Hands on Hardware

Electrical Engineering Department, VJTI

Prithvi Tambewagh
F.Y. B.Tech. EXTC (2023-2027)

Acknowledgements

1. Dr. Sushama R. Wagh,
   HOD, Electrical Engineering Department, VJTI
2. Ameya Chavekar, CoE, VJTI

Abstract

This is the submission of the Winter Internship January 2024 - a report of the 'Hands on Hardware' workshop which was conducted by the Electrical Engineering Department (EED), VJTI. This flagship programme aims to provide insights to the candidates in the field of Embedded Systems and Robotics, from a beginner level to an expert level.

This journey starts at the basics of the field of Embedded Systems and Robotics, like the microcontrollers and communication protocols and gradually progresses to working on complex projects like understanding the sensors and various hardware accessories and methods and various kinds of projects.

An important aspect of this programme is that this internship opportunity was provided to students right from First Year to Final Year of B.Tech. programme. Also, the students were provided with the opportunity to work on various kinds of hardware components themselves, empowering them with skills, from basics like soldering, to advanced skills like coding microcontrollers.

Contents

Chapter 1

Soldering

Soldering is a very important skill in the field of electronics. It is the means by which we can join various electronic components together strongly.

1.1 Components used in Soldering Process

Figure 1.1: Soldering Iron, Soldering Flux and Soldering Wire

The main instrument used for soldering is the soldering iron, which is to be heated. This soldering iron, when heated, melts the soldering wire at the required spot. Soldering flux helps to make durable contact between the solder and the things to be soldered.

1.2 Soldering 2 wires together

1. Preheat the Soldering Iron.
2. Apply solder flux to the ends of both wires to be soldered.
3. By touching the soldering wire to the soldering iron, melt some drops of the soldering wire and put these drops on the intersection of the 2 wires.

1.2.1 Task :

Make shapes - rectangle, triangle and L-shape on a perf board using solder (soldering wire).

1.3 Removing solder from PCB (Printed Circuit Board)

Figure 1.2: Desoldering

1. Heat the soldering Iron.
2. Touch desoldering wick to the part of PCB from where solder is to be removed.
3. Touch the hot holdering iron on the other, exposed surface of the desoldering wick. The solder will come off, absorbed to the desoldering wick.

1.4 Removing a SMD (Surface Mount Device) from PCB

1. Apply SMD Flux paste on the IC/SMD to be removed.

Figure 1.3: Hot Air Gun

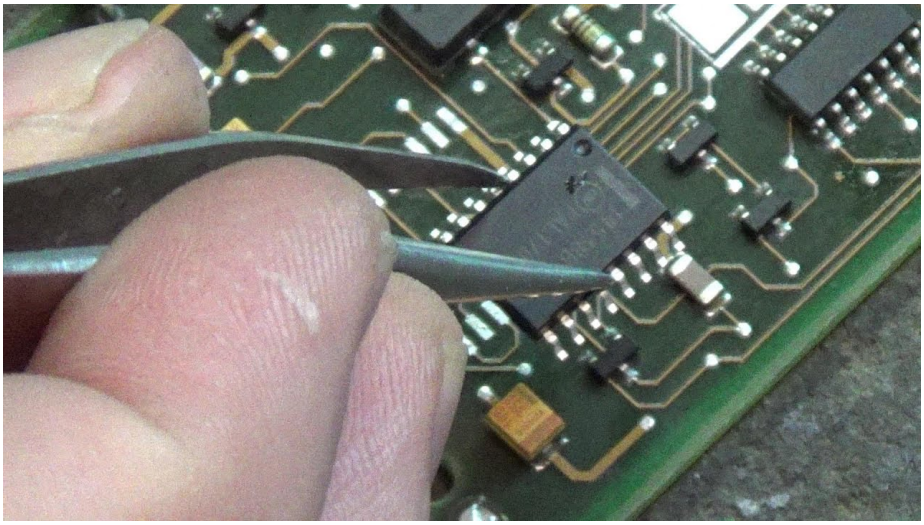

Figure 1.4: Removing SMD/IC from PCB

2. Heat the region of the PCB from where the SMD/IC is to be removed using a heat gun.
3. Keep blowing hot air from the hot air gun on the SMD.
4. Check whether the SMD starts becoming loose; as soon as it starts becoming loose, start removing it with tweezers.

1.5 Precautions :

1. Always keep soldering iron on soldering stand.
2. Never point hot air gun towards oneself or anyone.
3. Always check whether soldering iron has an intact wire; only in that case, plug it in & proceed to use it.

Chapter 2

Micrcontrollers

2.1 Concept

A microcontroller is a compact circuit, which does the processing part of an embedded/robotic system. It takes input from various sensor(s), applies logic based on its code and provides some output.

2.2 Parts of Microcontroller

2.2.1 RAM and ROM

'RAM' and 'ROM' respectively stand for 'Random Access Memeory' and 'Read Only Memory'. These are two types of memories, which store particular type of information. RAM is a volatile memory. The process of microprocessor chip to fetch required information from the large array of memory cells is time-consuming. So instead, some required pieces of information are transferred from the main memory i.e. ROM to RAM, which is a smaller sized memory compartment, making accessing information from it, easier. ROM is a kind of memory which cannot be directly accessed by the user, hence its name. It is the main memory of the microcontroller.

2.2.2 Ports

Ports are the places where different kinds of devices can be connected to the microcontroller. For instance, we can connect an ESP-32 using an USB port, to the computer.

2.2.3 Clock

'Clock' refers to the clock signal. It is a constantly pulsating signal, which causes changes in the state of the microcontroller, enabling it to perform computations. It is generally provided by a pulsating quartz crystal. Between the clock and the microcontroller is 'Prescaler'. It is a component, which scales down the clock by a factor; e.g. if the prescaler has a value 2, for a clock signal of 16MHz, the clock frequency received by the microcontroller would be

$$\frac{16\ MHz}{2} = 8\ MHz$$

We also can provide a higher than optimum value of the clock signal which is termed as 'Overclocking'; however, it reduces the lifetime of the microcontroller. Clock signal helps in synchronisation of the functioning of each circuit element.

2.2.4 Communication

Communication is what connects the microcontroller to its surroundings, empowering it with the capabilities of sensors.

Serial Communication :-

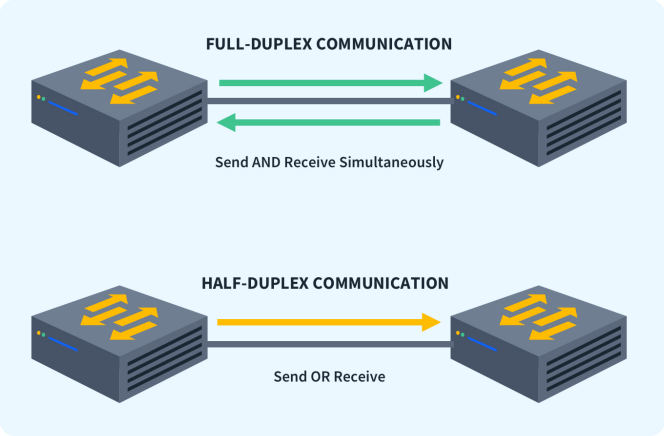

Figure 2.1: Half Duplex and Full Duplex

Here, there is kind of only 1 wire for all bits of communication, e.g. for every piece of data there is only 1 wire, so each bit is sent after a particular time interval. Serial Communication is of 2 types :-

1. Half Duplex :- Here, one of the 2 components which are computing, both are capable of sending and receiving information; however, only one can send/receive information at a time.
2. Full Duplex : -Here, both the components capable of sending and receiving information, however, both can send/receive information simultaneously.

Apart from these types, there are several types of Communication Protocols, as follows :-

1. **Serial Peripheral Interface (SPI) :-** In SPI, there is a pro-

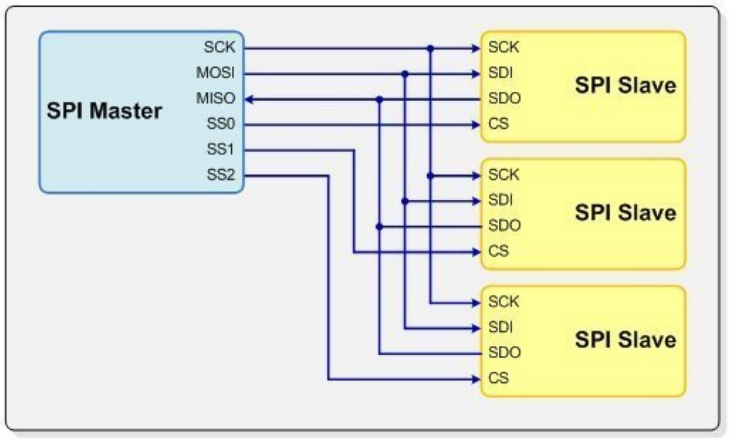

Figure 2.2: SPI Communication Protocol

cessor, which initiates commands, and hence, termed as 'Master'; there are several peripherals which respond to these commands; hence, they are termed as 'Slaves'. SCK i.e. Serial Clock Line maintains common clock signal between the master and the slaves for synchronisation of data transfer. There are 2 lines/paths of data transfer -

(a) MOSI (Master Out Slave In) - This line is used to send data from master to slave(s).

(b) MISO (Master In Slave Out) - This line is used by the master to receive response/data from the slaves.

Each slave also has a 'Chip select' Line, which tells the slave whether the master is instructing it to perform any task. The master has a slave select line for each slave. When the master turns the Slave Select line low, the slave is selected for communication. The master sends the data serially. If data is to be sent by the slave, it is also sent serially.

2. **Inter Integrated Circuit (I2C / IIC) :-** In I2C, there are

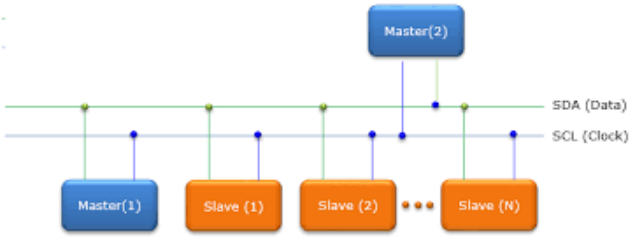

Figure 2.3: I2C Communication Protocol

only 2 lines - SCL i.e. Serial Clock Line for synchronisation of slaves and masters by clock signal and SDA i.e. Serial Data Line. The advantage of I2C is that in such network there can be more than one master, unlike SPI, where there can be only 1 master. Also, due to presence of only 2 data lines, the network is very compact. The format of data sent by I2C is as follows : Start bit initiates the conversation. The data is sent via

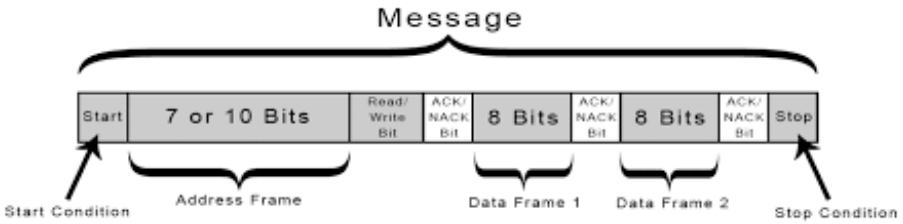

Figure 2.4: Format of data sent in I2C

the common SDA line. Address frame contains address of the intended slave; it is checked by each slave and the intended slave responds by sending a low signal acknowledgement bit

(ACK).Then it receives the first data frame. It is arranged such that most significant bit is placed first. After receiving each data frame, the slave sends acknowledgement bit to the corresponding master. To stop the communication, first the master makes SCL high, then SDA is made high.

3. **Universal Asyncronous Receiver/Transmitter (UART) and USART (Universal Synchronous/Asyncronous Receiver/Transmitter) :-** In UART, there is a transmitter and a receiver on both communicating devices. Data is transferred serially between the 2 devices at a constant rate known as 'baud rate' measured in bits/second. The baud rate cannot differ more than 10% than the predetermined value. As no clock signal is used for syncronisation, it is termed as 'asyncronous'. The 2 devices may receive data parallelly from their other ends, but while communicating with each other, they communicate serially. UART works in Full Duplex mode. The message sent over UART has a specific format as :- The start bit tells the

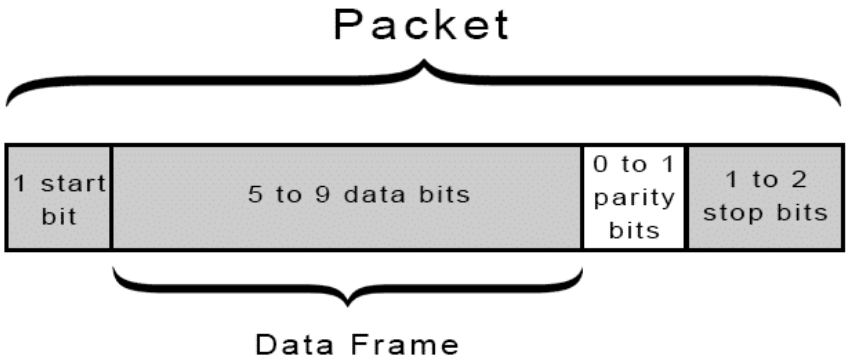

Figure 2.5: Format of data sent in UART

receiver that communication is being initiated. It is sent by turning transmission line from transmitter, low. Then the data is transferred serially. Data can be 5-8 bits long. If parity bit isn't used, it can even be 9 bits long. Parity bit is a bit used to ensure that data communicated is securely sent. If parity bit is low, the total number of high bits in the data being sent should be even, and vice versa. For stopping the communication, the

stop bits are sent, which have high state.

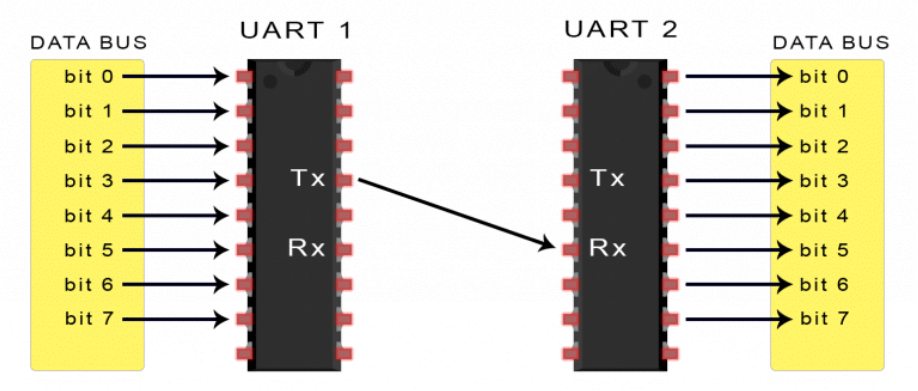

Figure 2.6: UART Communication Protocol

In USART, there are 4 pins on both of the communicating devices - one for data transmission, one for receiving data, one for clock (for synchronisation) and one for determining direction of flow of data.

There exist several families of microcontrollers like Arduino, ESP, STM, etc.

Chapter 3

Arduino Microcontroller

3.1 Structure of Arduino Uno Microcontroller Board

Arduino Board contains several components :-

Figure 3.1: Parts of Arduino Uno board

3.1.1 Microcontroller

Arduino Board uses ATmega328P microcontroller. It has RAM, ROM, Communication System and pins for various ports, all embedded into a single integrated circuit. It is a 8-bit system.

3.1.2 Power Port

Supplies power to whole Arduino Board. We can provide 7-12V DC through this power port.

3.1.3 Voltage Regulator

ATmega328P works on 1.8 - 5.5 V DC; so, to reduce the voltage from power port to the required value, a 'Voltage Regulator' is used.

3.1.4 Crystal Oscillator

The crystal oscillator provides the clock signal for the microcontroller as well as for the whole Arduino Board. It has a frequency of 16MHz.

3.1.5 Reset Switch

The reset switch, when pressed, briefly disconnects and reconnects the power supply to the board. After that, the microcontroller starts execution of the sketch (code) from the beginning.

3.1.6 Pins

There are 2 arrays of pins available on the Arduino Board.

Analog Input Pins -

The Analog input pins take input analog data from sensors, and send it for further processing by the method of Analog To Digital (ADC) Conversion. This is because the microontroller works on digital data. The ADC in Arduino Board Supports 10-bit resolution, i.e. data is converted from Analog to a digital signal with $2^{10} = 1024$ steps.

Digital Pins and PWM Pins -

The Digital Pins take digital data as input and send it to the microcontroller for further processing. Some of these digital pins can provide PWM (Pulse Width Modulation) Signal functionality. This is useful for operating components like Servo motors.

Power Pins -

These pins are located in the array of Analog Pins. These pins provide various pins for various standard voltage levels, like 0V (GND), 3.3V and 5V. There is also a pin termed as V_{in}. We can provide 7-12V DC Input Voltage to it; it is just an alternative to the Power Port.

3.1.7 USB Connector

USB Connector is a port where we can connect the Arduino Board to our computer for uploading sketches to it.

3.1.8 USB Interface Chip

This chip is another microcontroller, which converts the data being uploaded to the ATmeega328P microcontroller, via USB, to serial data, for it to be sent to ATmega328P. This is done with the help of a program known as a 'Bootloader'.

3.1.9 T_x, R_x LEDs

The T_x and R_x LEDs glow when data is being transmitted from the Arduino Board or received by the Arduino Board respectively.

3.2 Structure of Arduino Sketch :

Codes written in Arduino are termed as 'Sketches'. Sketches are compiled and uploaded to the Arduino Board via Arduino IDE. We can also code for ESP32 using Arduino IDE. Sketches have a general structure as follows :-

```
//Libraries and constants are present in this part
void setup(){
    //definition of input and output pins and variables is done here
}

void loop(){
    //code to be executed repeatedly is present here
}
//custom functions can be defined here
```

3.3 Common Errors

While working with Arduino, there can be several points of errors. this section describes some of the common errors -

1. Driver Issue - Download necessary driver/update/software.
2. Uploading error - Ensure proper connection of the cable between Arduino Board and the computer.

3.4 Motors

Motors are one of the important hardware components. They provide the aspect of motion to the embedded/robotic system

3.4.1 Types of Motors

1. **DC Motor** :- DC Motor stands for 'Direct Current' Motor. As its name suggests, it works on DC Supply and thus, single phase supply. As there are 2 inputs of a DC motor, it has 4 combinations of power supply. When there exists potential difference between the 2 pins, the motor exhibits rotational motion of its axle. Whenever the 2 pins are at 0V, no change occurs in the state of the motor, just the power supply to it is interrupted. If both pins are at operating voltage then the motor doesn't rotate, which is equivalent to suddenly braking the motor.

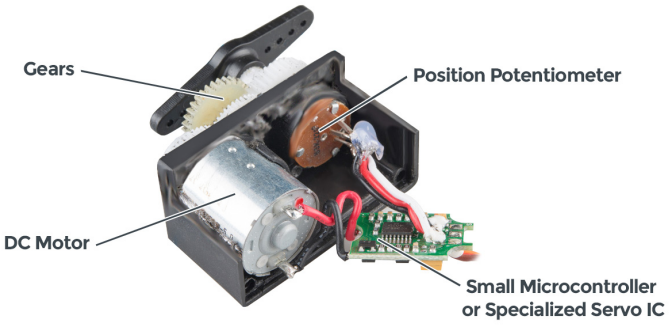

Figure 3.2: Parts of Servo Motor

2. **Servo Motor** :- Servo motor is a motor which can be programmed to hold the angular position steadily. General specifications include :-

   (a) Voltage : 5-8V DC

   (b) Stall Current : upto 1.4A

   Servo motors move slowly and hence provide greater torque. Their speed of operation is generally mentioned as their speed while moving from 0° to 30°. A servo motor has 3 terminals - V_{CC} (Red wire), GND (Brown wire) and PWM (Orange Wire). Voltage is provided to V_{CC} wire, GND wire is connected to ground and PWM signal is provided to PWM wire. The PWM signal provided, determines the final position of the Servo motor, from its range, which is usually 0° to 180°. Inside, a DC motor is connected to a small circuit and its axle to a geartrain, which magnifies the torque of the DC motor. The geartrain is connected to a potentiometer, which provides a feedback to the circuit of the servo motor regarding its current position. Based on this data, the circuit estimates the correction required and powers the DC motor accordingly. Servo Motors are used in making joints of robotic arms and manipulator arm to a great extent.

3. **Stepper Motor**:- Stepper Motors are another type of motors;

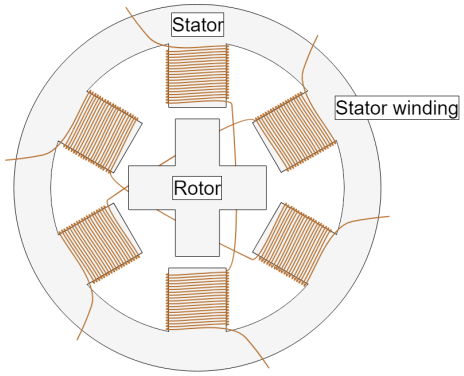

Figure 3.3: Basic Structure of a Stepper Motor

they are different because they use 2 or more coils for causing the rotation. Due to increased number of coils, the magnetic

field and hence, the power of the Stepper motor increases. The magnetic field varies alternately/in a pattern, causing the shaft to rotate in 'steps' in a small time interval.

4. **BLDC Motor**:- BLDC i.e. Brushless DC Motor, is a DC mo-

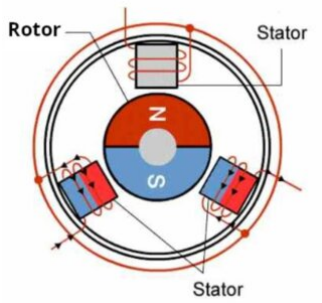

Figure 3.4: BLDC Motor

tor which doesn't use brushes which are used in conventional Electrical Motors. Here, in BLDC, the current carrying conductor is the stator, which is fixed; there are several coils in the stator. The rotor is a magnet. As the current in the stator coils is varied, the electromagnetic interaction between the stator and the rotor causes rotation of the rotor. Due to absence of brushes, BLDC motors can operate at high speeds.

3.5 Motor Driver

Motor Driver is an electronic circuit which interfaces motors with the microcontrller. Mtors work at voltage and current levels which are much higher than those of the microcontroller. Hence, we cannot interface motors with microcontroller directly. That's why, motor drivers are used. Some of the example of motor drivers include L293D, L2003D, L298N, etc. Most of the motor drivers are based on H-bridge circuit. Motor drivers are available as small, independent circuits or even as Integrated Circuits (ICs).

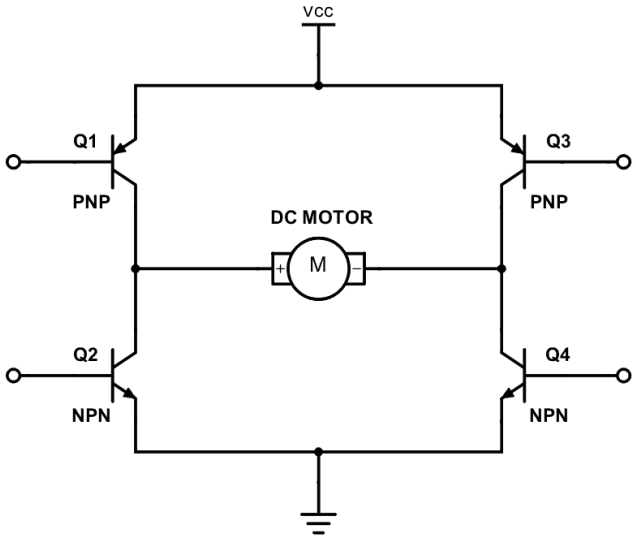

Figure 3.5: H-Bridge Circuit

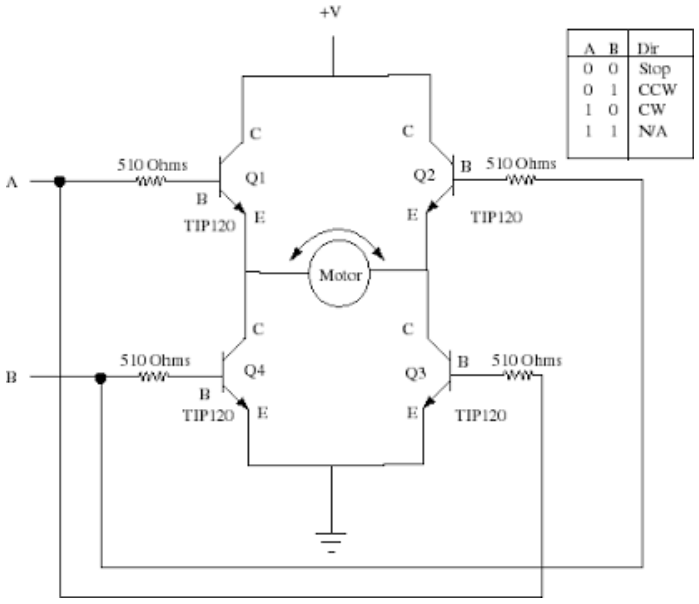

Figure 3.6: H-Bridge Operation

3.5.1 H-Bridge Circuit

Construction :-

In H-Bridge circuit, 4 transistors are used. Control signals are provided to base terminals of the transistors. Emitters of 2 pnp transistors are connected to the operating voltage of motors; their collectors are connected to the 2 terminals of the motor and the collectors of 2 npn transistors. The emitters of the 2 npn transistors are connected to ground. As shown in Fig. 3.6, the base terminals of transistors Q1 and Q3, and of Q2 and Q4 are shorted.

Working :-

Suppose, we give A a high signal and B, a low signal. In this case, current will flow from the motor via Q1 and Q3, and the motor will start rotating in 1 direction (let it be Clockwise for now). Likewise, if we interchange the signals provided to A and B, the motor will start rotating in opposite direction. If both A and B are low, the motor won't rotate. However, if both A and B are high, then the motor will come to a sudden halt from any of its previous state (if in motion); hence this condition is also used for braking.

3.5.2 Analysis of L298N Motor Driver

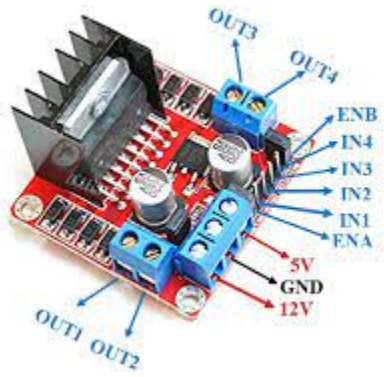

Figure 3.7: Pinout Diagram of L298N Motor Driver

L298N motor driver can control 2 motors simultaneously. The terminals of L298N are shown in Fig. 2.5. The 2 motors are connected to pair of pins OUT1, OUT2 and OUT3, OUT4. The volatge

required by motor i.e. 12V, voltage for the L298N circuit i.e. 5V and 0V for ground are supplied to the indicated pins. Each motor has 3 controlling pins. EN pin is the 'Enable' pin, an active high pin. It enables the motor to receive commands from the microcontroller. Pair of pins IN1, IN2 amd IN3, IN4 respectively perform the functions of pin A and pin B with reference to Section 3.5 above.

L298N has a voltage limit of 12V and a stall current of 1.4A. It is suitable for several applications.

3.6 Pulse Width Modulation

Pulse Width Modulation (PWM) refers to the technique in which we can control the average voltage delivered to a component. PWM is applicable to uniformly pulsating signals, like clock signals. Here, another term - 'Duty Cycle' comes into play.

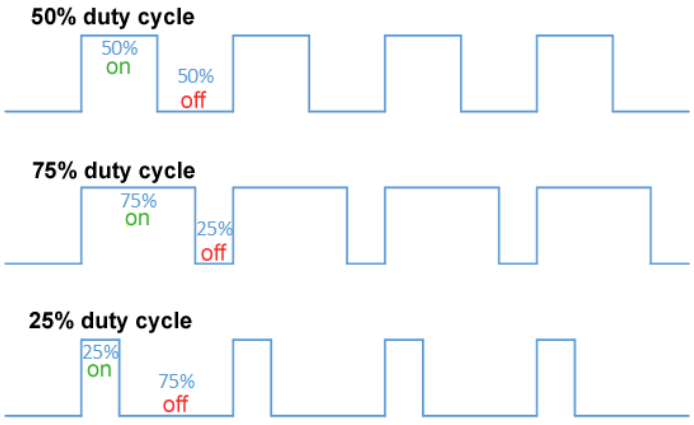

Figure 3.8: Duty Cycle

Duty Cycle is the percentage of the total time period of a uniformly pulsating signal, when the signal is high. The signal produced by applying a duty cycle is termed as a PWM signal.

For instance, if a PWM signal has maximum voltage of 12V and a duty cycle of 40%, then the average voltage delivered will be :

$$V_{out} = \frac{40 * 12}{100} = 4.8V$$

For instance, if this signal is connected to a motor, then the average voltage and thus, speed of the motor will be quite less, though in

practice we are providing it with 12V supply.Hence, PWM signal gives rise to apparent speed control. In reality, the motor is roatating for 40% of the time period of the signal and stopping for the remaining time; this repeats fr each of the pulse of the PWM signal which causes apparent speed reduction.

3.7 Diving Deep in the world of Sensors

Figure 3.9: Thermal Imaging

1. **Using Infrared Light -** IR sensor emits infrared light. A thermal camera works by using an IR sensor's analog output and mapping it to visible color range, then displaying the image. Night Vision goggles also work on similar principle. PIR (Passive Infrared) Sensor works by sensing Infrared light from its surroundings rather than emitting it.

2. **Using LDR (Light Dependent Resistor) Sensor -** Color sensors are made using LDR. An LDR is placed in the centre of the arrangement and red, blue and green colored LEDs are attached around it (as these are the primary colors of light). Depending on which wavelength i.e. color of light is reflected from the object in front of the sensor, on the LDR, the color of the object can be determined.

3.8 Planning involved in Robotics

Let us take a look in the processes involved in making a robot. Suppose we intend to make an agricultural robot. First of all, we will make a list of all the functionalities that we wish to implement in the robot. Then we check whether each of them is feasible or not, and work on the feasible ones.

3.8.1 Considerations

1. Soil Moisture sensor
2. Motor Driver
3. Power Source
4. Remote Control Implementation
5. Watering Pump using Servo motors

3.8.2 Calculations

Component-wise Distribution -

Sr. No.	Component	Voltage	Current	Quantity	Total Power
1.	MG996R	5V	4A	2	40W
2.	DC Gear Motor	12V	3A	4	144W
3.	Motor Driver	5V	0.3A	2	3W
4.	Bluetooth Module	5V	0.1A	1	0.5W
5.	Water Pump	12V	2A	1	24W

Battery calculations -

Now consider we have a 5000mAh battery. This means that the battery can provide a constant current of 5000mA=5A for 1 hour. This indicates that if a current of 10A is extracted from the battery, it can sustain for only half hour.

In our case, maximum required voltage is 12V and total power required is 211.5W. Hence, maximum current for this robot would be :

$$I_{ckt} = \frac{P}{V} = \frac{211.5}{12} = 17.625A$$

Thus, if we use a 10Ah battery, the time for which it will work is :

$$t = \frac{10\ Ah}{17.625\ A} = 0.5674\ hr = 0.5674 * 60 \approx 34\ min$$

Generally, Lithium polymer batteries are preferred, as they can sustain high current, are less temeperature sensitive and have high energy density. Batteries are generally available in configuartions like 3S (3 cells in paralel), 3S2P (2 batteries, each of 3 cells connected in series, in parallel), 6S, etc.

We also need to ensure that the wiring that we use for our robot can sustain the amount of current that will be flowing through it.

We can use sources of power like DC to AC converter or regulated power supply while testing. We must use a switch at the very beginning of the circuit, which makes it safer to replace components/battery independently, while avoiding any risk. We can also use a fuse in the circuit to prevent risk of short circuit.

There is a number known as 'C Rating' is present on batteries. When c rating of a battery is multiplied by its capacity/charge (in Ah) we get the maximum current that the battery can provide. For instance, for a battery of 10Ah, having a C rating of 26, maximum current,

$$I_{max} = 26 * 10 = 260A$$

For currents greater than this, battery degradation occurs.

DC-DC Buck Converter -

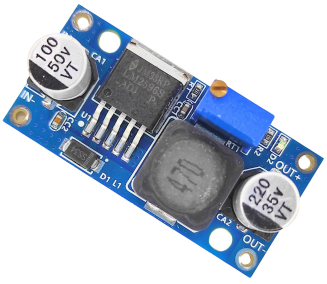

Figure 3.10: DC-DC Buck Converter

For the above robot, it is clear that we will need two power lines - one, of 5V and other, of 12V. Changing the supply voltage from a high voltage, like from 14.8V (4S battery) cannot be done directly; it can be done using DC-DC Buck Converter. Some buck converters also have a knob, connected to a potentiometer, by which we can set a value of output voltage, as well as, in some cases, output current, in the specified range. DC-DC Buck Converters have an efficiency of or over 90%.

Remote Control :-

For a remotely controlled robot, we can use a Bluetooth module. We can either use hardware like a remote or a software alternative like a mobile app, to control the robot remotely. The software alternative is also cost efficient. A great platform for making customized apps is MIT App Inventor.

3.9 Some commonly used Arduino IDE Commands

1. Serial.begin(9600) initializes the seial communication at a baud rate i.e. number of communicated bits per second.
2. Serial.print(" ") prints the text in the double quotation marks on the serial monitor.
3. 'Servo s' creates an object of the servo motor from the library 'Servo.h'.
4. s.attach(9) indicates that PWM signal is to be provided to the servo motor from pin D9.
5. s.write(57) indicates that we are making the servo motor attain an angular position of 57°.
6. pulseIn(pingPin, HIGH) measures time from the point pulse received at pingPin becomes HIGH until it becomes LOW again.
7. analogRead(A-pin) function is used to read data from analog pin 'A-pin'.

8. digitalRead(D-pin) function is used to read data from digital pin ‘D-pin’.

9. analogWrite(A-pin, A-state) is used to give analog data to pin ‘D-pin’; this analog data is ’A-state’, which can have a value from 0-255.

10. digitalWrite(D-pin, D-state) is used to give digital data to pin ‘D-pin’; this digital data is ‘D-state’, which can either be ‘HIGH’/1 or ‘LOW’/0. HIGH status means a current of 40mA.

11. delay(t) function pauses the program execution for ‘t’ milliseconds. Otherwise, the least delay is the inverse of the clock frequency.

12. map(var, from-low, from-high, to-low, to-high) maps the values of the variable ’var’ from limits [from-low, from-high] to limits [to-low, to-high].

3.10 Tasks

Note :- For downloading new libraries :-
Go to **Sketch →Library →Manage Libraries**

3.10.1 Varying LED Blink Rate using a Potentiometer

Here we aim to change the rate of blinking of inbuilt T_x LED, which is directly connected to pin D13 (‘D’ stands for Digital Pin) of the Arduino Board.

Connections :-

1. Connect the middle terminal of the potentiometer to pin A0 (’A’ stands for Analog Pin).

2. Connect the other 2 terminals of the potentiometer, one to a 5V pin and other, to a GND pin.

,

Code :-

```
void setup(){
    int pot;
    int del;
    pinMode(A0, INPUT);
    pinMode(D13, OUTPUT);
}

void loop(){
    pot=analogRead(A0);
    del=map(pot, 0, 1023, 0, 1000);
    digitalWrite(D13, HIGH);
    delay(del);
    digitalWrite(D13, LOW);
    delay(del);
}
```

Working :-

As per the connections of the potentiometer to the Arduino Board, as we move the wiper of the potentiometer, the resistance between the wiper pin and pin of the potemtiometer connected to GND and hence, the voltage at pin A0 will vary from 0-5V. As per this varying voltage, the rate of blinking of the LED will change.

3.10.2 Setting the position of a servo motor

Here we aim to set the position of a servo motor and gain insights into its programming.

Connections :-

1. Connect the orange, red and brown wires of the servo motors to pins D9, 5V and GND pins of Arduino Board.

Code :-

```
#include<Servo.h>
Servo s; //creating an object of servo motor from its library
```

```
void setup(){
    s.attach(9);
}

void loop(){
    s.write(57); //servo motor angular position at 57 degrees
}
```

Working :-

Here, as the value of the function 'write()' is changed, the angular position of the servo motor changes.

3.10.3 Oscillating Servo Motor

Here we aim to change the position of servo motor fro 0° to 90° and vice versa, so that it oscillates among these 2 positions.

Connections :-

Code :-

```
#include<Servo.h>
Servo s; //creating an object of servo motor from its library

void setup(){
    s.attach(9);
}

void loop(){
    s.write(0); //servo motor angular position at 57 degrees
    delay(500);
    s.write(90);
    delap(500);
}
```

Working :-

Here, as the value of the function 'write()' is changed, the angular position of the servo motor changes and we can change the position of servo motor continuously.

3.10.4 Controlling speed of DC Motor with Arduino Board

Here we aim to control speed of a DC motor with the Arduino board using PWM signal.

Connections :-

1. Connect the 2 terminals of the DC motor to OUT1 and OUT2 pins of the motor driver L298N.
2. Connect ENA pin of L298N to pin A0 of Arduino Board.
3. Connect the respective 5V, 12V and GND pins of the motor driver to respective power supply and Arduino power pins.
4. Connect IN1 and IN2 to pins D3 and D4 of the Arduino Board.

Code :-

```
void setup() {
  pinMode(A0, OUTPUT)
  piMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  analogWrite(A0, 210); //can give any value between 0-255
  digitalWrite(3, 1);
  digitalWrite(4, 0);
  delay(1000); //delay of 1 second
}
```

Working :-

The PWM signal has a resolution such that at the value of 255, duty cycle is 100%. So we can give any value from 0-255 on the analog pin A0. Higher the value, higher the speed.

3.10.5 Controlling direction a DC motor with Arduino Board

Here we aim to move DC motor colockwise and anticlockwise with the Arduino board and study its functioning.

Connections :-

1. Connect the 2 terminals of the DC motor to OUT1 and OUT2 pins of the motor driver L298N.
2. Connect ENA pin of L298N to pin A0 of Arduino Board.
3. Connect the respective 5V, 12V and GND pins of the motor driver to respective power supply and Arduino power pins.
4. Connect IN1 and IN2 to pins 3 and 4 of the Arduino Board.

Code :-

```
void setup() {
  pinMode(A0, OUTPUT);
  piMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop() {
  analogWrite(A0, 255); //enabling motor
  digitalWrite(3, 1); //Clockwise
  digitalWrite(4, 0);
  delay(1000); //delay of 1 second
  digitalWrite(3, 0); //Anticlockwise
  digitalWrite(4, 1);
  delay(1000); //delay of 1 second
}
```

Working :-

Initially, the motor rotates in clockwise direction for 1 second; then it rotates in anticlockwise direction for 1 second. This motion is repeated.

3.10.6 Using Custom Functions to control the motor

Here we aim to implement user defined functions to control the motor.

Connections :-

1. Connect the 2 terminals of the DC motor to OUT1 and OUT2 pins of the motor driver L298N.
2. Connect ENA pin of L298N to pin A0 of Arduino Board.
3. Connect the respective 5V, 12V and GND pins of the motor driver to respective power supply and Arduino power pins.
4. Connect IN1 and IN2 to pins 3 and 4 of the Arduino Board.

Code :-

```
#define enA A0
#define pinA 3
#define pinB 4

void setup() {
  pinMode(enA, OUTPUT);
  piMode(pinA, OUTPUT);
  pinMode(pinB, OUTPUT);
}

void loop() {
  forward();
  delay(1000); //delay of 1 second
  backward();
  delay(1000); //delay of 1 second
  stop();
  delay(1000); //delay of 1 second
}
void forward(){
  analogWrite(enA, 255); //enabling motor
  digitalWrite(pinA, 1);
  digitalWrite(pinB, 0);
}
void backward(){
  analogWrite(enA, 255); //enabling motor
  digitalWrite(pinA, 0);
  digitalWrite(pinB, 1);
```

```
}
void stop(){
  analogWrite(enA, 255);
  digitalWrite(pinA, 1);
  digitalWrite(pinB, 1);
}
```

Working :-

Here, we implement user defined functions to move the motor in forward and backward directions and to stop it, all with an interval of 1 second.

3.10.7 Interfacing IR Sensor with Arduino Board

Here we aim to connect an IR (Digital) sensor with the Arduino board and study its functioning.

Figure 3.11: Digital IR Sensor

Connections :-

1. Connect the OUT, V_{CC} and ground pins of the IR sensor to pins D2, 5V and GND pins of Arduino Board.

Code :-

```
void setup(){
    pinMode(2, INPUT);
    Serial.begin(9600);
}
void loop(){
    ir=digitalRead(2);
    if(ir==0){
    Serial.print("Near");
    }
    else{
    Serial.print("Far");
    };
}
```

Working :-

Here, as the object is near to the sensor, the IR radiation emitted by the transmitter is reflected by the object and detected by the receiver. However, this is not possible when the object is far. the sensor gives the signals accordingly.

3.10.8 Interfacing LDR Sensor with Arduino Board

Here we aim to connect an LDR (Digital) sensor with the Arduino board and study its functioning.

Connections :-

1. Connect the DO, V_{CC} and ground pins of the IR sensor to pins D2, 5V and GND pins of Arduino Board.

Code :-

```
void setup(){
    pinMode(2, INPUT);
    Serial.begin(9600);
}
void loop(){
    ldr=digitalRead(2);
```

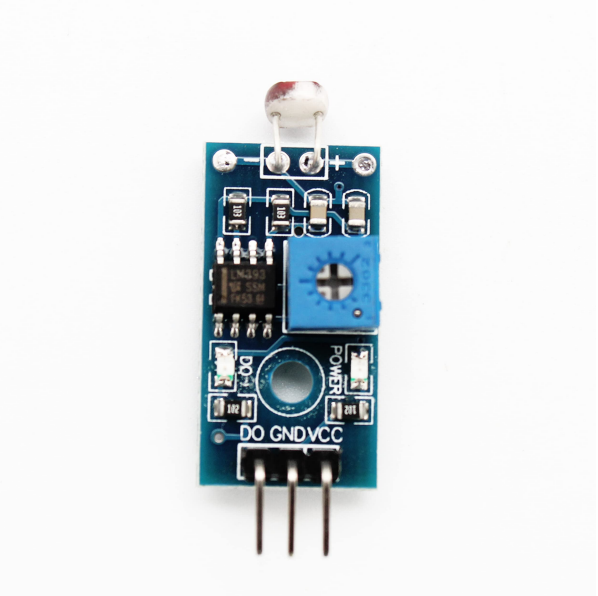

Figure 3.12: Digital LDR Sensor

```
    if(ir==0){
    Serial.print("Light present");
    }
    else{
    Serial.print("It is dark");
    };
}
```

Working :-

Here, as the light falling on the LDR reduces its resistance, hence it gives a low signal; else, if it is dark, then it gives high signal.

3.10.9 Creating a system using a potentiometer, servo motor, an DC motor, a LDR, an IR sensor and an Arduino Board

This system integrates several sensors. Whenever darkness is detected by the LDR, the servo motor is set to some angular position as decided by the position of the wiper of the potentiometer. Also,

if a large distance is detected by the IR sensor, then the DC motor starts to rotate.

Connections :-

1. Connnect the orange wire of servo motor to pin D9 of Arduino Board.
2. Connect wiper of the potentiometer to pin A0 of the Arduino Board.
3. connect the DO pin of LDR to pin D10 of Arduino Board.
4. Connect the pins IN1 and IN2 of L298N motor driver to pins D5 and D6 of the Arduino Board.
5. Connect the DC motor to OUT1 and OUT2 pins of L298N.
6. Connect the OUT pin of IR Sensor to pin D2 of Arduino Board.
7. Connect the V_{CC} and ground pins of various pins to 5V and GND pins of the Arduino Board.

Note : the connections or the architecture of this task can be changed/customized.

Code :-

```
#include<Servo.h>
Servo s1 //creating an object of servo motor from its library

void setup(){
    s1.attach(9);       //for servo motor
    pinMode(10, INPUT); //for LDR sensor
    pinMode(2, INPUT);  //for IR sensor
    pinMode(5, OUTPUT); //for DC motor
    pinMode(6, OUTPUT); //for DC motor
    pinMode(A0, INPUT); //for potentiometer
}

void loop(){
    int pot=analogRead(A0);
    int resis=map(pot, 0, 1023, 0, 10000); //ADC to resistance
    int ang=map(resis, 0, 10000, 1, 124); //converting resistance to angle
```

```
    int ldr, ir;
    ldr=digitalRead(10);
    ir=digitalRead(2);
    if(ldr==1){         //when dark
        s1.write(ang);
    };
    if(ir==1){         //when far
        digitalWrite(5, 1);
        digitalWrite(6, 0);
    };
}
```

Working :-

As the wiper of potentiometer is moved, the value of angle, which will be stored in 'ang' variable changes. However, only when the LDR sensor detects darkness, the servo motor is moved to this angular position. Also, only when the IR sensor detects large distance, then the DC motor is moved.

3.10.10 Interfacing Ultrasonic Sensor with Arduino Board and obtaining Distance from it

Here we aim to measure distance using an ultrasonic sensor.

Figure 3.13: Ultrasonic Sensor

Connections :-

1. Connect Trig and Echo pins of the Ultrasonic Sensor to pin 7 of Arduino Board.
2. Connect V_{CC} and GND pins of Ultrasonic Sensor to 5V and GND pins of Arduino Board.

Code :-

Note :- This code is also available in File →Examples →06.Sensors →Ping

```
// this constant won't change. It's the pin number of the sensor's output:
const int pingPin = 7;

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
}

void loop() {
  // establish variables for duration of the ping, and the distance result
  // in inches and centimeters:
  long duration, inches, cm;

  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // The same pin is used to read the signal from the PING))): a HIGH pulse
  // whose duration is the time (in microseconds) from the sending of the ping
  // to the reception of its echo off of an object.
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);
```

```
  // convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}

long microsecondsToInches(long microseconds) {
  // According to Parallax's datasheet for the PING))), there are 73.746
  // microseconds per inch (i.e. sound travels at 1130 feet per second).
  // This gives the distance travelled by the ping, outbound and return,
  // so we divide by 2 to get the distance of the obstacle.
  // See: https://www.parallax.com/package/ping-ultrasonic-distance-sensor-dow
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the object we
  // take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Working :-

The Transmitter emits ultrasonic waves; they are reflected from the objects in front of it and received by the Receiver. The time passed since release of wave till its capture is noted, and computing on this time we get distance between the ultrasonic sensor and the object.

3.10.11 Interfacing MPU6050 sensor with Arduino Board and obtaining the linear and angular acceleration along x, y and z axes from it

Here we aim to measure linear and angular acceleration along x, y and z axes using a MPU6050 sensor.

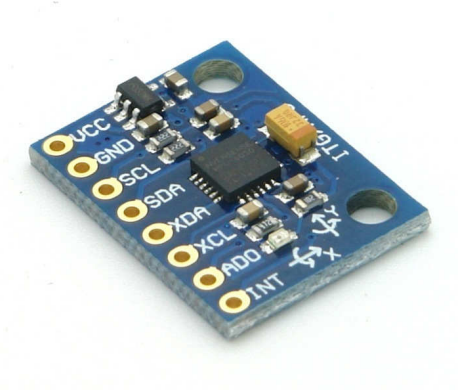

Figure 3.14: MPU6050

Note :- Download 'Adafruit MPU6050' library.

Connections :-

1. Connect SDA and SCL pins of MPU6050 to pins A4 and A5 of Arduino Board.
2. Connect V_{CC} and GND pins of MPU6050 to 5V and GND pins of Arduino Board.

Code :-

Note :- This code is also available in File →Examples →Adafruit MPU6050 →motion_detection

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
```

```
void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  //setupt motion detection
  mpu.setHighPassFilter(MPU6050_HIGHPASS_0_63_HZ);
  mpu.setMotionDetectionThreshold(1);
  mpu.setMotionDetectionDuration(20);
  mpu.setInterruptPinLatch(true); // Keep it latched.  Will turn off when rein
  mpu.setInterruptPinPolarity(true);
  mpu.setMotionInterrupt(true);

  Serial.println("");
  delay(100);
}

void loop() {

  if(mpu.getMotionInterruptStatus()) {
    /* Get new sensor events with the readings */
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    /* Print out the values */
    Serial.print("AccelX:");
    Serial.print(a.acceleration.x);
```

```
    Serial.print(",");
    Serial.print("AccelY:");
    Serial.print(a.acceleration.y);
    Serial.print(",");
    Serial.print("AccelZ:");
    Serial.print(a.acceleration.z);
    Serial.print(", ");
    Serial.print("GyroX:");
    Serial.print(g.gyro.x);
    Serial.print(",");
    Serial.print("GyroY:");
    Serial.print(g.gyro.y);
    Serial.print(",");
    Serial.print("GyroZ:");
    Serial.print(g.gyro.z);
    Serial.println("");
  }

  delay(10);
}
```

Working :-

The proof mass in MPU6050 moves with motion applied to the whole MPU6050 sensor and hence, motion is detected by means of MEMS system on it.

3.10.12 Interfacing LCD Display with I2C module and with Arduino Board

Here we aim to connect a LCD display with Arduino Board via I2C module and study its functioning.
Note :- Download 'LiquidCrystal I2C' library.

Connections :-

1. Connect the I2C module to the LCD display as shown in Fig. 3.15.
2. Connect pins SCL and SDA of I2C Module to pins A4 and A5 of Arduino Board.

3. Connect Pins V_{CC} and GND to 5V and GND pins of Arduino Board.

Code :-

1. For finding the address of the I2C Device -

```
// I2C address finding
#include <Wire.h>

void setup()
{
//Initializing wire
Wire.begin();
//Initializing seraial monitor at the baudrate of 9600
Serial.begin(9600);
}

void loop()
{
byte err, addr;
//Declaring variable to detect and count no. of I2C device found
int devices = 0;

// For loop to try multiple combinations of Address
for (addr = 1; addr < 127; addr++)
{
Wire.beginTransmission(addr);
err = Wire.endTransmission();

if (!err)
{
Serial.print("Address 0x");
if (addr < 16)
{
Serial.print("0");
}
Serial.println(addr, HEX);
devices++;
}
```

```
else if (err == 4)
{
Serial.print("Error at address 0x");
if (addr < 16)
{
Serial.print("0");
}
Serial.println(addr, HEX);
}
}

//Exception, when there is no I2C device found
if (!devices)
{
Serial.println("Please connect your I2C device");
}

//Waiting for 2 seconds
delay(2000);
}
```

2. For displaying text on LCD Display -

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x20, 16, 2); // Format -> (Address,Width,Height )

void setup()
{
// initialize the lcd
lcd.init();
// Turn on the Backlight
lcd.backlight();
}

void loop()
{
// Clear the display buffer
lcd.clear();
```

```
// Set cursor (Column, Row)
lcd.setCursor(0, 0);
// print "Hello" at (0, 0)
lcd.print("Hello");
// Set cursor (Column, Row)
lcd.setCursor(0,1);
// print "World" at (0, 1)
lcd.print("World");

delay(100);
}
```

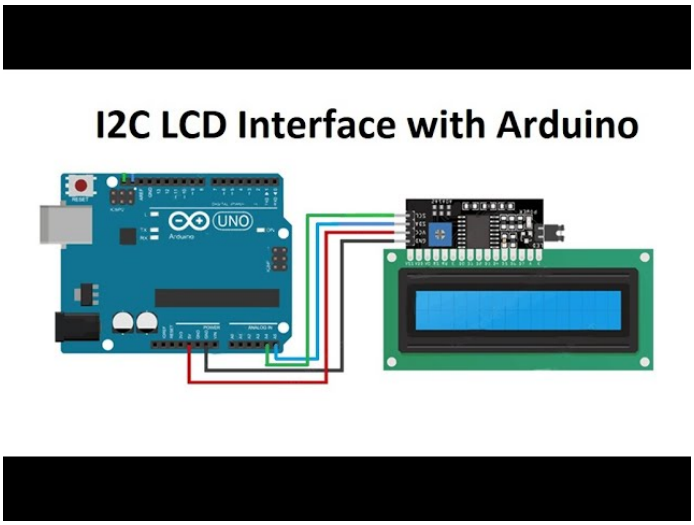

Figure 3.15: Interfacing LCD Display (16*2) with I2C module and arduino Board

Working :-

We attach an I2C module to communicate with the LCD display by I2C Communication Protocol. Data regarding text is sent to the display via I2C protocol. The address of the I2C module can also be changed by giving power supply to A2, A1 and A0 pins.

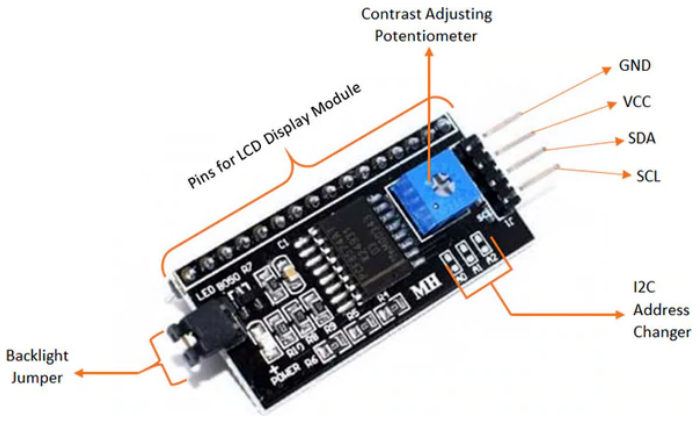

Figure 3.16: I2C module

3.10.13 Interfacing 28BYJ-48 Stepper Motor with Arduino Board

Here we aim to connect a 28BYJ-48 Stepper motor with the Arduino board and study its functioning.

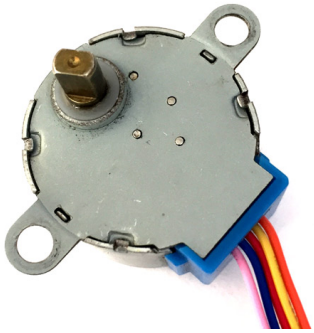

Figure 3.17: Stepper Motor

Connections :-

1. Connect the one coil of the stepper motor to OUT1 and OUT2 pins of the motor driver L298N.
2. Connect the second coil of the stepper motor to OUT3 and OUT4 pins of the motor driver.

3. Connect the respective 5V, 12V and GND pins of the motor driver to respective power supply and Arduino power pins.
4. Connect IN1 and IN2 to pins 8 and 9; and IN3 and IN4 to pins 10 and 11, of the Arduino Board.

Code :-

Note :- This code is also available in File →Examples →Stepper →stepper_oneRevolution

```
/*
 Stepper Motor Control - one revolution

 This program drives a unipolar or bipolar stepper motor.
 The motor is attached to digital pins 8 - 11 of the Arduino.

 The motor should revolve one revolution in one direction, then
 one revolution in the other direction.


 Created 11 Mar. 2007
 Modified 30 Nov. 2009
 by Tom Igoe

 */

#include <Stepper.h>

const int stepsPerRevolution = 200;  // change this to fit the number of steps
// for your motor

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
  // set the speed at 60 rpm:
  myStepper.setSpeed(60);
  // initialize the serial port:
  Serial.begin(9600);
```

```
}

void loop() {
  // step one revolution  in one direction:
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);

  // step one revolution in the other direction:
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
```

Working :-

Due to the proper sequence of magnetization and demagnetization of the coils of the stepper motor, the axle rotates in steps and achieves rotational motion.

3.10.14 Interfacing SSD1306 128*64 pixels OLED Display with Arduino Board

Here we aim to connect an OLED display - SSD1306 128*64 pixels with the Arduino board and study its functioning.

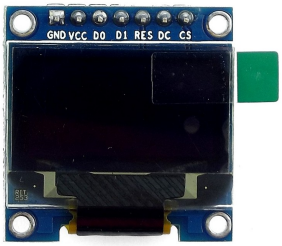

Figure 3.18: SSD1306 OLED Display

Note :- Download 'Adafruit SSD1306' library.

Connections :-

1. Install the latest version of 'Adafruit SSD1306' library.
2. Connect the DO/SCK/SCL/CLK pin of SSD1306 to pin 10 of the Arduino Board.
3. Connect the D1/SDA/MOSI pin of SSD1306 to pin 9 of the Arduino Board.
4. Connect the RES/RST/RESET of SSD1306 pin to pin 13 of the Arduino Board.
5. Connect the DC/A0 pin of SSD1306 to pin 11 of the Arduino Board.
6. Connect the CS/Chip Select pin of SSD1306 to pin 12 of the Arduino Board.

Code :-

Note :- This code is also available in File →Examples →Adafruit SSD1306 →ssd1306_128x64_spi

```
/*************************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306 drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x64 pixel display using SPI to communicate
 4 or 5 pins are required to interface.

 Adafruit invests time and resources providing this open
 source code, please support Adafruit and open-source
 hardware by purchasing products from Adafruit!

 Written by Limor Fried/Ladyada for Adafruit Industries,
 with contributions from the open source community.
 BSD license, check license.txt for more information
```

```
 All text above, and the splash screen below must be
 included in any redistribution.
 *********************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using software SPI (default case)
#define OLED_MOSI   9
#define OLED_CLK   10
#define OLED_DC    11
#define OLED_CS    12
#define OLED_RESET 13
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
  OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

/* Comment out above, uncomment this block to use hardware SPI
#define OLED_DC     6
#define OLED_CS     7
#define OLED_RESET  8
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
  &SPI, OLED_DC, OLED_RESET, OLED_CS);
*/

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ 0b00000000, 0b11000000,
  0b00000001, 0b11000000,
  0b00000001, 0b11000000,
  0b00000011, 0b11100000,
  0b11110011, 0b11100000,
```

```
  0b11111110, 0b11111000,
  0b01111110, 0b11111111,
  0b00110011, 0b10011111,
  0b00011111, 0b11111100,
  0b00001101, 0b01110000,
  0b00011011, 0b10100000,
  0b00111111, 0b11100000,
  0b00111111, 0b11110000,
  0b01111100, 0b11110000,
  0b01110000, 0b01110000,
  0b00000000, 0b00110000 };

void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }


  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
```

```
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...

  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)

  testdrawchar();      // Draw characters of the default font

  testdrawstyles();    // Draw 'stylized' characters

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
```

```
}

void testdrawline() {
  int16_t i;

  display.clearDisplay(); // Clear display buffer

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn line
    delay(1);
  }
  for(i=0; i<display.height(); i+=4) {
    display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHIT
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHIT
    display.display();
    delay(1);
```

```
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_WHIT
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHIT
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i, SSD1306_
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testfillrect(void) {
  display.clearDisplay();
```

```
  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2, SSD1306_
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testdrawcircle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SSD1306_WHITE
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillcircle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
    // The INVERSE color is used so circles alternate white/black
    display.fillCircle(display.width() / 2, display.height() / 2, i, SSD1306_I
    display.display(); // Update screen with each newly-drawn circle
    delay(1);
  }

  delay(2000);
}

void testdrawroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
```

```
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    // The INVERSE color is used so round-rects alternate white/black
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawtriangle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfilltriangle(void) {
```

```
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }

  display.display();
  delay(2000);
}

void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
```

```
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);

  display.display();
  delay(2000);
}

void testscrolltext(void) {
  display.clearDisplay();

  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
```

```
}

void testdrawbitmap(void) {
  display.clearDisplay();

  display.drawBitmap(
    (display.width()  - LOGO_WIDTH ) / 2,
    (display.height() - LOGO_HEIGHT) / 2,
    logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
  display.display();
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];

  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: "));
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: "));
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: "));
    Serial.println(icons[f][DELTAY], DEC);
  }

  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer

    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h, SSD1306
```

```
    }

    display.display(); // Show the display buffer on the screen
    delay(200);        // Pause for 1/10 second

    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

Working :-

'OLED' stands for Organic Light Emitting Diode. SSD1306 Display is connected with Arduino Board via SPI Interface. Various shapes and patterns are tested on the display.