POLITECNICO
MILANO 1863

School of Industrial and Information Engineering

Division of Automation and Control Engineering

Master's Degree in Automation and Control Engineering

HARDWARE DESIGN FIRMWARE DEVELOPMENT & MULTI-SENSOR BASED CONTROL OF WATER TANK

Submitted by: Atif Jamshaid Person Code: 862452

Supervised by: Prof. Francesco Castelli Dezza Politecnico Di Milano Supervisor

Ing. Mohamed Eleuche Agena SRL Supervisor

Academic Year 2019-2020

HARDWARE DESIGN FIRMWARE DEVELOPMENT & MULTI-SENSOR BASED CONTROL OF WATER TANK

Master's Industrial Thesis

Submitted by

Atif Jamshaid

Supervised by

Prof. Francesco Castelli Dezza — Politecnico Di Milano Supervisor

Ing. Mohamed Eleuche — Agena SRL Supervisor

School of Industrial and Information Engineering

Division of Automation and Control Engineering

POLITECNICO DI MILANO

Milano, Italia, 2020

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Almighty God to bless me with an opportunity to pursue the academic path I wished. Secondly, I would like to express my sincere gratitude to Prof. Francesco Castelli Dezza for guiding me, sparing his valuable time and suggestions during my thesis work.

I would also like to express heartiest indebtedness to Engineer Mohamed Eleuche of Agena S.R.L, for suggesting this project and continuously guiding and supervising my work. I thank him for providing me all the hardware: the microcontroller, sensors, LCDs and other electronic devices.

I would like to also thank my parents for their continuous support in all fields of my life, for inspiring me and motivating me through ups and downs, my siblings and my friends and colleagues who are constant source of encouragement for me.

Last but not the least, I would also like to thank Politecnico Di Milano and its academic and organizational staff for humble dealing in all aspects and for providing me a chance to pursue higher education here that I had wished once.

ABSTRACT

Hardware Design and embedded firmware development constitutes of four steps: 1) Development of product requirement 2) System design and algorithm development 3) Coding 4) Testing.

In the first phase, the requirements of the product are defined. In this project an electronic board has been developed that takes analogue temperature, digital temperature, humidity, level and tilt measurements from various sensors and based on their values pump, fan, buzzer and LEDs are controlled. An external LCD and Keypad have been used to determine the status of every component.

In the second phase, schematic and printed circuit board have been designed using Computer Aided Design software, KiCad. Then Gerber files have been generated and sent to the printed circuit board manufacturer. Meanwhile all the required components based on Bill of Materials (BOM) have been ordered keeping in mind the price and requirements of the product

In the third phase, coding has been done using C language based on the following things: 1) Requirements of the product mentioned in the first phase and algorithm developed in the second phase 2) According to the datasheets of the electronic components, sensors and microcontroller

In the fourth and final phase, printed circuit board has been assembled and programmed using the code developed in the third phase. Then working of product has been tested using black box and white box testing techniques.

SOMMARIO

La progettazione hardware e lo sviluppo del firmware incorporato prevedono quattro passaggi: 1) Sviluppo del requisito del prodotto 2) Progettazione del sistema e sviluppo dell'algoritmo 3) Codifica 4) Test.

Nella prima fase vengono definiti i requisiti del prodotto. In questo progetto è stata sviluppata una scheda elettronica che prende misure analogiche di temperatura, temperatura digitale, umidità, livello e inclinazione da vari sensori e in base ai loro valori sono controllati pompa, ventola, cicalino e LED.

Nella seconda fase, schematic e printed circuit board sono stati progettati utilizzando il software Computer Aided Design, KiCad. Quindi i file Gerber sono stati generati e inviati al produttore del circuito stampato. Nel frattempo tutti i componenti richiesti basati su bill of materials (BOM) sono stati ordinati tenendo presente il prezzo e i requisiti del prodotto.

Nella terza fase, la codifica è stata eseguita usando il linguaggio C secondo le seguenti cose: 1) Requisiti del prodotto menzionati nella prima fase e algoritmo sviluppato nella seconda fase 2) Secondo le schede tecniche dei componenti elettronici, sensori e microcontrollore.

Nella quarta e ultima fase, il circuito stampato è stato assemblato e programmato utilizzando il codice sviluppato nella terza fase. Quindi la lavorazione del prodotto è stata testata usando le tecniche di test "Black Box" e "White Box".

This page is intentionally left blank

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1 INTRODUCTION

In electronic development projects hardware design and firmware development are two most integral parts. In engineering, hardware design refers to the identification of a system's physical components and their interrelationships.[1] Hardware design of electronic boards refers to the development of schematics on computer aided design software based on the specifications of the product and then using that schematic to develop printed circuit boards. Circuit is designed keeping in mind the product specifications, price of the components and availability of the components.

In the field of technology there are two very important terms: hardware and software. Hardware refers to the physical devices which run programs or other applications. On the other hand, software is what is run on the hardware. Firmware is a term that lies in the middle ground between hardware and software. Firmware is a type of software that is more integrally linked with the device that it is managing so the commands that are received from a remote control are essentially firmware. Software is typically separate from the device itself, and it simply uses the hardware to run, but firmware is part of the machine, and it would not function without it. To summarize, the firmware is essentially the software that manages a device's core functions and allows it to interact with media and other hardware. The most crucial reason why businesses need to focus on firmware when releasing new products is that hardware can often end up bricked by malfunctioning firmware. For those that want to release reliable products consistently and improve their customer experience, stable firmware is a must.[2]

Hardware and firmware engineering design teams often run into problems and conflicts when trying to work together. They come from different development environments, have different tool sets and use different terminology. Often, they are in different locations within the same company or work for different companies. The two teams have to work together, but often have conflicting differences in procedures and methods. Since their resulting hardware and firmware work have to integrate successfully to build a product, it is imperative that the hardware/firmware interface – including people, technical disciplines, tools and technology – be designed properly.[3]

1.1 Problem Statement

In industries fluid tanks ranging from small size to huge size are present that are exposed to normal and severe environmental conditions like heat and humidity levels. High levels of temperature and humidity are not ideal working conditions for the workers around the tank. Therefore, it is necessary to control the temperature and humidity levels and their status should be known to operator.

Additionally, in many applications the need to maintain the level of fluid inside the tank is of extreme importance and appropriate action is required if level of fluid gets out of nominal range. Finally, it is of great important to keep orientation of tank exactly straight in order to measure accurately the level of fluid inside the tank and to take appropriate steps to control fluid level.

1.2 Objectives

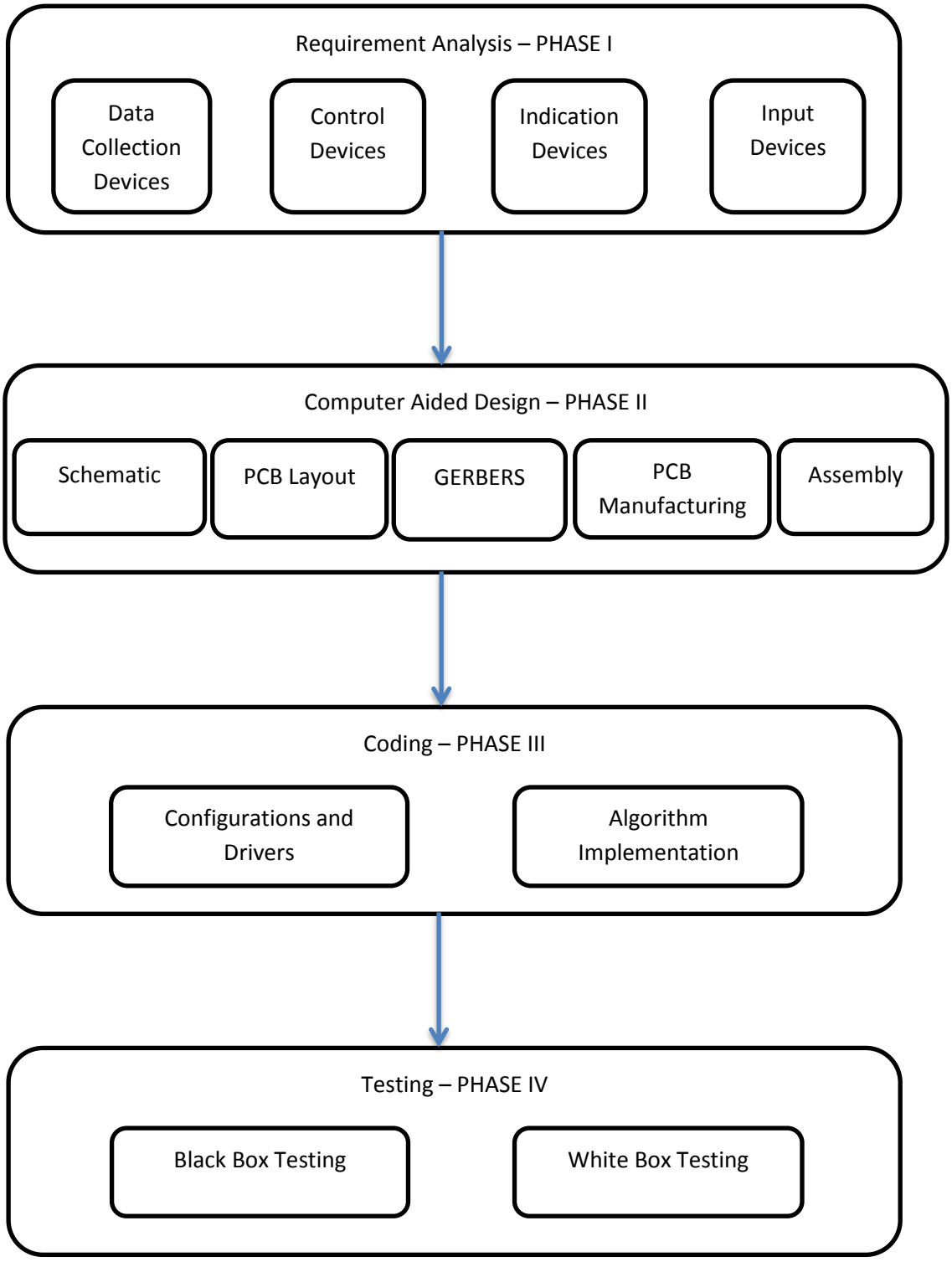
1. Measure the following sensor values
    - Analogue Temperature
    - Digital Temperature
    - Humidity
    - Fluid Level
    - Tilt Measurement

2. Control following devices based on sensor values
    - Fan
    - Pump
    - Buzzer
    - LEDs

3. Display statuses of devices and sensor values on LCD

4. Control display using Keypad

1.3 Project Overview



1.4 Requirement Analysis - Phase-I

First phase of the project revolved around the analysis of the components required to carry out the objectives mentioned in Section 1.2. A brief overview of components used is as follows:

1.4.1 Data Collection Devices

These devices correspond to all the sensors required and microcontroller used to process information received from the sensors and act on control and indication devices. Following sensors have been used:

- NTC10k for Analogue Temperature measurement
- DHT22 for Digital Temperature and Humidity measurements
- 59630 Floating Device for Level Indication
- MPU6050 IMU Module for Tilt Measurement

STM32F103RBT6 micro controller has been used.

1.4.2 Control Devices

Based on sensor values following external devices have been used:

- 12V Fan to keep control of temperature
- Pump to maintain level of fluid inside the tank

1.4.3 Indication Devices

In order to know the status of Control Devices and values obtained from sensors following indication devices have been used:

- LCD1602 for Display
- HSMC150 Red, Green and Yellow LEDs to show status of Control Devices and Power
- Buzzer 1233 as an Alarm device

1.4.4 Input Devices

External Keypad has been used to switch results on LCD.

1.5 Computer Aided Design - PHASE-II

This second phase of project constitutes of three parts.

1.5.1 Schematic

Schematic of the project has been developed in KiCad software. For proper selection of electronic components, circuit analysis has been done. Proper footprints have been either downloaded or manually developed and inserted inside components so that proper PCB Layout could be developed in second part of this phase.

1.5.2 PCB Layout

After creation of schematics and testing them for connections, PCB Layout has been developed in this part of the project. Components according to their footprints are placed on PCB keeping in mind easy routing and size of the printed circuit board.

1.5.3 GERBERS

Once PCB layout has been designed, output Gerber files have been generated which describes various layers of the PCB

1.5.4 PCB Manufacturing

After sending Gerber files to the manufacturer, PCB is manufactured and sent back to us

1.5.5 Assembly

Once the PCB has arrived back after manufacturing, all components are assembled according to the design.

1.6 Coding - PHASE-III

In this third phase of thesis, software part of the project has been developed. All coding has been done using C language using Eclipse IDE. STM32 Nucleo Board has been used in this phase to test and debug the code. This phase constitutes of two major parts:

1.6.1 Configurations

Several sensors have been used and they need to communicate with the microcontroller. So, in this part of coding various modules of microcontroller have been configured to receive data from sensors e.g. ADC, GPIOs, I2C. Additionally LCD and Keypad has been configured. All these configurations are based on datasheet of STM32 microcontroller and datasheets of corresponding device with which micro controller is communicating.

1.6.2 Algorithm implementation

In second part of coding algorithm has been implemented in order to achieve the desired objectives of the project.

1.7 Testing - PHASE-IV

Once the coding part is done and PCB has arrived and assembled using the components of schematic, code is tested on the hardware to complete the firmware development of the project. Two basic methods of testing have been used

1.7.1 Black Box Testing

In this type of testing desired output has been checked without going through the implementation.

1.7.2 White Box Testing

In this type of testing code has been modified to achieve the desired result.

Testing phase completes the project.

CHAPTER 2

2 REQUIREMENT ANALYSIS

This is the first phase of an electronic development project. In this phase all the specifications of the project are determined and how an end user will see the final product. Along with the specifications it is determined which are the devices or sensors that are needed, in order to achieve the final product. Therefore, firstly the specifications of the product are covered in detail and then description of devices and sensors used to achieve those specifications are discussed.

2.1 Specifications

- In order to guarantee safety of the workers around the fluid tank it is very important to measure the temperature of the environment. This can be achieved by using an analogue temperature sensor and digital temperature sensor.
- Secondly humidity measurement is very important to guarantee ideal working condition for the workers. Therefore, a humidity sensor is required to measure the humidity of the environment.
- Another important specification is to maintain the fluid level of the tank. For this either proximity sensor could be used, or floating device could be used. As it is required to maintain fluid level in a nominal range at least two sensors would be required to determine low level and high level of fluid.
- Finally, in order to measure fluid level accurately it is important for tank to be exactly horizontal. As there is no support available for the tank in this application it is required to measure tilt value of the tank in order to guarantee accurate level measurement.

Once these sensor measurements are taken, they are sent to the microcontroller and depending on the values of these measurements following actions should be taken:

- If temperature is above a certain level fan is turned on unless temperature gets in the nominal range. In normal situation fan should be off and if temperature is too high that fan cannot lower its level then it is useless to use fan and therefore it should be off. In such case buzzer should turn on to indicate emergency.
- In this project no device has been used to control level of humidity. Therefore, if humidity is above nominal value, buzzer should be turned on to indicate emergency.
- If level of fluid is below the low level, pump should be turned on to increase level of fluid to its nominal value. Similarly, if level of fluid is above the high-level fluid should be drained manually as no device has been used to drain fluid out in this project.
- Finally, if tilt measurement is not in a range then buzzer should beep to indicate that level measurements cannot be done accurately.

Other Specification include:

- Eight different LEDs are used to indicate working of various devices
    - One green LED should be used to indicate a running power supply.
    - One green LED should be used to indicate a Turned-On fan.
    - One green LED should be used to indicate a Turned-on Pump.
    - Three red LEDs are used to indicate sensor measurements taken correctly from analogue temperature sensor, humidity sensor and digital temperature sensor.
    - One green LED should be used to indicate up level of fluid
    - One Yellow LED should be used to indicate down level of fluid
- LCD is used to indicate the values of all sensor measurements and status of fan, pump and buzzer
- Keypad is used to display on LCD as following:
    - Temperature Value and Fan Status when Key 1 is pressed
    - Humidity Value and Buzzer Status when Key 2 is pressed
    - Level Status and Pump Status when Key 3 is pressed

2.2 Data Collection Devices

2.2.1 Micro Controller

Micro Controller family used in this project is STM32. Reason to use STM32 is prior know how and knowledge of this microcontroller. Specifically, STM32F103RBT6 is used on the PCB. For coding and debugging STM Nucleo board is used.

It is a 64-pin microcontroller in LQFP64 package with 12-bit ADC Channel, I2C supported by 7 channel DMA controller. Those functionalities that have been used in the project are only mentioned here.

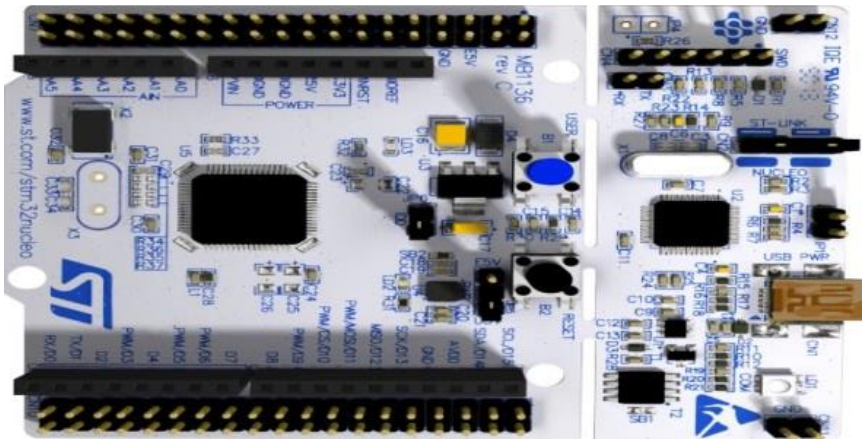


Figure 1: STM32F1RB Nucleo Board

2.2.2 Analogue Temperature Sensor

Thermistor is a type of resistor whose resistance is dependent on temperature.[4] Negative temperature coefficient thermistors are widely used as temperature sensors. In NTC thermistors, with the increase of temperature, resistance decreases. Therefore, by measurement of resistance, temperature value can easily be determined. Following NTC10K sensor has been used to measure analogue temperature of the environment.

Figure 2: NTC10K

NTC10K sensor has nominal resistance value of 10K at nominal temperature i.e 25 ^{o}C. The variation of resistance with temperature can be seen from the table below that is taken from the datasheet of the sensor.[5]

T (°C)	$B_{25/100}$ = 4300 K, R_{25} = 10000 Ω, T_R = 25 °C, $\Delta R_R/R_R$ = ± 5%					
	R_{nom}[Ω]	R_{min}[Ω]	R_{max}[Ω]	$\Delta R_R/R_R$[±%]	ΔT[±°C]	α (%/K)
−55.0	1214600	960540	1468600	20.9	2.9	7.3
−50.0	844390	678960	1009800	19.6	2.7	7.1
−45.0	592430	483870	701000	18.3	2.6	7.0
−40.0	419380	347620	491150	17.1	2.5	6.9
−35.0	299480	251710	347240	16.0	2.4	6.7
−30.0	215670	183670	247670	14.8	2.3	6.5
−25.0	156410	134870	177940	13.8	2.2	6.3
−20.0	114660	100050	129270	12.7	2.1	6.2
−15.0	84510	74576	94443	11.8	2.0	6.0
−10.0	62927	56128	69726	10.8	1.9	5.8
−5.0	47077	42421	51733	9.9	1.8	5.6
0.0	35563	32359	38767	9.0	1.6	5.5
5.0	27119	24905	29332	8.2	1.5	5.3
10.0	20860	19328	22391	7.3	1.4	5.2
15.0	16204	15143	17266	6.6	1.3	5.0
20.0	12683	11949	13418	5.8	1.2	4.9
25.0	**10000**	**9500**	**10500**	**5.0**	**1.1**	**4.7**
30.0	7942	7484	8400	5.8	1.3	4.6
35.0	6327	5918	6735	6.5	1.4	4.5
40.0	5074	4713	5435	7.1	1.6	4.3
45.0	4103	3784	4421	7.8	1.8	4.2
50.0	3336	3056	3616	8.4	2.0	4.1
55.0	2724	2479	2970	9.0	2.3	4.0
60.0	2237	2022	2452	9.6	2.5	3.9
65.0	1846	1658	2034	10.2	2.7	3.8
70.0	1530	1366	1695	10.7	2.9	3.7
75.0	1275	1132	1419	11.3	3.1	3.6
80.0	1068	941.9	1194	11.8	3.4	3.5
85.0	899.3	788.7	1010	12.3	3.6	3.4
90.0	760.7	663.4	858.0	12.8	3.8	3.3
95.0	645.2	559.6	730.9	13.3	4.1	3.2
100.0	549.4	473.9	624.9	13.7	4.3	3.2
105.0	470.0	403.3	536.8	14.2	4.6	3.1
110.0	403.6	344.5	462.7	14.6	4.8	3.0
115.0	347.4	295.0	399.8	15.1	5.1	3.0
120.0	300.1	253.5	346.6	15.5	5.4	2.9
125.0	260.1	218.7	301.5	15.9	5.6	2.8

Table 1 : NTC10K

2.2.3 Digital Temperature and Humidity Measurement

DHT22 is a Digital-output relative humidity & temperature sensor/module which is used in this project to measure the digital value of temperature and humidity. Small size, low power consumption and long transmission distance enables DHT22 to be suited in all kinds of harsh application occasions.

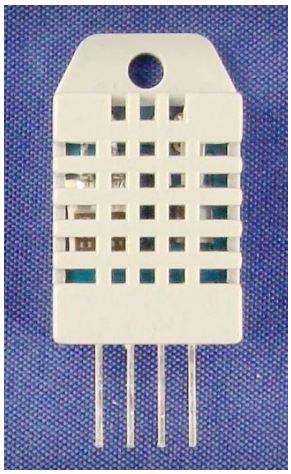


Figure 3 DHT22 Module

Single row packaged with four pins, making the connection very convenient. Technical specifications of the module are mentioned in the table below taken from the datasheet of the module.[6]

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

Table 2: DHT22 Technical Specifications

2.2.4 Level Sensor

In order to determine the level of the fluid in the tank, two 59630 sensors have been used. The 59630 is a reed level sensor with integral float actuator and an M8 x 1.25mm pitch thread with a choice of normally open, normally open high voltage, normally closed or change over contacts. It can switch up to 265Vac/300Vdc at 10VA.

Figure 4: 59630 Sensor

Two such sensors have been used in order to read high and low-level values. Electrical characteristics of the sensor are mentioned in the table below which is taken from the data sheet of the device.[7]

Contact Type			Normally Open	Normally Open High Voltage	Change Over	Normally Closed
Switch Type			1	2	3	4
Contact Rating [1]		VA/Watt - max.	10	10	5	10
Voltage [4]	Switching [2]	Vdc - max.	200	300	175	200
		Vac - max.	140	265	120	120
	Breakdown [3]	Vdc - min.	250	400	200	250
Current [4]	Switching [2]	Adc - max.	0.5	0.4	0.25	0.5
		Aac - max.	0.35	0.30	0.18	0.18
	Carry	Adc - max.	1.2	1.4	1.5	1.2
Resistance [5]	Contact, Initial	Ω - max.	0.2	0.2	0.2	0.2
	Insulation	Ω - min.	10^{10}	10^{10}	10^{9}	10^{10}
Capacitance	Contact	pF - typ.	0.3	0.2	0.3	0.3
Temperature	Operating	°C	-40 to +105	-20 to +105	-40 to +105	-40 to +105

Table 3: Electrical Ratings

2.2.5 Tilt Measurement

MPU 6050 IMU module has been used to measure tilt value of the tank. The module features 3 axis accelerometer and 3 axis gyroscope. This is one of the most common IMU module used in microcontrollers.

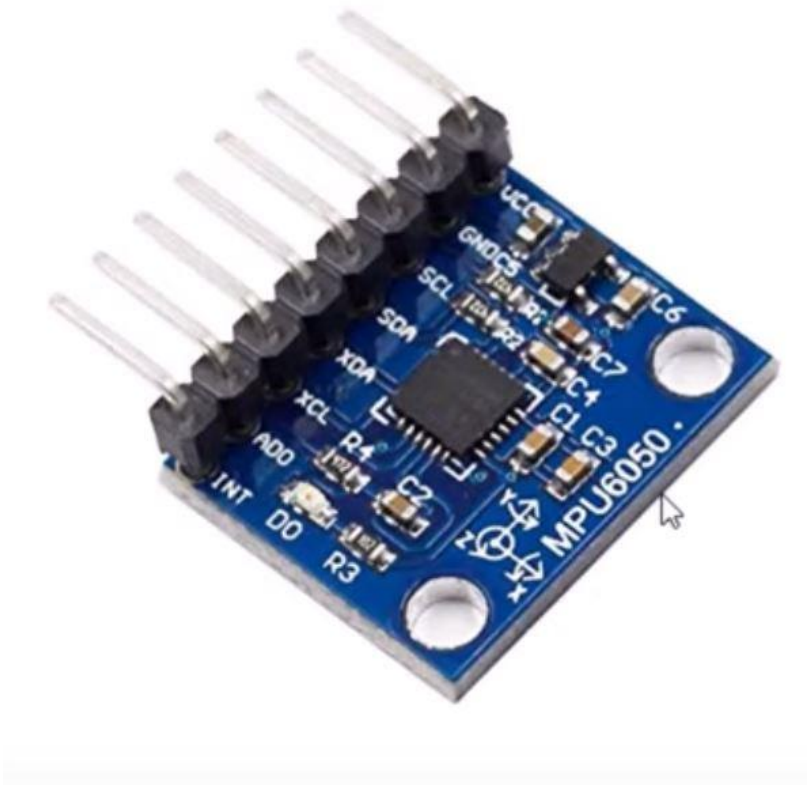


Figure 5: MPU 6050 IMU Module

Main features of the accelerometer are:

- User programmable full scale range 2g/4g/8g/16g
- I2c serial interface
- 16 bit data output
- Magnetic field scale 1.3 to 8.1 gauss

Main features of gyroscope are:

- User programmable full scale range of 250/500/1000/2000 dps
- I2C digital output interface
- 16 bit data output
- Wide supply voltage: 2.4 V – 3.6V

More details will be discussed in Coding part of the report.

2.3 Control Devices

2.3.1 Fan

To control the temperature an external 12 V fan with DC brushless motor has been used.

Figure 6: 12V Fan

Main features of the Fan taken from the datasheet[8] are

- Motor: Brushless DC
- Motor Protection: Impedance Protected
- Bearing Type: Two ball, Sleeve or Hydro Dynamic

2.3.2 Pump

To control the level of fluid in the tank a 12 V pump has been used.[9]

Figure 7:12V Pump

2.4 Indication Devices

2.4.1 LCD

The 1602 LCD has been used in this project to display the status of various sensors and indication devices. As the name suggests it has 16 characters and 2 lines.

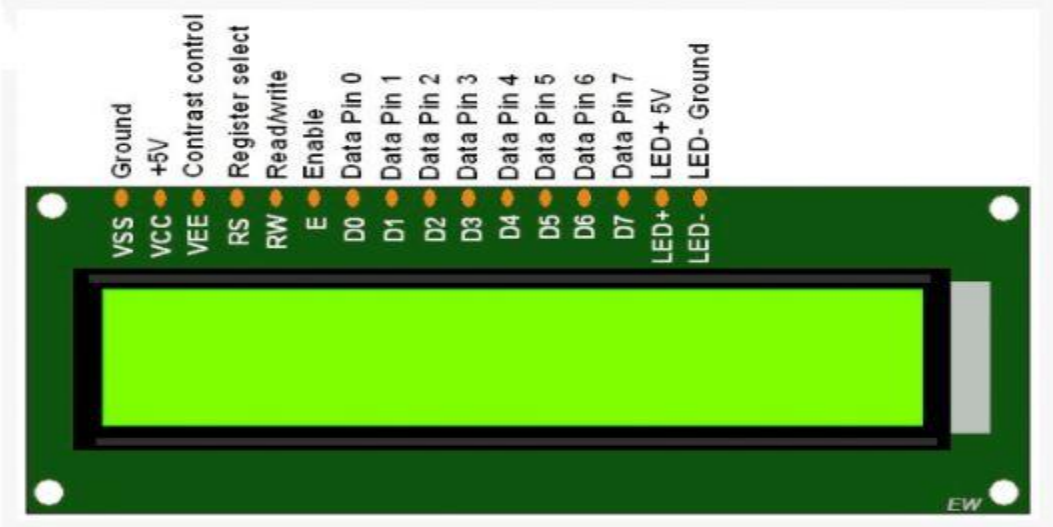


Figure 8: LCD1602

Other features of LCD include:

- Input Data: 4 Bits or 8 Bits interface available
- Display Font: 5 x 8 Dots
- Single 5V Power Supply
- Back Light: LED(White)

Electrical characteristics of the LCD taken from the datasheet [10] are shown in the table below:

(Ta=25 C;Vdd=3.0V±10%,otherwise specified)

Item	Symbol	Test Condition	Min.	Typ.	Max.	Unit
Power Supply for Logic	Vdd	--	4.7	5.0	5.5	V
Operating Voltage for LCD	Vdd-Vo	--	--	5.0	--	V
Input High voltage	Vih	--	2.2	--	Vdd	V
Input Low voltage	Vil	--	-0.3	--	0.6	V
Output High voltage	Voh	-Ioh=0.2mA	2.4	--	--	V
Output Low voltage	Vol	Iol=1.2mA	--	--	0.4	V
Power supply current	Idd	Vdd=3.0v	--	1.1	--	mA

Table 4: Electrical Characteristics

2.4.2 LED

HSMx-C150 LEDs[11] have been used in this project to indicate the status of sensors and whether control devices are turned on or off.

Figure 9: HSMx-C150 LED

Main features of this series of LEDs are:

- Small size
- Industry standard footprint
- Compatible with IR solder
- Diffused optics
- Operating temperature range of -30 to +85 oC
- Various colors available

2.4.3 Buzzer

To indicate if humidity or temperature values are above nominal values, we use a piezo buzzer[12] which beeps to indicate warning. Beep frequency is 2 KHz. Applied voltage is between 3V – 5V.

Figure 10: Piezo Buzzer

2.5 Input Device

2.5.1 Keypad

A four button Keypad[13] has been used to switch the display of LCD showing the status of sensors and control devices upon the discretion of user.

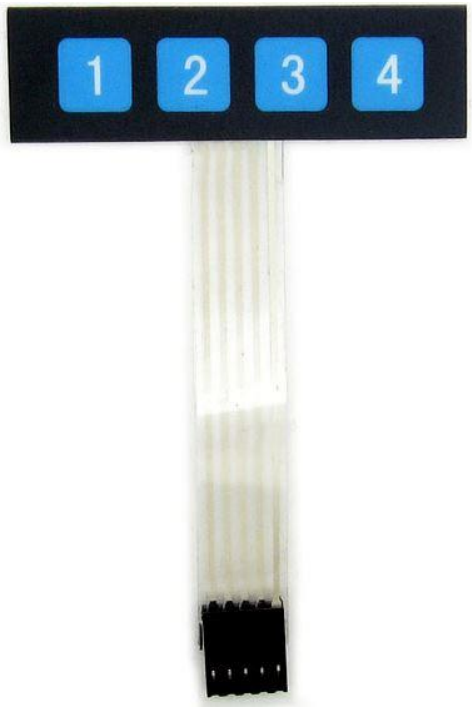


Figure 11: KeyPad

After doing detailed analysis of specifications and main devices to be used in the project next step is design.

CHAPTER 3

3 COMPUTER AIDED DESIGN

This is the second phase of the project. After analyzing the specifications and major devices that would be used it is required to design the schematic diagram of the whole system. Next part after schematic is to design the printed circuit board. Once printed circuit board has been designed, Gerber files have been generated and sent to the PCB manufacturer for developing PCB. KiCad software has been used in this project for computer aided design.

3.1 Schematic

To design the schematic, many important considerations are kept in mind. The values of resistors, capacitors and inductors and type of ICs and electronic components all depends on the electrical requirements of the application and individual components. In the following sections every schematic of this project has been discussed.

3.1.1 NTC 10K Schematic

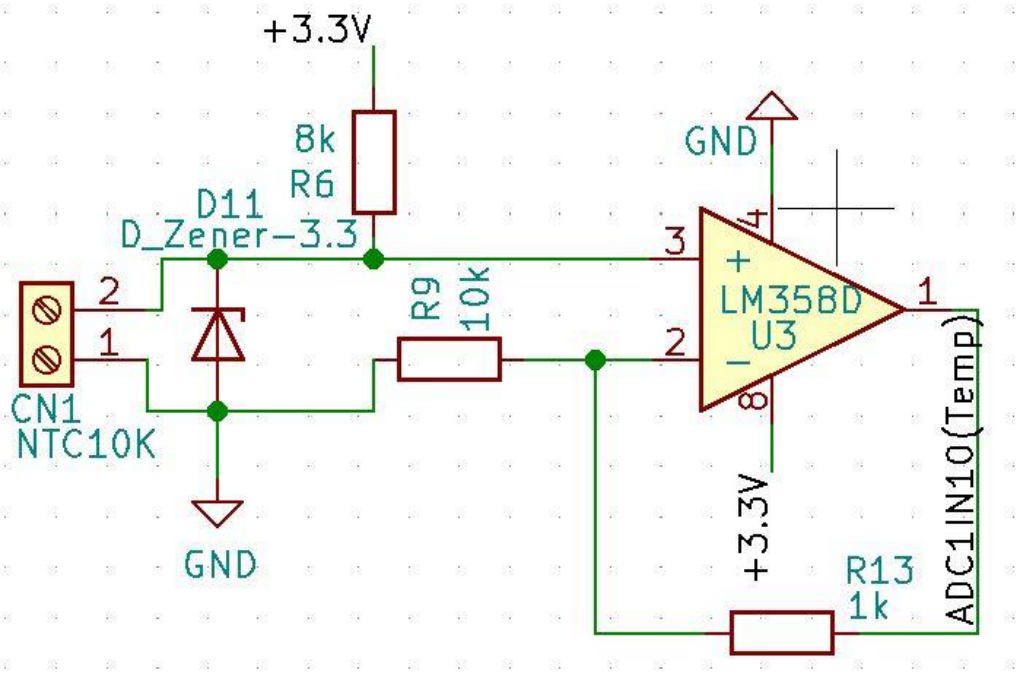


Figure 12: NTC10K Schematic

3.1.1.1 Description

This temperature sensing circuit uses a resistor in series with a negative temperature coefficient (NTC) thermistor to form a voltage divide, which has the effect of producing an output voltage that is linear over temperature. The circuit uses an operational amplifier in a non-inverting configuration with inverting reference to offset and gain the signal, which helps to utilize the full ADC resolution and increase measurement accuracy.[14]

3.1.1.1.1 LM358D

LM358D[15] has been used as operational amplifier. These amplifiers have several distinct advantages over standard operational amplifier types in single supply applications. They can operate at supply voltages as low as 3.0 V or as high as 32 V. Some features of LM358D are:

- Short Circuit Protected Outputs
- True Differential Input Stage
- Single Supply Operation: 3.0 V to 32 V
- Low Input Bias Currents • Internally Compensated
- Common Mode Range Extends to Negative Supply
- Single and Split Supply Operation

Figure 13: LM358D

3.1.1.1.2 Other Components

CN1 is the connector where temperature sensor has to be connected. A Zener diode D11 is used in order to avoid the flow of current in opposite direction.

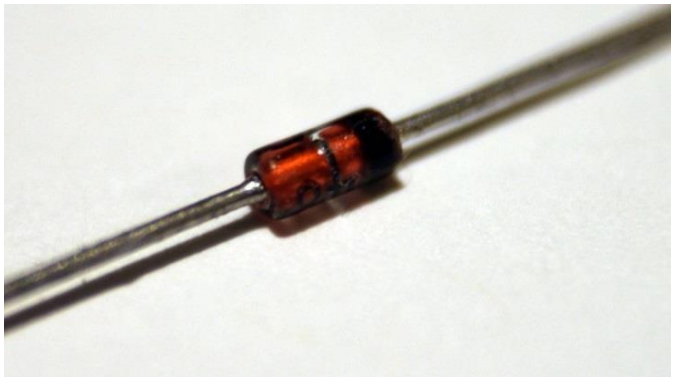


Figure 14: Zener Diode

3.1.2 DHT22 Schematic

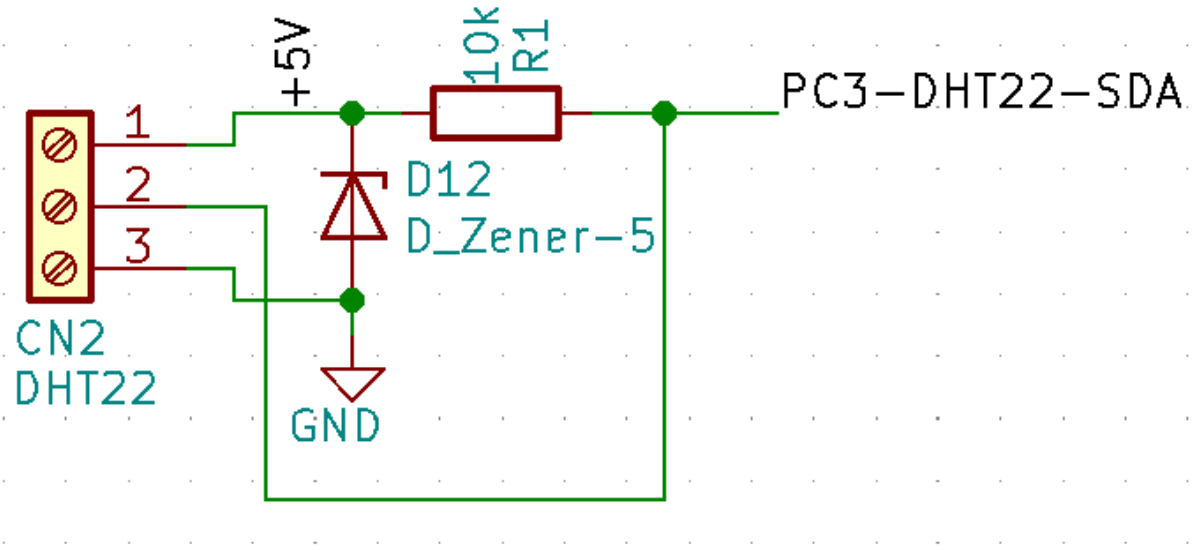


Figure 15: DHT22 Schematic

3.1.2.1 Description

The DHT22 sensors have four pins, VCC, GND, data pin and a not connected pin which has no usage. A pull-up resistor from 5K to 10K Ohms is required to keep the data line high and in order to enable the communication between the sensor and the microcontroller. Zener diode D12 is connected to block reverse current.

3.1.2.1.1 DHT22

DHT22 consist of a humidity sensing component, a NTC temperature sensor (or thermistor) and an IC on the back side of the sensor.

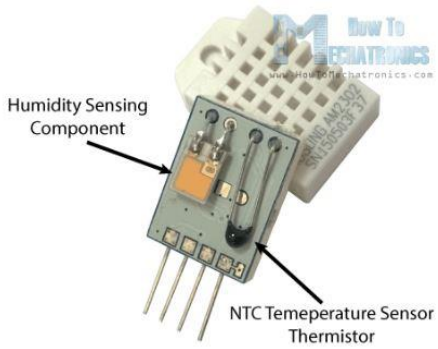


Figure 16: DHT22 Internal View

For measuring humidity it uses the humidity sensing component which has two electrodes with moisture holding substrate between them. So, when the humidity changes, the conductivity of the substrate changes, or the resistance between these electrodes changes. This change in resistance is measured and processed by the IC which makes it ready to be read by a microcontroller.

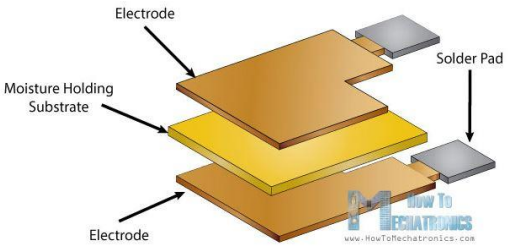


Figure 17: Humidity Measuring Components

On the other hand, for measuring temperature these sensors use a NTC temperature sensor or a thermistor. A thermistor is actually a variable resistor that changes its resistance with change of the temperature. These sensors are made by sintering of semi conductive materials such as ceramics or polymers in order to provide larger changes in the resistance with just small changes in temperature. The term "NTC" means "Negative Temperature Coefficient", which means that the resistance decreases with increase of the temperature.[16]

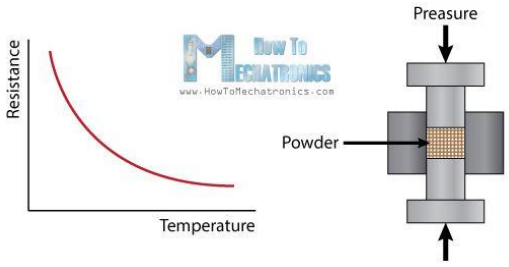


Figure 18: NTC Working

3.1.3 59630 Reed Level Sensors Schematic

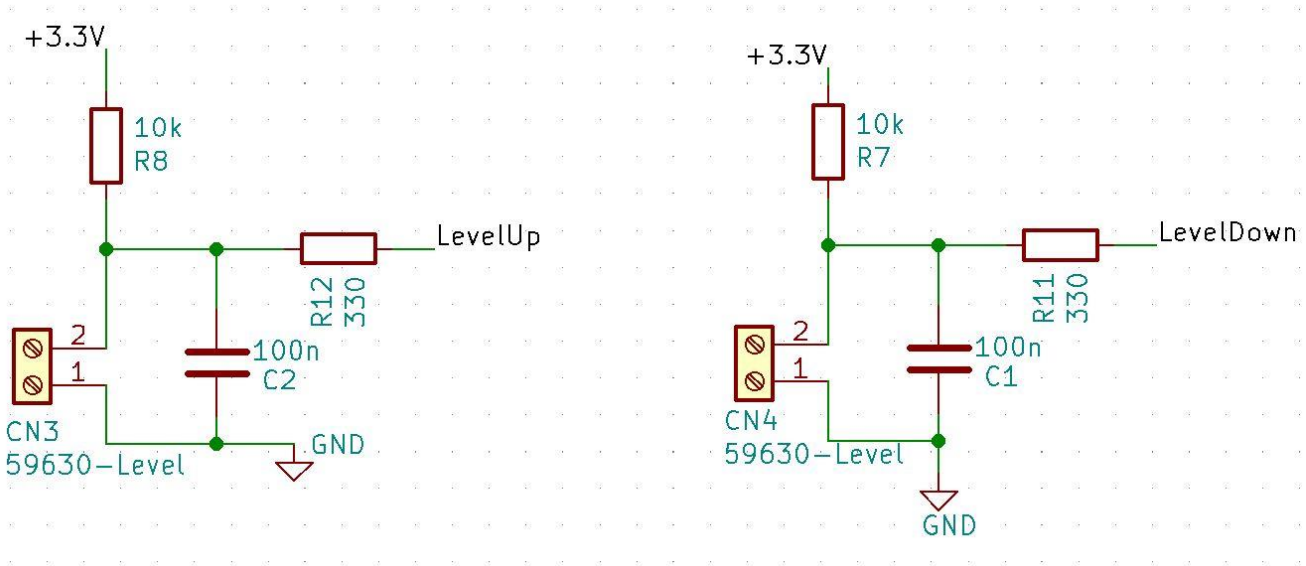
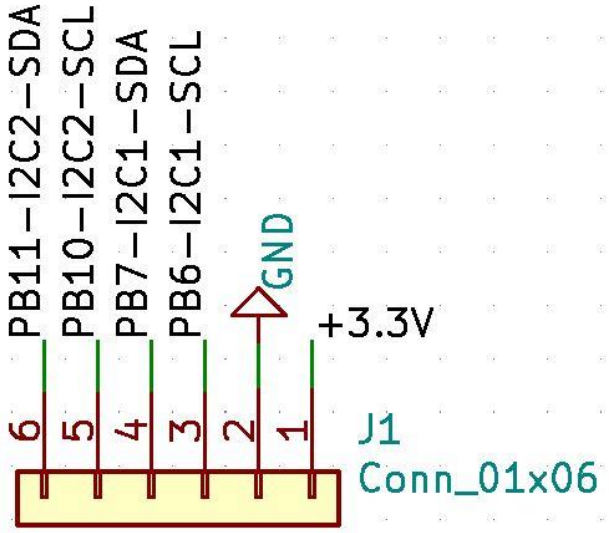


Figure 19: 59630 Sensors Schematic

3.1.3.1 Description

59630 sensor has two cables connected through CN3 and CN4 connectors. Resistor capacitor connection shown in the schematic is used to make output on controller pin low when sensor contact is open. The other side of schematic is directly connected to micro controllers GPIO pins to read the value.

3.1.4 I2C Connector for IMU Module



3.1.4.1 Description

In the project MPU6050 IMU module has been used to measure the value of accelerometer and gyroscope. MPU6050 module uses I2C to connect to microcontroller. The data and clock lines of both available I2C pins of the microcontroller are connected to J1 connector along with power and ground connections.

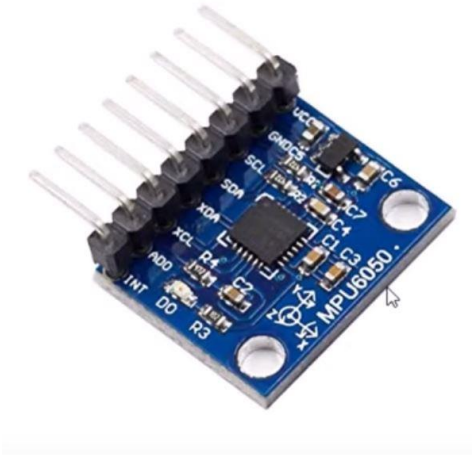


Figure 20: MPU6050

It can be seen in the figure that MPU has data and clock pins and they are connected to the I2C pin connector for serial data transfer from IMU module to microcontroller.

3.1.5 12 V Pump Schematic

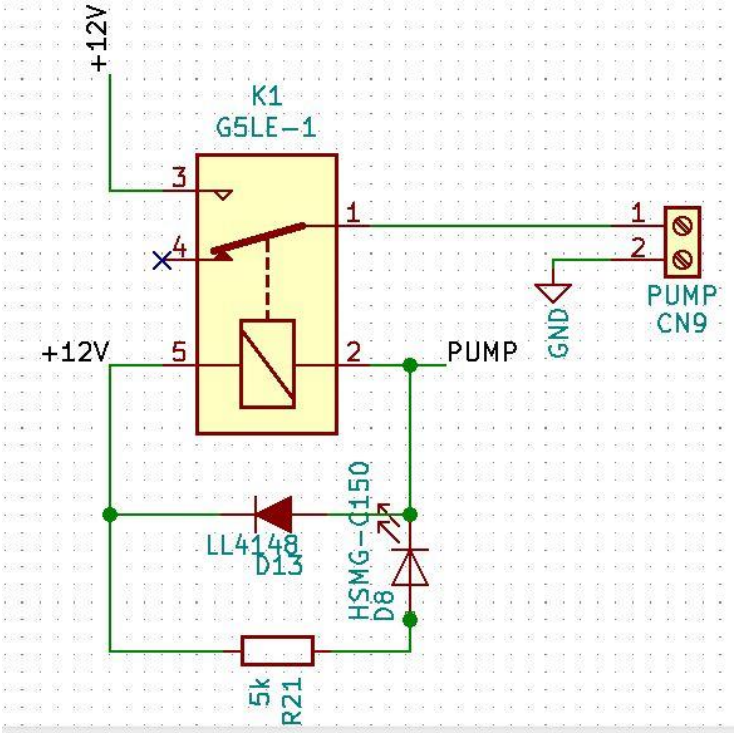


Figure 21: Pump Schematic

3.1.5.1 Description

In the schematic of Pump in order to isolate the rest of the circuit from the pump a relay has been used. There is an LED in the circuit which turns on when microcontroller gives one on the pump pin. and therefore, turning on the pump. A diode has been connected to relay to avoid damaging nearby components sensitive to high voltage. Relay is not directly connected to the microcontroller. A Darlington pair has been used instead which would be described in a later section.

3.1.5.2 Pump

A pump is a device whose purpose is to push the liquid to a certain place. In this project we are using pump to push the fluid inside the tank in case it is below nominal value. A Durevole 12V DC pump has been used here. It has a diaphragm structure and high-pressure capability.

Figure 22: Durevole 12V DC Pump

3.1.5.3 Relay

Relay are the switches which aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts of an electronic circuit. When the relay contact is open, the relay is not energized with the open contact. However, if it is closed (NC), the relay is not energized given the closed contact. Relay can detect overcurrent, overload, undercurrent and reverse current to ensure the protection of electronic equipment. Relay used in this project to isolate pump from the rest of the circuit is G5LE PCB Power Relay.

Figure 23: G5LE Relay

The diagram below sheds focus on the internal section of the relay in the circuit. There is an iron core delimited with the control coin. The power source has been connected with electromagnet through load contacts and a control switch. When energy is supplied to the circuit through the control coil, magnetic fields are intensified given the commencement of energizing. This way, upper contact arms gets attracted by the lower fixed arm which closes the contacts leading to the short circuit. However, if the relay was de-energized, an open circuit is created with the opposite movement of the contact.[17]

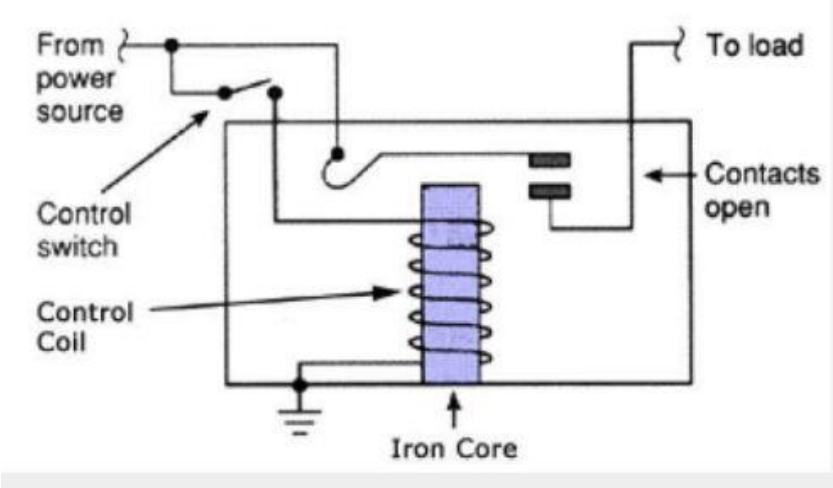


Figure 24: Relay Function

3.1.5.4 Flywheel Diode

A flyback/flywheel diode is placed with reverse polarity from the power supply and in parallel to the relay's inductance coil. The use of a diode in a relay circuit prevents huge voltage spikes from arising when the power supply is disconnected. When the power supply is connected to the relay, the inductance coil's voltage builds up to match that of the power source. The speed at which current can change in an inductor is limited by its time

constant. In this case, the time it takes to minimize current flow through the coil is longer than the time it takes for the power supply to be removed. Upon disconnection, the inductance coil reverses its polarity to keep current flowing according to its dissipation curve (i.e., % of maximum current flow with respect to time). This causes a huge voltage potential to build up on the open junctions of the component that controls the relay. This voltage built up is called flyback voltage. It can result in an electrical arc and damage the components controlling the relay. It can also introduce electrical noise that can couple into adjacent signals or power connections and cause microcontrollers to crash or reset. To mitigate this issue, a diode is connected with reverse polarity to the power supply. Placing a diode across a relay coil passes the back EMF and its current through the diode when the relay is energized as the back EMF drives the flyback protection diode in forward bias. When the power supply is removed, the voltage polarity on the coil is inverted, and a current loop forms between the relay coil and protection diode; the diode again becomes forward biased. The diode allows current to pass with minimal resistance and prevents flyback voltage from building up.[18]

3.1.6 12 V Fan Schematic

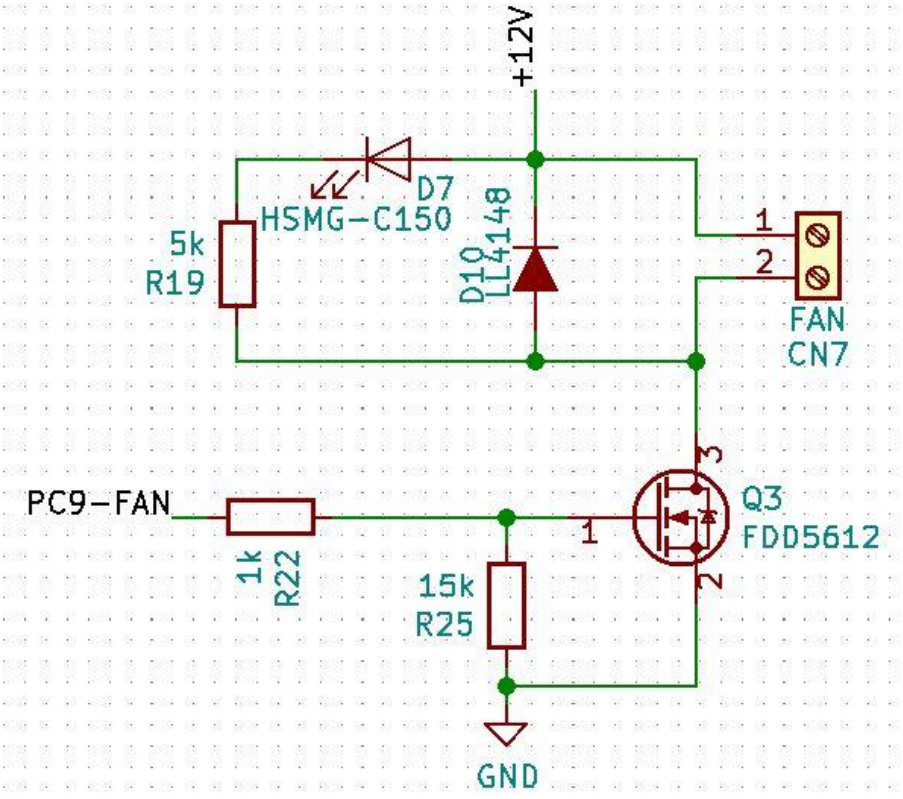


Figure 25: Fan Schematic

3.1.6.1 Description

A 12 V fan is used to get temperature in control. Fan is not connected directly to microcontroller instead it is isolated through a MOSFET and switch the fan. An LED is connected in series with a resistor which will turn on when Fan will receive power from microcontroller. The series resistor is used because supply is 12 V and LED has low power rating. A flywheel diode is connected in reverse bias to avoid any reverse current.

3.1.6.2 12 V Fan

A 12 V fan with DC brushless motor is connected to connector CN7

Figure 26: DC Axial Fan

3.1.6.3 MOSFET

60V, 18A N Channel MOSFET FDD5612 is used to switch the fan. From the datasheet [8] of fan it can be noted that fan consumes 50 mA current. From the datasheet[19] of this MOSFET following values can be recorded:

$$R_{DS(on)} = 64m\Omega$$
$$P_D = 1.6W$$

If we compute power dissipation it can be found by:

$$P = I^2 R$$
$$P = 50mA^2 \; x \; 64m\Omega$$
$$P = 0.16 \, mW$$

As FDD5612 can support maximum of 1.6 W therefore 0.16mW consumption is well inside the range. Therefore, this MOSFET has been used to switch the fan and isolate the circuitry of Fan from micro Controller.

Figure 27: FDD5612 MOSFET

3.1.6.4 Other Components

A flywheel diode is used to stop the reverse current. Detailed explanation is mentioned in last section. An HSMG C150 green LED is connected in parallel to fan. It will turn on when the fan will turn on. A series resistance is used with the LED to get voltage across LED to nominal value i.e. 3V.

3.1.7 Buzzer Schematic

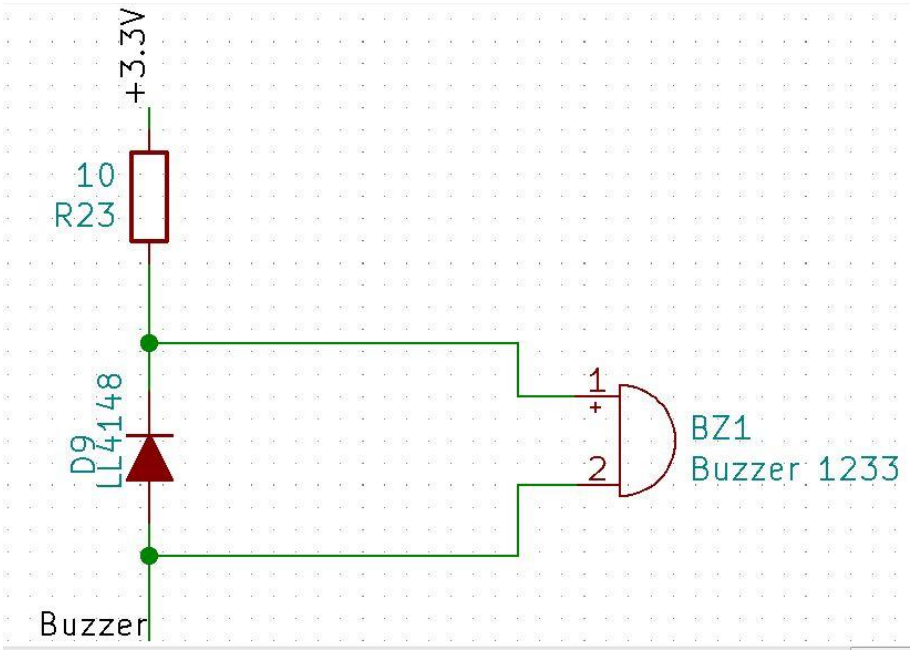


Figure 28: Buzzer Schematic

3.1.7.1 Description

Buzzer is connected to microcontroller according to the circuitry shown above. Again, diode is connected to avoid reverse current towards microcontroller. A very small 10Ω resistor is used to supple complete 3.3 V to the buzzer. Buzzer is not directly connected to the microcontroller. A Darlington pair has been used instead which would be described in a later section

3.1.8 LCD Schematic

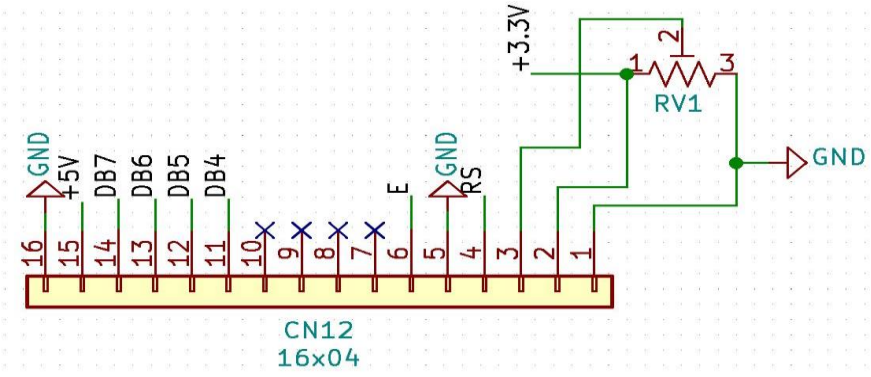


Figure 29: LCD16x02 Connector

3.1.8.1 Description

LCD 16x02 is used to display status of sensors and indication devices. LCD 16x02 has 16 pins which are connected to CN12 connector. First 3 pins of the connector are attached to a potentiometer which is used to control the brightness of LCD screen. Pin 4,5 and 6 are Register Select (RS), Read/Write (R/W) and Enable (E) respectively and they are connected to microcontroller GPIOS. There are eight data lines and only four of them are used to transmit data from controller to LCD. Therefore, first four data lines are not connected while last four are connected to GPIOs of microcontroller.

3.1.9 Darlington Connections Schematic

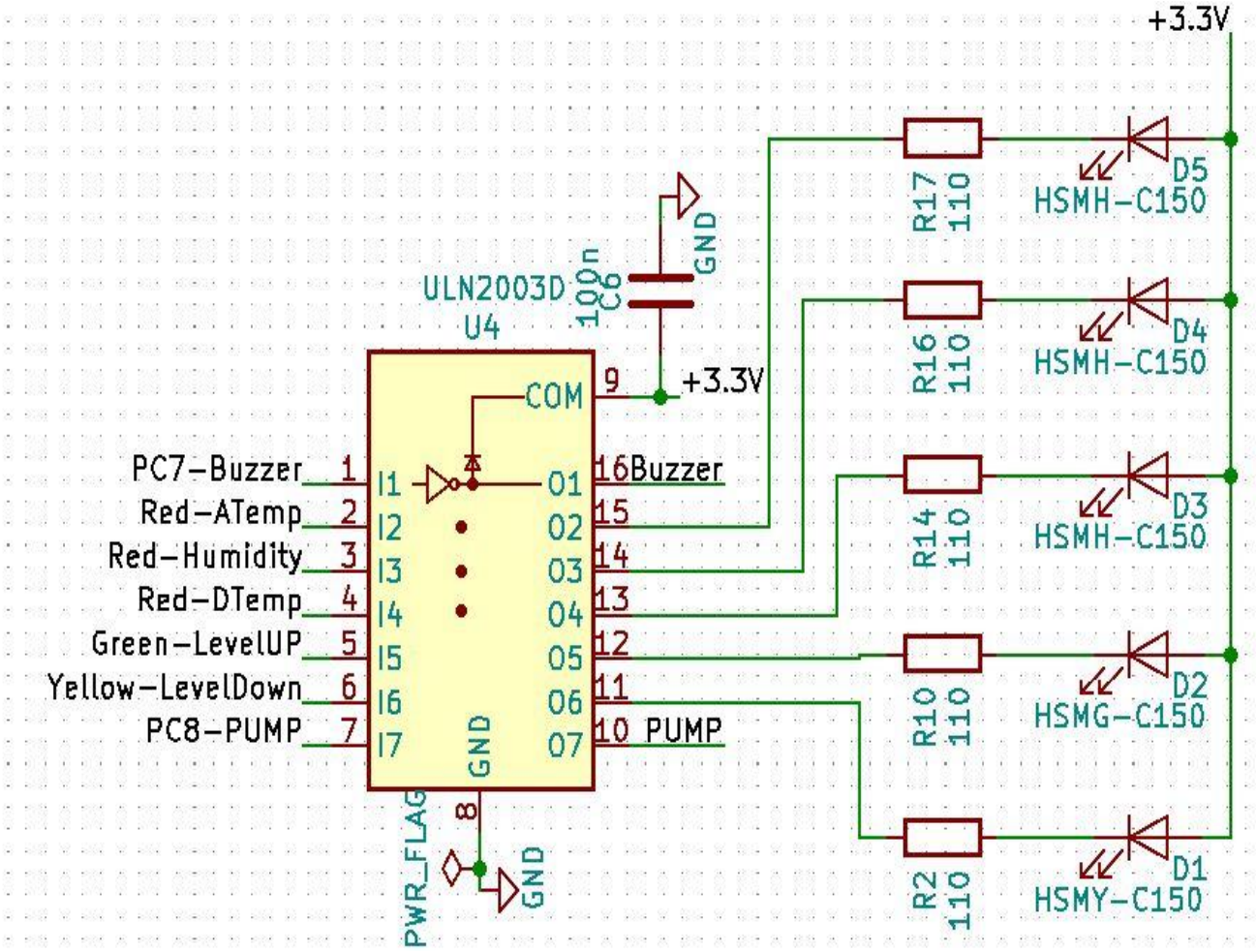


Figure 30: LED Schematic

3.1.9.1 Description

LEDs are used to show the status of sensors. LEDs are not connected directly to microcontroller but isolated through transistors. Similarly pump and buzzer are also connected to microcontroller through transistors. More specifically Darlington pair has been used.

3.1.9.2 HSMx-C150 LED

Five LEDs are connected in series with small value resistances to Darlington pair. These resistances are used to give optimal voltages to LEDs.

3.1.9.3 ULN2003D

The ULN2003[20] are high-voltage (50V), high-current Darlington arrays each containing seven open collector Darlington pairs with common emitters. Each channel is rated at 500 mA and can withstand peak currents of 600 mA. Suppression diodes are included for inductive load driving and the inputs are pinned opposite the outputs to simplify board layout.

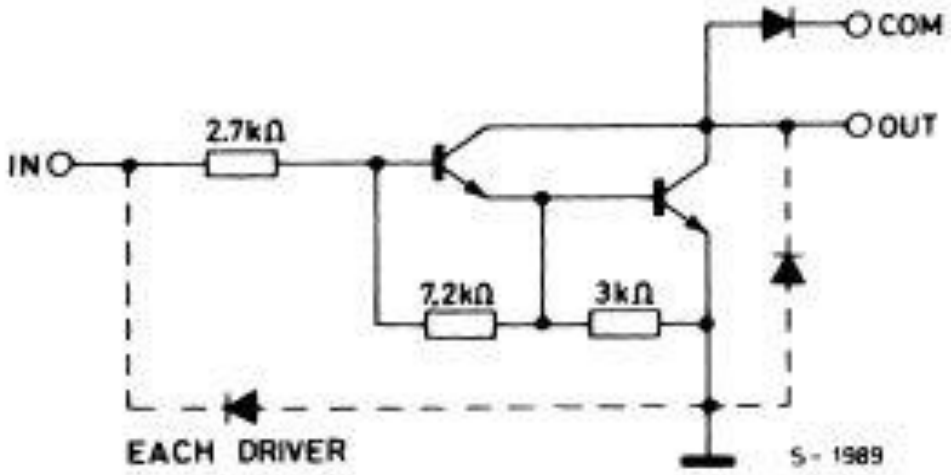


Figure 31: ULN2003D (each driver)

A Darlington pair is two transistors that act as a single transistor but with a much higher current gain. This mean that a tiny amount of current from a sensor, micro-controller or similar can be used to drive a larger load for example in this project a fan. Transistors have a characteristic called current gain. This is referred to as its hFE. The amount of current that can pass through the load in the circuit above when the transistor is turned on is:

Load current = input current x transistor gain (hFE)

The current gain varies for different transistors and can be looked up in the data sheet for the device. For a normal transistor this would typically be about 100. This would mean that the current available to drive the load would be 100 times larger than the input to the transistor. In some applications the amount of input current available to switch on a transistor is very low. This may mean that a single transistor may not be able to pass enough current required by the load.

Now as Load current equals the input current x the gain of the transistor (hFE). If it is not possible to increase the input current, then the gain of the transistor will need to be increased. This can be achieved by using a Darlington Pair. A Darlington Pair acts as one transistor but with a current gain that equals:

Total current gain (hFE total) = current gain of transistor 1 (hFE t1) x current gain of transistor 2 (hFE t2)

So, for example if there are two transistors with a current gain (hFE) = 100:

(hFE total) = 100 x 100

(hFE total) = 10,000

This gives a vastly increased current gain when compared to a single transistor. Therefore, this will allow a very low input current to switch a much bigger load current. Normally to turn on a transistor the base input voltage of the transistor will need to be greater than 0.7V. As two transistors are used in a Darlington Pair this value is doubled. Therefore, the base voltage will need to be greater than 0.7V x 2 = 1.4V. It will not be a problem in our case because microcontroller pin gives 3.3V.[21]

It is also worth noting that the voltage drops across collector and emitter pins of the Darlington Pair when the turn on will be around 0.9V. Therefore, if the supply voltage is 3.3V (as above) the voltage across the load will be around 2.2V (3.3V – 0.9V) which is optimal in case of HSMx C150 LEDs.

3.1.10 Key Pad Schematic

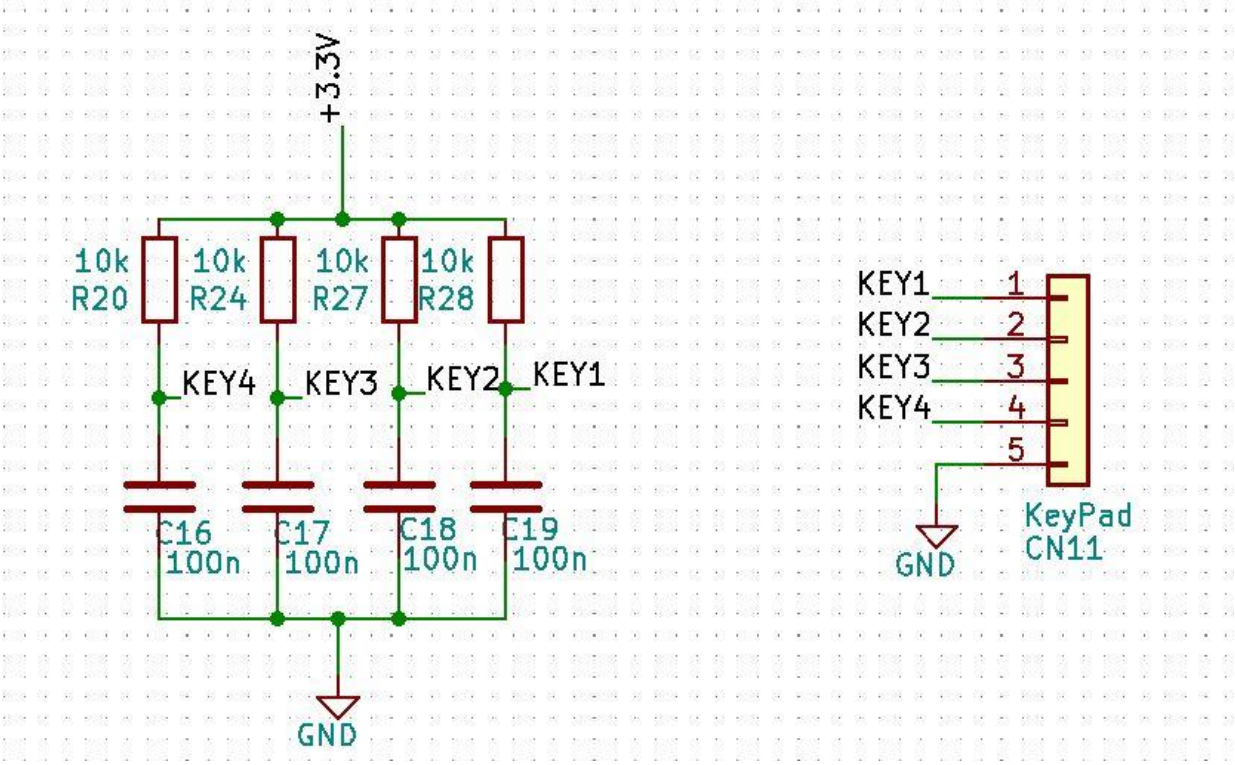


Figure 32KeyPad Schematic

3.1.10.1 Description

An external 4 button keypad has been used which is attached to connector CN11. The connector is attached to microcontroller GPIO pins according to the above shown circuitry. The resistors of 10kΩ are used to pull down the value when no button is pressed.

3.1.11 Programmer Schematic

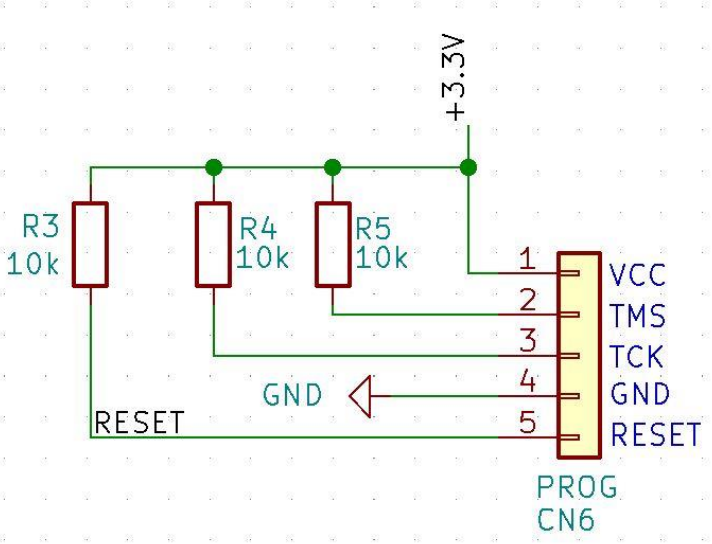


Figure 33: Programmer Schematic

3.1.11.1 Description

In order to program the microcontroller on the printed circuit board, an external programming device is required. Programmer is connected to CN6 connector. Pins of connector are connected to supply and ground. Other pins are connected to TMS, TCK and Reset of the microcontroller.

3.1.12 12V-5V Voltage Regulation Schematic

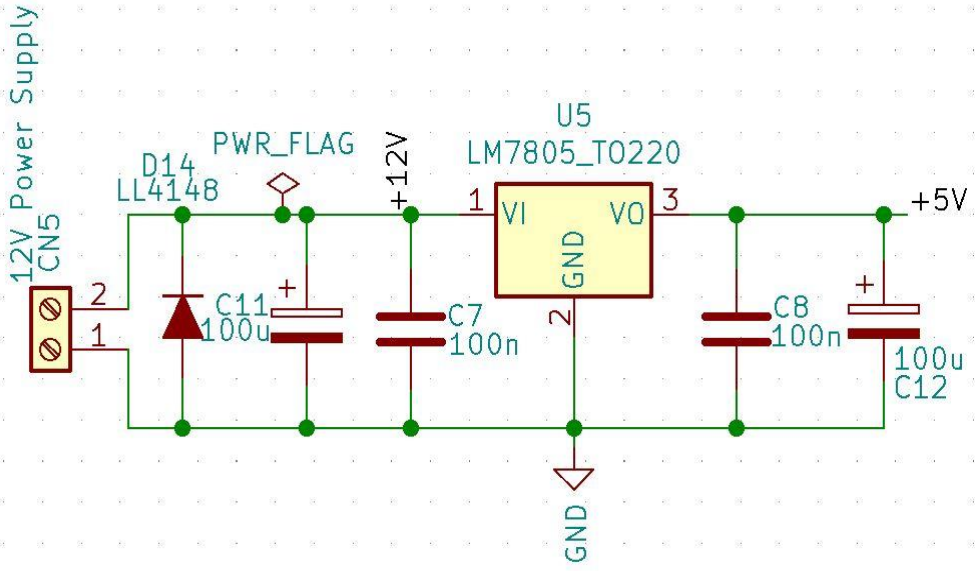


Figure 34: 5V Regulation Circuit

3.1.12.1 Description

In laboratory 12V power supply is available but many devices and components need 5V supply. So, in order to achieve 5V regulation circuit has been design as shown in the above figure. 12V power supply is connected to CN5 connector which is connected to LM7805 Voltage Regulator IC. Several capacitors have been used to reduce the ripples in the supply voltage and in the output voltage. A diode is used in reverse bias to avoid reverse current.

3.1.12.2 LM7805

Voltage sources in a circuit may have fluctuations resulting in not providing fixed voltage outputs. A voltage regulator IC maintains the output voltage at a constant value. 7805 IC, a member of 78xx series of fixed linear voltage regulators used to maintain such fluctuations, is a popular voltage regulator integrated circuit (IC). The xx in 78xx indicates the output voltage it provides. 7805 IC provides +5 volts regulated power supply with provisions to add a heat sink. IC ratings are:

- Input voltage range 7V- 35V
- Current rating Ic = 1A
- Output voltage range VMax=5.2V ,VMin=4.8V

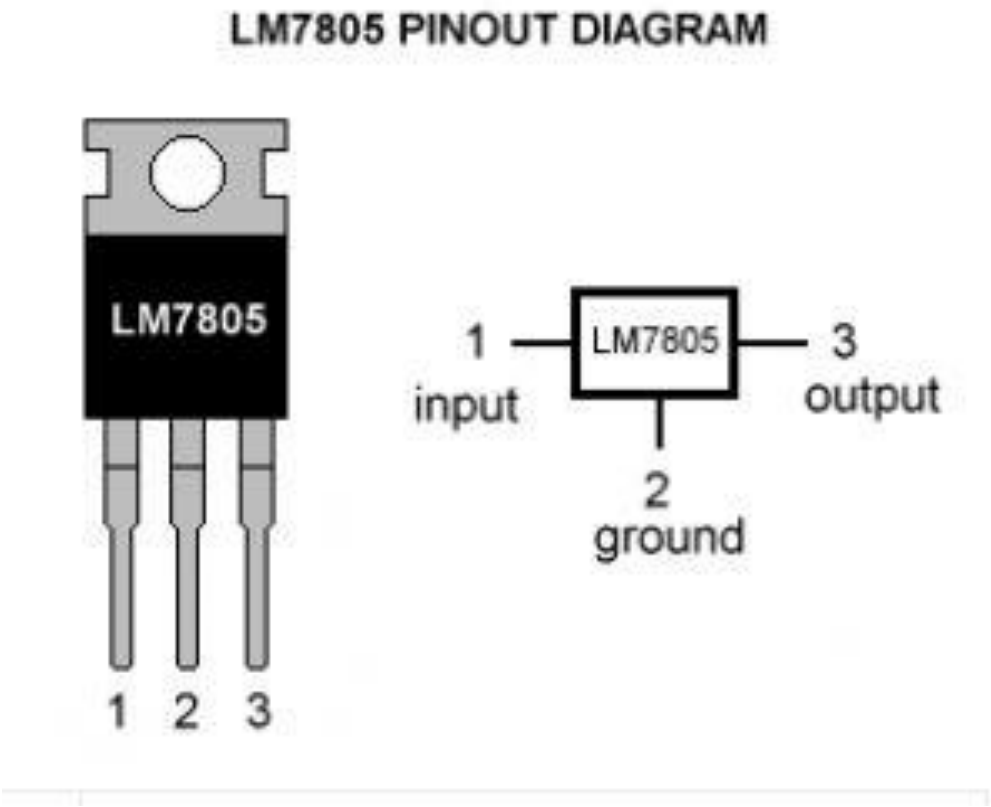


Figure 35: LM7805 PINOUT

Pin No.	Pin	Function	Description
1	INPUT	Input voltage (7V-35V)	In this pin of the IC positive unregulated voltage is given in regulation.
2	GROUND	Ground (0V)	In this pin where the ground is given. This pin is neutral for equally the input and output.
3	OUTPUT	Regulated output; 5V (4.8V-5.2V)	The output of the regulated 5V volt is taken out at this pin of the IC regulator.

Table 5: LM7805 Pins

It can be noticed, there is a significant difference between the input voltage & the output voltage of the voltage regulator. This difference between the input and output voltage is released as heat. The greater the difference between the input and output voltage, more the heat generated. If the regulator does not have a heat sink to dissipate this heat, it can get destroyed and malfunction. Hence, it is advisable to limit the voltage to a maximum of 2-3 volts above the output voltage. So, we now have 2 options. Either design the circuit so that the input voltage going into the regulator is limited to 2-3 volts above the output regulated voltage or place an appropriate heatsink, that can efficiently dissipate heat.

3.1.12.2.1 Schematic of LM7805 IC

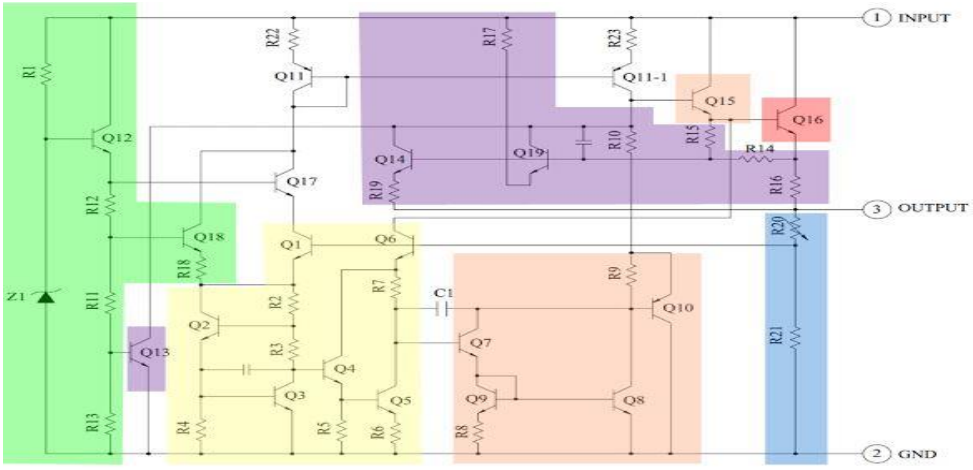


Figure 36: LM7805 schematic

The heart of the 7805 IC is a transistor (Q16) that controls the current between the input and output and thus controlling the output voltage. The bandgap reference (yellow) keeps the voltage stable. It takes the scaled output voltage as input (Q1 and Q6) and provides an error signal (to Q7) for indication if the voltage is too high or low. The key task of the bandgap is to provide a stable and accurate reference, even as the chip's temperature changes.

The error signal from the bandgap reference is amplified by the error amplifier (orange). This amplified signal controls the output transistor through Q15. This closes the negative feedback loop controlling the output voltage. The startup circuit (green) provides initial current to the bandgap circuit, so it doesn't get stuck in an "off" state. The circuit in purple provides protection against overheating (Q13), excessive input voltage (Q19) and excessive output current (Q14). These circuits reduce the output current or shutdown the regulator, protecting it from damage in case of a fault. The voltage divider (blue) scales down the voltage on the output pin for use by the bandgap reference.[22]

3.1.13 3.3V Regulator

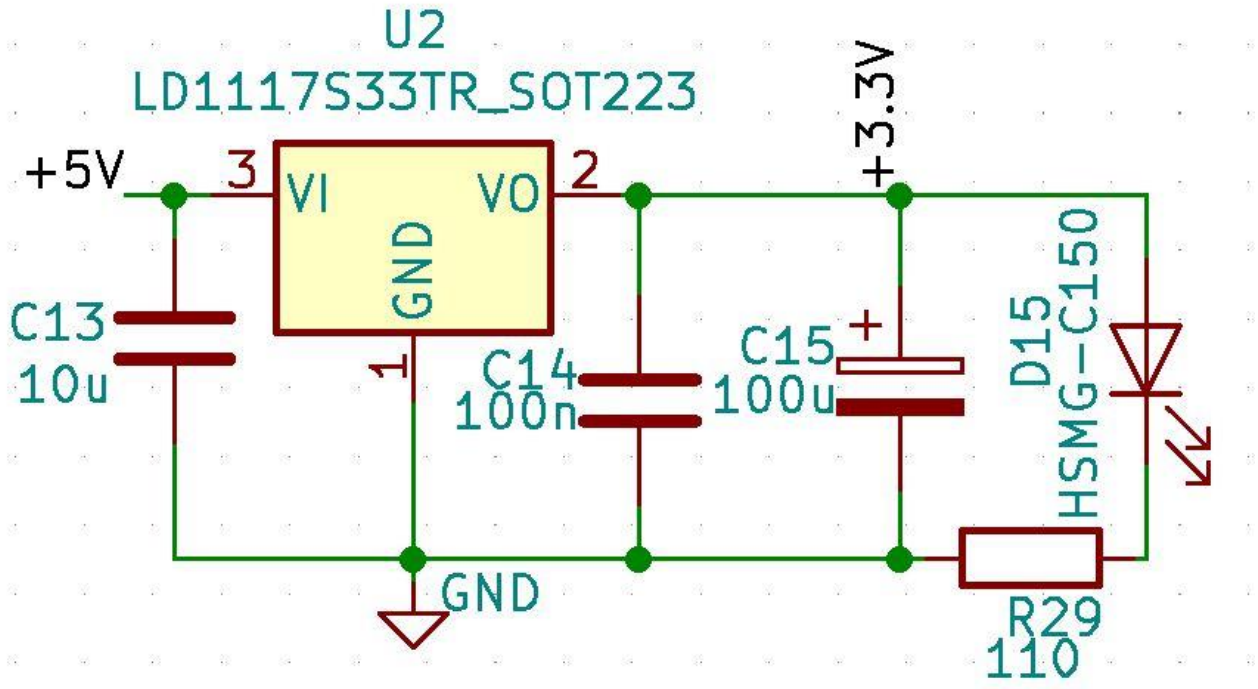


Figure 37: 3.3V Regulator Circuit

3.1.13.1 Description

STM microcontroller needs 3.3V supply. So LD1117S33[23] regulator IC is used to get 3.3V from 5V. Capacitor before the regulator and after it has been used to remove the ripple in DC voltage. A green HSMG C150 LED has been used to indicate when power in on.

3.1.13.2 LD1117

The LD1117 is a LOW DROP Voltage Regulator able to provide up to 800mA of Output Current, available even in adjustable version (Vref=1.25V). Concerning fixed versions, are offered the following Output Voltages: 1.2V,1.8V,2.5V,2.85V, 3.0V 3.3V and 5.0V. Here it is used for 3.3V.

SOT-223

Figure 38: LD1117

3.1.14 Switch Button Schematics

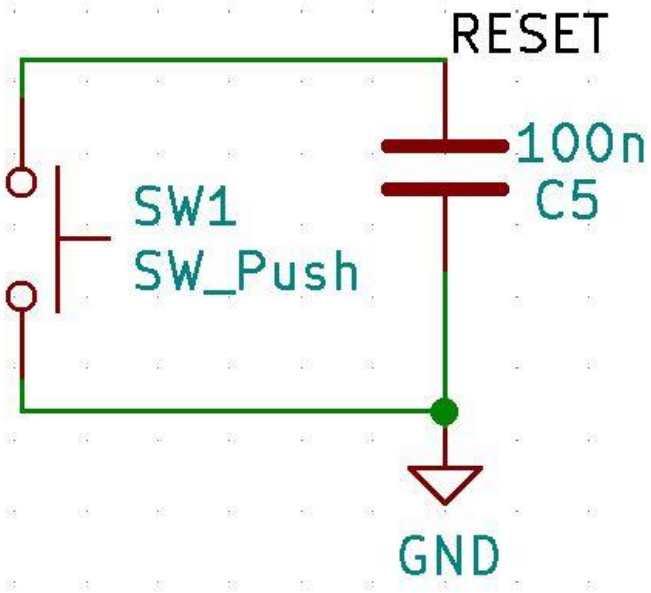
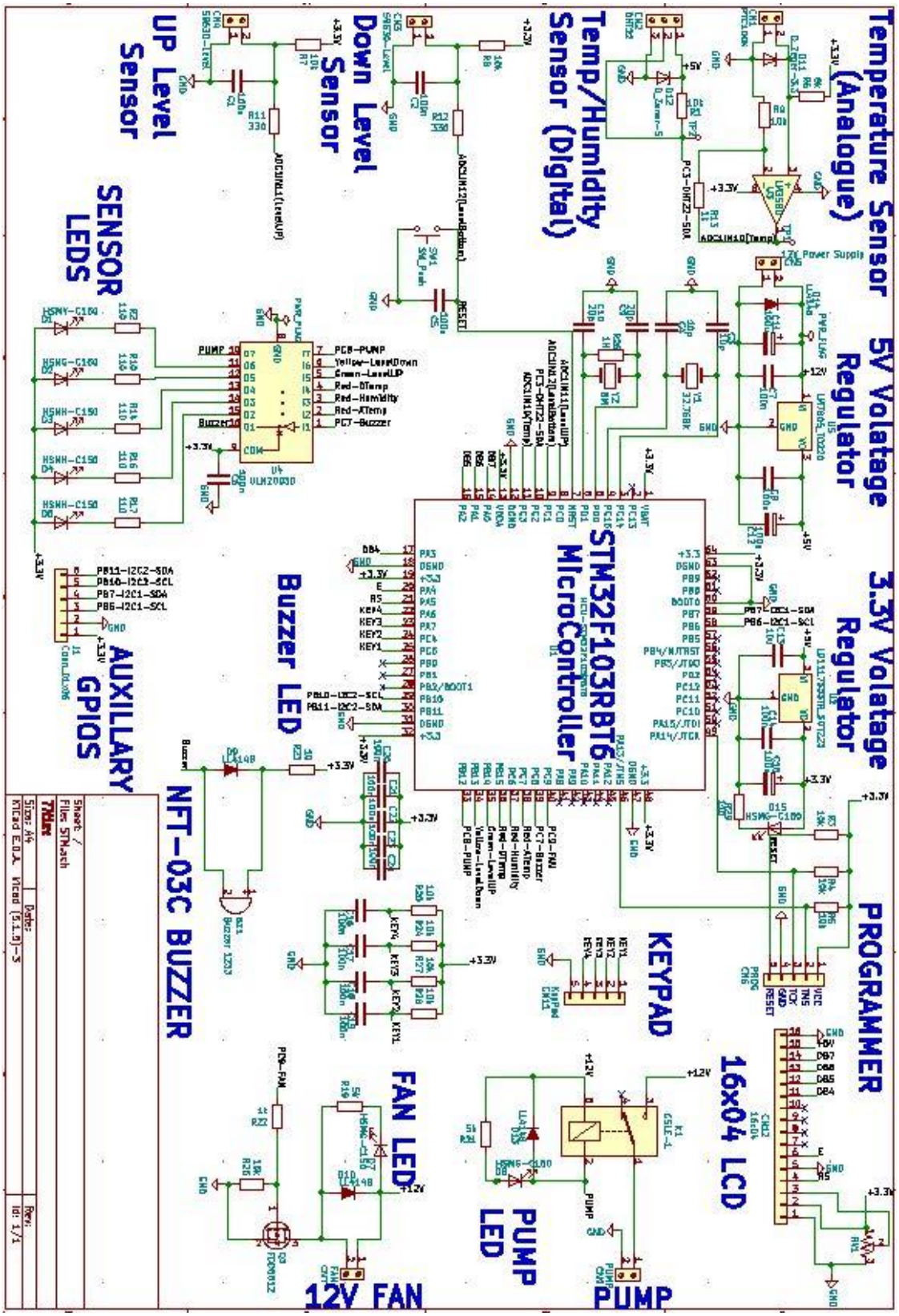


Figure 39: Switch Button Schematic

3.1.14.1 Description

Switch button is used to reset the Micro Controller. Capacitor is used to smoothen the signal.

3.1.15 Complete Schematic



3.2 PCB Layout

Once the schematic of the project has been designed on the KiCad, printed circuit board needs to be designed in order to assemble the electronic components on the board. KiCad allows to make a PCB layout. Main steps of PCB design are:

- Footprints Design
- Defining the size of the board
- Placing the components on the board according to their footprints and schematic
- Clean Routing
- 3D view

3.2.1 Footprints Design

First and the foremost step of PCB Design is to design the footprints of all the components that would be used in the project. KiCad has an extensive library which has footprints of most of the very commonly used components. In the datasheet of the components, footprints are defined in terms of dimensions. If the components footprints are not available in libraries, they need to be designed manually. There is a footprint editor section in KiCad where new footprints are designed. The footprints used in this project are shown in the diagram below.

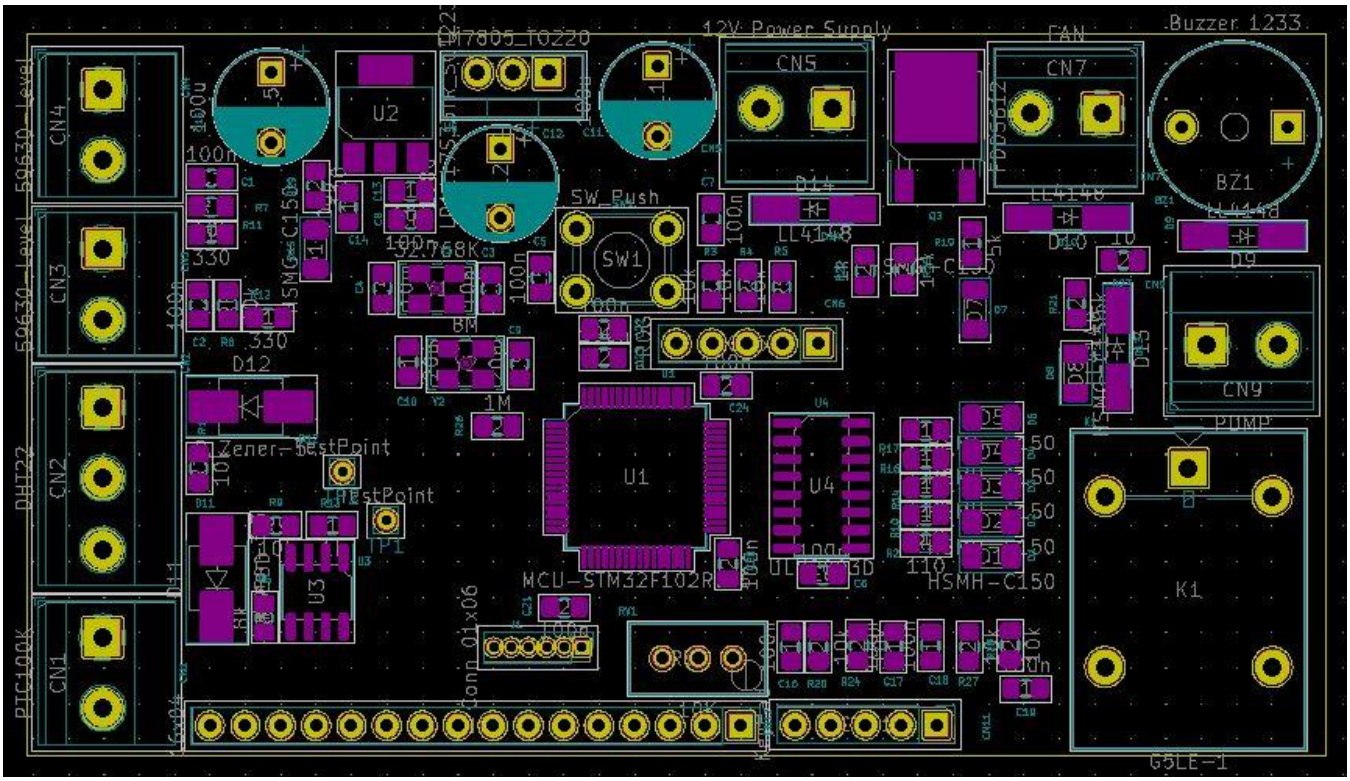


Figure 40: Footprints

3.2.2 Size of the board

After analyzing the size of the components, the board size used in this project is 93mm x 52mm.

3.2.3 Components Placing

Components are placed on the PCB keeping in mind the schematic structure build on KiCad schematic. Another important factor kept in notice while placing the component is the routing connections. There should be enough space between components for routing. General rule of thumb is that micro controller should be at the centre of the board as most of the connections to the components come from controller.

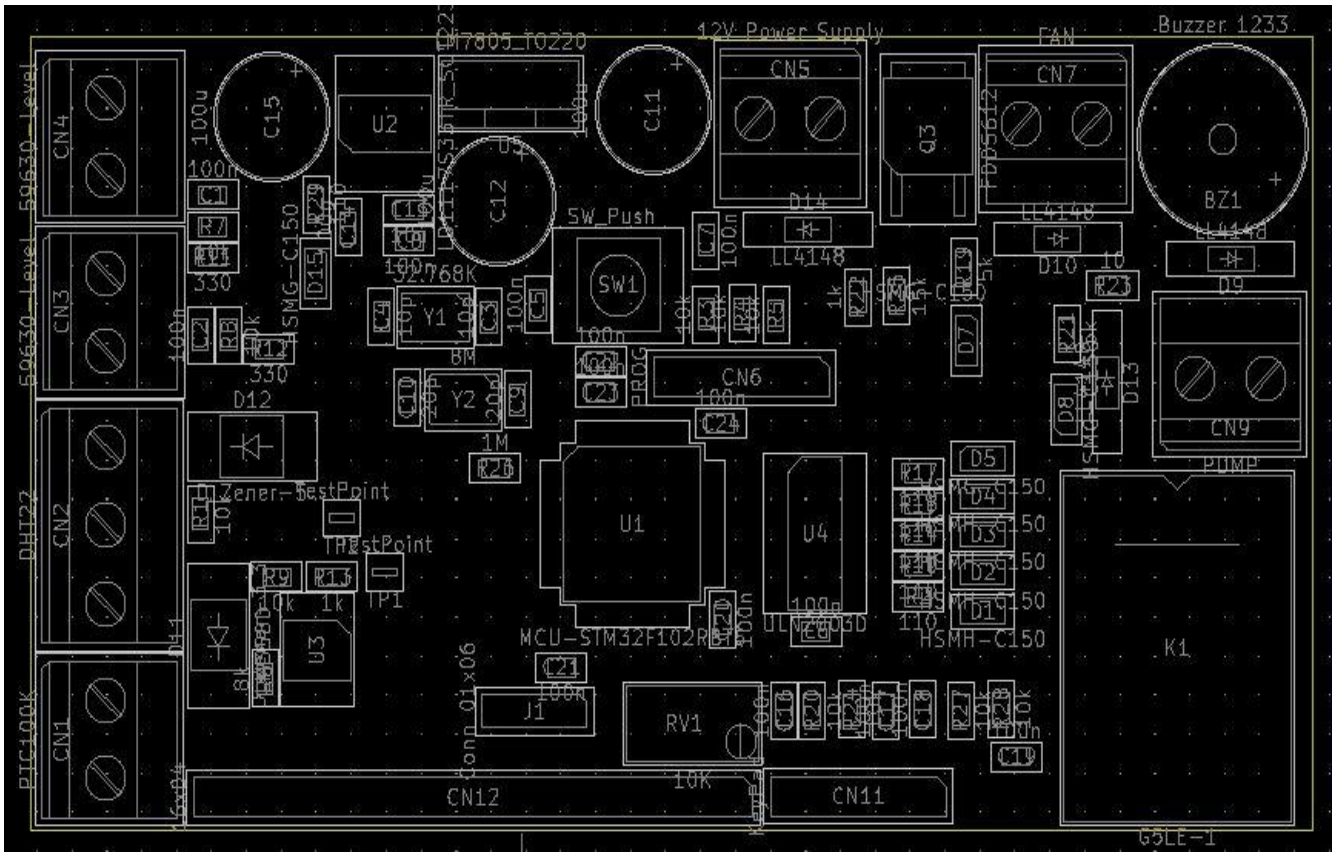


Figure 41: Components Placement

It can be seen in the PCB Layout; all sensor connectors are placed on the left side of the board. On the right of each connector their respective components are placed keeping in mind the common connections and aesthetic look of the board. At the center of the board, U1 is the STM microcontroller. Below the controller CN12 connector is for LCD and on the right CN11 connector is for Keypad. Above LCD connector there is J1 connector for I2C pins used for IMU module. Exactly on the right of microcontroller there is U4 which is ULN2003D Darlington IC and right to the IC there are LEDs. On the extreme right bottom K1 is Relay for the pump and above it is the connector for the pump at extreme top right there is buzzer BZ1 and on the left of buzzer, the connector CN7 is for Fan. On the top of microcontroller, there is push button and power supplies. On the top left of micro controller external oscillators are placed.

3.2.4 Routing

Once components are placed on the PCB, next step is to route the connections. Thumb rule is to leave all supplies and ground and connect all other connections. Once connections other that supplies are done, connect supplies directly if possible or through vias. As ground is widespread across the board, lower and upper sides are completely filled with ground and therefore all ground connections are either done directly if possible or through vias to the downside of the board.

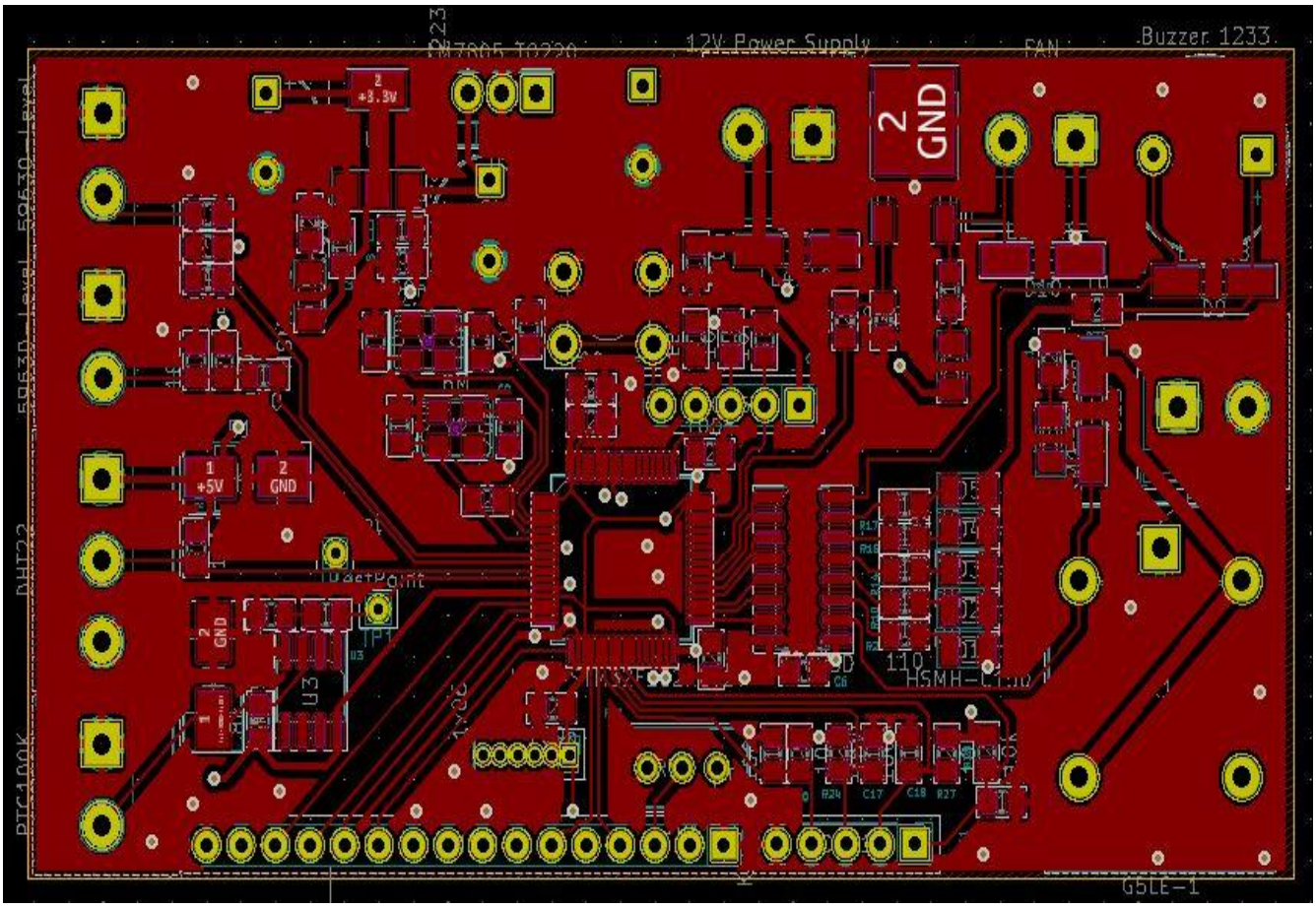


Figure 42: Front Side Copper

The above diagram is the front side of the board showing copper connections (Red) between various pads. Copper connections are separate for each label. Apart from the normal connections all other Red part is grounded. Connections are made always with some angle different than 90 degree. It is considered a good practice. Other important thing to note is routes are not made in between two consecutive pads even if it is allowed be CAD software.

In the figure below back side of the Printed Circuit Board is shown. Green color shows the copper connections. There are vias present which connect one side of the board to the other side of the board. Use of vias is very important as it reduces the complexity of routing and from aesthetic point of view, board appears more elegant. As mentioned before both upper and bottom side is filled with ground copper, therefore all the green connections other than connections of components and supplies is ground

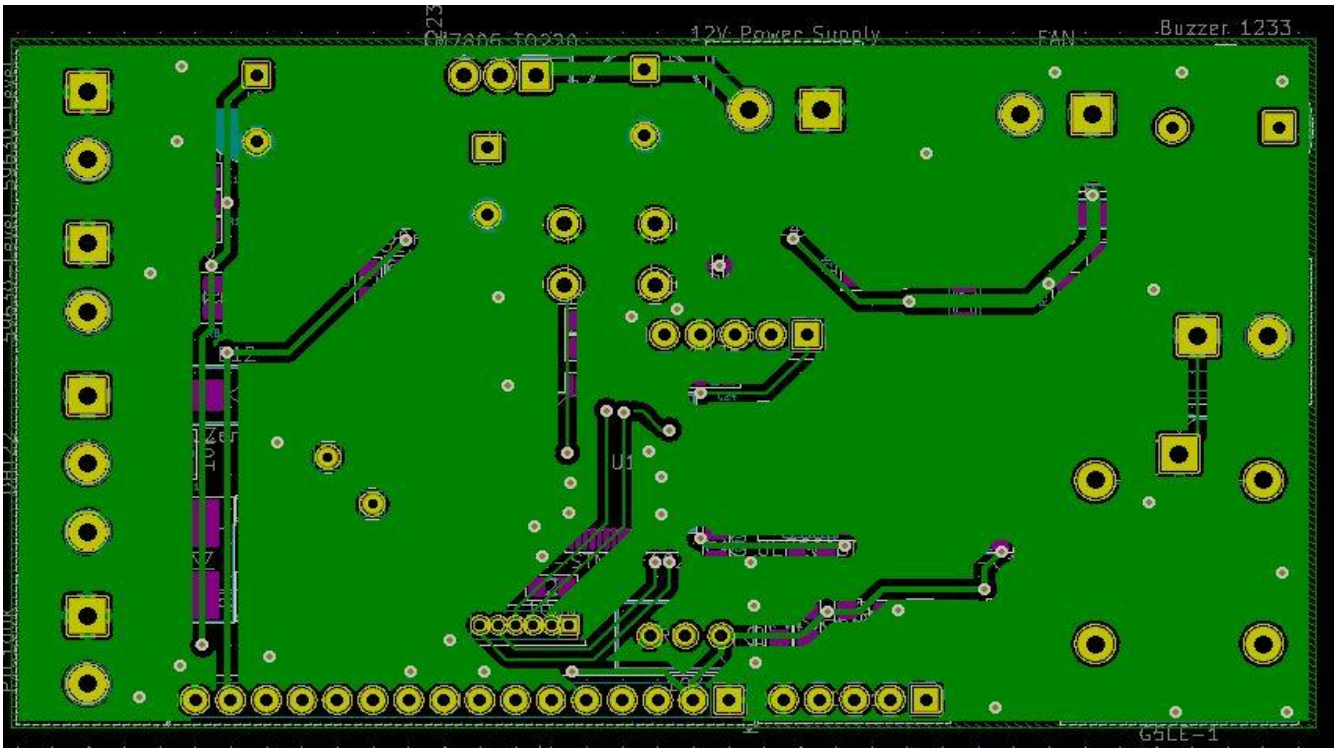


Figure 43: Back Side Copper

3.2.5 3D View

There is a feature available in KiCad which allows to view the printed circuit board as it would appear in Real World. It is called 3D Viewer. While designing PCB this feature is very useful because designer can see the real component as it is placed. So, if component seems to be not in the right place one can change its position so that it looks more elegant.

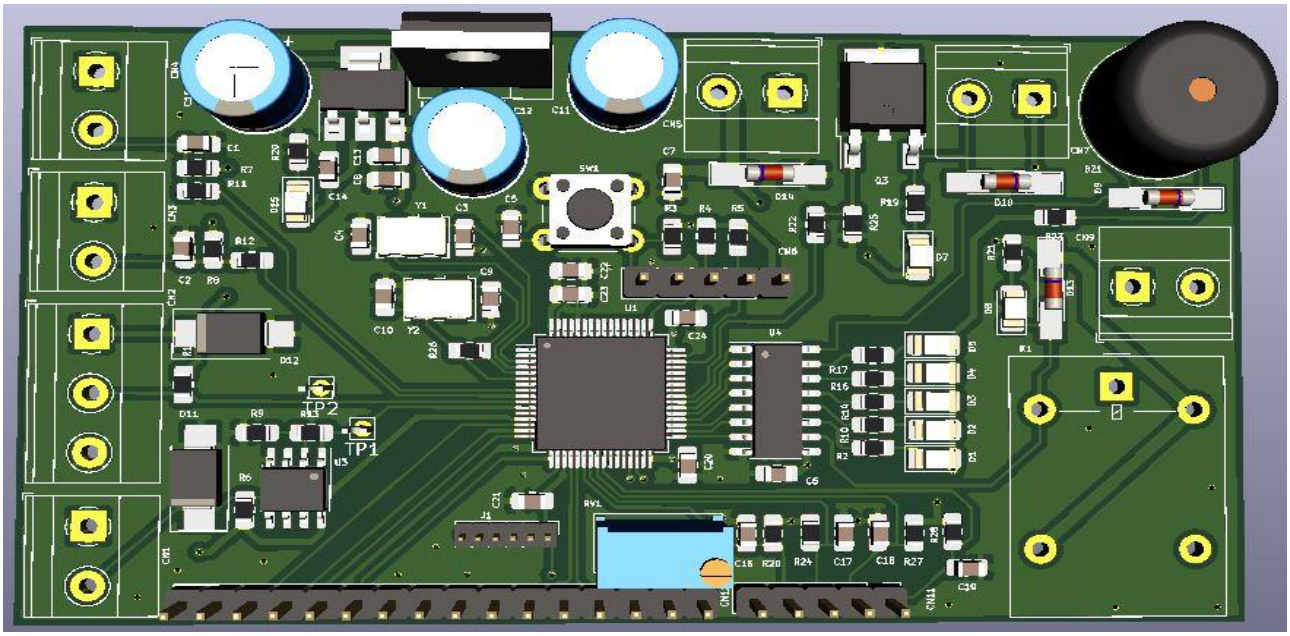


Figure 44: 3D View front

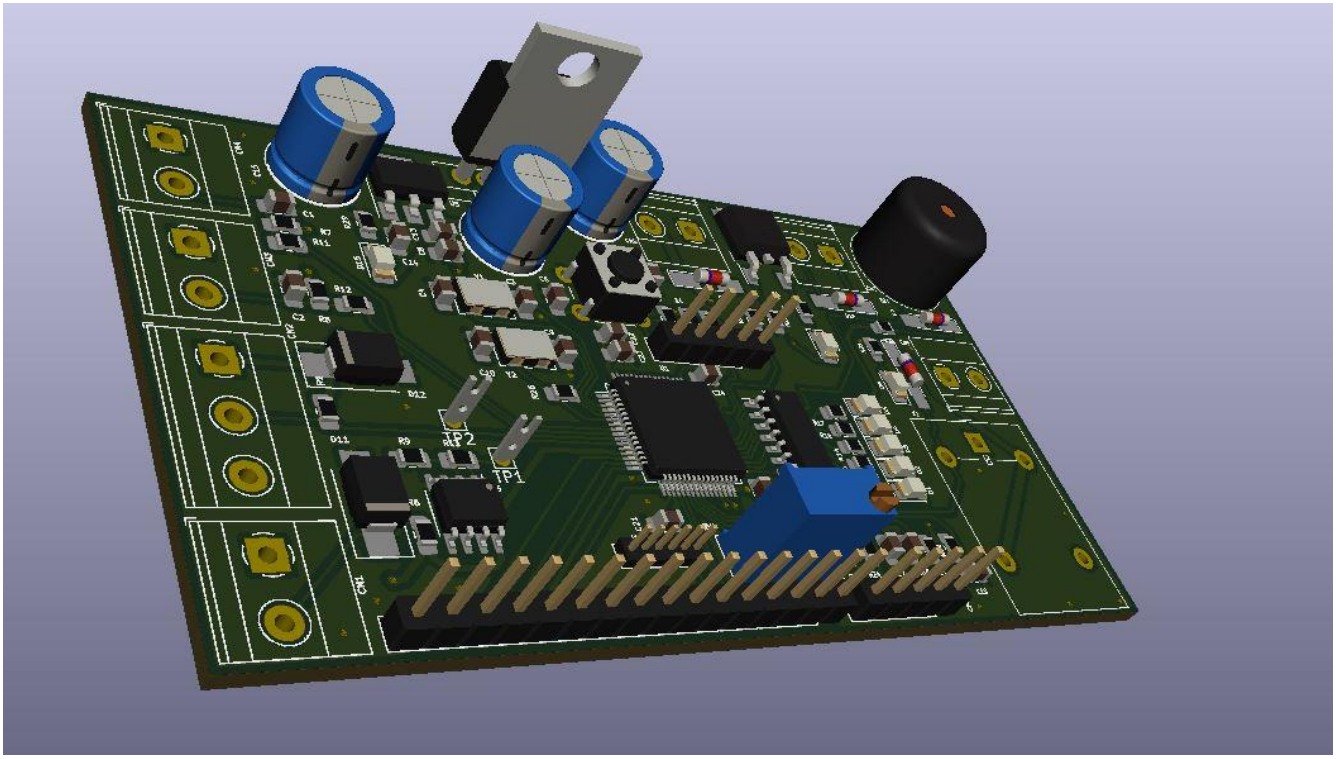


Figure 45: 3D View Tilt

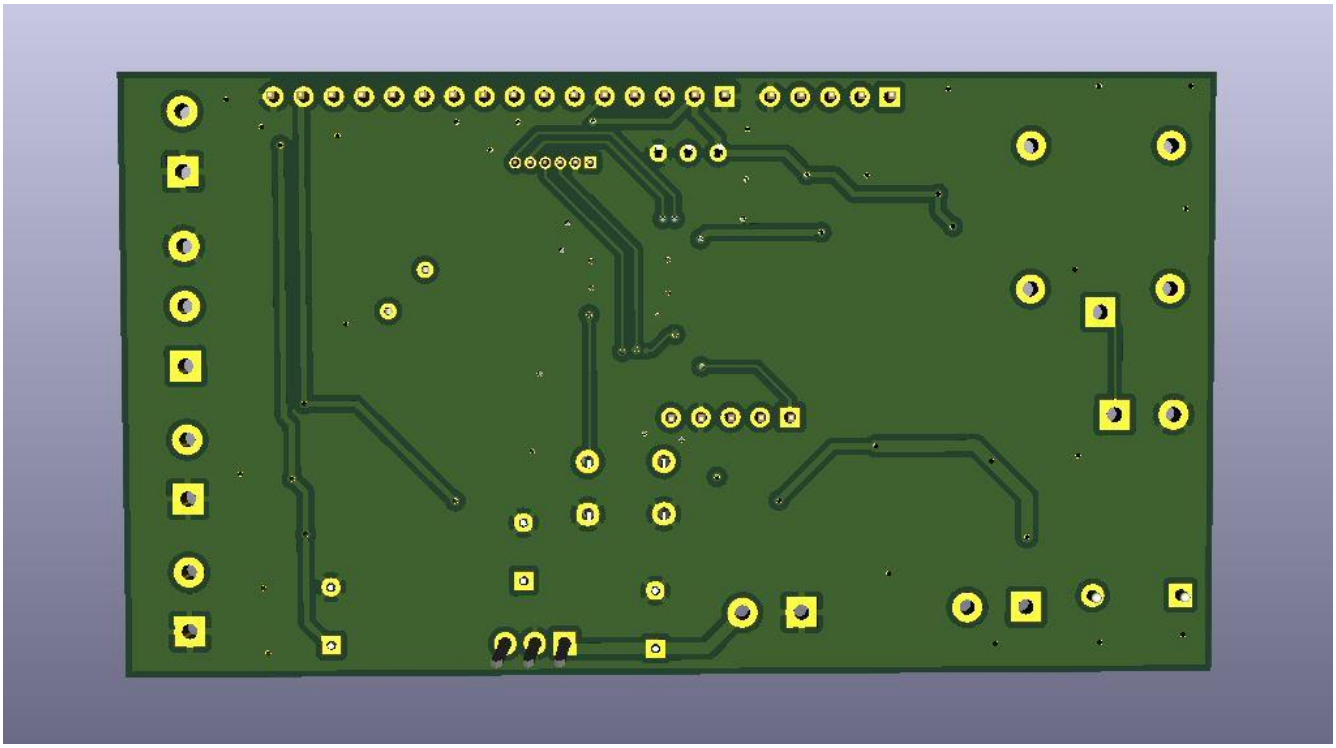


Figure 46: 3D View Back

3.3 GERBERS

The Gerber format is an open ASCII vector format for printed circuit board (PCB) designs. It is the de facto standard used by PCB industry software to describe the printed circuit board images: copper layers, solder mask, legend, drill data, etc. Gerber is used in PCB fabrication data. PCBs are designed on a specialized electronic design automation (EDA) or a computer-aided design (CAD) system. The CAD systems output PCB fabrication data to allow fabrication of the board. This data typically contains a Gerber file for each image layer (copper layers, solder mask, legend or silk).[24][25] Typically, all these files are "zipped" into a single archive that is sent to the PCB bare board fabrication shop. The fabricator loads them into a computer-aided manufacturing (CAM) system to prepare data for each step of the PCB production process. Below are the various Gerber files generated by KiCad for this project:

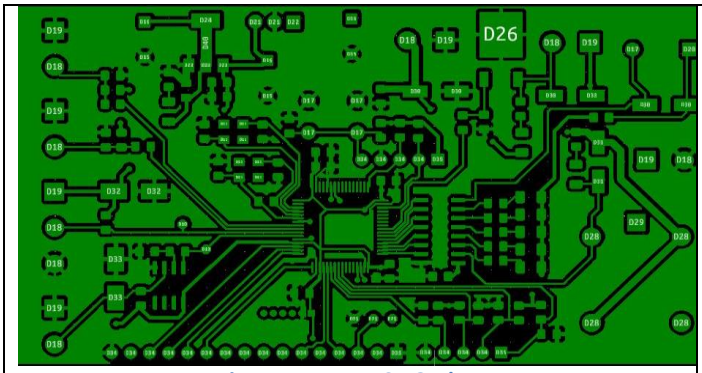


Figure 47: Front Cu Gerber

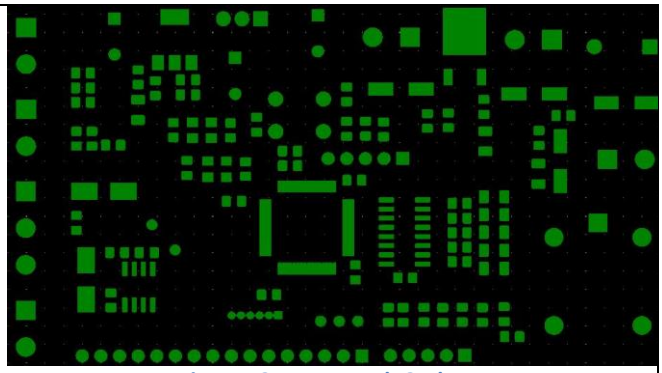


Figure 48: Front Mask Gerber

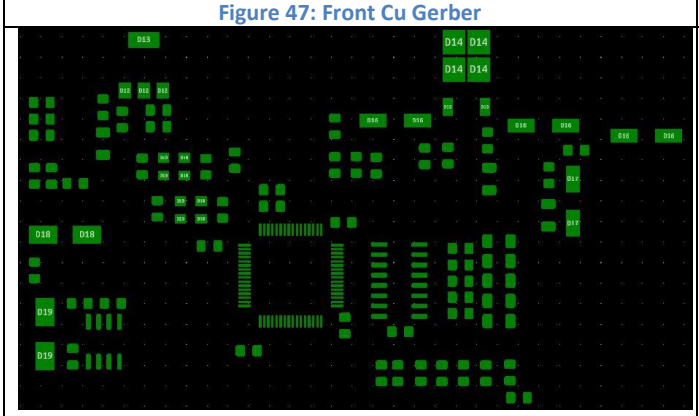
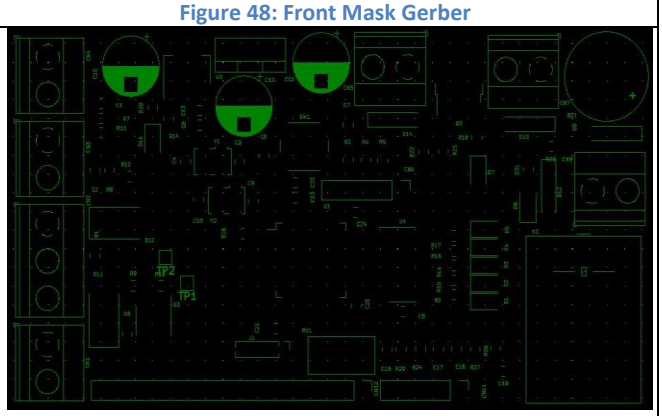


Figure 49: Front paste Gerber



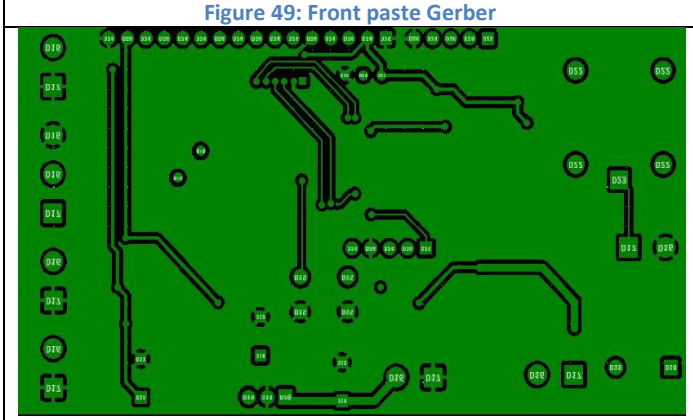


Figure 50: Back Cu Gerber

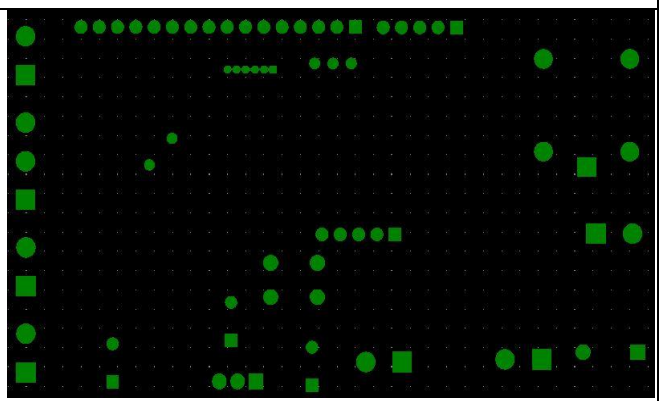


Figure 51: Back Mask Gerber

3.4 PCB Manufacturing

Once the Gerber files are sent to the manufacturer, after two weeks printed circuit board is ready. Following are the pictures of real PCB after manufacturing:

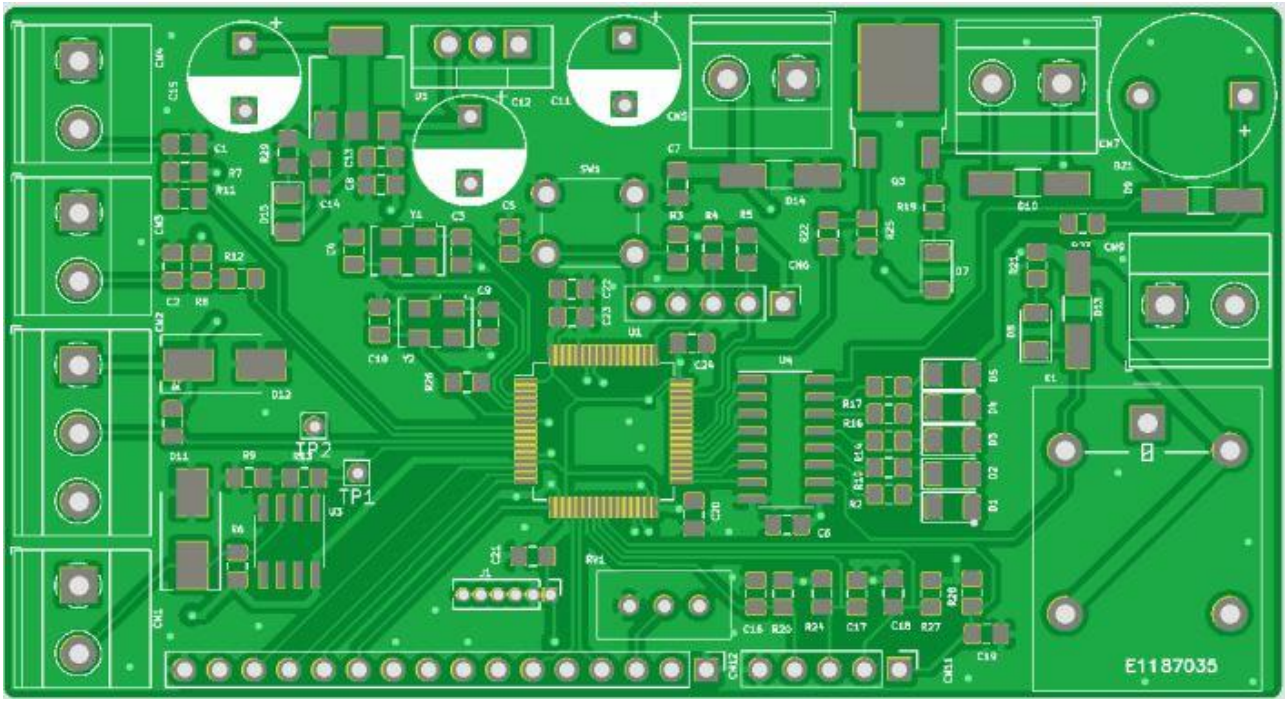


Figure 52: PCB Top View

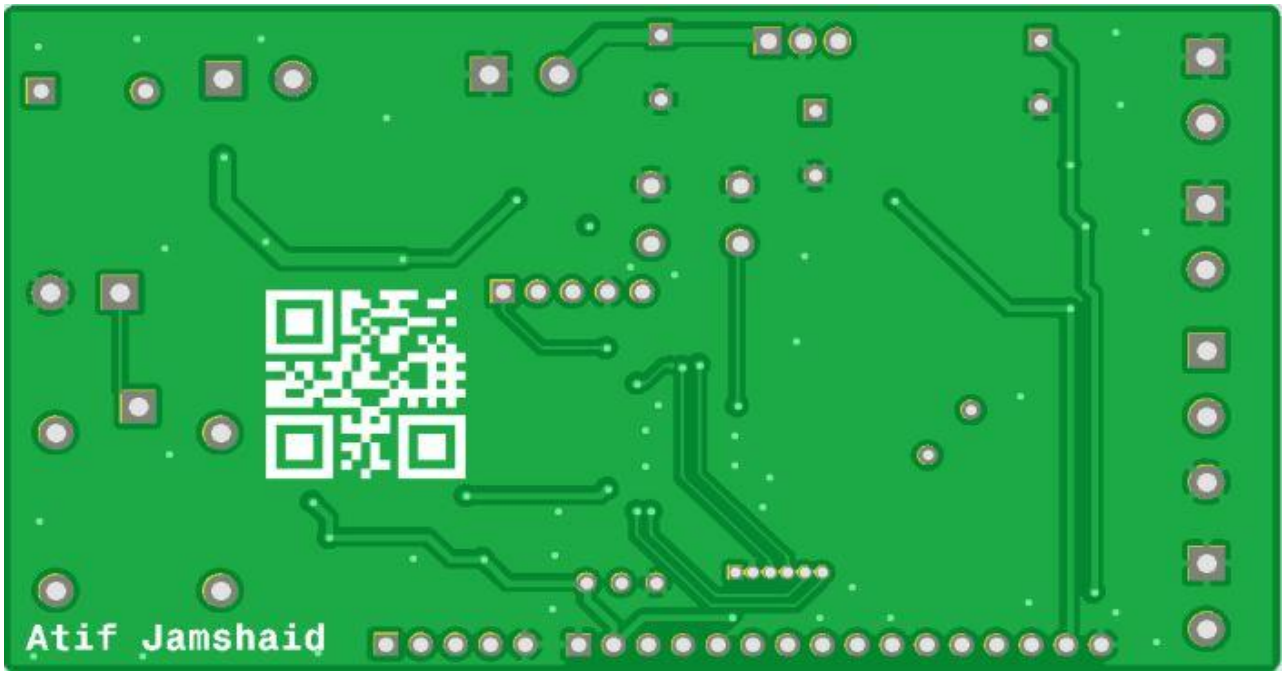


Figure 53: PCB Bottom View

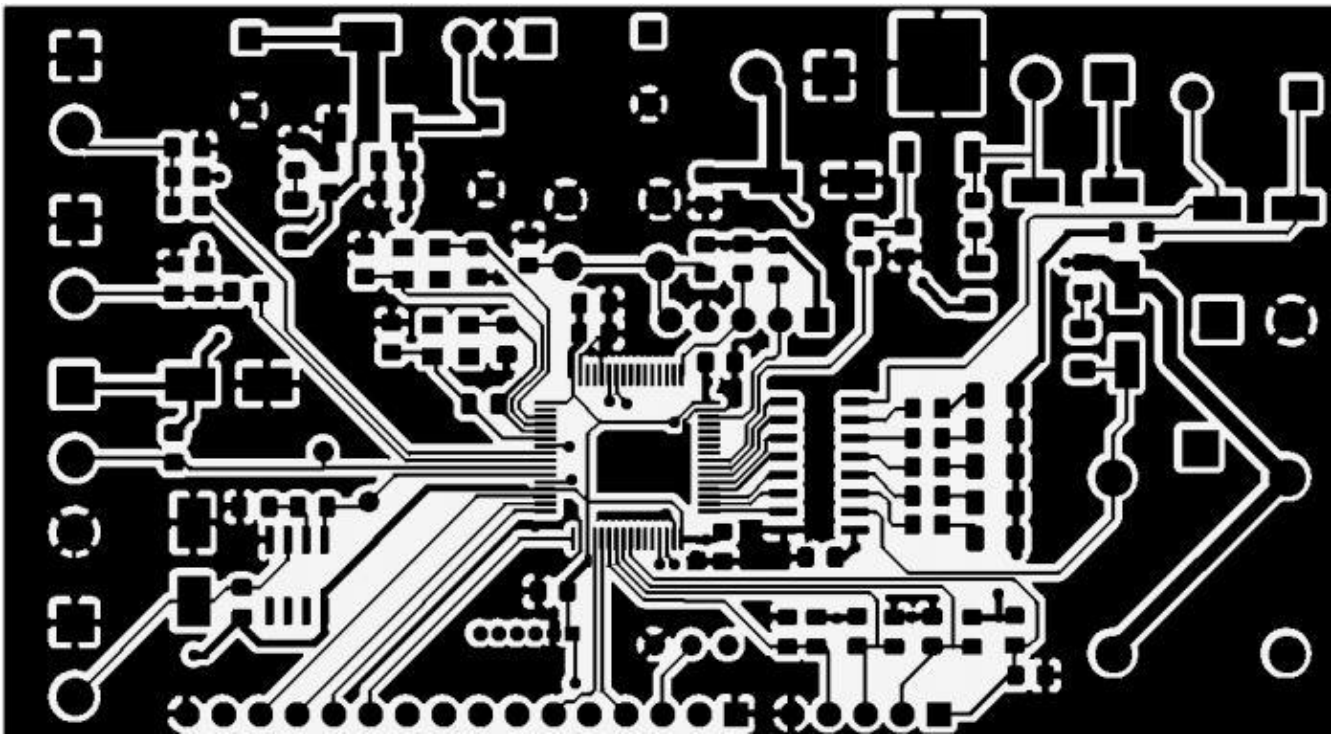


Figure 54: Top Layer Image

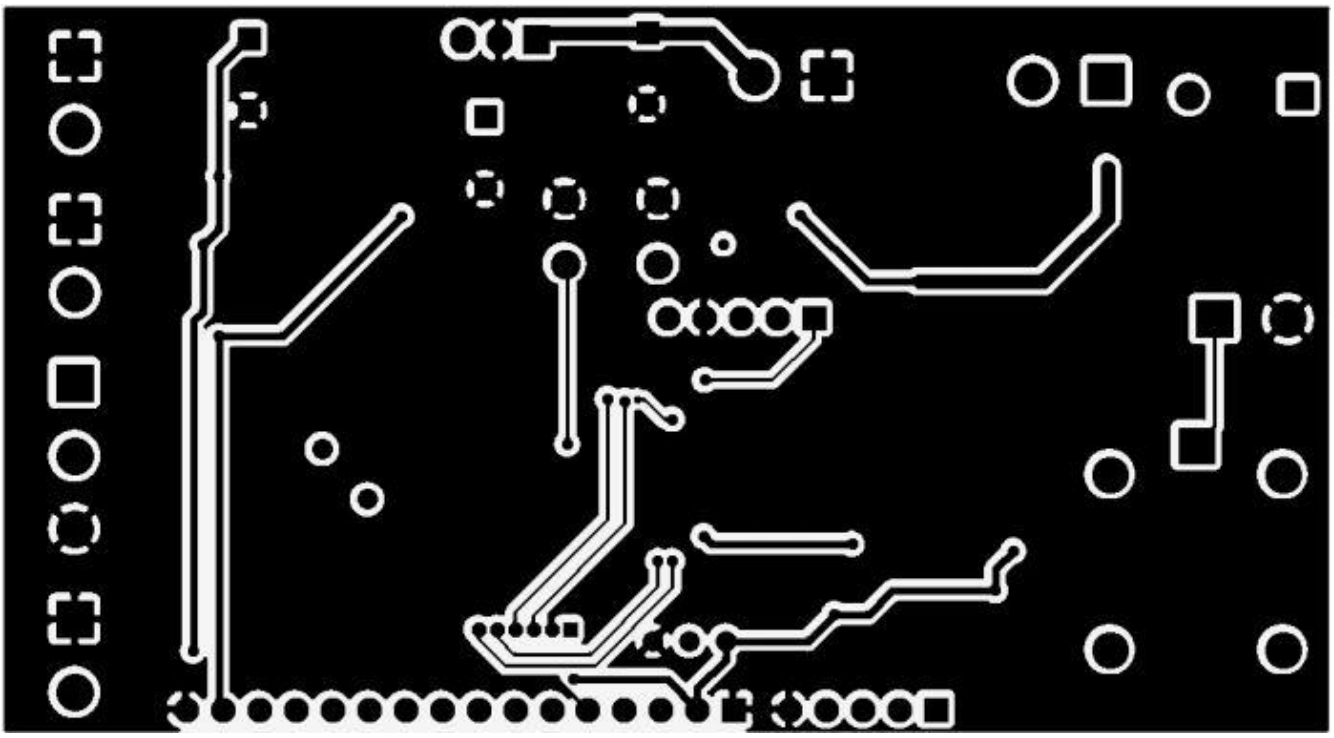
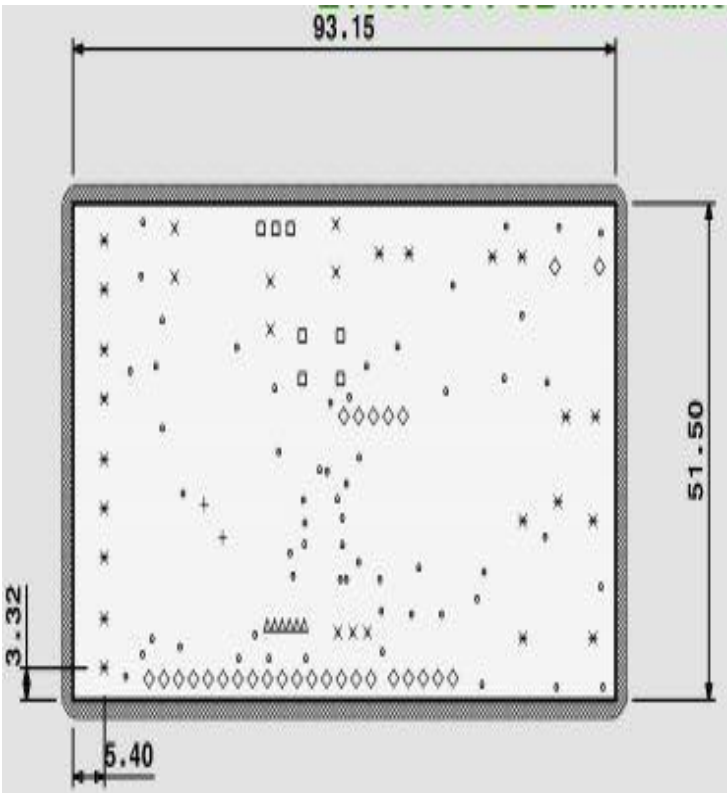


Figure 55: Bottom Layer Image



Layer: E1187035-60

Symbol	Type	Kind	End Dia (mm)	Tool Dia (mm)	Count	+ Tol	- Tol
◦	Via	Drill	0.40	0.50	58	0.10	0.10
△	Plated	Drill	0.65	0.75	6	0.10	0.10
+	Plated	Drill	0.70	0.80	2	0.10	0.10
×	Plated	Drill	0.80	0.90	9	0.10	0.10
◇	Plated	Drill	1.00	1.10	28	0.10	0.10
□	Plated	Drill	1.10	1.20	7	0.10	0.10
*	Plated	Drill	1.30	1.40	20	0.10	0.10

Figure 56: Top Mechanical view

3.5 Assembly

Once printed circuit board has arrived from the manufacturer, next is to assemble the components of the project on the board. Following is the final board look after it is assembled.

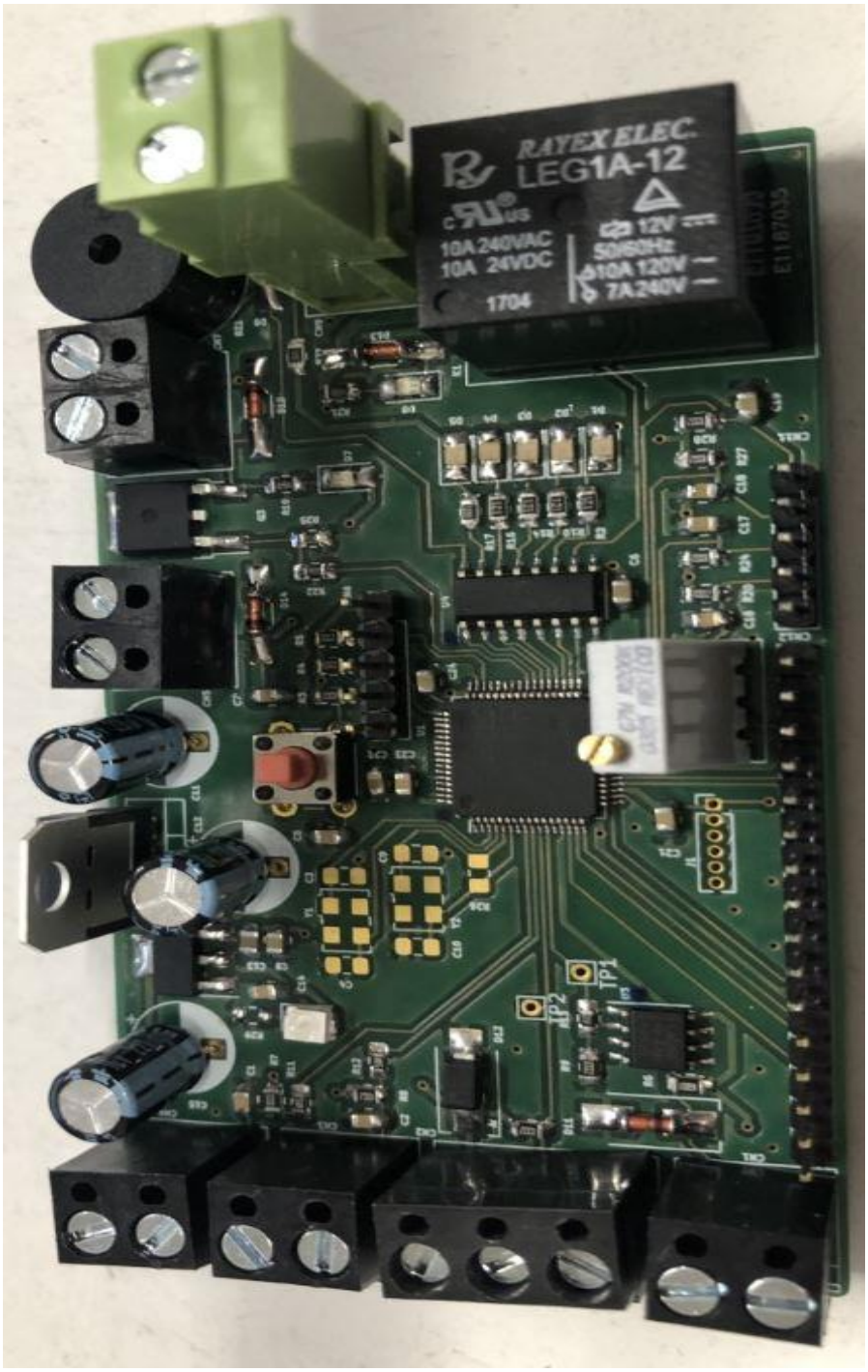


Figure 57: Final PCB

CHAPTER 4

4 CODING

After developing the computer aided hardware design, third phase of the project is coding. In this part, programming is done in order to achieve the objectives of the project. The programming language used for this project is C. Eclipse IDE is used as a developing environment software. STM32 Nucleo board[26] is used to program the code and debug it. All the separate modules and devices that are used in the project are discussed ahead. How these devices and modules are configured and how data is transmitted or received from these devices has been discussed in detail in this section.

4.1 Configurations

4.1.1 Analogue Temperature

Data received from NTC10K is a continuous data. In order to read this data Analogue to Digital converter should be used. There are several modes of ADC configuration available: Discontinuous mode, Continuous mode and Scan mode. Since we need only one single channel pin as there are no other ADC conversions required in this project discontinuous mode could be selected. Instead continuous mode has been used in circular configuration so that ADC channel is continuously scanned. In this mode ADC will acquire data nonstop unlike a single conversion where we trigger it if we want to get conversion and then go to sleep. Here ADC will continuously convert the data and send it to memory. In this mode it is a good idea to use timer trigger for the ADC so that the ADC does not go as fast as it can. It is a good mode when data is needed to be transferred to the memory using Direct Memory Access (DMA). DMA will take converted data straight from the ADC output data register and take them to the memory. Timer 3 has been used and its configuration is as follows:

- Prescaler = 800
- CounterMode=Up
- Period=1000
- AutoRelaodPreload=Disable
- Clock running frequency = 8 MHz

Time = (Prescaler * Period) / Clock Running Frequency = 0.1s

This implies ADC will convert data after every 0.1s. Now ADC configuration parameters are following:

- Resolution = 12 bits
- Mode = SCAN
- Data Alignment = Right
- Sampling Time = 239.5 ADC clock cycles

DMA request setting are following:

- Data Width = WORD
- Mode = Circular

ADC global interrupt has been enabled so that after every sequence of conversion ADC will interrupt and could do operations like blink LED or read the data. Finally, PC3 pin which can be used as ADC1 Input 13, has been used for getting data from NTC10K. If we look in the schematic of the microcontroller it is pin 11.

4.1.2 Level Measurement

For level measurement two simple floating devices have been used. Whenever liquid comes in contact with the devices, switch closes and output switches from 0 to 1. Therefore, we do not need any Analogue to digital converter in this case. Simple configuration of two GPIOs as inputs is required. For this project PC0 and PC1 are used to take inputs from the floating devices. They are pin 8 and pin 9 of the microcontroller. Therefore, following configurations are done for PC0 and PC1

- Mode = Input
- Pull = Pull Down

4.1.3 DHT22 Measurement

As discussed in previous sections, DHT22 is a digital-output temperature and humidity sensor/module. It uses single one wire interface. It has simple connections which requires 3.3V, ground and connect a data line to a digital input or output on the STM because we need to swap between input and output for data to go in both directions

Figure 58: DHT22 with 3 Pin Connector

As there is only one data line so PC2 (Pin 10) has been configured as GPIO output which will be later changed into input when data would be received from the sensor. Configuration of DHT22 can be done using the timing diagram available in the data sheet[6] of the device.

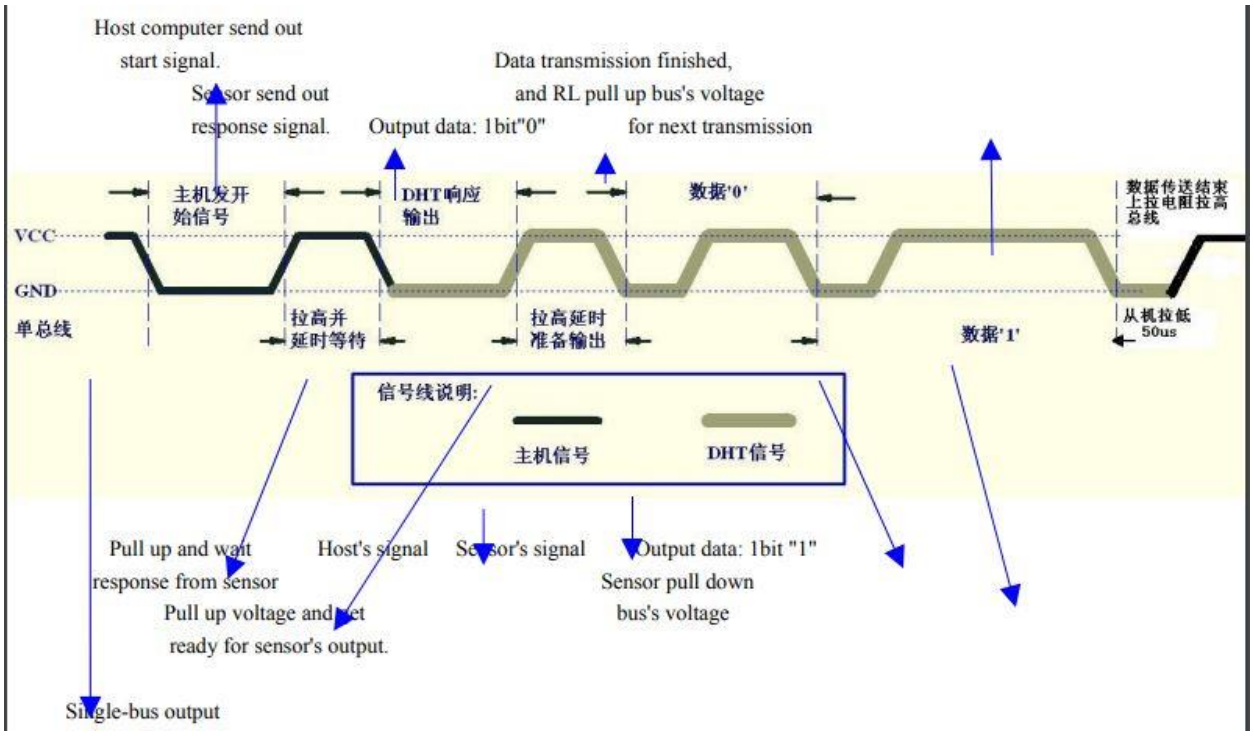


Figure 59:DHT22 Overall Timing Diagram

4.1.3.1 *Step 1: MCU send out start signal to DHT22*

Data-bus's free status is high voltage level. When communication between MCU and DHT22 begin, program of MCU will transform data-bus's voltage level from high to low level and this process must beyond at least 1ms to ensure DHT22 could detect MCU's signal, then MCU will wait 20-40us for DHT22's response. At this point after waiting GPIO Pin Mode changed from output to input.

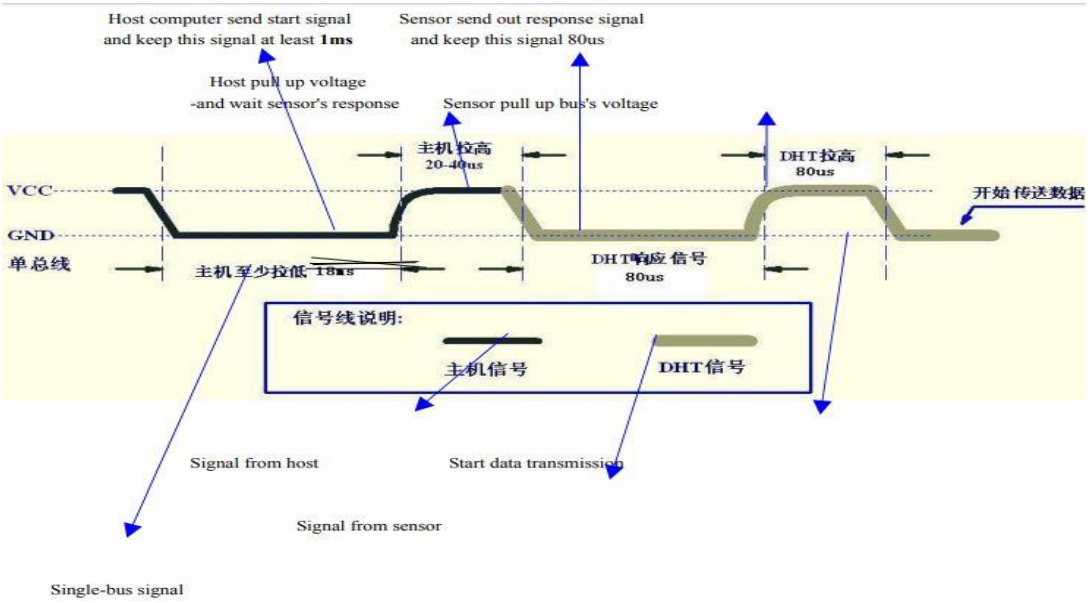


Figure 60: Step 1 DHT22

4.1.3.2 Step 2: DHT22 send response signal to MCU

When DHT22 detect the start signal, DHT22 will send out low-voltage-level signal and this signal last 80us as response signal, then program of DHT22 transform data-bus's voltage level from low to high level and last 80us for DHT22's preparation to send data.

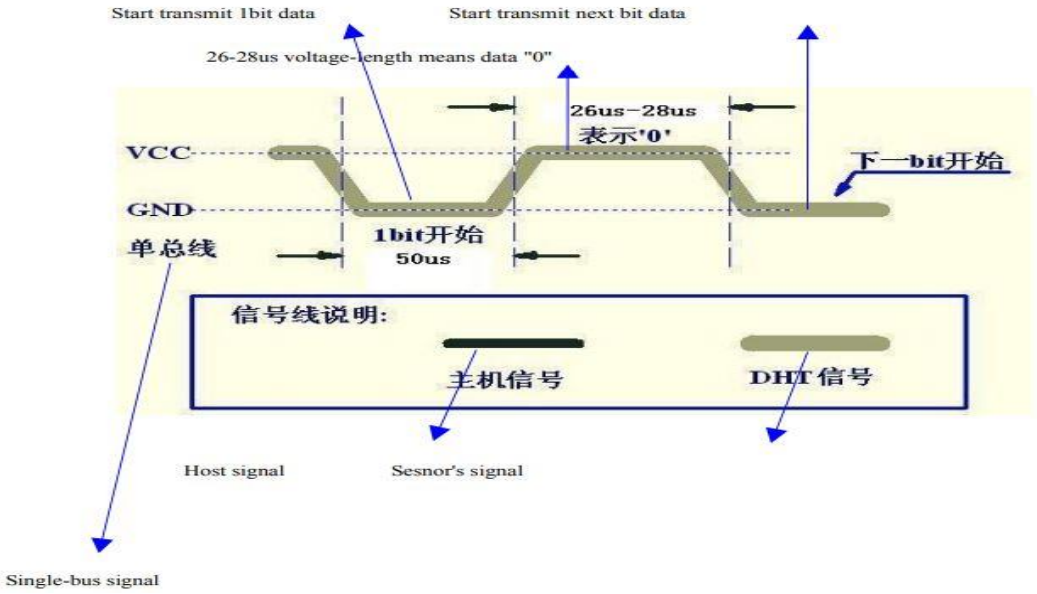


Figure 61: Step 2 DHT22

4.1.3.3 Step 3: DHT22 send data to MCU

When DHT22 is sending data to MCU, every bit's transmission begin with low-voltage-level that last 50us, the following high-voltage-level signal's length decide the bit is "1" or "0".

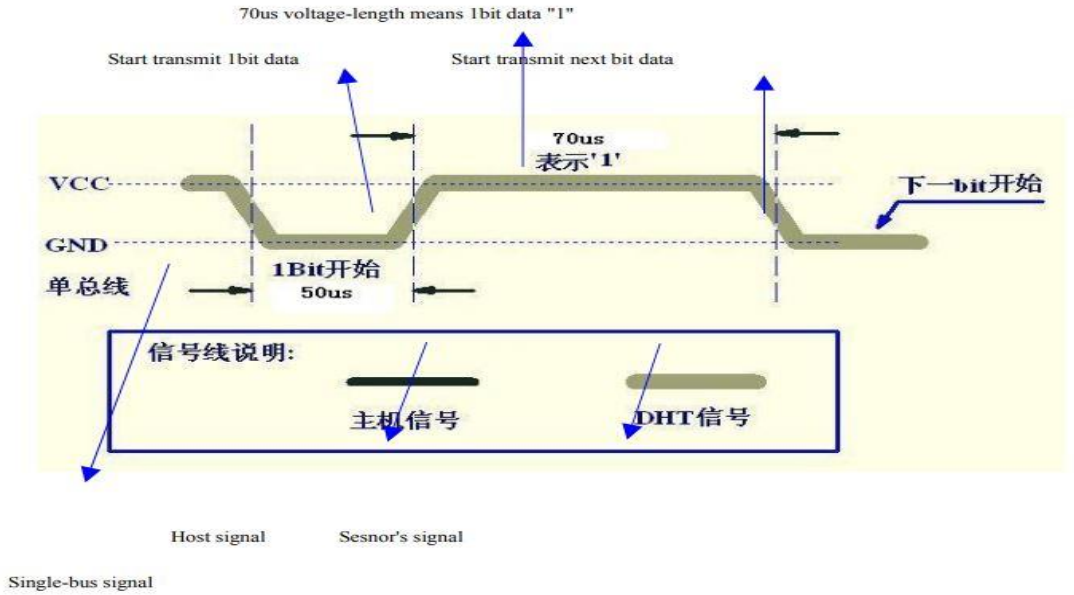


Figure 62: Step 3 DHT22

4.1.4 IMU Configuration

4.1.4.1 GY521/MPU6050

In order to measure the level of water tank, it has to be straight. If there is any tilt, then level measurement is not accurate. There GY521/MPU6050 sensor module is used in this project to measure if there is any tilt in the tank through the measurement of 3 axis accelerometer values. Basically, GY521 is a breakout board for the so called MPU6050 which is a micro electromechanical system. This device has

- 3 axis accelerometer
- 3 axis gyroscope
- Temperature sensor
- Digital Motion Processor

An accelerometer measures non gravitational acceleration whereas gyroscope uses earth gravity to measure orientation. The digital motion processor can be used to process many complex algorithms directly on the board. Usually it processes algorithms to turn the raw values from the sensors into stable position data. From the datasheet of MPU6050[27] it can be seen; it is a very powerful but complex device. Our scope of project only needs to extract raw sensor values from the device.

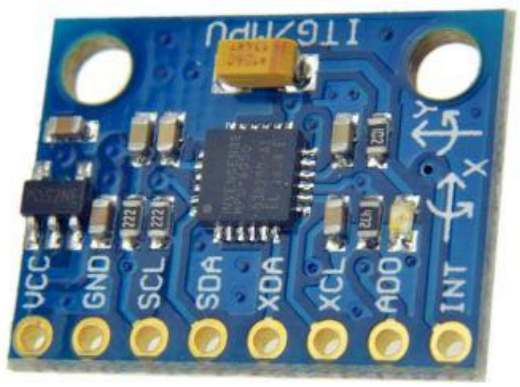


Figure 63: GY521/MOU6050

PIN	Description
VCC	Voltage Supply
GND	Ground
SCL	Serial Clock line of the I2C
SDA	Serial Data line of the I2C
XDA	Auxiliary Data line of the I2C
XCL	Auxiliary Clock line of the I2C
AD0	Address Pin
INT	Interrupt Pin

Table 6: GY521/MPU6050 Pins Description

Supply Voltage can be both 3.3V or 5V as there is an onboard voltage regulator. SCL and SDA are serial clock and data lines are part of I2C interface. These are used as I2C Slave while STM32 microcontroller as I2C Master. XDA and XCL are Auxiliary data and Auxiliary clock pins which are used to connect the module with external sensors. In this case module is used as I2C Master while external sensors are used as I2C. AD0 is the address pin. If it is low address is 0x68 and if it is high address is 0x69. Interrupt pin is used to handle the interrupts. In this project only first four pins are used.

4.1.4.2 I2C Interface

I2C[27] is a two-wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I2C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master. The MPU-60X0 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400 kHz. The slave address of the MPU-60X0 is b110100X which is 7 bits long. The LSB bit of the 7-bit address is determined by the logic level on pin AD0. This allows two MPU-60X0s to be connected to the same I2C bus. When used in this configuration, the address of the one of the devices should be b1101000 (pin AD0 is logic low) and the address of the other should be b1101001 (pin AD0 is logic high).

4.1.4.3 I 2C Communications Protocol

4.1.4.3.1 START (S) and STOP (P) Conditions

Communication on the I2C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below). Additionally, the bus remains busy if a repeated START (Sr) is generated instead of a STOP condition.

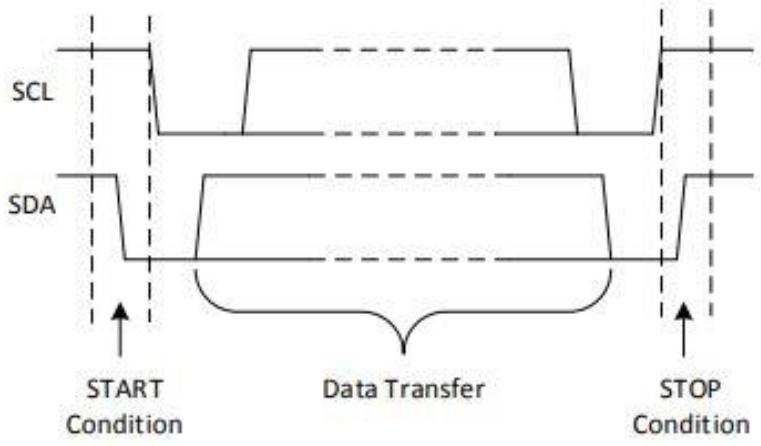


Figure 64: I2C START and STOP Condition

4.1.4.3.2 Data Format / Acknowledge

I 2C data bytes are defined to be 8-bits long. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse. If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (refer to the following figure).

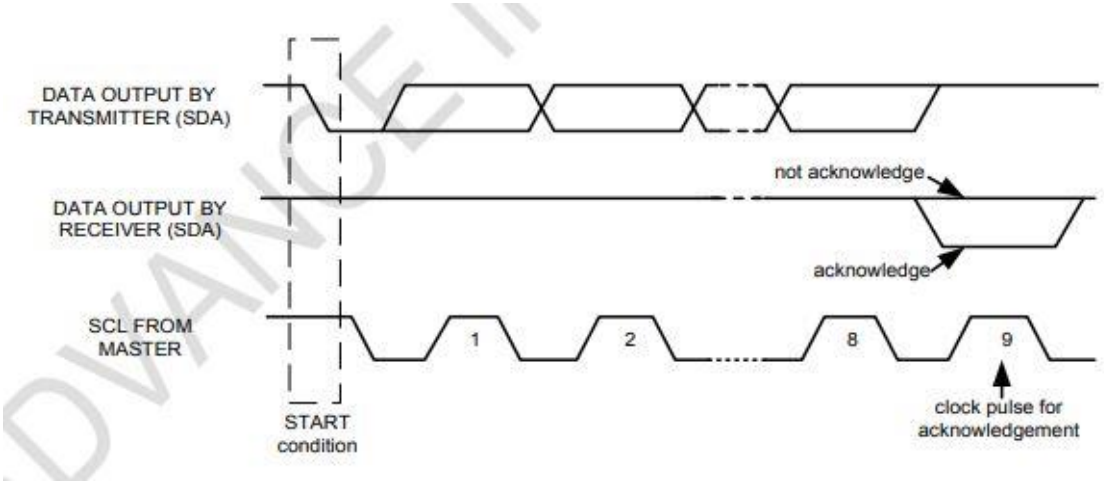


Figure 65: Acknowledge on the I2C Bus

4.1.4.3.3 Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions

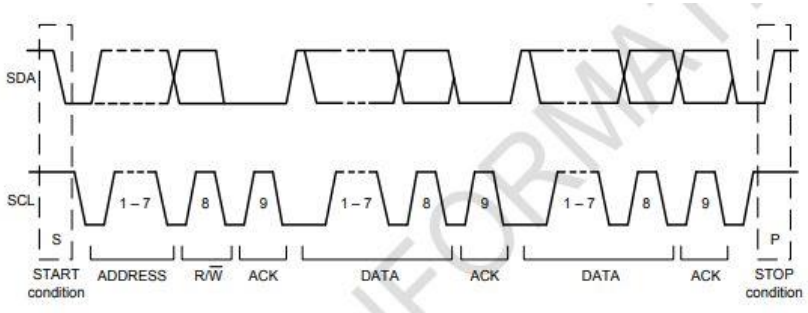


Figure 66: Complete I2C Data Transfer

To write the internal MPU-60X0 registers, the master transmits the start condition (S), followed by the I2C address and the write bit (0). At the 9th clock cycle (when the clock is high), the MPU-60X0 acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the MPU-60X0 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the MPU-60X0 automatically increments the register address and loads the data to the appropriate register. The following figures show single and two-byte write sequences.

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Figure 67: Single Byte Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Figure 68: Burst Write Sequence

To read the internal MPU-60X0 registers, the master sends a start condition, followed by the I2C address and a write bit, and then the register address that is going to be read. Upon receiving the ACK signal from the MPU-60X0, the master transmits a start signal followed by the slave address and read bit. As a result, the MPU-60X0 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9 th clock cycle. The following figures show single and two-byte read sequences.

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Figure 69: Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Figure 70: Burst Read Sequence

4.1.4.4 I 2C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I^2C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9^{th} clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9^{th} clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

Table 7: I2C Terms

4.1.4.5 Micro Controller Connection

There are two I2C connections available on the STM32F103RB. These connections are available on J1 Connector. PB6 is used for clock signal and PB7 is used for data signal. Connection of the first four pins of GY521/MOU6050 is done with first four pins of J1 connector. To measure the value of tilt we just need raw acceleration values which are obtained through accelerometer values.

4.1.5 Fan Pin Configuration

In the project Fan is turned on or off depending on the value of temperature. Fan's configuration itself is very simple. A single GPIO pin PC9 (PIN 40) of the micro controller is used to control the fan. It is configured as following:

- Mode = Output
- Pull = Pull Down
- Speer = Low

Pin has been configured as pull down so that when there is no signal provided to the pin, it is low which means fan is off.

4.1.6 Pump Pin Configuration

Pump control is like fan control. In this as well we must control a single GPIO pin PB12 (PIN 33) of the micro controller to control the pump. It is configured as following:

- Mode = Output
- Pull = Pull Down
- Speer = Low

4.1.7 Buzzer Pin Configuration

To control the buzzer PC8 (PIN 39) of the micro controller has been configures as following:

- Mode = Output
- Pull = Pull Down
- Speed = Low

Pin has been configured as pull down so that when there is no signal provided to the pin, it is low which means fan is off.

4.1.8 LCD Configuration

4.1.8.1 LCD 1602

LCD 1602 [10] is used to display all the sensors and indication devices statuses. It is a 16 pin LCD module with following pin interface:

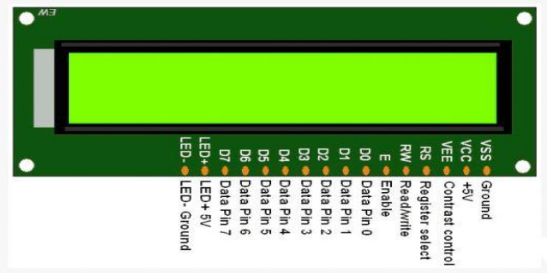


Figure 71: LCD 1602

4.1.8.2 Pins Description

PIN NO.	SYMBOL	DESCRIPTION	FUNCTION
1	VSS	GROUND	0V (GND)
2	VCC	POWER SUPPLY FOR LOGIC CIRCUIT	+5V
3	VEE	LCD CONTRAST ADJUSTMENT	
4	RS	INSTRUCTION/DATA REGISTER SELECTION	RS = 0 : INSTRUCTION REGISTER RS = 1 : DATA REGISTER
5	R/W	READ/WRITE SELECTION	R/W = 0 : REGISTER WRITE R/W = 1 : REGISTER READ
6	E	ENABLE SIGNAL	
7	DB0	DATA INPUT/OUTPUT LINES	8 BIT: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	LED+	SUPPLY VOLTAGE FOR LED+	+5V
16	LED-	SUPPLY VOLTAGE FOR LED-	0V

Table 8: LCD Pins Description

4.1.8.3 Timing Diagram

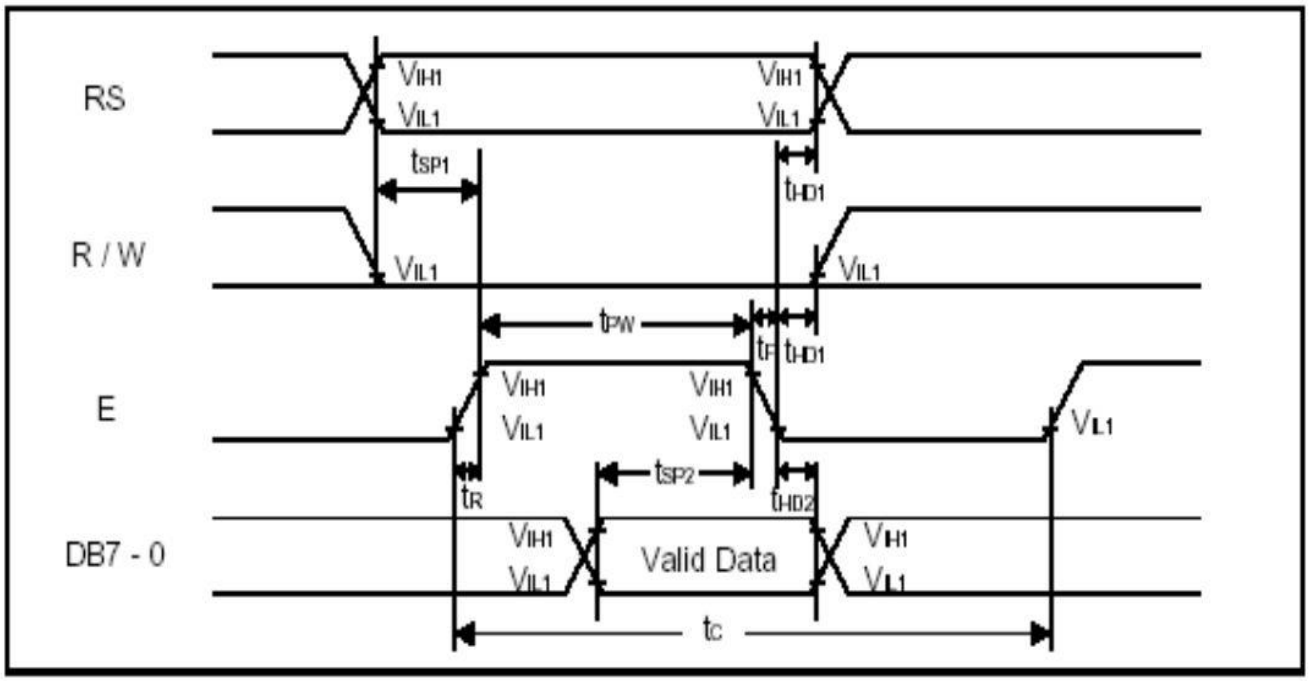


Figure 72: Write Sequence

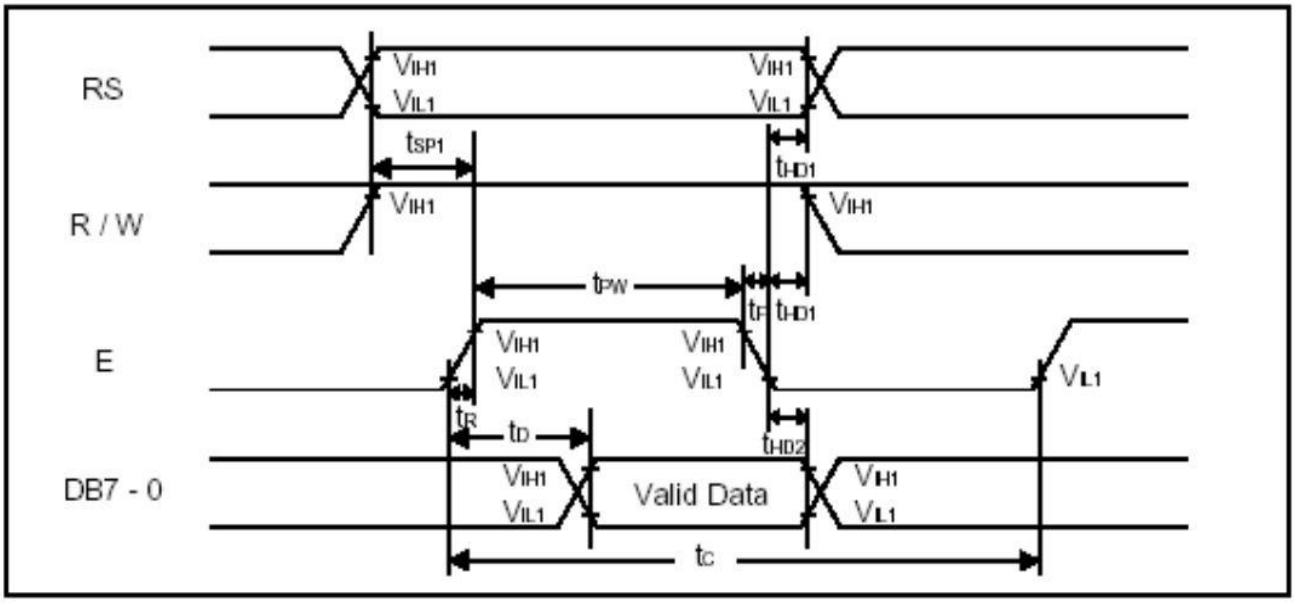


Figure 73: Read Sequence

As only write operation is done on the LCD R/W pin is always set to ground.

4.1.8.4 Initialization Sequence

Last four data bits are used instead of all 8 bits. Therefore initialization sequence of 4 bit interface is mentioned below:

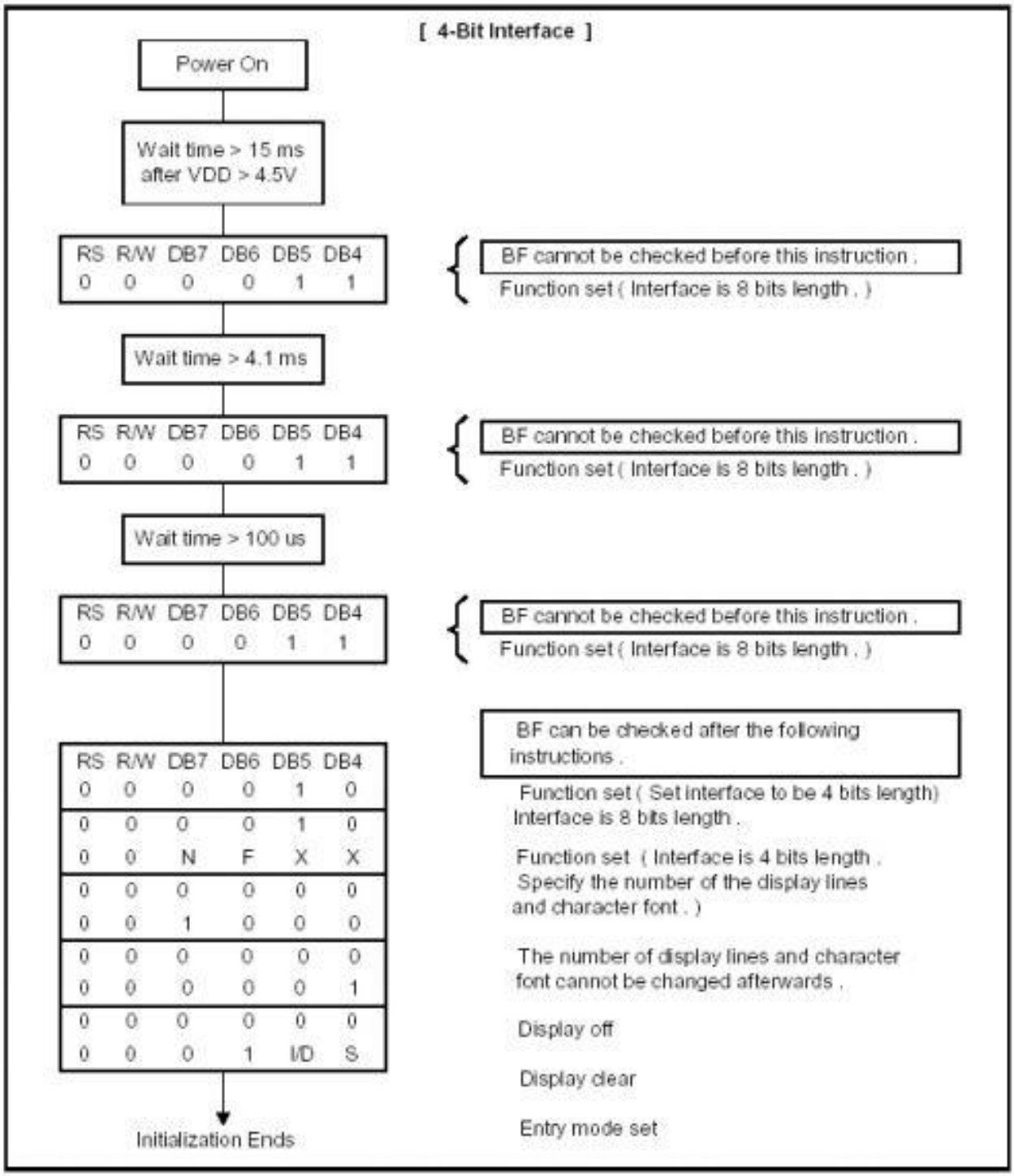


Figure 74: Initialization Sequence

4.1.8.5 Micro Controller Connection

Other than power supplies, 6 pin of STM32 are connected with the LCD. These pins include Data pins, Enable and Register Select Pin. All these pins are configured as following:

- Mode = Output
- Pull = No Pull
- Speed = Low

LCD Pin	STM Pin
DB4	PA3 (PIN 17)
DB5	PA2 (PIN 16)
DB6	PA1 (PIN 15)
DB7	PA0 (PIN 14)
E	PA4 (PIN 20)
RS	PA5 (PIN 21)

Table 9: LCD Pins Connection

4.1.9 Keypad Configuration

Keypad is connected to STM32 though the schematic mentioned in the last chapter. It is a 4 button Keypad so four GPIO pin are configured as following:

- Mode = Input
- Pull = No Pull

Button	STM Pin
KEYPAD1	PC5 (PIN 25)
KEYPAD1	PC4 (PIN 24)
KEYPAD1	PA7 (PIN 23)
KEYPAD1	PA6 (PIN 22)

Table 10: KeyPad Pins Connection

4.1.10 LED Configuration

Final components which need to be configured are the LEDs which are directly controlled by the micro controller based on the algorithm. The pins connected to the Darlington which is connected to LEDS are configured as following:

- Mode = Output
- Pull = No Pull
- Speed = Low

LED	STM Pin
KEYPAD1	PC5 (PIN 25)
KEYPAD1	PC4 (PIN 24)
KEYPAD1	PA7 (PIN 23)
KEYPAD1	PA6 (PIN 22)

Table 11: KeyPad Pins Connection

4.2 Source Codes

Once all the pins are configured, now is the time to implement the algorithm. Source Codes can be seen in Appendix A section.

CHAPTER 5

5 TESTING

One of the most important part of any project is Testing. Before explaining how testing has been used and implemented in our project, it is worth mentioning two common types of testing approaches: Black box and White Box Testing. In the following it is explained how these methods of testing work and how they are applied in this case.

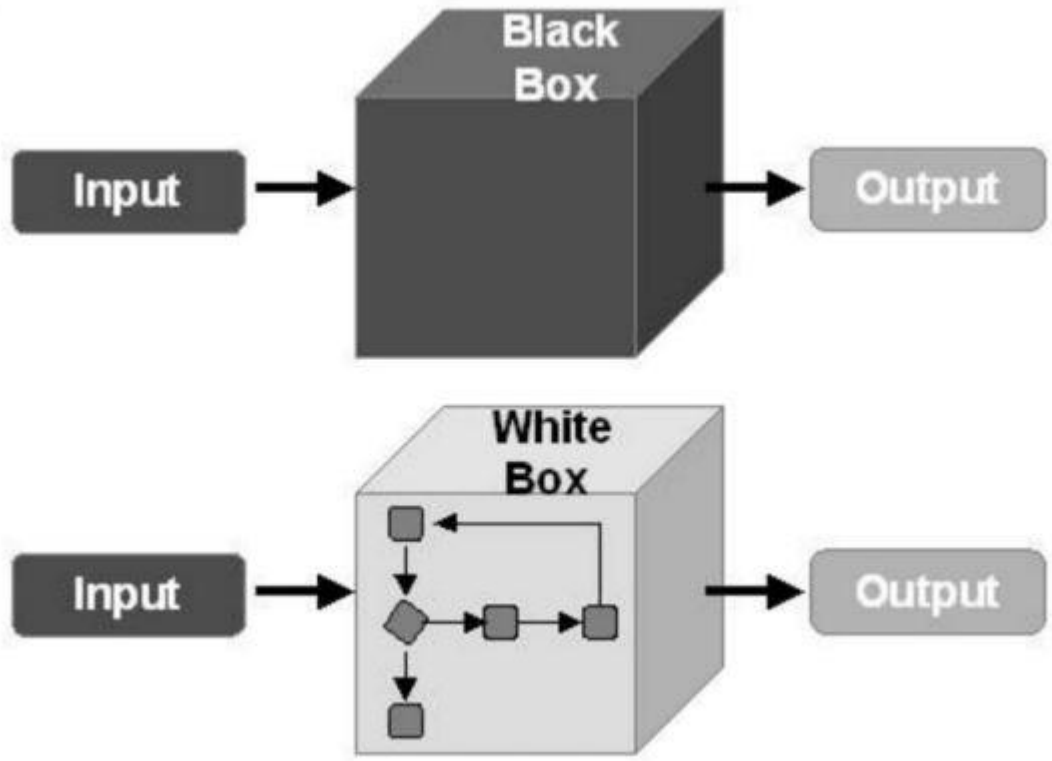


Figure 75: Testing approaches

5.1 Black Box Testing

Black-box testing (also known as functional testing) treats software under test as a black-box without knowing its internals. Tests use software interfaces and try to ensure that they work as expected. As long as functionality of interfaces remains unchanged, tests should pass even if internals are changed. Tester is aware of what the program should do but does not have the knowledge of how it does it. Black-box testing is most used type of testing in traditional organizations that have testers as a separate department, especially when they are not

proficient in coding and have difficulties to understand the code. It provides external perspective of the software under test. Some of the advantages of black-box testing are:

- Efficient for large segments of code
- Code access is not required
- Separation between user's and developer's perspectives

Some of the disadvantages of black-box testing are:

- Limited coverage since only a fraction of test scenarios is performed
- Inefficient testing due to tester's luck of knowledge about software internals
- Blind coverage since tester has limited knowledge about the application

The way we use black box testing approach is that during the implementation, modular approach is used. When a function or a peripheral is implemented, it is tested using the LCD Display and results have been noted Once all the separate peripherals are tested then combination of various peripherals and logics is tested. Different test cases have been discussed later.

5.2 White Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) looks inside the software that is being tested and uses that knowledge as part of the testing process. If, for example, exception is thrown under certain conditions, test might want to reproduce those conditions. White-box testing requires internal knowledge of the system and programming skills. It provides internal perspective of the software under test. Some of the advantages of white-box testing are:

- Efficient in finding errors (hidden) and problems
- Helps optimizing the code
- Due to required internal knowledge of the software, maximum coverage is obtained

Some of the disadvantages of white-box testing are:

- Might not find unimplemented or missing features
- Requires high level knowledge of internals of the software under test
- Requires code access

It is evident that, accessing to the code, having enough know-how about both the code and the way it works, provide us the opportunity to use white box testing. To this end, we put break point in any part we had a problem during black box testing procedure or those parts that are complicated to trace. In this way, we verify that our code works in any scenario. As mentioned in previous part, LCD helps us to recognize problems faster, especially in tracking easily our vital variables.

5.3 Test Cases and Results

5.3.1 Sensors

5.3.1.1 NTC10K Check

The value of ADC pin which is connected to NTC10K amplifier output is displayed on LCD. The temperature value should vary depending on the temperature of the environment. So first of all, temperature is measured at room temperature and then temperature is varied by touching the sensor as body heat increases the temperature and result is checked on the LCD. Below are the displays of LCD at various temperature conditions:

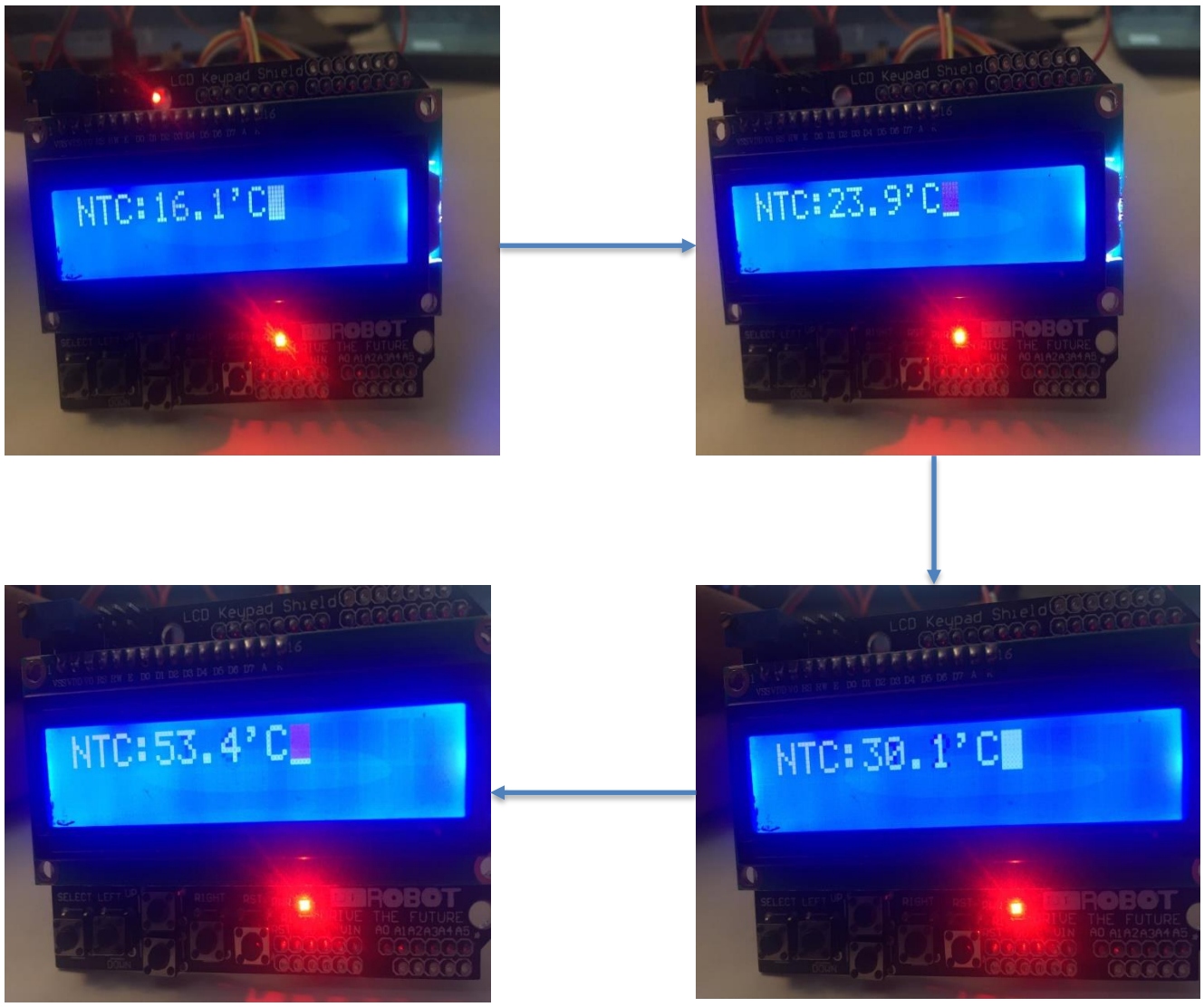


Figure 76: NTC10K Test Results

5.3.1.2 Level and Tilt Check

In order to check whether level sensors and accelerometer are working fine first LCD output is checked for low liquid level. Then level has been increased to nominal range and result has been verified. Similarly, level above upper sensor has been checked. Finally, container is tilted and result is check for level. Below are the displays of LCD at various level and tilt conditions:

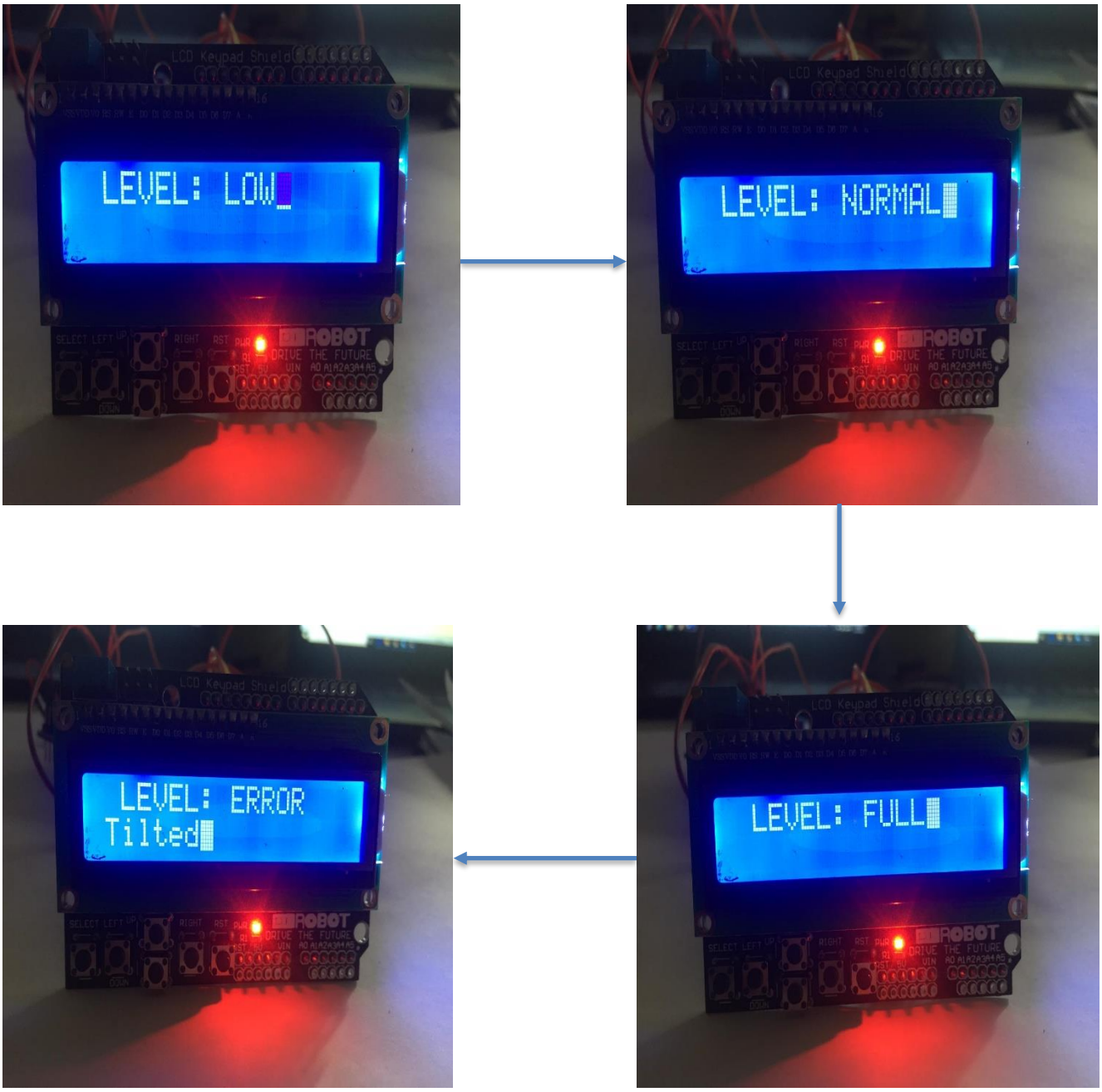


Figure 77: Level and Tilt Check

5.3.1.3 DHT22 Check

DHT22 output is checked on the LCD as before. In order to change humidity different conditions have been applied like checking output in kitchen with boiling water and in normal room condition. For temperature room heater has been used to control the temperature and for higher temperature kitchen environment has been considered as well. Below are the displays of LCD at various temperature and humidity conditions:

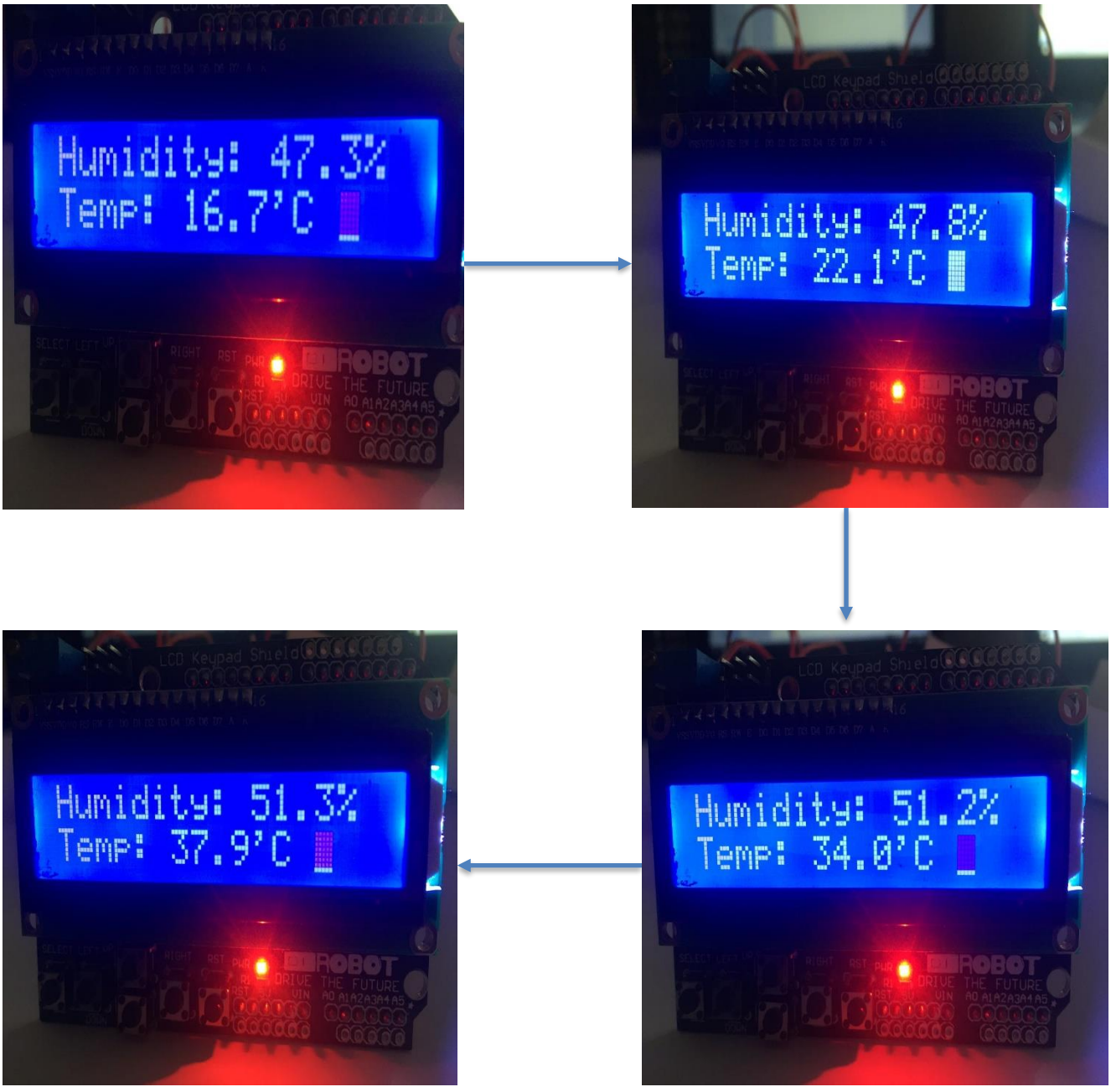


Figure 78: DHT22 Check

5.3.2 Control Devices Check

5.3.2.1 Fan Control

Fan control is checked in such a way that FAN turns on when temperature exceeds 35'C. When it is below this temperature fan should be off. Status of FAN appears alongside temperature on LCD. LCD result has been displayed for various temperature values. One thing to note is that only temperature and fan control is implemented, other part of the code is commented.

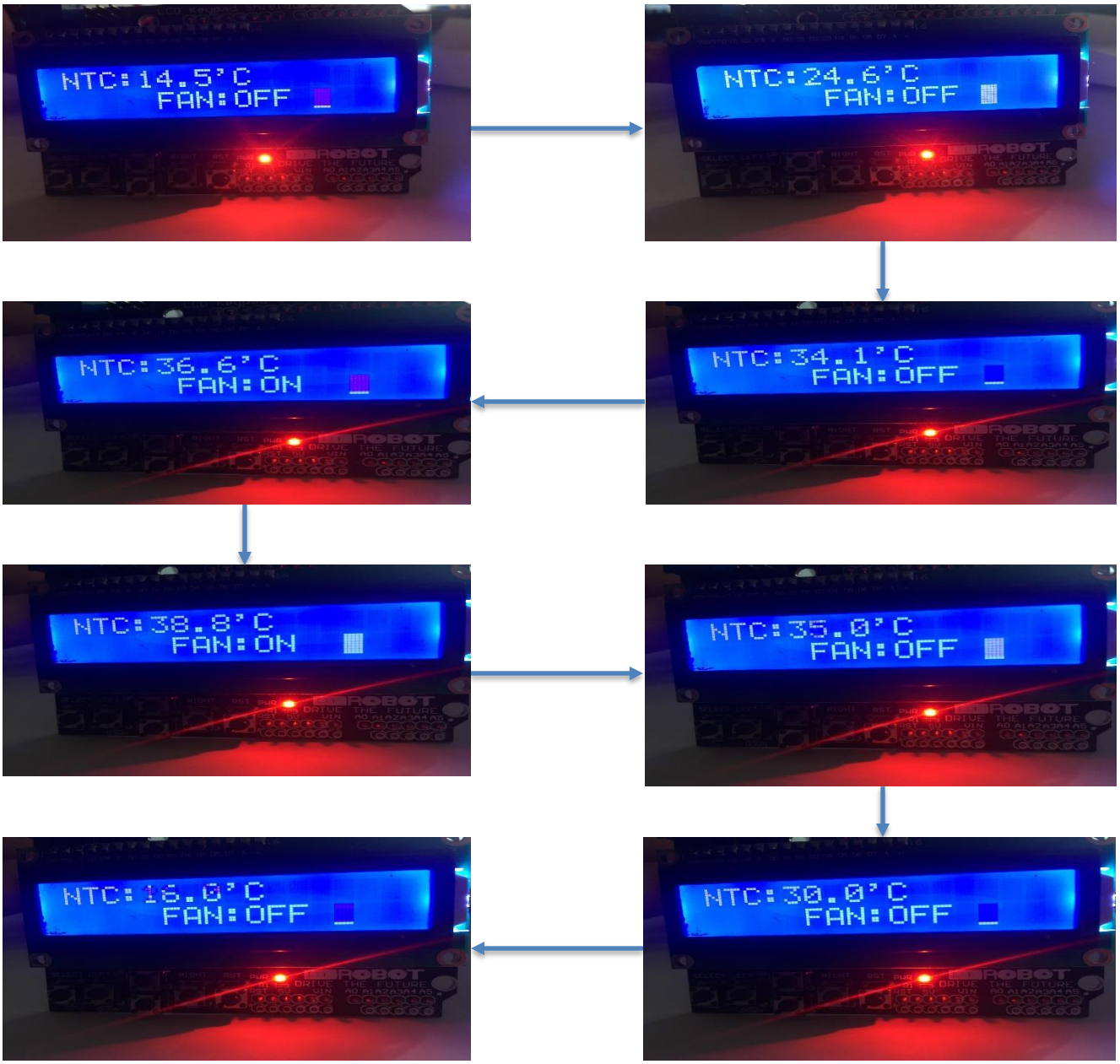


Figure 79: Fan Control Check

5.3.2.2 Pump Control

In this test case pump control is checked. If level is low pump should turn on and status should appear on LCD. If level is nominal or full pump should be off and status should display off. If container is tilted, then error message should appear and pump should be off. Following pictures show the result of LCD:

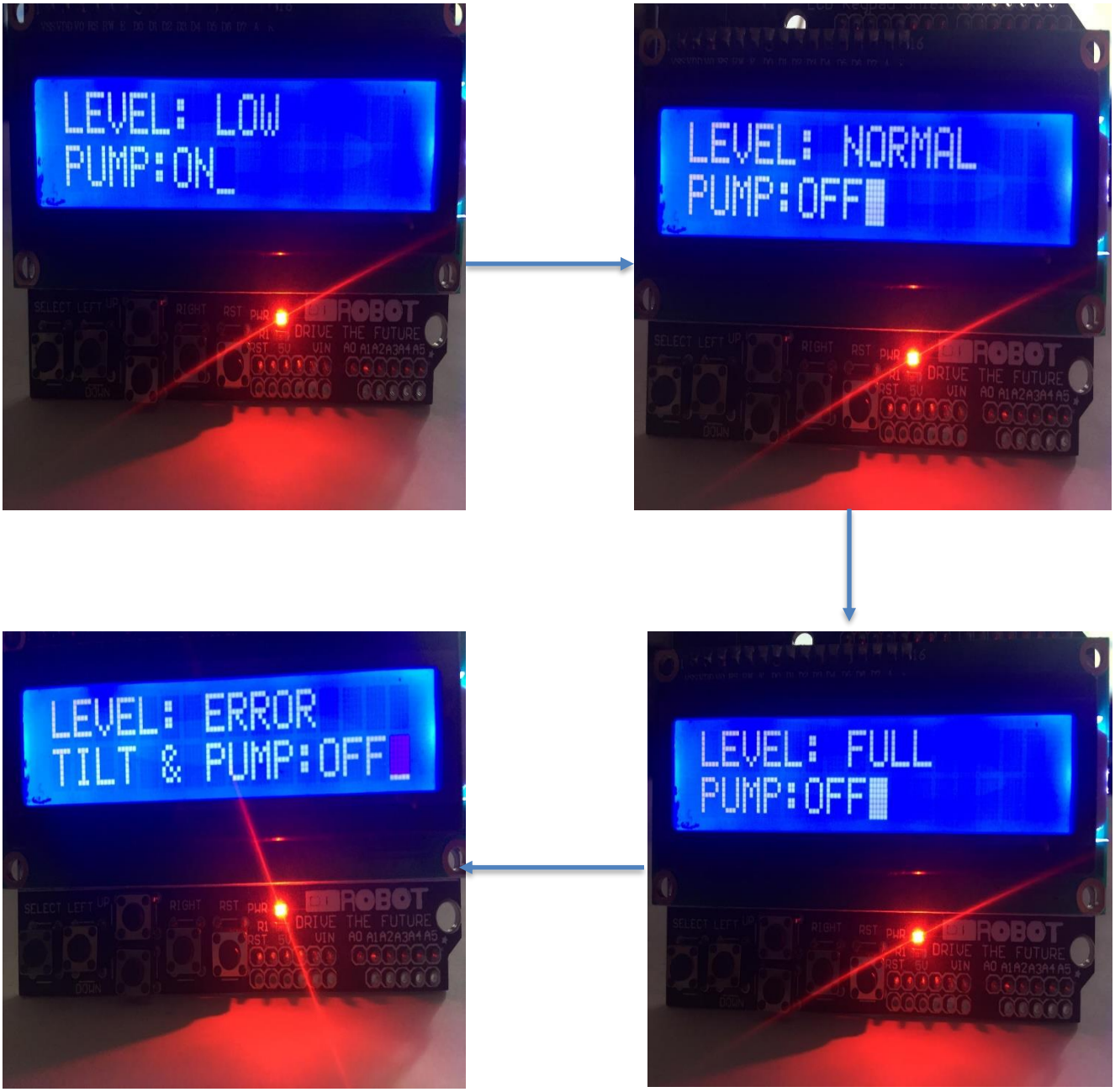


Figure 80: Pump Control Check

5.3.2.3 Buzzer Control

In this test case temperature is controlled using kitchen stove and heater while humidity is increased by boiling water. If either of temperature or humidity values are above nominal values buzzer turns on. Following pictures show the results on LCD:

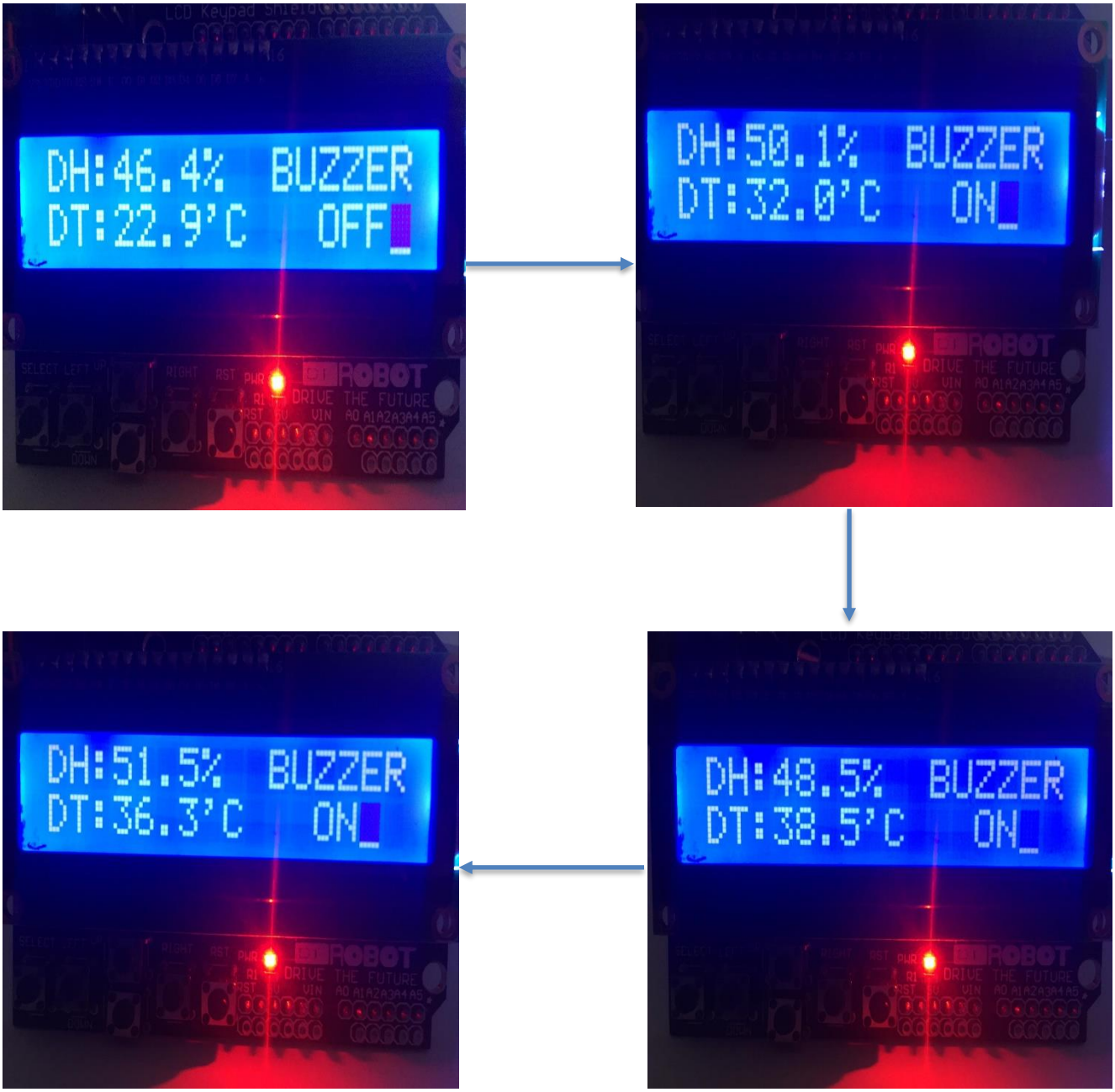


Figure 81: Buzzer Control Check

5.3.3 Complete Test

In final test whole code is run and display on LCD is checked. All sensor values and control device statuses should appear. Whenever a key is pressed from the KEYPAD specific results should appear All possible test cases have been selected

5.3.3.1 Test Case No 1 (FAN 0FF; PUMP 0FF; BUZZER OFF)

No Button Pressed

KeyPad 1 Pressed

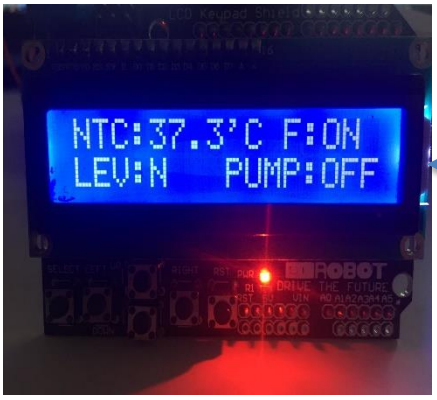
KeyPad 2 Pressed

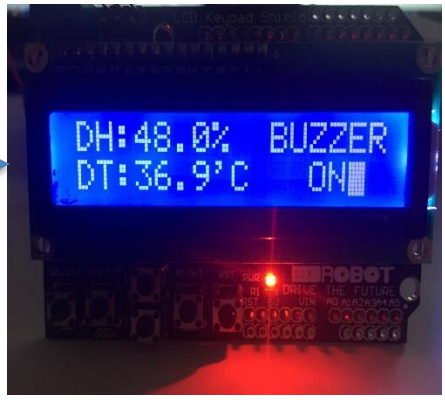
KeyPad 3 Pressed

Table 12: Test Case 1 Result

5.3.3.2 Test Case No 2 (FAN 0N; PUMP 0FF; BUZZER ON due to temperature only)

No Button Pressed





KeyPad 1 Pressed

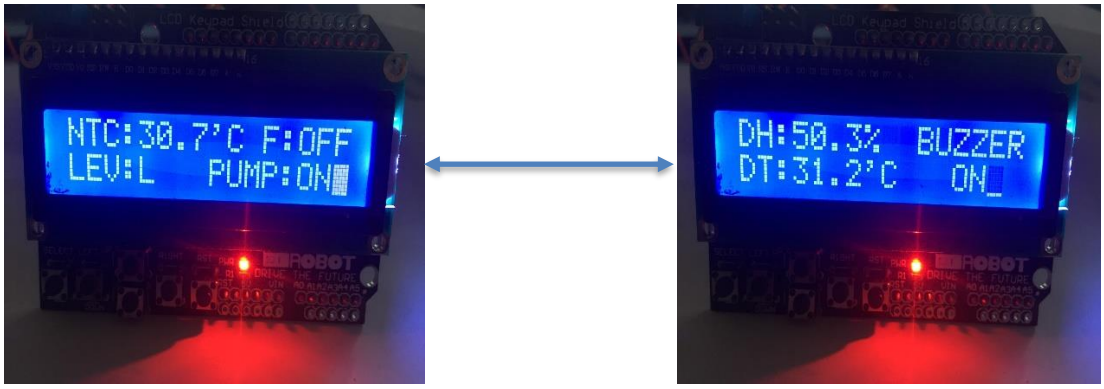
KeyPad 2 Pressed

KeyPad 3 Pressed

Table 13: Test Case 2 Result

5.3.3.3 Test Case No 3 (FAN 0FF; PUMP 0N; BUZZER ON due to Humidity only)

No Button Pressed



KeyPad 1 Pressed

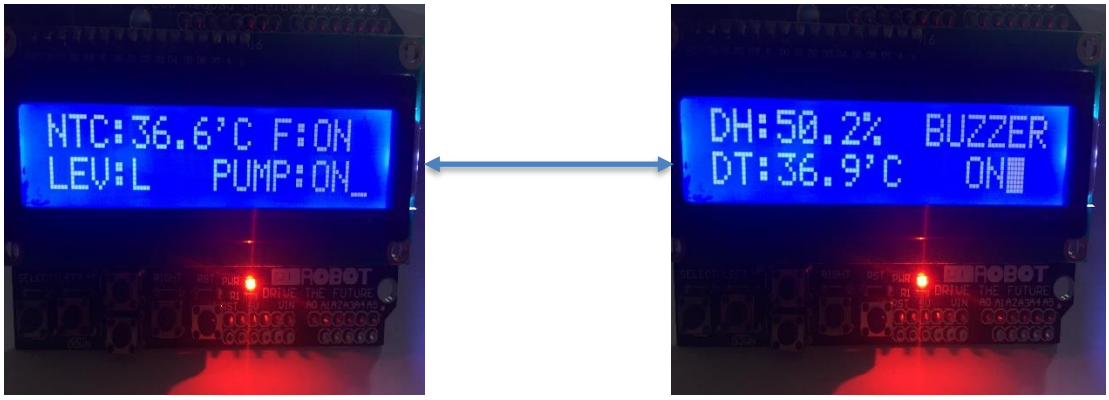
KeyPad 2 Pressed

KeyPad 3 Pressed

Table 14: Test Case 3 Result

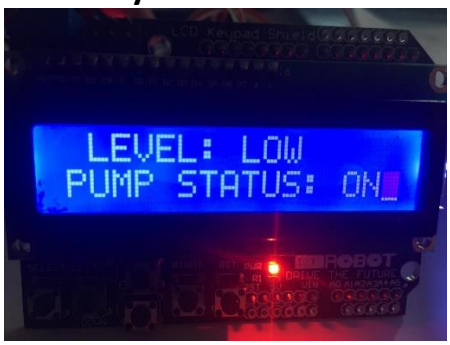
5.3.3.4 Test Case No 4 (FAN 0N; PUMP 0N; BUZZER ON due to Temperature & Humidity both)

No Button Pressed



KeyPad 1 Pressed

KeyPad 2 Pressed



KeyPad 3 Pressed

Table 15: Test Case 4 Result

CHAPTER 6

6 CONCLUSION

The focus of this thesis work was to develop an electronic board that takes data from various sensors and after processing this data control certain devices in order to maintain environmental conditions for the workers and to maintain level of water in a tank.

In order to achieve these objectives standard electronic development project steps have been followed. These steps include 1) Requirement Specification 2) Design 3) Coding and 4) Testing

In Requirement Specification phase all the objectives were defined. These objectives include getting temperature measurements both analogue and digital, humidity measurement, tilt measurement and level measurements. Then based on temperature measurements fan should be controlled, humidity measurement buzzer should be controlled and based on tilt and level measurements pump should be controlled. All the results should be displayed on LCD and LEDs should indicate the statuses of sensors and control devices.

In Design phase first, all the schematics were developed to achieve the objectives defined in the first phase: which ICs, operational amplifiers, values of resistors and capacitors to be used were calculated and defined in this phase. Additionally, based on electrical characteristics, sensors, control devices and microcontroller were selected. After the completion of schematics, all footprints of the devices were developed manually or using KiCad libraries and PCB Layout was developed. After completing the design of PCB, Gerber files were generated and sent to PCB manufacturer. Once PCB was returned by manufacturer, it was assembled with the components of the schematic.

In Coding phase, all the devices were configured using datasheets of the devices and micro Controller STM32F1RB. These configurations include initialization of DHT22 sensor, NTC10k ADC, floating sensors GPIOs, MPU6050, I2C, fan, pump buzzer and LED GPIOs and finally LCD1602 initialization sequence. After these configurations algorithm was developed to get reading from sensors, control fan, pump, buzzer and LEDs and display results and statuses on LCD.

In Testing phase, code was programmed on the electronic board and tested using black box and white box techniques. Various test cases were defined, and results were verified according to desired objectives mentioned in first phase of the project. This concludes the thesis work.

Appendix A

Source Codes

Main C File

```
/* Includes */
#include "main.h"
#include "STM_MY_LCD16X2.h"
#include <stdbool.h>
#include <math.h>
#include "MY_DHT22.h"
#include "TJ_MPU6050.h"

/* Peripheral Instances */
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim3;

/* Functions Declaration */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_I2C1_Init(void);

/* Global Variables */
uint32_t adcVal[2];
uint32_t Level[2];
uint8_t DHT22_data[5];
uint32_t data=0;
uint8_t checksum=0;
int button=0;
float Vout;
float invB=(float)1/3950;
float NTC_Voltage, NTC_Res;
float Atemp_reciprocal;
float Atemp;
float Dtemp, Humidity;
RawData_Def myAccelRaw, myGyroRaw;
ScaledData_Def myAccelScaled, myGyroScaled;
/* Main Function */
int main(void)
{
  int level;
  MPU_ConfigTypeDef myMpuConfig;

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

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_I2C1_Init();

/* Start Timer */
HAL_TIM_Base_Start(&htim3);
/* Start ADC as DMA */
HAL_ADC_Start_DMA(&hadc1, adcVal, 2);
/* LCD Initialization*/
LCD1602_Begin4BIT(RS_GPIO_Port, RS_Pin, E_Pin, DB4_GPIO_Port, DB4_Pin, DB5_Pin, DB6_Pin, DB7_Pin);
/* DHT22 Initialization */
DHT22_Init(DHT22_SDA_GPIO_Port,DHT22_SDA_Pin);
/* Initialize the MPU6050 module and I2C */
MPU6050_Init(&hi2c1);
/* Configure Accel and Gyro parameters */
myMpuConfig.Accel_Full_Scale = AFS_SEL_4g;
myMpuConfig.ClockSource = Internal_8MHz;
myMpuConfig.CONFIG_DLPF = DLPF_184A_188G_Hz;
myMpuConfig.Gyro_Full_Scale = FS_SEL_500;
myMpuConfig.Sleep_Mode_Bit = 0;
MPU6050_Config(&myMpuConfig);

while (1)
{
	/* Scaled Acceleration and Gyroscope Data */
	MPU6050_Get_Accel_Scale(&myAccelScaled);
	MPU6050_Get_Gyro_Scale(&myGyroScaled);

	/* NTC10K data and Display on LCD */
	Vout= (adcVal[0]*3.3)/4032;
	NTC_Voltage=Vout/1.1;
	NTC_Res=(NTC_Voltage*8000/3.3)/(1-(NTC_Voltage/3.3));
	Atemp_reciprocal=0.003354+log((double)NTC_Res/10000)*invB;
	Atemp=1/Atemp_reciprocal-273.15;
	LCD1602_clear();
	LCD1602_print("NTC:");
	LCD1602_PrintFloat(Atemp,1);
	LCD1602_print("'C");
	/* Level Data and Display on LCD */
	LCD1602_print(" LEV:");
	Level[0]=HAL_GPIO_ReadPin(LevelUp_GPIO_Port, LevelDown_Pin);
	Level[1]=HAL_GPIO_ReadPin(LevelDown_GPIO_Port, LevelUp_Pin);
	if(Level[0]==0 && Level[1]==0)
	{
		LCD1602_print("L");
		level=0;
		HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 0);
		HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 0);
	}
```

```
else if (Level[0]==1 && Level[1]==1)
{
	LCD1602_print("F");
	level=1;
	HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 1);
	HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 1);
}
else if (Level[0]==1 && Level[1]==0)
{
	LCD1602_print("N");
	level=1;
	HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 1);
	HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 0);
}
else
{
	LCD1602_print("E");
	level=2;
	HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 1);
	HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 0);
}

/* Pump,Fan and their respective LEDs Control and Display on LCD */
LCD1602_2ndLine();
if (Atemp>35)
{
	HAL_GPIO_WritePin(FAN_GPIO_Port, FAN_Pin,1);
	LCD1602_print("FAN:ON  ");
	HAL_GPIO_WritePin(NTC_LED_GPIO_Port, NTC_LED_Pin, 1);
}
else
{
	HAL_GPIO_WritePin(FAN_GPIO_Port, FAN_Pin,0);
	LCD1602_print("FAN:OFF ");
	HAL_GPIO_WritePin(NTC_LED_GPIO_Port, NTC_LED_Pin, 0);
}

if (level==0)
{
	HAL_GPIO_WritePin(PUMP_GPIO_Port, PUMP_Pin, 1);
	LCD1602_print("PUMP:ON");
}
else if (level==1)
{
	HAL_GPIO_WritePin(PUMP_GPIO_Port, PUMP_Pin, 0);
	LCD1602_print("PUMP:OFF");
}
else
{
	LCD1602_print("TILTED");
}
HAL_Delay(1000);
```

```
/* DHT22 Data and Display on LCD */
LCD1602_clear();
LCD1602_1stLine();
LCD1602_print("DH:");
LCD1602_PrintFloat(Humidity,1);
LCD1602_print("% ");
LCD1602_print("DT:");
LCD1602_PrintFloat(Dtemp,1);
LCD1602_print("'C ");

/* Buzzer Control and Display on LCD */
LCD1602_2ndLine();
if (Humidity>=30 && Humidity<=50 && Dtemp>35)
{
        HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 1);
        LCD1602_print("   Buzzer:ON ");
}
else
{
        HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 0);
        LCD1602_print("   Buzzer:OFF ");
}
HAL_Delay(1000);

/* KeyPad Control */
while ( HAL_GPIO_ReadPin(Button_GPIO_Port,Button_Pin)==0 ||
        HAL_GPIO_ReadPin(KEY1_GPIO_Port,KEY1_Pin)==0 ||
        HAL_GPIO_ReadPin(KEY2_GPIO_Port,KEY2_Pin)==0 ||
        HAL_GPIO_ReadPin(KEY3_GPIO_Port,KEY3_Pin)==0 ||
        HAL_GPIO_ReadPin(KEY4_GPIO_Port,KEY4_Pin)==0 )
{
        if ( HAL_GPIO_ReadPin(KEY1_GPIO_Port,KEY1_Pin)==0 )
        {
                LCD1602_clear();
                LCD1602_1stLine();
                Vout= (adcVal[0]*3.3)/4032;
                NTC_Voltage=Vout/1.1;
                NTC_Res=(NTC_Voltage*8000/3.3)/(1-(NTC_Voltage/3.3));
                Atemp_reciprocal=0.003354+log((double)NTC_Res/10000)*invB;
                Atemp=1/Atemp_reciprocal-273.15;
                LCD1602_print("NTC:");
                LCD1602_PrintFloat(Atemp,1);
                LCD1602_print("'C");
                LCD1602_2ndLine();
                if (Atemp>35)
                {
                        HAL_GPIO_WritePin(FAN_GPIO_Port, FAN_Pin,1);
                        LCD1602_print("FAN:ON  ");
                        HAL_GPIO_WritePin(NTC_LED_GPIO_Port, NTC_LED_Pin, 1);
                }
                else
                {
                        HAL_GPIO_WritePin(FAN_GPIO_Port, FAN_Pin,0);
                        LCD1602_print("FAN:OFF ");
```

```
                HAL_GPIO_WritePin(NTC_LED_GPIO_Port, NTC_LED_Pin, 0);
        }
        HAL_Delay(1000);
}
 else if ( HAL_GPIO_ReadPin(Button_GPIO_Port,Button_Pin)==0 ||
        HAL_GPIO_ReadPin(KEY2_GPIO_Port,KEY2_Pin)==0 )
{
        LCD1602_clear();
        LCD1602_1stLine();
        LCD1602_print(" LEVEL: ");
        Level[0]=HAL_GPIO_ReadPin(LevelUp_GPIO_Port, LevelDown_Pin);
        Level[1]=HAL_GPIO_ReadPin(LevelDown_GPIO_Port, LevelUp_Pin);
        if(Level[0]==0 && Level[1]==0)
        {
                LCD1602_print("LOW");
                level=0;
                HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 0);
                HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 0);
        }
        else if (Level[0]==1 && Level[1]==1)
        {
                LCD1602_print("FULL");
                level=1;
                HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 1);
                HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 1);
        }
        else if (Level[0]==1 && Level[1]==0)
        {
                LCD1602_print("NOMINAL");
                level=1;
                HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 1);
                HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 0);
        }
        else
        {
                LCD1602_print("ERROR");
                level=2;
                HAL_GPIO_WritePin(Up_LED_GPIO_Port, Up_LED_Pin, 1);
                HAL_GPIO_WritePin(Down_LED_GPIO_Port, Down_LED_Pin, 0);
        }
        LCD1602_2ndLine();
        if (level==0)
        {
                HAL_GPIO_WritePin(PUMP_GPIO_Port, PUMP_Pin, 1);
                LCD1602_print("PUMP STATUS: ON");
        }
        else if (level==1)
        {
                HAL_GPIO_WritePin(PUMP_GPIO_Port, PUMP_Pin, 0);
                LCD1602_print("PUMP STATUS: OFF");
        }
        else
        {
```

```
                        LCD1602_print("TILTED");
                    }
                    HAL_Delay(1000);

                }
                else if ( HAL_GPIO_ReadPin(KEY3_GPIO_Port,KEY3_Pin)==0 )
                {
                    LCD1602_clear();
                    LCD1602_1stLine();
                    LCD1602_print("DH:");
                    LCD1602_PrintFloat(Humidity,1);
                    LCD1602_print("% ");
                    LCD1602_print("DT:");
                    LCD1602_PrintFloat(Dtemp,1);
                    LCD1602_print("'C ");

                    LCD1602_2ndLine();
                    if (Humidity>=30 && Humidity<=50 && Dtemp>35)
                    {
                        HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 1);
                        LCD1602_print("   Buzzer:ON ");
                    }
                    else
                    {
                        HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, 0);
                        LCD1602_print("   Buzzer:OFF ");
                    }
                    HAL_Delay(1000);
                }
                else if ( HAL_GPIO_ReadPin(KEY4_GPIO_Port,KEY4_Pin)==0 )
                {
                    LCD1602_clear();
                    LCD1602_print("MPU6050/GY521");
                    LCD1602_2ndLine();
                    LCD1602_print("   NOT CONNECTED");
                    HAL_Delay(1000);
                }
            }
  }
}
/* System Clock Configuration */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /* Initializes the CPU, AHB and APB busses clocks  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
```

```
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /* Initializes the CPU, AHB and APB busses 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_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/* Initialization Function */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};
  ADC_InjectionConfTypeDef sConfigInjected = {0};

  /* Common config */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
  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_13;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /* Configure Injected Channel */
  sConfigInjected.InjectedChannel = ADC_CHANNEL_13;
  sConfigInjected.InjectedRank = ADC_INJECTED_RANK_1;
  sConfigInjected.InjectedNbrOfConversion = 2;
  sConfigInjected.InjectedSamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  sConfigInjected.ExternalTrigInjecConv = ADC_INJECTED_SOFTWARE_START;
  sConfigInjected.AutoInjectedConv = DISABLE;
  sConfigInjected.InjectedDiscontinuousConvMode = DISABLE;
  sConfigInjected.InjectedOffset = 0;
  if (HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
  {
    Error_Handler();
  }

  /* Configure Injected Channel */
  sConfigInjected.InjectedChannel = ADC_CHANNEL_8;
  sConfigInjected.InjectedRank = ADC_INJECTED_RANK_2;
  if (HAL_ADCEx_InjectedConfigChannel(&hadc1, &sConfigInjected) != HAL_OK)
  {
    Error_Handler();
  }
}

/* I2C1 Initialization Function */
static void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
}
/* TIM3 Initialization Function */
static void MX_TIM3_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 800;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1000;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

/* Enable DMA controller clock */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init & DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}

/* GPIO Initialization Function */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, DHT22_SDA_Pin|DH_LED_Pin|NTC_LED_Pin|Buzzer_Pin
                          |FAN_Pin, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, DB6_Pin|DB7_Pin|PUMP_Pin|Down_LED_Pin
                    |Up_LED_Pin|DT_LED_Pin|DB5_Pin|DB4_Pin
                    |E_Pin|RS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : Button_Pin KEY2_Pin KEY1_Pin */
  GPIO_InitStruct.Pin = Button_Pin|KEY2_Pin|KEY1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : LevelUp_Pin LevelDown_Pin */
  GPIO_InitStruct.Pin = LevelUp_Pin|LevelDown_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : DHT22_SDA_Pin DH_LED_Pin NTC_LED_Pin Buzzer_Pin FAN_Pin */
  GPIO_InitStruct.Pin = DHT22_SDA_Pin|DH_LED_Pin|NTC_LED_Pin|Buzzer_Pin
                        |FAN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : KEY4_Pin KEY3_Pin */
  GPIO_InitStruct.Pin = KEY4_Pin|KEY3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : DB6_Pin DB7_Pin PUMP_Pin Down_LED_Pin  Up_LED_Pin DT_LED_Pin DB5_Pin DB4_Pin E_Pin
     RS_Pin */
  GPIO_InitStruct.Pin = DB6_Pin|DB7_Pin|PUMP_Pin|Down_LED_Pin
                        |Up_LED_Pin|DT_LED_Pin|DB5_Pin|DB4_Pin
                        |E_Pin|RS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
/* This function is executed in case of error occurrence */
void Error_Handler(void)
{}

#ifdef  USE_FULL_ASSERT
/* Reports the name of the source file and the source line number where the assert_param error has occurred.*/
void assert_failed(uint8_t *file, uint32_t line)
{}
#endif
```

Main Header File

```
/* Define to prevent recursive inclusion */
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes */
#include "stm32f1xx_hal.h"

/* Exported functions prototypes */
void Error_Handler(void);
```

```
/* Private defines */
#define Button_Pin GPIO_PIN_13
#define Button_GPIO_Port GPIOC
#define LevelUp_Pin GPIO_PIN_11
#define LevelUp_GPIO_Port GPIOC
#define LevelDown_Pin GPIO_PIN_10
#define LevelDown_GPIO_Port GPIOC
#define DHT22_SDA_Pin GPIO_PIN_2
#define DHT22_SDA_GPIO_Port GPIOC
#define NTC10K_Pin GPIO_PIN_3
#define NTC10K_GPIO_Port GPIOC
#define KEY4_Pin GPIO_PIN_6
#define KEY4_GPIO_Port GPIOA
#define KEY3_Pin GPIO_PIN_7
#define KEY3_GPIO_Port GPIOA
#define KEY2_Pin GPIO_PIN_4
#define KEY2_GPIO_Port GPIOC
#define KEY1_Pin GPIO_PIN_5
#define KEY1_GPIO_Port GPIOC
#define DB6_Pin GPIO_PIN_10
#define DB6_GPIO_Port GPIOB
#define DB7_Pin GPIO_PIN_11
#define DB7_GPIO_Port GPIOB
#define PUMP_Pin GPIO_PIN_12
#define PUMP_GPIO_Port GPIOB
#define Down_LED_Pin GPIO_PIN_13
#define Down_LED_GPIO_Port GPIOB
#define Up_LED_Pin GPIO_PIN_14
#define Up_LED_GPIO_Port GPIOB
#define DT_LED_Pin GPIO_PIN_15
#define DT_LED_GPIO_Port GPIOB
#define DH_LED_Pin GPIO_PIN_6
#define DH_LED_GPIO_Port GPIOC
#define NTC_LED_Pin GPIO_PIN_7
#define NTC_LED_GPIO_Port GPIOC
#define Buzzer_Pin GPIO_PIN_8
#define Buzzer_GPIO_Port GPIOC
#define FAN_Pin GPIO_PIN_9
#define FAN_GPIO_Port GPIOC
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
#define DB5_Pin GPIO_PIN_4
#define DB5_GPIO_Port GPIOB
#define DB4_Pin GPIO_PIN_5
#define DB4_GPIO_Port GPIOB
#define E_Pin GPIO_PIN_8
#define E_GPIO_Port GPIOB
#define RS_Pin GPIO_PIN_9
#define RS_GPIO_Port GPIOB
```

```
#ifdef __cplusplus
}
#endif

#endif
```

DHT22 C File

```
/* Header files */
#include "MY_DHT22.h"

/* Bit fields manipulations */
#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))

/* 1. One wire data line */
static GPIO_TypeDef* oneWire_PORT;
static uint16_t oneWire_PIN;
static uint8_t oneWirePin_Idx;

/* Functions prototypes */
/* OneWire Initialise */
void DHT22_Init(GPIO_TypeDef* DataPort, uint16_t DataPin)
{
	oneWire_PORT = DataPort;
	oneWire_PIN = DataPin;
	for(uint8_t i=0; i<16; i++)
	{
		if(DataPin & (1 << i))
		{
			oneWirePin_Idx = i;
			break;
		}
	}
}
/* Change pin mode */
static void ONE_WIRE_PinMode(OnePinMode_Typedef mode)
{
	GPIO_InitTypeDef GPIO_InitStruct;
	GPIO_InitStruct.Pin = oneWire_PIN;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	GPIO_InitStruct.Pull = GPIO_NOPULL;

	if(mode == ONE_OUTPUT)
	{
		GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
	}
	else if(mode == ONE_INPUT)
	{
		GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
        }
        HAL_GPIO_Init(oneWire_PORT, &GPIO_InitStruct);
}
/* One Wire pin HIGH/LOW Write */
static void ONE_WIRE_Pin_Write(bool state)
{
        if(state) HAL_GPIO_WritePin(oneWire_PORT, oneWire_PIN, GPIO_PIN_SET);
        else HAL_GPIO_WritePin(oneWire_PORT, oneWire_PIN, GPIO_PIN_RESET);
}
static bool ONE_WIRE_Pin_Read(void)
{
        return (1&HAL_GPIO_ReadPin(oneWire_PORT, oneWire_PIN));
}

/* Microsecond delay */
static void DelayMicroSeconds(uint32_t uSec)
{
        uint32_t uSecVar = uSec;
        uSecVar = uSecVar* ((SystemCoreClock/1000000)/3);
        while(uSecVar--);
}

/* DHT Begin function */
static void DHT22_StartAcquisition(void)
{
        /* Change data pin mode to OUTPUT */
        ONE_WIRE_PinMode(ONE_OUTPUT);
        /* Put pin LOW */
        ONE_WIRE_Pin_Write(0);
        /* 500uSec delay */
        DelayMicroSeconds(500);
        /* Bring pin HIGH */
        ONE_WIRE_Pin_Write(1);
        /* 30 uSec delay */
        DelayMicroSeconds(30);
        /* Set pin as input */
        ONE_WIRE_PinMode(ONE_INPUT);
}
/* Read 5 bytes */
static void DHT22_ReadRaw(uint8_t *data)
{
        uint32_t rawBits = 0UL;
        uint8_t checksumBits=0;

        DelayMicroSeconds(40);
        while(!ONE_WIRE_Pin_Read());
        while(ONE_WIRE_Pin_Read());
        for(int8_t i=31; i>=0; i--)
        {
                while(!ONE_WIRE_Pin_Read());
                DelayMicroSeconds(40);
                if(ONE_WIRE_Pin_Read())
                {
```

```
                rawBits |= (1UL << i);
            }
            while(ONE_WIRE_Pin_Read());
        }

        for(int8_t i=7; i>=0; i--)
        {
            while(!ONE_WIRE_Pin_Read());
            DelayMicroSeconds(40);
            if(ONE_WIRE_Pin_Read())
            {
                checksumBits |= (1UL << i);
            }
            while(ONE_WIRE_Pin_Read());
        }

        /* Copy raw data to array of bytes */
        data[0] = (rawBits>>24)&0xFF;
        data[1] = (rawBits>>16)&0xFF;
        data[2] = (rawBits>>8)&0xFF;
        data[3] = (rawBits>>0)&0xFF;
        data[4] = (checksumBits)&0xFF;
}

/* Get Temperature and Humidity data */
bool DHT22_GetTemp_Humidity(float *Temp, float *Humidity)
{
        uint8_t dataArray[6], myChecksum;
        uint16_t Temp16, Humid16;
        /* Implement Start data Aqcuisition routine */
        DHT22_StartAcquisition();
        /* Aqcuire raw data */
        DHT22_ReadRaw(dataArray);
        /* calculate checksum */
        myChecksum = 0;
        for(uint8_t k=0; k<4; k++)
        {
            myChecksum += dataArray[k];
        }
        if(myChecksum == dataArray[4])
        {
            Temp16 = (dataArray[2] <<8) | dataArray[3];
            Humid16 = (dataArray[0] <<8) | dataArray[1];

            *Temp = Temp16/10.0f;
            *Humidity = Humid16/10.0f;
            return 1;
        }
        return 0;
}
```

DHT22 Header File

```
/* Header files */
#include "stm32f1xx_hal.h"
#include <stdbool.h>
#include <string.h>
#include <math.h>

/* Pin Mode enum */
typedef enum
{
	ONE_OUTPUT = 0,
	ONE_INPUT,
}OnePinMode_Typedef;

/ Functions prototypes */
/* One Wire Initialize */
void DHT22_Init(GPIO_TypeDef* DataPort, uint16_t DataPin);
/* Change pin mode */
static void ONE_WIRE_PinMode(OnePinMode_Typedef mode);
/* One Wire pin HIGH/LOW Write */
static void ONE_WIRE_Pin_Write(bool state);
static bool ONE_WIRE_Pin_Read(void);
/* Microsecond delay */
static void DelayMicroSeconds(uint32_t uSec);
/* Begin function */
static void DHT22_StartAcquisition(void);
/* Read 5 bytes */
static void DHT22_ReadRaw(uint8_t *data);
/* Get Temperature and Humidity data */
bool DHT22_GetTemp_Humidity(float *Temp, float *Humidity);
```

MPU6050 C File

```
/* Header files */
#include "TJ_MPU6050.h"

/* Library Variable */
/* 1- I2C Handle */
static I2C_HandleTypeDef i2cHandler;
/* 2- Accel & Gyro Scaling Factor */
static float accelScalingFactor, gyroScalingFactor;
/* 3- Bias varaibles */
static float A_X_Bias = 0.0f;
static float A_Y_Bias = 0.0f;
static float A_Z_Bias = 0.0f;

static int16_t GyroRW[3];

/* Fucntion Definitions */
/* 1- i2c Handler */
void MPU6050_Init(I2C_HandleTypeDef *I2Chnd)
{
	/* Copy I2C CubeMX handle to local library */
	memcpy(&i2cHandler, I2Chnd, sizeof(*I2Chnd));
```

```
}

/* 2- i2c Read */
void I2C_Read(uint8_t ADDR, uint8_t *i2cBif, uint8_t NofData)
{
	uint8_t i2cBuf[2];
	uint8_t MPUADDR;
	/* Need to Shift address to make it proper to i2c operation */
	MPUADDR = (MPU_ADDR<<1);
	i2cBuf[0] = ADDR;
	HAL_I2C_Master_Transmit(&i2cHandler, MPUADDR, i2cBuf, 1, 10);
	HAL_I2C_Master_Receive(&i2cHandler, MPUADDR, i2cBif, NofData, 100);
}

/* 3- i2c Write */
void I2C_Write8(uint8_t ADDR, uint8_t data)
{
	uint8_t i2cData[2];
	i2cData[0] = ADDR;
	i2cData[1] = data;
	uint8_t MPUADDR = (MPU_ADDR<<1);
	HAL_I2C_Master_Transmit(&i2cHandler, MPUADDR, i2cData, 2,100);
}

/* 4- MPU6050 Initialaztion Configuration */
void MPU6050_Config(MPU_ConfigTypeDef *config)
{
	uint8_t Buffer = 0;
	I2C_Write8(PWR_MAGT_1_REG, 0x80);
	HAL_Delay(100);
	Buffer = config ->ClockSource & 0x07; //change the 7th bits of register
	Buffer |= (config ->Sleep_Mode_Bit << 6) &0x40; // change only the 7th bit in the register
	I2C_Write8(PWR_MAGT_1_REG, Buffer);
	HAL_Delay(100); // should wait 10ms after changing the clock setting.

	/* Set the Digital Low Pass Filter */
	Buffer = 0;
	Buffer = config->CONFIG_DLPF & 0x07;
	I2C_Write8(CONFIG_REG, Buffer);
	/* Select the Gyroscope Full Scale Range */
	Buffer = 0;
	Buffer = (config->Gyro_Full_Scale << 3) & 0x18;
	I2C_Write8(GYRO_CONFIG_REG, Buffer);

	/* Select the Accelerometer Full Scale Range */
	Buffer = 0;
	Buffer = (config->Accel_Full_Scale << 3) & 0x18;
	I2C_Write8(ACCEL_CONFIG_REG, Buffer);
	/* Set SRD To Default */
	MPU6050_Set_SMPRT_DIV(0x04);


	/* Accelerometer Scaling Factor, Set the Accelerometer and Gyroscope Scaling Factor */
```

```
	switch (config->Accel_Full_Scale)
	{
		case AFS_SEL_2g:
			accelScalingFactor = (2000.0f/32768.0f);
			break;

		case AFS_SEL_4g:
			accelScalingFactor = (4000.0f/32768.0f);
				break;

		case AFS_SEL_8g:
			accelScalingFactor = (8000.0f/32768.0f);
			break;

		case AFS_SEL_16g:
			accelScalingFactor = (16000.0f/32768.0f);
			break;

		default:
			break;
	}
	/* Gyroscope Scaling Factor */
	switch (config->Gyro_Full_Scale)
	{
		case FS_SEL_250:
			gyroScalingFactor = 250.0f/32768.0f;
			break;

		case FS_SEL_500:
				gyroScalingFactor = 500.0f/32768.0f;
				break;

		case FS_SEL_1000:
			gyroScalingFactor = 1000.0f/32768.0f;
			break;

		case FS_SEL_2000:
			gyroScalingFactor = 2000.0f/32768.0f;
			break;

		default:
			break;
	}

}

/* 5- Get Sample Rate Divider */
uint8_t MPU6050_Get_SMPRT_DIV(void)
{
	uint8_t Buffer = 0;

	I2C_Read(SMPLRT_DIV_REG, &Buffer, 1);
	return Buffer;
```

```
}

/* 6- Set Sample Rate Divider */
void MPU6050_Set_SMPRT_DIV(uint8_t SMPRTvalue)
{
	I2C_Write8(SMPLRT_DIV_REG, SMPRTvalue);
}

/* 7- Get External Frame Sync.*/
uint8_t MPU6050_Get_FSYNC(void)
{
	uint8_t Buffer = 0;

	I2C_Read(CONFIG_REG, &Buffer, 1);
	Buffer &= 0x38;
	return (Buffer>>3);
}

/* 8- Set External Frame Sync. */
void MPU6050_Set_FSYNC(enum EXT_SYNC_SET_ENUM ext_Sync)
{
	uint8_t Buffer = 0;
	I2C_Read(CONFIG_REG, &Buffer,1);
	Buffer &= ~0x38;

	Buffer |= (ext_Sync <<3);
	I2C_Write8(CONFIG_REG, Buffer);

}

/* 9- Get Accel Raw Data */
void MPU6050_Get_Accel_RawData(RawData_Def *rawDef)
{
	uint8_t i2cBuf[2];
	uint8_t AcceArr[6], GyroArr[6];

	I2C_Read(INT_STATUS_REG, &i2cBuf[1],1);
	if((i2cBuf[1]&&0x01))
	{
		I2C_Read(ACCEL_XOUT_H_REG, AcceArr,6);
		/* Accel Raw Data */
		rawDef->x = ((AcceArr[0]<<8) + AcceArr[1]); // x-Axis
		rawDef->y = ((AcceArr[2]<<8) + AcceArr[3]); // y-Axis
		rawDef->z = ((AcceArr[4]<<8) + AcceArr[5]); // z-Axis
		/* Gyro Raw Data */
		I2C_Read(GYRO_XOUT_H_REG, GyroArr,6);
		GyroRW[0] = ((GyroArr[0]<<8) + GyroArr[1]);
		GyroRW[1] = (GyroArr[2]<<8) + GyroArr[3];
		GyroRW[2] = ((GyroArr[4]<<8) + GyroArr[5]);
	}
}

/* 10- Get Accel scaled data (g unit of gravity, 1g = 9.81m/s2) */
void MPU6050_Get_Accel_Scale(ScaledData_Def *scaledDef)
```

```
{
	RawData_Def AccelRData;
	MPU6050_Get_Accel_RawData(&AccelRData);
	/* Accel Scale data */
	scaledDef->x = ((AccelRData.x+0.0f)*accelScalingFactor);
	scaledDef->y = ((AccelRData.y+0.0f)*accelScalingFactor);
	scaledDef->z = ((AccelRData.z+0.0f)*accelScalingFactor);
}

/* 11- Get Accel calibrated data */
void MPU6050_Get_Accel_Cali(ScaledData_Def *CaliDef)
{
	ScaledData_Def AccelScaled;
	MPU6050_Get_Accel_Scale(&AccelScaled);
	/* Accel Scale data */
	CaliDef->x = (AccelScaled.x) - A_X_Bias; // x-Axis
	CaliDef->y = (AccelScaled.y) - A_Y_Bias; // y-Axis
	CaliDef->z = (AccelScaled.z) - A_Z_Bias; // z-Axis
}

/* 12- Get Gyro Raw Data */
void MPU6050_Get_Gyro_RawData(RawData_Def *rawDef)
{
	/* Accel Raw Data */
	rawDef->x = GyroRW[0];
	rawDef->y = GyroRW[1];
	rawDef->z = GyroRW[2];
}

/* 13- Get Gyro scaled data */
void MPU6050_Get_Gyro_Scale(ScaledData_Def *scaledDef)
{
	RawData_Def myGyroRaw;
	MPU6050_Get_Gyro_RawData(&myGyroRaw);
	/* Gyro Scale data */
	scaledDef->x = (myGyroRaw.x)*gyroScalingFactor; // x-Axis
	scaledDef->y = (myGyroRaw.y)*gyroScalingFactor; // y-Axis
	scaledDef->z = (myGyroRaw.z)*gyroScalingFactor; // z-Axis
}
```

MPU6050 Header File

```
/* Header Files */
#include "stm32f1xx_hal.h"
#include <string.h>
#include <stdbool.h>		//Boolean
#include <math.h>		//Pow()
/* Define Registers */
#define WHO_AM_I_REG		0x75
#define MPU_ADDR		0x68
#define PWR_MAGT_1_REG		0x6B
#define CONFIG_REG		0x1A
#define GYRO_CONFIG_REG		0x1B
#define ACCEL_CONFIG_REG	0x1C
```

```
#define SMPLRT_DIV_REG          0x19
#define INT_STATUS_REG          0x3A
#define ACCEL_XOUT_H_REG        0x3B
#define TEMP_OUT_H_REG          0x41
#define GYRO_XOUT_H_REG         0x43
#define FIFO_EN_REG             0x23
#define INT_ENABLE_REG          0x38
#define I2CMACO_REG             0x23
#define USER_CNT_REG            0x6A
#define FIFO_COUNTH_REG         0x72
#define FIFO_R_W_REG            0x74
/* TypeDefs and Enums */
/* 1- MPU Configuration */
typedef struct
{
        uint8_t ClockSource;
        uint8_t Gyro_Full_Scale;
        uint8_t Accel_Full_Scale;
        uint8_t CONFIG_DLPF;
        bool Sleep_Mode_Bit;
}MPU_ConfigTypeDef;
/* 2- Clock Source ENUM */
enum PM_CLKSEL_ENUM
{
        Internal_8MHz = 0x00,
        X_Axis_Ref= 0x01,
        Y_Axis_Ref= 0x02,
        Z_Axis_Ref= 0x03,
        Ext_32_768KHz= 0x04,
        Ext_19_2MHz= 0x05,
        TIM_GENT_INREST= 0x07
};
/* 3- Gyro Full Scale Range ENUM (deg/sec) */
enum gyro_FullScale_ENUM
{
        FS_SEL_250 = 0x00,
        FS_SEL_500 = 0x01,
        FS_SEL_1000 = 0x02,
        FS_SEL_2000= 0x03
};
/* 4- Accelerometer Full Scale Range ENUM (1g = 9.81m/s2) */
enum accel_FullScale_ENUM
{
        AFS_SEL_2g= 0x00,
        AFS_SEL_4g,
        AFS_SEL_8g,
        AFS_SEL_16g
};
/* 5- Digital Low Pass Filter ENUM */
enum DLPF_CFG_ENUM
{
        DLPF_260A_256G_Hz = 0x00,
        DLPF_184A_188G_Hz = 0x01,
```

```
	DLPF_94A_98G_Hz = 0x02,
	DLPF_44A_42G_Hz = 0x03,
	DLPF_21A_20G_Hz = 0x04,
	DLPF_10_Hz = 0x05,
	DLPF_5_Hz = 0x06
};
/* 6- e external Frame Synchronization ENUM */
enum EXT_SYNC_SET_ENUM
{
	input_Disable = 0x00,
	TEMP_OUT_L= 0x01,
	GYRO_XOUT_L= 0x02,
	GYRO_YOUT_L= 0x03,
	GYRO_ZOUT_L= 0x04,
	ACCEL_XOUT_L= 0x05,
	ACCEL_YOUT_L= 0x06,
	ACCEL_ZOUT_L= 0x07
};

/* 7. Raw data typedef */
typedef struct
{
	int16_t x;
	int16_t y;
	int16_t z;
}RawData_Def;

/* 8. Scaled data typedef */
typedef struct
{
	float x;
	float y;
	float z;
}ScaledData_Def;

/* Function Prototype */
/* 1- i2c Handler */
void MPU6050_Init(I2C_HandleTypeDef *I2Chnd);
/* 2- i2c Read */
void I2C_Read(uint8_t ADDR, uint8_t *i2cBuf, uint8_t NofData);
/* 3- i2c Write 8 Bit */
void I2C_Write8(uint8_t ADDR, uint8_t data);
/* 4- MPU6050 Initialization Configuration */
void MPU6050_Config(MPU_ConfigTypeDef *config);
/* 5- Get Sample Rate Divider */
uint8_t MPU6050_Get_SMPRT_DIV(void);
/* 6- Set Sample Rate Divider */
void MPU6050_Set_SMPRT_DIV(uint8_t SMPRTvalue);
/* 7- External Frame Sync.*/
uint8_t MPU6050_Get_FSYNC(void);
/* 8- Set External Frame Sync.*/
void MPU6050_Set_FSYNC(enum EXT_SYNC_SET_ENUM ext_Sync);
/* 9- Get Accel Raw Data */
```

```
void MPU6050_Get_Accel_RawData(RawData_Def *rawDef);
/* 10- Get Accel scaled data */
void MPU6050_Get_Accel_Scale(ScaledData_Def *scaledDef);
/* 11- Get Accel calibrated data */
void MPU6050_Get_Accel_Cali(ScaledData_Def *CaliDef);
/* 12- Get Gyro Raw Data */
void MPU6050_Get_Gyro_RawData(RawData_Def *rawDef);
/* 13- Get Gyro scaled data */
void MPU6050_Get_Gyro_Scale(ScaledData_Def *scaledDef);
```

LCD 1602 C File

```
/* Header files */
#include "STM_MY_LCD16X2.h"

/* Variables */
static const uint32_t writeTimeConstant = 10;
static uint8_t mode_8_4_I2C = 1;
static GPIO_TypeDef* PORT_RS_and_E;          // RS and E PORT
static uint16_t PIN_RS, PIN_E;               // RS and E pins
static GPIO_TypeDef* PORT_LSB;               // LSBs D0, D1, D2 and D3 PORT
static uint16_t D0_PIN, D1_PIN, D2_PIN, D3_PIN;   // LSBs D0, D1, D2 and D3 pins
static GPIO_TypeDef* PORT_MSB;               // MSBs D5, D6, D7 and D8 PORT
static uint16_t D4_PIN, D5_PIN, D6_PIN, D7_PIN;   // MSBs D5, D6, D7 and D8 pins

static uint8_t DisplayControl = 0x0F;
static uint8_t FunctionSet = 0x38;


/* Functions definitions */
/* Private functions */
/* 1) Enable EN pulse */
static void LCD1602_EnablePulse(void)
{
	HAL_GPIO_WritePin(PORT_RS_and_E, PIN_E, GPIO_PIN_SET);
	LCD1602_TIM_MicorSecDelay(writeTimeConstant);
	HAL_GPIO_WritePin(PORT_RS_and_E, PIN_E, GPIO_PIN_RESET);
	LCD1602_TIM_MicorSecDelay(60);
}
/* 2) RS control */
static void LCD1602_RS(_Bool state)
{
	if(state) HAL_GPIO_WritePin(PORT_RS_and_E, PIN_RS, GPIO_PIN_SET);
	else HAL_GPIO_WritePin(PORT_RS_and_E, PIN_RS, GPIO_PIN_RESET);
}

/* 3) Write Parallel interface */
static void LCD1602_write(uint8_t byte)
{
	uint8_t LSB_nibble = byte&0xF, MSB_nibble = (byte>>4)&0xF;

	if(mode_8_4_I2C == 1)		//8bits mode
	{
```

```
		/* write data to output pins */
		/* LSB data */
		HAL_GPIO_WritePin(PORT_LSB, D0_PIN, (GPIO_PinState)(LSB_nibble&0x1));
		HAL_GPIO_WritePin(PORT_LSB, D1_PIN, (GPIO_PinState)(LSB_nibble&0x2));
		HAL_GPIO_WritePin(PORT_LSB, D2_PIN, (GPIO_PinState)(LSB_nibble&0x4));
		HAL_GPIO_WritePin(PORT_LSB, D3_PIN, (GPIO_PinState)(LSB_nibble&0x8));
		/* MSB data */
		HAL_GPIO_WritePin(PORT_MSB, D4_PIN, (GPIO_PinState)(MSB_nibble&0x1));
		HAL_GPIO_WritePin(PORT_MSB, D5_PIN, (GPIO_PinState)(MSB_nibble&0x2));
		HAL_GPIO_WritePin(PORT_MSB, D6_PIN, (GPIO_PinState)(MSB_nibble&0x4));
		HAL_GPIO_WritePin(PORT_MSB, D7_PIN, (GPIO_PinState)(MSB_nibble&0x8));
		/* Write the Enable pulse */
		LCD1602_EnablePulse();
	}
	else if(mode_8_4_I2C == 2)		//4 bits mode
	{
		/* write data to output pins */
		/* MSB data */
		HAL_GPIO_WritePin(PORT_MSB, D4_PIN, (GPIO_PinState)(MSB_nibble&0x1));
		HAL_GPIO_WritePin(PORT_MSB, D5_PIN, (GPIO_PinState)(MSB_nibble&0x2));
		HAL_GPIO_WritePin(PORT_MSB, D6_PIN, (GPIO_PinState)(MSB_nibble&0x4));
		HAL_GPIO_WritePin(PORT_MSB, D7_PIN, (GPIO_PinState)(MSB_nibble&0x8));
		/* Write the Enable pulse */
		LCD1602_EnablePulse();

		/* LSB data */
		HAL_GPIO_WritePin(PORT_MSB, D4_PIN, (GPIO_PinState)(LSB_nibble&0x1));
		HAL_GPIO_WritePin(PORT_MSB, D5_PIN, (GPIO_PinState)(LSB_nibble&0x2));
		HAL_GPIO_WritePin(PORT_MSB, D6_PIN, (GPIO_PinState)(LSB_nibble&0x4));
		HAL_GPIO_WritePin(PORT_MSB, D7_PIN, (GPIO_PinState)(LSB_nibble&0x8));
		/* Write the Enable pulse */
		LCD1602_EnablePulse();
	}
}
/* 4) Microsecond delay functions */
static void LCD1602_TIM_Config(void)
{
	RCC_ClkInitTypeDef myCLKtypeDef;
	uint32_t clockSpeed;
	uint32_t flashLatencyVar;
	HAL_RCC_GetClockConfig(&myCLKtypeDef, &flashLatencyVar);
	if(myCLKtypeDef.APB1CLKDivider == RCC_HCLK_DIV1)
	{
		clockSpeed = HAL_RCC_GetPCLK1Freq();
	}
	else
	{
		clockSpeed = HAL_RCC_GetPCLK1Freq()*2;
	}
	clockSpeed *= 0.000001;
	/* Enable clock for TIM2 timer */
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;  // 0x1
	/* Set the mode to Count up */
```

```
	TIM2->CR1 &= ~(0x0010);
	/* Enable Update Event */
	TIM2->CR1 &= ~(0x0001);
	/* Update request source */
	TIM2->CR1 &= ~(1UL << 2);
	/* Set bit 3 High to enable One-Pulse mode */
	TIM2->CR1 |= (1UL << 3);
	/* Set the Prescalar */
	TIM2->PSC = clockSpeed-1;
	/* Set and Auto-Reload Value to delay the timer 1 sec */
	TIM2->ARR = 10-1;		// The Flag sets when overflows
	/* Event generation handling to reset the counter */
	TIM2->EGR = 1;			//Update generate auto
	TIM2->SR &= ~(0x0001);	//Clear Update interrupt flag
}
void LCD1602_TIM_MicorSecDelay(uint32_t uSecDelay)
{
	TIM2->ARR = uSecDelay-1;
	TIM2->SR &= ~(0x0001);  // Clear UEV flag
	TIM2->CR1 |= 1UL;
	while((TIM2->SR&0x0001) != 1);
}
/* 5) Write command */
static void LCD1602_writeCommand(uint8_t command)
{
	/* Set RS to 0 */
	LCD1602_RS(0);
	/* Call low level write parallel function */
	LCD1602_write(command);
}
/* 6) Write 4 bits command, *FOR 4 BITS MODE ONLY */
static void LCD1602_write4bitCommand(uint8_t nibble)
{
	uint8_t LSB_nibble = nibble&0xF;
	/* Set RS to 0 */
	LCD1602_RS(0);
	/* LSB data */
	HAL_GPIO_WritePin(PORT_MSB, D4_PIN, (GPIO_PinState)(LSB_nibble&0x1));
	HAL_GPIO_WritePin(PORT_MSB, D5_PIN, (GPIO_PinState)(LSB_nibble&0x2));
	HAL_GPIO_WritePin(PORT_MSB, D6_PIN, (GPIO_PinState)(LSB_nibble&0x4));
	HAL_GPIO_WritePin(PORT_MSB, D7_PIN, (GPIO_PinState)(LSB_nibble&0x8));
	/* Write the Enable pulse */
	LCD1602_EnablePulse();
}
/* Public functions */
/* 1) LCD begin 4 bits function */
void LCD1602_Begin4BIT(GPIO_TypeDef* PORT_RS_E, uint16_t RS, uint16_t E, GPIO_TypeDef* PORT_MSBs4to7, uint16_t
D4, uint16_t D5, uint16_t D6, uint16_t D7)
{
	/* Set GPIO Ports and Pins data */
	PORT_RS_and_E = PORT_RS_E;
	PIN_RS = RS;
	PIN_E = E;
```

```
	PORT_MSB = PORT_MSBs4to7;
	D4_PIN = D4;
	D5_PIN = D5;
	D6_PIN = D6;
	D7_PIN = D7;
	/* Initialise microsecond timer */
	LCD1602_TIM_Config();
	/* Set the mode to 4 bits */
	mode_8_4_I2C = 2;
	/* Function set variable to 4 bits mode */
	FunctionSet = 0x28;
	/* Initialise LCD */
	/* 1. Wait at least 15ms */
	HAL_Delay(100);
	/* 2. Attentions sequence */
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(50);
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(10);
	LCD1602_write4bitCommand(0x3);
	HAL_Delay(10);
	LCD1602_write4bitCommand(0x2);  //4 bit mode
	HAL_Delay(10);
	/* 3. Display control (Display ON, Cursor ON, blink cursor) */
	LCD1602_writeCommand(LCD_DISPLAYCONTROL | LCD_DISPLAY_B | LCD_DISPLAY_C | LCD_DISPLAY_D);
	/* 4. Clear LCD and return home */
	LCD1602_writeCommand(LCD_CLEARDISPLAY);
	HAL_Delay(30);
	/* 5. Function set; Enable 2 lines, Data length to 8 bits */
	LCD1602_writeCommand(LCD_FUNCTIONSET | LCD_FUNCTION_N);
	HAL_Delay(30);
}
/* 2) LCD print string */
void LCD1602_print(char string[])
{
	for(uint8_t i=0;  i< 16 && string[i]!=NULL; i++)
	{
		LCD1602_writeData((uint8_t)string[i]);
	}
}
/* 3) set cursor position */
void LCD1602_setCursor(uint8_t row, uint8_t col)
{
	uint8_t maskData;
	maskData = (col-1)&0x0F;
	if(row==1)
	{
		maskData |= (0x80);
		LCD1602_writeCommand(maskData);
	}
	else
	{
		maskData |= (0xc0);
```

```
                LCD1602_writeCommand(maskData);
        }
}
void LCD1602_1stLine(void)
{
        LCD1602_setCursor(1,1);
}
void LCD1602_2ndLine(void)
{
        LCD1602_setCursor(2,1);
}
/* 4) Enable two lines */
void LCD1602_TwoLines(void)
{
        FunctionSet |= (0x08);
        LCD1602_writeCommand(FunctionSet);
}
void LCD1602_OneLine(void)
{
        FunctionSet &= ~(0x08);
        LCD1602_writeCommand(FunctionSet);
}
/* 5) Cursor ON/OFF */
void LCD1602_noCursor(void)
{
        DisplayControl &= ~(0x02);
        LCD1602_writeCommand(DisplayControl);
}
void LCD1602_cursor(void)
{
        DisplayControl |= (0x02);
        LCD1602_writeCommand(DisplayControl);
}
/* 6) Clear display */
void LCD1602_clear(void)
{
        LCD1602_writeCommand(LCD_CLEARDISPLAY);
        HAL_Delay(3);
}
/* 7) Blinking cursor */
void LCD1602_noBlink(void)
{
        DisplayControl &= ~(0x01);
        LCD1602_writeCommand(DisplayControl);
}
void LCD1602_blink(void)
{
        DisplayControl |= 0x01;
        LCD1602_writeCommand(DisplayControl);
}
/* 8) Display ON/OFF */
void LCD1602_noDisplay(void)
{
```

```
	DisplayControl &= ~(0x04);
	LCD1602_writeCommand(DisplayControl);
}
void LCD1602_display(void)
{
	DisplayControl |= (0x04);
	LCD1602_writeCommand(DisplayControl);
}
/* 9) Shift Display, right or left */
void LCD1602_shiftToRight(uint8_t num)
{
	for(uint8_t i=0; i<num;i++)
	{
		LCD1602_writeCommand(0x1c);
	}
}
void LCD1602_shiftToLeft(uint8_t num)
{
	for(uint8_t i=0; i<num;i++)
	{
		LCD1602_writeCommand(0x18);
	}
}
/* Print numbers to LCD */
/* 1. Integer */
void LCD1602_PrintInt(int number)
{
	char numStr[16];
	sprintf(numStr,"%d", number);
	LCD1602_print(numStr);
}
/* 2. Float */
void LCD1602_PrintFloat(float number, int decimalPoints)
{
	char numStr[16];
	sprintf(numStr,"%.*f",decimalPoints, number);
	LCD1602_print(numStr);
}
```

LCD1602 Header File

```
/* Header files */
#include <stdbool.h>
#include "stm32f1xx_hal.h"
#include <stdlib.h>

/* List of COMMANDS */

#define LCD_CLEARDISPLAY        0x01
#define LCD_RETURNHOME          0x02
#define LCD_ENTRYMODESET        0x04
#define LCD_DISPLAYCONTROL      0x08
#define LCD_CURSORSHIFT         0x10
#define LCD_FUNCTIONSET         0x20
```

```
#define LCD_SETCGRAMADDR        0x40
#define LCD_SETDDRAMADDR        0x80

/* List of commands Bitfields */

/* 1) Entry mode Bitfields */
#define LCD_ENTRY_SH            0x01
#define LCD_ENTRY_ID            0x02
/* 2) Entry mode Bitfields */
#define LCD_ENTRY_SH            0x01
#define LCD_ENTRY_ID            0x02
/* 3) Display control */
#define LCD_DISPLAY_B           0x01
#define LCD_DISPLAY_C           0x02
#define LCD_DISPLAY_D           0x04
/* 4) Shift control */
#define LCD_SHIFT_RL            0x04
#define LCD_SHIFT_SC            0x08
/* 5) Function set control */
#define LCD_FUNCTION_F          0x04
#define LCD_FUNCTION_N          0x08
#define LCD_FUNCTION_DL         0x10

/* Functions prototypes */

/* Private functions */
/* 1) Enable EN pulse */
static void LCD1602_EnablePulse(void);
/* 2) RS control */
static void LCD1602_RS(bool state);
/* 3) Write Parallel interface */
static void LCD1602_write(uint8_t byte);
/* 4) Microsecond delay functions */
static void LCD1602_TIM_Config(void);
/* 5) Write command */
static void LCD1602_writeCommand(uint8_t command);
/* 6) Write 4 bits command, *FOR 4 BITS MODE ONLY* */
static void LCD1602_write4bitCommand(uint8_t nibble);
/* Public functions */
/* 1) LCD begin 8 bits function */
void LCD1602_Begin8BIT(GPIO_TypeDef* PORT_RS_E, uint16_t RS, uint16_t E, GPIO_TypeDef* PORT_LSBs0to3, uint16_t
D0, uint16_t D1, uint16_t D2, uint16_t D3, GPIO_TypeDef* PORT_MSBs4to7, uint16_t D4, uint16_t D5, uint16_t D6,
uint16_t D7);
/* 2) LCD begin 4 bits function */
void LCD1602_Begin4BIT(GPIO_TypeDef* PORT_RS_E, uint16_t RS, uint16_t E, GPIO_TypeDef* PORT_MSBs4to7, uint16_t
D4, uint16_t D5, uint16_t D6, uint16_t D7);
/* 3) LCD print string */
void LCD1602_print(char string[]);
/* 4) set cursor position */
void LCD1602_setCursor(uint8_t row, uint8_t col);
void LCD1602_1stLine(void);
void LCD1602_2ndLine(void);
/* 5) Enable two lines */
```

```
void LCD1602_TwoLines(void);
void LCD1602_OneLine(void);
/* 6) Cursor ON/OFF */
void LCD1602_noCursor(void);
void LCD1602_cursor(void);
/* 7) Clear display */
void LCD1602_clear(void);
/* 8) Blinking cursor */
void LCD1602_noBlink(void);
void LCD1602_blink(void);
/* 9) Display ON/OFF */
void LCD1602_noDisplay(void);
void LCD1602_display(void);
/* 10) Shift Display, right or left */
void LCD1602_shiftToRight(uint8_t num);
void LCD1602_shiftToLeft(uint8_t num);

void LCD1602_TIM_MicorSecDelay(uint32_t uSecDelay);

/* Print numbers to LCD */
/* 1. Integer */
void LCD1602_PrintInt(int number);
/* 2. Float */
void LCD1602_PrintFloat(float number, int decimalPoints);
```

REFERENCES

[1] S. J. (2008) Rai, L.; Kang, “Rule-based modular software and hardware architecture for multi-shaped robots using real-time dynamic behavior identification and selection". Knowledge-Based Systems,” pp. 273–283.

[2] FOUNDERS GUIDE, “What is Firmware and Why is Firmware Development Crucial?,” *FOUNDERS GUIDE*, 2018. http://foundersguide.com/firmware-and-why-firmware-development-crucial/.

[3] Embedded Staff, “Basics of hardware/firmware interface codesign,” 2013. https://www.embedded.com/basics-of-hardware-firmware-interface-codesign/.

[4] “Thermistor.” http://wiki.sunfounder.cc/images/d/d1/Thermistoren.pdf.

[5] E. Ag, “NTC thermistors for temperature measurement,” no. March, pp. 1–22, 2013.

[6] L. Aosong ELectronics Co, “Dht22 (Am2302),” vol. 22, pp. 1–10, 2015, Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf%0Ahttps://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf.

[7] R. Sensors, “59630 Sensor with Integral Float Actuator 59630 Sensor with Integral Float Actuator,” 2015.

[8] “DC Fan.” Available: https://www.farnell.com/datasheets/1878245.pdf

[9] “Durevole Dc 12 V 70PSI 3.5L/Min Agricola Pompa Ad Acqua Elettrica Nero Micro a Membrana Ad Alta Pressione Spruzzatore di Acqua Auto di Lavaggio 12 V.”

[10] “WaveShare LCD1602.”

[11] “HSMx-Cxxx Surface Mount Chip LEDs.”

[12] TDK, “Pin terminal / Lead Without oscillator circuit PS series Piezoelectronic Buzzers (without circuit) PS Series (Pin Terminal / Lead),” 2011. .

[13] adafruit, “MEMBRANE 1X4 KEYPAD + EXTRAS.” https://www.adafruit.com/product/1332.

[14] Texas Instruments, “Temperature sensing with NTC circuit,” 2018. .

[15] ON Semiconductor, “Single Supply Dual Operational Amplifiers,” *Order A J. Theory Ordered Sets Its Appl.*, pp. 1–14, 2003.

[16] Dejan, “DHT11 & DHT22 Sensors Temperature and Humidity,” 2016. https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino/.

[17] ELECT GO, “Relay,” 2019. https://www.electgo.com/what-is-a-relay/.

[18] Altium Designer, “Using Flyback Diodes in Relays Prevents Electrical Noise in Your Circuits,” 2017. https://resources.altium.com/p/why-you-should-use-a-flyback-diode-in-a-relay-to-prevent-electrical-noise-in-your-circuits.

[19] M. Brown, “FDD5612.” http://www.mouser.com/ds/2/308/LM7805-1124977.pdf.

[20] STM, “Uln2001, uln2002 uln2003, uln2004,” 2019. .

[21] Kitronik, “Kitronik Ltd - How a Darlington pair transistor works What is a Darlington Pair ? What is current gain? Why use a Darlington Pair? Base Activation Voltage.” https://www.kitronik.co.uk/blog/how-a-darlington-pair-transistor-works/.

[22] electronicsforu, “All About IC 7805 | Voltage Regulator,” 2019. https://www.electronicsforu.com/resources/learn-electronics/7805-ic-voltage-regulator.

[23] STM, “Adjustable and fixed low drop positive voltage regulator,” no. November, pp. 1–44, 2013, [Online]. Available: http://www.ti.com/lit/ds/symlink/tlv1117-50.pdf.

[24] A. Williams, “Build your own printed circuit board.” p. 206, 2003.

[25] C. Schroeder, “PCB Design Using AutoCAD,” *PCB Design Using AutoCAD*. 1997, doi: 10.1016/b978-0-7506-9834-4.x5000-0.

[26] ST, “STM32F101xx,102xx,103xx,105xx 107 Reference manual,” 2018. .

[27] InvenSense, “MPU-6000 and MPU-6050 Product Specification,” vol. 1, no. 408, 2012.