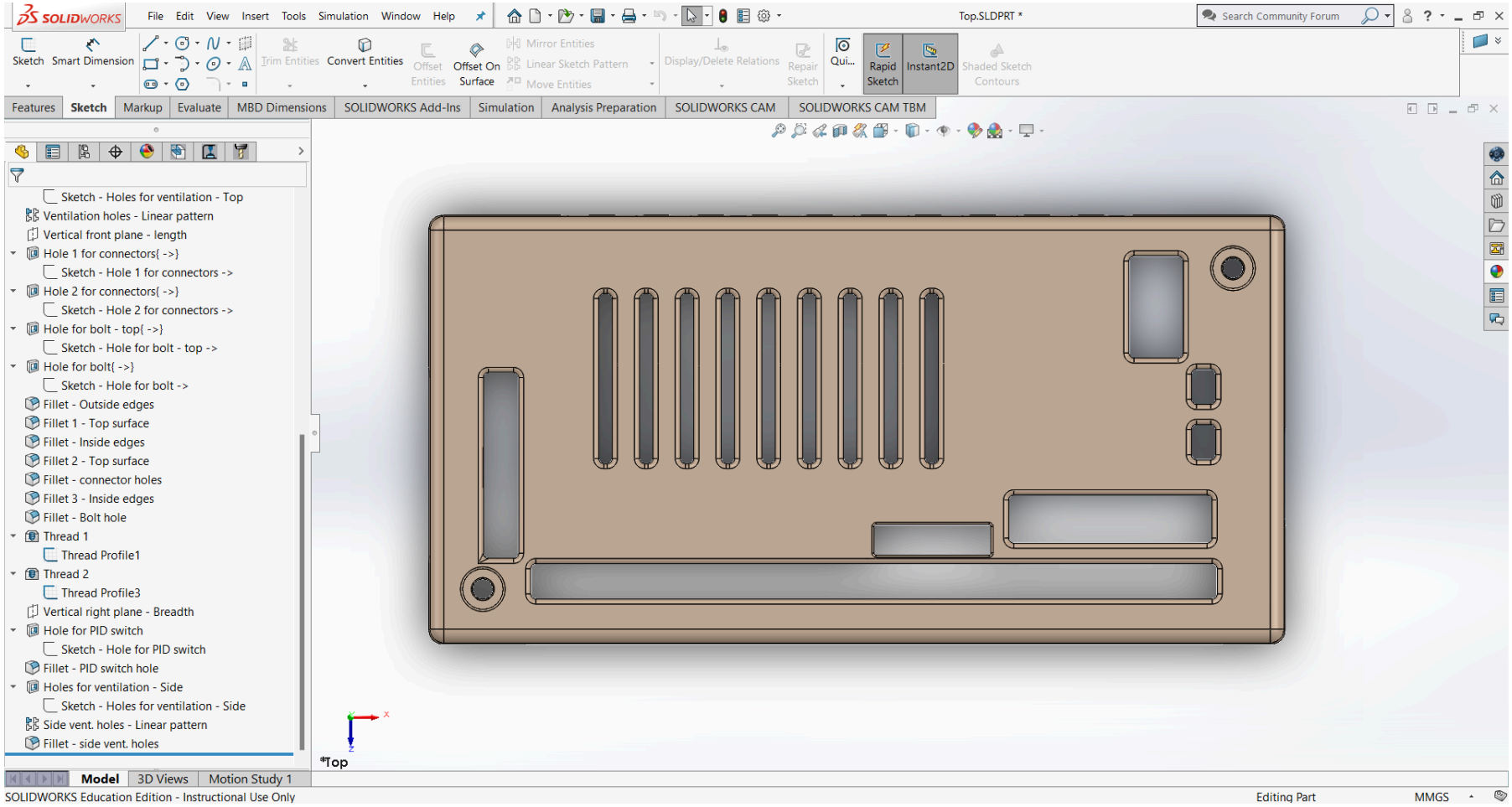

SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Assem1.SLDASM *
Search Community Forum
Edit Component
Insert Components
Mate
Component Preview Window
Linear Component Pattern
Smart Fasteners
Move Component
Show Hidden Components
Assemb...
Referenc...
New Motion Study
Bill of Materials
Exploded View
Instant3D
Update Speedpak
Take Snapshot
Large Assembly Settings
Assembly Layout Sketch Markup Evaluate SOLIDWORKS Add-Ins SOLIDWORKS CAM Simulation
Assem1 (Default<Display Sta
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) PCB5.step<1> (Default<
Mates in Assem100
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Board.step<1> (Defa
History
Sensors
Annotations
Front Plane
Top Plane
Right Plane
Origin
(f) Open CASCADE
Mates
(f) Y1.step<1> (Default
*Front
Model
3D Views
Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Under Defined
Editing Assembly
MMGS


SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Assem101.SLDASM *
Search Community Forum
Edit Component
Insert Components
Mate
Component Preview Window
Linear Component Pattern
Smart Fasteners
Move Component
Show Hidden Components
Assemb...
Referenc...
New Motion Study
Bill of Materials
Exploded View
Instant3D
Update Speedpak
Take Snapshot
Large Assembly Settings
Assembly Layout Sketch Markup Evaluate SOLIDWORKS Add-Ins SOLIDWORKS CAM Simulation
(f) Top<1> -> (Default<<[
(-) Bottom<1> -> (Default
History
Sensors
Annotations
Solid Bodies(1)
Material <not specified>
Front Plane
Top Plane
Right Plane
Origin
Base box - Bottom
Shell1 - Bottom
Bottom horizontal plane
Bolt holder
Vertical mid plane - Breadt
Vertical mid plane - Lengtl
Mirror 1 - Bolt holder
Mirror 2 - Bolt holder
Top plane - Bolt holder
Ventilation holes - Bottom
Linear Pattern - Ventilation
Vertical front plane - Lengt
Hole 1 for connectors{ ->}
Sketch - Hole 1 for con
Vertical left plane - Breadtl
Hole 2 for connectors
Fillet1 - Inside edges
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Under Defined Editing Assembly IPS

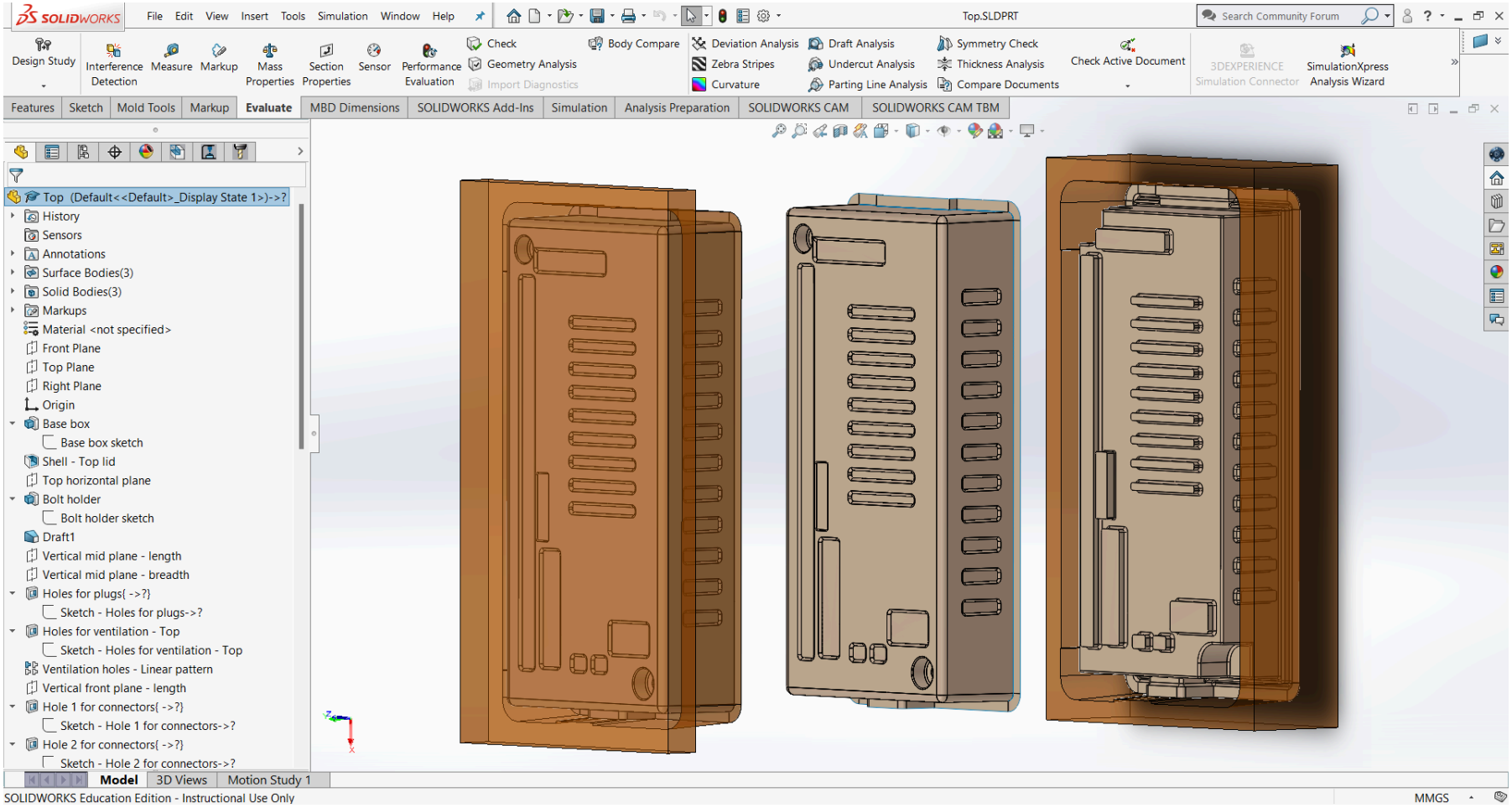
Top Part

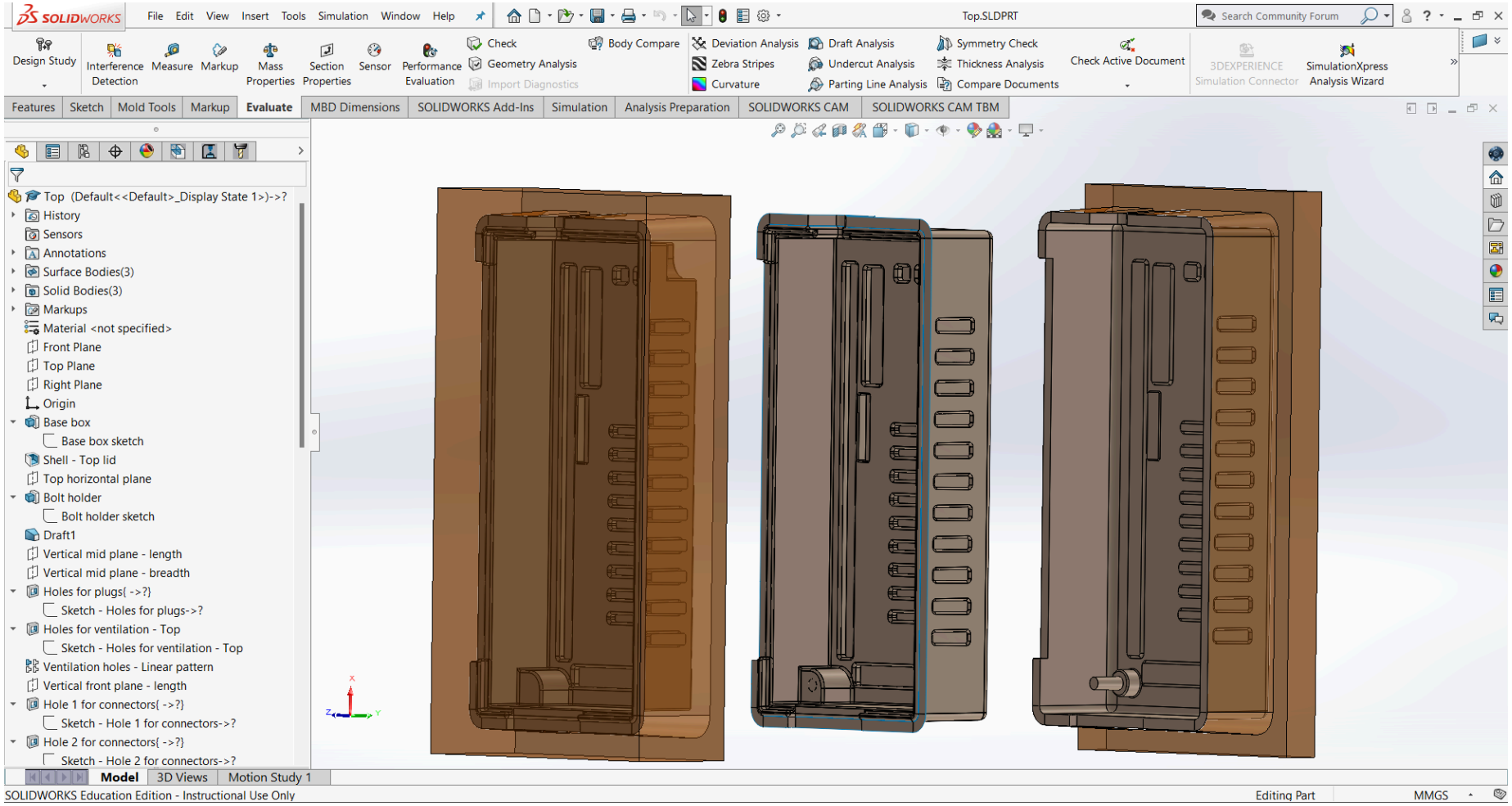

SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Top.SLDPRT *
Search Community Forum
Sketch
Smart Dimension
Trim Entities
Convert Entities
Offset Entities
Offset On Surface
Mirror Entities
Linear Sketch Pattern
Move Entities
Display/Delete Relations
Repair Sketch
Rapid Sketch
Instant2D
Shaded Sketch Contours
Features Sketch Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Top (Default<<Default>_Display State 1>) ->
History
Sensors
Annotations
Solid Bodies(1)
Markups
Material <not specified>
Front Plane
Top Plane
Right Plane
Origin
Base box
Base box sketch
Shell - Top lid
Top horizontal plane
Bolt holder
Bolt holder sketch
Vertical mid plane - length
Vertical mid plane - breadth
Holes for plugs{ ->}
Sketch - Holes for plugs ->
Holes for ventilation - Top
Sketch - Holes for ventilation - Top
Ventilation holes - Linear pattern
Vertical front plane - length
Hole 1 for connectors{ ->}
Sketch - Hole 1 for connectors ->
Hole 2 for connectors{ ->}
Sketch - Hole 2 for connectors ->
Hole for bolt - top{ ->}
Sketch - Hole for bolt - top ->
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS


SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Top.SLDPRT *
Search Community Forum
Sketch
Smart Dimension
Trim Entities
Convert Entities
Offset Entities
Offset On Surface
Mirror Entities
Linear Sketch Pattern
Move Entities
Display/Delete Relations
Repair Sketch
Rapid Sketch
Instant2D
Shaded Sketch Contours
Features Sketch Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Sketch - Holes for ventilation - Top
Ventilation holes - Linear pattern
Vertical front plane - length
Hole 1 for connectors{ ->}
Sketch - Hole 1 for connectors ->
Hole 2 for connectors{ ->}
Sketch - Hole 2 for connectors ->
Hole for bolt - top{ ->}
Sketch - Hole for bolt - top ->
Hole for bolt{ ->}
Sketch - Hole for bolt ->
Fillet - Outside edges
Fillet 1 - Top surface
Fillet - Inside edges
Fillet 2 - Top surface
Fillet - connector holes
Fillet 3 - Inside edges
Fillet - Bolt hole
Thread 1
Thread Profile1
Thread 2
Thread Profile3
Vertical right plane - Breadth
Hole for PID switch
Sketch - Hole for PID switch
Fillet - PID switch hole
Holes for ventilation - Side
Sketch - Holes for ventilation - Side
Side vent. holes - Linear pattern
Fillet - side vent. holes
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

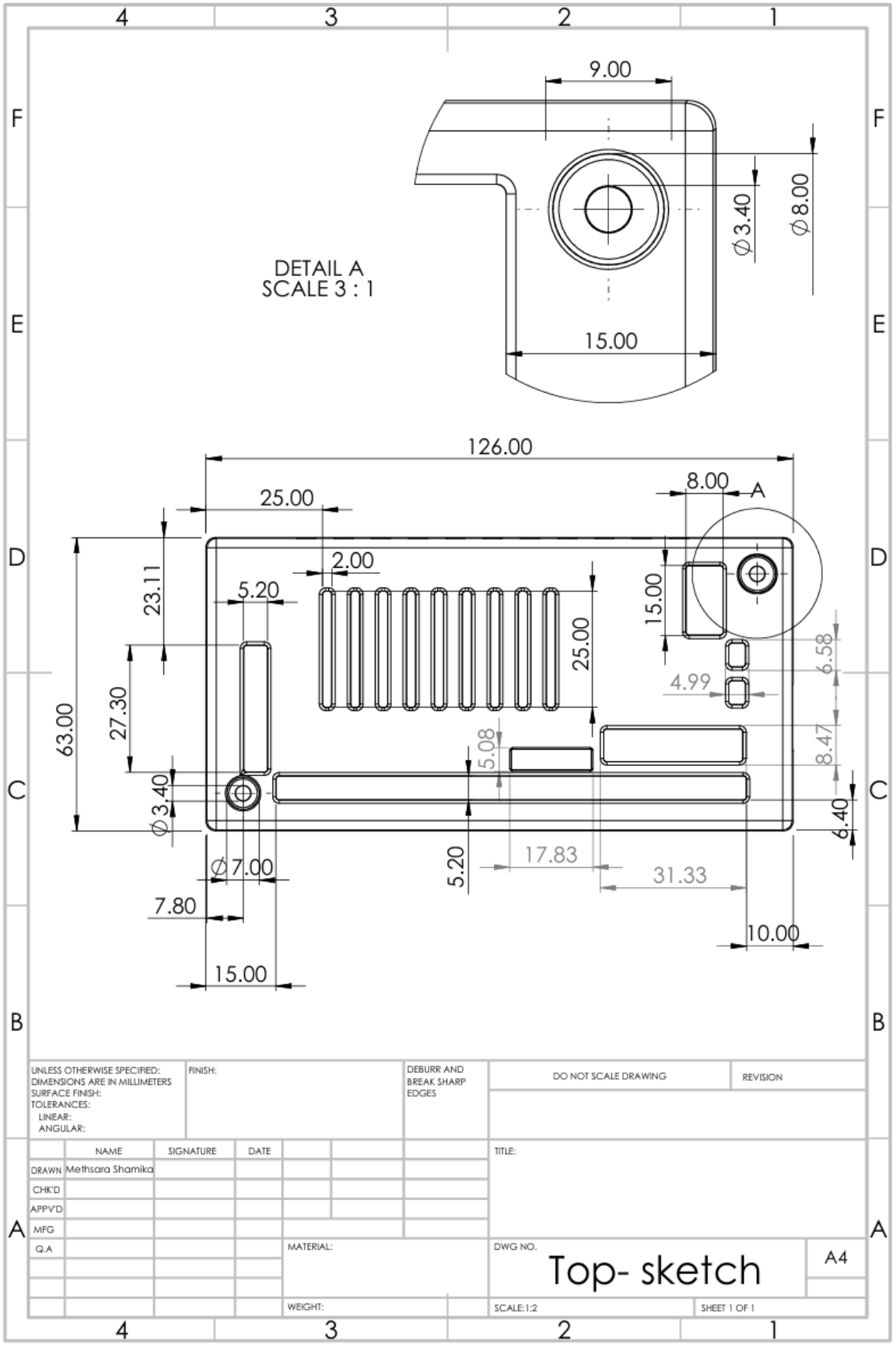

File Edit View Insert Tools Simulation Window Help
Top.SLDPRT *
Search Community Forum
Sketch
Smart Dimension
Trim Entities
Convert Entities
Offset Entities
Offset On Surface
Mirror Entities
Linear Sketch Pattern
Move Entities
Display/Delete Relations
Repair Sketch
Rapid Sketch
Instant2D
Shaded Sketch Contours
Features
Sketch
Markup
Evaluate
MBD Dimensions
SOLIDWORKS Add-Ins
Simulation
Analysis Preparation
SOLIDWORKS CAM
SOLIDWORKS CAM TBM
Sketch - Holes for ventilation - Top
Ventilation holes - Linear pattern
Vertical front plane - length
Hole 1 for connectors{ ->}
Sketch - Hole 1 for connectors ->
Hole 2 for connectors{ ->}
Sketch - Hole 2 for connectors ->
Hole for bolt - top{ ->}
Sketch - Hole for bolt - top ->
Hole for bolt{ ->}
Sketch - Hole for bolt ->
Fillet - Outside edges
Fillet 1 - Top surface
Fillet - Inside edges
Fillet 2 - Top surface
Fillet - connector holes
Fillet 3 - Inside edges
Fillet - Bolt hole
Thread 1
Thread Profile1
Thread 2
Thread Profile3
Vertical right plane - Breadth
Hole for PID switch
Sketch - Hole for PID switch
Fillet - PID switch hole
Holes for ventilation - Side
Sketch - Holes for ventilation - Side
Side vent. holes - Linear pattern
Fillet - side vent. holes
*Top
Model
3D Views
Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

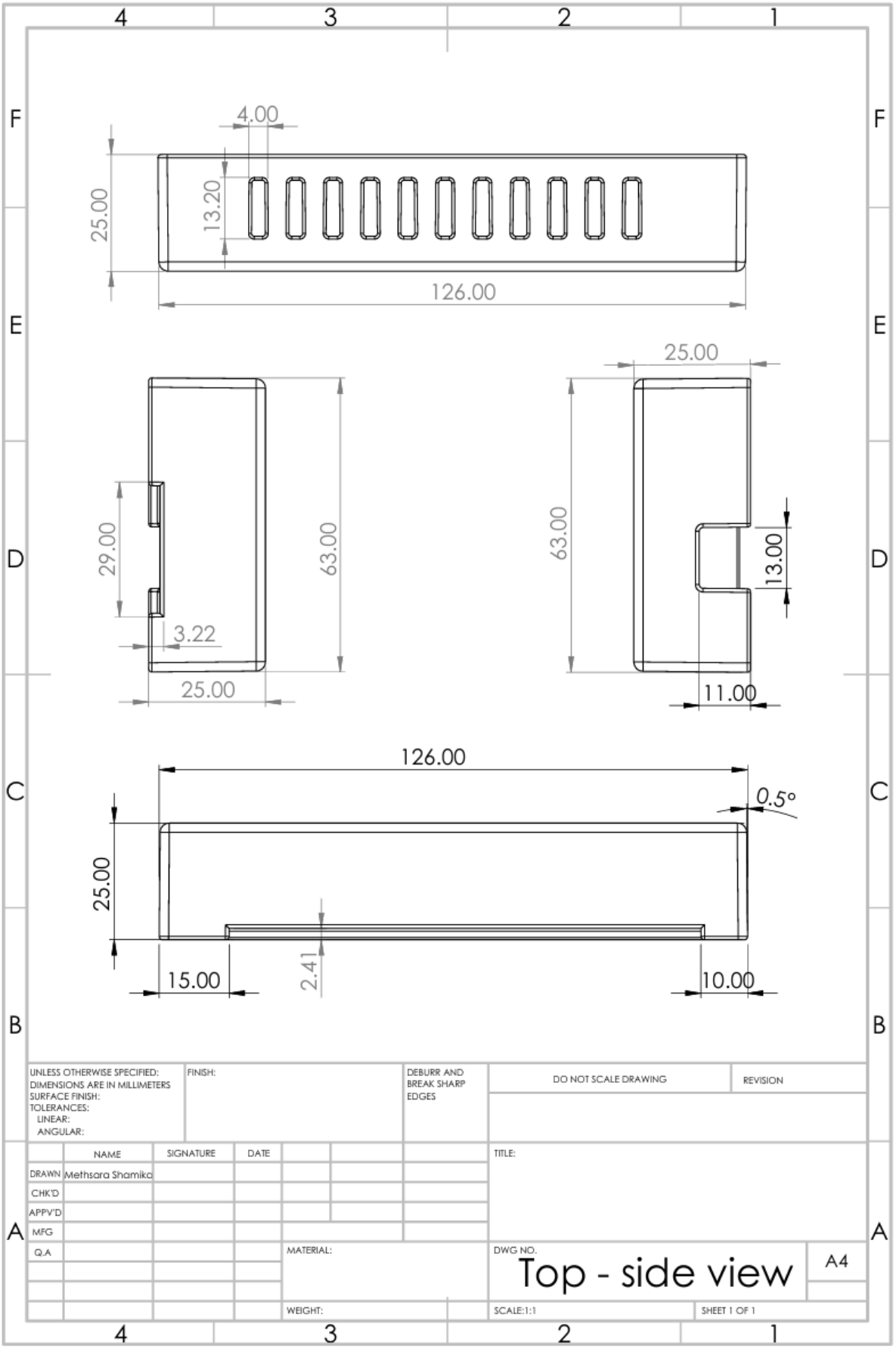
Top Part – Mold Design

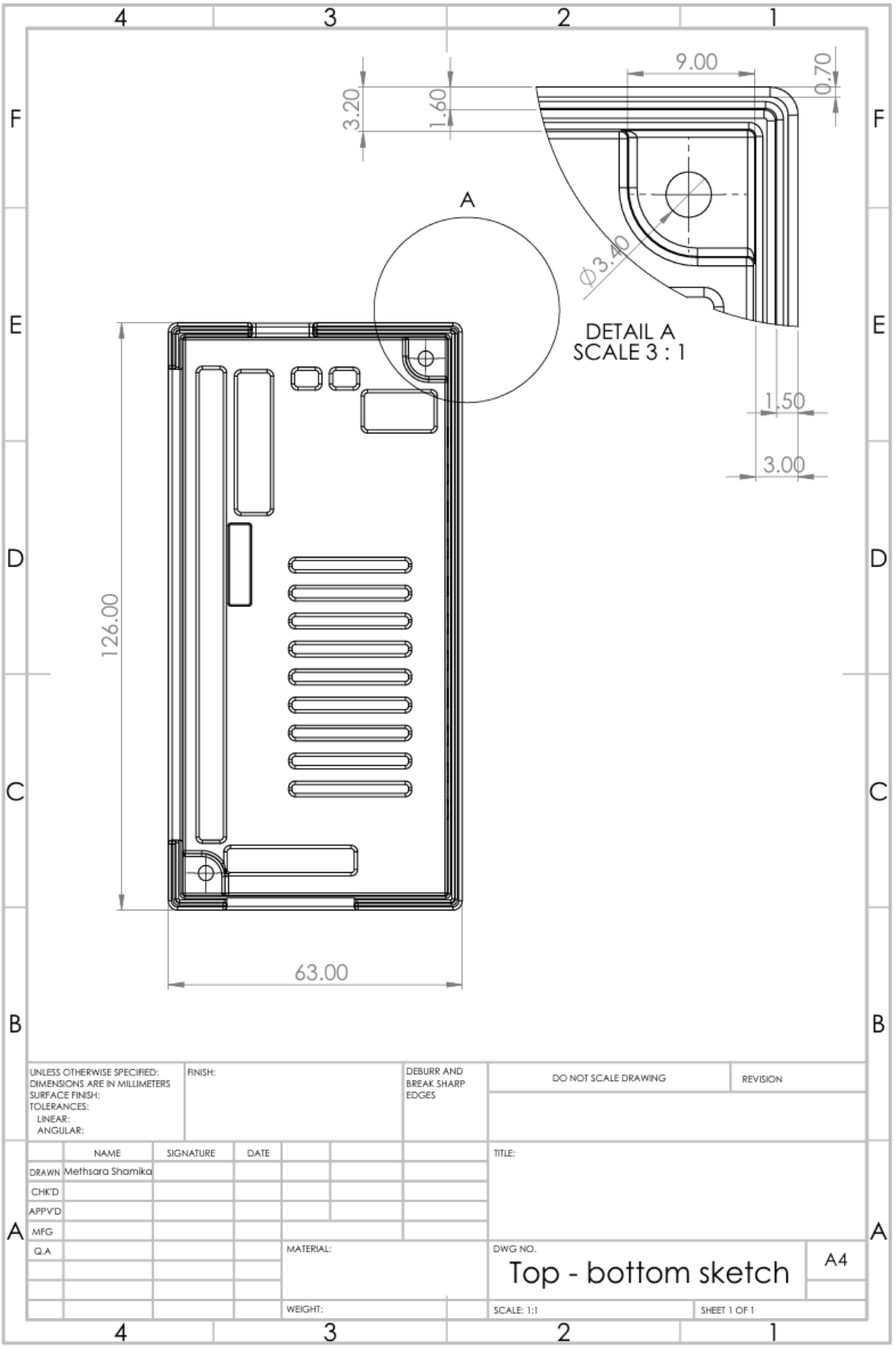



SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Top.SLDPRT
Search Community Forum
Design Study
Interference Detection
Measure
Markup
Mass Properties
Section Properties
Sensor
Performance Evaluation
Check
Geometry Analysis
Import Diagnostics
Body Compare
Deviation Analysis
Zebra Stripes
Curvature
Draft Analysis
Undercut Analysis
Parting Line Analysis
Symmetry Check
Thickness Analysis
Compare Documents
Check Active Document
3DEXPERIENCE Simulation Connector
SimulationXpress Analysis Wizard
Features Sketch Mold Tools Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Top (Default<<Default>_Display State 1>)->?
History
Sensors
Annotations
Surface Bodies(3)
Solid Bodies(3)
Markups
Material <not specified>
Front Plane
Top Plane
Right Plane
Origin
Base box
Base box sketch
Shell - Top lid
Top horizontal plane
Bolt holder
Bolt holder sketch
Draft1
Vertical mid plane - length
Vertical mid plane - breadth
Holes for plugs{ ->?}
Sketch - Holes for plugs->?
Holes for ventilation - Top
Sketch - Holes for ventilation - Top
Ventilation holes - Linear pattern
Vertical front plane - length
Hole 1 for connectors{ ->?}
Sketch - Hole 1 for connectors->?
Hole 2 for connectors{ ->?}
Sketch - Hole 2 for connectors->?
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

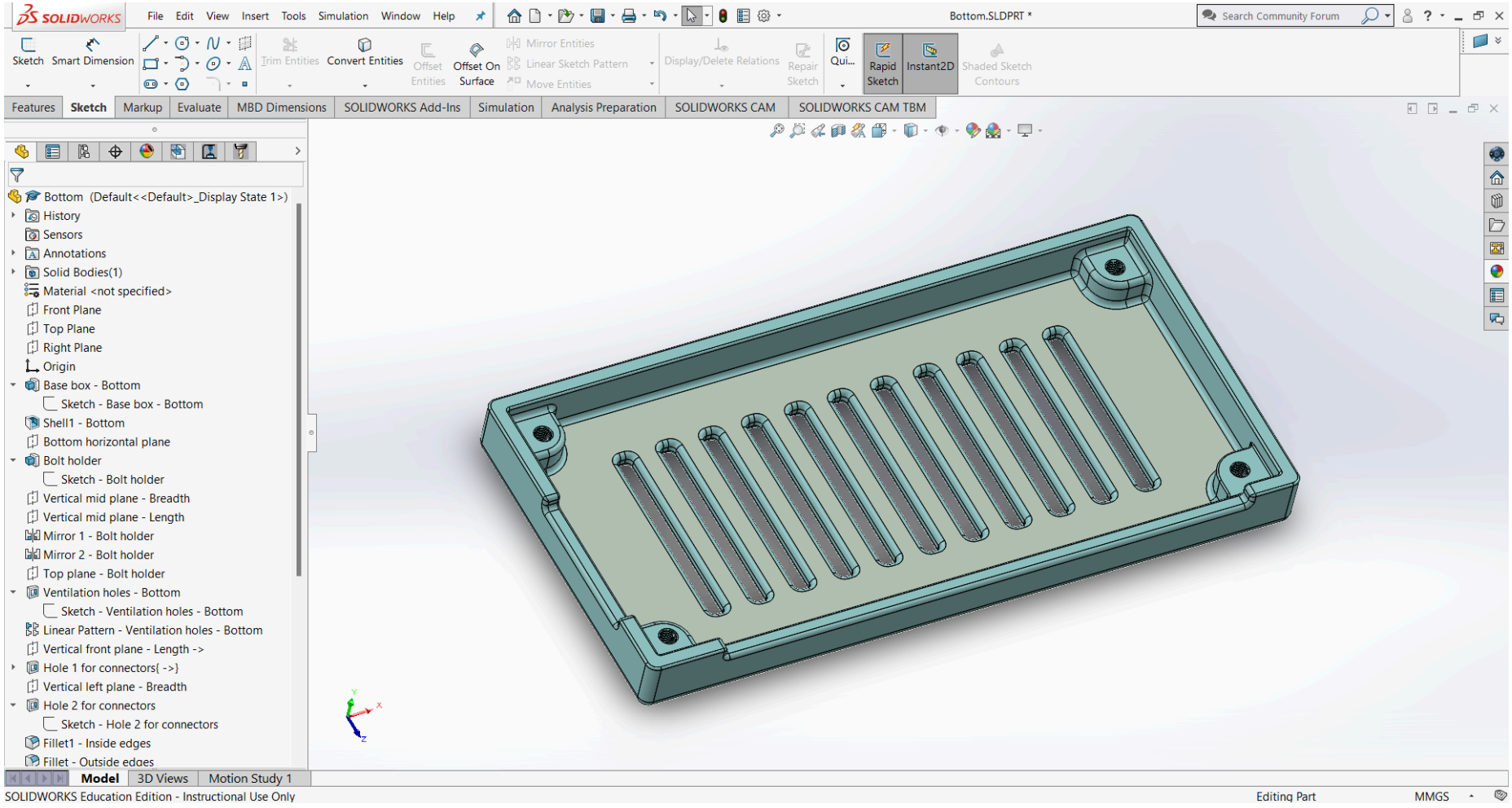
Detailed Design Drawings – Top Part

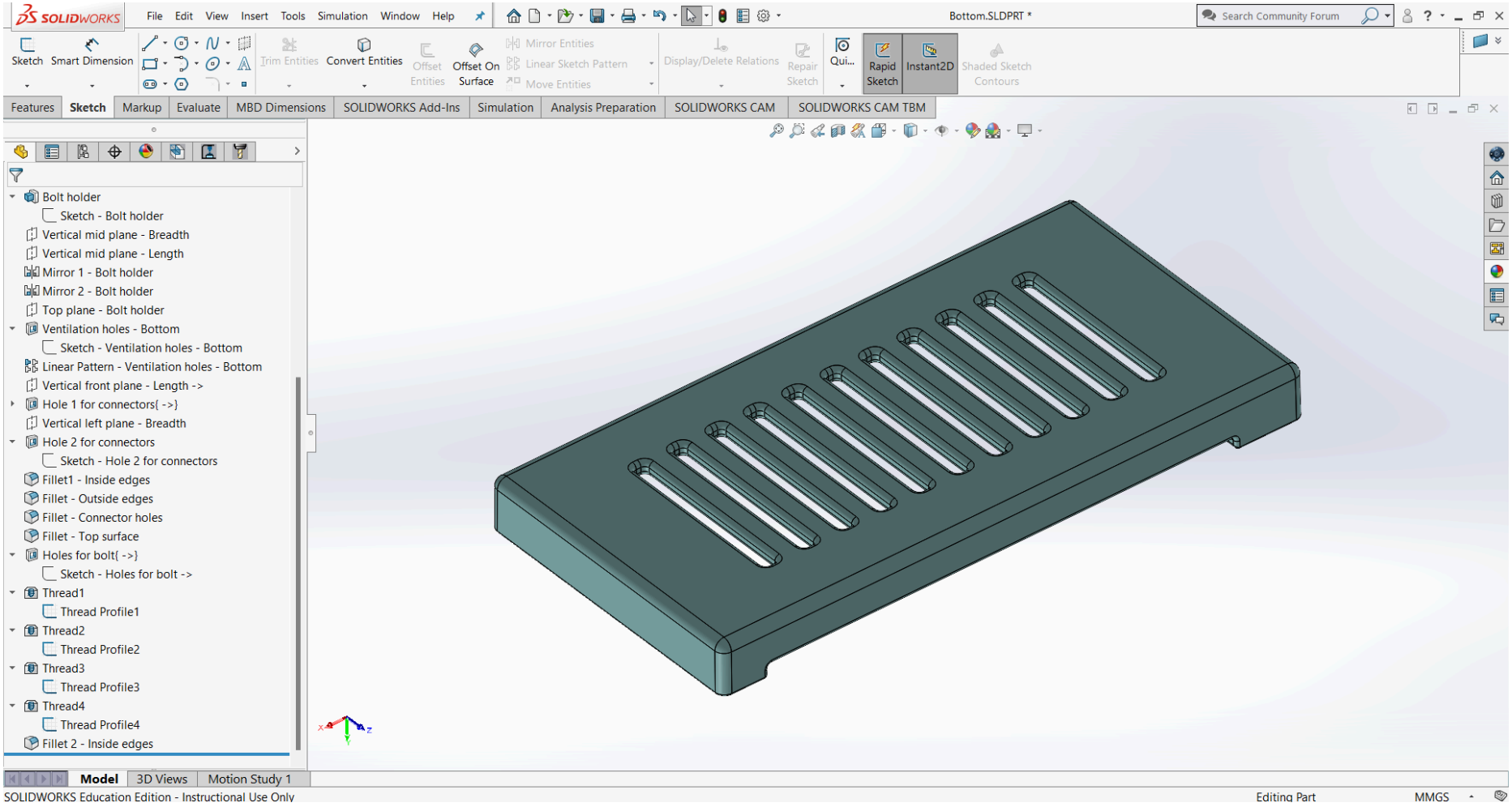



4.00
25.00
13.20
126.00
25.00
29.00
63.00
3.22
25.00
63.00
13.00
11.00
126.00
0.5°
25.00
15.00
2.41
10.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
Methsara Shamika
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
Top - side view
A4
SCALE:1:1
SHEET 1 OF 1

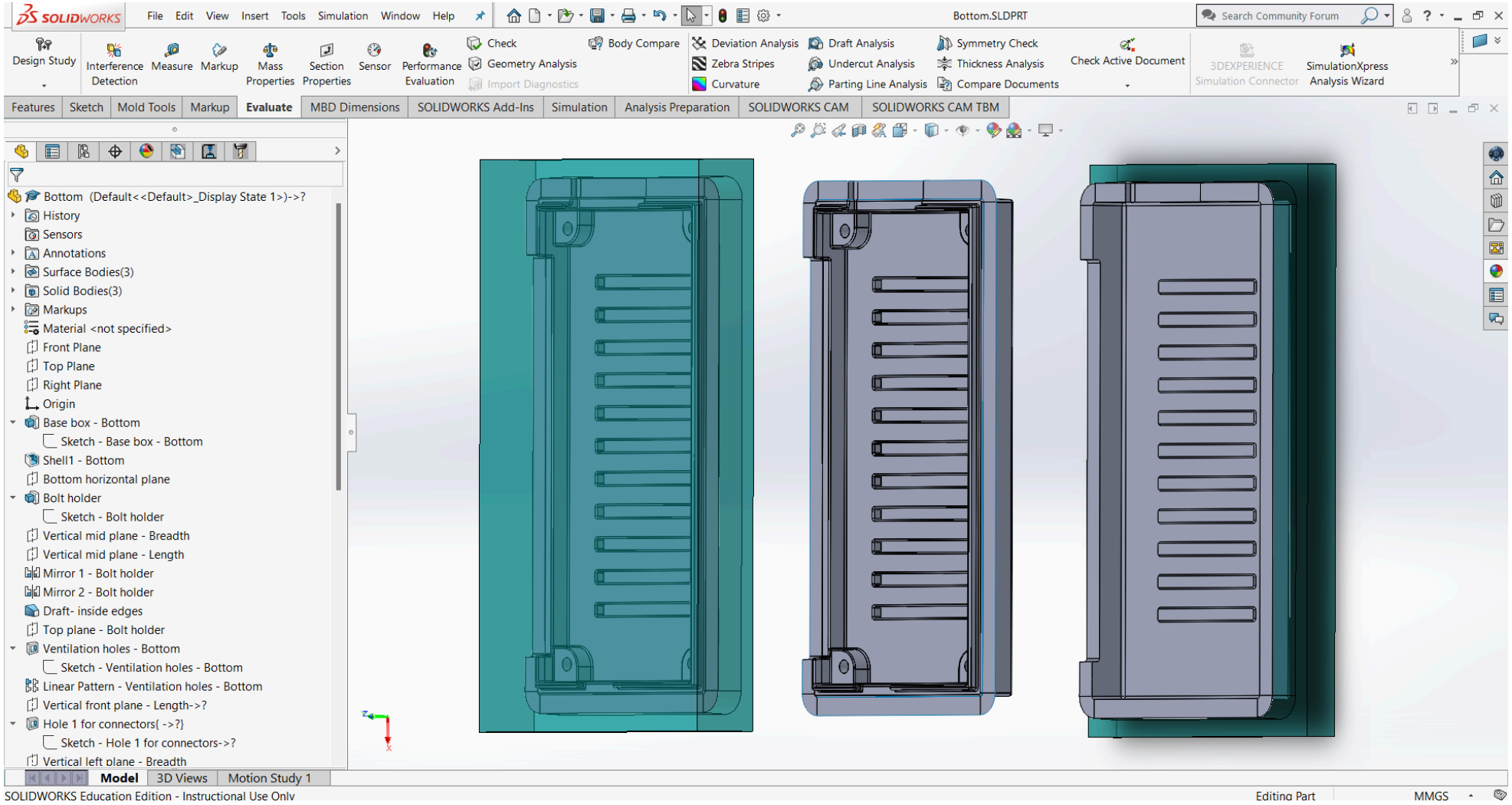

9.00
0.70
3.20
1.60
A
Ø3.40
DETAIL A
SCALE 3 : 1
1.50
3.00
126.00
63.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
Methsara Shamika
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
Top - bottom sketch
A4
SCALE: 1:1
SHEET 1 OF 1

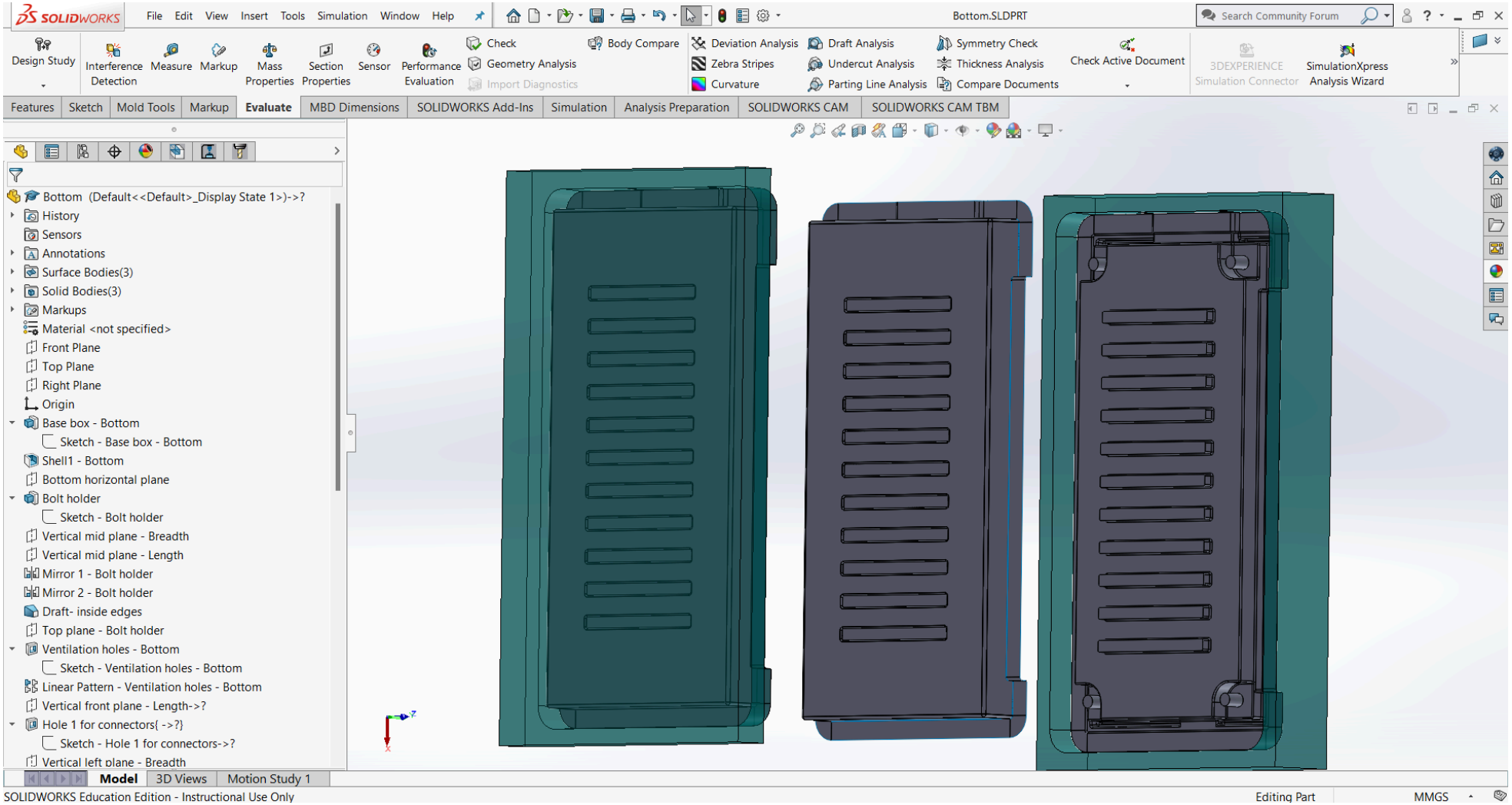
Bottom Part


SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Bottom.SLDPRT *
Search Community Forum
Sketch
Smart Dimension
Trim Entities
Convert Entities
Offset Entities
Offset On Surface
Mirror Entities
Linear Sketch Pattern
Move Entities
Display/Delete Relations
Repair Sketch
Rapid Sketch
Instant2D
Shaded Sketch Contours
Features Sketch Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Bottom (Default<<Default>_Display State 1>)
History
Sensors
Annotations
Solid Bodies(1)
Material <not specified>
Front Plane
Top Plane
Right Plane
Origin
Base box - Bottom
Sketch - Base box - Bottom
Shell1 - Bottom
Bottom horizontal plane
Bolt holder
Sketch - Bolt holder
Vertical mid plane - Breadth
Vertical mid plane - Length
Mirror 1 - Bolt holder
Mirror 2 - Bolt holder
Top plane - Bolt holder
Ventilation holes - Bottom
Sketch - Ventilation holes - Bottom
Linear Pattern - Ventilation holes - Bottom
Vertical front plane - Length ->
Hole 1 for connectors{ ->}
Vertical left plane - Breadth
Hole 2 for connectors
Sketch - Hole 2 for connectors
Fillet1 - Inside edges
Fillet - Outside edges
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

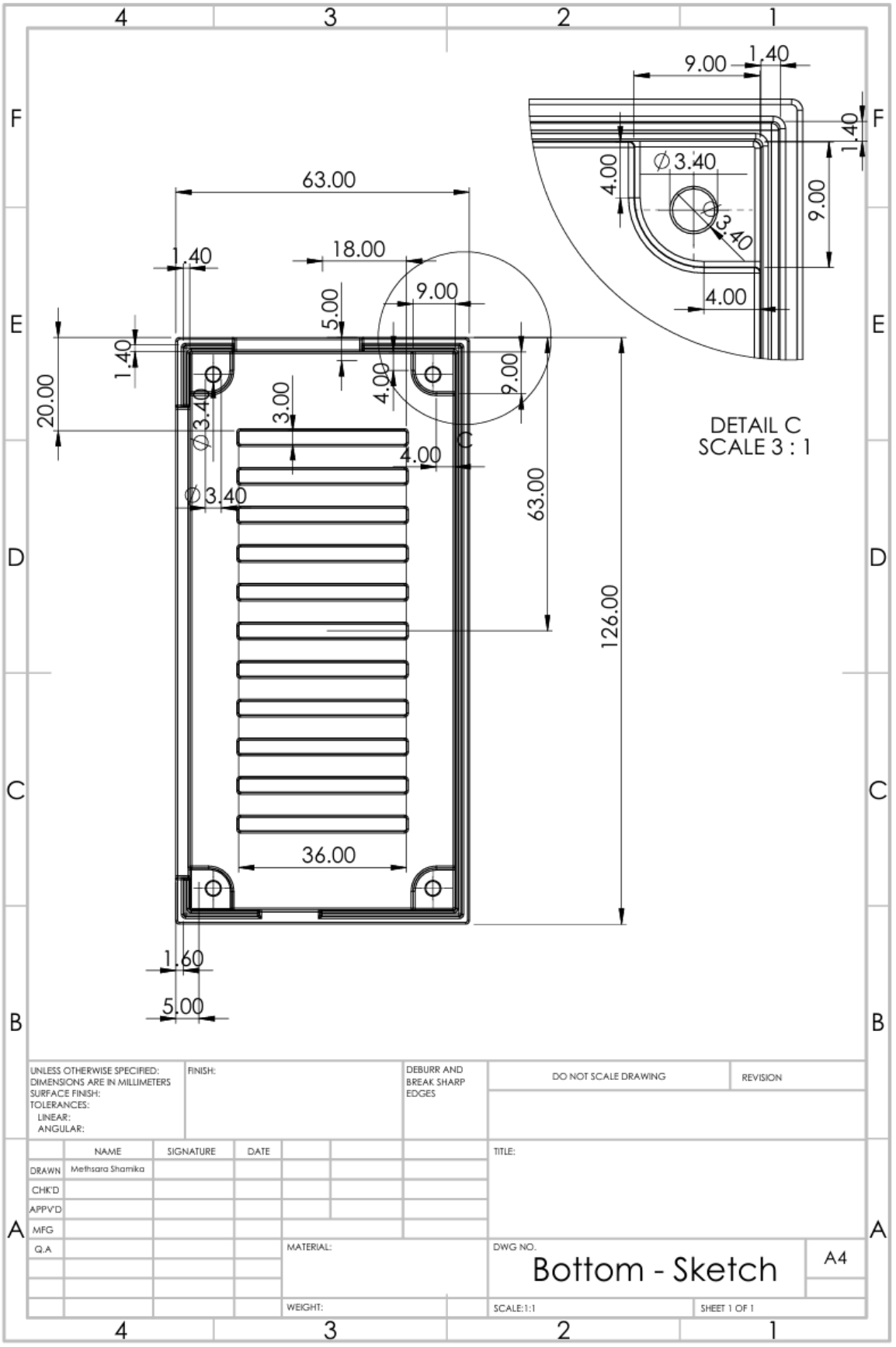

SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Bottom.SLDPRT *
Search Community Forum
Sketch
Smart Dimension
Trim Entities
Convert Entities
Offset Entities
Offset On Surface
Mirror Entities
Linear Sketch Pattern
Move Entities
Display/Delete Relations
Repair Sketch
Rapid Sketch
Instant2D
Shaded Sketch Contours
Features Sketch Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Bolt holder
Sketch - Bolt holder
Vertical mid plane - Breadth
Vertical mid plane - Length
Mirror 1 - Bolt holder
Mirror 2 - Bolt holder
Top plane - Bolt holder
Ventilation holes - Bottom
Sketch - Ventilation holes - Bottom
Linear Pattern - Ventilation holes - Bottom
Vertical front plane - Length ->
Hole 1 for connectors{ ->}
Vertical left plane - Breadth
Hole 2 for connectors
Sketch - Hole 2 for connectors
Fillet1 - Inside edges
Fillet - Outside edges
Fillet - Connector holes
Fillet - Top surface
Holes for bolt{ ->}
Sketch - Holes for bolt ->
Thread1
Thread Profile1
Thread2
Thread Profile2
Thread3
Thread Profile3
Thread4
Thread Profile4
Fillet 2 - Inside edges
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

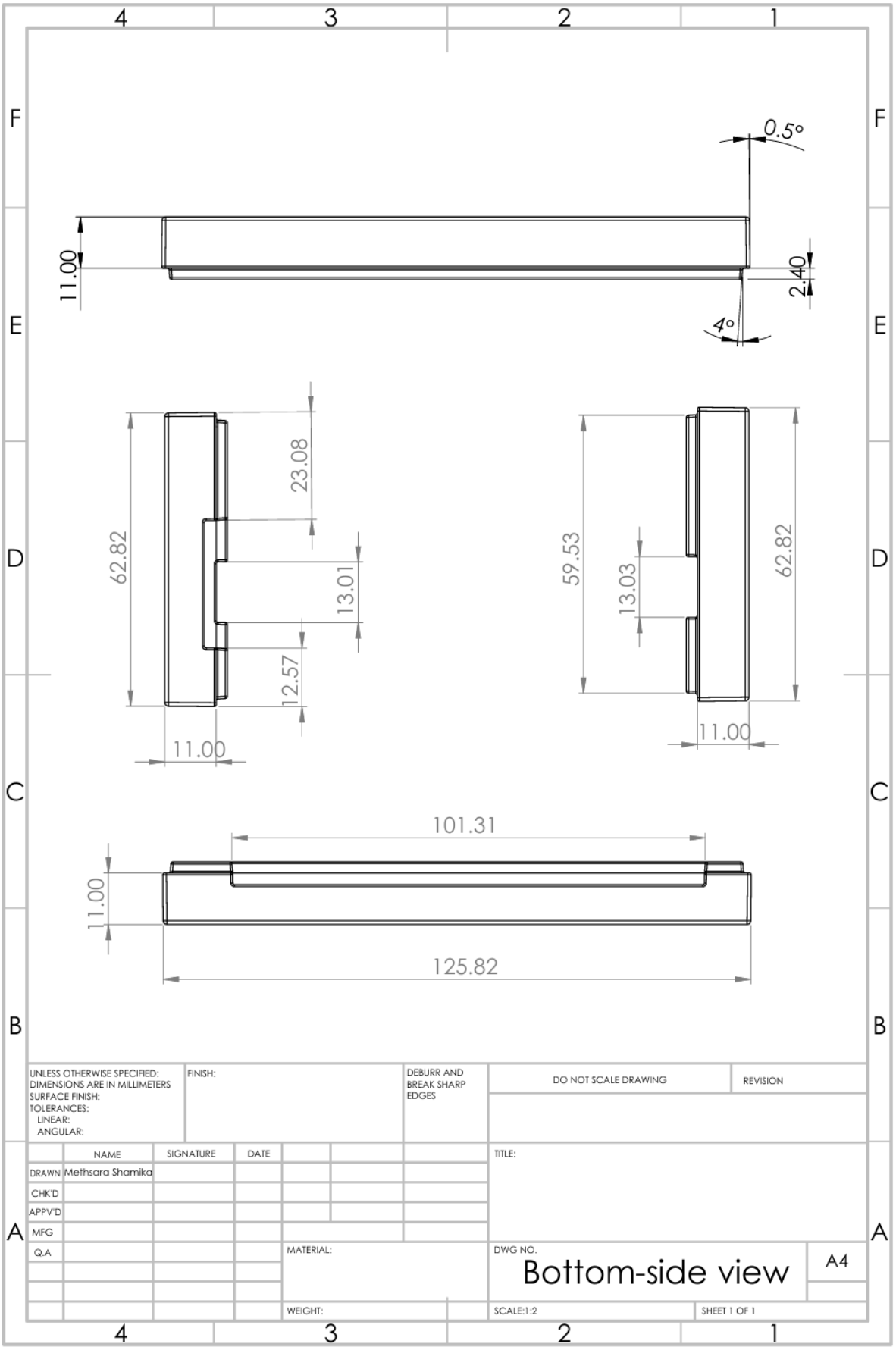
Bottom Part – Mold Design




SOLIDWORKS
File Edit View Insert Tools Simulation Window Help
Bottom.SLDPRT
Search Community Forum
Design Study
Interference Detection
Measure
Markup
Mass Properties
Section Properties
Sensor
Performance Evaluation
Check
Geometry Analysis
Import Diagnostics
Body Compare
Deviation Analysis
Zebra Stripes
Curvature
Draft Analysis
Undercut Analysis
Parting Line Analysis
Symmetry Check
Thickness Analysis
Compare Documents
Check Active Document
3DEXPERIENCE Simulation Connector
SimulationXpress Analysis Wizard
Features Sketch Mold Tools Markup Evaluate MBD Dimensions SOLIDWORKS Add-Ins Simulation Analysis Preparation SOLIDWORKS CAM SOLIDWORKS CAM TBM
Bottom (Default<<Default>_Display State 1>)->?
History
Sensors
Annotations
Surface Bodies(3)
Solid Bodies(3)
Markups
Material <not specified>
Front Plane
Top Plane
Right Plane
Origin
Base box - Bottom
Sketch - Base box - Bottom
Shell1 - Bottom
Bottom horizontal plane
Bolt holder
Sketch - Bolt holder
Vertical mid plane - Breadth
Vertical mid plane - Length
Mirror 1 - Bolt holder
Mirror 2 - Bolt holder
Draft- inside edges
Top plane - Bolt holder
Ventilation holes - Bottom
Sketch - Ventilation holes - Bottom
Linear Pattern - Ventilation holes - Bottom
Vertical front plane - Length->?
Hole 1 for connectors{ ->?}
Sketch - Hole 1 for connectors->?
Vertical left plane - Breadth
Model 3D Views Motion Study 1
SOLIDWORKS Education Edition - Instructional Use Only
Editing Part
MMGS

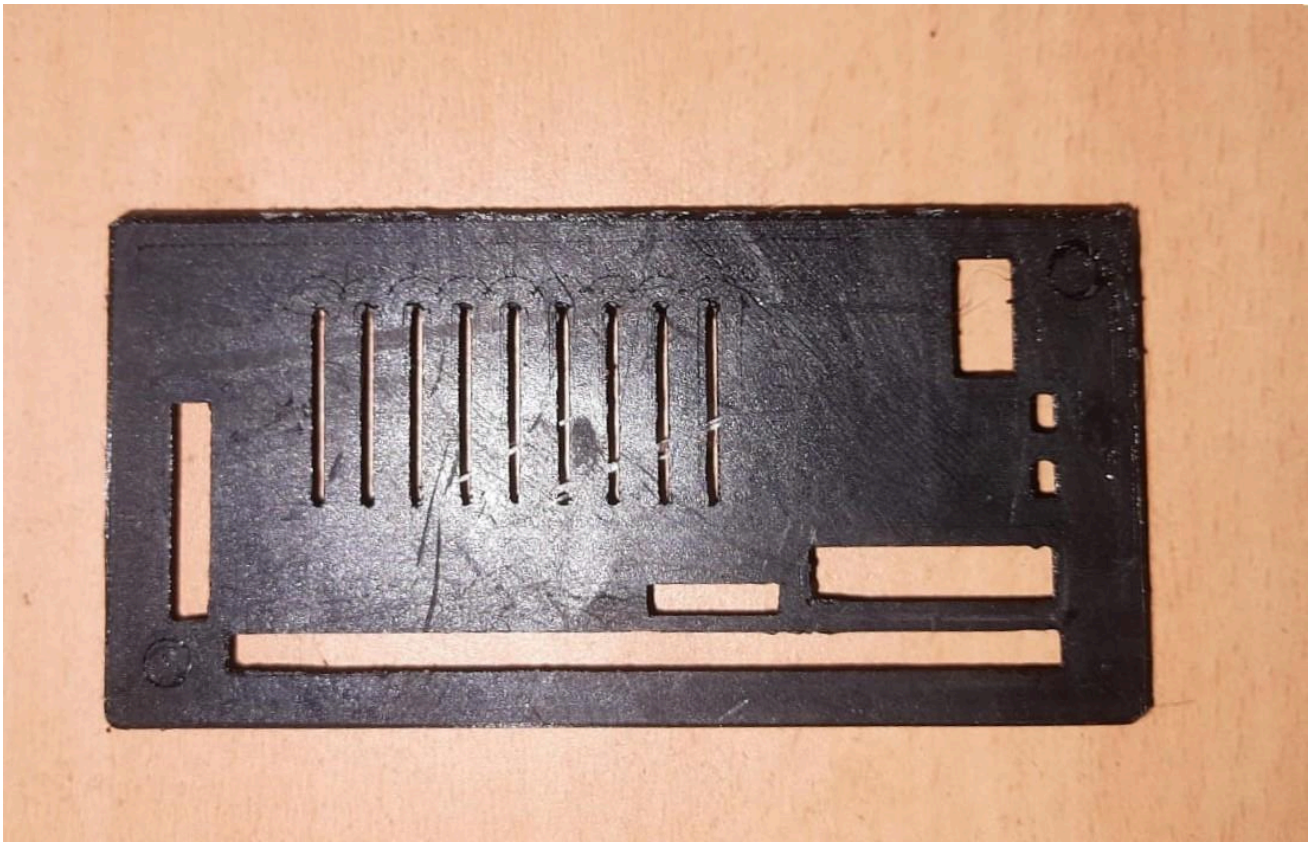
Detailed Design Drawings – Bottom Part

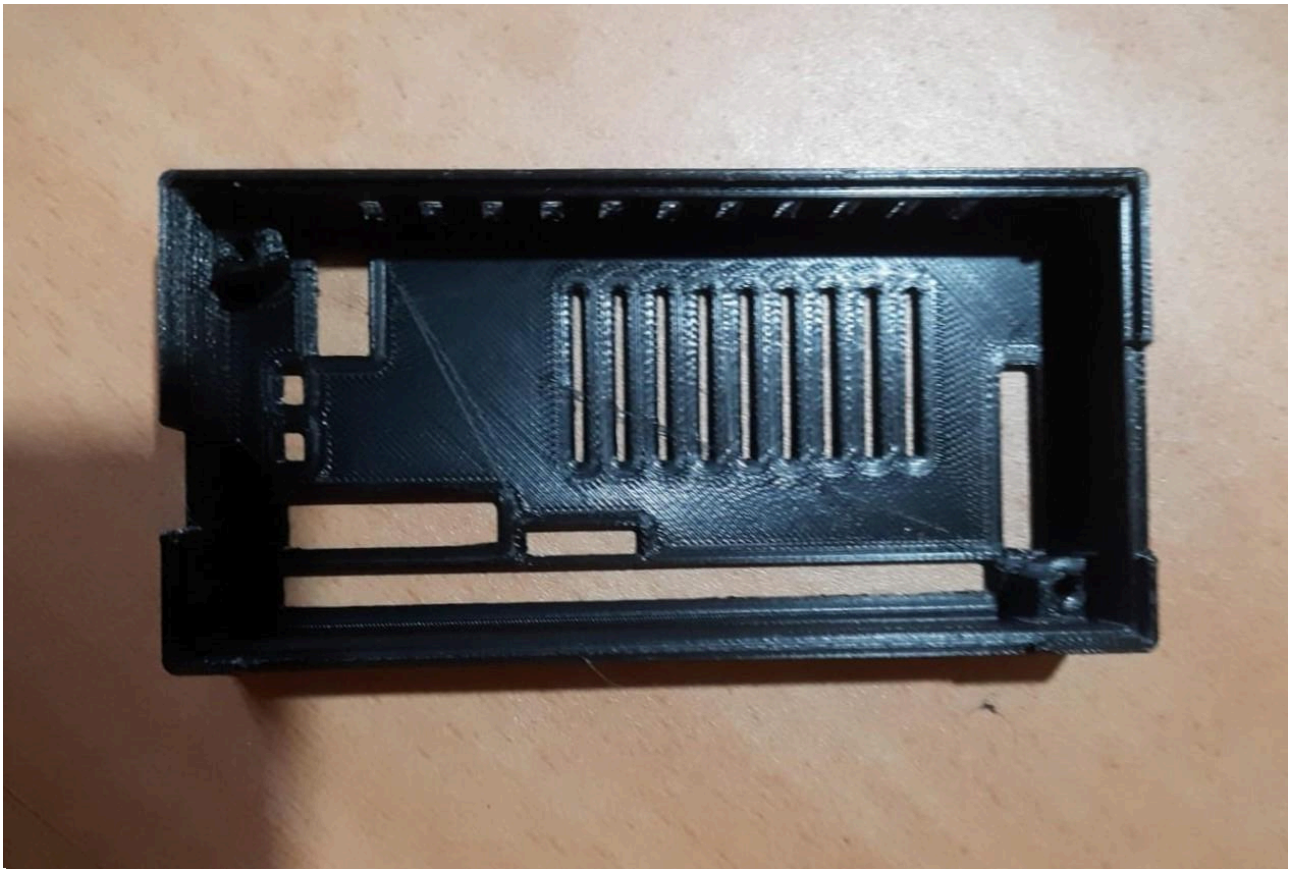




UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE				TITLE:
DRAWN	Methsara Shamika						
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:	DWG NO.		A4
					Bottom-side view		
				WEIGHT:	SCALE:1:2	SHEET 1 OF 1	

Photographs of the physically built enclosure





Software Design

Introduction

This outlines the design and implementation of a closed-loop stepper motor driver using the STM32F103CBT6 microcontroller unit (MCU) and programmed via STM32CubeIDE. The purpose of this driver is to achieve precise control over the position and speed of a stepper motor by employing feedback mechanisms.

Development Environment

- **IDE:** STM32CubeIDE
- **HAL Library:** STM32CubeMX

Firmware Architecture

Initialization

The initialization process involves setting up the system clock, peripherals, GPIO pins, timers, encoder interface, and communication interfaces.

1. **System Clock Configuration:** Configures the MCU clock for optimal performance.
2. **Peripheral Initialization:** Initializes all necessary peripherals using HAL functions.
3. **GPIO Configuration:** Sets up the GPIO pins for motor control and encoder inputs.
4. **Timer Configuration:** Configures timers for PWM signal generation and encoder reading.
5. **Encoder Interface Configuration:** Initializes the encoder interface to read motor position.
6. **UART/I2C/SPI Configuration:** Configures communication interfaces for debugging or external communication if needed.

Main Loop

The main loop runs continuously to control the stepper motor based on the feedback from the encoder.

1. **Read Encoder Feedback:** Continuously reads the encoder to get the current position of the motor.

2. **Compute Error:** Calculates the error as the difference between the desired position and the current position.

3. **Apply PID Control Algorithm:** Uses the PID algorithm to compute the control signal based on the error.
4. **Generate PWM Signals for Motor Control:** Converts the control signal to PWM signals to adjust the motor speed and direction.
5. **Update Motor Driver:** Sends the PWM signals to the motor driver to control the stepper motor.

Coding part for the microcontroller

All codes are included in this link :
https://github.com/RavijaDul/closed_loop_stepper_motor_driver

main.h file

```
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes
------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"
/* Private includes
----------------------------------------------------------*/
/* Exported functions prototypes
---------------------------------------------*/
void Error_Handler(void);
/* Private defines
-----------------------------------------------------------*/
#define D1_Pin GPIO_PIN_13
#define D1_GPIO_Port GPIOC
#define D2_Pin GPIO_PIN_14
#define D2_GPIO_Port GPIOC
#define OSC_IN_Pin GPIO_PIN_0
#define OSC_IN_GPIO_Port GPIOD
#define OSC_OUT_Pin GPIO_PIN_1
#define OSC_OUT_GPIO_Port GPIOD
#define Temp_Pin GPIO_PIN_1
#define Temp_GPIO_Port GPIOA
#define IN_BM_Pin GPIO_PIN_2
#define IN_BM_GPIO_Port GPIOA
#define IN_BP_Pin GPIO_PIN_3
#define IN_BP_GPIO_Port GPIOA
#define IN_AM_Pin GPIO_PIN_4
#define IN_AM_GPIO_Port GPIOA
#define IN_AP_Pin GPIO_PIN_5
#define IN_AP_GPIO_Port GPIOA
```

```
#define DIR_Pin GPIO_PIN_6
#define DIR_GPIO_Port GPIOA
#define BUTTON_1_Pin GPIO_PIN_2
#define BUTTON_1_GPIO_Port GPIOB
#define IN_APB10_Pin GPIO_PIN_10
#define IN_APB10_GPIO_Port GPIOB
#define IN_AMB11_Pin GPIO_PIN_11
#define IN_AMB11_GPIO_Port GPIOB
#define BUTTON_2_Pin GPIO_PIN_12
#define BUTTON_2_GPIO_Port GPIOB
#define ID_0_Pin GPIO_PIN_8
#define ID_0_GPIO_Port GPIOA
#define ID_1_Pin GPIO_PIN_9
#define ID_1_GPIO_Port GPIOA
#define ID_2_Pin GPIO_PIN_10
#define ID_2_GPIO_Port GPIOA
#define SW_DIO_Pin GPIO_PIN_13
#define SW_DIO_GPIO_Port GPIOA
#define SW_CLK_Pin GPIO_PIN_14
#define SW_CLK_GPIO_Port GPIOA
#define Enable_Pin GPIO_PIN_15
#define Enable_GPIO_Port GPIOA
#define CAN_RX_Pin GPIO_PIN_8
#define CAN_RX_GPIO_Port GPIOB
#define CAN_TX_Pin GPIO_PIN_9
#define CAN_TX_GPIO_Port GPIOB
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
```

main.c file

```
#include "main.h"
#include <math.h>
#include "stm32f1xx_hal.h"
#define MAX_INTEGRAL 1000.0
#define MAX_ANGLE 360.0

/* Private variables
---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi1;

volatile float encoder_value = 0.0;  // Encoder value variable
float setpoint = 1000.0;  // Desired encoder value (adjust as needed)
float kp = 0.1;  // Proportional gain
float ki = 0.01;  // Integral gain
float integral = 0.0;  // Integral term
float error = 0.0;  // Error term
float control_signal = 0.0;  // Control signal
int direction;
```

```
int desired_position;
int enable;
int full_rotations = 0;
float remaining_degrees = 0.0;

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
void PI_Controller(void);
void readEncoderData(void);
void readInputs(void);
void StepMotor(uint8_t);
void readDesiredPositionSPI(void);

const uint8_t step_sequence[4][4] = {
   {1, 0, 1, 0}, // Step 1: A+ B+
   {0, 1, 1, 0}, // Step 2: A- B+
   {0, 1, 0, 1}, // Step 3: A- B-
   {1, 0, 0, 1}  // Step 4: A+ B-
};


int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_I2C1_Init();
 MX_SPI1_Init();
 uint8_t step = 0;
 while (1)
 {
          if (enable)
          {
            readInputs();
            readDesiredPositionSPI();  // Read desired position via SPI
            readEncoderData();  // Read encoder data
            PI_Controller();  // Compute PI control signal
            if (error>0)
            {
              StepMotor(step);
              HAL_Delay((int)(1000 / fabs(control_signal))); // Delay
inversely proportional to control signal
              step = (step + 1) % 4; // Increment step for forward
direction
            }
            else if (error<0)
            {
              StepMotor(step);
              HAL_Delay((int)(1000 / fabs(control_signal))); // Delay
inversely proportional to control signal
              step = (step + 3) % 4; // Decrement step for reverse
```

```
direction
            }
          }
 }
}
void SetHBridge1Pins(uint8_t in1, uint8_t in2) {
   HAL_GPIO_WritePin(IN_AM_GPIO_Port, IN_AP_Pin, in1);
   HAL_GPIO_WritePin(IN_AM_GPIO_Port, IN_AM_Pin, in2);
}
// Function to set GPIO pins for H-Bridge 2
void SetHBridge2Pins(uint8_t in1, uint8_t in2) {
   HAL_GPIO_WritePin(IN_BP_GPIO_Port, IN_BP_Pin, in1);
   HAL_GPIO_WritePin(IN_BP_GPIO_Port, IN_BM_Pin, in2);
}
// Function to step the motor
void StepMotor(uint8_t step) {
   // Set H-Bridge 1 pins
   SetHBridge1Pins(step_sequence[step][0], step_sequence[step][1]);
   // Set H-Bridge 2 pins
   SetHBridge2Pins(step_sequence[step][2], step_sequence[step][3]);
}
void PI_Controller(void)
{
            //Normalize the encoder value within 0 to 360 degrees
          float normalized_encoder_value = fmod(encoder_value, MAX_ANGLE);
          if (normalized_encoder_value < 0) normalized_encoder_value +=
MAX_ANGLE;
            // Calculate the full rotations for the encoder
          int encoder_rotations = encoder_value / MAX_ANGLE;
            // Normalize the desired position within 0 to 360 degrees
          float normalized_position = fmod(desired_position, MAX_ANGLE);
          if (normalized_position < 0) normalized_position += MAX_ANGLE;
            // Calculate the full rotations for the desired position
          int desired_rotations = desired_position / MAX_ANGLE;
            // Calculate the total error in terms of rotations and
remaining degrees
          int rotation_error = desired_rotations - encoder_rotations;
          float degree_error = normalized_position -
normalized_encoder_value;
            // Combine the rotation and degree errors
          error = rotation_error * MAX_ANGLE + degree_error;
          error = desired_position - encoder_value;  // Calculate error
          if (direction == GPIO_PIN_RESET)  // Check if direction pin is
reset (logic low)
            {
            error = -error;  // Invert error for reverse direction
            }
          integral += error;  // Update integral term
          if (integral > MAX_INTEGRAL) integral = MAX_INTEGRAL;  // Limit
integral term
          if (integral < -MAX_INTEGRAL) integral = -MAX_INTEGRAL;  //
Limit integral term
          control_signal = kp * error + ki * integral;  // Calculate
control signal
}
```

```
void readEncoderData(void)
{
 // Placeholder - Implement based on hardware
 uint8_t data[2];  // Data buffer
 if (HAL_I2C_Master_Receive(&hi2c1, 0x36 << 1, data, 2, HAL_MAX_DELAY) ==
HAL_OK)
 {
   encoder_value = ((data[0] << 8) | data[1]) / 256.0 * 360.0;  // Convert
to degrees
 }
 else
 {
   Error_Handler();  // Error handler
 }
}
void readInputs()
{
   direction = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6); // PA6 for DIR
   enable = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_15);  // PA15 for Enable
}
void readDesiredPositionSPI(void)
{
   uint8_t spi_rx_buffer[2];
   if (HAL_SPI_Receive(&hspi1, spi_rx_buffer, 2, HAL_MAX_DELAY) == HAL_OK)
   {
       desired_position = (spi_rx_buffer[0] << 8) | spi_rx_buffer[1];
   }
   else
   {
       Error_Handler();
   }
}
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
   Error_Handler();
 }
 /** Initializes the CPU, AHB and APB buses clocks
 */
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
   Error_Handler();
 }
}
static void MX_I2C1_Init(void)
{
 hi2c1.Instance = I2C1;
 hi2c1.Init.ClockSpeed = 100000;
 hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
 hi2c1.Init.OwnAddress1 = 0;
 hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
 hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
 hi2c1.Init.OwnAddress2 = 0;
 hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
 hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
 if (HAL_I2C_Init(&hi2c1) != HAL_OK)
 {
   Error_Handler();
 }
}
static void MX_SPI1_Init(void)
{
 hspi1.Instance = SPI1;
 hspi1.Init.Mode = SPI_MODE_MASTER;
 hspi1.Init.Direction = SPI_DIRECTION_2LINES_RXONLY;
 hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
 hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
 hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
 hspi1.Init.NSS = SPI_NSS_SOFT;
 hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
 hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
 hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
 hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
 hspi1.Init.CRCPolynomial = 10;
 if (HAL_SPI_Init(&hspi1) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOC_CLK_ENABLE();
 __HAL_RCC_GPIOD_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOC, D1_Pin|D2_Pin, GPIO_PIN_RESET);
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOA, IN_BM_Pin|IN_BP_Pin|IN_AM_Pin|IN_AP_Pin,
GPIO_PIN_RESET);
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOB, IN_APB10_Pin|IN_AMB11_Pin|CAN_TX_Pin,
GPIO_PIN_RESET);
 /*Configure GPIO pins : D1_Pin D2_Pin */
 GPIO_InitStruct.Pin = D1_Pin|D2_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
 /*Configure GPIO pin : Temp_Pin */
 GPIO_InitStruct.Pin = Temp_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
 HAL_GPIO_Init(Temp_GPIO_Port, &GPIO_InitStruct);
 /*Configure GPIO pins : IN_BM_Pin IN_BP_Pin IN_AM_Pin IN_AP_Pin */
 GPIO_InitStruct.Pin = IN_BM_Pin|IN_BP_Pin|IN_AM_Pin|IN_AP_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pins : DIR_Pin ID_0_Pin ID_1_Pin ID_2_Pin
                          SW_DIO_Pin SW_CLK_Pin Enable_Pin */
 GPIO_InitStruct.Pin = DIR_Pin|ID_0_Pin|ID_1_Pin|ID_2_Pin
                         |SW_DIO_Pin|SW_CLK_Pin|Enable_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pins : BUTTON_1_Pin BUTTON_2_Pin PB5 CAN_RX_Pin */
 GPIO_InitStruct.Pin = BUTTON_1_Pin|BUTTON_2_Pin|GPIO_PIN_5|CAN_RX_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
 /*Configure GPIO pins : IN_APB10_Pin IN_AMB11_Pin CAN_TX_Pin */
 GPIO_InitStruct.Pin = IN_APB10_Pin|IN_AMB11_Pin|CAN_TX_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
void Error_Handler(void)
{
 /* USER CODE BEGIN Error_Handler_Debug */
 /* User can add his own implementation to report the HAL error return
state */
 __disable_irq();
 while (1)
 {
 }
 /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
```

```
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
 /* USER CODE BEGIN 6 */
 /* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
 /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Conclusion

Summary of the Project

In this project, we hope to successfully design and implement a closed-loop stepper motor driver with a feedback mechanism. The primary objective was to create a motor driver that can accurately control the position of the stepper motor and correct any positional errors through real-time feedback. The system comprises a robust power circuit, an efficient motor control unit, and a reliable communication interface. Key components used include the STM microcontroller, magnetic encoder, PWM chopper type DC brushed motor driver, and CAN interfaces.Firmware for the CAN was not included here.

The project involved meticulous design considerations, such as selecting appropriate trace widths for current capacity, ensuring stable power supply through efficient voltage regulators, and implementing a solid grounding strategy to reduce noise. The feedback mechanism, central to our design, utilized a magnetic encoder to provide real-time positional feedback, enabling precise and stable motor control.

Key Takeaways and Learning Outcomes

1. **Integrated Design Approach:**
   - Learned the importance of integrating various subsystems, such as power, control, and communication, to achieve a cohesive and functional design.
2. **Component Selection and Justification:**
   - Gained insights into selecting appropriate components based on performance requirements and design constraints. Each component was chosen for its specific role and contribution to the overall system efficiency and reliability.
   - All the components except connectors were ordered from mouser electronics and
3. **Feedback Mechanism Implementation:**
   - Developed an understanding of how to implement and utilize feedback mechanisms for error correction and precise control. The use of a magnetic encoder and real-time signal processing by the microcontroller was crucial in achieving accurate motor positioning.
4. **Power Management:**
   - Learned about designing efficient power circuits to provide stable voltage levels, which are essential for the reliable operation of electronic components. The use of switching and LDO regulators was key to maintaining a clean power supply.
5. **Signal Integrity and Grounding:**
   - Recognized the importance of proper grounding techniques and trace width considerations to ensure signal integrity and minimize electromagnetic interference.
6. **CAN Communication:**
   - Acquired knowledge on implementing CAN communication interfaces for connecting and controlling multiple motor drivers, enabling synchronized and coordinated operation in larger systems.
7. **Practical Challenges and Solutions:**
   - Encountered and addressed various practical challenges, such as managing current spikes and ensuring accurate feedback. This involved implementing coupling capacitors and optimizing feedback loops.
8. **Collaboration and Project Management:**
   - Improved skills in collaboration and project management, working as a team to design, test, and refine the stepper motor driver system. Effective communication and task delegation were critical to the project's success.

In conclusion, this project provided valuable hands-on experience in designing a complex electronic system with precise control capabilities. The knowledge and skills gained from this project will be beneficial for future endeavors in electronics and embedded systems design.

Appendix

Daily Log Entries

Feb 5-11:

Activities:

- Through the second round of proposal submission, under guidance of the professor we began the project “Closed loop stepper motor driver”
- As we start we were given the requirements that our final product must reach such as the presion in micro stepping to 6000 , size of the product and we were taught to arrange a meeting with the professor for theoretical guidance .

Outcomes:

- Gained insights into the project and started to research and background learning process on the project .

Feb 12-18:

Activities:

- Had a meeting with the professor where we gained a brief understanding about the theoretical background of the project. Here we learned about space vector PWM which we can use with microstepping, then PID control feedback,Clarke and Park transformations… etc.
- Conducted initial research on the project, focusing on understanding existing solutions in the market related to stepper motor drivers.
- As in the market there was both open and closed loop stepper motor drivers, Though open loop drivers were cost effective there were certain disadvantages when consider with the functionality as follows,

1. Lack of Real-Time Feedback:

Open-loop control lacks real-time feedback from the motor. Without feedback, it cannot precisely monitor the motor’s position or detect errors.

In contrast, closed-loop systems use feedback (usually from encoders) to constantly adjust and correct the motor’s position, ensuring accuracy and reliability.

2. Stepping Errors:

In open-loop systems, stepping errors can occur if the input signal is incorrect or if the load changes unexpectedly. These errors reduce acceleration and accuracy2.

Closed-loop systems continuously adjust the motor's position based on feedback, minimizing stepping errors.

3. Continuous Current Flow

In open-loop systems, current continues to flow through the motor even when it holds a position (e.g., when the rotor is locked). This results in high power consumption and heat generation.

Closed-loop systems optimize current flow based on real-time feedback, reducing unnecessary power consumption.

4. Loss of Synchronization:

Open-loop systems can lose synchronization when the load changes unexpectedly, especially at high speeds or during sudden speed changes.

Closed-loop systems actively correct for any deviations, maintaining synchronization and accuracy.

Outcomes:

- Gained insights into the current state of stepper motor driver technologies and identified key features and functionalities.
- After identifying the importance of closed loop stepper motor drivers for more precise stepping with comparison to the open loop drivers we came to the conclusion that closed loop stepper motor drivers are much preferable but cost and Complexity is higher so that it is needed to implement the driver cost effective and much more simpler in order for practical use.

Feb 19-25:

Activities:

- Experimented with different approaches for calculations using FPGA and microcontrollers to determine the most suitable technology within budget constraints.
- As the driver should get feedback signals in order to control the driver movements we had to consider whether we use Microcontroller or a FPGA, We understood that whatever we choose should has high frequency abilities, so after considering many FPGAs and micro controllers we came to the final decision to use STM chip as it meet our requirements.

Outcomes:

- Identified that an STM32F103CBT6 microcontroller is cost-effective and meets the project's computational requirements.

Feb 26- March 3:

Activities:

- Developed multiple conceptual designs for the stepper motor driver, evaluating each for feasibility and effectiveness.Here we identified making a separate motor driver block is the most viable design among the ones we proposed for the conceptual designs.

Outcomes:

- Selected a conceptual design that optimally separates the motor from other components for better functionality and control.

March 4 - 10:

Activities:

- Researched and selected circuit designs that align with the chosen conceptual design, focusing on reliability and performance.
- After considering many approaches towards required design the final circuits designs were concluded as to require following requirements

Using CAN Bus Transceiver ,so that multiple drivers can be controlled by user simultaneously.

To use two PWM Chopper Type DC Brushed Motor Drivers to control two poles

Use an external magnetic encoder (if we use design no 2 in conceptual design we had to attach it to the pcb) that should fix behind the stepper motor , thus itis need to make an small pcb to fix behind the stepper motor.

Outcomes:

- Identified and adopted circuit designs that integrate well with the selected conceptual approach.

March 11-17:

Activities:

- Explored and finalized component selections based on performance, compatibility, and availability.
- Most of the components were selected from mouser electronics and order was placed.
- Other components such as screw connectors, jst conncters , headers are purchased from electronic shops.

Outcomes:

- Compiled a list of components necessary for the successful implementation of the stepper motor driver.

March 18-24:

Activities:

- Initiated the design phase by creating circuit diagrams using Altium and visualizing the designs in solidworks for further analysis.
- First we did the component placement without much reasoning , so for again a time we had to make it properly because we identified we had to place the component more conveniently. Placed the coupling capacitors,crystal oscillator as near as the ports of MCU and tried to keep power action separated , to design with sufficient trace widths.

Outcomes:

- Progressed with the circuit design phase and established visual models for evaluation and feedback.

March 25-31:

Activities:

- Reviewed the designed circuits with the lecturer and discussed progress with colleagues to ensure alignment with project goals. This date is not sure but the professor looked through our progress and gave us instructions.

Outcomes:

- Received feedback on the circuit designs, fostering collaboration and refining design elements.

April 1-10 :

Activities:

- Finalized component selections and initiated procurement by placing orders through suppliers like Mouser.
- Those components supposed to by from shops were also purchased.

Outcomes:

- Started the procurement process for components and sent the PCB design for manufacturing.

April 22-May 15:

Activities:

- After arriving the components (after the mid brake)Commenced soldering the PCB with the components, marking the beginning of the physical assembly phase.
- Soldering phase was done keenly.
- Most of the parts were smd components and others were through hole ones.we used a soldering station for this. Yet the testing part was not done due academic strikes.
- Meanwhile we started looking into the software cubeIDE that we are supposed to use for coding the MCU.

Outcomes:

- Started the assembly process, moving towards integration and testing stages to validate the designed stepper motor driver.
- Started software side moving towards coding and debugging.

References

1. **An Introduction to Stepper Motors - University of Arizona**:
   - URL: https://wp.optics.arizona.edu/optomech/wp-content/uploads/sites/53/2016/10/Tutorial_Xinda-Hu.pdf
2. **PLC AS A DRIVER FOR STEPPER MOTOR CONTROL - Laurean Bogdan University**:
   - URL: PLC AS A DRIVER FOR STEPPER MOTOR CONTROL (researchgate.net)
3. **Design and Realization of Stepping Motor Drive System - Springer**:
   - URL: https://link.springer.com/article/10.1007/s11277-022-09534-z
4. **Stepper motor driver, Stepper motor drive - All industrial manufacturers**:
   - URL: Stepper motor driver, Stepper motor drive - All industrial manufacturers (directindustry.com)
5. **Texas Instruments Closed Loop Stepper Design Resources**:
   - URL: https://www.ti.com/solution/closed-loop-stepper
6. **24V/36W BLDC Motor Driver Reference Design With Close-Loop Speed Control**:
   - URL: https://www.ti.com/lit/ug/tiduds5/tiduds5.pdf
7. **Closed Loop Stepper Motor Design With Encoder for Stall-Detection**:
   - URL: https://www.ti.com/lit/pdf/tiducy6
8. **Y Series Closed Loop Driver Manual**:
   - URL: https://www.omc-stepperonline.com/download/Y-series.pdf