



Workshop on “Gesture Controlled Robot”

Presented By

Mohammed Abdul Kader

Assistant Professor, Dept. of EEE, IIUC

Email: kader05cueta@gmail.com **Website:** kader05cueta.wordpress.com

Organized by
IEEE SB & WIE, IIUC

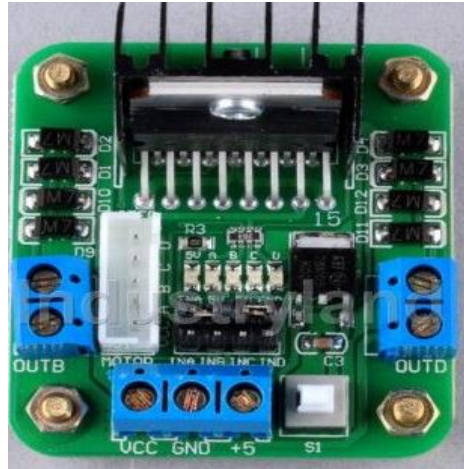
Objectives

- ☐ To know about microcontroller.
- ☐ To familiarize with I/O pins of microcontroller.
- ☐ To learn the method of pairing two BT devices.
- ☐ To familiarize with 3-axis accelerometer (ADXL 345).
- ☐ To know about I2C Protocol and data acquisition from 3-axis accelerometer.
- ☐ To control the speed and direction of DC motor by microcontroller.
- ☐ **Controlling Robot by gesture.**

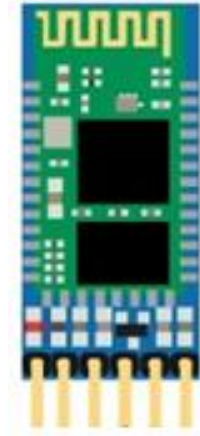
Required Components



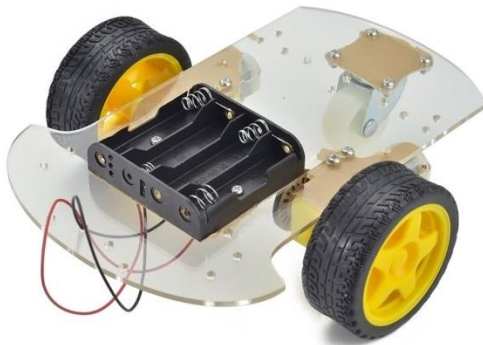
DC Motor



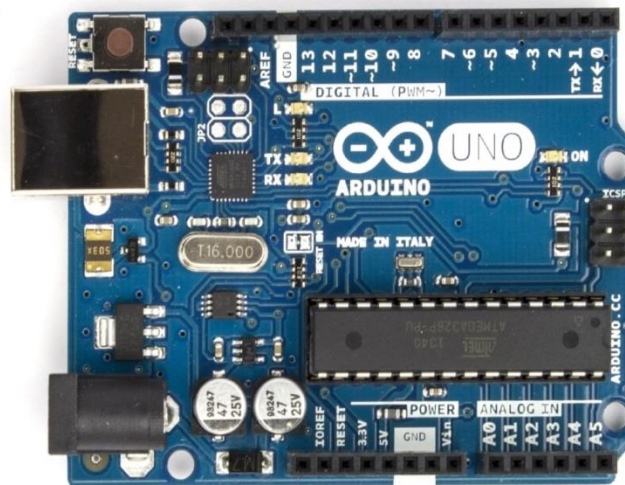
Motor Driver



Bluetooth Module



Chassis

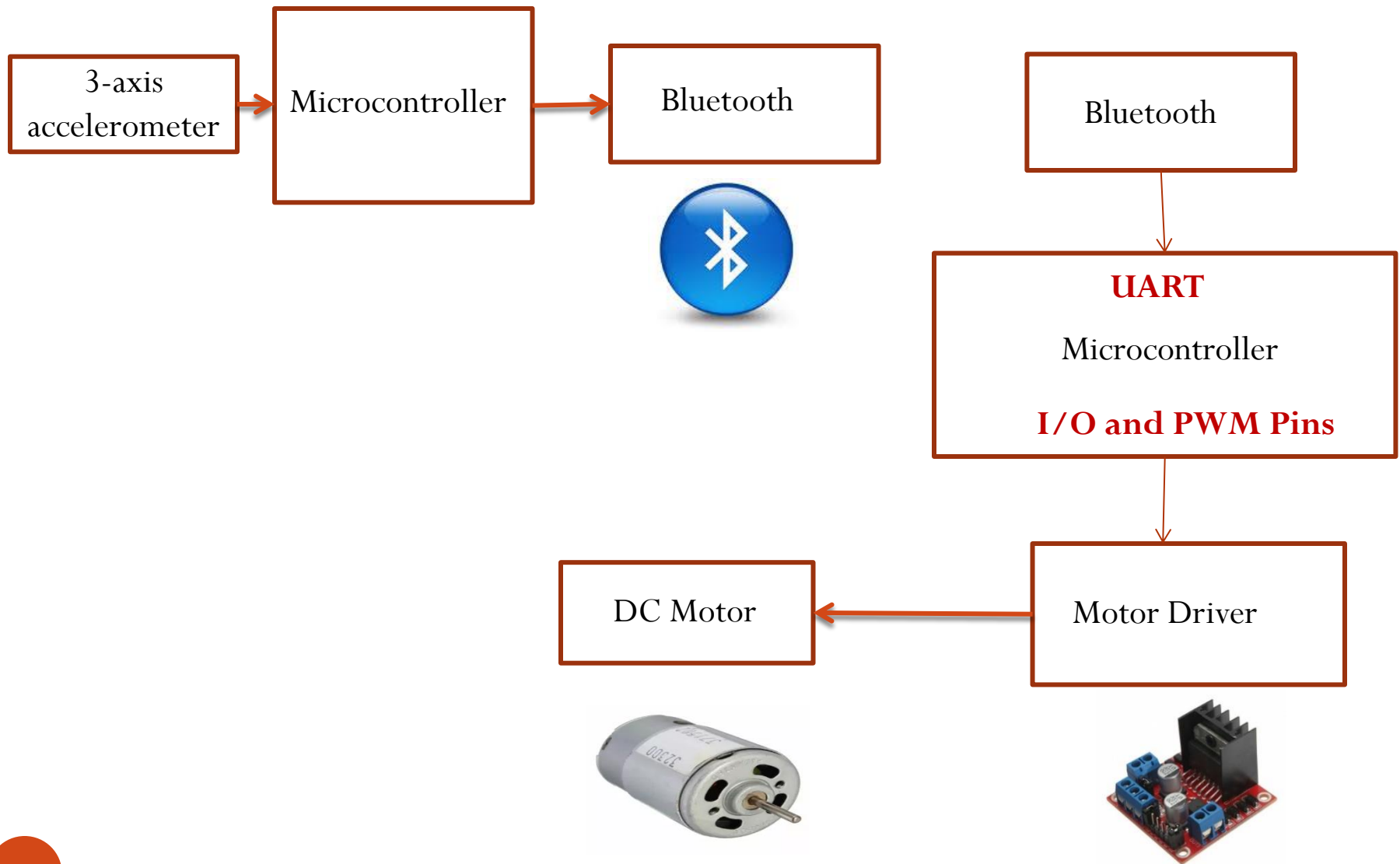


Microcontroller



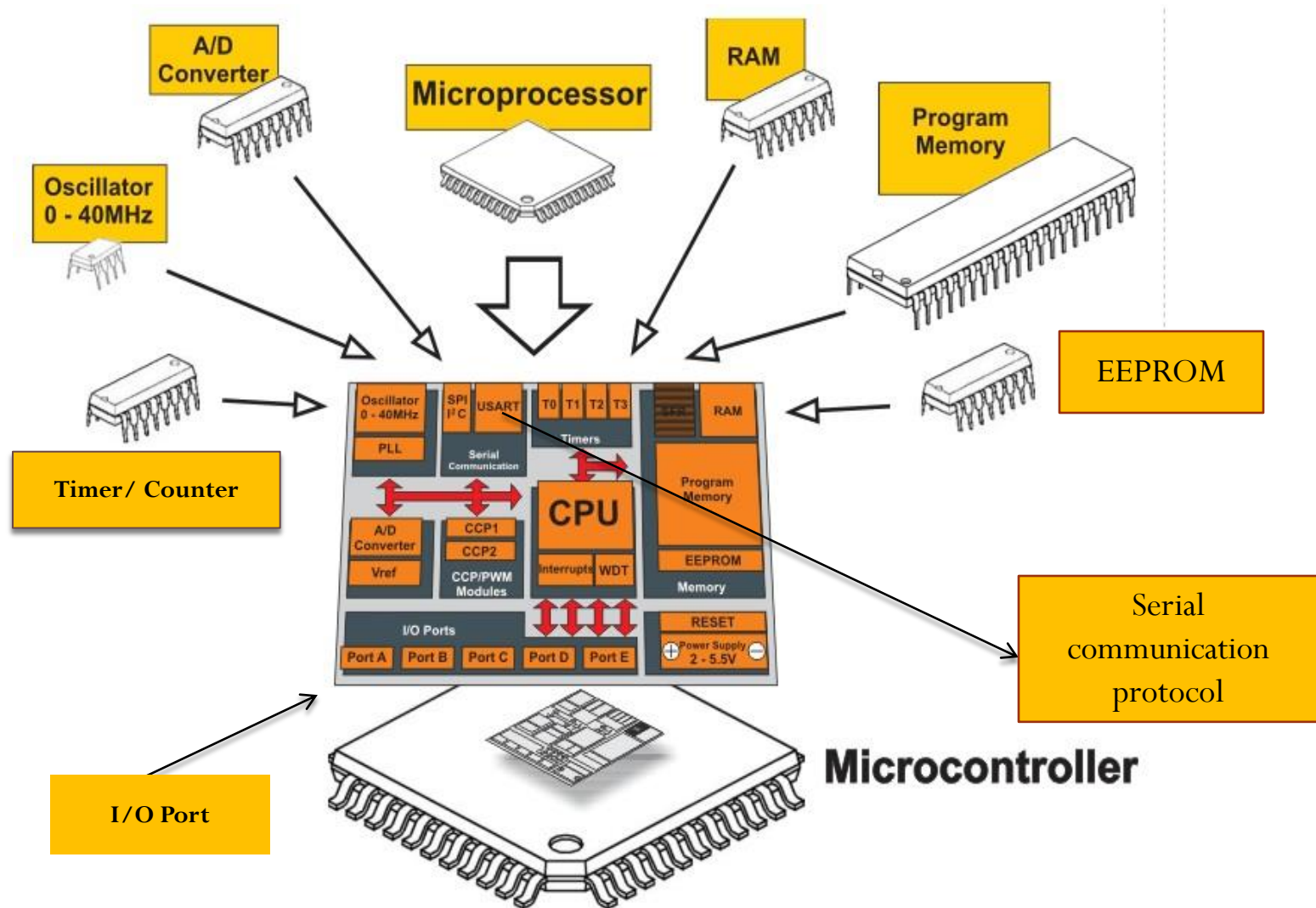
3-axis Accelerometer

Methodology: Gesture Control



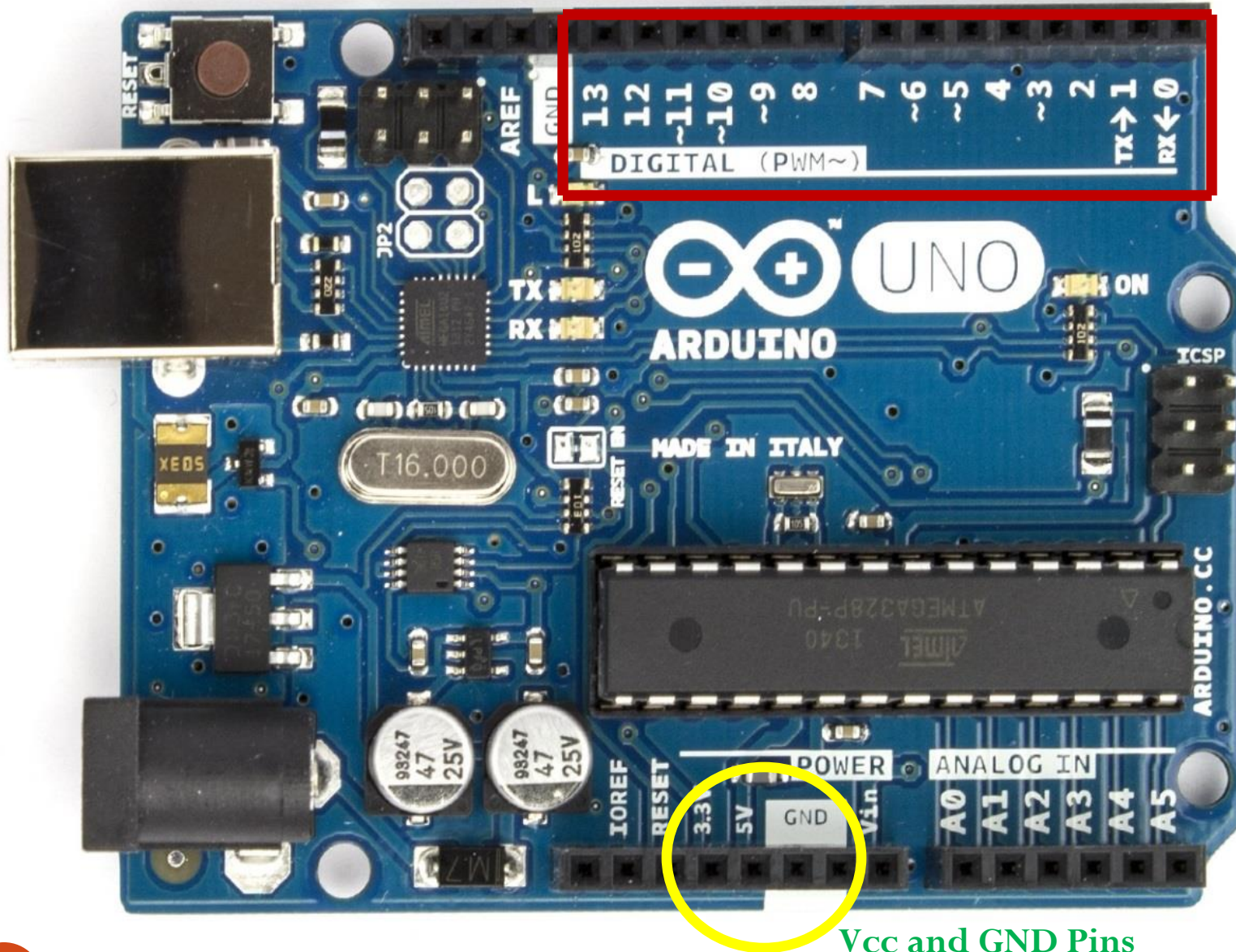
What is Microcontroller?

Micro-computer in a single chip.



Arduino UNO

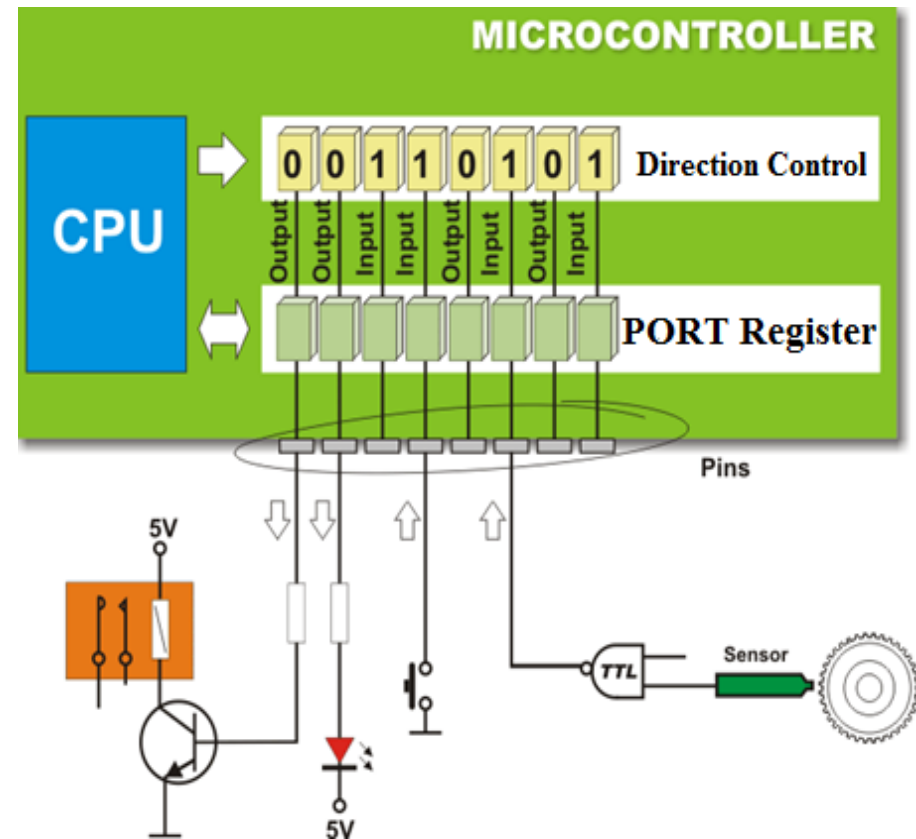
Digital I/O Pins



Vcc and GND Pins

Digital I/O Pins

- Most of the important pins of microcontroller are digital input-output pins.
- These pins are used to connect **INPUT Device** (i.e. Push Button, keypad, digital sensors etc) and **OUTPUT Device** (i.e. LED, Display, Relay, Motor etc.) with microcontroller.
- These pins can act as INPUT or OUTPUT.
- Digital Output pin means microcontroller can **make this pin HIGH or LOW state** (Write Operation).
- Digital Input pin means microcontroller can **read HIGH or LOW state from other devices** (Read Operation).



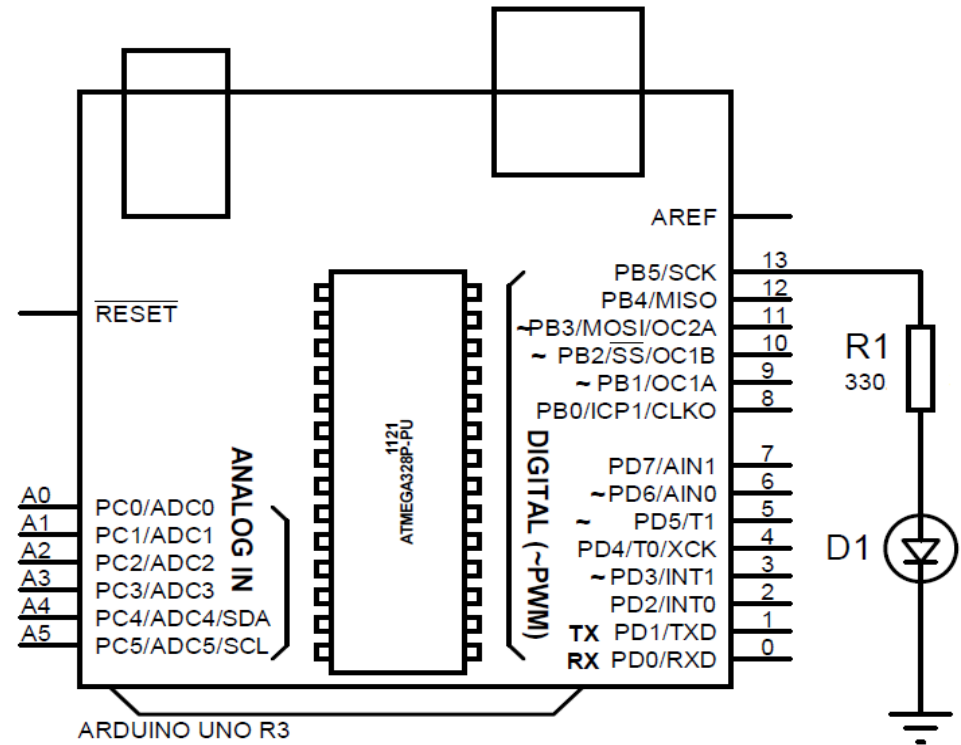
Configuring Digital I/O Pins

Configuring as OUTPUT

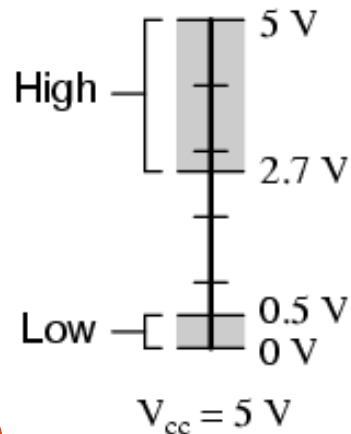
- An LED is connected with pin 13.
- The pin should be an OUTPUT pin.
- We can configure a pin as OUTPUT by “pinMode” function.

`pinMode(pin Number, OUTPUT)`

`pinMode(13, OUTPUT)`



Acceptable TTL gate output signal levels



Making a Pin HIGH or LOW

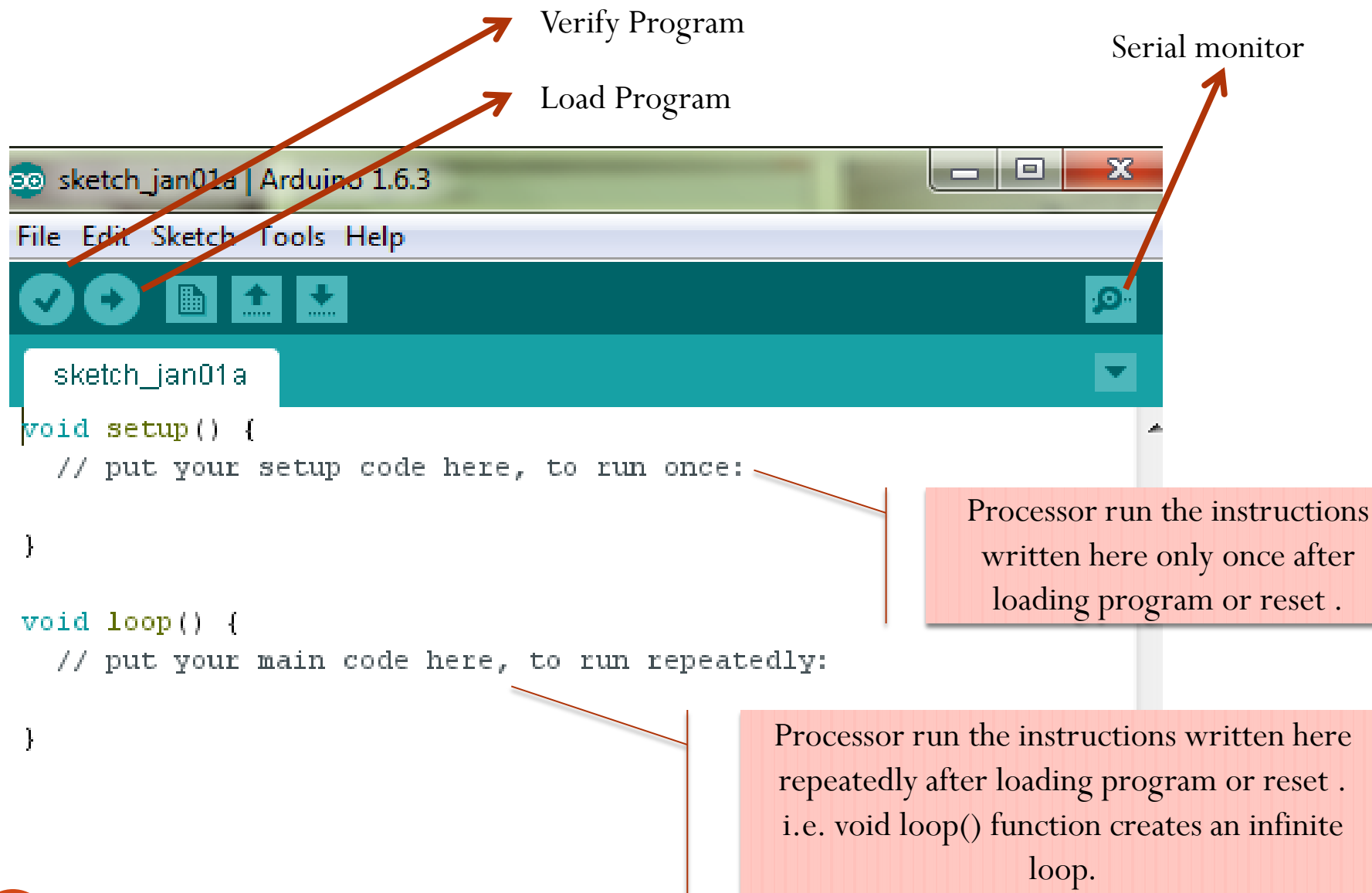
- Microcontroller can make a digital pin HIGH or LOW by digitalWrite function.

`digitalWrite(pin number, HIGH/LOW)`

`digitalWrite(13,HIGH)`

`digitalWrite(13,LOW)`

Program Structure in ARDUINO



The image shows the Arduino IDE interface with several annotations:

- Verify Program:** An arrow points from the text "Verify Program" to the checkmark icon in the toolbar.
- Load Program:** An arrow points from the text "Load Program" to the right-pointing arrow icon in the toolbar.
- Serial monitor:** An arrow points from the text "Serial monitor" to the magnifying glass icon in the toolbar.

The code editor displays the following code:

```
sketch_jan01a
void setup() {
  // put your setup code here, to run once:
}

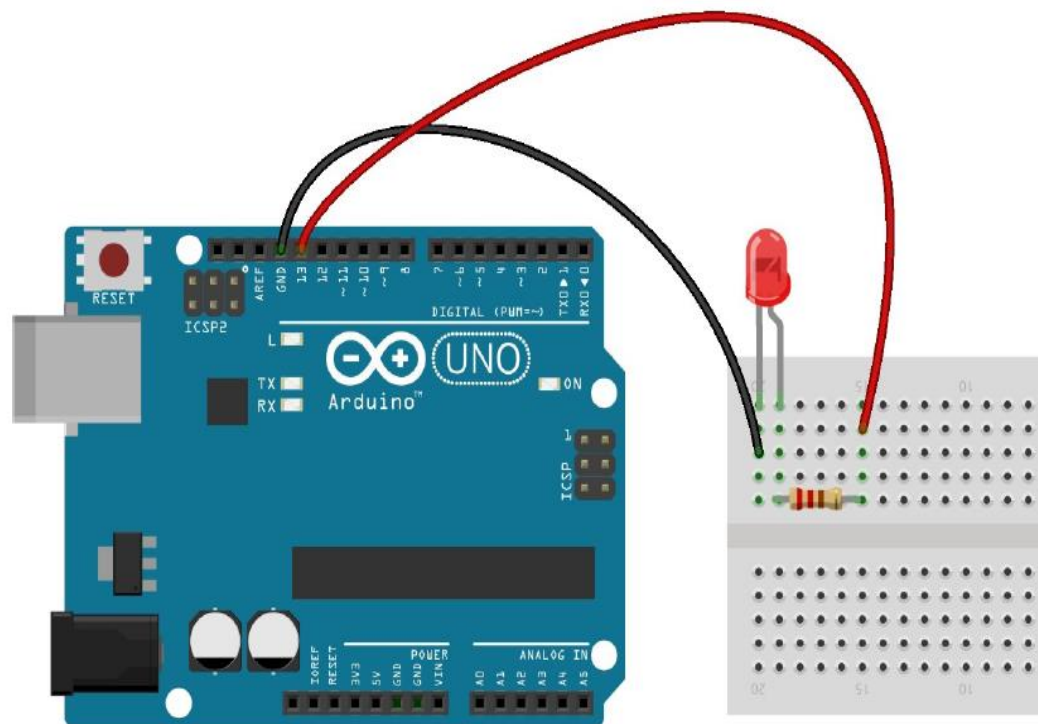
void loop() {
  // put your main code here, to run repeatedly:
}
```

Two callout boxes provide further explanation:

- Processor run the instructions written here only once after loading program or reset .** (Points to the `void setup()` function)
- Processor run the instructions written here repeatedly after loading program or reset . i.e. void loop() function creates an infinite loop.** (Points to the `void loop()` function)

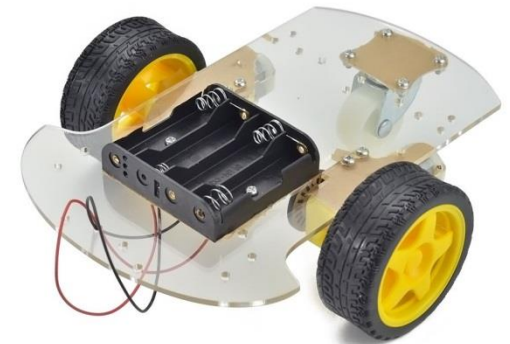
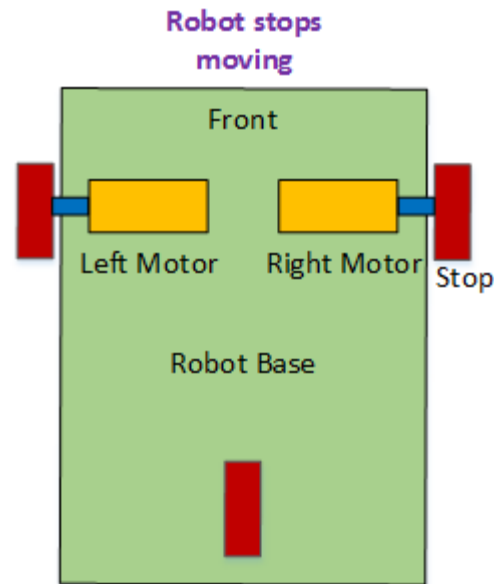
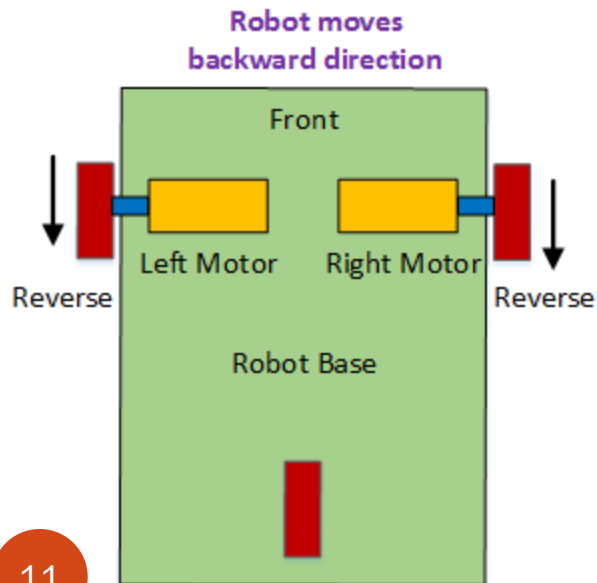
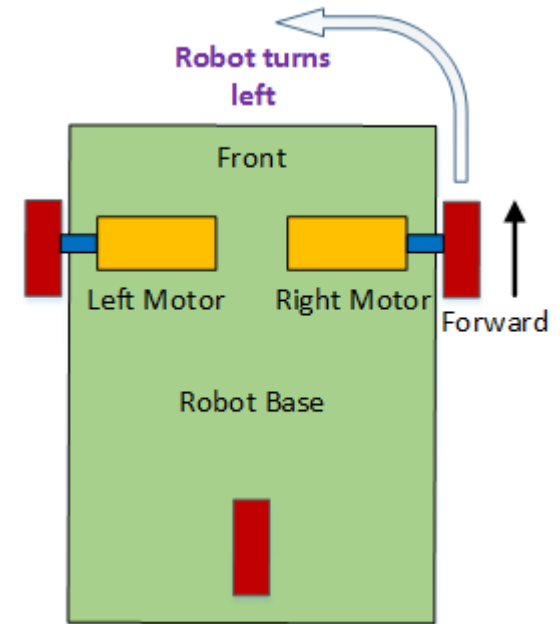
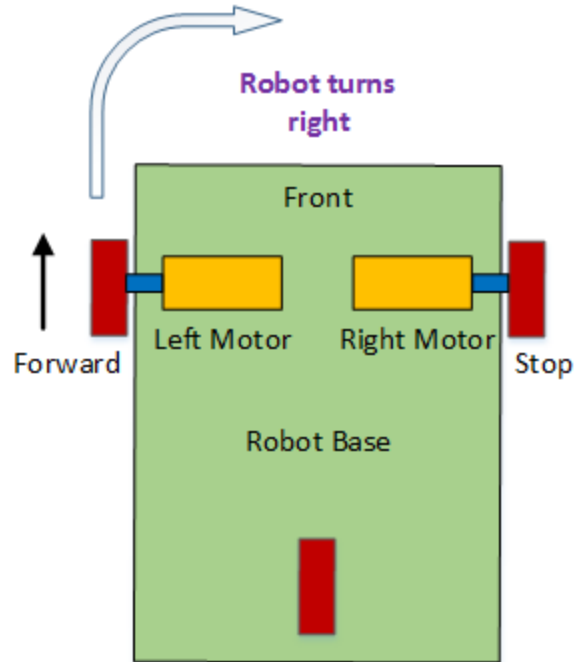
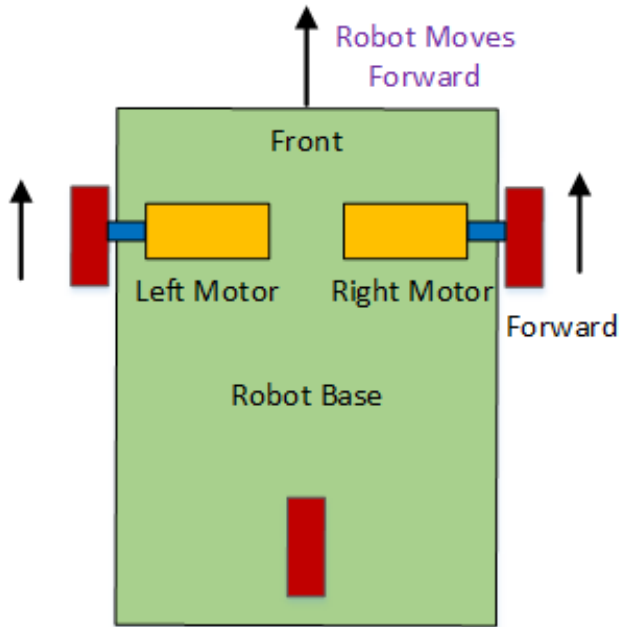
LED Blinking

```
void setup()
{
  pinMode(13,OUTPUT);
}
void loop()
{
  digitalWrite(13,HIGH);
  delay(300);
  digitalWrite(13,LOW);
  delay(300);
}
```

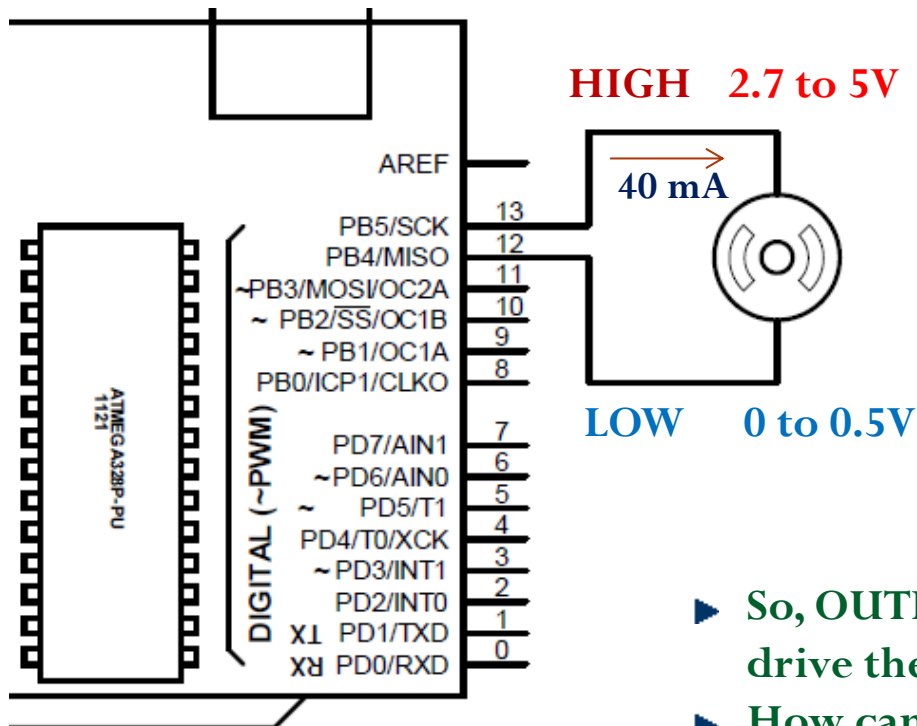
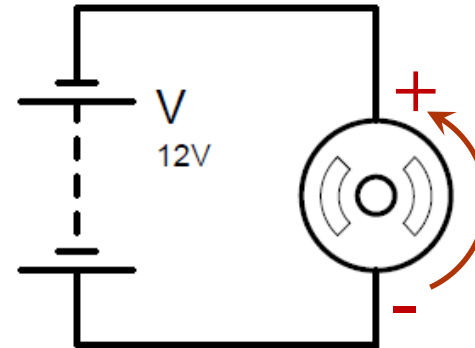
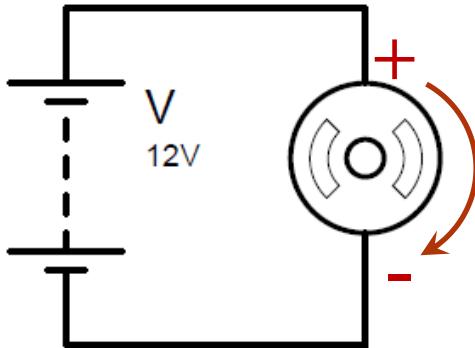


Note: To insert a time delay in the program use following function-
delay(time_in_ms);

Moving a Robot



Direction control of DC Motor

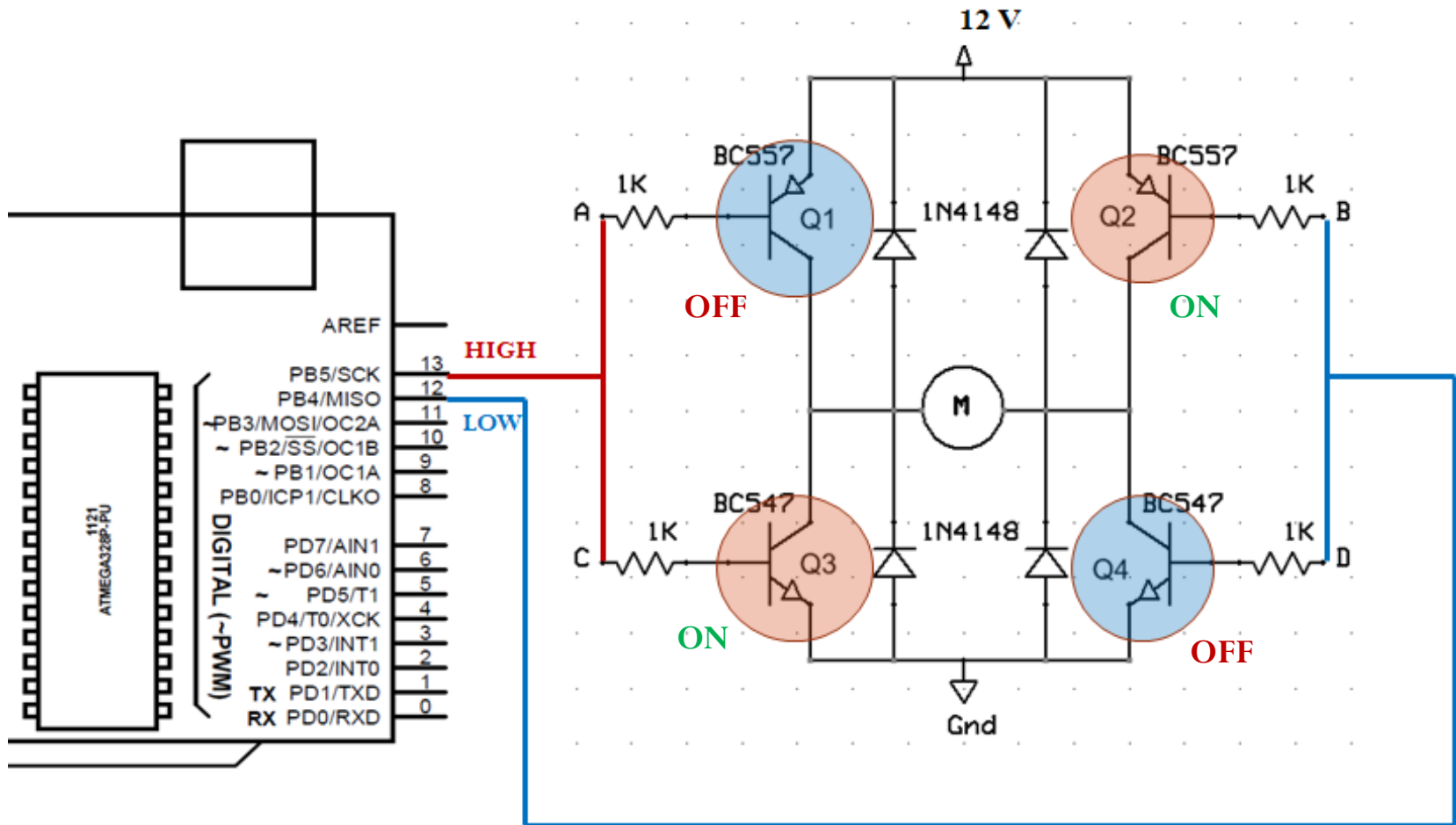


$$P = VI = 5 \times 40 \text{ mA} = 200 \text{ mW} = 0.2 \text{ W}$$

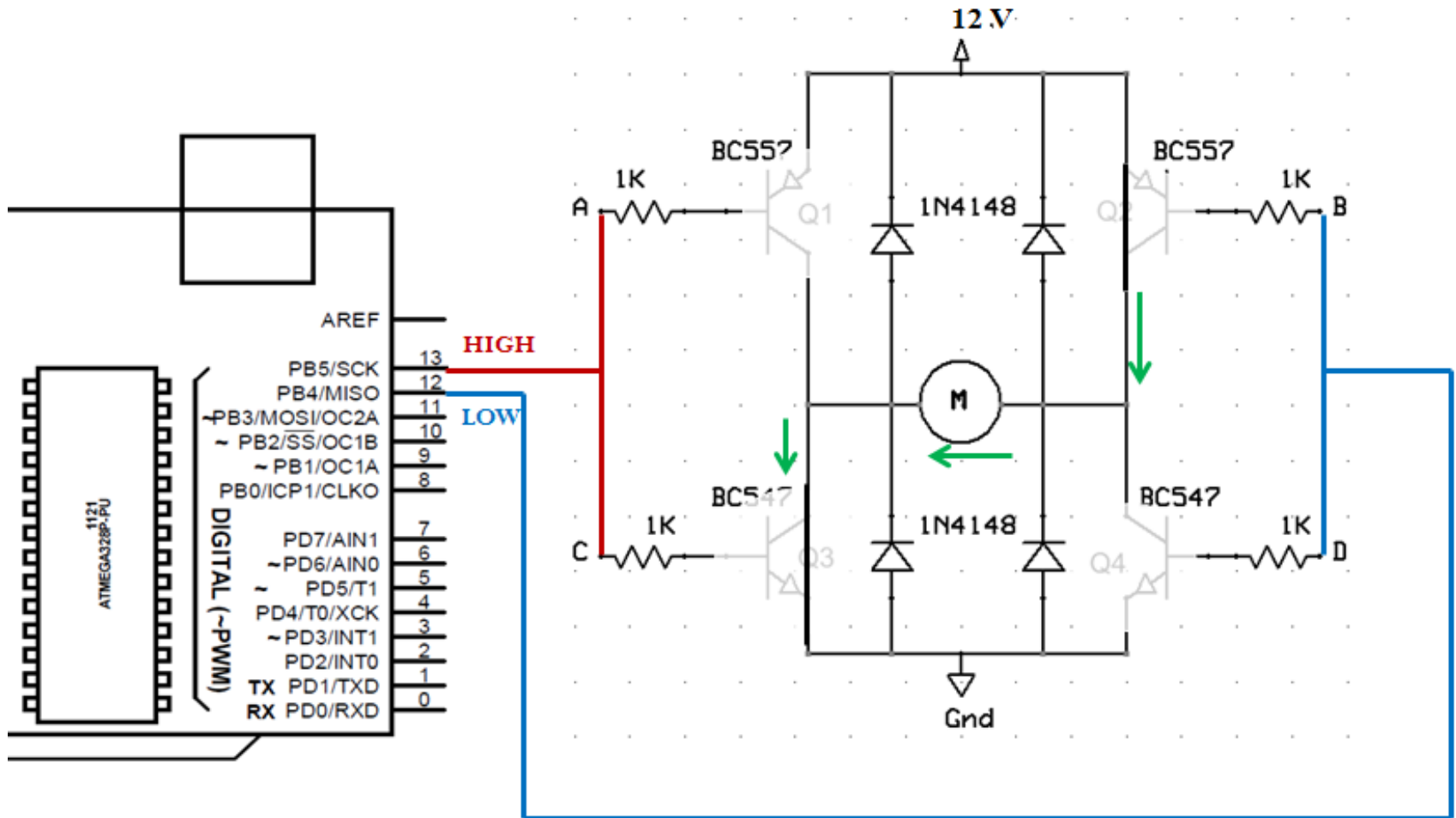
- ▶ Not enough power to drive a DC motor.
- ▶ The DC motor used in our project needs voltage of 6 to 12 V and 300-1000 mA current. Minimum of $(6 \times 300) = 1800 \text{ mW}$ or, 1.8 W power

- ▶ So, OUTPUT pins of microcontroller can not drive the DC motor required in our project.
- ▶ How can we control the direction of DC motor by microcontroller?

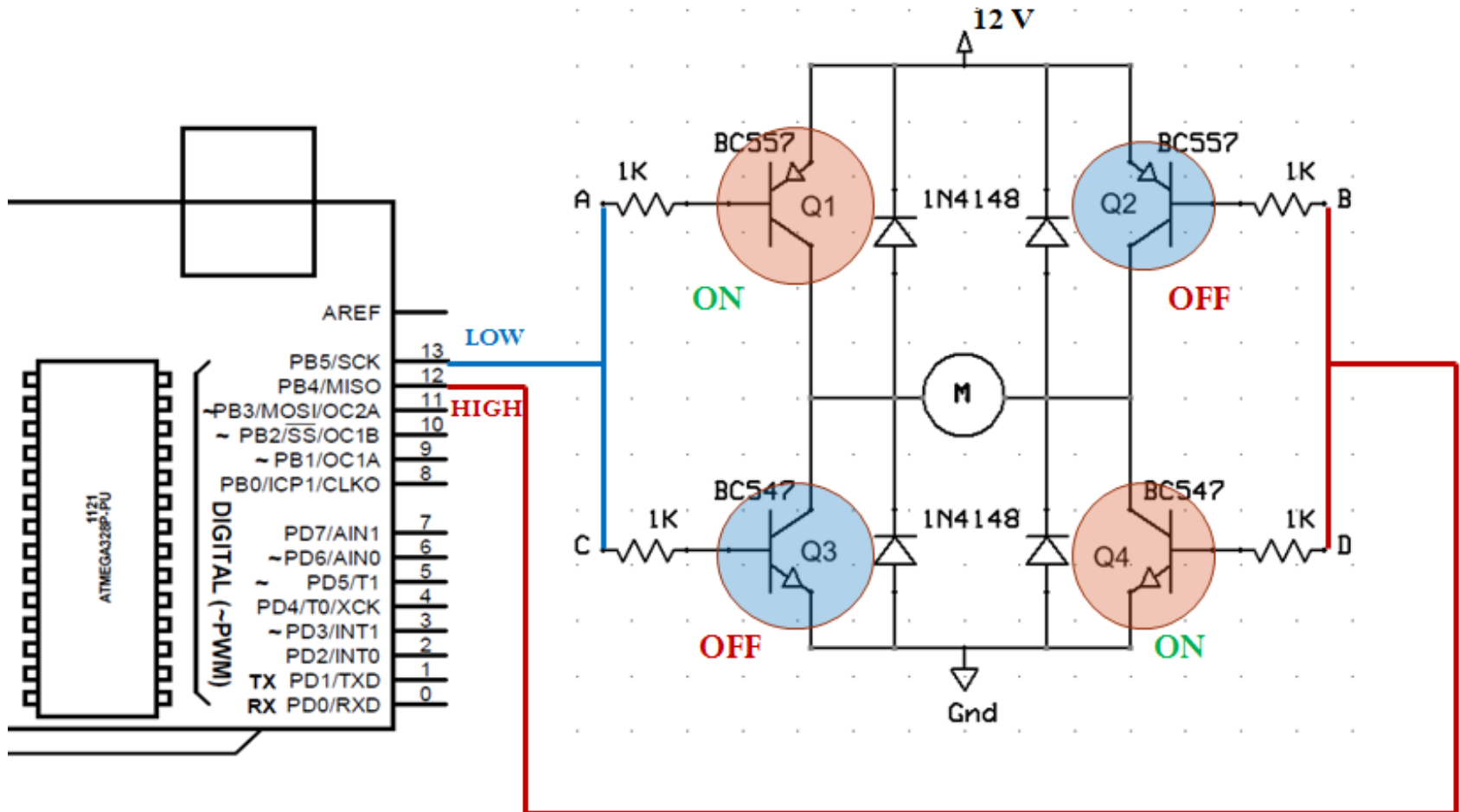
H-Bridge Motor driver



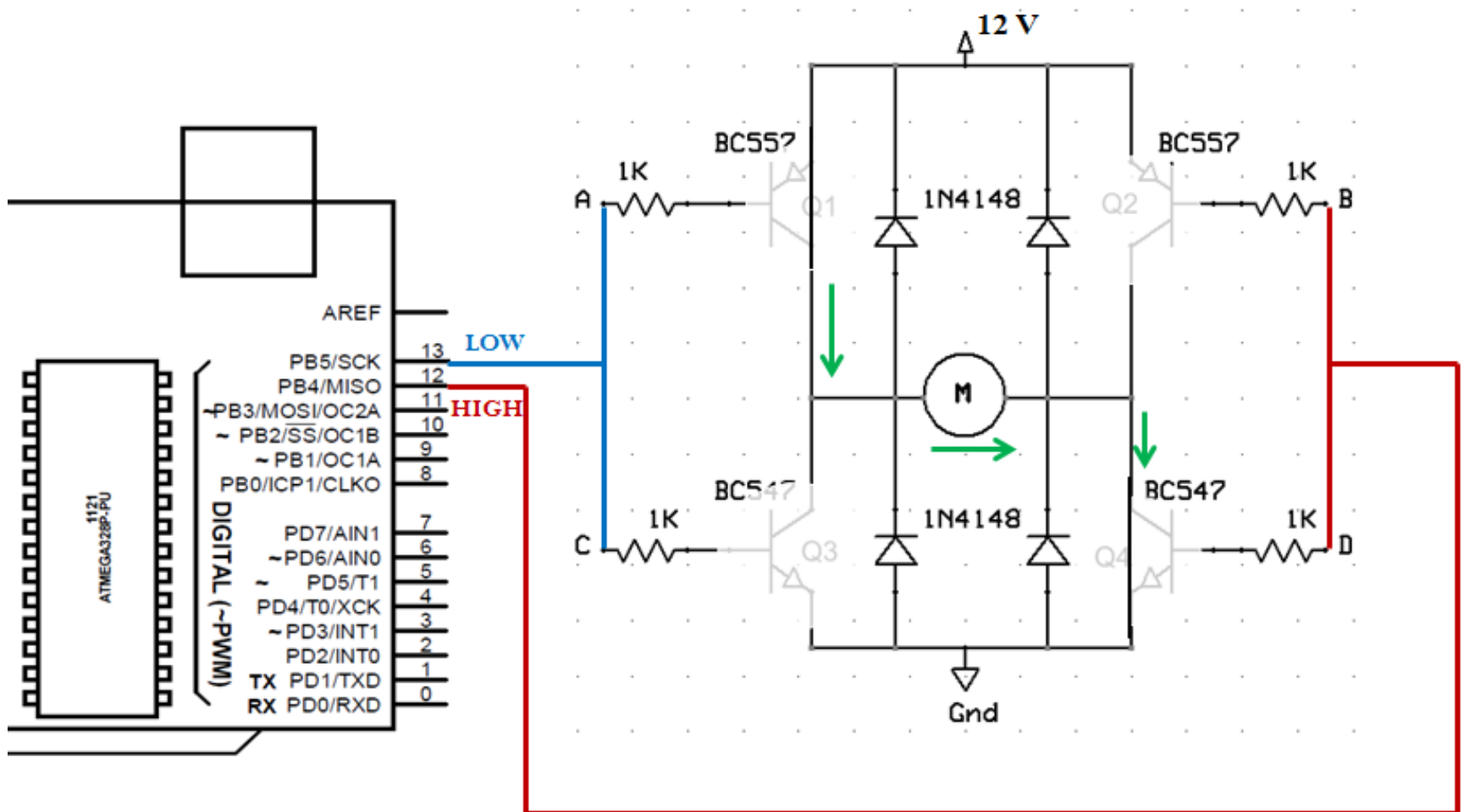
H-Bridge Motor driver



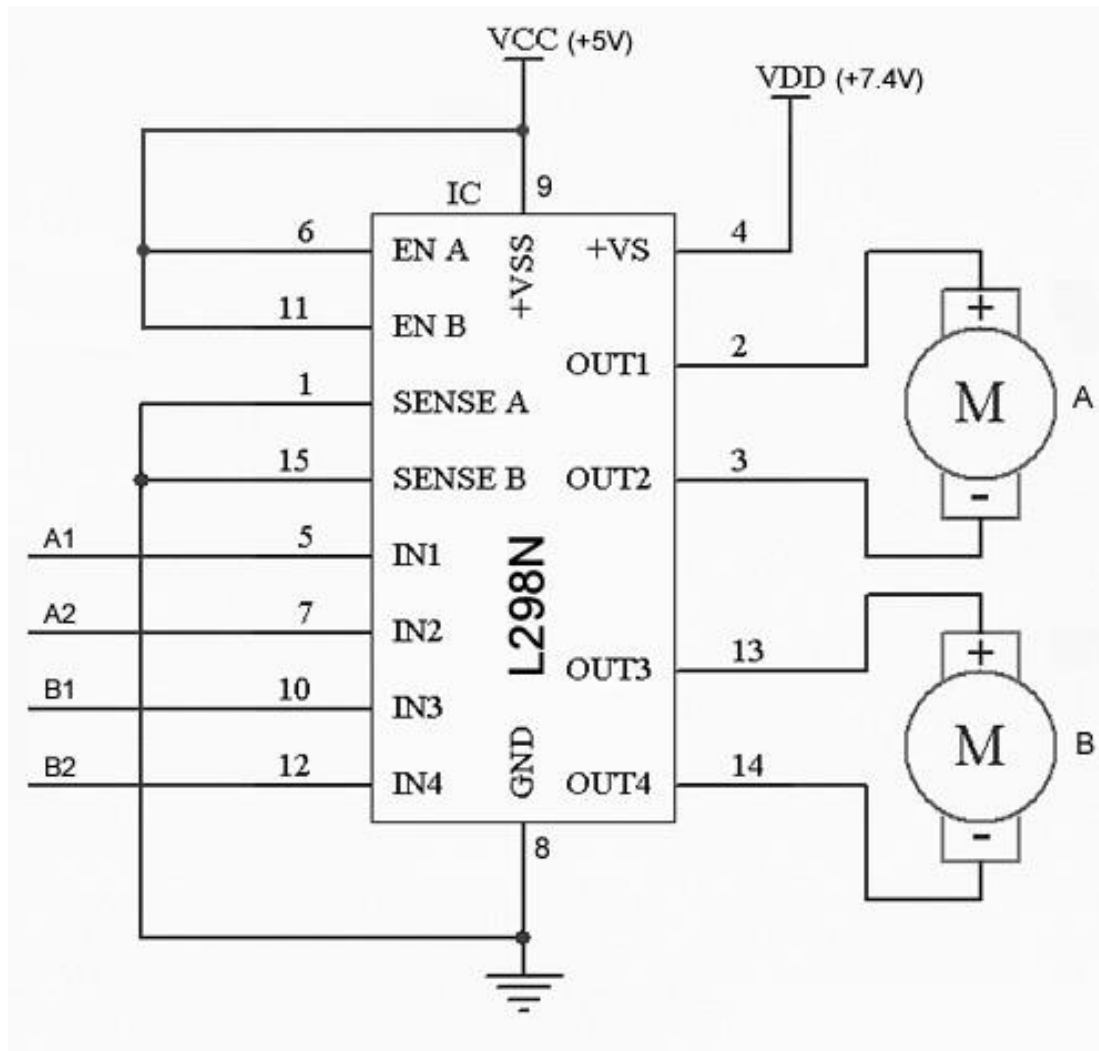
H-Bridge Motor driver



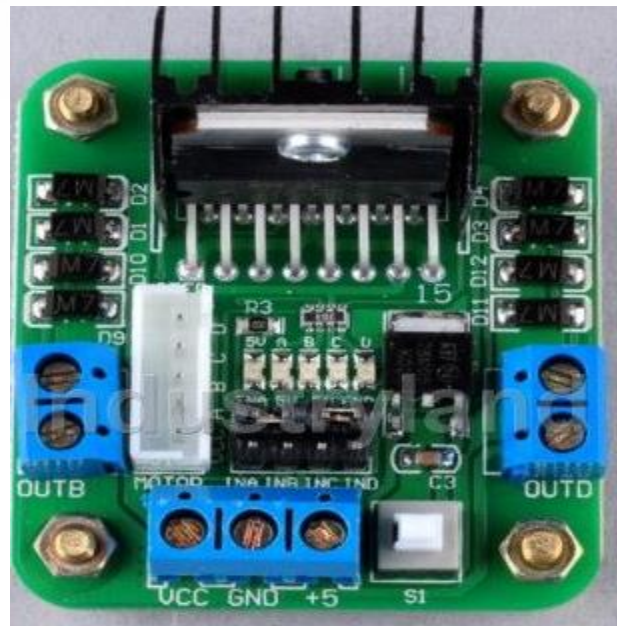
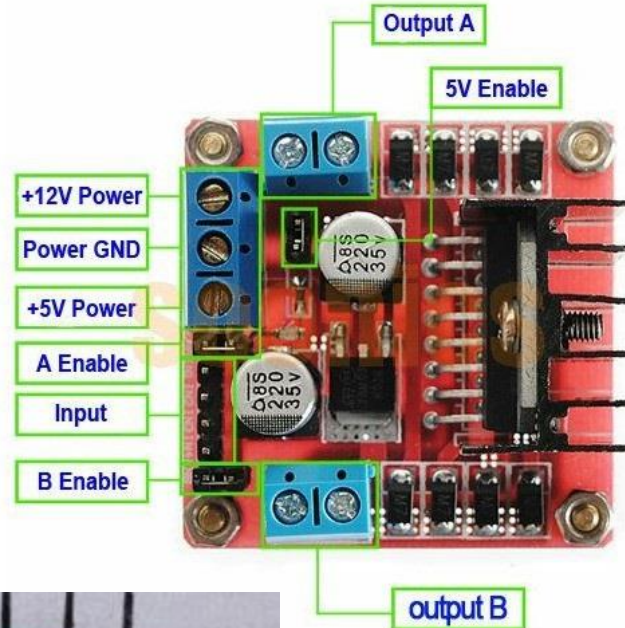
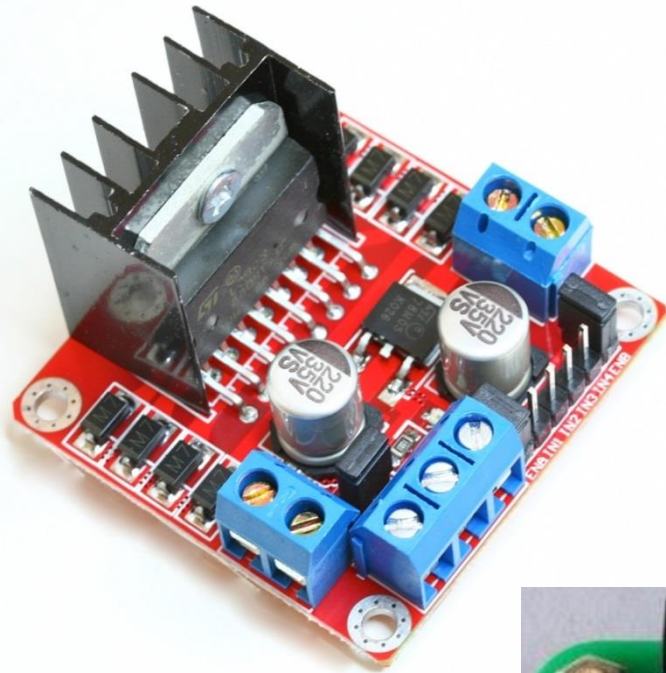
H-Bridge Motor driver



Motor Driver IC (L298N)

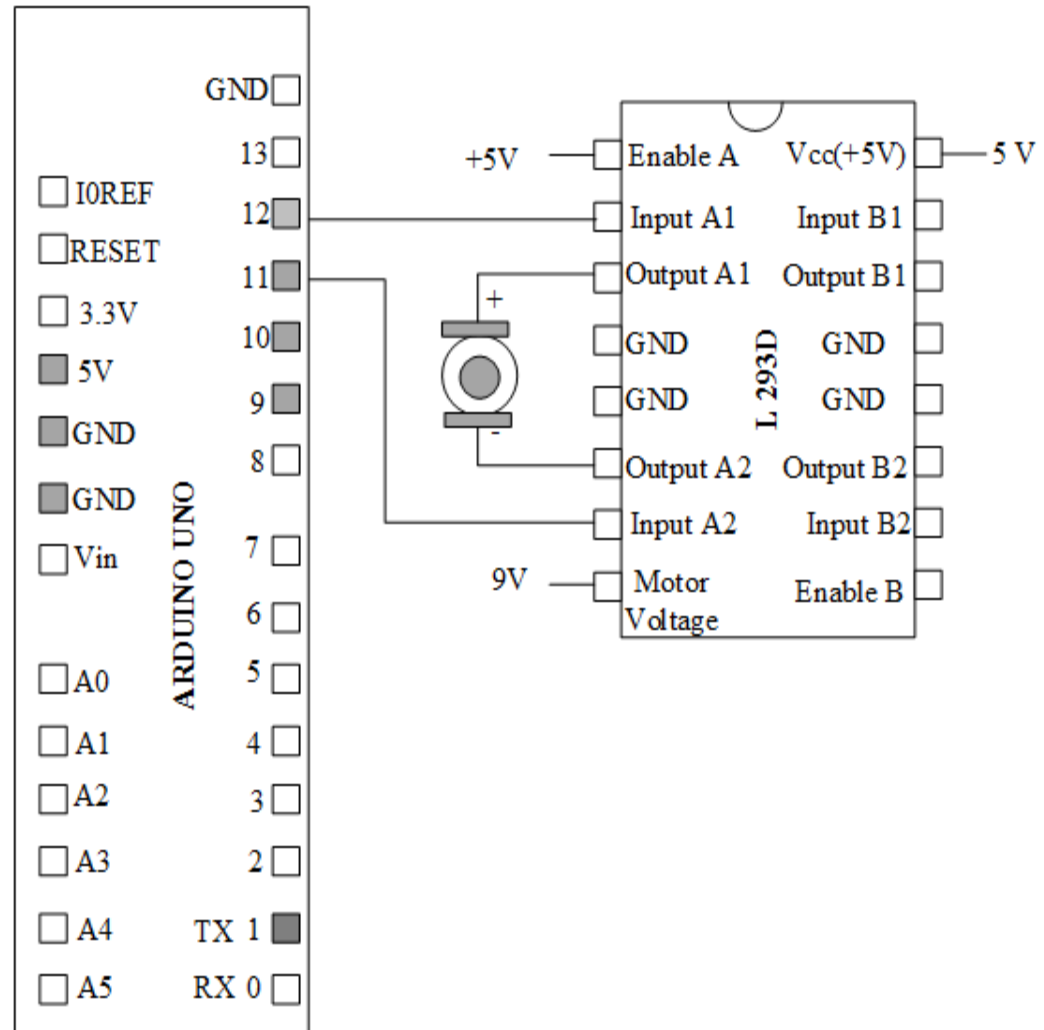


L298 Motor Driver Module



Direction control of DC motor.

```
const int MT1=12;  
const int MT2=11;  
void setup()  
{  
  pinMode(MT1, OUTPUT);  
  pinMode(MT2, OUTPUT);  
}  
void loop() {  
  digitalWrite(MT1,HIGH);  
  digitalWrite(MT2,LOW);  
  delay(5000);  
  digitalWrite(MT1,LOW);  
  digitalWrite(MT2,LOW);  
  delay(1000);  
  digitalWrite(MT1,LOW);  
  digitalWrite(MT2,HIGH);  
  delay(5000);  
  digitalWrite(MT1,LOW);  
  digitalWrite(MT2,LOW);  
  delay(1000);  
}
```



Pairing two Bluetooth Modules

Steps

Step-1: Loading program and setting serial Monitor.

Step-2: Connection of BT module with ARDUINO.

Step-3: Setting at AT mode.

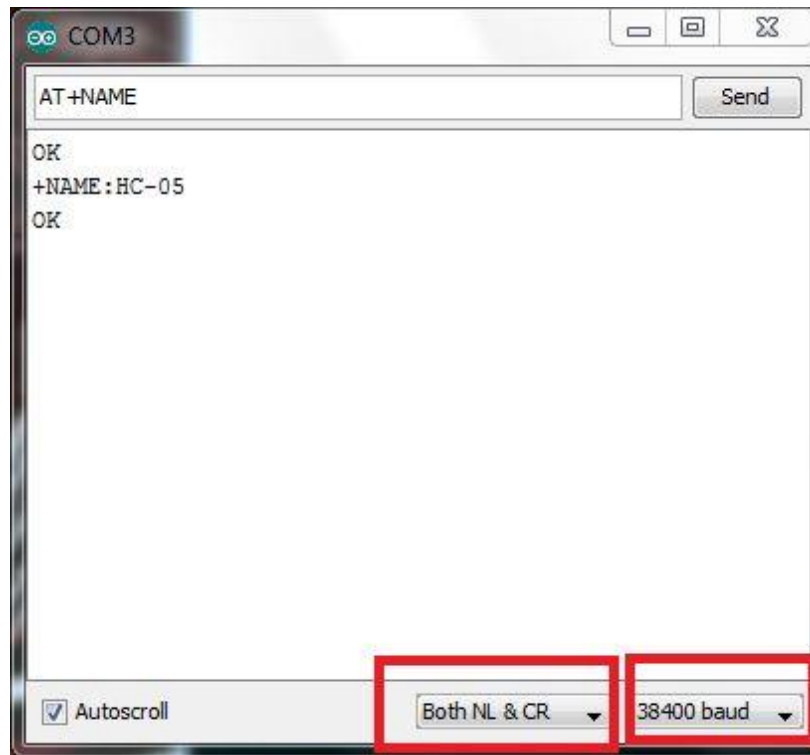
Step-4: Configuring Bluetooth module by AT command mode.



Pairing two Bluetooth Modules (Cont.)

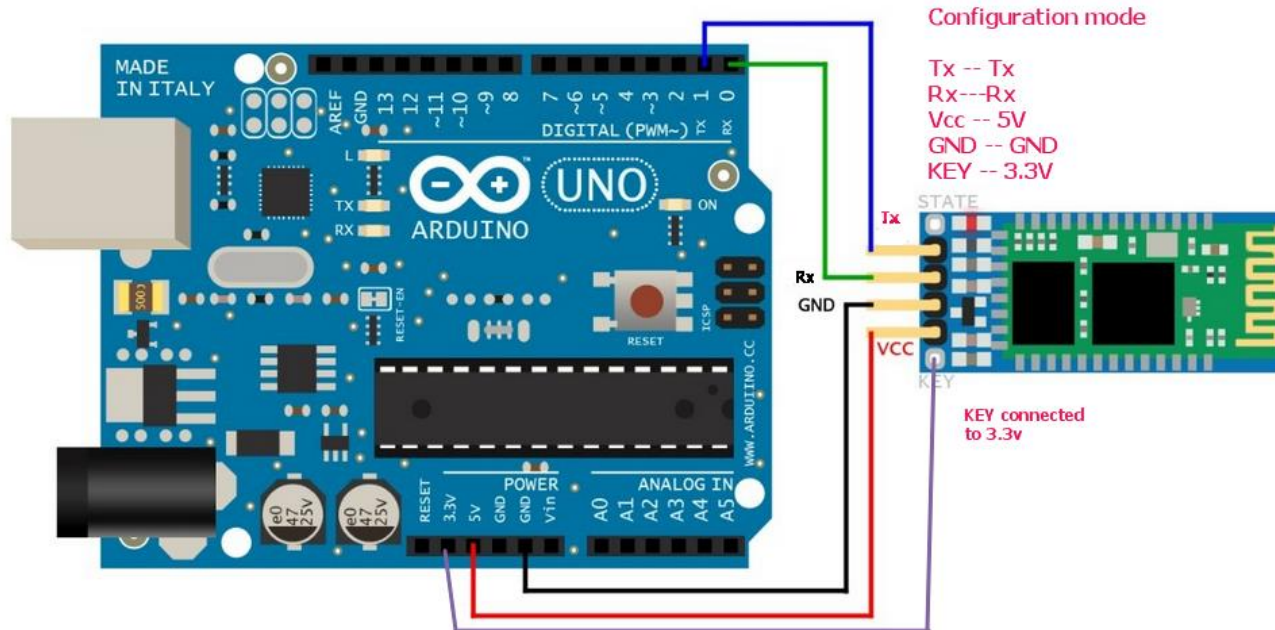
Step-1: Loading program and setting serial Monitor.

- a) Load a blank sketch to ARDUINO board.
- b) Open the serial monitor. The bluetooth module is communicating at a **baud rate of 38400 in AT command mode**. So change the baud rate to 38400 at bottom right corner of the serial monitor. Also change "no line ending " to "**both NL & CR**" found just beside the baud rate.



Pairing two Bluetooth Modules (Cont.)

Step-2: Connection of BT module with ARDUINO.



Pairing two Bluetooth Modules (Cont.)

Step-3: Setting at AT mode.

Press the reset button of BT Module [on the module without a button, connect the “Key Pin” to 3.3V on Arduino] and then connect the module with power. The module will enter into AT command mode, A blink at every 2 sec represent that the module is in AT command mode.

Step-4: Configuring Bluetooth Module by AT command

For HC05: Type "AT" (without the quotes) on the serial monitor and press enter. if "OK" appears then everything is all right and the module is ready to take command. Now you can change the name of the module, retrieve address or version or even reset to factory settings. To see the default name, type AT+NAME. The name will be prompted, by default it is HC-05 or JY_MCU or something like that. To change the name just type AT+NAME=your desired name.

Pairing two Bluetooth Modules (Cont.)

Step-4: Configuring Bluetooth Module by AT command (Cont.)

Here is an important note, if the key pin is not high, i.e. not connected to Vcc while receiving AT commands(if you released it after the module entered AT mode), it will not show the default name even after giving right command. But you can still change the name by the command mentioned above. To verify if the name has really changed, search the device from your pc/mobile. The changed name will appear. To change baud rate, type AT+UART=desired baud rate. Exit by sending AT+RESET command.

Most useful AT commands are

AT : Check the connection.

AT+NAME : See default name

AT+ADDR : see default address

AT+VERSION : See version

AT+UART : See baudrate

AT+ROLE: See role of bt module(1=master/0=slave)

AT+RESET : Reset and exit AT mode

AT+ORGL : Restore factory settings

AT+PSWD: see default password

To check status:

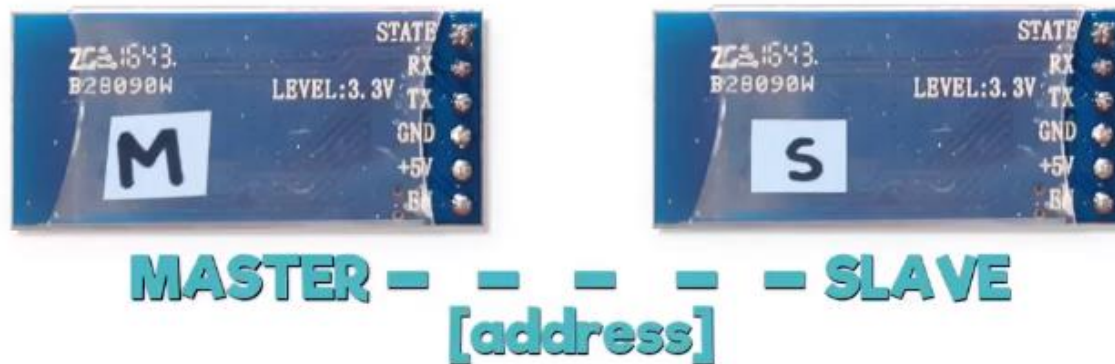
AT+COMMAND?

To set value:

AT+COMMAND= value

Pairing two Bluetooth Modules (Cont.)

Step-4: Configuring Bluetooth Module by AT command (Cont.)



Configuring Slave

AT+ROLE=0 (Slave mode)

AT+ADDR? (Find address of slave)-Copy address, change all colons (:) by comma (,)

Configuring Master

Change Role to Master: AT+ROLE=1 (Master Mode)

Change Connection Mode: AT+CMODE=0 (Connect to one device only) [N.B. AT+CMODE=1, indicates the device can be connected to any device]

Bind to slave module: AT+BIND= slave address (address of the slave copied during configuring slave)

Communication between two Bluetooth Modules: Functions

Setting baud rate:

```
Serial.begin(9600);
```

Transmitting/Sending data:

```
Serial.write(data);
```

Receiving data:

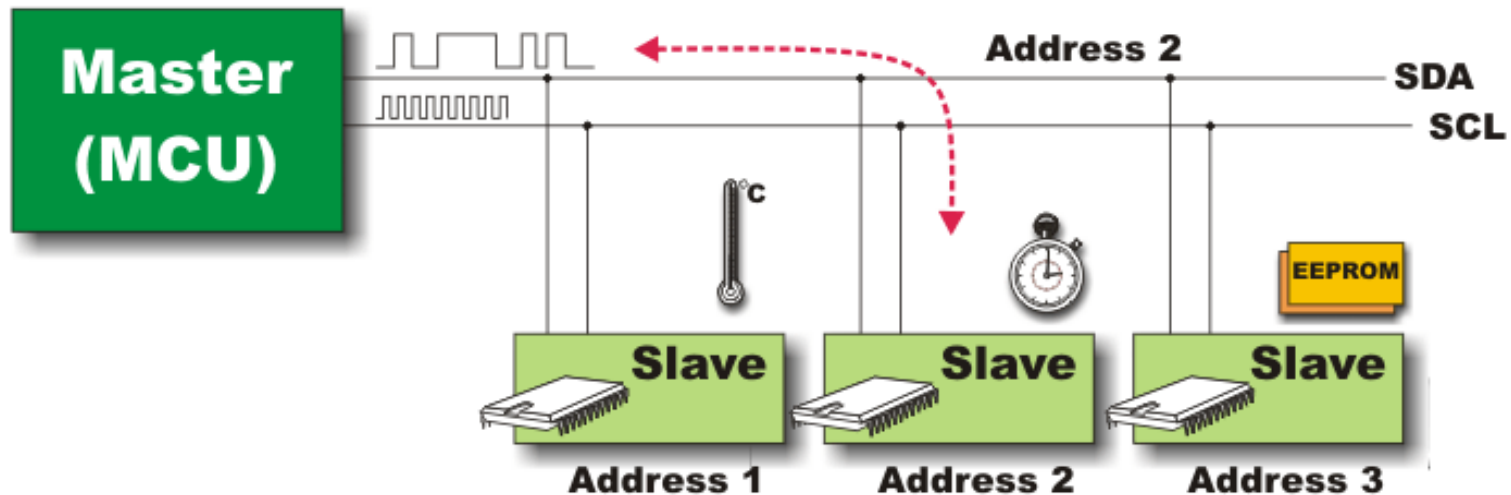
```
void serialEvent() {  
    while (Serial.available()) {  
        data = Serial.read();  
    }  
}
```

I2C (*Inter IC Bus*) Communication protocol

Inter-integrated circuit is a system for serial data exchange between the microcontrollers and specialized integrated circuits of a new generation of so called 'smart peripheral components' (memories, temperature sensors, real-time clocks etc.)

. It is used when the distance between them is short (receiver and transmitter are usually on the same printed board). Connection is established via two conductors. These are the SDA (Serial Data) and SCL (Serial Clock) pins.

By observing particular rules (protocols), this mode enables up to 122 different components to be simultaneously connected in a simple way by using only two valuable I/O pins



Functions Related to I2C Communication Protocol

Wire.begin()

Initiate the Wire library and join the I2C bus as a master.

Wire.begin(address)

Initiate the Wire library and join the I2C bus as a slave. Address is the 7-bit slave address.

Wire.requestFrom()

Used by the master to request bytes from a slave device. The bytes may then be retrieved with the `available()` and `read()` functions.

Syntax: `Wire.requestFrom(address, quantity)`

`Wire.requestFrom(address, quantity, stop)`

Parameters: address > the 7-bit address of the device to request bytes from

quantity > the number of bytes to request

stop > boolean. true will send a stop message after the request, releasing the bus.

false will continually send a restart after the request, keeping the connection active.

Return: the number of bytes returned from the slave device

Wire.beginTransmission(address)

Begin a transmission to the I2C slave device with the given address.

Parameter: address > the 7-bit address of the device to transmit to

Return: None

Wire.endTransmission()

Ends a transmission to a slave device that was begun by [beginTransmission\(\)](#)

Syntax: Wire.endTransmission()

Wire.endTransmission(stop)

Parameter: stop > boolean. true will send a stop message, releasing the bus after transmission.
false will send a restart, keeping the connection active.

Returns: byte, which indicates the status of the transmission-

0:success 1:data too long to fit in transmit buffer

2:received NACK on transmit of address 3:received NACK on transmit of data

4:other error

write()

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to beginTransmission() and endTransmission()).

Syntax: `Wire.write(value)`

`Wire.write(string)`

`Wire.write(data, length)`

Parameter: value> a value to send as a single byte ; string> a string to send as a series of bytes

data> an array of data to send as bytes; length> the number of bytes to transmit

Returns: byte> write() will return the number of bytes written, though reading that number is optional.

Wire.available()

Returns the number of bytes available for retrieval with [read\(\)](#). This should be called on a master device after a call to [requestFrom\(\)](#) or on a slave inside the [onReceive\(\)](#) handler.

Parameter: None; **Returns:** The number of bytes available for reading.

read()

Reads a byte that was transmitted from a slave device to a master after a call to [requestFrom\(\)](#)

Syntax: `Wire.read()`

Parameter: None

Returns: The next byte received

Wire.setClock()

This function modifies the clock frequency for I2C communication. I2C slave devices have no minimum working clock frequency, however 100KHz is usually the baseline.

Syntax: `Wire.setClock(clockFrequency)`

Parameter: `clockFrequency`: the value (in Hertz) of desired communication clock. Accepted values are 100000 (standard mode) and 400000 (fast mode). Some processors also support 10000 (low speed mode), 1000000 (fast mode plus) and 3400000 (high speed mode). Please refer to the specific processor documentation to make sure the desired mode is supported.

Returns: None

3-axis Accelerometers (ADXL-345)



I2C Protocol

Resolution (13 bit)

Measurement Range: -16g to +16g

x value: 10 bit (2 byte) signed number

y value: 10 bit (2 byte) signed number

z value: 10 bit (2 byte) signed number

Total 6 byte data

- ✓ The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to $\pm 16g$.
- ✓ Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

Data acquisition from a 3-axis Accelerometers (ADXL-345).

```
#include <Wire.h>
#define DEVICE (0x53) // Device address as specified in data sheet
byte data[6];
char POWER_CTL = 0x2D; //Power Control Register
char DATA_FORMAT = 0x31;
char DATA_X0 = 0x32; //X-Axis Data 0
void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
  Serial.begin(9600); // start serial for output. Make sure you set your Serial Monitor to the same!
  //Put the ADXL345 into +/- 4G range by writing the value 0x01 to the DATA_FORMAT register.
  writeTo(DATA_FORMAT, 0x01);
  //Put the ADXL345 into Measurement Mode by writing 0x08 to the POWER_CTL register.
  writeTo(POWER_CTL, 0x08); }

void loop()
{
  uint8_t howManyBytesToRead = 6;
  readFrom( DATA_X0, howManyBytesToRead, data);
  int x = (((int)data[1]) << 8) | data[0];
  int y = (((int)data[3]) << 8) | data[2];
  int z = (((int)data[5]) << 8) | data[4];
```

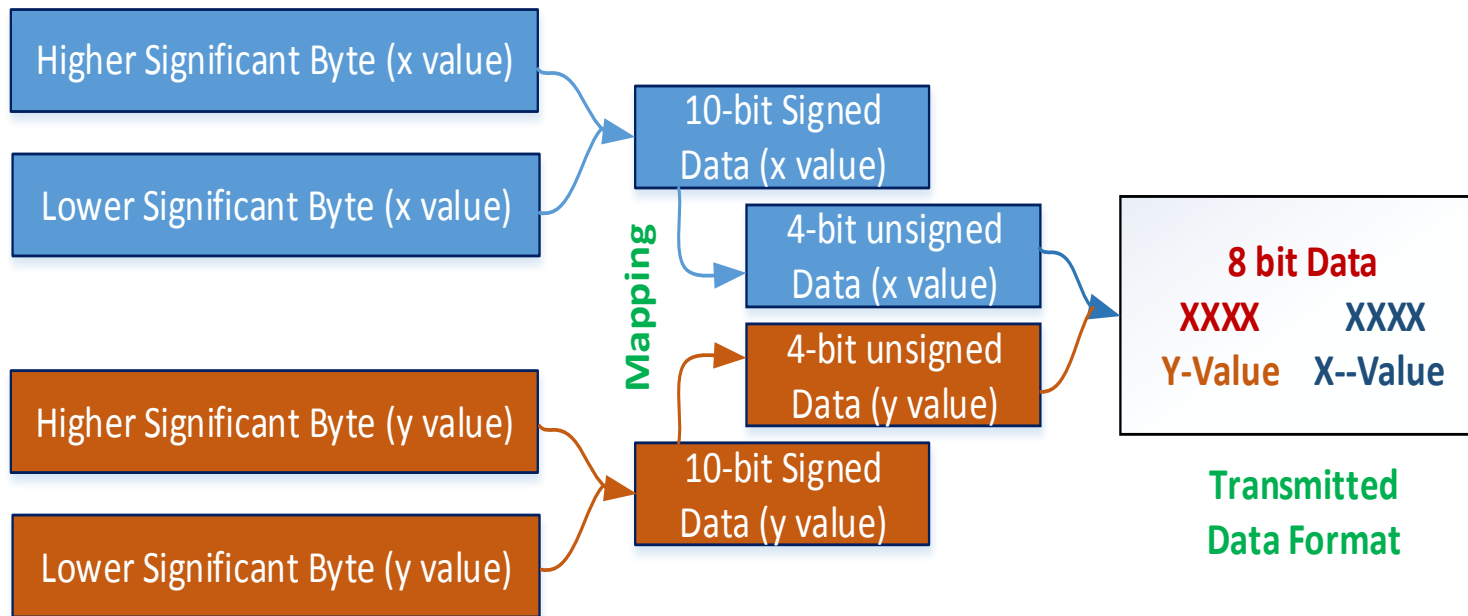
Data acquisition from (ADXL-345)- Cont.

```
Serial.print( x );  
Serial.print(",");  
Serial.print( y );  
Serial.print(",");  
Serial.print( z );  
Serial.print(",");  
Serial.print("\n");  
delay(1000);  
}  
void writeTo(byte address, byte val) {  
Wire.beginTransmission(DEVICE); // start transmission to device  
Wire.write(address); // send register address  
Wire.write(val); // send value to write  
Wire.endTransmission(); // end transmission  
}
```


Data acquisition from (ADXL-345)- Cont.

```
// Reads num bytes starting from address register on device in to _buff array
void readFrom(byte address, int num, byte data[]) {
Wire.beginTransmission(DEVICE); // start transmission to device
Wire.write(address); // sends address to read from
Wire.endTransmission(); // end transmission
Wire.beginTransmission(DEVICE); // start transmission to device
Wire.requestFrom(DEVICE, num); // request 6 bytes from device
int i = 0;
while(Wire.available()) // device may send less than requested (abnormal)
{
    data[i] = Wire.read(); // receive a byte
    i++;
}
Wire.endTransmission(); // end transmission
}
```

Transmitted data format

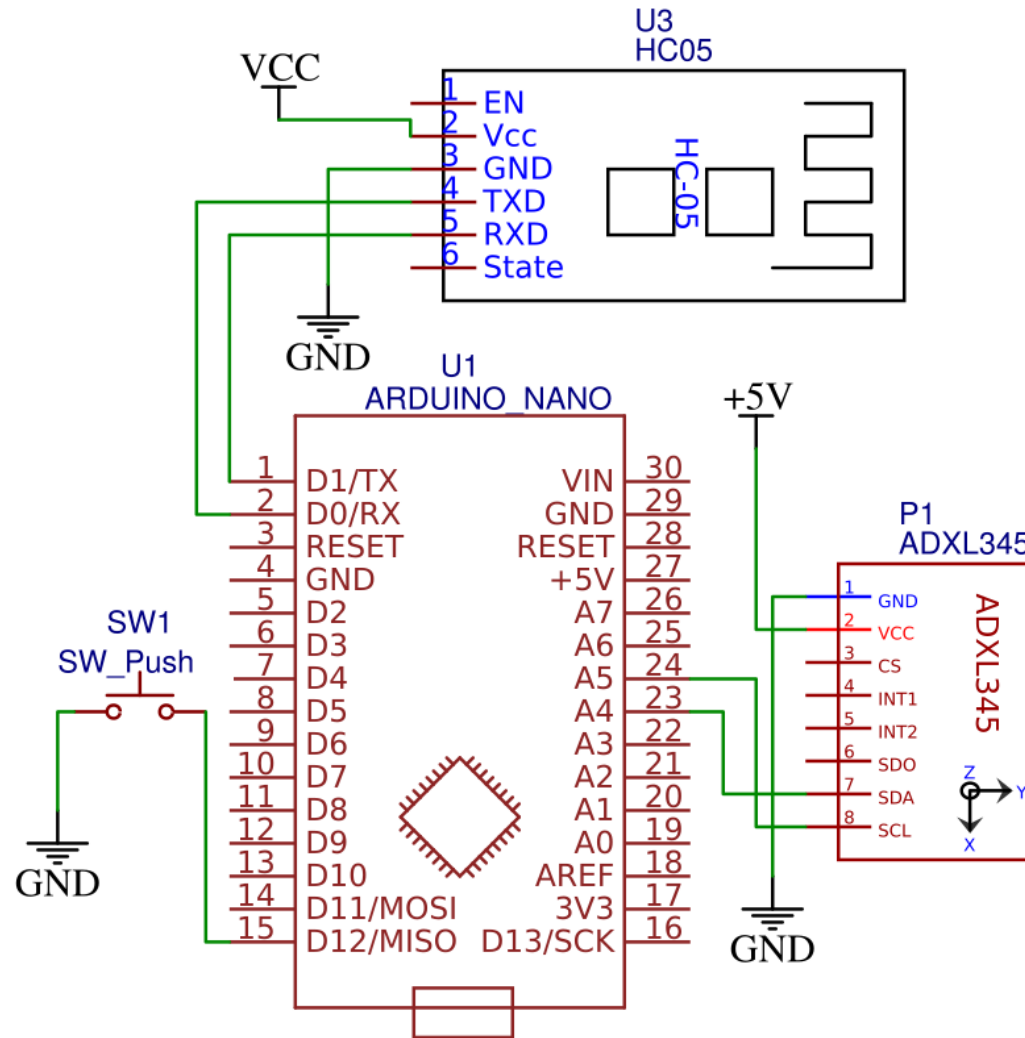


x value (-200 to +200) $\xrightarrow{\text{Mapping}}$ **0 to 15**
y value (-200 to +200) $\xrightarrow{\text{Mapping}}$ **0 to 15**

In normal condition: x value may change from -5 to 5 After mapping: 7 (0111)
y value may change from -5 to 5 After mapping: 7 (0111)

Transmitted data in normal condition: **01110111 (119)**

Transmitter Circuit



Gesture Controlled Robot (Transmitter)

```
#include <Wire.h>
#include <Bounce.h>
const int SWPin = 12;
#define DEVICE (0x53)
Bounce SW = Bounce(SWPin,10);
byte acc_data[6];
char POWER_CTL = 0x2D;
char DATA_FORMAT = 0x31;
char DATA_X0 = 0x32;
byte x_value,y_value,data;
void setup() {
  Wire.begin();
  Serial.begin(9600);
  writeTo(DATA_FORMAT, 0x01);
  writeTo(POWER_CTL, 0x08);
  pinMode(SWPin,INPUT);
  digitalWrite(SWPin, HIGH);
  pinMode(13,OUTPUT);
  digitalWrite(13, LOW);
}
```

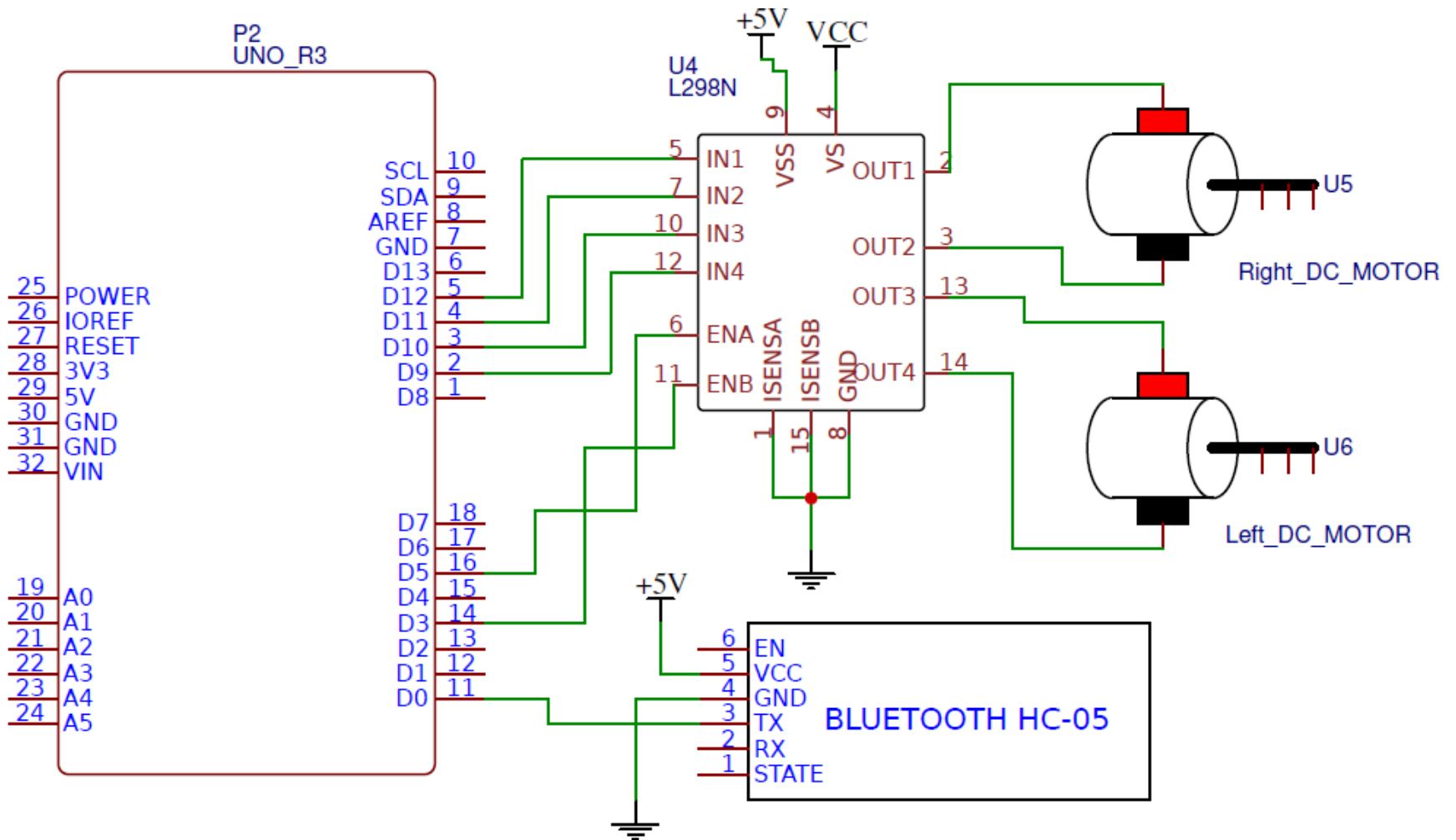
```
boolean a=0;
byte constant=0b01110111;
void loop()
{
  if (SW.update())
  {
    if( SW.risingEdge())
    a=!a;
    digitalWrite(13,!a);
  }
  if(a==0)
  {
    uint8_t howManyBytesToRead = 6;
    readFrom( DATA_X0, howManyBytesToRead,
    acc_data);
    int x = (((int)acc_data[1]) << 8) | acc_data[0]);
    int y = (((int)acc_data[3]) << 8) | acc_data[2]);
    x_value=map(x,-200,200,0,15);
    y_value=map(y,-200,200,0,15);
    data=x_value+(y_value<<4);
    Serial.write(data);
  }
}
```

```

} else
{ Serial.write(constant);
}
}
void writeTo(byte address, byte val) {
Wire.beginTransmission(DEVICE);
Wire.write(address);
Wire.write(val);
Wire.endTransmission();
}
void readFrom(byte address, int num, byte acc_data[]) {
Wire.beginTransmission(DEVICE);
Wire.write(address);
Wire.endTransmission();
Wire.beginTransmission(DEVICE);
Wire.requestFrom(DEVICE, num);
int i = 0;
while(Wire.available())
{
acc_data[i] = Wire.read();
i++;
}
Wire.endTransmission(); }

```

Receiver Circuit



Gesture Controlled Robot (Receiver)

```
#define RmotorP A0
#define RmotorN A1
#define LmotorP A2
#define LmotorN A3
int data=0;
void setup() {
  Serial.begin(9600);
  pinMode(RmotorP,OUTPUT);
  pinMode(RmotorN,OUTPUT);
  pinMode(LmotorP,OUTPUT);
  pinMode(LmotorN,OUTPUT);
  analogWrite(3,255);
  analogWrite(5,255);
}
void serialEvent() {
  while (Serial.available()) {
    data = Serial.read();
  }
  Serial.println(data);
}
```

```
void loop()
{

  byte x=(data& 0x0F);
  byte y=(data & 0xF0)>>4;
  if(x>8)
  {
    digitalWrite(RmotorP,HIGH);
    digitalWrite(RmotorN,LOW);
    digitalWrite(LmotorP,HIGH);
    digitalWrite(LmotorN,LOW);
  }
  else if(x<6)
  {
    digitalWrite(RmotorP,LOW);
    digitalWrite(RmotorN,HIGH);
    digitalWrite(LmotorP,LOW);
    digitalWrite(LmotorN,HIGH);
  }
}
```


Gesture Controlled Robot (Receiver)-Cont.

```
else {  
  if(y>8)  
  {  
    digitalWrite(RmotorP,LOW);  
    digitalWrite(RmotorN,LOW);  
    digitalWrite(LmotorP,HIGH);  
    digitalWrite(LmotorN,LOW);  
  }  
  else if(y<6) {  
    digitalWrite(RmotorP,HIGH);  
    digitalWrite(RmotorN,LOW);  
    digitalWrite(LmotorP,LOW);  
    digitalWrite(LmotorN,LOW);  
  }  
  else {  
    digitalWrite(RmotorP,LOW);  
    digitalWrite(RmotorN,LOW);  
    digitalWrite(LmotorP,LOW);  
    digitalWrite(LmotorN,LOW);  
  }  
}
```

Formation of IIUC IEEE Robotics and Automation Society

