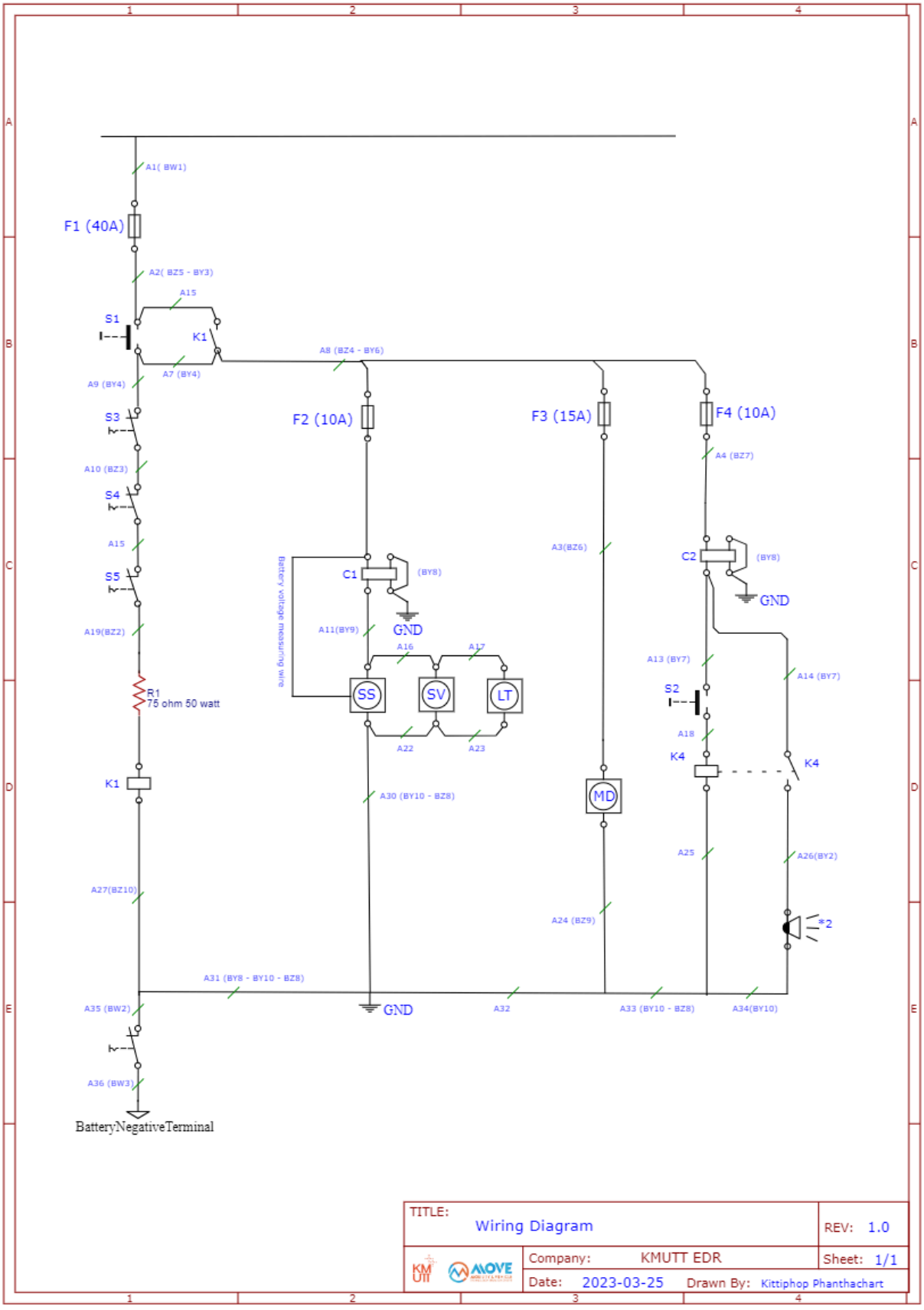
Electrical Document

Team name : KMUTT EDR Epsilon
Team ID : TH0008001
Country : Thailand
Vehicle Category : Urban Concept
Energy Category : Battery Electric

Table of Contents

Energy Supply Diagram



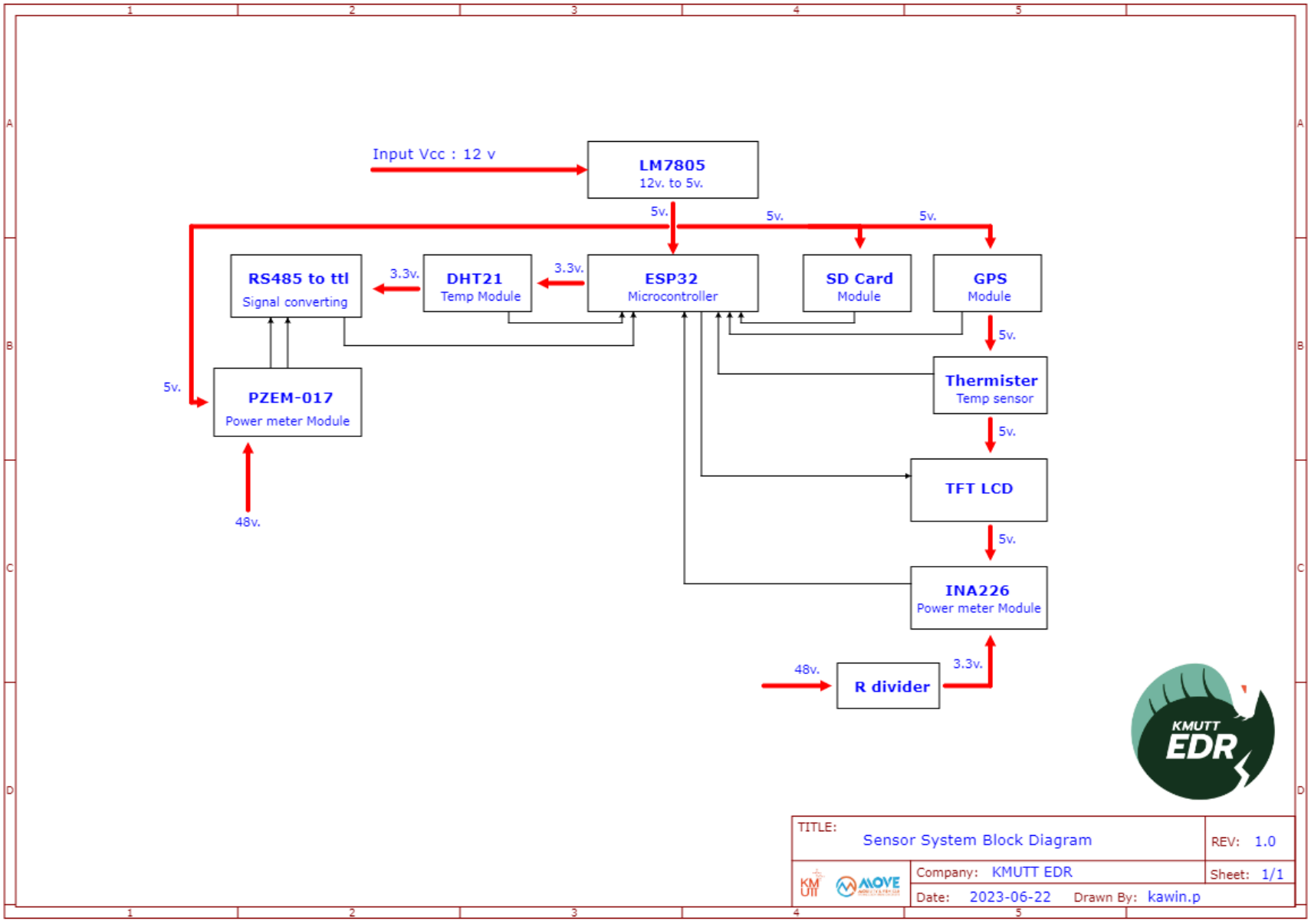
Included:

- 40A fuse as main fuse (F1)
- 10A fuse as sub-system's fuse (F2)
- 15A fuse as motor driver and motor fuse (F3)
- 10A fuse as horn fuse (F4)
- Car start switch (S1)
- Horn switch (S2)
- Internal emergency stop switch (S3)
- External emergency stop switch (S4, S5)
- Battery cut-off switch (S6)
- 12V main relay with 1000VDC 150A contact rating(K1)
- 12V horn relay (K2)
- 48-60 VDC to 12VDC step-down converter (C1,C2)
- The sensor system (SS)
- Windscreen wiper (SV)
- Lighting system (LT)
- Motor System (MD)
- 75 Ohm 50 Watt relay coil current limiting resistor (R1)

Overall Description

This car uses a 48V lithium iron phosphate battery with BMS. The positive terminal of the battery is connected to the car via the 40A main fuse then the power is distributed to each system of the car via the main relay (K1) contact which can be activated by the S1 start switch and can be de-activate by one of the emergency stop switch. The power goes through the fuse and through the dc-to-dc step-down converter to step down the voltage to 12V this car has three step-down converters first converter is connected to the sensor system, windscreen wiper, and lighting system the second converter is connected to the horn system and the last converter is the motor controller system external converter. The ground of these systems is connected together and then connect to the battery negative terminal via the battery cut-off switch this kind of connection can provide mechanical isolation when the battery cut-off switch is opened all system (except for motor controller which has seperate chapter) can be shown as following block diagrams

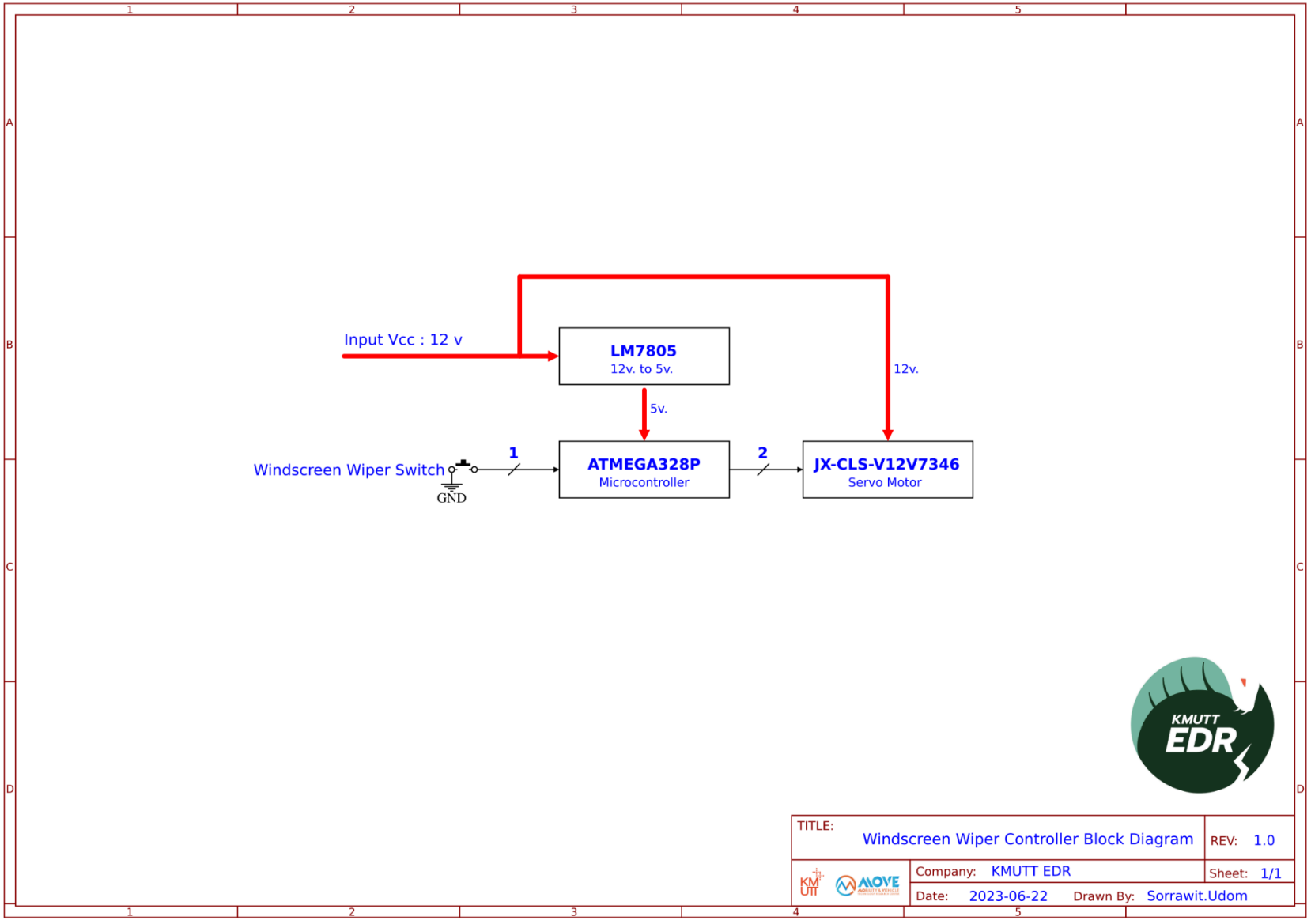
Sensors system block diagram and description



Sensors system description

From our sensor system we provide basic data that we want to know by using ESP32 to be the microcontroller for controlling the sensor and recording the data. The sensor that we used such as DHT21 and Thermistor for recording the temperature and humanity value, for the power measurement part we using the Pzem-017 module for measuring the voltage and we also use INA226 module to be the power meter backup module in case that Pzem-017 is not ready to used. For GPS module we using L86 GNSS for measuring the location, time and speed and we record all of the data into the microsd card by using the microsd card module and display the data to the driver by using tft lcd display.

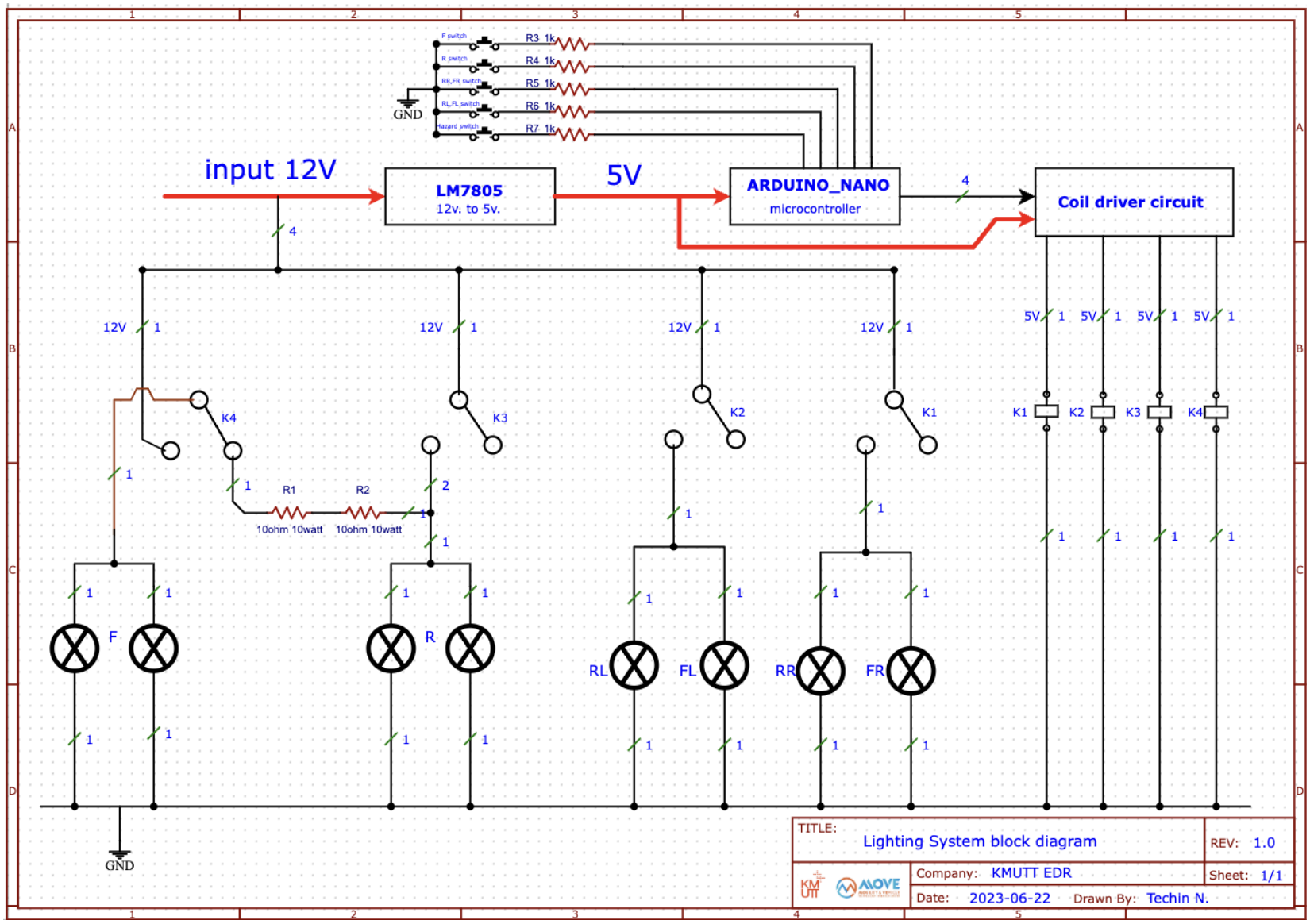
Windscreen Wiper block diagram and description



Windscreen wiper description

This system is designed for controlling the servo motor to work as a car windscreen wiper using the Arduino nano development board which has ATMEGA328P as its microcontroller which receives the control signal from the windscreen wiper switch located on the steering wheel.

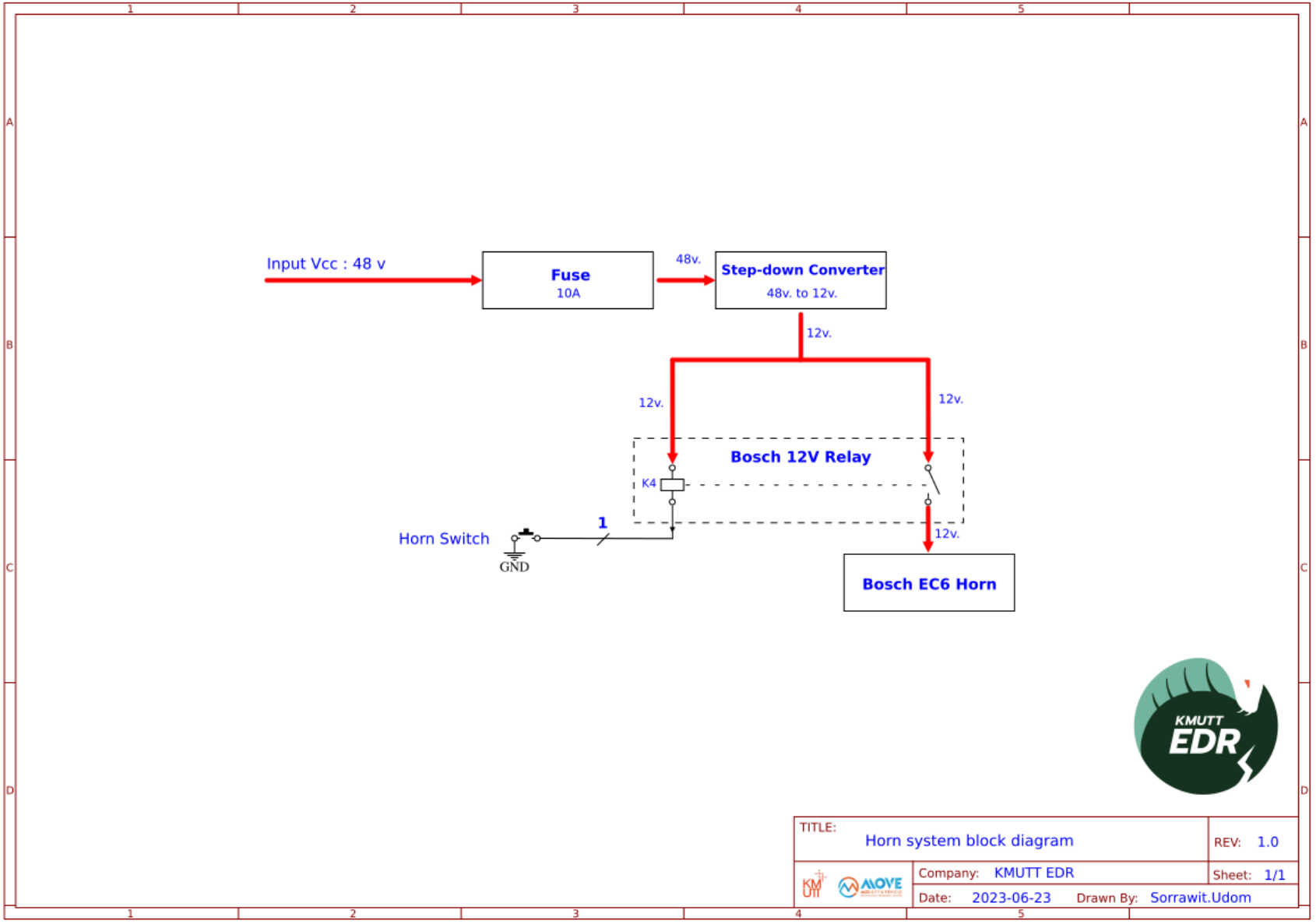
Lighting System block diagram and descrip



Lighting System description

Lighting system is the system designed for controlling the car lighting system that consists of two front headlights, brake signal, turn signal and hazard light signal. Lighting system receives control signals from the switch panel located on the steering wheel.

Horn System block diagram and description

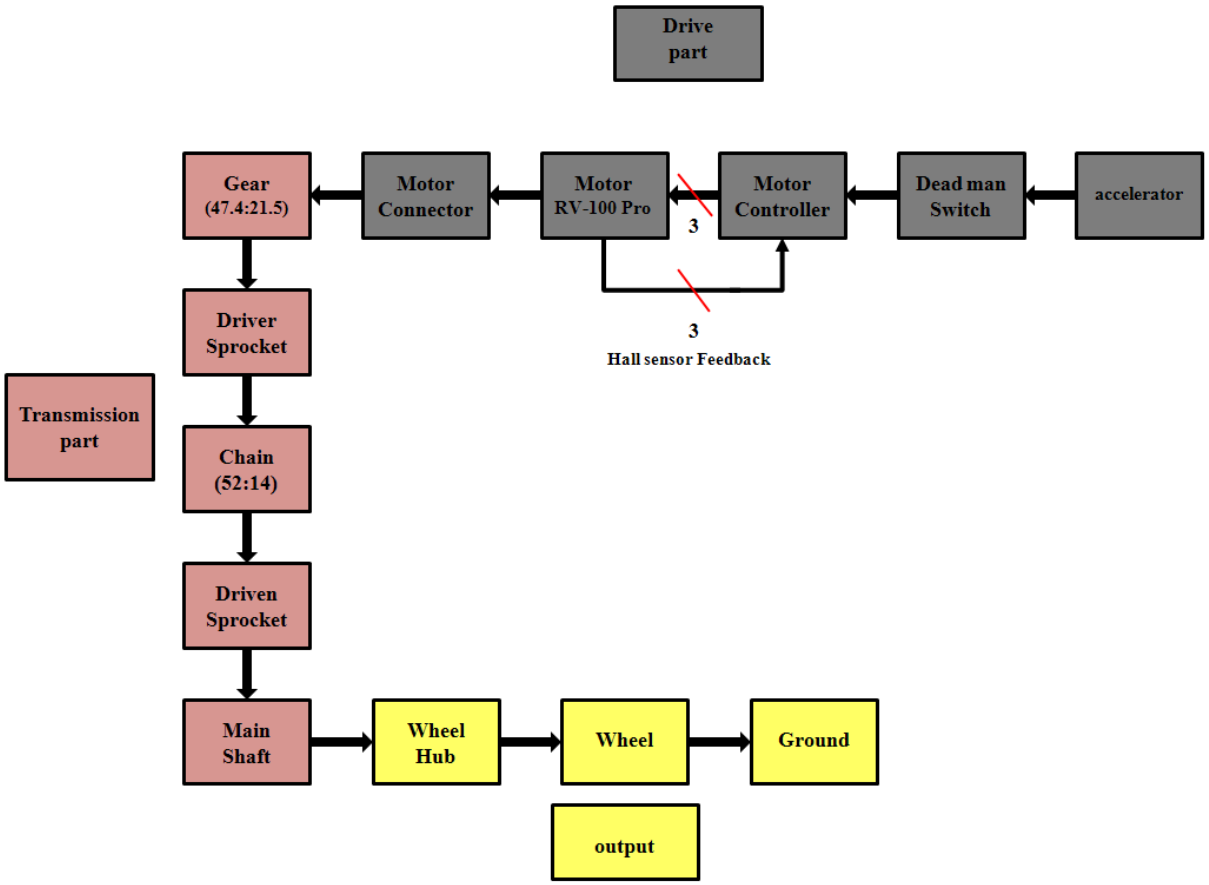


Horn System description

Car horns are connected to the step-down converter through the fuse and normally open contact of the relay and the relay is controlled by the momentary switch. The horn model that is used in this car is BOSCH EC6. According to the datasheet, this horn draws 2.5A to 4A of current, this car has two horns so the horn fuse is 10 A fuse.

Propulsion System Diagram

Motor to road block diagram

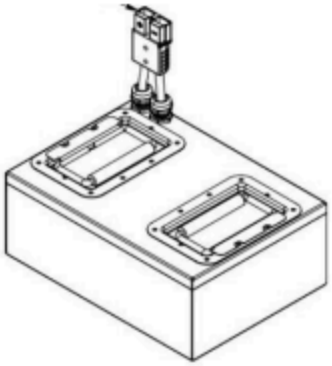


Battery and BMS Documentation

EQ Tech Energy

Green and Affordable Energy for Everyone.

Solution Provider and Equality Enabler in Energy Applications



SPECIFICATIONS

Model: EQ48V18.6Ah

FEATURES

Maintenance free
Longer cycle life
Safe and stable
Constant output power
Environmental friendly
Low self-discharge

APPLICATIONS

Electric vehicles,
Electric solar / wind mobility,
UPS storage system,
Telecommunication backup power,
Medical equipment lighting

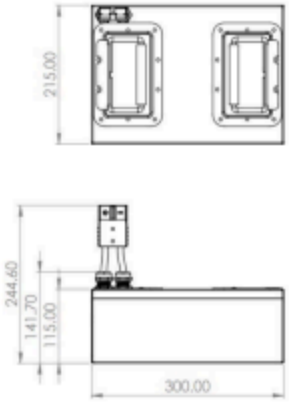
FUNCTIONAL SPECIFICATIONS

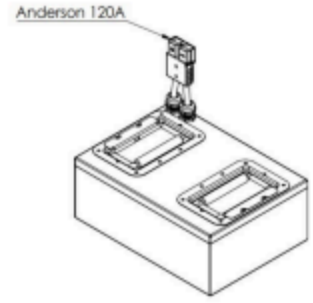
Battery type	Lithium iron phosphate (LFP)
Standard capacity	18.60 Ah
Nominal Voltage	51.20 V
Max. Charge voltage	58.40V
Std. Cut-off voltage	40.00V
Std. Charge current	9.30A (0.50C)
Max. Charge current	18.60A (1.00C)
Charging Time	About 1.00 – 2.00 hours
Max. Cont. Discharging Current	98.58A (5.30C)
Max discharge current (5 mins)	120.00A (6.45C)
Diameter (W x H x L)	21.50 X 11.50 X 30.00 cm
Weight (Approx, including case)	10.00 kg
Charge method (CC/CV)	0.50C (CC), 58.40V cut off
	58.40V (CV), 0.05C cut off
Operate temperature	Charge 0℃~45℃
	Discharge -20℃~65℃
Safety protections	BMS / Active balancer
Life cycle (charge 1.0C, discharge 1.0C at 25 ℃)	70% of initial capacity at ≥ 3,000 cycles

DIMENSION

OUTER DIMENSIONS & TERMINAL TYPE

- **Length (mm)**
  300
- **Width (mm)**
  215
- **Total Height (mm)**
  115





www.gobelpower.com

Intelligent protection board for lithium battery
Operation and maintenance instructions

Product warranty terms

Product Name: battery active equalizer

Warranty period: one year

1 Overview

The intelligent protection board of lithium battery is a management system specially designed for large-capacity series lithium battery packs. which has the functions of voltage acquisition, high current active balance, overcharge, over discharge, over current and over temperature protection, coulomb counter, Bluetooth communication, GPS remote, etc. It can be applied to lithium iron phosphate, ternary lithium and other battery types.

Based on the energy transfer active balance technology with independent intellectual property rights, the protection board can achieve the maximum continuous 2A balance current. High current active balance technology can guarantee the battery consistency, improve the battery life and delay the battery aging to the greatest extent.

The protection board has a supporting mobile app, supporting Android and IOS operating systems. The app can be connected to the protection board via Bluetooth to check the battery working status, modify the working parameters of the protection board, control the switch of charging and discharging, etc. The protection panel is small in size, simple in operation and full in function, which can be widely used in battery pack of small sightseeing bus, scooter, shared car, high-power energy storage, base station standby power supply, solar power station and other products.

2 Main Technical Parameters

2.1 Main Technical Indicators

The main technical indicators of the protection board are shown in Table 1.

Table 1. Main technical indicators of protection board

Technical index	Product model						
	BD6A17S6P	BD6A20S6P	BD6A20S10P	BD6A24S10P	B1A24S15P	B2A24S15P	B2A24S20P
Number of battery strings	13~17	13~20	13~20	13~24	13~24	13~24	13~24

Number of battery strings	15~17	15~20	15~20	15~24	15~24	15~24	15~24
Number of battery strings	17	17~20	17~20	17~24	17~24	17~24	17~24
balance mode	Active balance	Active balance	Active balance	Active balance	Active balance	Active balance	Active balance
Equalizing current	0.6 A	0.6 A	0.6 A	0.6 A	1 A	2 A	2 A
Main circuit conduction internal	1.3 mΩ	1.3 mΩ	0.8mΩ	0.8mΩ	0.5mΩ	0.5mΩ	0.3mΩ
Continuous discharge current	60A	60A	100A	100A	150A	150A	200A
Maximum discharge current	100A	100A	200A	200A	300A	300A	350A
Charging overcurrent	10~60 A	10~60 A	10~100 A	10~100 A	10~150 A	10~150 A	10~200 A
Other interfaces (customized)	RS485	RS485	RS485	RS485	RS485/CAN	RS485/CAN	RS485/CAN
Entry cable	Same port						
Single voltage range	1~5 V						
Voltage acquisition	±5 mV						
Overcharge protection voltage	1.2~4.35 V adjustable						
Overcharge release voltage	1.2~4.35 V adjustable						
Discharge time of charging	2~120S adjustable						
Over discharge protection voltage	1.2~4.35 V adjustable						
Over discharge recovery voltage	1.2~4.35 V adjustable						
Number of temperature	3 个						
Temperature protection	yes						
Short circuit protection	yes						
Coulomb meter	yes						

Bluetooth function	Support Android and IOS
GPS remote (optional)	RS485/GPS

2.2 Application Enviroment

a) Operating temperature range: - 20 ℃ ~ 70 ℃;
b) Power requirements: 40 ~ 100V.
c) Power consumption: 10mA @ 100V in balanced state and 6mA @ 100V in unbalanced state.

3 Connector and Interface Description

3.1 Connector and LED Position Description

The positions of two types of protection board connectors and LED lights are shown in Figure 1 and Figure 2.

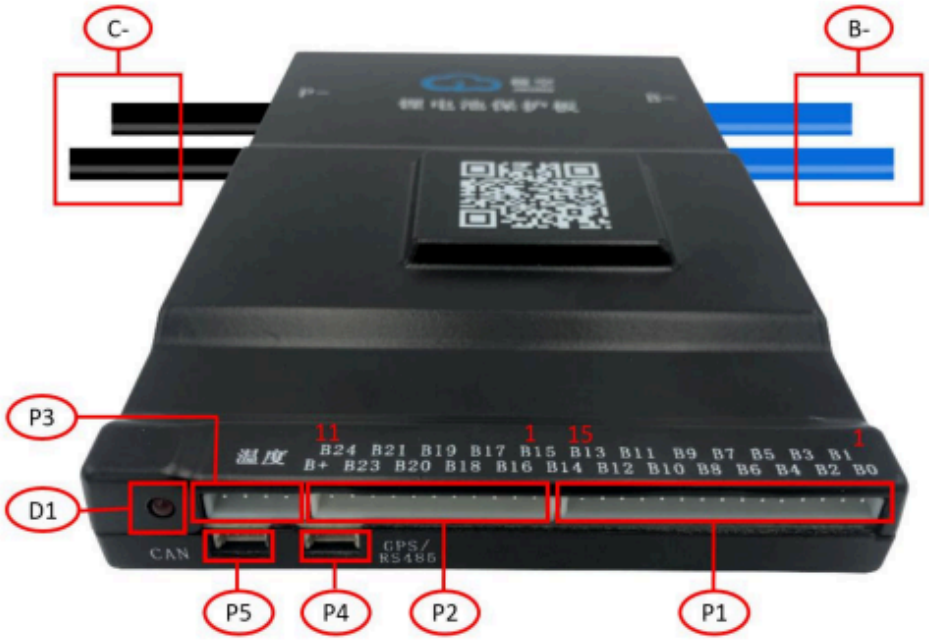


Figure 1. BD6A20S10P/B2A24S10P/ B1A24S15P/B2A24S15P/B2A24S20P Connector Diagram

Motor Controller Hardware Documentation

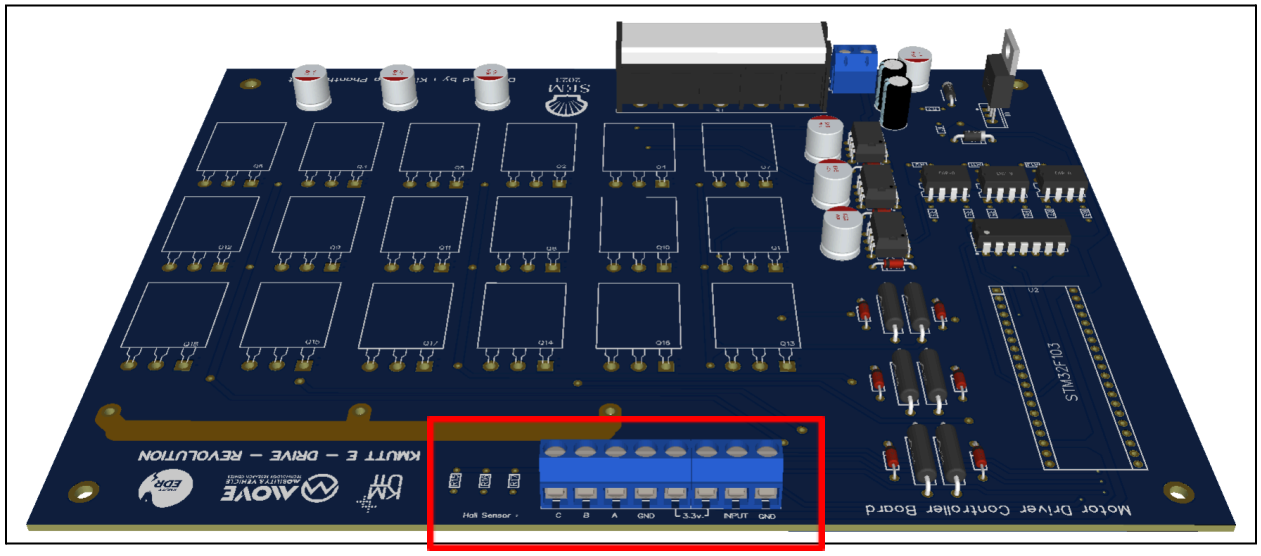
List of Components

ID	Name	Designator	Quantity
1	10uF / 25V	C1	1
2	100UF/63V	C2	1
3	10uF / 25V	C3	1
4	10UF/160V-C-RB-8	C4,C5,C6	3
5	100UF/630V	C7,C8,C9	3
6	1N4007	D1,D2	2
7	1N4148	D3,D4,D5,D6,D7,D8,D9,D10	8
8	1N4148_C406389	D11	1
9	NCE85H21TC	Q1,Q2,Q3,Q4,Q5,Q6 ,Q7,Q8,Q9,Q10,Q11,Q12 ,Q13,Q14,Q15,Q16,Q17,Q18	18
10	240	R1	1
11	390	R2,R3,R4,R5,R6,R7,R8	7
12	1k	R9,R10,R11,R12,R13 ,R14,R15,R16,R17	9
13	120k	R18	1
14	47	R_O1,R_O2,R_O3 ,R_O4,R_O5,R_O6	6
15	KF301-5.0-2P	T1,T4	2
16	KF301-5.0-3P	T2,T3	2
17	HB9500SS-9.5-5P	T5	1
18	LM317	U1	1
19	STM21F103-BluePill	U2	1
20	74HC14	U3	1
21	HCPL-2630	U4,U7,U9	3
22	IR2101	U5,U6,U8	3

Motor Controller Hardware Documentation

Description

Our motor controller board consists of , Power output section , and Power control input section



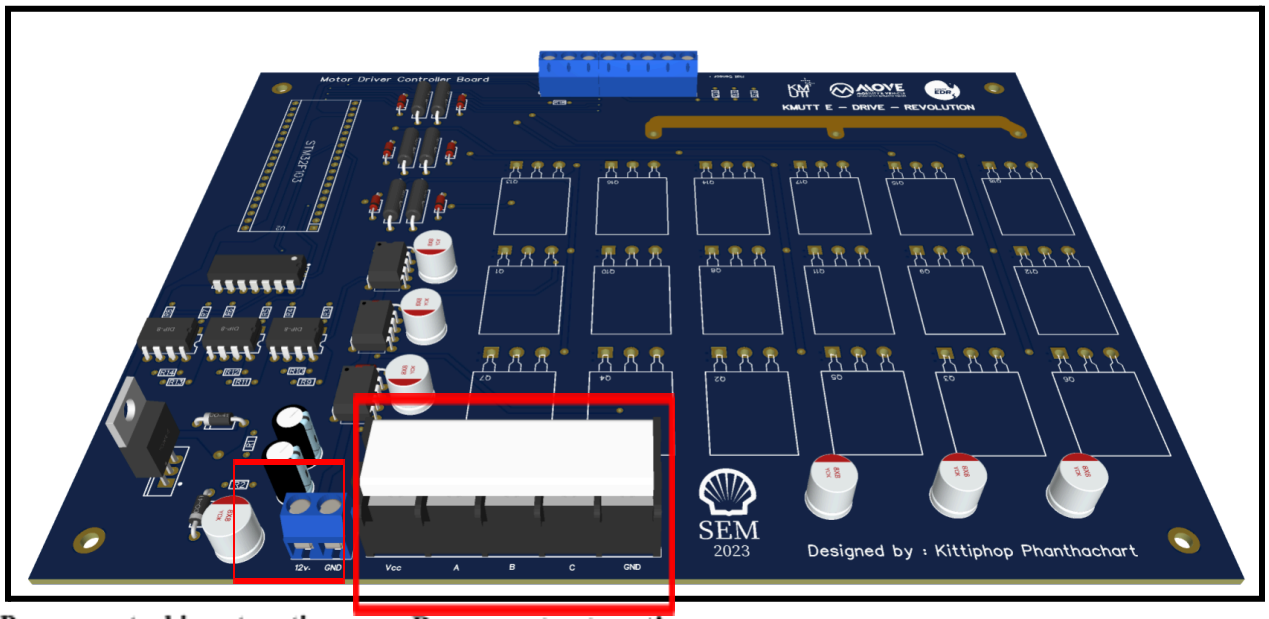
Signal Input section

Signal input section

- **Hall - sensor (Hall_A , Hall_B ,Hall_C)** is sensor for BLDC-motor .the terminal for Hall - sensor has 1k Ohms pull up Resistor for make sure the real signal
- **INPUT** is analog input (0 - 3.3v) to control RPM of BLDC - motor

Motor Controller Hardware Documentation

Description



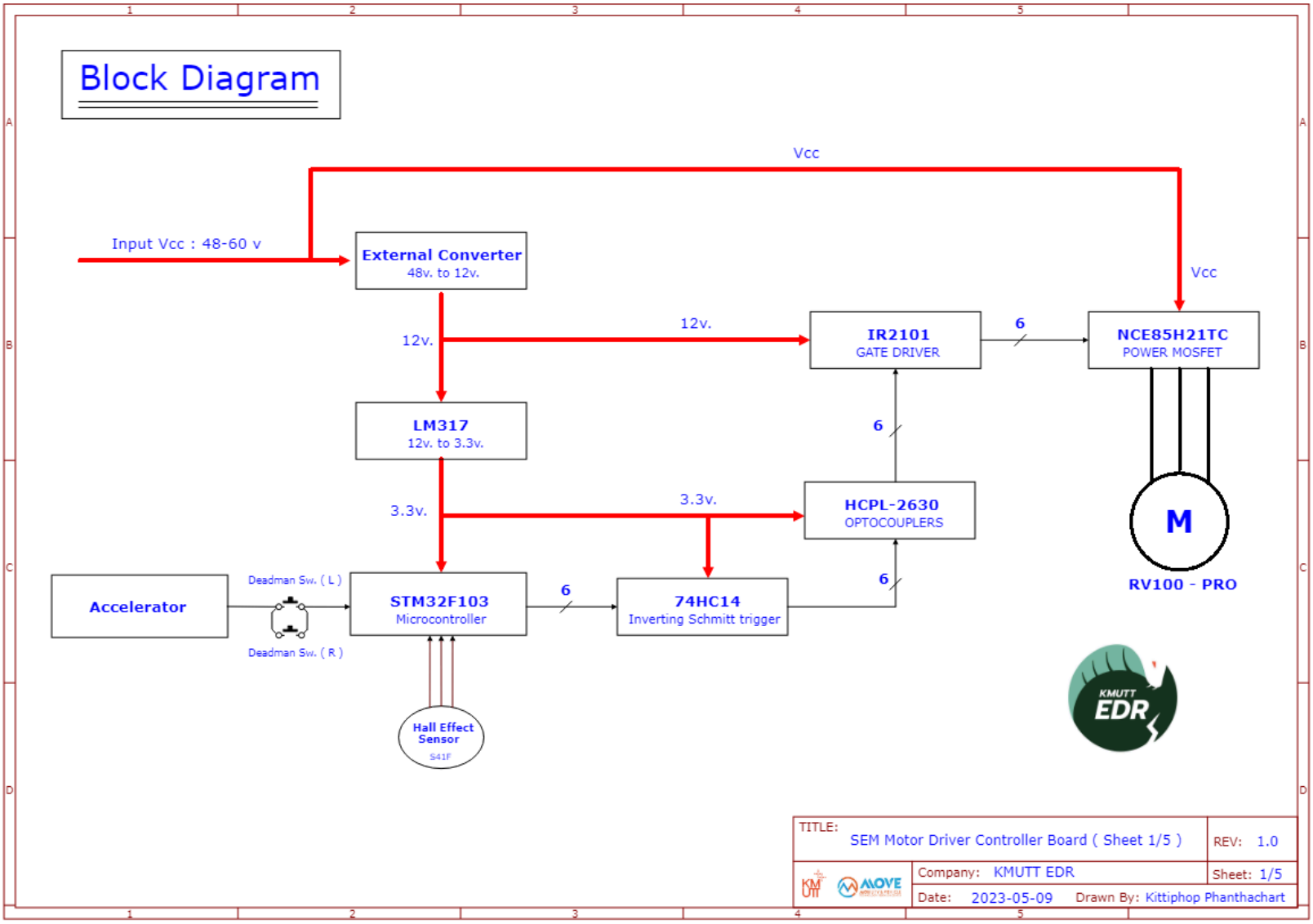
Power control input section Power output section

Power control input section is power 12 volt for control part

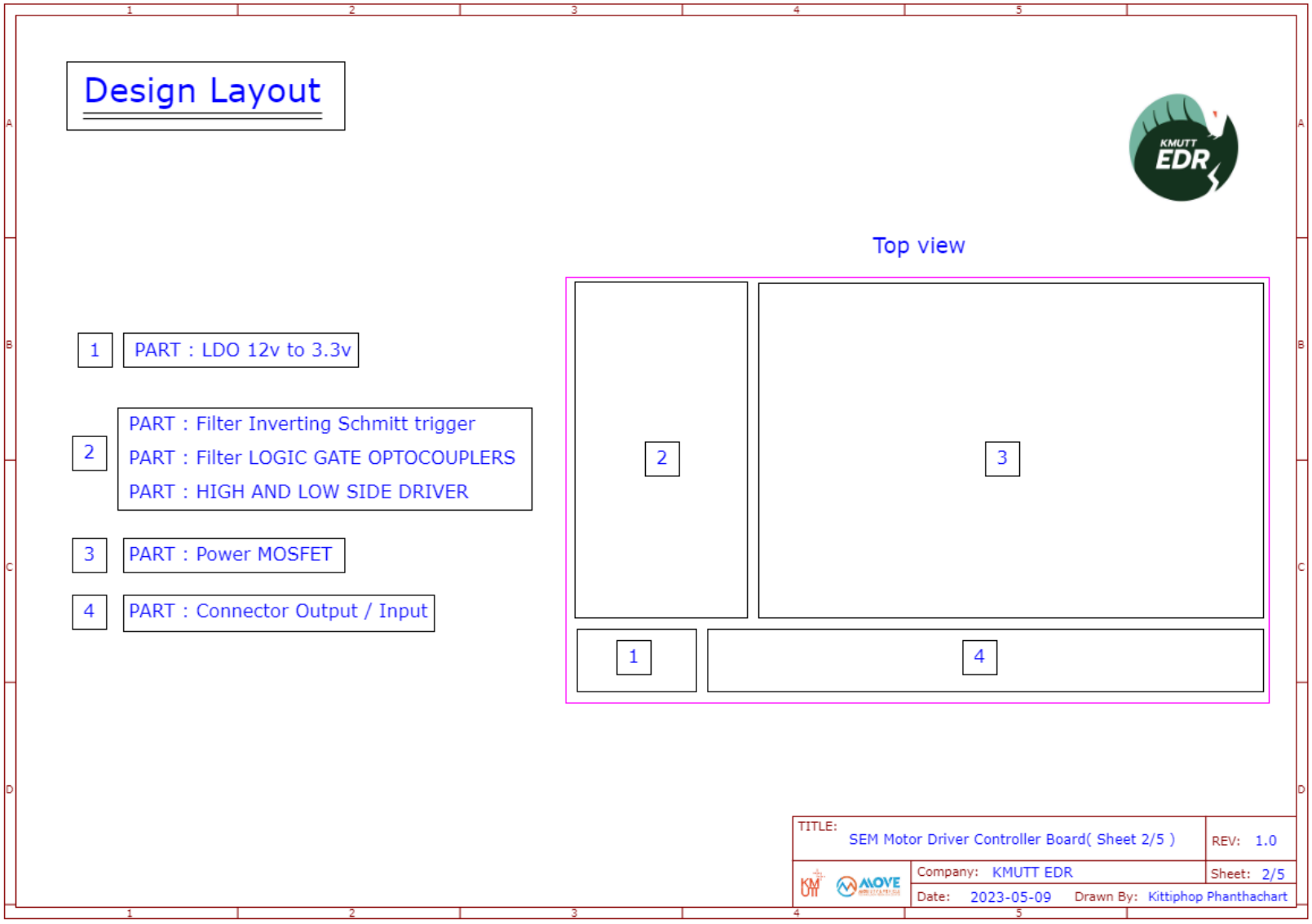
Power output section

- **Vcc and GND** is power to the motor from the Power mosfet on motor controller board can handle voltages up to 85 volts. But on our system we use 48 - 60 volt for competition.
- **Phase motor (A , B , C)** for the main wire to transfer power to BLDC - motor

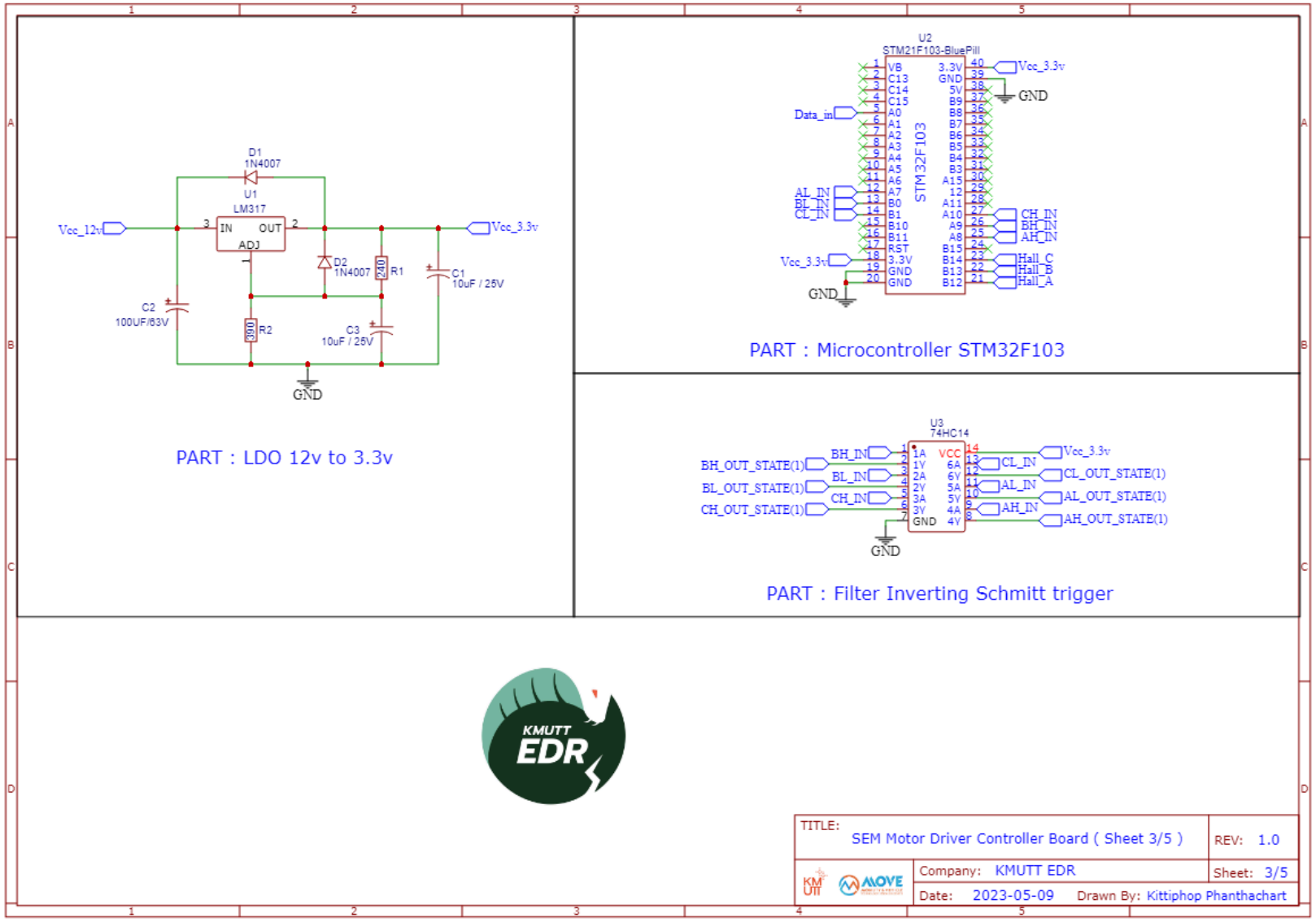
Block Diagram



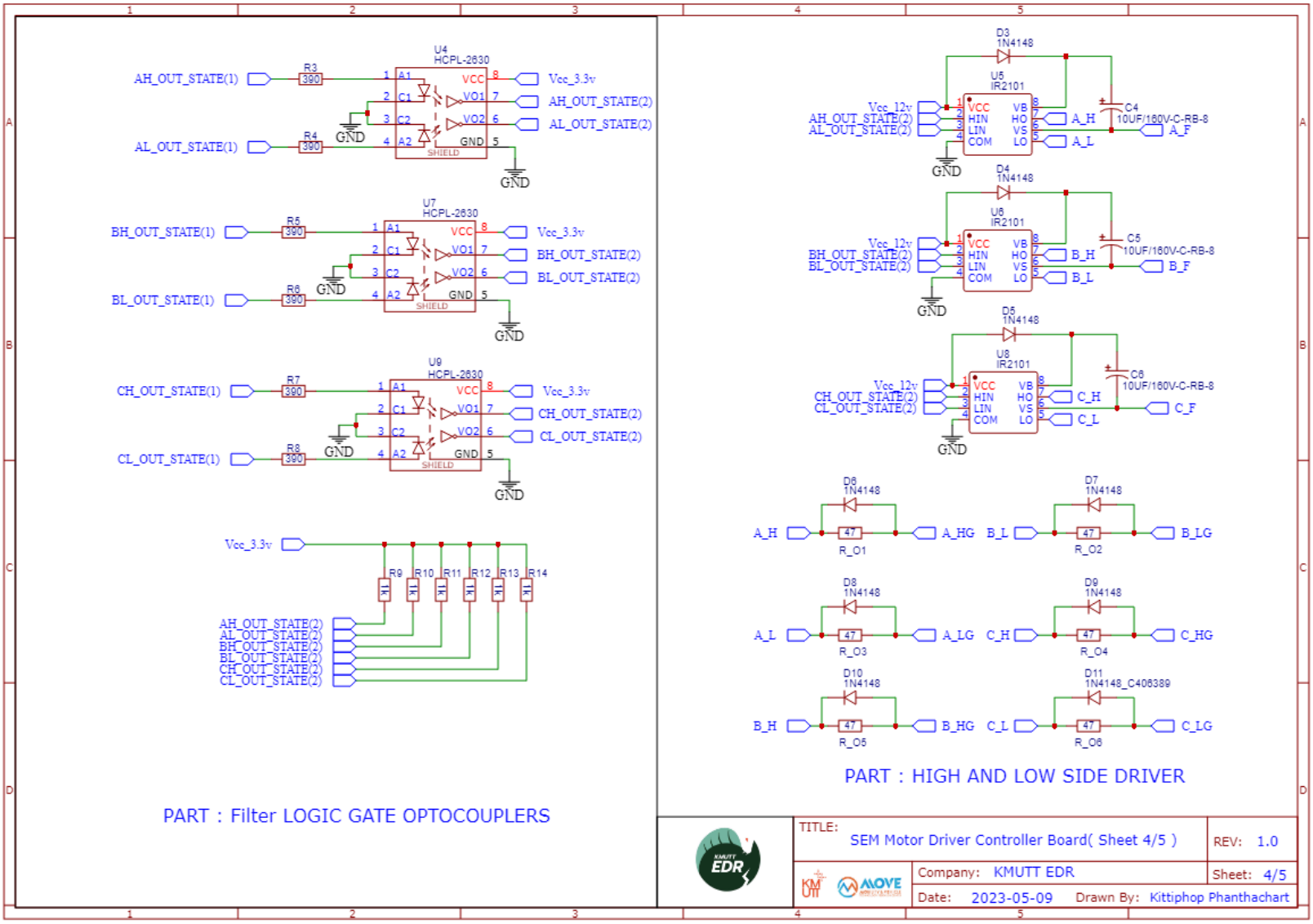
Design Layout



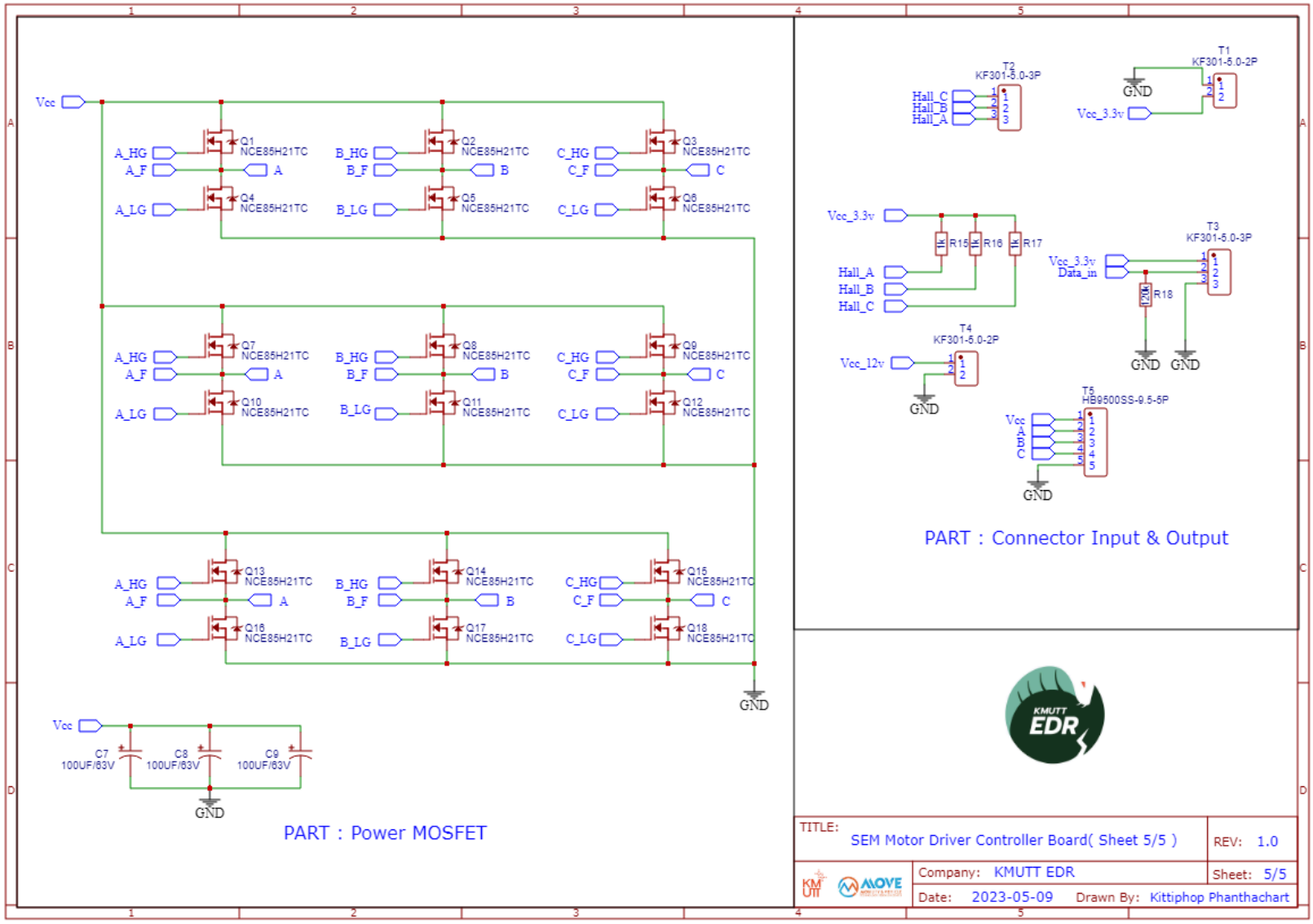
Schematic


D1
1N4007
U1
LM317
IN
OUT
ADJ
Vcc_12v
Vcc_3.3v
D2
1N4007
R1
230
C1
10uF / 25V
C2
100UF/63V
R2
390
C3
10uF / 25V
GND
PART : LDO 12v to 3.3v
U2
STM21F103-BluePill
STM32F103
Data_in
AL_IN
BL_IN
CL_IN
CH_IN
BH_IN
AH_IN
Hall_C
Hall_B
Hall_A
PART : Microcontroller STM32F103
U3
74HC14
BH_OUT_STATE(1)
BL_OUT_STATE(1)
CH_OUT_STATE(1)
CL_OUT_STATE(1)
AL_OUT_STATE(1)
AH_OUT_STATE(1)
PART : Filter Inverting Schmitt trigger
KMUTT EDR
TITLE:
SEM Motor Driver Controller Board (Sheet 3/5)
REV: 1.0
Company: KMUTT EDR
Sheet: 3/5
Date: 2023-05-09
Drawn By: Kittiphop Phanthachart

Schematic

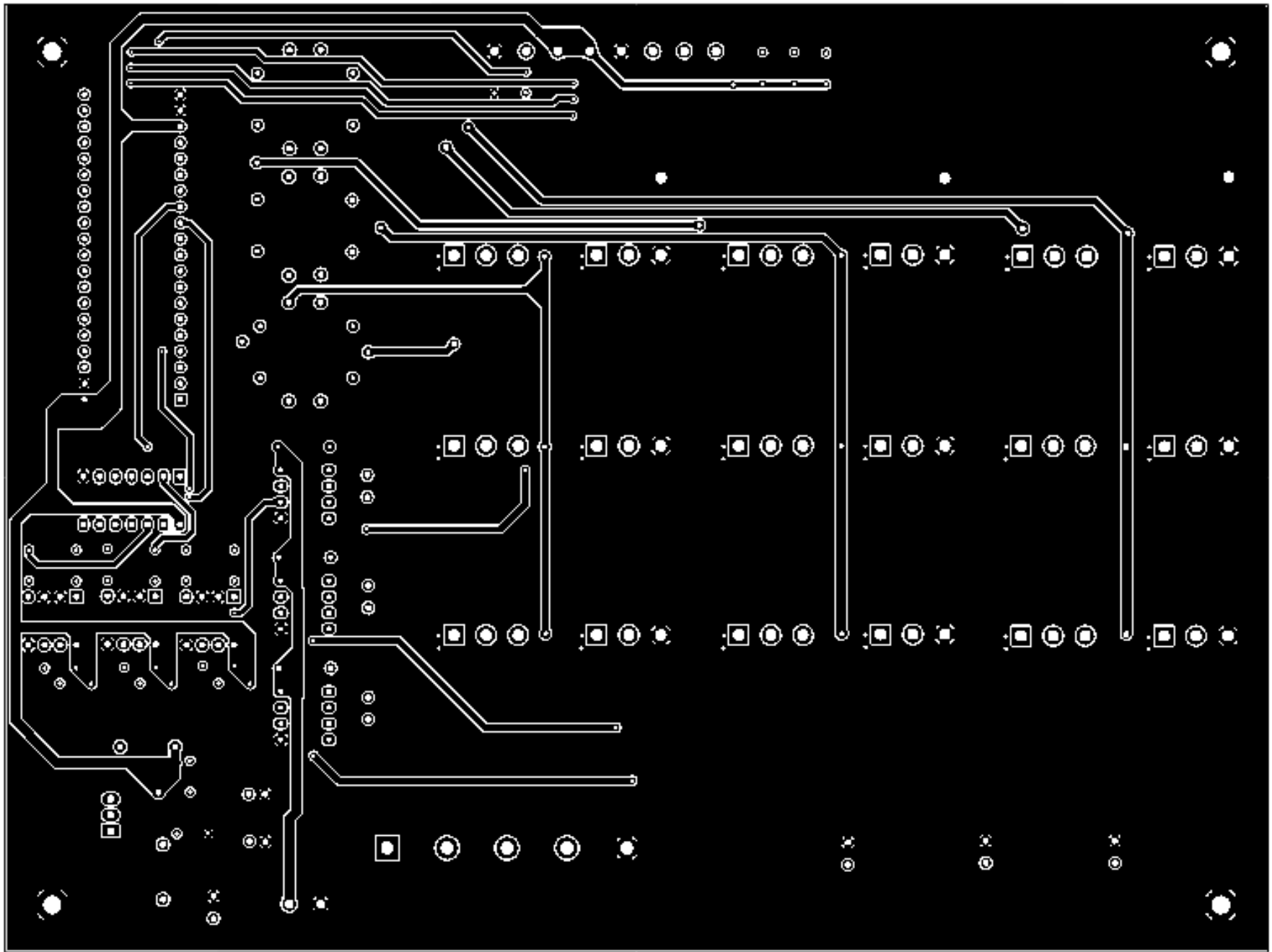

U4
HCPL-2630
AH_OUT_STATE(1)
AL_OUT_STATE(1)
R3
R4
390
GND
Vcc_3.3v
AH_OUT_STATE(2)
AL_OUT_STATE(2)
SHIELD
U7
HCPL-2630
BH_OUT_STATE(1)
BL_OUT_STATE(1)
R5
R6
BH_OUT_STATE(2)
BL_OUT_STATE(2)
U9
HCPL-2630
CH_OUT_STATE(1)
CL_OUT_STATE(1)
R7
R8
CH_OUT_STATE(2)
CL_OUT_STATE(2)
R9
R10
R11
R12
R13
R14
PART : Filter LOGIC GATE OPTOCOUPLERS
D3
1N4148
U5
IR2101
Vcc_12v
A_H
A_L
C4
10UF/160V-C-RB-8
A_F
D4
U6
B_H
B_L
C5
B_F
D5
U8
C_H
C_L
C6
C_F
D6
D7
R_O1
R_O2
A_HG
B_LG
D8
D9
R_O3
R_O4
C_HG
D10
D11
1N4148_C406389
R_O5
R_O6
B_HG
C_LG
PART : HIGH AND LOW SIDE DRIVER
TITLE:
SEM Motor Driver Controller Board(Sheet 4/5)
REV: 1.0
Company: KMUTT EDR
Sheet: 4/5
Date: 2023-05-09
Drawn By: Kittiphop Phanthachart

Schematic


Vcc
A_HG
A_F
A_LG
B_HG
B_F
B_LG
C_HG
C_F
C_LG
Q1 NCE85H21TC
Q2 NCE85H21TC
Q3 NCE85H21TC
Q4 NCE85H21TC
Q5 NCE85H21TC
Q6 NCE85H21TC
Q7 NCE85H21TC
Q8 NCE85H21TC
Q9 NCE85H21TC
Q10 NCE85H21TC
Q11 NCE85H21TC
Q12 NCE85H21TC
Q13 NCE85H21TC
Q14 NCE85H21TC
Q15 NCE85H21TC
Q16 NCE85H21TC
Q17 NCE85H21TC
Q18 NCE85H21TC
A
B
C
GND
C7 100UF/63V
C8 100UF/63V
C9 100UF/63V
PART : Power MOSFET
T1 KF301-5.0-2P
T2 KF301-5.0-3P
T3 KF301-5.0-3P
T4 KF301-5.0-2P
T5 HB9500SS-9.5-5P
Hall_C
Hall_B
Hall_A
Vcc_3.3v
Vcc_12v
Data_in
R15
R16
R17
R18
PART : Connector Input & Output
TITLE:
SEM Motor Driver Controller Board(Sheet 5/5)
REV: 1.0
Company: KMUTT EDR
Sheet: 5/5
Date: 2023-05-09
Drawn By: Kittiphop Phanthachart

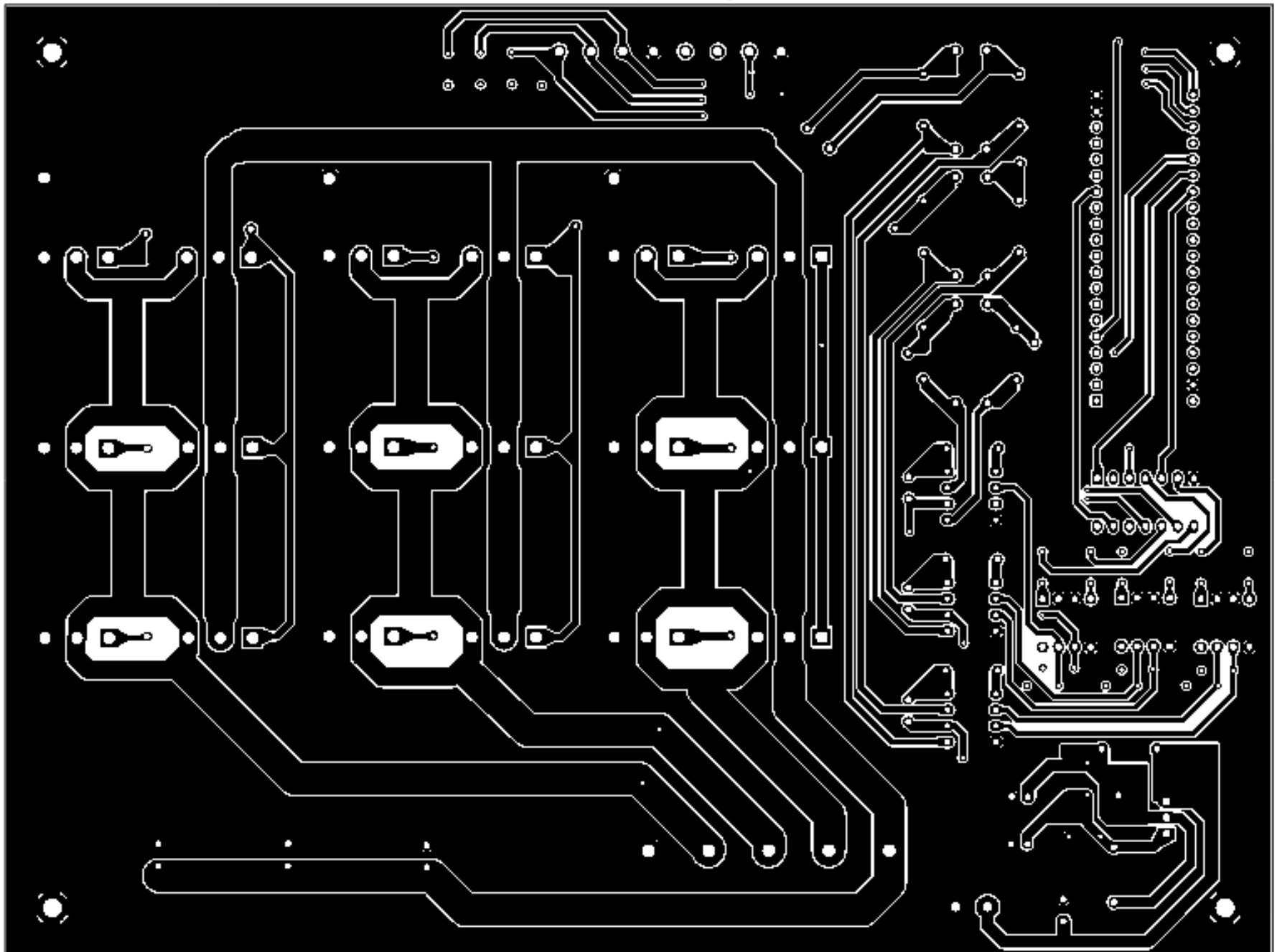
PCB

TopLayer



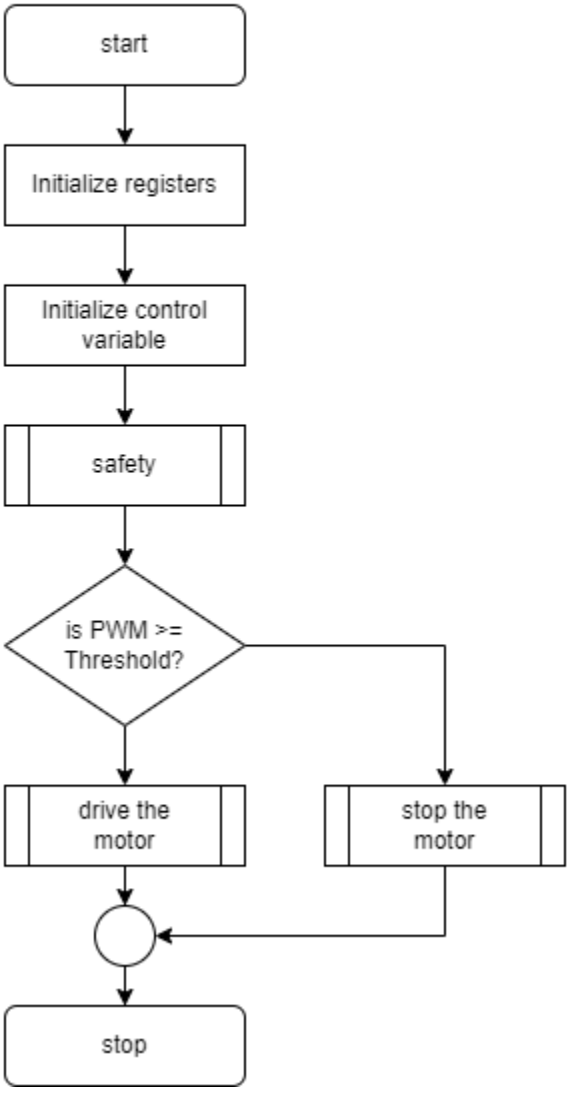
PCB

BottomLayer



Motor Controller Software Documentation

Overall Control Flow Diagram



Our motor control system uses a trapezoidal commutation control algorithm which is a technique that provides high torque and high speed, low switching loss, and also the easiest way to implement.

About Trapezoidal Commutation

Trapezoidal Commutation is a Speed control technique for BLDC motors that uses the duty cycle of PWM to regulate the motor's speed. To achieve maximum motor torque control system need to send the correct pattern to each motor phase at the right moment, by reading three hall-effect sensors state the motor controller will know which pattern of the signal should send to each phase of the BLDC motor.

About Control System

We have developed the software for an STM32F103C8T6 microcontroller to generate the complementary PWM signal to drive the bootstrap MOSFETs driver circuit, the pattern of the signals is determined by the pattern of three hall-effect sensors, and the duty cycle of the PWM is determined by accelerator pedal value. Patterns of hall-effect sensors and Driving signals are described in the table below.

Components	**Phase**	**BLDC Motor Sequence**					
		1	**2**	**3**	**4**	**5**	**6**
BLDC Motor	A	Low	Low	-	High	High	-
	B	High	-	Low	Low	-	High
	C	-	High	High	-	Low	Low
Hall Effect Sensor	Hall A	1	1	1	0	0	0
	Hall B	0	0	1	1	1	0
	Hall C	1	0	0	0	1	1

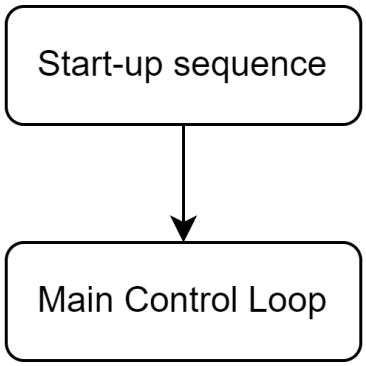
From the table, we defined the symbol "High" as The phase that received a complementary PWM driving signal its duty cycle is determined by the value of an

accelerator pedal, and the symbol “LOW” is defined as the phase that will receive 0-volt driving signal (in this control system the low side MOSFET of that phase is turned on), and the symbol “-” is defined as no connection or both high and low side MOSFETs are turned off.

The microcontroller reads three hall-effect sensors pattern and the accelerator pedal value then it will generate the proper driving pattern and feed it to the driving circuit to drive the motor after applying the proper driving pattern the motor will move to the next sequence and the speed of changing sequence are determined by the duty cycle which is determined by the accelerator pedal.

Motor Controller Software Flow

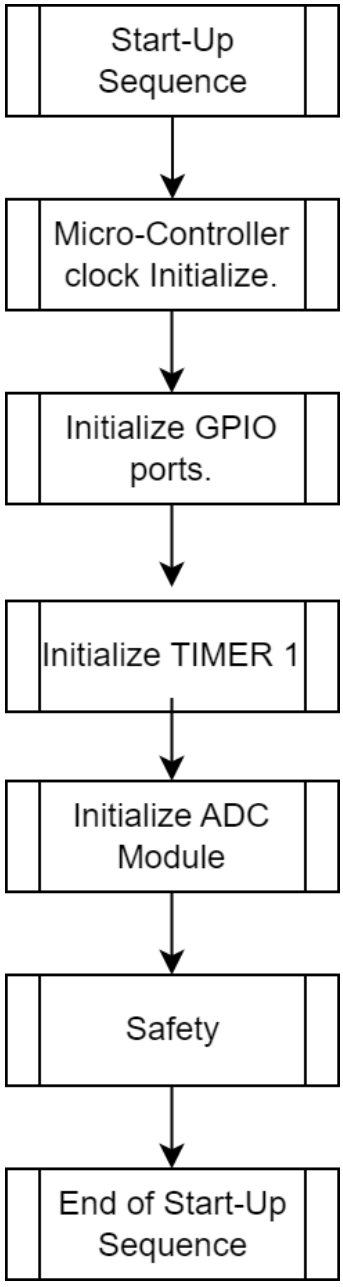
The software used in this motor controller can be divided into two parts which are the “Start-up sequence” and “Main control loop”. When the driver power on the car controller will execute through the Start-up sequence first after finishing the start-up sequence it starts executing code in the main control loop. This can be shown in the flow diagram below.



Flow Diagram Showing 2 main executing sequence

Start-up sequence

The start-up sequence is the set of codes that are used for initializing all necessary peripherals of STM32F103C8T6 for use with the BLDC motor controller application. The flow of the start-up sequence can be shown in the flowchart below.



According to the flow chart, the start-up sequence can be divided into 5 processes which are 1. Clock initialization 2.GPIO Initialization 3. TIMER 1 module initialization 4. Analog to digital converter module Initialization and 5. Safety Function.

1. Clock Initialization.

This is the process of turning on and stabilizing the system clock using an 8 MHz external oscillator combined with a phase lock loop module to multiply the frequency up to 72MHz and then feed this clock signal and pre-scale to the proper speed for each peripheral.

2. GPIO Initialization.

This is the process of initializing all input and output ports that are necessary for the motor controller which can be divided into three group

2.1 Driving Signal output group consists of

- GPIOA8 as Phase A High-side driving signal output.
- GPIOA9 as Phase B High-side driving signal output.
- GPIOA10 as Phase C High-side driving signal output.
- GPIOA7 as Phase A Low-side driving signal output.
- GPIOB0 as Phase B Low-side driving signal output.
- GPIOB1 as Phase C Low-side driving signal output.

2.2 Input Signal group consists of

- GPIOB12 as Hall-Effect sensor A input.
- GPIOB13 as Hall-Effect sensor B input.
- GPIOB14 as Hall-Effect sensor C input.
- GPIOA0 as Accelerator pedal input.

2.3 GPIOC13 as a running indicator.

3. TIMER1 Initialization.

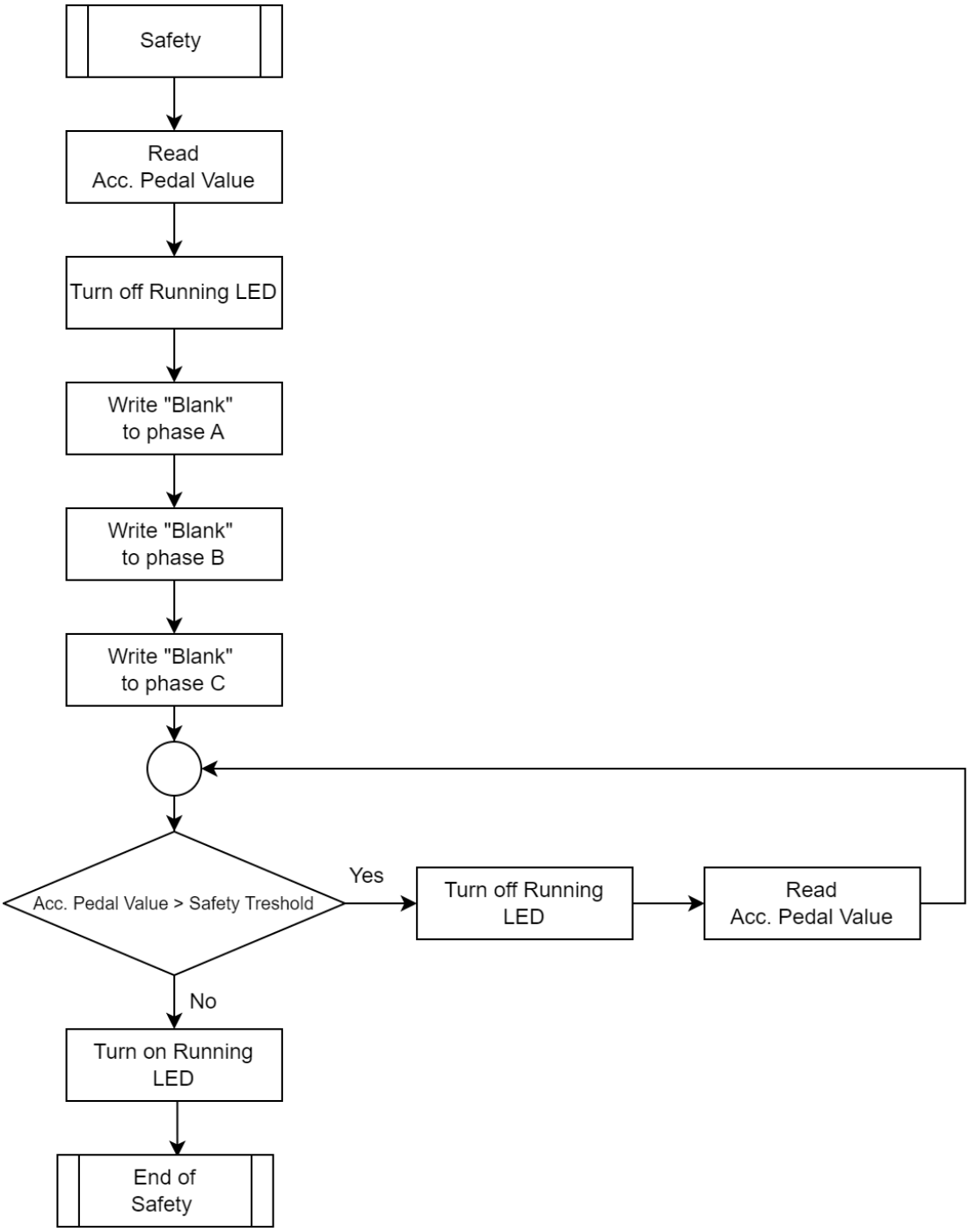
This is the process of initializing and setting up the TIMER1 module of STM32F103C8T6 to make this module generate a complementary PWM driving signal to drive the bootstrap gate driver circuit at 23KHz to eliminate all audible noise from driving circuitry. The PWM has a 9-bit resolution (0-511) and from the characteristic of bootstrapping, we can go up to 80% of the PWM duty cycle.

4. Analog to digital converter module Initialization.

This is the process of initializing the analog to digital converter module of the microcontroller in this application we have set this module to work in continuous mode to achieve continuity and fast response to changing levels of analog signal from the accelerator pedal.

5. Safety Function.

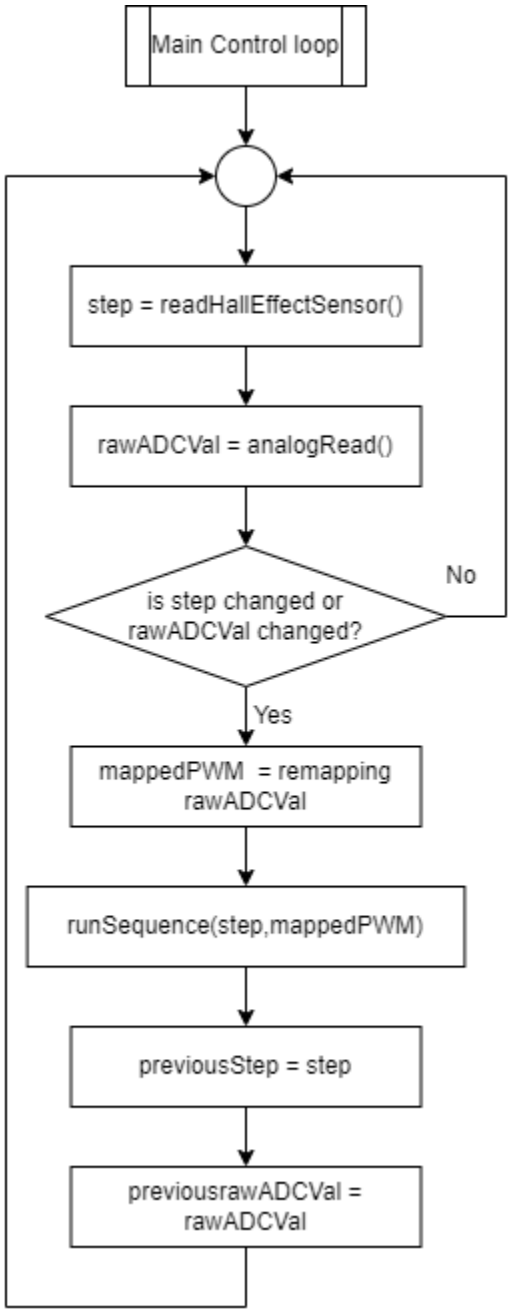
This is the function that designs to prevent the driver from accidentally accelerating the car while starting up. The working process of this function can be described in the following flowchart.



While this function is executing it receives a safety threshold as a value between 0-511 (in this case is set at 30) then it will read the accelerator pedal value and turn off the running LED then the controller sends a blank signal to all three motor phases (turn off both low-side and high-side MOSFETs of each phase) if the pedal value is greater than the safety threshold (pedal value > safety threshold) controller program will stuck in a loop that constantly read pedal value and turn off the running LED, and after the controller can read the pedal value that are equal or less than the safety threshold it will turn the running LED back on, return from safety function and start executing main control loop to drive the motor.

Main-Control Loop

Main-control loop is the control loop that uses for driving the motor, the process in the main-control loop can be written as the following flowchart.



First, the main control loop gets the motor sequence by reading three motor hall-effect sensors using the function readHallEffectSensor(); this function will return numbers 1 to 6 according to the pattern that reads from sensors and store the value in “step”.

Second, read the accelerator pedal using the analogRead() function and store the raw analogRead value in the rawADCValue variable.

After reading these two values the microcontroller will check that is it necessary to call the process that generates the driving signal because the microcontroller runs at a very high clock speed which means both analogRead and read readHallEffectSensor will read a lot of the same values and if we keep calling the function to write the same value it will resulting in the slow running control loop. So if both step and rawADCvalue are the same the microcontroller will skip calling the driving signal generation function and restart the control loop.

But, If at least one of these two values is not the same the microcontroller will remap the rawADCVal to the proper 9-bit value for driving signal generation function and then call the runSequence function that takes step and mappedPWM as an inputs parameter to write the proper driving signal to drive the motors and restart the main control loop.

All of the sub-functions that we mention in this topic will be explained in the next topic.

readHallEffectSensor function

This function is used for getting the motor sequence by reading three hall-effect sensors Hall-A, Hall-B, and Hall-C connected to the GPIOB12, GPIOB13, and GPIOB14 respectively. Since they are connected in the series of GPIO that means when the microcontroller read the pattern we can handle these patterns as a BCD (Binary coded decimal) as shown in the table below

Motor Seq.	Hall-C (GPIOB14)	Hall-B (GPIOB13)	Hall-A (GPIOB12)	BCD Decoded
1	1	0	1	5
2	0	0	1	1
3	0	1	1	3
4	0	1	0	2
5	1	1	0	6
6	1	0	0	4

When the microcontroller reads Hall-C, Hall-B, and Hall-A values these bits' states will save to the GPIOB_IDR register at bits 14,13, and 12 respectively, for example at motor sequence 1 Hall-C, Hall-B, and Hall-A have 1,0, and 1 state respectively if we do GPIOB read operation this following value will be saved to GPIOB_IDR register.

GPIOB_IDR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR 15	IDR 14	IDR 13	IDR 12	IDR 11	IDR 10	IDR 9	IDR 8	IDR 7	IDR 6	IDR 5	IDR 4	IDR 3	IDR 2	IDR 1	IDR 0
x	1	0	1	x	x	x	x	x	x	x	x	x	x	x	x

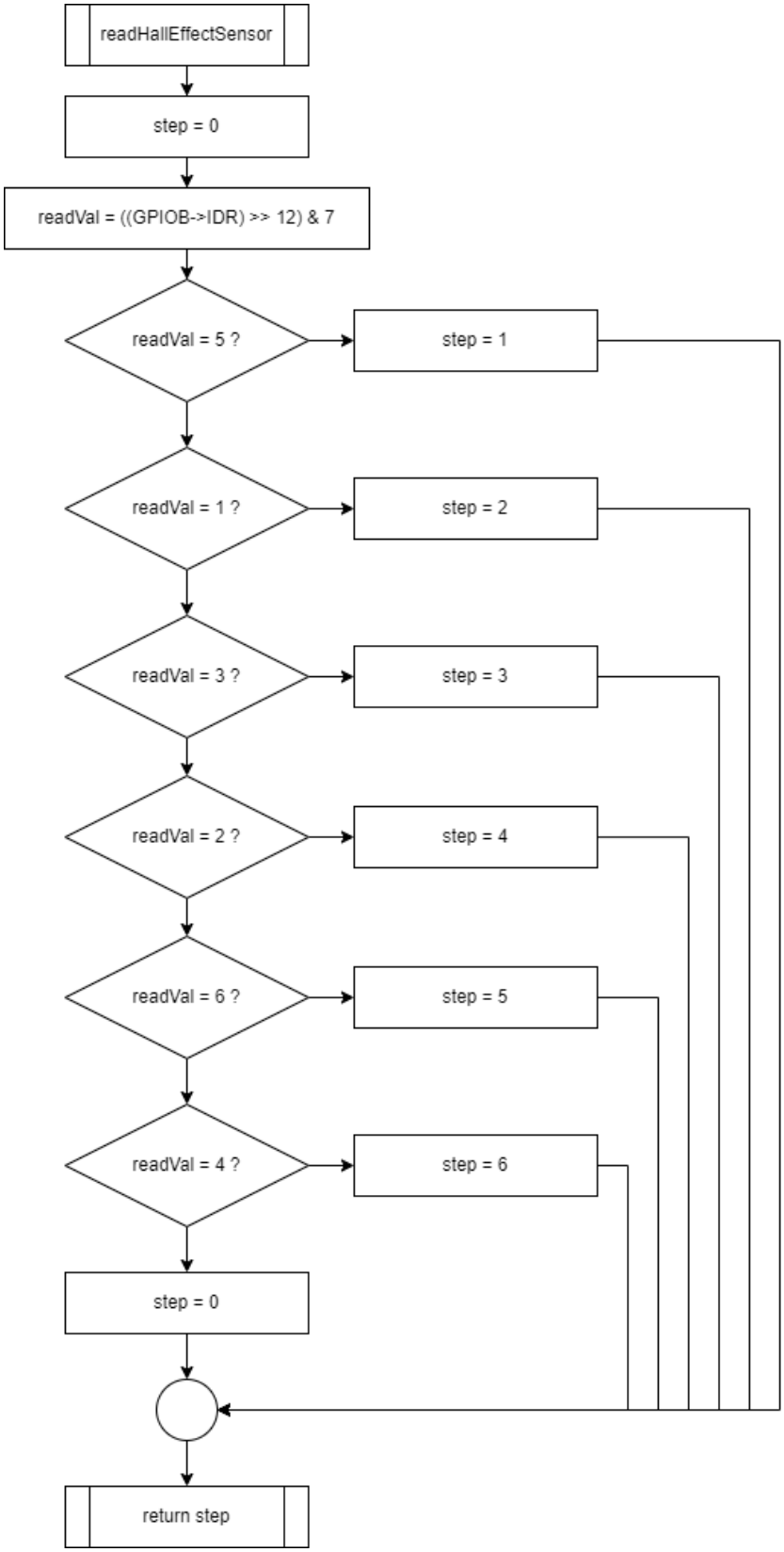
1 and 0 are bit states and ‘x’ is don’t care bits or undefined states when the software read the value from the GPIOB_IDR register software will get 0bx101xxxxxxxxxxx to convert to the proper BCD we need to do a 12-bit left-shift operation and then do AND bitwise operation with 7.

Example: Read value 0bX101XXXXXXXXXXX from GPIOB_IDR

GPIOB_IDR: 0bx101xxxxxxxxxxx
GPIOB_IDR >>12: 0bxxxxxxxxxxxx101

(GPIOB_IDR >>12)&7: 0bxxxxxxxxxxxx101 &
0b0000000000000111
Hall sensor pattern value: 0b0000000000000101 (5)

Save this value to the “readVal” variable and use a switch-case to determine the motor sequence and return that sequence value from this function if the pattern is not valid this function will return 0, these operations can be written as this following flowchart.

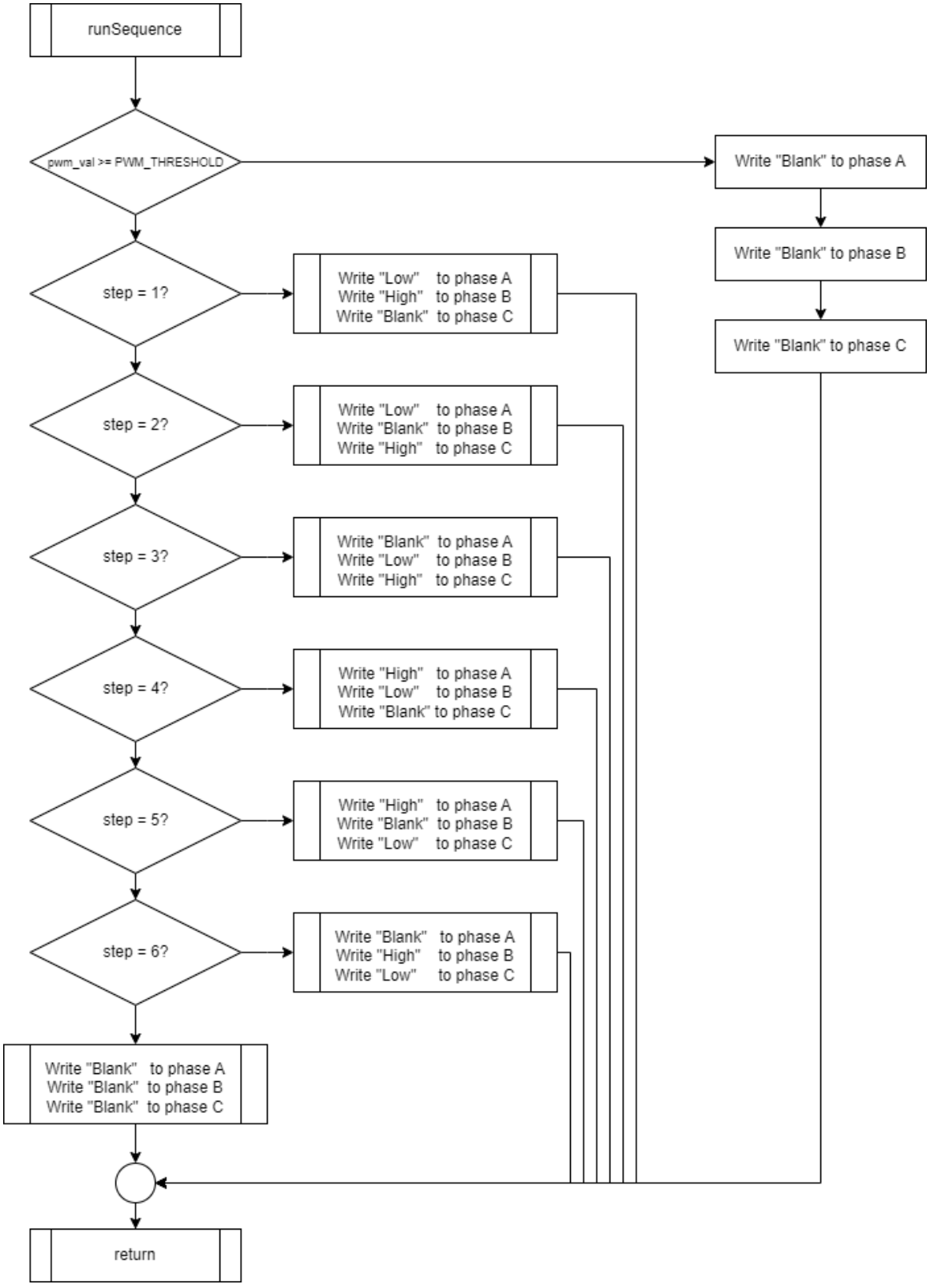


Flowchart of readHallEffectSensor function

RunSequence function

This function is used for generating all driving signals. It takes two input parameters which are “step” to determine the motor sequence to generate the proper signal and “speed” to determine the duty cycle of the PWM that regulates the motor speed.

The first thing that this function does is compare the speed input to the minimum PWM threshold if the value is less than the threshold microcontroller will send a no-connection state to all motor phases if the speed value is equal to or greater than the threshold microcontroller will generate the driving signal according to the step value if the step that sends to function is not the valid step microcontroller will send a no-connection state to all motor phases. These operations can be written as the following flowchart.



Flowchart of RunSequence function

Function for “write PWM”

As we explained in the “About Control System” topic the driving signal has three states which are “High”, “Low” and No-connection or “Blank” To achieve these three states of signal we have developed two sets of functions to make our microcontroller generate the proper driving signal which is the phase(x)Out, and (x)Low(), when (x) is A, B, and C.

phase(x)Out function group

This group of functions consists of phaseAOut, phaseBOut, and phaseCOut these three functions take one input parameter which is the PWM duty cycle value(void phase(x)Out (unsigned short _PWM);).

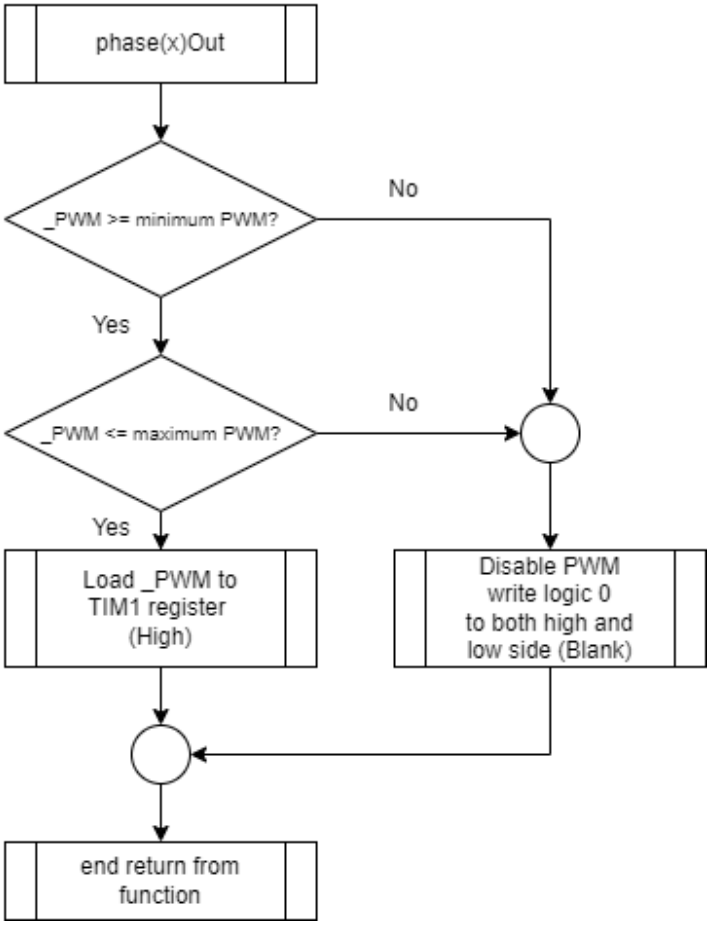
When the input parameter is greater or equal to the minimum threshold and less or equal to the maximum duty cycle limit function will load the input value to the appropriate register to make the microcontroller generate PWM, But if the input parameter is less than the minimum threshold this function will disable both high and low side of that channel this will result in the microcontroller to generate “Blank” state signal. These processes can be written in the following flowchart.

phase(x)Out(unsigned short _PWM)

phaseAOut(unsigned short _PWM);

phaseBOut(unsigned short _PWM);

phaseCOut(unsigned short _PWM);



Flowchart of phase(x)Out() function

(x)Low() function group

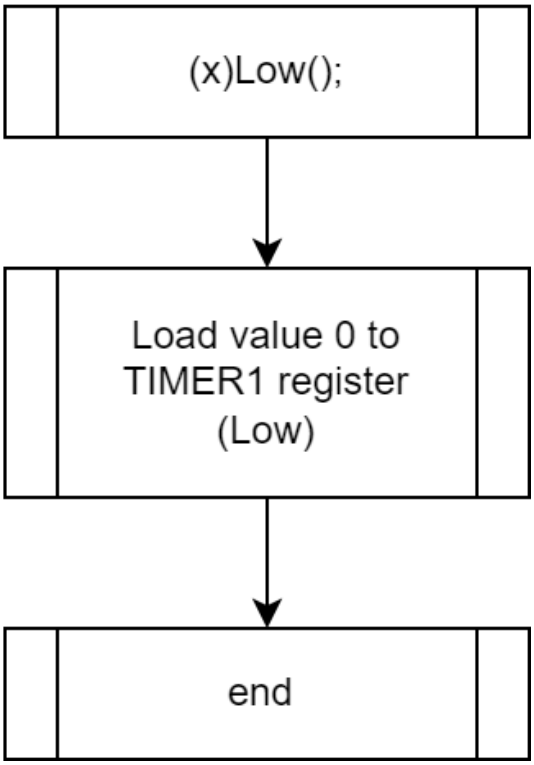
This group of functions consists of ALow();, BLow();, and CLow(); when this group of functions is called program will load value 0 to the appropriate register since it is a “Complementary PWM” This will result in high output stayed at logic level 0 and low output (or complementary output) will stay at logic level 1 and this will turn on low-side MOSFETs and this will result in sending “Low” driving signal to motor phase. This function can be written in the following flowchart.

(x)Low();

ALow();

BLow();

CLow();



Read and Map Value from ADC module

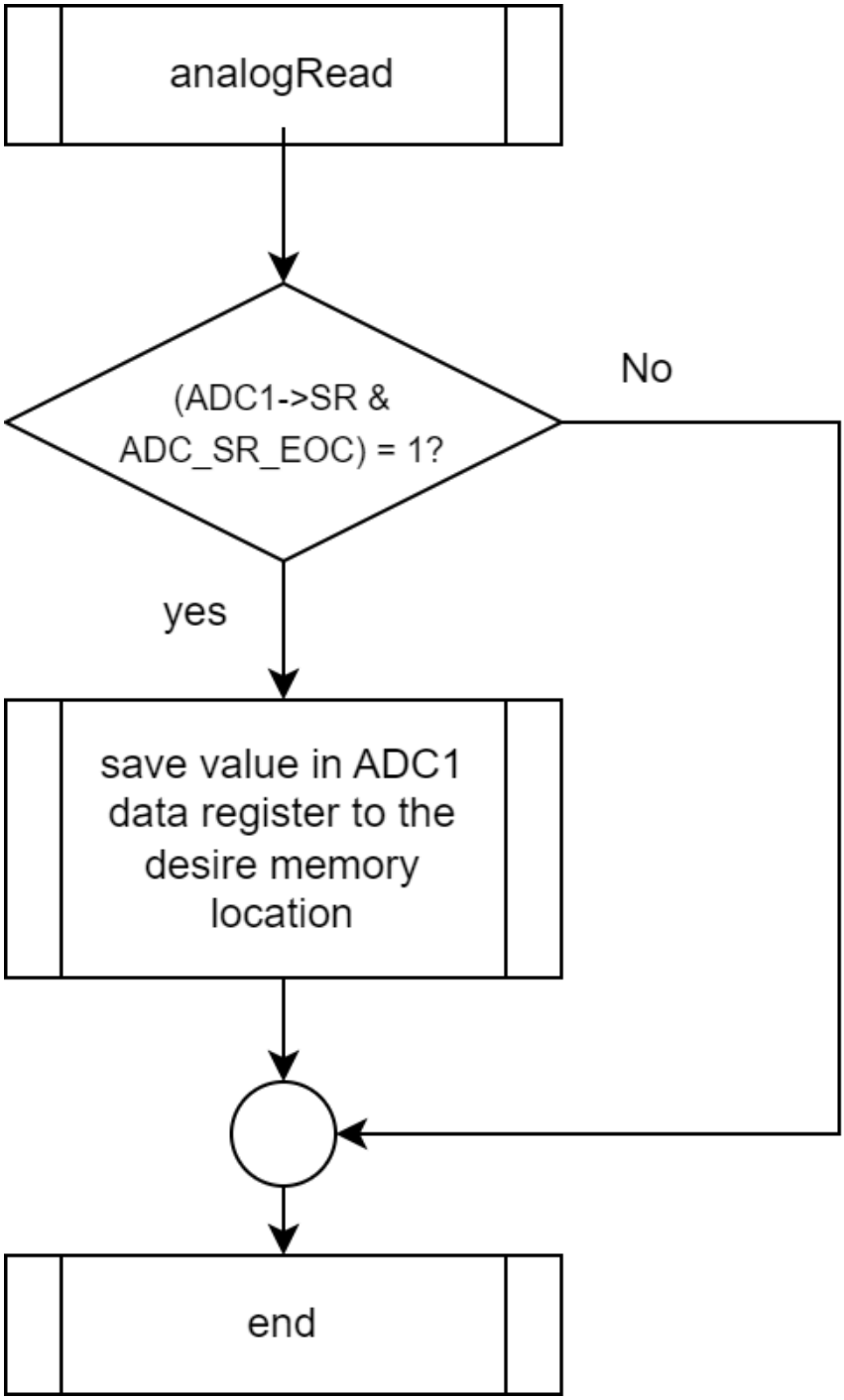
As we mentioned in the earlier topics this motor controller uses the ADC module in continuous conversion mode to take the converted value from the ADC data register we use the hardware EOC flag (End Of Conversion flag) If the ADC module finished the conversion, the microcontroller will automatically set the EOC flag (change from 0 to 1). After the data have been read from the data register microcontroller hardware will automatically clear the EOC flag and start the next conversion.

This software uses the analogRead function to read the value from the ADC1 data register. This function expects the pointer to an integer that points to the location that wants to store the data as an input parameter.

When the analogRead function is called the first operation that has to be performed is to check the EOC flag because if the read operation is performed during the conversion it has a chance to get a corrupted conversion value.

If the EOC flag was set, the analogRead function will read the value in the ADC1 data register and store the location that the function received.

These processes can be written as the following flow chart.



Flowchart of the analogRead function

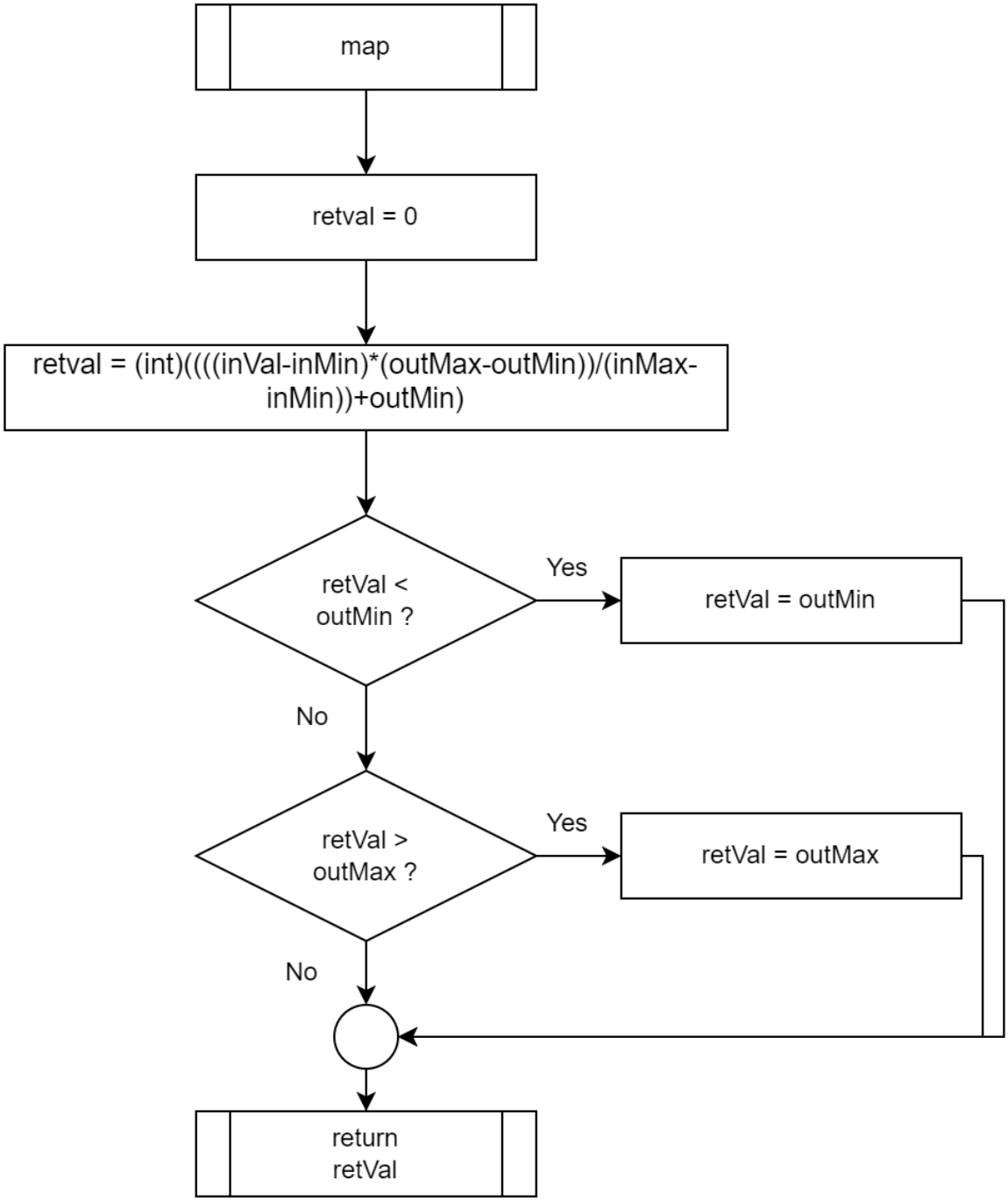
After the software have been loaded the value from the ADC1 data register we call this value a “Raw ADC Value ” This value is a 12-bit integer because this microcontroller has a 12-bit resolution ADC module which makes remapping value range procedure needed.

This software uses the map function to remap the 12-bit range value to the desired 9-bit range with upper and lower bounds. The map function takes 5 input parameters which are raw value(inVal), minimum raw value(inMin), maximum raw value(inMax), minimum mapped value(outMin), and maximum mapped value(outMax).

The map function will take all input parameters and put them into this equation.

mapped Value = (int)((((inVal-inMin)*(outMax-outMin))/(inMax-inMin))+outMin)

But this equation can result in a negative value or the value over the upper desired bound. So, before returning the mapped value from the function software checks the computed value if the value is less than the lower-output bound then the software set the return value to the lower-output bound (outMin) but if the computed value is greater than the upper-output bound the software set the return value to the lower-output bound (outMax). The map function can be written as the following flowchart.



Flowchart of the map function

Motor Controller Software Code

```
/*
	KMUTT EDR EPSILON FIRMWARE VERSION 3_2
	DESIGNED FOR STM32F103 BASE BLDC MOTOR DRIVER BOARD
*/
#include "stm32f10x.h"

//#define DEBUG

#ifdef DEBUG
	#include "stm32f1uart.h"
#endif

// 9 bit PWM generator module value
// Driver board will send the signal to drive the motor when the pwmIn > PWM_THRESHOLD
#define PWM_LIMIT 408
#define PWM_THRESHOLD 5

//The safety function is designed to prevent the driver from accidentally accelerate the car while starting the car
#define SAFETY_THRESHOLD 30

//Pedal Constant : 12 bit ADC value
#define PEDAL_IDLE	1250
//#define PEDAL_IDLE	0
#define PEDAL_MAX		3363


void phaseAOut (unsigned short _PWM);
void phaseBOut (unsigned short _PWM);
void phaseCOut (unsigned short _PWM);
void ALow (void);
void BLow (void);
void CLow (void);

int readHallEffectSensor (void);
int map(float inVal, float inMin, float inMax, float outMin, float outMax);

void runSequence (int stp, unsigned short pwm_val);
void analogRead(int *retVal);
void safety(int threshold);

unsigned long val = 0;

int main (void){
	int step = 0, previousStep = 0, stepOut = 0,rawADCVal = 0,previousRawADCVal = 0;

	while(!((RCC->CR & RCC_CR_HSIRDY) >> 1));
	RCC->CR |= RCC_CR_HSEON;
	while(!((RCC->CR & RCC_CR_HSERDY) >> 1));
	RCC->CR |= RCC_CR_PLLON;
```

```
	while(!((RCC->CR & RCC_CR_PLLRDY) >> 1));
	RCC->CR |= RCC_CR_CSSON;

	RCC->CFGR |= RCC_CFGR_PLLSRC;
	RCC->CFGR |= (RCC_CFGR_PLLMULL_0 | RCC_CFGR_PLLMULL_1 | RCC_CFGR_PLLMULL_2);
	RCC->CFGR |= RCC_CFGR_PPRE1_2      ;
	RCC->CFGR |= RCC_CFGR_SW_1;

	RCC->CFGR &= (~(RCC_CFGR_PLLMULL_3 | RCC_CFGR_PLLXTPRE | RCC_CFGR_SW_0 |
RCC_CFGR_HPRE | RCC_CFGR_PPRE1_0 | RCC_CFGR_PPRE1_1 | RCC_CFGR_PPRE2));
	RCC->CFGR |= RCC_CFGR_ADCPRE_DIV6;

	//Enable clock for various module

	RCC->APB2ENR |= (RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN |
RCC_APB2ENR_IOPCEN);
	RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
	RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;
	RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;


	// Alternative Function pin remapping
	// and do partial pins remapping fot TIM1 module
	AFIO->MAPR |= (AFIO_MAPR_TIM1_REMAP_0);
	AFIO->MAPR &= (~(AFIO_MAPR_TIM1_REMAP_1));


/*
	Configure GPIOA8 GPIOA9 GPIOA10 as the high-side-output motor driving signals by using
	GPIOA8  >> as phase A high output
	GPIOA9  >> as phase B high output
	GPIOA10 >> as phase C high output
	*/
	GPIOA->CRH |= (GPIO_CRH_MODE8 | GPIO_CRH_CNF8_1);
	GPIOA->CRH &= ~(GPIO_CRH_CNF8_0);
	GPIOA->CRH |= (GPIO_CRH_MODE9 | GPIO_CRH_CNF9_1);
	GPIOA->CRH &= ~(GPIO_CRH_CNF9_0);
	GPIOA->CRH |= (GPIO_CRH_MODE10 | GPIO_CRH_CNF10_1);
	GPIOA->CRH &= ~(GPIO_CRH_CNF10_0);


	// Configure GPIOB12 GPIOB13 GPIOB14 as input for reading hall-efffect sensor signals.
	GPIOB->CRH |= GPIO_CRH_CNF12_0;
	GPIOB->CRH &= ~(GPIO_CRH_MODE12 | GPIO_CRH_CNF12_1);
	GPIOB->CRH |= GPIO_CRH_CNF13_0;
	GPIOB->CRH &= ~(GPIO_CRH_MODE13 | GPIO_CRH_CNF13_1);
	GPIOB->CRH |= GPIO_CRH_CNF14_0;
	GPIOB->CRH &= ~(GPIO_CRH_MODE14 | GPIO_CRH_CNF14_1);
```

```
/*
        Configure GPIOA7 GPIOB0 GPIOB1 as the low-side-output motor driving signals by using
        GPIOA7  >> as phase A low output
        GPIOB0  >> as phase B low output
        GPIOB1  >> as phase C low output
        */
        GPIOA->CRL |= (GPIO_CRL_MODE7 | GPIO_CRL_CNF7_1);
        GPIOA->CRL &= ~(GPIO_CRL_CNF7_0);
        GPIOB->CRL |= (GPIO_CRL_MODE0 | GPIO_CRL_CNF0_1);
        GPIOB->CRL &= ~GPIO_CRL_CNF0_0;
        GPIOB->CRL |= (GPIO_CRL_MODE1 | GPIO_CRL_CNF1_1);
        GPIOB->CRL &= ~GPIO_CRL_CNF1_0;


// Configure GPIOC13 as running indicator (GPIOC13 is connected to built in led on STM32 bluepill)
        GPIOC->CRH &= ~GPIO_CRH_CNF13;
        GPIOC->CRH |= GPIO_CRH_MODE13;
        GPIOC->BSRR = GPIO_BSRR_BS13;




        // Prepare timer1 module for pwm mode
        TIM1->PSC = 2;
        TIM1->ARR = 511;
        TIM1->CR1 |= (TIM_CR1_ARPE | TIM_CR1_CMS_0 | TIM_CR1_CEN);
        TIM1->CR1 &= (uint32_t)(~(TIM_CR1_CKD | TIM_CR1_CMS_1 | TIM_CR1_DIR));
        TIM1->EGR |= (TIM_EGR_UG);
        TIM1->CCMR1 |= (TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1PE);
        TIM1->CCMR1 &= (~(TIM_CCMR1_OC1M_0 | TIM_CCMR1_CC1S));
        TIM1->CCMR1 |= (TIM_CCMR1_OC2M_2 | TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2PE);
        TIM1->CCMR1 &= (~(TIM_CCMR1_OC2M_0 | TIM_CCMR1_CC2S));
        TIM1->CCMR2 |= (TIM_CCMR2_OC3M_2 | TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3PE);
        TIM1->CCMR2 &= (~(TIM_CCMR2_OC3M_0 | TIM_CCMR2_CC3S));

        TIM1->BDTR |= (TIM_BDTR_MOE | TIM_BDTR_AOE );
        TIM1->BDTR &= (~(TIM_BDTR_OSSR));

        TIM1->BDTR &= (~(TIM_BDTR_DTG_7));
        TIM1->BDTR |= (TIM_BDTR_DTG_0 | TIM_BDTR_DTG_2 | TIM_BDTR_DTG_3
|TIM_BDTR_DTG_4 | TIM_BDTR_DTG_6 );

        TIM1->CCER &= (~(TIM_CCER_CC1P | TIM_CCER_CC1NP | TIM_CCER_CC2P |
TIM_CCER_CC2NP | TIM_CCER_CC3P | TIM_CCER_CC3NP));

        TIM1->CCER |= (TIM_CCER_CC2E | TIM_CCER_CC2NE);
        TIM1->CCER |= (TIM_CCER_CC3E | TIM_CCER_CC3NE);
```

```
	//Init ADC
	GPIOA->CRL &= ~(GPIO_CRL_CNF0_0 | GPIO_CRL_MODE0);
	ADC1->SMPR2 |= (ADC_SMPR2_SMP0);
	ADC1->SQR1 &= ~(ADC_SQR1_L);
	ADC1->SQR3 &= ~(ADC_SQR3_SQ1);
	ADC1->CR2 &= ~(ADC_CR2_ALIGN);
	ADC1->CR2 |= (ADC_CR2_ADON | ADC_CR2_CONT);

	for(int i = 0; i < 150000; i++);
	ADC1->CR2 |= (ADC_CR2_ADON);
	ADC1->CR2 |= (ADC_CR2_CAL);
	while(ADC1->CR2 & ADC_CR2_CAL);
	GPIOC->BSRR = (GPIO_BSRR_BR13);
	for(int i = 0; i < 150000; i++);

	#ifdef DEBUG
		seriaiInit(72000000UL,9600UL);
	#endif

	int mappedPWM = 0;

	safety(SAFETY_THRESHOLD);
	GPIOC->BSRR = (GPIO_BSRR_BR13);
	while(1){
		#ifdef DEBUG
			serialprintf("%d step > %d\n",mappedPWM,step);
		#endif
		step = readHallEffectSensor();
		analogRead(&rawADCVal);
		if(rawADCVal != previousRawADCVal || step != previousStep ){
			mappedPWM = map(rawADCVal,PEDAL_IDLE,PEDAL_MAX,0,PWM_LIMIT);
			runSequence(step,mappedPWM);
			previousStep = step;
			previousRawADCVal = rawADCVal;
		}
	}
}


void phaseAOut(unsigned short _PWM){
	if(_PWM >= PWM_THRESHOLD && _PWM <= PWM_LIMIT){
			TIM1->CCER |= (TIM_CCER_CC1E | TIM_CCER_CC1NE);
			TIM1->CCR1 = _PWM;
	}
	else
		TIM1->CCER &= ~(TIM_CCER_CC1E | TIM_CCER_CC1NE);
```

```
}

void phaseBOut(unsigned short _PWM){
	if(_PWM >= PWM_THRESHOLD && _PWM <= PWM_LIMIT){
			TIM1->CCER |= (TIM_CCER_CC2E | TIM_CCER_CC2NE);
			TIM1->CCR2 = _PWM;
	}
	else
		TIM1->CCER &= ~(TIM_CCER_CC2E | TIM_CCER_CC2NE);

}
void phaseCOut(unsigned short _PWM){
	if(_PWM >= PWM_THRESHOLD && _PWM <= PWM_LIMIT){
			TIM1->CCER |= (TIM_CCER_CC3E | TIM_CCER_CC3NE);
			TIM1->CCR3 = _PWM;
	}
	else
		TIM1->CCER &= ~(TIM_CCER_CC3E | TIM_CCER_CC3NE);

}

void ALow (void){
			TIM1->CCER |= (TIM_CCER_CC1E | TIM_CCER_CC1NE);
			TIM1->CCR1 = 0;
}

void BLow (void){
			TIM1->CCER |= (TIM_CCER_CC2E | TIM_CCER_CC2NE);
			TIM1->CCR2 = 0;
}

void CLow (void){
			TIM1->CCER |= (TIM_CCER_CC3E | TIM_CCER_CC3NE);
			TIM1->CCR3 = 0;
}

int readHallEffectSensor (void){
	unsigned short step = 0;
	unsigned short readVal = ((GPIOB->IDR) >> 12) & 7;
		switch (readVal)
	{
		case 5:
		step = 1;
		break;
		case 1:
		step = 2;
		break;
		case 3:
		step = 3;
```

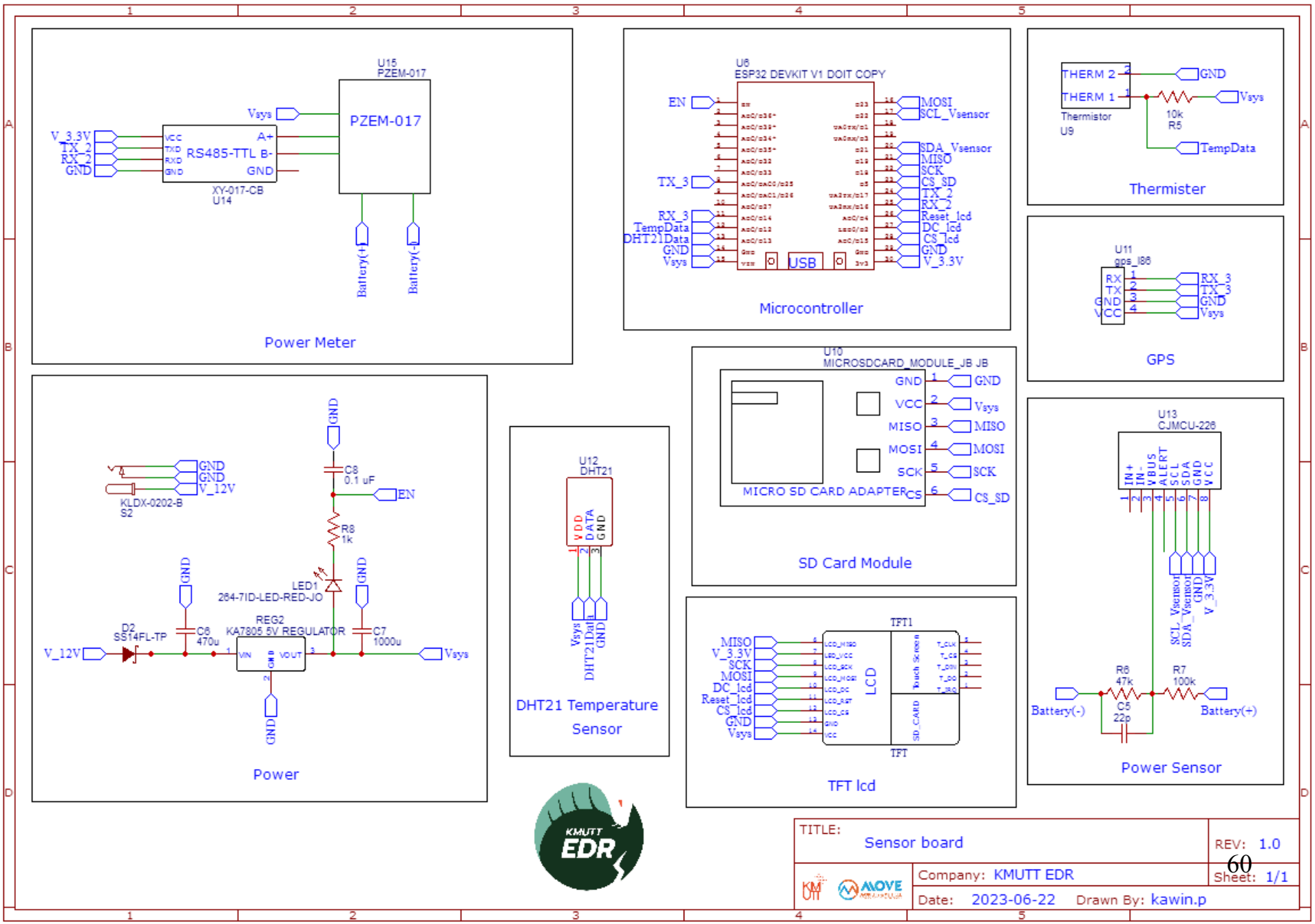
```
                break;
                case 2:
                step = 4;
                break;
                case 6:
                step = 5;
                break;
                case 4:
                step = 6;
                break;
                default:
                step = 0;
        }
        return step;
}

void runSequence (int stp, unsigned short pwm_val){
        if (pwm_val >= PWM_THRESHOLD)
        {
                switch (stp)
                {
                        case 1:
                                ALow();
                                phaseBOut(pwm_val);
                                phaseCOut(0);
                        break;
                        case 2:
                                ALow();
                                phaseBOut(0);
                                phaseCOut(pwm_val);
                        break;
                        case 3:
                                phaseAOut(0);
                                BLow();
                                phaseCOut(pwm_val);
                        break;
                        case 4:
                                phaseAOut(pwm_val);
                                BLow();
                                phaseCOut(0);
                        break;
                        case 5:
                                phaseAOut(pwm_val);
                                phaseBOut(0);
                                CLow();
                        break;
                        case 6:
                                phaseAOut(0);
                                phaseBOut(pwm_val);
```

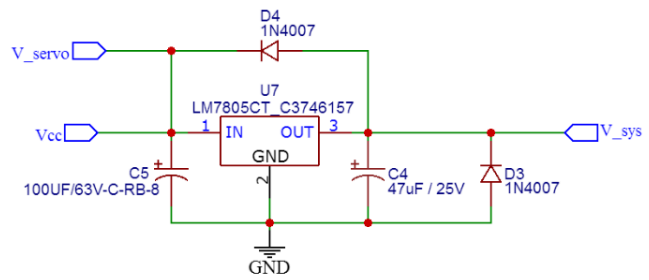
```
                        CLow();
                    break;
                    default:
                        phaseAOut(0);
                        phaseBOut(0);
                        phaseCOut(0);

                }
        }
        else{
                phaseAOut(0);
                phaseBOut(0);
                phaseCOut(0);
        }
}

void analogRead(int *retVal){
                    if(ADC1->SR & ADC_SR_EOC){
                        *retVal = ADC1->DR;
                }
}

int map(float inVal, float inMin, float inMax, float outMin, float outMax){
        int retval = 0;
        retval = (int)((((inVal-inMin)*(outMax-outMin))/(inMax-inMin))+outMin);
        if(retval < outMin)
                retval = outMin;
        else if (retval > outMax)
                retval = outMax;
        return retval;
}

void safety(int threshold){
        int rawADCVal = 0;
        int pwm = 0;
        while(!(ADC1->SR & ADC_SR_EOC));
        rawADCVal = ADC1->DR;
        pwm = map(rawADCVal,PEDAL_IDLE,PEDAL_MAX,0,PWM_LIMIT);
        GPIOC->BSRR = (GPIO_BSRR_BS13);
                phaseAOut(0);
                phaseBOut(0);
                phaseCOut(0);
        while(pwm > threshold){
                GPIOC->BSRR = (GPIO_BSRR_BS13);
                analogRead(&rawADCVal);
                pwm = map(rawADCVal,PEDAL_IDLE,PEDAL_MAX,0,PWM_LIMIT);

                #ifdef DEBUG
                        serialprintf("Safety\n");
```

```
        #endif
}
GPIOC->BSRR = (GPIO_BSRR_BR13);
```

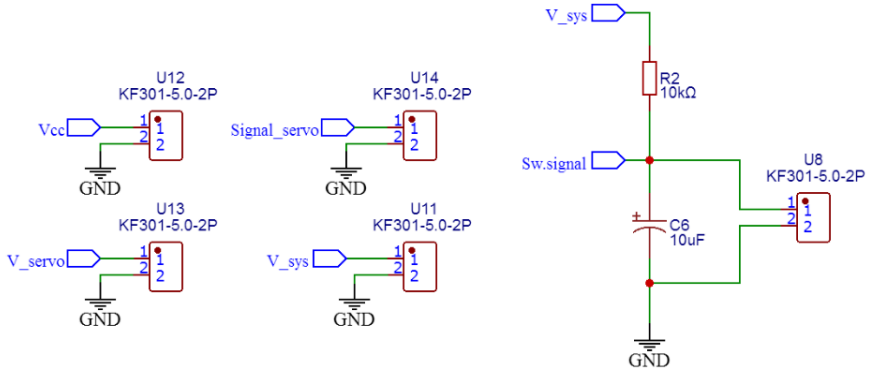
Appendix A

System Electrical PCB schematic

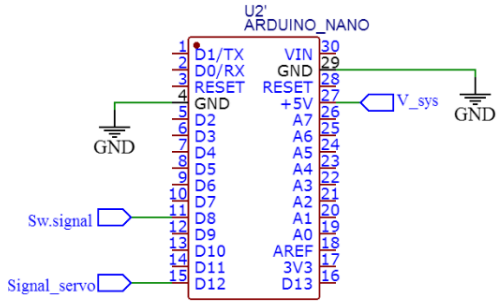

U15
PZEM-017
Vsys
V_3.3V
TX_2
RX_2
GND
VCC
TXD
RXD
GND
RS485-TTL
A+
B-
GND
XY-017-CB
U14
PZEM-017
Battery(+)
Battery(-)
Power Meter
U6
ESP32 DEVKIT V1 DOIT COPY
EN
TX_3
RX_3
TempData
DHT21Data
GND
Vsys
MOSI
SCL_Vsensor
SDA_Vsensor
MISO
SCK
CS_SD
TX_2
RX_2
Reset_lcd
DC_lcd
CS_lcd
GND
V_3.3V
USB
Microcontroller
THERM 2
THERM 1
GND
Vsys
10k
R5
Thermistor
U9
TempData
Thermister
U11
gps_l86
RX
TX
GND
VCC
RX_3
TX_3
GND
Vsys
GPS
GND
GND
V_12V
KLDX-0202-B
S2
C8
0.1 uF
EN
R8
1k
LED1
264-7ID-LED-RED-JO
D2
SS14FL-TP
C6
470u
REG2
KA7805 5V REGULATOR
VIN
GND
VOUT
C7
1000u
V_12V
Vsys
Power
U12
DHT21
VDD
DATA
GND
Vsys
DHT21Data
GND
DHT21 Temperature
Sensor
U10
MICROSDCARD_MODULE_JB JB
MICRO SD CARD ADAPTER
GND
VCC
MISO
MOSI
SCK
CS
Vsys
MISO
MOSI
SCK
CS_SD
SD Card Module
TFT1
MISO
V_3.3V
SCK
MOSI
DC_lcd
Reset_lcd
CS_lcd
GND
Vsys
LCD
Touch Screen
SD_CARD
TFT
TFT lcd
U13
CJMCU-226
IN+
IN-
VBUS
ALERT
SCL
SDA
GND
VCC
SCL_Vsensor
SDA_Vsensor
GND
V_3.3V
R6
47k
R7
100k
C5
22p
Battery(-)
Battery(+)
Power Sensor
TITLE:
Sensor board
REV: 1.0
Company: KMUTT EDR
Sheet: 1/1
Date: 2023-06-22
Drawn By: kawin.p



PART : LDO 12v to 3.3v

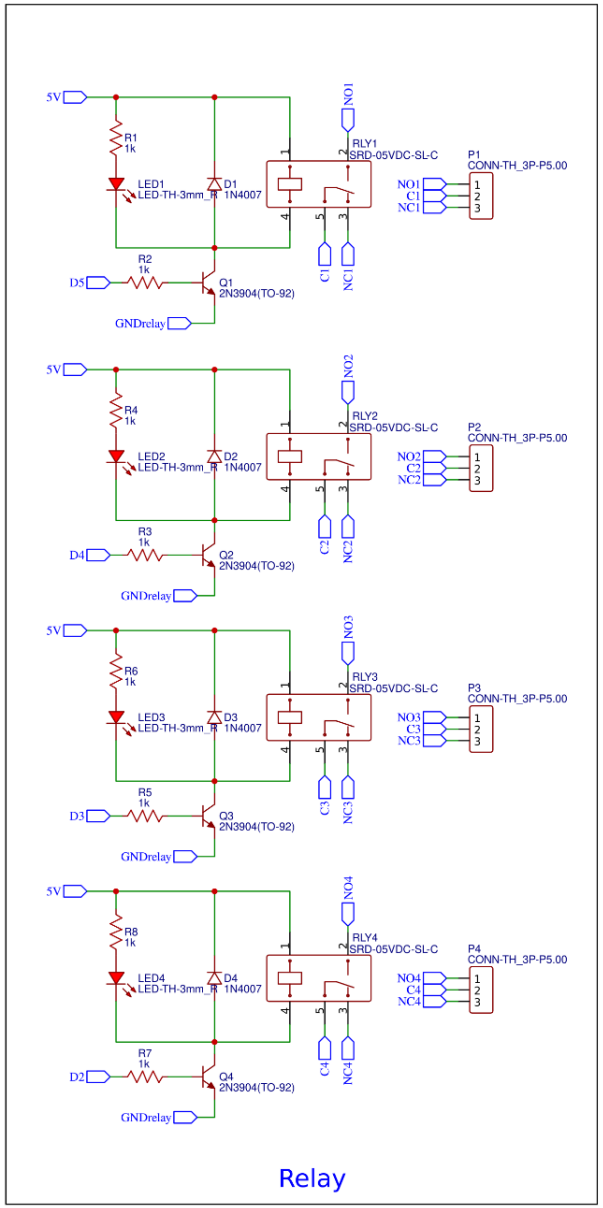


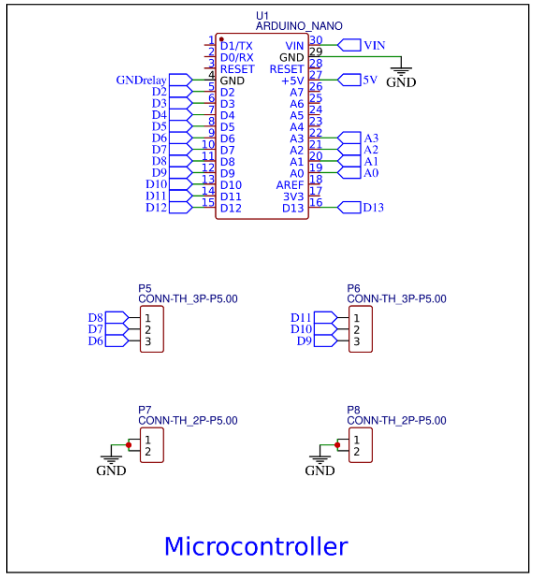
PART : Connector Input & Output

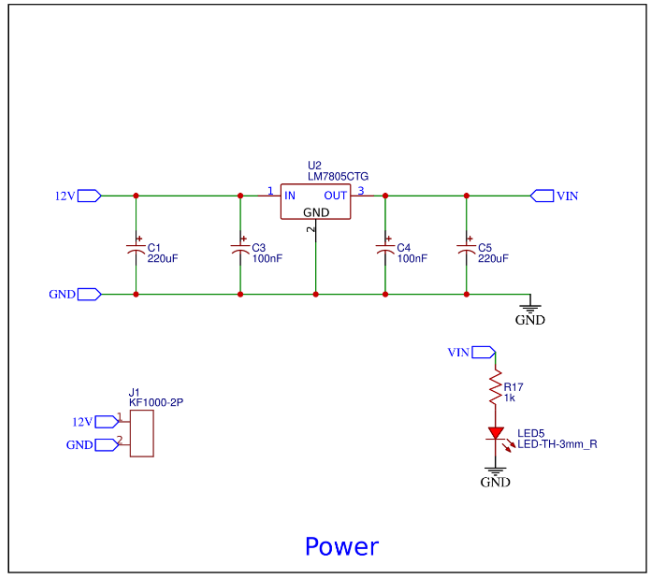


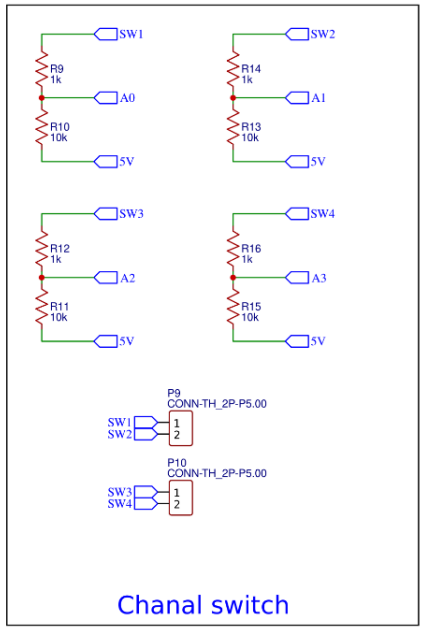
PART : Microcontroller ATMEGA328P

TITLE: Windscreen Wiper Controller		REV: 1.0
Company: KMUTT EDR		Sheet: 1/1
Date: 2023-06-22	Drawn By: Su Riya	









TITLE: Lightning System		REV: 1.0
Company: KMUTT EDR		Sheet: 1/1
Date: 2023-03-17	Drawn By: kanokpol512tsk	

Appendix B

Components Datasheet