EN2160 Electronic Design Realization
Design Documentation

Module:	EN2160
Group:	Battery Profiler
Index Numbers:	210027C, 210236P, 210253N, 210272V
Date:	2024.06.23

Contents

1 Introduction

This report covers the detailed documentation with regard to the battery profiler designed in fulfillment of the Electronic Design Realization module. This documentation covers the design process, conceptual designs, finalized designs and the progress with regard to the product.

2 Schematic Explanation

In this section, the schematics of the device are explained in detail

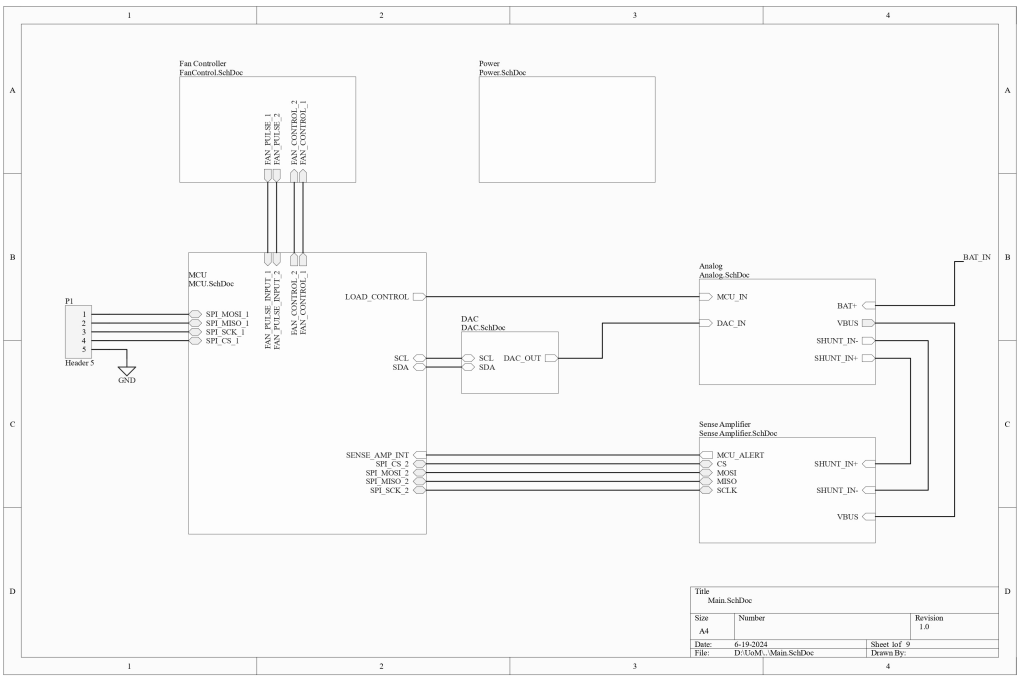

Figure 1: Block Diagram

This is the overall block diagram of the product. As shown, a microcontroller is used to control the functionality of the product. This is responsible for setting the appropriate discharge rate for the battery, controlling the fans and monitoring the battery voltage. An analog circuit is responsible for controlling the current which flows through the battery. An external digital-to-analog converter has been used to set the voltage which controls the discharge rate. The sense amplifier senses the current and provides feedback to appropriately control it. Finally, a circuit is used to control the fan speeds based on the temperature of the heatsink.

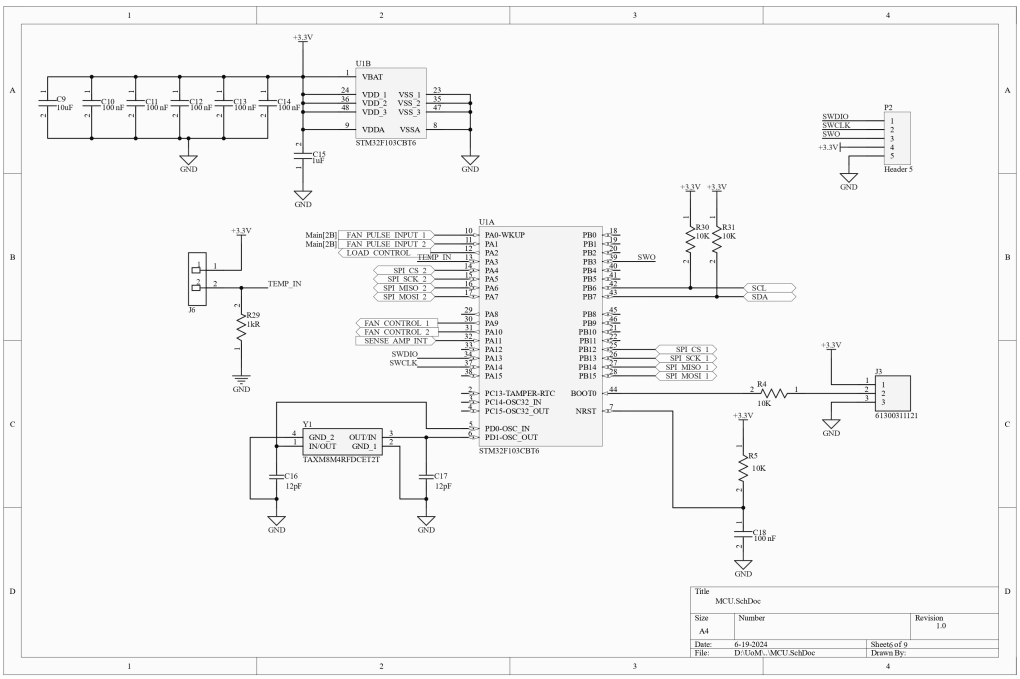

Figure 2: Microcontroller Unit

This is the microcontroller of the product. The selected micro-controller is the **STM32F103CBT6**. The data sheet for the microcontroller can be accessed here: Datasheet. The reasons for selecting the microcontroller are:

- Availability of a Floating Point Unit (FPU): Since the sense amplifier outputs 20-bit floating point numbers and since they would need to be multiplied to obtain power, a dedicated floating point unit is beneficial for speed compared to conventional 8-bit microcontrollers.
- High clock speed of 72MHz
- Availability of sufficient I/O ports :80
- Support for communication interfaces such as I2C, UART and SPI
- Low power consumption

The microcontroller operates at a supply voltage of 3.3V. As shown in the schematics, decoupling capacitors have been used for the power ports in order to maintain a stable power supply voltage to the chip. The values for these capacitances have been decided as instructed in the datasheet of the processor. An external 72MHz oscillator has been added between pins 5 and 6. The microcontroller. The two SPI interfaces are used to communicate with the display and the current sense amplifier. Additionally, pins have been allocated to sense the temperature and fan speed and control the fans. The reset pin has been pulled up to 3.3V.

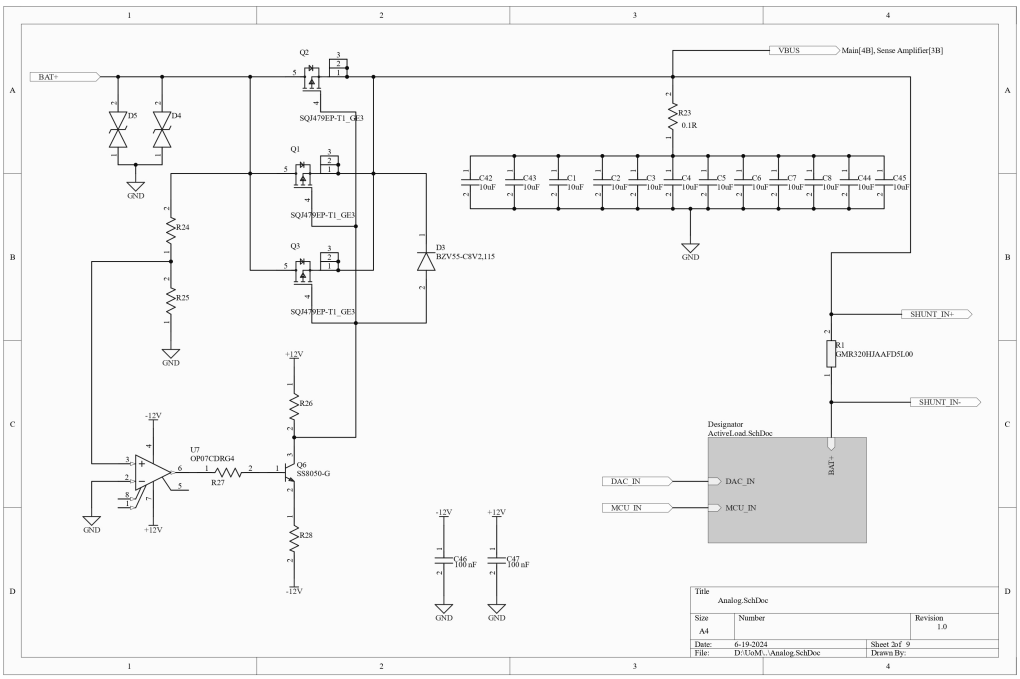

Figure 3: Analog Circuit

This is the analog circuitry of the device. The battery is connected to the circuit. Transient Voltage Suppression diodes have been used to prevent damage to the circuit due to voltage surges. Additionally, the OP-AMP U7 has been used along with the MOSFETS to provide reverse voltage protection to the circuit. An array of capacitors has been used to maintain a stable voltage.

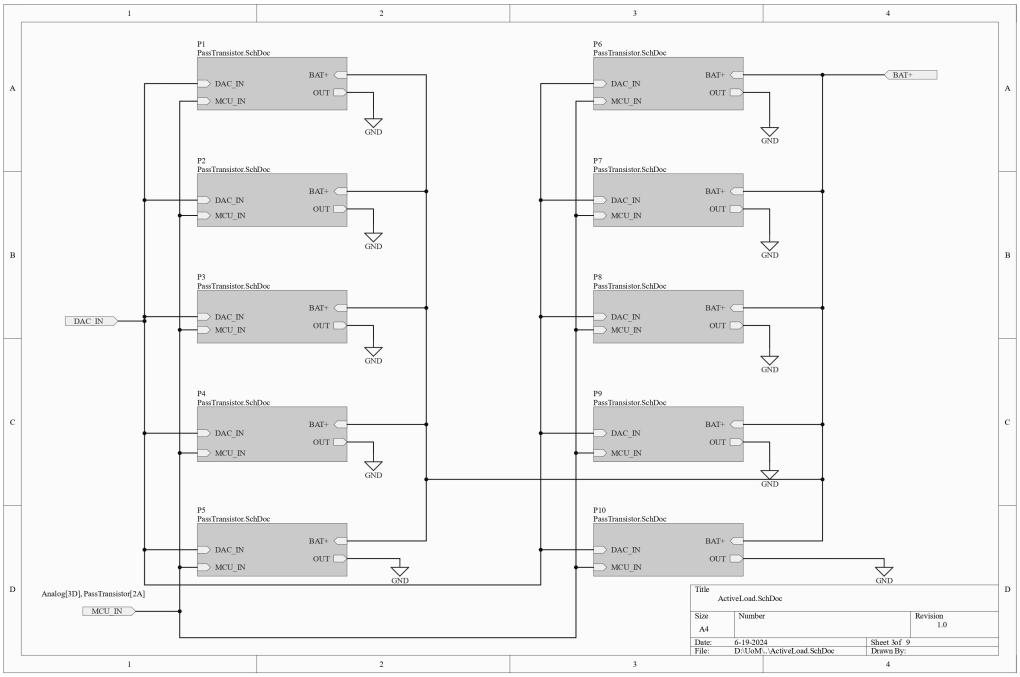

Figure 4: Active Load

When discharging a battery, that power is dissipated in these circuits. The device incorporates 10 parallel circuits for power dissipation in order to reduce the overall temperature of components and simultaneously increase the maximum possible power dissipation. The internals of this circuitry are shown in the next page.

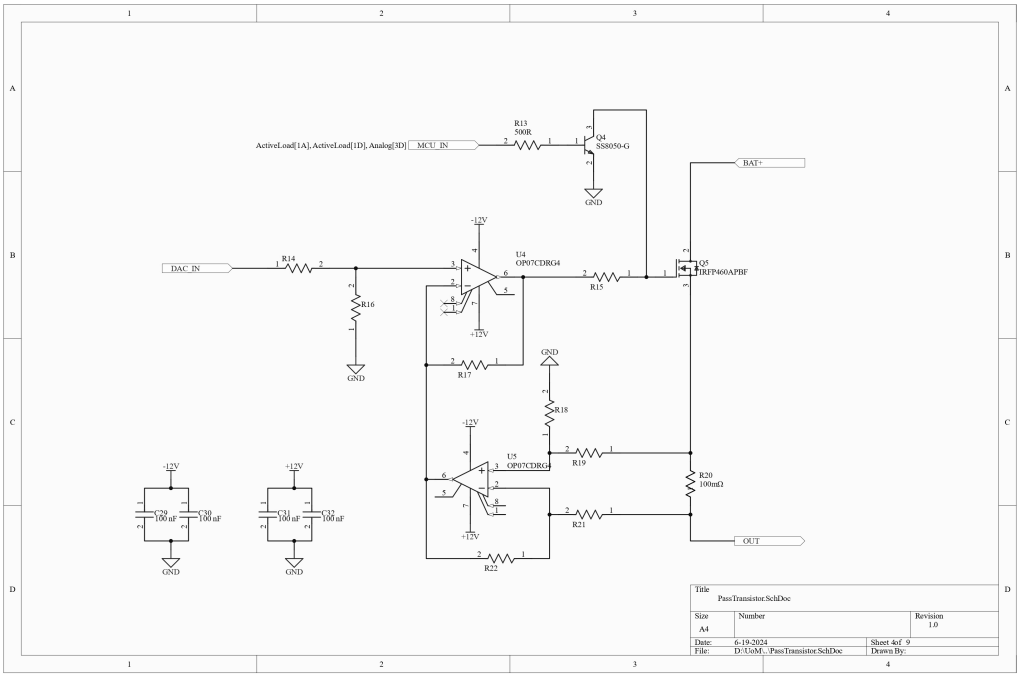

Figure 5: Pass transistor

This is the internal circuit of an active load element. The power is dissipated in the MOSFET labeled as **Q5** in the schematic. The specifications are given below:

- Maximum continuous drain current at 25°C: 20A
- Maximum continuous drain current at 100°C: 13A
- Maximum power dissipation: 280W
- Maximum drain-source voltage: 500V

Our product requires a maximum continuous current of just 3A per transistor and a maximum power dissipation of 30W per transistor. Therefore, this component was suitable for our needs. The datasheet can be found here.
This circuit operates based on the feedback principle. The required current is set by the microcontroller and that is converted to an analog voltage level by the DAC. The feedback loop of the U4 OP-AMP adjusts the gate voltage of the MOSFET such that both inputs are at an equal voltage level. By adjusting the gate voltage, the current through the MOSFET can be controlled. Resistor R20 is a current sense resistor that is used as part of the differential amplifier which has been set up with the U5 OP-AMP. U5 amplifies the difference in voltage across R20 and feeds it back into the inverting terminal of U4. The resistor is a high-power resistor capable of dissipating up to 3W of power. The maximum power dissipation required by the circuit is 1.8W and hence this is suitable to be used.

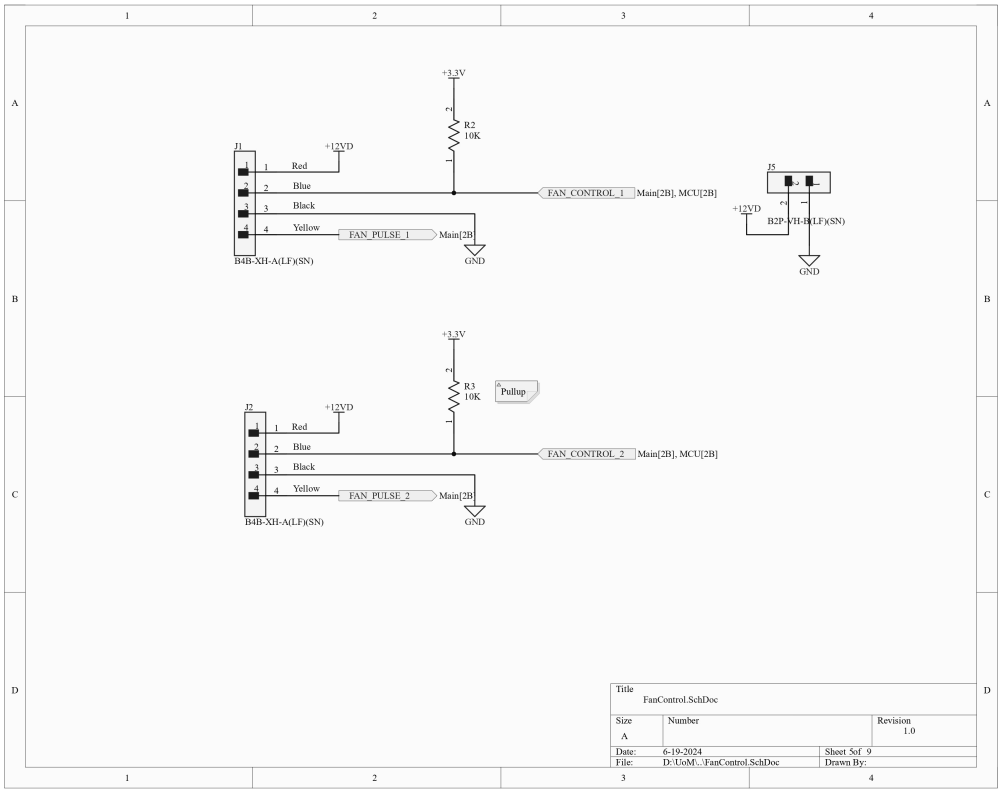

Figure 6: Fan Control

The following is the required circuit for measuring the speed of the fan and using the microcontroller to control it. The port marked Fan Pulse is connected to the fan motor's encoders. This outputs 2 pulses per rotation and can be used to measure the speed of the fan. The port marked Fan Control gives a PWM signal from the microcontroller. The maximum RPM of the fan is 6300 RPM. The duty cycle of the PWM can be adjusted by the microcontroller which in turn affects the speed of the fan.

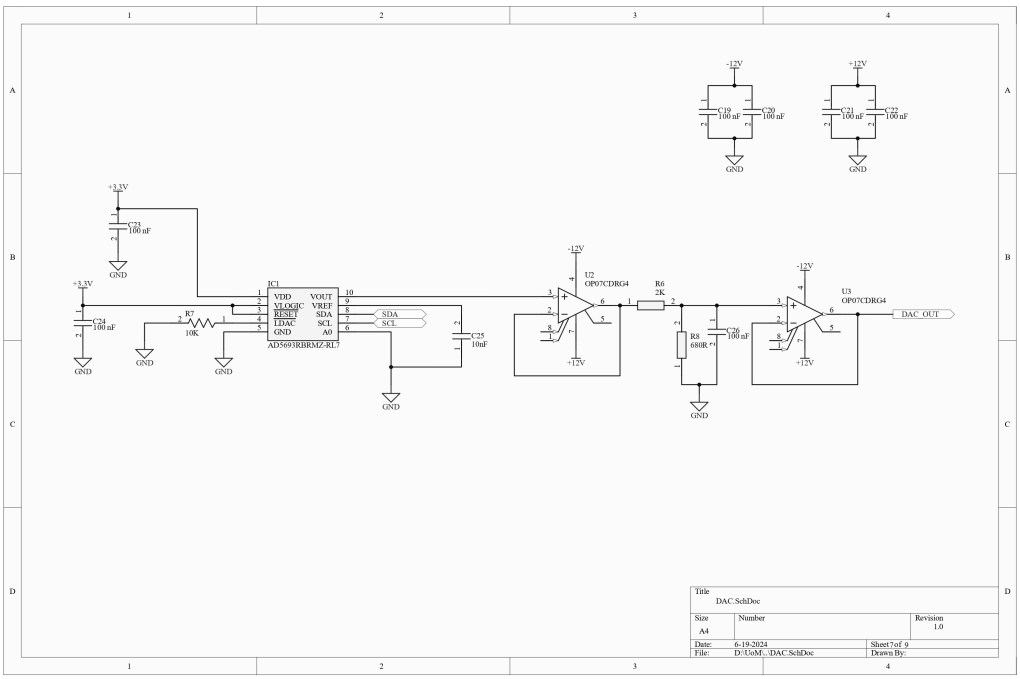

Figure 7: DAC Circuit

This is the circuit for the DAC. The DAC used is the AD5693RBRMZ from Analog devices. Thsi was chosen due to it's high relative accuracy of ± 2 LSB maximum at 16 bits and low drift, 2.5 V on-chip reference: 2 ppm/ °C. The output of the DAC is connected to a unity gain amplifier, an RC filter for stability and another unity gain amplifier.

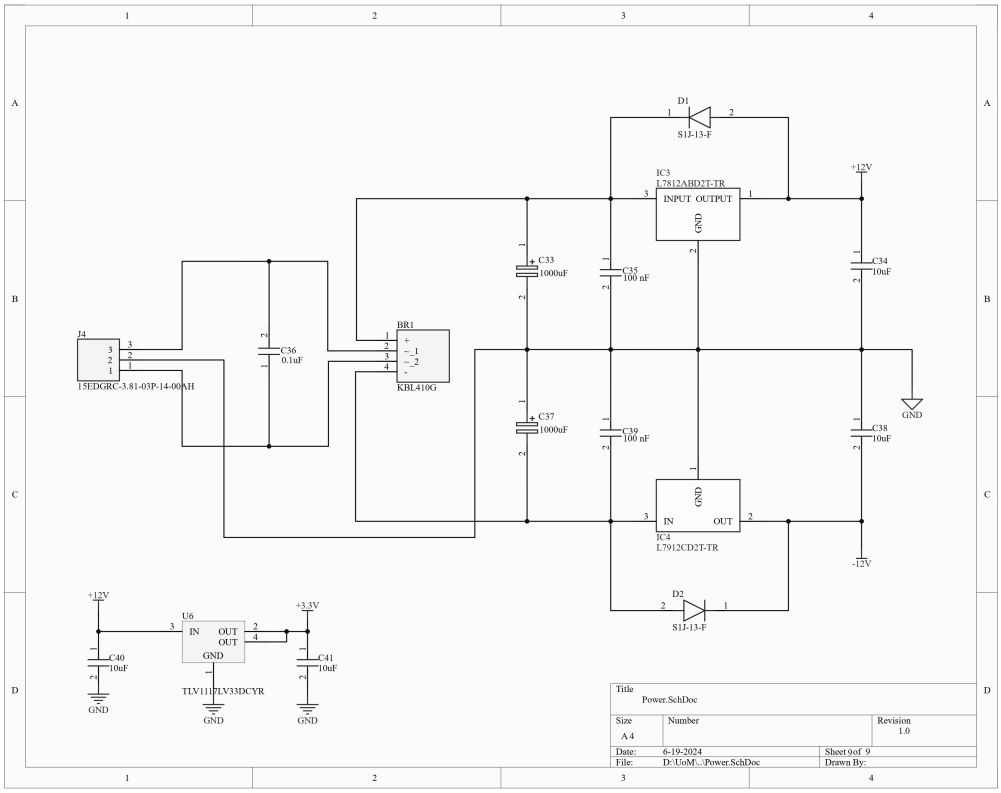

Figure 8: Power Circuit

This is the power supply circuit of the product. The input AC signal is converted to a dual supply voltage of +12V and -12V. A dual supply is used since the OP-AMP circuits have been designed with a dual supply in mind. In order to power the microcontroller and the other components, the +12V is further converted to +3.3V as well.

3 Printed Circuit Board

The following is the printed circuit board of the product. It is of 4 layers and is was manufactured by JLC PCB in China. Component placement and routing were done by us.

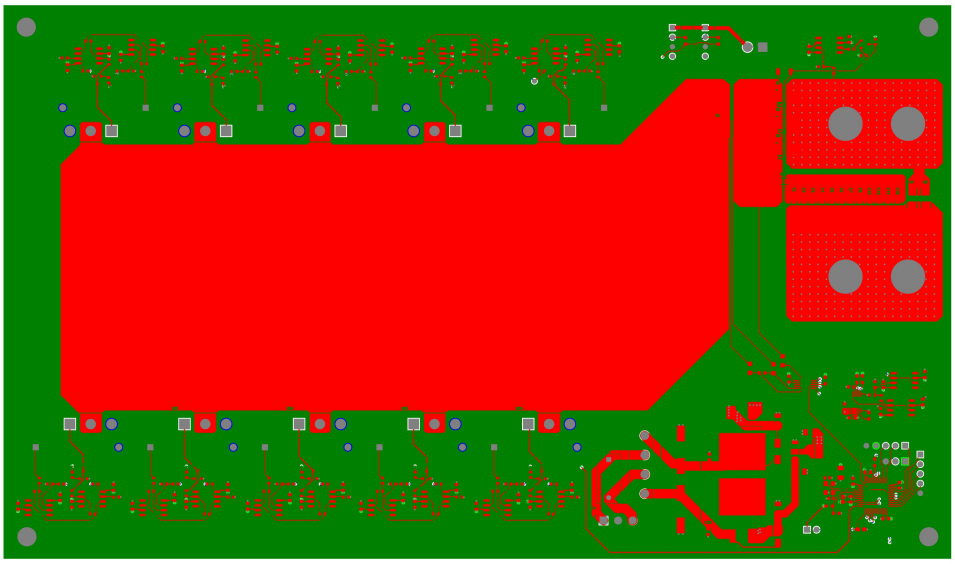

Figure 9: All Layers

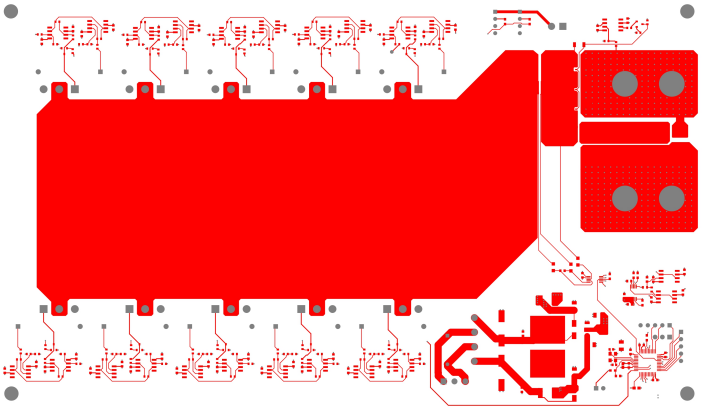

Figure 10: Top Layer

This is the top layer of the PCB. Most of the routing is done in this layer.The large area of copper unpopulated with components is the area the heatsink will be placed. The 10 parallel power dissipation circuits are placed around this area. The second layer is the ground plane.

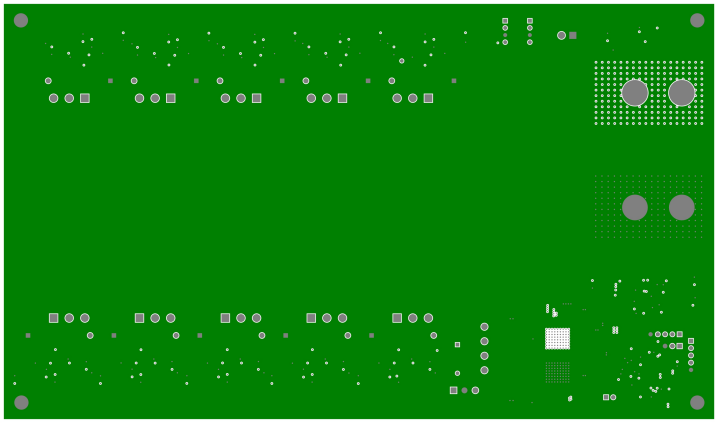

Figure 11: Ground Layer

As shown, no routing is done here and the entire layer is used as the ground.

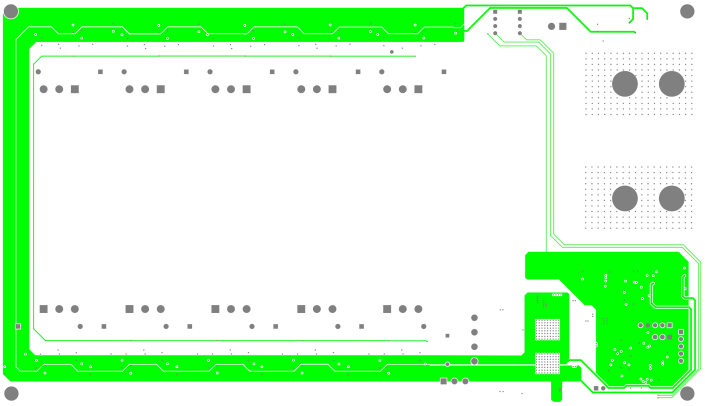

Figure 12: Power Plane

This is the power plane of the PCB. Here, routing is done for the power traces required for the OP-AMPs and the microcontroller The bottom layer is also used for routing and additionally

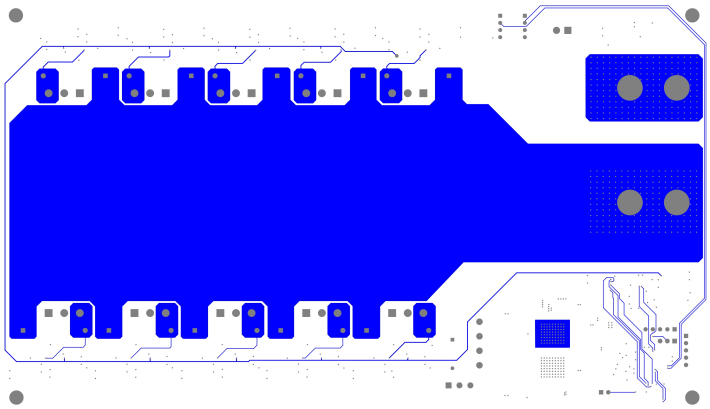

Figure 13: Bottom Layer

a large portion of this plane is also used as ground.

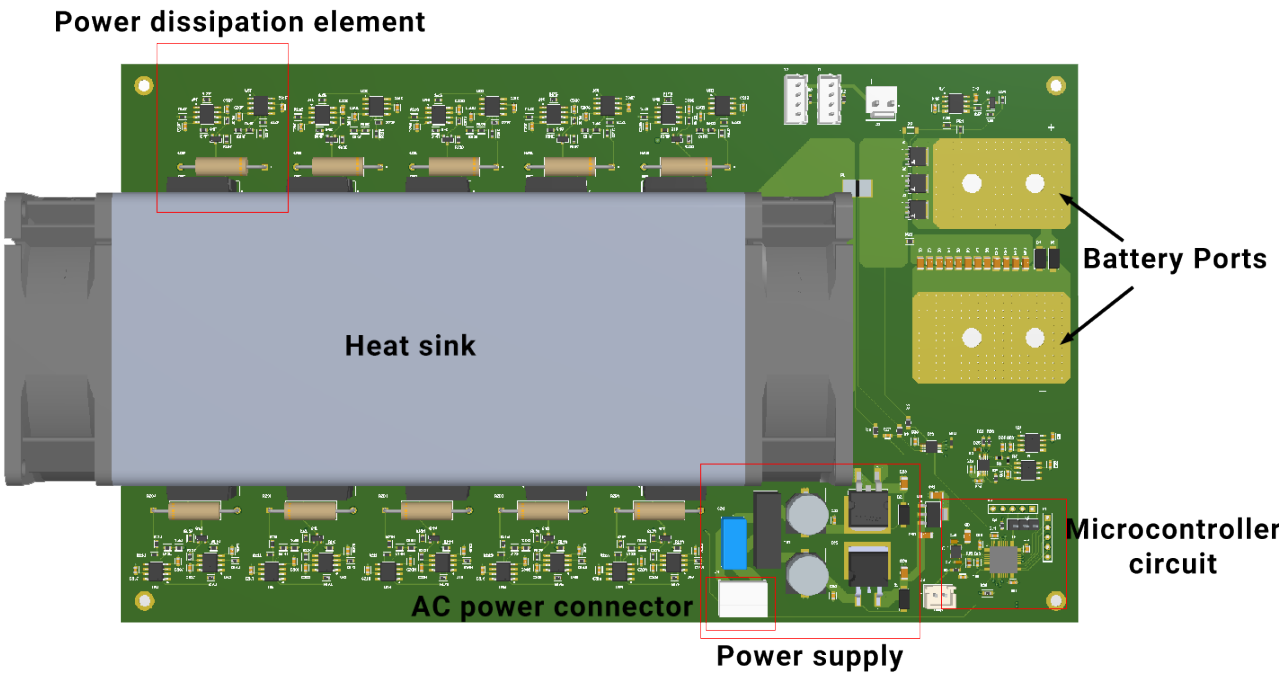

Figure 14: 3d model of PCB (top)

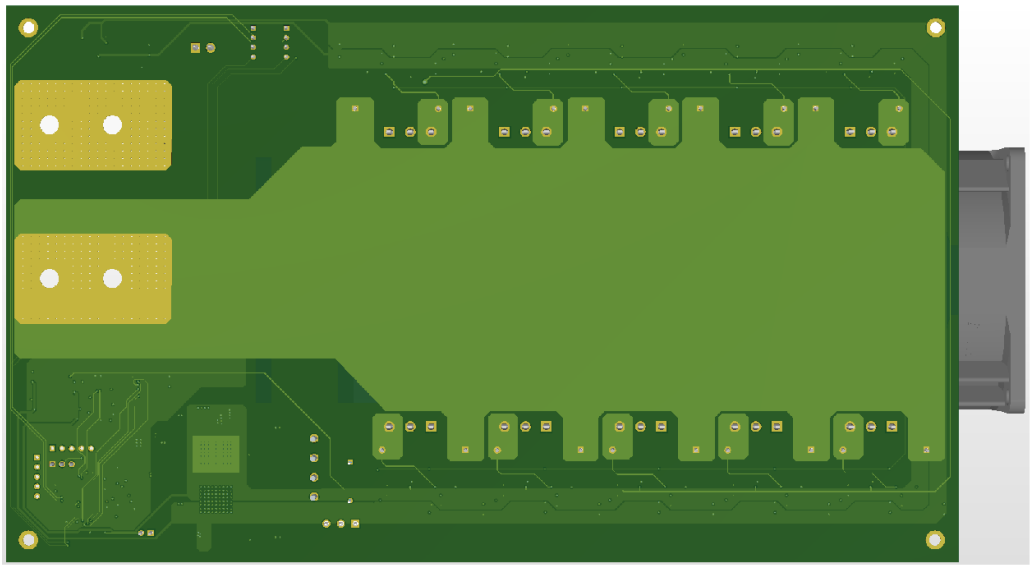

Figure 15: 3d model of PCB (bottom)

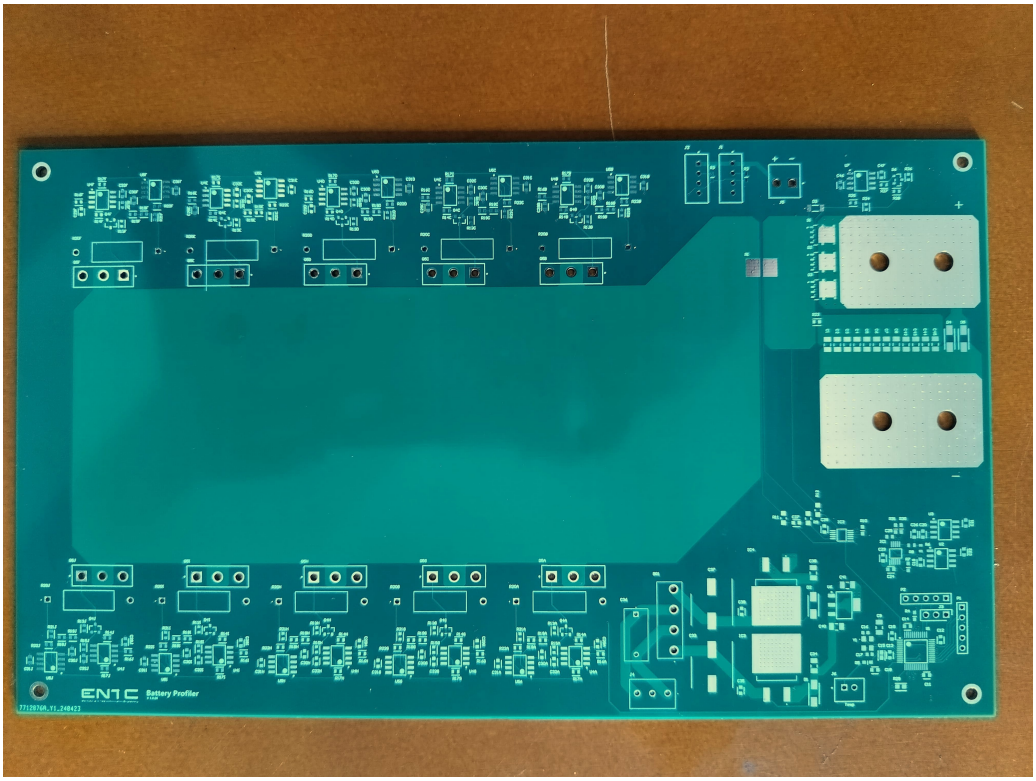

Figure 16: Photo of the unsoldered PCB

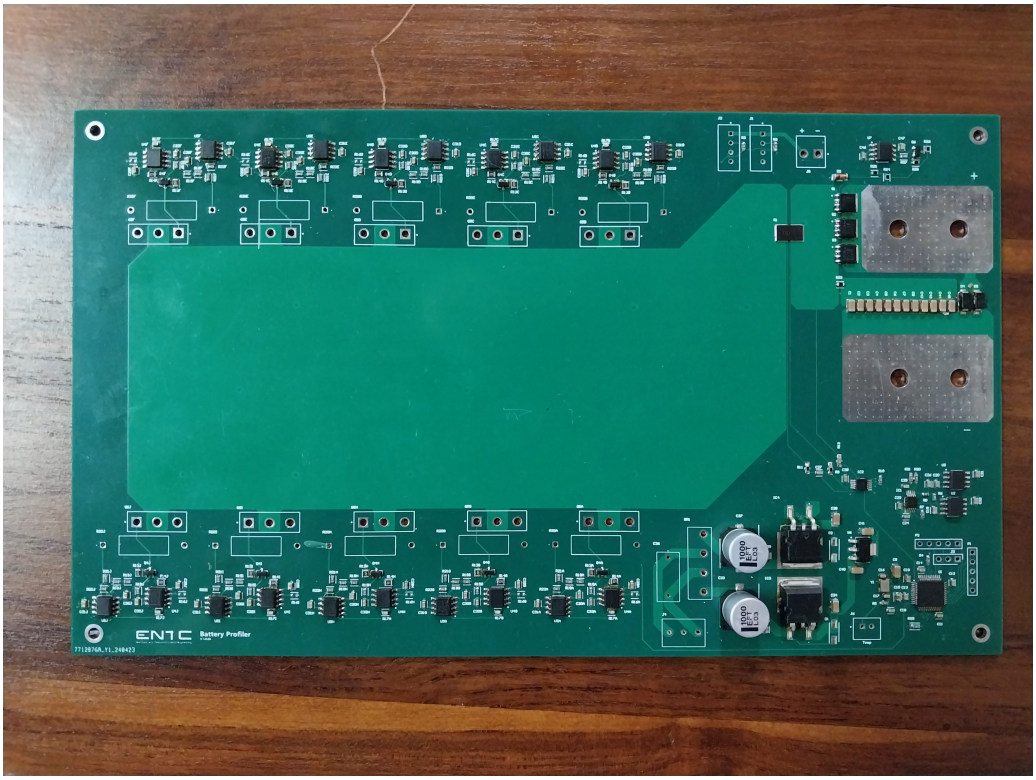

Figure 17: Photo of the soldered PCB

4 Enclosure

The following are designs of the enclosure of the product. It is made out of sheet metal and the front cover is of plastic which has been designed to be capable of being injection molded. The reason for using sheet metal is because the components inside are would get hot and sheet metal is a much better dissipator of heat than plastic. Additionally, air vents are present for cooling purposes.

Figure 18: Overall View of the product

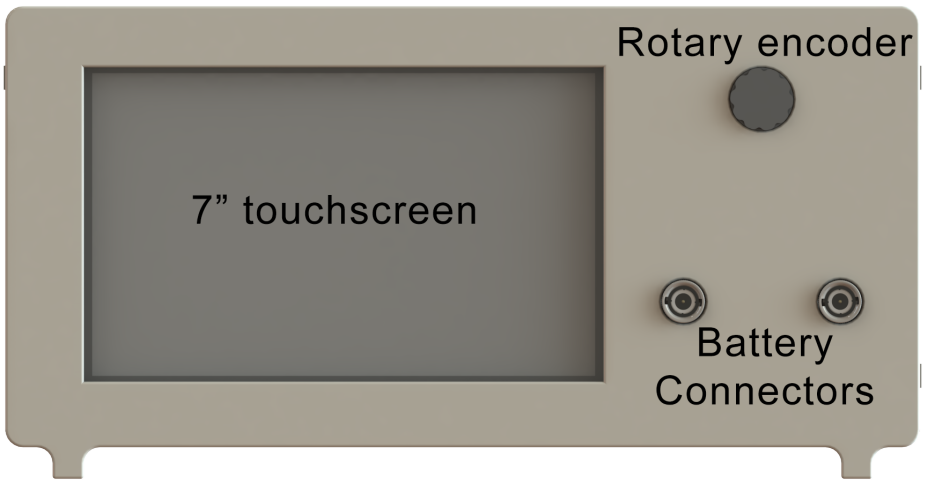

Figure 19: Front View of the product

Shown above is the front view of the product. A 7 inch touch screen of resolution 1024 x 600 is present for the user interface. The required outputs of the device are shown here and users may interact with the device via this screen. The number pad is used to input numerical input to the device. This can be used to set the current or power required to discharge the battery. The rotary encoder is present as an additional form of input in addition to the number pad. The battery needs to be connected tp the connector terminals shown

Below is the rear view of the product. It consists of the power connector which is fused and

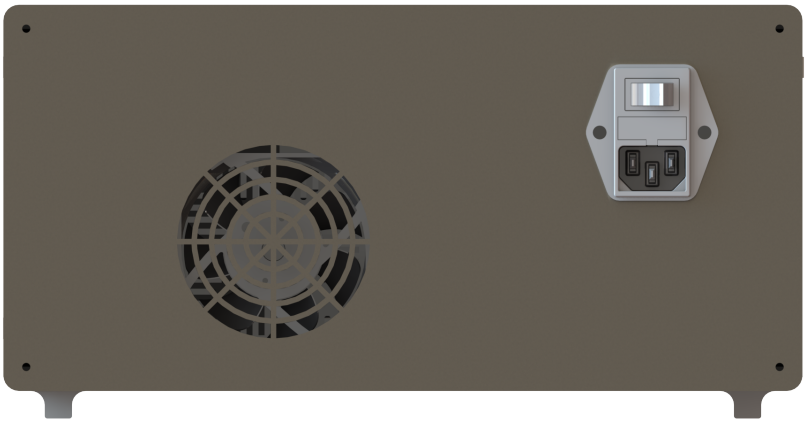

Figure 20: Rear View of the product

the hot air exhaust. The exhaust vent should not be blocked in order to allow optimal airflow.

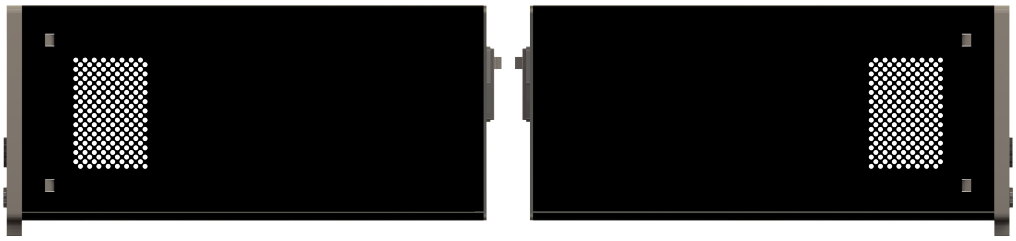

Figure 21: Left and right views

These are the side views of the product. The cool air intakes are present here. Similar to the exhaust, these too should not be blocked.

This is the model tree of the main sheet metal part. Modeling started with first getting

Figure 22: Sheet metal model tree

the overall shape of the part. To achieve this, modeling commenced from the top surface due to it's large surface area and then the edge was bent and extended to the sides. Using another bend, it was possible to obtain the bottom two surfaces. Afterwards, relevant screw holes were added and air vents were also added making sure the fans will have a good intake by placing the vents in the appropriate locations.

The model tree of the front plastic cover is also shown. This was modelled starting from the overall structure of the cover as a rectangle. This was extruded and holes were added for the display, number pad, rotary encoder and battery connectors. Feet were added to make the product face upwards giving users a better angle of the display, buttons and encoder. Snap hooks were added to fit the cover to the sheet metal part.

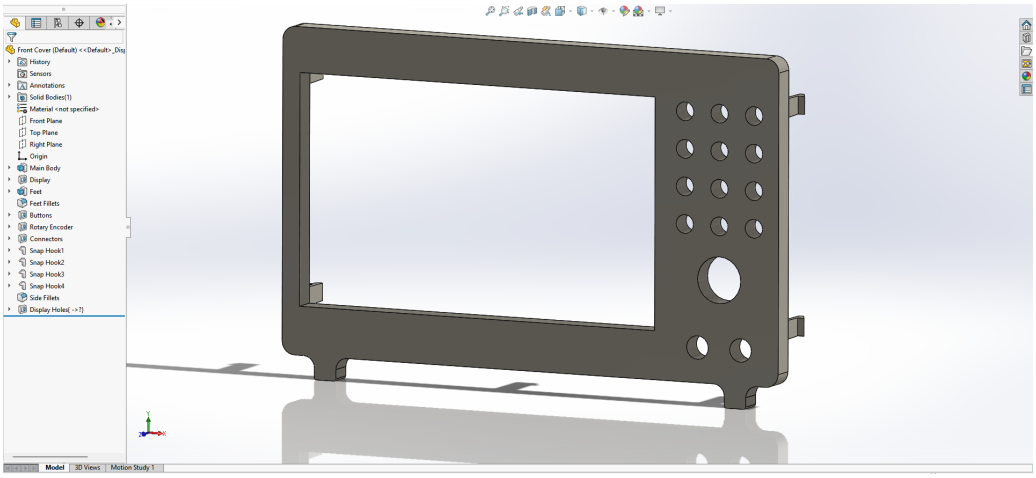

Figure 23: Plastic model tree

5 Code

The code for our project is included here. For the purpose of coding, we used embedded C as the language and the STM32 Cube IDE as the development environment. In order to facilitate modularity, we created libraries for individual functions of the code.

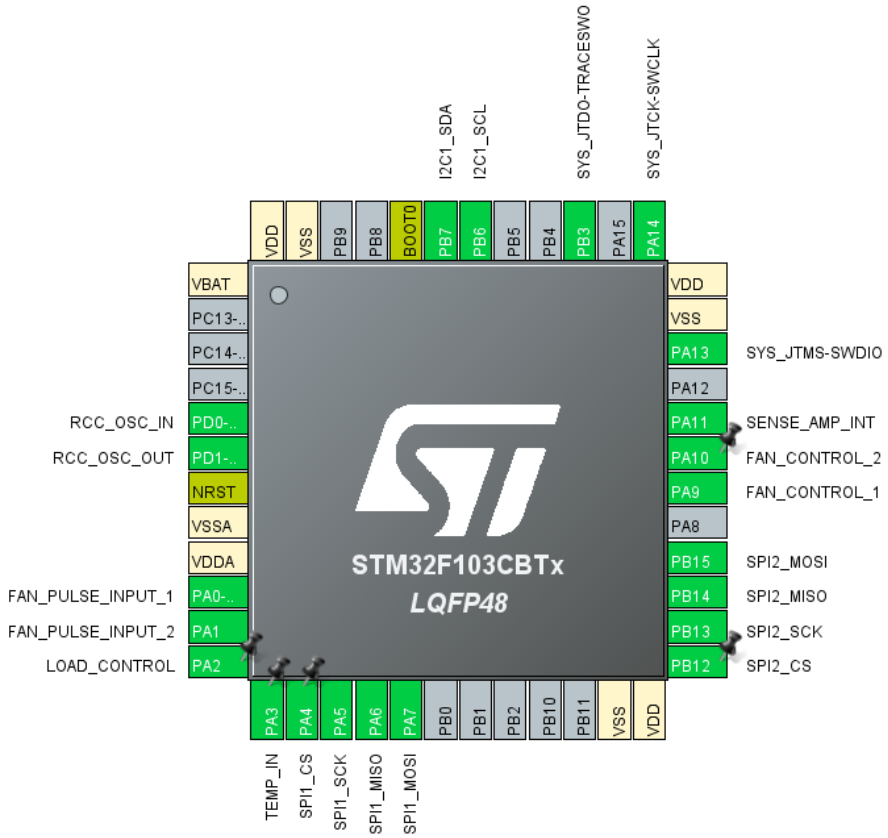

Figure 24: Pin usage in the microcontroller

5.1 fancontrol.h

This is the library responsible for controlling the fan speed based on the temperature measured. It implements a PID algorithm.

```
// Author: Warren Jayakumar

// Define a type definition for PID
typedef struct {
    float kp;
    float ki;
    float kd;
    float prev_error;
    float integral;
    float dt;
} PID_HandleTypeDef;

// Function prototypes
void PID_Init(PID_HandleTypeDef *pid, float kp, float ki, float kd, float dt);
float PID_Compute(PID_HandleTypeDef *pid, float setpoint, float measured);
float Convert_ADCValue_To_Temperature(uint32_t adcValue);
ADC_HandleTypeDef hadc1;
TIM_HandleTypeDef htim2;

// Initialize PID
void PID_Init(PID_HandleTypeDef *pid, float kp, float ki, float kd, float dt) {
    pid->kp = kp;
    pid->ki = ki;
    pid->kd = kd;
    pid->prev_error = 0;
    pid->integral = 0;
    pid->dt = dt;}

// Function to calculate PID output
float PID_Compute(PID_HandleTypeDef *pid, float setpoint, float measured) {
    float error = setpoint - measured;
    pid->integral += error * pid->dt;
    float derivative = (error - pid->prev_error) / pid->dt;
    pid->prev_error = error;

    float output = (pid->kp * error) + (pid->ki * pid->integral)
            + (pid->kd * derivative);
    return output;
}

// Temeprature conversion for ADC
float Convert_ADCValue_To_Temperature(uint32_t adcValue) {
    // Convert the ADC value to temperature
    float voltage = (adcValue / 4095.0) * 3.3;
    float temperature = voltage * 100.0;
    return temperature;
}
```

5.2 movingaverage.h

This library is implemented to calculate the moving average of the measured data points in order to smoothen the values.

```
// Written by Yasiru Alahakoon

uint8_t size = 10; // Here, we use 10 point moving average
typedef struct {
    float array[10];
    uint8_t index;
} MovingAverageArray;

void InitMovingAverageArray(MovingAverageArray *maArray)
{
    for (uint8_t i = 0; i < size; i++) {
        maArray->array[i] = 0.0f;
    }
    maArray->index = 0;
}

void UpdateMovingAverageArray(MovingAverageArray *maArray, float newValue)
{
    maArray->array[maArray->index] = newValue;
    maArray->index = (maArray->index + 1) % size;
}

float ComputeMovingAverage(MovingAverageArray *maArray)
{
    float sum = 0;
    for (uint8_t i = 0; i < size; i++) {
        sum += maArray->array[i];
    }
    return sum / size;
}
// Below functions will need to be implemented.
For now, we just return average readings
float GetVoltage(void)
{
    return (float)(rand() % 100);
}

float GetCurrent(void)
{
    return (float)(rand() % 100);
}

float GetPower(void)
{
    return (float)(rand() % 100);
}
```

```
float GetEnergy(void)
{
    return (float)(rand() % 100);
}
```

5.3 AD5693R.c

The following libraries are used to communicate via I2C with the Digital to Analog Converter.

```
/*
 * AD5693R.c
 *
 *  Created on: Mar 14, 2024
 *      Author: Nirosh Lakshan
 */

#include "AD5693R.h"

uint8_t AD5693R_Initialize(AD5693R *dev, I2C_HandleTypeDef *i2cHandle){
    dev->i2cHandle = i2cHandle;
    HAL_StatusTypeDef status;
    uint8_t errNum = 0;

    uint16_t dacinputValue = 0x0000;
    status = AD5693R_WriteDacIn(dev,&dacinputValue);
    errNum += (status !=HAL_OK);
    return errNum;
}

HAL_StatusTypeDef AD5693R_WriteInput(AD5693R *dev ,uint16_t *regData ){
    HAL_StatusTypeDef status;
    uint8_t data[2] = {((*regData >> 8) & 0x00FF),(*regData & 0x00FF)};
    status = AD5693R_WriteRegister(dev,AD5693R_WRITE_INPUT,data,2);
    return status;
}

HAL_StatusTypeDef AD5693R_UpdateDac(AD5693R *dev , uint16_t *regData ){
    HAL_StatusTypeDef status;
    uint8_t data[2] = {((*regData >> 8) & 0x00FF),(*regData & 0x00FF)};
    status = AD5693R_WriteRegister(dev,AD5693R_UPDATE_DAC,data,2);
    return status;
}

HAL_StatusTypeDef AD5693R_WriteDacIn(AD5693R *dev , uint16_t *regData ){
    HAL_StatusTypeDef status;
    uint8_t data[2] = {((*regData >> 8) & 0x00FF),(*regData & 0x00FF)};
    status = AD5693R_WriteRegister(dev,AD5693R_WRITE_DAC_IN,data,2);
    return status;
}

HAL_StatusTypeDef AD5693R_ChangeGain(AD5693R *dev ,uint8_t gain_val ){
```

```
    HAL_StatusTypeDef status;
    uint16_t controlReg;
    if(gain_val == 1){
    controlReg = 0x0000;
    }else if(gain_val == 2){
    controlReg = 0x0001;
    }
    uint8_t data[2] = {((controlReg >> 8) & 0x00FF),(controlReg & 0x00FF)};
    status = AD5693R_WriteRegister(dev,AD5693R_WRITE_CONTROL,data,2);
    return status;
}

HAL_StatusTypeDef AD5693R_WriteRegister(AD5693R *dev , uint8_t reg,
uint8_t *data, uint8_t length ){
    return HAL_I2C_Mem_Write( dev->i2cHandle, AD5693R_I2C_ADDR, reg, I2C_MEMADD_SIZE
}
```

The following is the AD5693R.h library including all the type definitions

```
/*
 * AD5693R.h
 *
 *  Created on: Mar 14, 2024
 *      Author: Nirosh Lakshan
 */

#ifndef AD5693R_H
#define AD5693R_H

#include "stm32f1xx_hal.h" /*Needed for I2C*/

//I2C address
#define AD5693R_I2C_ADDR (0x68 << 1)

//Commands
#define AD5693R_WRITE_INPUT 0x10    /*0001XXXX*/
#define AD5693R_UPDATE_DAC 0x20^^I^^I/*0010XXXX*/
#define AD5693R_WRITE_DAC_IN 0x30^^I/*0011XXXX*/
#define AD5693R_WRITE_CONTROL 0x40^^I/*0100XXXX*/


typedef struct {
    I2C_HandleTypeDef *i2cHandle;


}AD5693R;

uint8_t AD5693R_Initialize(AD5693R *dev, I2C_HandleTypeDef *i2cHandle);
HAL_StatusTypeDef AD5693R_WriteInput(AD5693R *dev ,uint16_t *regData );
HAL_StatusTypeDef AD5693R_UpdateDac(AD5693R *dev , uint16_t *data );
HAL_StatusTypeDef AD5693R_WriteDacIn(AD5693R *dev , uint16_t *data );
```

```
HAL_StatusTypeDef AD5693R_ChangeGain(AD5693R *dev ,uint8_t gain_val );
HAL_StatusTypeDef AD5693R_WriteRegister(AD5693R *dev , uint8_t reg,
uint8_t *data ,uint8_t length);
#endif /* INC_TPS55288_H_ */
```

5.4 INA229.c

This library is responsible for communicating with the sense amplifier and the Raspberry PI via SPI protocol.

```
 /*
 * INA229.c
 * Library for 20 bit ADC
 * Created on: April 18, 2024
 *      Authors: Nirosh Lakshan , Tharushi Karvita
 *
 */

#include "INA229.h"
#include "main.h"
uint8_t INA229_Initialize(INA229 *dev, SPI_HandleTypeDef *spiHandle){
    dev->spiHandle = spiHandle;
    return 0;
}

// Bit manipulation functions
void INA229_SetRST(INA229 *dev) {
    dev->config |= (1 << 15);
}

void INA229_ClearRST(INA229 *dev) {
    dev->config &= ~(1 << 15);
}
 //
void INA229_SetRSTACC(INA229 *dev) { //Resets the contents of
accumulation registers ENERGY and CHARGE to 0
    dev->config |= (1 << 14);
}

void INA229_ClearRSTACC(INA229 *dev) {
    dev->config &= ~(1 << 14);
}
//

void INA229_SetTEMPCOMP(INA229 *dev) {
    dev->config |= (1 << 5);
}

void INA229_ClearTEMPCOMP(INA229 *dev) {
    dev->config &= ~(1 << 5);
}
```

```
//

void INA229_SetADCRANGE(INA229 *dev) {
    dev->config |= (1 << 4);
}

void INA229_ClearADCRANGE(INA229 *dev) {
    dev->config &= ~(1 << 4);
}

//
void INA229_SetCONVDLY(INA229 *dev, uint8_t value) { //Sets the Delay for
initial ADC conversion in steps of 2 ms.
    dev->config &= ~(0xFF << 6);  // Clear the CONVDLY bits
    dev->config |= ((value & 0xFF) << 6);  // Set the new value
}

// Bit manipulation functions for the ADC_CONFIG Register
void INA229_SetMODE(INA229 *dev, uint8_t mode) {
    dev->adc_config &= ~(0xF << 12);  // Clear the MODE bits
    dev->adc_config |= ((mode & 0xF) << 12);  // Set the new mode
}

uint8_t INA229_GetMODE(uint16_t reg_value) {
    return (reg_value >> 12) & 0xF;  // Extract the MODE bits
}

void INA229_SetVBUSCT(INA229 *dev, uint8_t value) {
    dev->adc_config &= ~(0x7 << 9);  // Clear the VBUSCT bits
    dev->adc_config |= ((value & 0x7) << 9);  // Set the new value
}

uint8_t INA229_GetVBUSCT(uint16_t reg_value) {
    return (reg_value >> 9) & 0x7;  // Extract the VBUSCT bits
}

void INA229_SetVSHCT(INA229 *dev, uint8_t value) {
    dev->adc_config &= ~(0x7 << 6);  // Clear the VSHCT bits
    dev->adc_config |= ((value & 0x7) << 6);  // Set the new value
}

uint8_t INA229_GetVSHCT(uint16_t reg_value) {
    return (reg_value >> 6) & 0x7;  // Extract the VSHCT bits
}

void INA229_SetVTCT(INA229 *dev, uint8_t value) {
    dev->adc_config &= ~(0x7 << 3);  // Clear the VTCT bits
    dev->adc_config |= ((value & 0x7) << 3);  // Set the new value
}

uint8_t INA229_GetVTCT(uint16_t reg_value) {
```

```
    return (reg_value >> 3) & 0x7;  // Extract the VTCT bits
}

void INA229_SetAVG(INA229 *dev, uint8_t value) {
    dev->adc_config &= ~0x7;  // Clear the AVG bits
    dev->adc_config |= (value & 0x7);  // Set the new value
}

uint8_t INA229_GetAVG(uint16_t reg_value) {
    return reg_value & 0x7;  // Extract the AVG bits
}

// Functions to read from the ADC
HAL_StatusTypeDef INA229_ReadShuntVoltage(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[3];
    status = INA229_ReadRegister(dev,INA229_VSHUNT,data,3);
    if(status ==HAL_OK){
        dev->shunt_voltage = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}

HAL_StatusTypeDef INA229_ReadBusVoltage(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[3];
    status = INA229_ReadRegister(dev,INA229_VBUS,data,3);
    if(status ==HAL_OK){
        dev->bus_voltage = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}

HAL_StatusTypeDef INA229_ReadCurrent(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[3];
    status = INA229_ReadRegister(dev,INA229_CURRENT,data,3);
    if(status ==HAL_OK){
        dev->current = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}

HAL_StatusTypeDef INA229_ReadEnergy(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[5];
    status = INA229_ReadRegister(dev,INA229_ENERGY,data,5);
    if(status ==HAL_OK){
        dev->current = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
```

```
}

HAL_StatusTypeDef INA229_ReadPower(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[3];
    status = INA229_ReadRegister(dev,INA229_POWER,data,3);
    if(status ==HAL_OK){
        dev->current = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}
HAL_StatusTypeDef INA229_ReadFlags(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[2];
    status = INA229_ReadRegister(dev,INA229_DIAG_ALRT,data,2);
    if(status ==HAL_OK){
        dev->flags = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}
HAL_StatusTypeDef INA229_ReadManufactureId(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[2];
    status = INA229_ReadRegister(dev,INA229_MANUFACTURER_ID,data,2);
    if(status ==HAL_OK){
        dev->manufacture_id = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}
HAL_StatusTypeDef INA229_ReadDeviceId(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[2];
    status = INA229_ReadRegister(dev,INA229_DEVICE_ID,data,2);
    if(status ==HAL_OK){
        dev->device_id = (data[0] << 16) | (data[1] << 8) | data[2];
    }
    return status;
}

HAL_StatusTypeDef INA229_ReadConfigReg(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[2];
    status = INA229_ReadRegister(dev,INA229_CONFIG,data,2);
    dev->config = (data[0] << 8) | data[1];
    return status;
}

HAL_StatusTypeDef INA229_ReadAdcConfigReg(INA229 *dev){
    HAL_StatusTypeDef status;
    uint8_t data[2];
    status = INA229_ReadRegister(dev,INA229_ADC_CONFIG,data,2);
```

```
    dev->adc_config = (data[0] << 8) | data[1];
    return status;
}


// Functions to write to the ADC
HAL_StatusTypeDef INA229_WriteConfig(INA229 *dev){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_CONFIG,dev->config,2);
    return status;
}

HAL_StatusTypeDef INA229_WriteAdcConfigs(INA229 *dev){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_ADC_CONFIG,dev->adc_config,1);
    return status;
}


HAL_StatusTypeDef INA229_WriteShuntConfig(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_SHUNT_CAL,*config,2);
    return status;
}

HAL_StatusTypeDef INA229_WriteShuntTempConfig(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_SHUNT_TEMPCO,*config,2);
    return status;
}

HAL_StatusTypeDef INA229_WriteSOVL(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_SOVL,*config,2);
    return status;
}

HAL_StatusTypeDef INA229_WriteSUVL(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_SUVL,*config,2);
    return status;
}


HAL_StatusTypeDef INA229_WriteBOVL(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_BOVL,*config,2);
    return status;
}

HAL_StatusTypeDef INA229_WriteBUVL(INA229 *dev, uint16_t *config){
```

```
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_BUVL,*config,2);
    return status;
}



HAL_StatusTypeDef INA229_WriteTempLimit(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_TEMP_LIMIT,*config,1);
    return status;
}

HAL_StatusTypeDef INA229_WritePowerLimit(INA229 *dev, uint16_t *config){
    HAL_StatusTypeDef status;
    status = INA229_WriteRegister(dev,INA229_PWR_LIMIT,*config,1);
    return status;
}


HAL_StatusTypeDef INA229_ReadRegister(INA229 *dev , uint8_t reg,
uint8_t *rxBuf, uint8_t length ){
    HAL_StatusTypeDef status;
    uint8_t txBuf[1] = {(((reg << 2) & 0xFC) | 0x01)}; //B7,B6,B5,B4,B3,B2,0,1
    HAL_GPIO_WritePin(GPIOA,SPI1_CS_Pin,GPIO_PIN_RESET);
    status= HAL_SPI_TransmitReceive(dev->spiHandle,txBuf,rxBuf,
    length,HAL_MAX_DELAY);
    HAL_GPIO_WritePin(GPIOA,SPI1_CS_Pin,GPIO_PIN_SET);
    return status;
}

HAL_StatusTypeDef INA229_WriteRegister(INA229 *dev , uint8_t reg,
uint16_t value, uint8_t length ){
    HAL_StatusTypeDef status;
    uint8_t txBufLength= length+1;
    uint8_t txBuf[txBufLength];
    txBuf[0] = (((reg << 2) & 0xFC) | 0x00); //B7,B6,B5,B4,B3,B2,0,0
    for (int i = 1; i < txBufLength; i++) {
        txBuf[i] = (value >> (8 * i)) & 0xFF;
    }
    HAL_GPIO_WritePin(GPIOA,SPI1_CS_Pin,GPIO_PIN_RESET);
    status= HAL_SPI_Transmit(dev->spiHandle,txBuf,length,HAL_MAX_DELAY);
    HAL_GPIO_WritePin(GPIOA,SPI1_CS_Pin,GPIO_PIN_SET);
    return status;
}
```

The following includes the type definitions

```
/*
 * INA229.h
 *
 *  Created on: April 18, 2024
```

```
 *      Authors: Nirosh Lakshan , Tharushi Karvita
 */

#ifndef INA229_H
#define INA229_H

#include "stm32f1xx_hal.h" /*Needed for I2C*/

//Commands
#define INA229_CONFIG 0x00
#define INA229_ADC_CONFIG 0x01
#define INA229_SHUNT_CAL 0x02
#define INA229_SHUNT_TEMPCO 0x03
#define INA229_VSHUNT 0x04
#define INA229_VBUS 0x05
#define INA229_DIETEMP 0x06
#define INA229_CURRENT 0x07
#define INA229_POWER 0x08
#define INA229_ENERGY 0x09
#define INA229_CHARGE 0x0A
#define INA229_DIAG_ALRT 0x0B
#define INA229_SOVL 0x0C
#define INA229_SUVL 0x0D
#define INA229_BOVL 0x0E
#define INA229_BUVL 0x0F
#define INA229_TEMP_LIMIT 0x10
#define INA229_PWR_LIMIT 0x11
#define INA229_MANUFACTURER_ID 0x3E
#define INA229_DEVICE_ID 0x3F


typedef struct {
^^ISPI_HandleTypeDef *spiHandle;
^^Iuint32_t shunt_voltage;
^^Iuint32_t bus_voltage;
^^Iuint32_t current;
^^Iuint16_t config;
^^Iuint16_t adc_config;
^^Iuint16_t flags;
^^Iuint64_t energy;
^^Iuint32_t power;
^^Iuint16_t device_id;
^^Iuint16_t manufacture_id;
}INA229;


uint8_t INA229_Initialize(INA229 *dev, SPI_HandleTypeDef *spiHandle);

HAL_StatusTypeDef INA229_ReadShuntVoltage(INA229 *dev);
HAL_StatusTypeDef INA229_ReadBusVoltage(INA229 *dev);
HAL_StatusTypeDef INA229_ReadCurrent(INA229 *dev);
```

```
HAL_StatusTypeDef INA229_ReadEnergy(INA229 *dev);
HAL_StatusTypeDef INA229_ReadPower(INA229 *dev);
HAL_StatusTypeDef INA229_ReadFlags(INA229 *dev);
HAL_StatusTypeDef INA229_ReadManufactureId(INA229 *dev);
HAL_StatusTypeDef INA229_ReadDeviceId(INA229 *dev);
HAL_StatusTypeDef INA229_ReadDieTemp(INA229 *dev);

HAL_StatusTypeDef INA229_ReadConfigReg(INA229 *dev);
HAL_StatusTypeDef INA229_WriteConfig(INA229 *dev );
HAL_StatusTypeDef INA229_WriteAdcConfigs(INA229 *dev );
HAL_StatusTypeDef INA229_WriteShuntConfig(INA229 *dev ,uint16_t *config );
HAL_StatusTypeDef INA229_WriteShuntTempConfig(INA229 *dev ,uint16_t *config );
HAL_StatusTypeDef INA229_WriteSOVL(INA229 *dev ,uint16_t *config );
//Shunt Overvoltage Threshold
HAL_StatusTypeDef INA229_WriteSUVL(INA229 *dev ,uint16_t *config );
//Shunt Undervoltage Threshold
HAL_StatusTypeDef INA229_WriteBOVL(INA229 *dev ,uint16_t *config );
//Bus Overvoltage Threshold
HAL_StatusTypeDef INA229_WriteBUVL(INA229 *dev ,uint16_t *config );
//Bus Undervoltage Threshold
HAL_StatusTypeDef INA229_WriteTempLimit(INA229 *dev ,uint16_t *config );
//Temperature Over-Limit Threshold
HAL_StatusTypeDef INA229_WritePowerLimit(INA229 *dev ,uint16_t *config );
// Power Over-Limit Threshold

HAL_StatusTypeDef INA229_ReadRegister(INA229 *dev , uint8_t reg,
uint8_t *rxBuf, uint8_t length );
HAL_StatusTypeDef INA229_WriteRegister(INA229 *dev , uint8_t reg,
uint16_t value, uint8_t length );



void INA229_SetRST(INA229 *dev);
void INA229_ClearRST(INA229 *dev);
void INA229_SetRSTACC(INA229 *dev);
void INA229_ClearRSTACC(INA229 *dev);
void INA229_SetTEMPCOMP(INA229 *dev);
void INA229_ClearTEMPCOMP(INA229 *dev);
void INA229_SetADCRANGE(INA229 *dev);
void INA229_ClearADCRANGE(INA229 *dev);
void INA229_SetCONVDLY(INA229 *dev, uint8_t value);

void INA229_SetMODE(INA229 *dev, uint8_t mode);
uint8_t INA229_GetMODE(uint16_t reg_value);
void INA229_SetVBUSCT(INA229 *dev, uint8_t value);
uint8_t INA229_GetVBUSCT(uint16_t reg_value);
void INA229_SetVSHCT(INA229 *dev, uint8_t value);
uint8_t INA229_GetVSHCT(uint16_t reg_value) ;
void INA229_SetVTCT(INA229 *dev, uint8_t value);
uint8_t INA229_GetVTCT(uint16_t reg_value);
void INA229_SetAVG(INA229 *dev, uint8_t value);
```

```
uint8_t INA229_GetAVG(uint16_t reg_value);

#endif /* INC_TPS55288_H_ */
```

5.5 Main.c

The following is the main code for the battery profiler. The main loop was coded by us while the pin initialization were done by the STM32 CubeIDE GUI.

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
  LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "movingaverage.h"
#include "fancontrol.h"
#include "AD5693R.h"
#include "INA229.h"

/* Private includes ----------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------*/
/* USER CODE BEGIN PM */
```

```c
/* USER CODE END PM */

/* Private variables ---------------------------------------------------*/
ADC_HandleTypeDef hadc1;

I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* 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);
static void MX_SPI2_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */
INA229 adc;
uint16_t adc_config = 0;
// Read the current configuration register value


/* USER CODE END PFP */

/* Private user code ---------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t ARRAY_SIZE = 10;
float setpoint = 25.0; // Desired temperature
float kp = 1.0, ki = 0.1, kd = 0.01; // PID coefficients
PID_HandleTypeDef hpid;
uint8_t index1 = 0, index2 = 0, index3 = 0, index4 = 0;
MovingAverageArray voltagearray;
MovingAverageArray currentarray;
MovingAverageArray powerarray;
MovingAverageArray energyarray;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
```

```
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash
  interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_SPI1_Init();
  MX_SPI2_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
  INA229_Initialize(&adc,&hspi1);
  INA229_ReadConfigReg(&adc);
  INA229_SetRST(&adc);
  //INA229_ClearRST(&adc);
  INA229_SetRSTACC(&adc);
  //INA229_ClearRSTACC(&adc);
  INA229_SetTEMPCOMP(&adc);
  //INA229_ClearTEMPCOMP(&adc);
  INA229_SetADCRANGE(&adc);
  //INA229_ClearADCRANGE(&adc);
  INA229_SetCONVDLY(&adc,0xFF);

  InitMovingAverageArray(&voltagearray);
  InitMovingAverageArray(&currentarray);
  InitMovingAverageArray(&powerarray);
  InitMovingAverageArray(&energyarray);
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {
                uint32_t adcValue = HAL_ADC_GetValue(&hadc1);
                float temperature = Convert_ADCValue_To_Temperature(adcValue);

                float control = PID_Compute(&hpid, setpoint, temperature);

                // Ensure control value is within PWM range (0 to Period)
                if (control < 0)
                    control = 0;
                if (control > 999)
                    control = 999;

                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, control);
                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, control);
            }

            float voltage = GetVoltage();
            float current = GetCurrent();
            float power = GetPower();
            float energy = GetEnergy();

            // Update arrays with new readings
            UpdateMovingAverageArray(&voltagearray, voltage);
            UpdateMovingAverageArray(&currentarray, current);
            UpdateMovingAverageArray(&powerarray, power);
            UpdateMovingAverageArray(&energyarray, energy);

            // Compute moving averages
            float avg1 = ComputeMovingAverage(&voltagearray);
            float avg2 = ComputeMovingAverage(&currentarray);
            float avg3 = ComputeMovingAverage(&powerarray);
            float avg4 = ComputeMovingAverage(&energyarray);

            // Use the moving averages (for demonstration, just print them)

            HAL_Delay(1000);  // Delay for demonstration purposes


  }
  /* USER CODE END 3 */
}
```

```c
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_3;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */
```

```
  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  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();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  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_4;
  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();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief SPI2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI2_Init(void)
{

  /* USER CODE BEGIN SPI2_Init 0 */

  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI2_Init 2 */

  /* USER CODE END SPI2_Init 2 */

}

/**
```

```
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 0;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
```

```
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65535;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

```
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @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_GPIOD_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LOAD_CONTROL_Pin|SPI1_CS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : LOAD_CONTROL_Pin SPI1_CS_Pin */
  GPIO_InitStruct.Pin = LOAD_CONTROL_Pin|SPI1_CS_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 pin : SPI2_CS_Pin */
  GPIO_InitStruct.Pin = SPI2_CS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(SPI2_CS_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : FAN_CONTROL_2_Pin */
  GPIO_InitStruct.Pin = FAN_CONTROL_2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(FAN_CONTROL_2_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```c
#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 */
```

6 User Interface

The user interface of the product has been designed to be intuitive and user-friendly. Following are the screenshots of the user interface. This is the main screen of the product. Here, the user

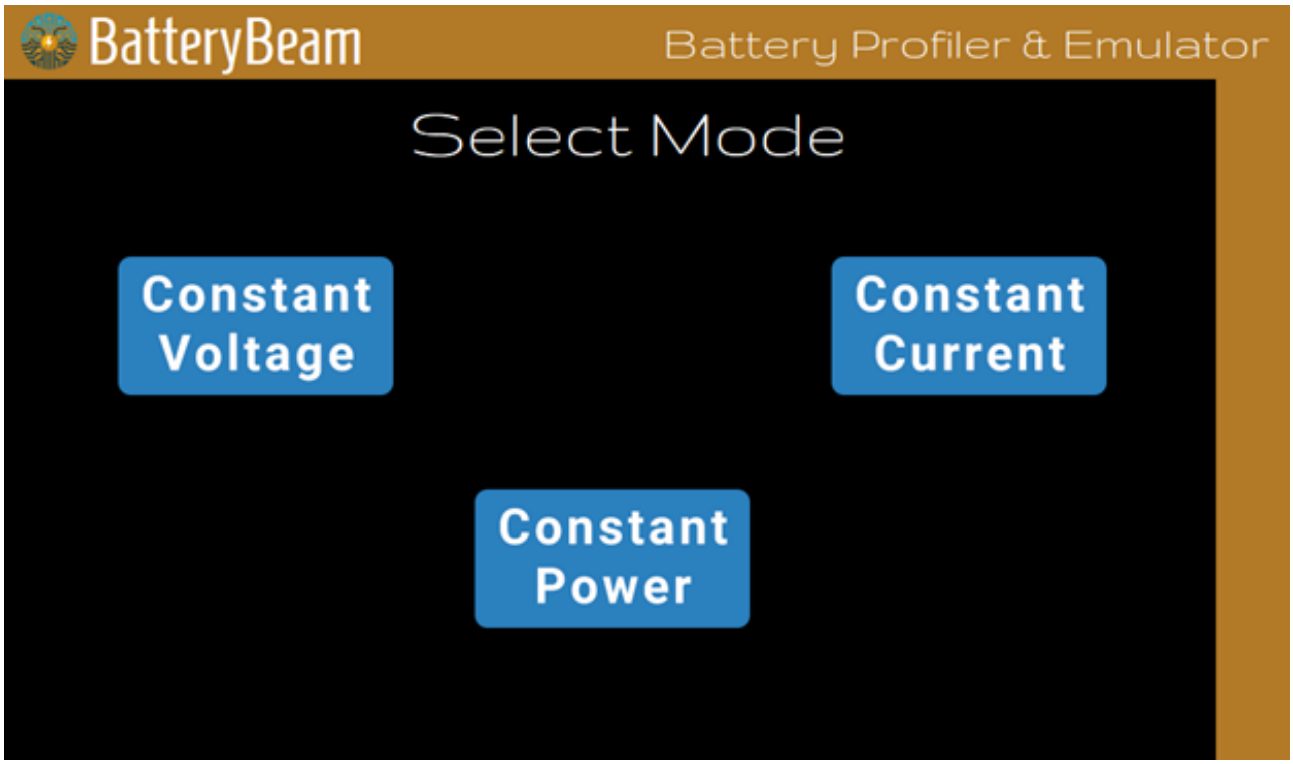

Figure 25: Main Screen

is allowed to select the relevant setting under which the battery is required to discharge for the type of test performed. This can be interacted with a touch screen and selecting a mode takes the user to the next screen shown.

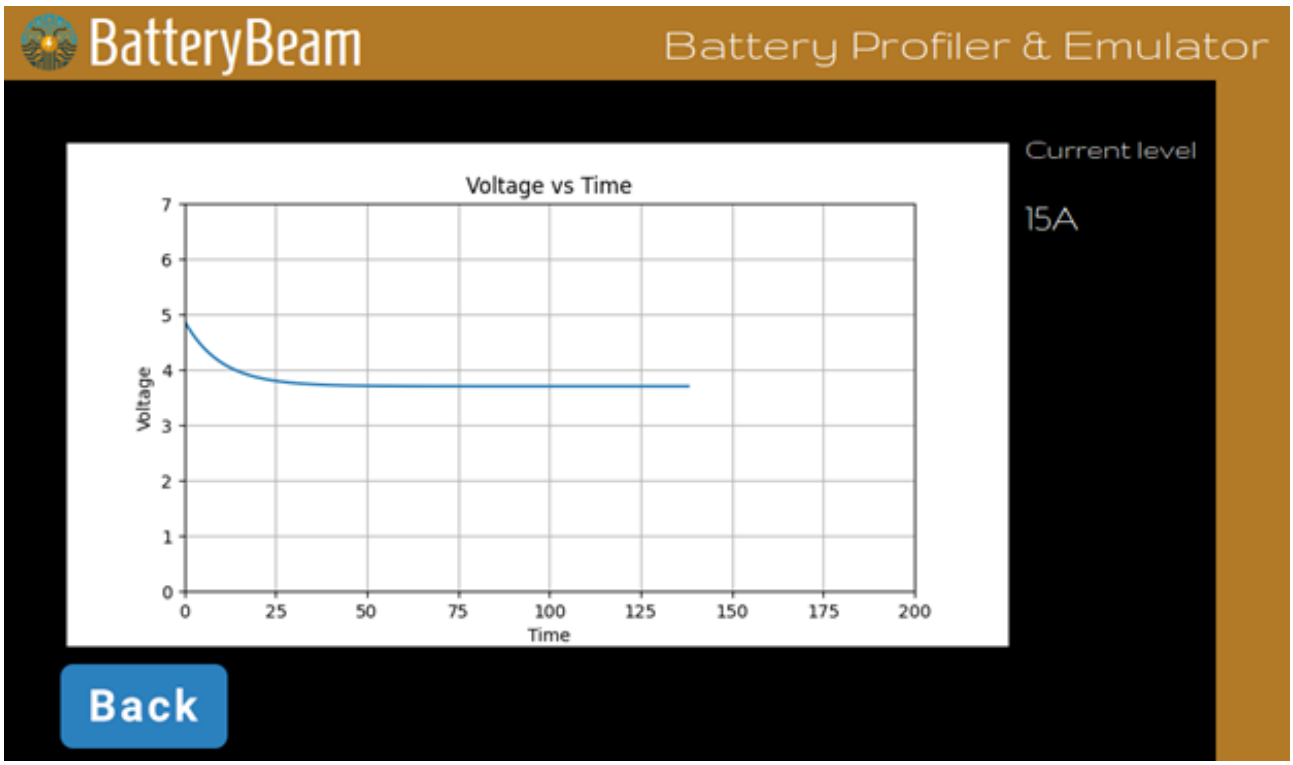

Figure 26: Graph

This is the measurement section. The user should first enter the value of the constant current, power or voltage using the rotary encoder or the number pad and press the start button in the button pad. This will start the discharge process. When discharging, the voltage at the output is measured and plotted on the graph.

7 Conclusion

The product has been designed to be able to appropriately satisfy the needs of the users. Thorough research has been done to make sure this product meets the needs of users and functions appropriately as expected. Further progress can be made by obtaining feedback from users after the product has been sold.

8 References

1. Lithium Battery Testing
2. Introducing Keysight's E36731A Battery Emulator and Profiler
3. M. I and V. Chayapathy, "Programmable DC Electronic Load for Testing on-Board Voltage Regulators," International Journal of Engineering Research & Technology (IJERT), vol. 6, no. 09, pp. 335-338, September 2017.

9 Appendix A - Timeline

This section describes the timeline of the project including the time period of the project's tasks and the details of the tasks performed during that period.

- 1st February 2024: Deciding on the project and preparing project proposal: The team members gathered to exchange ideas and thoughts on possible project ideas. Following a thorough discussion, we decided on the battery profiler project as it aligns well with the learning outcomes of the module. A brief project proposal was created.

- 1st February 2024 - 8th February 2024: Reviewing progress and planning next steps: During this week, the four of us individually searched for existing resources concerning the project. These resources included existing products, research papers and videos of such products being used in industry.

- 8th February 2024 - 14th February 2024: Creating stakeholder map, observing users and identifying needs: During this period, first we proceeded to identify the stakeholders of the project. These included internal as well as external stakeholders and helped us gain an understanding on how each stakeholder's need had to be satisfied. Following that, we started observing users. This was done online and we were able to find situations where users had used similar products before. We also came up with a list of needs that the project needs to satisfy. These included user expectations as well as functional specifications of the final product.

- 14th February 2024 - 16th February 2024: Stimulation of ideas. During this time, as a team, we came up with innovative ideas that would make our product stand out from the rest and offer users additional features that would make it more convenient for them. Some ideas included data analytics and mobile integration. Such ideas as these, would help us come up with better conceptual designs.

- 6th March 2024 - 7th March 2024: Development of conceptual designs: As the next stage of the project, we came up with conceptual designs that would be potential baselines to build up the final project. During this time, we paid close attention to the needs list making sure our conceptual designs satisfied the needs. We came up with three conceptual designs which included the block diagrams of the final product and the sketch of the proposed product. Upon careful discussion, we evaluated the three conceptual designs and decided on one

- 11th March 2024: Review: We reviewed our work and progress made so far. The purpose of this was to make sure our conceptual design was in line with our goals.

- 12th March 2024: Commencement of circuit design: We designed the block diagram of the March as a baseline for our circuit design

- 13th February 2024 - 15th March 2024: Deciding on components: During this time, we decided on what components will be used on the product. This was especially for the micro controller, pass elements and OP-AMPs.

- 13th February 2024 - 22nd March 2024: Circuit design: During these days, we designed the circuit for the product. We referred to existing schematics to gain additional knowledge. During this time, the main discharging circuit, pass element circuit, analog circuit and microcontroller circuit were designed.

- 26th March 2024: Circuit simulation and breadboard testing: The designed circuit was simulated to make sure it functions as expected. Additionally, one unit out of the 10 parallel elements of the power dissipation circuits was built on a breadboard and tested. The circuit needed a few adjustments and afterward, they functioned as expected.
- 28th March 2024: Addition of active cooling: It was decided to add active cooling to the product. Initially, it was decided to use the heatsink as a passive cooling element but considering safety, it was decided to add fans for active cooling. The new schematic was designed and updated.
- 1st April 2024 - 12th April 2024: PCB Design: During this time, we designed the Printed Circuit Board. Care was made to make sure the PCB attained to all PCB design standards and that it would be manufacturable.
- 1st April 2024 - 12th April 2024: Component Selection: Parallely with the PCB design, we also selected the rest of the components for the project such as resistors and capacitors.
- 17th April 2024: Review: Once more a review of the product was conducted before finalizing the PCB.
- 17th April 2024 - 19th April 2024: Enclosure Design: With the completed PCB, the enclosure design was started. This was done after the PCB design so that the size of the enclosure could be decided.
- 18th April 2024: Sending the PCB for manufacture. The PCB design files were sent to JLCPCB, China to be manufactured.
- 29th April 2024 - 3rd May 2024: UI design: We gathered and designed the user interface of the product. We had to make sure the UI was both functional and easy to use.
- 30th April 2024: Arrival of manufactured PCB
- 7th May 2024: Soldering: On this day, we soldered the components onto one of the PCBs.
- 28th June 2024: Assembly: We assembled the entire product on this day. This involved the system integration, where we placed the PCB and display inside the enclosure and assembled the enclosure.