

PROFESSIONAL TRAINING REPORT

at

Sathyabama Institute of Science and Technology

(Deemed to be University)

Submitted in partial fulfilment of the requirements for the award of

Bachelor of Engineering Degree in

Electrical and Electronics Engineering

By

AMUDHA SUBRAMANI (39140002)



DEPARTMENT OF ELECTRICAL AND ELECTRONICS

SCHOOL OF ELECTRICAL AND ELECTRONICS

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

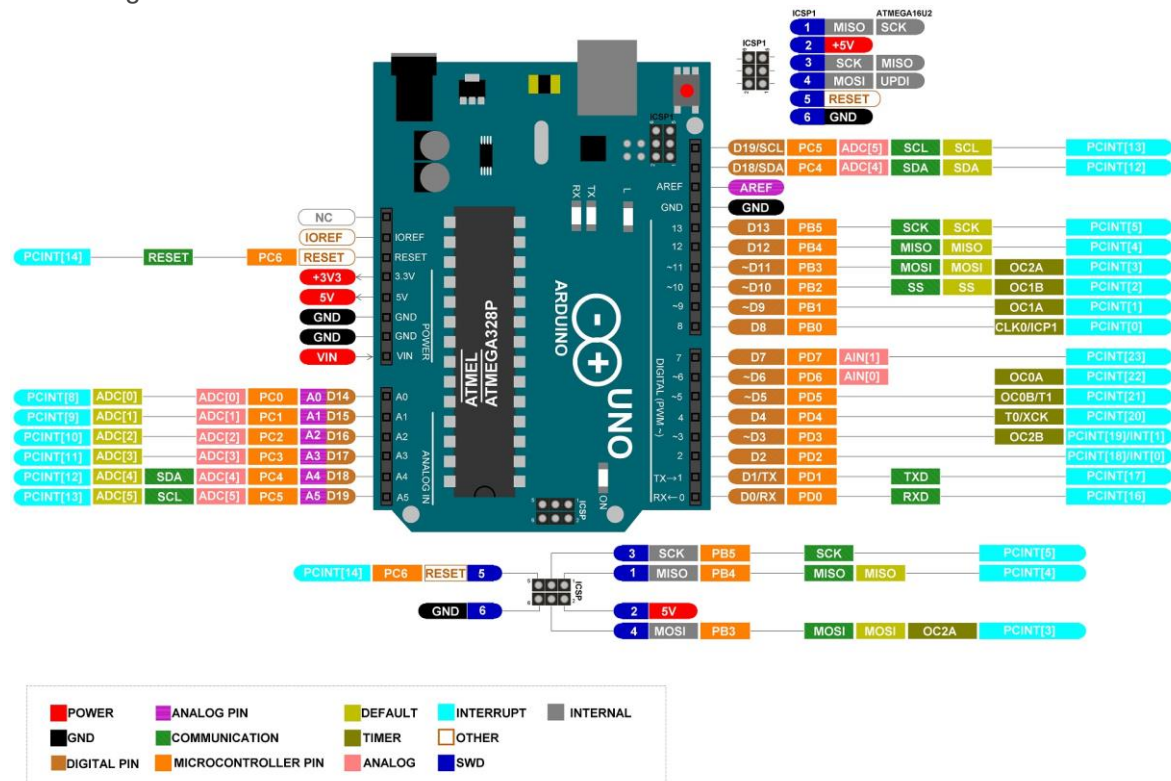
CHENNAI – 600119. TAMILNADU.



Introduction to Internet of Things (IoT) using Arduino

Introduction to Internet of Things (IoT) using Arduino This project's purpose is to discover a cheap but creative solution to monitor temperature and humidity that is caused by a very frequent air conditioning and electrical failure in our server room. After undergoing through a few study, our team learned that Arduino is the best project to explore since the total cost is less than RM100 and the hardware and information can be obtained easily. The Arduino acts as a sensor that will continuously sends information per minute to the MySQL server. The data obtained can be viewed through Web portal, Power BI and also Mobile Apps. This can be a game changer to our team especially to the standby staffs as they can use this tool as a part of their checklist in

monitoring server room's health.



What is IOT? IOT stands for Internet of Things

Is a system of interrelated computing devices, mechanical and and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction

IOT Examples

Examples of objects that can fall into the scope of Internet of Things include connected security systems, sensors, thermostats, cars, electronic appliances, light in the household and commercial environments, alarm clocks, speaker systems, vending machines and more.

The Future of IOT

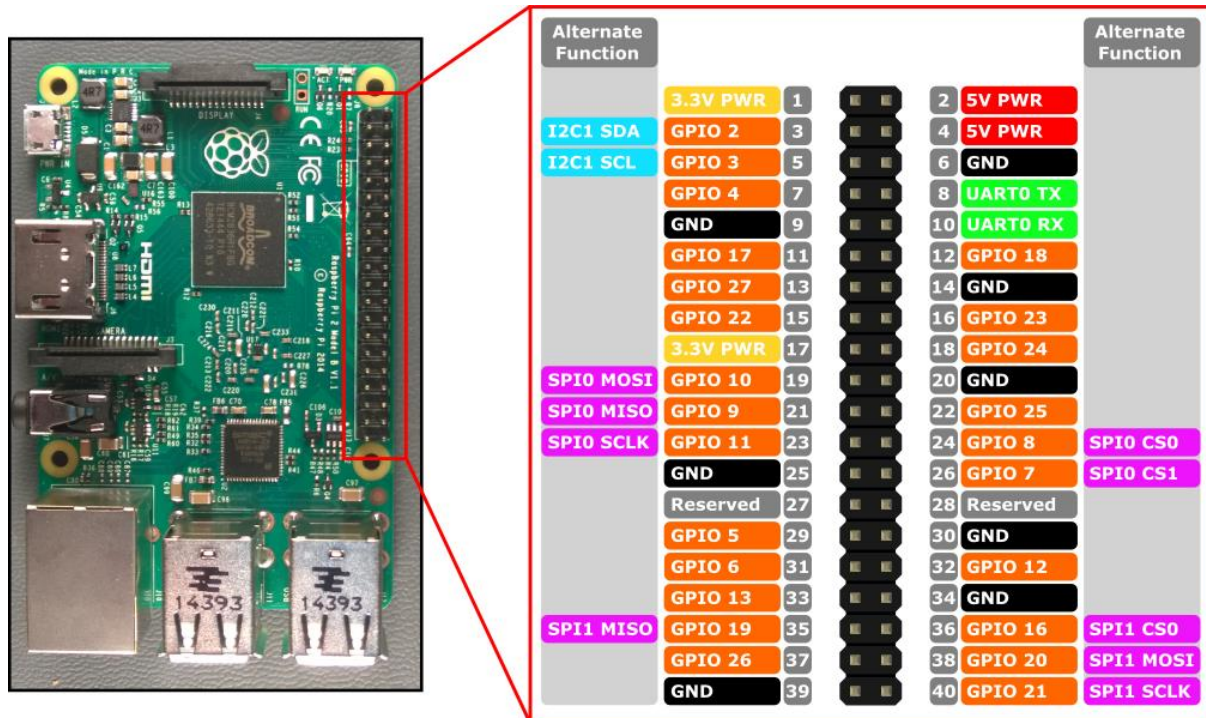
As far as the reach of the Internet of Things, there are more than 12 billion devices that can currently connect to the Internet, and researchers at IDC estimate that by 2020 there will be 26 times more connected things than people.

Why Arduino ?

It is an open-source project, software/hardware is extremely accessible and very flexible to be customized and extendedIt is flexible, offers a variety of digital and analog inputs, SPI and serial interface and digital and PWM outputsIt is easy to use, connects to computer via USB and communicates using standard serial protocol, runs in standalone mode and as interface connected to PC/Macintosh computersIt is inexpensive, around RM70 per board and comes with free authoring software Arduino is backed up by a growing online community, lots of source code is already available and we can share and post our examples for others to use, too.

Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs – lights on a sensor, a finger on a button, or a Twitter message – and turn it into an output – activating a motor, turning on a LED, publishing something online.



Features of the Arduino UNO:

- Microcontroller: ATmega328.
- Operating Voltage: 5V.
- Input Voltage (recommended): 7-12V.
-
- Input Voltage (limits): 6-20V.
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6.
- DC Current per I/O Pin: 40 mA.
- DC Current for 3.3V Pin: 50 mA.
-

Features of Arduino Uno Board

- The operating voltage is 5V.
- The recommended input voltage will range from 7v to 12V.
- The input voltage ranges from 6v to 20V.
- Digital input/output pins are 14.
- Analog i/p pins are 6.
- DC Current for each input/output pin is 40 mA.
- DC Current for 3.3V Pin is 50 mA.

- Flash Memory is 32 KB.

The key features are –

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

ARDUINO MATH LIBRARY

Arduino Math library (math.h) includes a number of useful mathematical functions for manipulating floating-point numbers.

Example

The following example shows how to use the most common math.h library functions –

```
double double__x = 45.45 ;
double double__y = 30.20 ;

void setup() {
  Serial.begin(9600);
  Serial.print("cos num = ");
  Serial.println (cos (double__x) ); // returns cosine of x
  Serial.print("absolute value of num = ");
  Serial.println (fabs (double__x) ); // absolute value of a
float
  Serial.print("floating point modulo = ");
  Serial.println (fmod (double__x, double__y)); // floating
point modulo
  Serial.print("sine of num = ");
  Serial.println (sin (double__x) ) ;// returns sine of x
  Serial.print("square root of num : ");
  Serial.println ( sqrt (double__x) );// returns square root of
x
  Serial.print("tangent of num : ");
  Serial.println ( tan (double__x) ); // returns tangent of x
  Serial.print("exponential value of num : ");
  Serial.println ( exp (double__x) ); // function returns the
exponential value of x.
  Serial.print("cos num : ");
```

```

    Serial.println (atan (double__x) ); // arc tangent of x
    Serial.print("tangent of num : ");
    Serial.println (atan2 (double__y, double__x) );// arc tangent
of y/x
    Serial.print("arc tangent of num : ");
    Serial.println (log (double__x) ) ; // natural logarithm of x
    Serial.print("cos num : ");
    Serial.println ( log10 (double__x)); // logarithm of x to base
10.
    Serial.print("logarithm of num to base 10 : ");
    Serial.println (pow (double__x, double__y) );// x to power of
y
    Serial.print("power of num : ");
    Serial.println (square (double__x)); // square of x
}

void loop() {

}

```

Result

```

cos num = 0.10
absolute value of num = 45.45
floating point modulo =15.25
sine of num = 0.99
square root of num : 6.74
tangent of num : 9.67
exponential value of num : ovf
cos num : 1.55
tangent of num : 0.59
arc tangent of num : 3.82
cos num : 1.66
logarithm of num to base 10 : inf
power of num : 2065.70

```

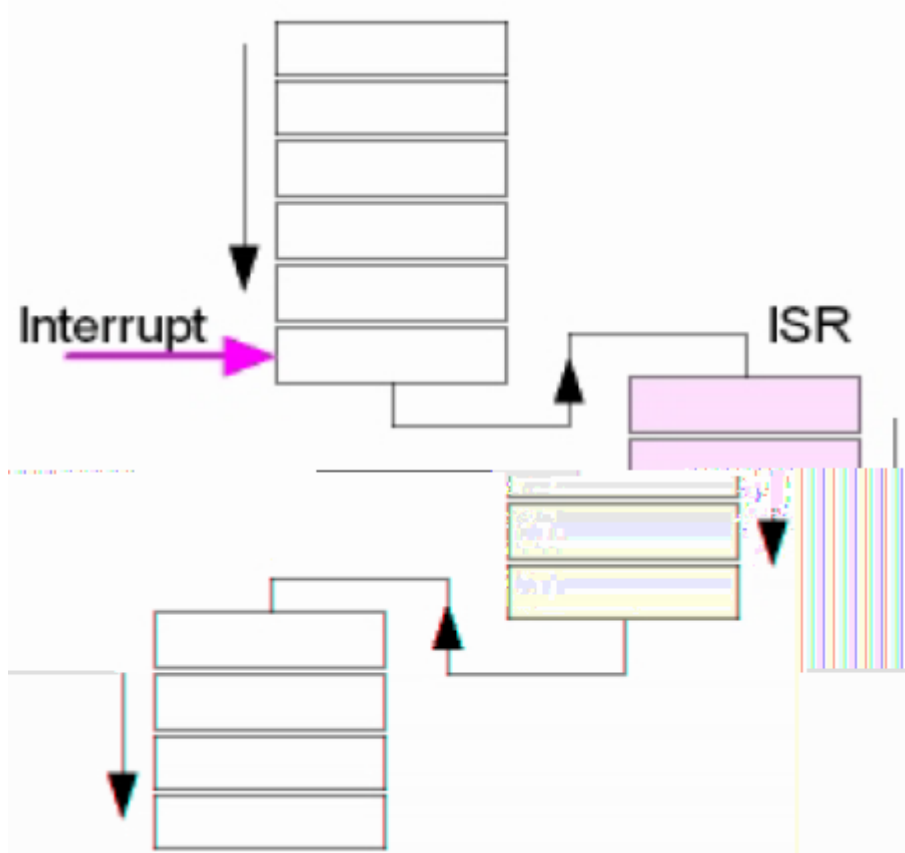
Arduino - Interrupts

Interrupts stop the current work of Arduino such that some other work can be done.

Suppose you are sitting at home, chatting with someone. Suddenly the telephone rings. You stop chatting, and pick up the telephone to speak to the caller. When you have finished your telephonic conversation, you go back to chatting with the person before the telephone rang.

Similarly, you can think of the main routine as chatting to someone, the telephone ringing causes you to stop chatting. The interrupt service routine is the process of talking on the telephone. When the telephone conversation ends, you then go back to your main routine of chatting. This example explains exactly how an interrupt causes a processor to act.

The main program is running and performing some function in a circuit. However, when an interrupt occurs the main program halts while another routine is carried out. When this routine finishes, the processor goes back to the main routine again.



Important features

Here are some important features about interrupts –

- Interrupts can come from various sources. In this case, we are using a hardware interrupt that is triggered by a state change on one of the digital pins.
- Most Arduino designs have two hardware interrupts (referred to as "interrupt0" and "interrupt1") hard-wired to digital I/O pins 2 and 3, respectively.
- The Arduino Mega has six hardware interrupts including the additional interrupts ("interrupt2" through "interrupt5") on pins 21, 20, 19, and 18.
- You can define a routine using a special function called as “Interrupt Service Routine” (usually known as ISR).
- You can define the routine and specify conditions at the rising edge, falling edge or both. At these specific conditions, the interrupt would be serviced.
- It is possible to have that function executed automatically, each time an event happens on an input pin.

Types of Interrupts

There are two types of interrupts –

- **Hardware Interrupts** – They occur in response to an external event, such as an external interrupt pin going high or low.
- **Software Interrupts** – They occur in response to an instruction sent in software. The only type of interrupt that the “Arduino language” supports is the `attachInterrupt()` function.

Using Interrupts in Arduino

Interrupts are very useful in Arduino programs as it helps in solving timing problems. A good application of an interrupt is reading a rotary encoder or observing a user input. Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time. Other interrupts will be executed after the current one finishes in an order that depends on the priority they have.

Typically, global variables are used to pass data between an ISR and the main program. To make sure variables shared between an ISR and the main program are updated correctly, declare them as `volatile`.

attachInterrupt Statement Syntax

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); //recommended for arduino board
attachInterrupt(pin, ISR, mode) ; //recommended Arduino Due, Zero only
//argument pin: the pin number
//argument ISR: the ISR to call when the interrupt occurs;
//this function must take no parameters and return nothing.
//This function is sometimes referred to as an interrupt service routine.
//argument mode: defines when the interrupt should be triggered.
```

The following three constants are predefined as valid values –

- **LOW** to trigger the interrupt whenever the pin is low.
- **CHANGE** to trigger the interrupt whenever the pin changes value.
- **FALLING** whenever the pin goes from high to low.

Example

```
int pin = 2; //define interrupt pin to 2
volatile int state = LOW; // To make sure variables shared between an ISR
//the main program are updated correctly, declare them as volatile.

void setup() {
    pinMode(13, OUTPUT); //set pin 13 as output
    attachInterrupt(digitalPinToInterrupt(pin), blink, CHANGE);
    //interrupt at pin 2 blink ISR when pin to change the value
}

void loop() {
    digitalWrite(13, state); //pin 13 equal the state value
}
```

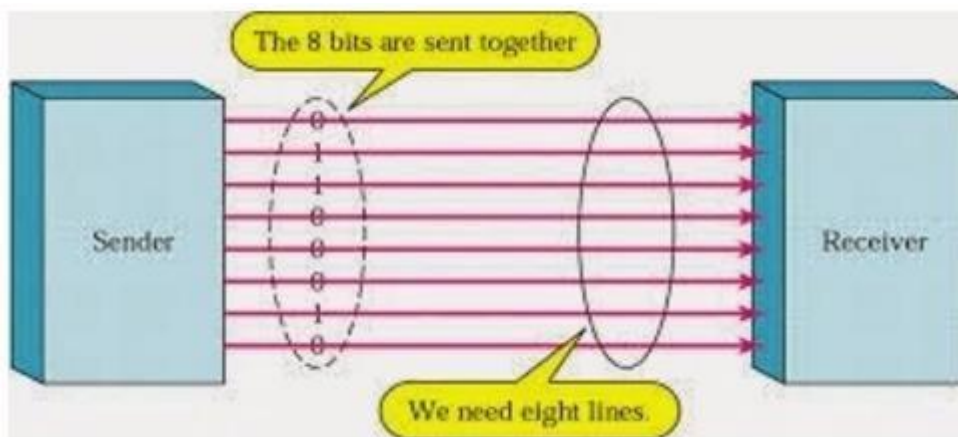
```
void blink() {  
    //ISR function  
    state = !state; //toggle the state when the interrupt occurs  
}
```

Arduino - Communication

Hundreds of communication protocols have been defined to achieve this data exchange. Each protocol can be categorized into one of the two categories: parallel or serial.

Parallel Communication

Parallel connection between the Arduino and peripherals via input/output ports is the ideal solution for shorter distances up to several meters. However, in other cases when it is necessary to establish communication between two devices for longer distances it is not possible to use parallel connection. Parallel interfaces transfer multiple bits at the same time. They usually require buses of data - transmitting across eight, sixteen, or more wires. Data is transferred in huge, crashing waves of 1's and 0's.



Advantages and Drawbacks of Parallel Communication

Parallel communication certainly has its advantages. It is faster than serial, straightforward, and relatively easy to implement. However, it requires many input/output (I/O) ports and lines. If you have ever had to move a project from a basic Arduino Uno to a Mega, you know that the I/O lines on a microprocessor can be precious and few. Therefore, we prefer serial communication, sacrificing potential speed for pin real estate.

Serial Communication Modules

Today, most Arduino boards are built with several different systems for serial communication as standard equipment.

Which of these systems are used depends on the following factors –

- How many devices the microcontroller has to exchange data with?
- How fast the data exchange has to be?
- What is the distance between these devices?

- Is it necessary to send and receive data simultaneously?

One of the most important things concerning serial communication is the **Protocol**, which should be strictly observed. It is a set of rules, which must be applied such that the devices can correctly interpret data they mutually exchange. Fortunately, Arduino automatically takes care of this, so that the work of the programmer/user is reduced to simple write (data to be sent) and read (received data).

Types of Serial Communications

Serial communication can be further classified as –

- **Synchronous** – Devices that are synchronized use the same clock and their timing is in synchronization with each other.
- **Asynchronous** – Devices that are asynchronous have their own clocks and are triggered by the output of the previous state.

It is easy to find out if a device is synchronous or not. If the same clock is given to all the connected devices, then they are synchronous. If there is no clock line, it is asynchronous.

For example, UART (Universal Asynchronous Receiver Transmitter) module is asynchronous.

The asynchronous serial protocol has a number of built-in rules. These rules are nothing but mechanisms that help ensure robust and error-free data transfers. These mechanisms, which we get for eschewing the external clock signal, are –

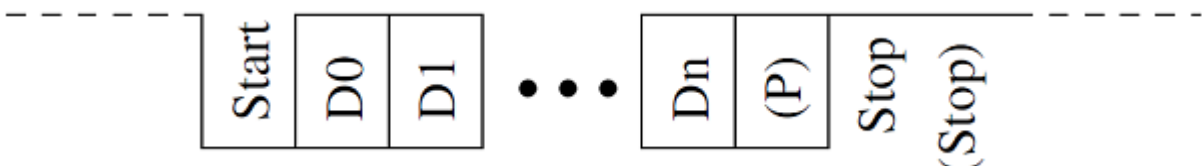
- Synchronization bits
- Data bits
- Parity bits
- Baud rate

Synchronization Bits

The synchronization bits are two or three special bits transferred with each packet of data. They are the start bit and the stop bit(s). True to their name, these bits mark the beginning and the end of a packet respectively.

There is always only one start bit, but the number of stop bits is configurable to either one or two (though it is normally left at one).

The start bit is always indicated by an idle data line going from 1 to 0, while the stop bit(s) will transition back to the idle state by holding the line at 1.



Data Bits

The amount of data in each packet can be set to any size from 5 to 9 bits. Certainly, the standard data size is your basic 8-bit byte, but other sizes have their uses. A 7-

bit data packet can be more efficient than 8, especially if you are just transferring 7-bit ASCII characters.

Parity Bits

The user can select whether there should be a parity bit or not, and if yes, whether the parity should be odd or even. The parity bit is 0 if the number of 1's among the data bits is even. Odd parity is just the opposite.

Baud Rate

The term baud rate is used to denote the number of bits transferred per second [bps]. Note that it refers to bits, not bytes. It is usually required by the protocol that each byte is transferred along with several control bits. It means that one byte in serial data stream may consist of 11 bits. For example, if the baud rate is 300 bps then maximum 37 and minimum 27 bytes may be transferred per second.

Arduino UART

The following code will make Arduino send hello world when it starts up.

```
void setup() {
    Serial.begin(9600); //set up serial library baud rate to 9600
    Serial.println("hello world"); //print hello world
}

void loop() {
}
```

After the Arduino sketch has been uploaded to Arduino, open the Serial monitor  at the top right section of Arduino IDE.

Type anything into the top box of the Serial Monitor and press send or enter on your keyboard. This will send a series of bytes to the Arduino.

The following code returns whatever it receives as an input.

The following code will make Arduino deliver output depending on the input provided.

```
void setup() {
    Serial.begin(9600); //set up serial library baud rate to 9600
}

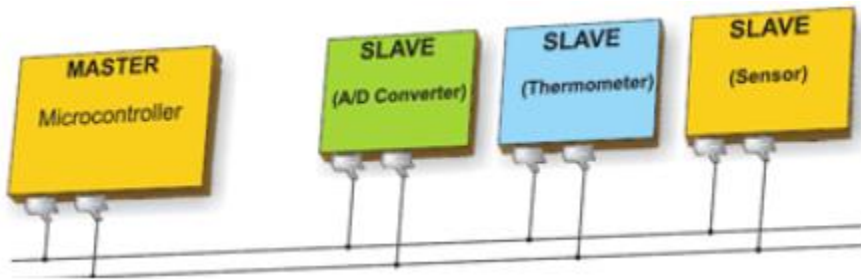
void loop() {
    if(Serial.available()) //if number of bytes (characters)
    available for reading from {
        serial port
        Serial.print("I received:"); //print I received
        Serial.write(Serial.read()); //send what you read
    }
}
```

Notice that **Serial.print** and **Serial.println** will send back the actual ASCII code, whereas **Serial.write** will send back the actual text. See ASCII codes for more information.

Arduino - Inter Integrated Circuit

Inter-integrated circuit (I2C) is a system for serial data exchange between the microcontrollers and specialized integrated circuits of a new generation. 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. One is used for data transfer and the other is used for synchronization (clock signal).

As seen in the following figure, one device is always a master. It performs addressing of one slave chip before the communication starts. In this way, one microcontroller can communicate with 112 different devices. Baud rate is usually 100 Kb/sec (standard mode) or 10 Kb/sec (slow baud rate mode). Systems with the baud rate of 3.4 Mb/sec have recently appeared. The distance between devices, which communicate over an I2C bus is limited to several meters.



Board I2C Pins

The I2C bus consists of two signals – SCL and SDA. SCL is the clock signal, and SDA is the data signal. The current bus master always generates the clock signal. Some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is known as “clock stretching”.

Following are the pins for different Arduino boards –

- Uno, Pro Mini A4 (SDA), A5 (SCL)
- Mega, Due 20 (SDA), 21 (SCL)
- Leonardo, Yun 2 (SDA), 3 (SCL)

Arduino I2C

We have two modes - master code and slave code - to connect two Arduino boards using I2C. They are –

- Master Transmitter / Slave Receiver
- Master Receiver / Slave Transmitter

Master Transmitter / Slave Receiver

Let us now see what is master transmitter and slave receiver.

Master Transmitter

The following functions are used to initialize the Wire library and join the I2C bus as a master or slave. This is normally called only once.

- **Wire.begin(address)** – Address is the 7-bit slave address in our case as the master is not specified and it will join the bus as a master.

Wire.beginTransmission(address) – Begin a transmission to the I2C slave device with the given address. Arduino - Trigonometric Functions

You need to use Trigonometry practically like calculating the distance for moving object or angular speed. Arduino provides traditional trigonometric functions (sin, cos, tan, asin, acos, atan) that can be summarized by writing their prototypes. Math.h contains the trigonometry function's prototype.

Trigonometric Exact Syntax

```
double sin(double x); //returns sine of x radians
double cos(double y); //returns cosine of y radians
double tan(double x); //returns the tangent of x radians
double acos(double x); //returns A, the angle corresponding to
cos (A) = x
double asin(double x); //returns A, the angle corresponding to
sin (A) = x
double atan(double x); //returns A, the angle corresponding to
tan (A) = x
```

Example

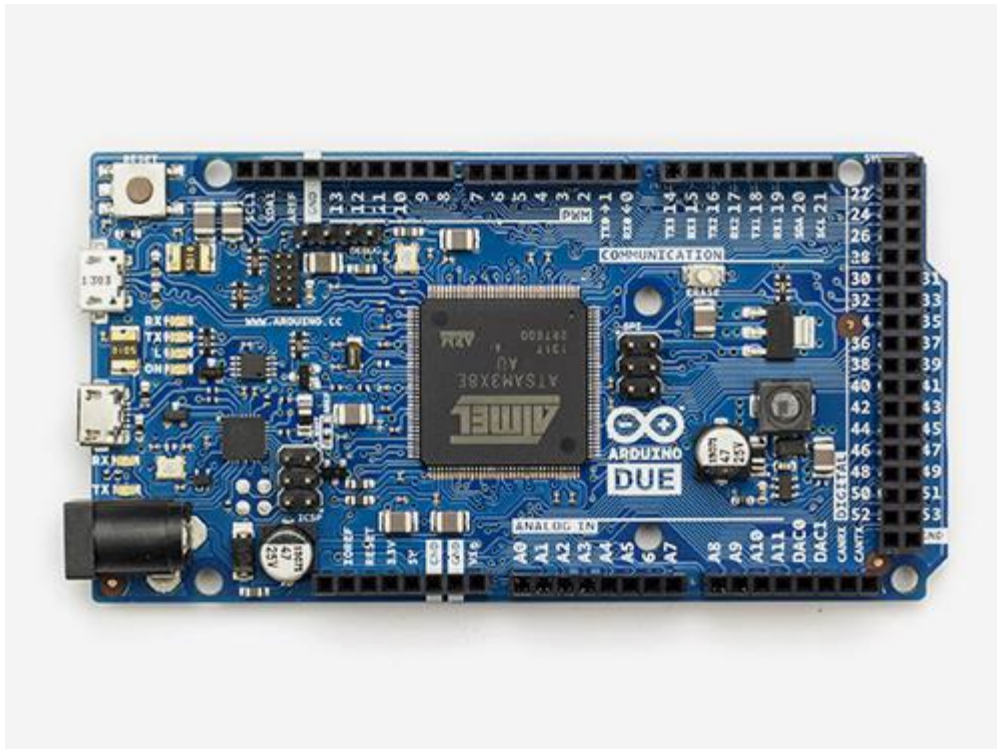
```
double sine = sin(2); // approximately 0.90929737091
double cosine = cos(2); // approximately -0.41614685058
double tangent = tan(2); // approximately -2.18503975868
```

Arduino - Due & Zero

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller.

Important features –

- It has 54 digital input/output pins (of which 12 can be used as PWM outputs)
- 12 analog inputs
- 4 UARTs (hardware serial ports)
- 84 MHz clock, an USB OTG capable connection
- 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header
- Reset button and an erase button



Characteristics of the Arduino Due Board

Operating volt	CPU speed	Analog in/out	Digital IO/ PWM	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
3.3 Volt	84 Mhz	12/2	54/12	-	96	512	2 micro	4

Communication

- 4 Hardware UARTs
- 2 I2C
- 1 CAN Interface (Automotive communication protocol)
- 1 SPI
- 1 Interface JTAG (10 pin)
- 1 USB Host (like as Leonardo)
- 1 Programming Port

Unlike most Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board.

The board contains everything needed to support the microcontroller. You can simply connect it to a computer with a micro-USB cable or power it with an AC-to-DC adapter or battery to get started. The Due is compatible with all Arduino shields that work at 3.3V.

Arduino Zero

The Zero is a simple and powerful 32-bit extension of the platform established by the UNO. The Zero board expands the family by providing increased performance, enabling a variety of project opportunities for devices, and acts as a great educational tool for learning about 32-bit application development.

Important features are –

- The Zero applications span from smart IoT devices, wearable technology, high-tech automation, to crazy robotics.
- The board is powered by Atmel's SAMD21 MCU, which features a 32-bit ARM Cortex® M0+ core.
- One of its most important features is Atmel's Embedded Debugger (EDBG), which provides a full debug interface without the need for additional hardware, significantly increasing the ease-of-use for software debugging.
- EDBG also supports a virtual COM port that can be used for device and bootloader programming.



Characteristics of the Arduino Zero board

Operating volt	CPU speed	Analog in/out	Digital IO/ PWM	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
3.3 Volt	48 Mhz	6/1	14/10	-	32	256	2 micro	2

Unlike most Arduino and Genuino boards, the Zero runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board.

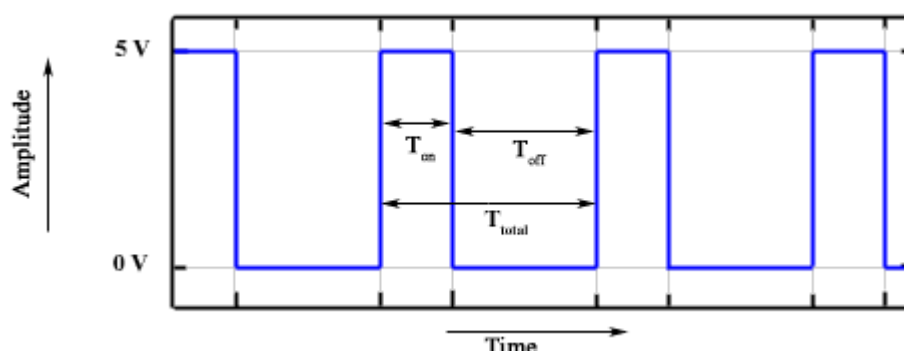
The board contains everything needed to support the microcontroller. You can simply connect it to a computer with a micro-USB cable or power it with an AC-to-DC adapter or a battery to get started. The Zero is compatible with all the shields that work at 3.3V.

Arduino - Pulse Width Modulation

Pulse Width Modulation or PWM is a common technique used to vary the width of the pulses in a pulse-train. PWM has many applications such as controlling servos and speed controllers, limiting the effective power of motors and LEDs.

Basic Principle of PWM

Pulse width modulation is basically, a square wave with a varying high and low time. A basic PWM signal is shown in the following figure.



There are various terms associated with PWM –

- **On-Time** – Duration of time signal is high.
- **Off-Time** – Duration of time signal is low.
- **Period** – It is represented as the sum of on-time and off-time of PWM signal.
- **Duty Cycle** – It is represented as the percentage of time signal that remains on during the period of the PWM signal.

Period

As shown in the figure, T_{on} denotes the on-time and T_{off} denotes the off-time of signal. Period is the sum of both on and off times and is calculated as shown in the following equation –

$$T_{total} = T_{on} + T_{off}$$

Duty Cycle

Duty cycle is calculated as the on-time of the period of time. Using the period calculated above, duty cycle is calculated as –

$$D = \frac{T_{on}}{T_{on} + T_{off}} = \frac{T_{on}}{T_{total}}$$

analogWrite() Function

The **analogWrite()** function writes an analog value (PWM wave) to a pin. It can be used to light a LED at varying brightness or drive a motor at various speeds. After a call of the **analogWrite()** function, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** or a call to **digitalRead()** or **digitalWrite()** on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support **analogWrite()** on pins 9, 10, and 11.



The Arduino Due supports **analogWrite()** on pins 2 through 13, and pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.

analogWrite() Function Syntax

`analogWrite ( pin , value ) ;`

value – the duty cycle: between 0 (always off) and 255 (always on).

Example

```
int ledPin = 9; // LED connected to digital pin 9
int analogPin = 3; // potentiometer connected to analog pin 3
int val = 0; // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, (val / 4)); // analogRead values go from 0
  // to 1023,
  // analogWrite values from 0 to 255
}
```

-
- **Wire.write(value)** – Queues bytes for transmission from a master to slave device (in-between calls to `beginTransaction()` and `endTransmission()`).

- **Wire.endTransmission()** – Ends a transmission to a slave device that was begun by beginTransmission() and transmits the bytes that were queued by wire.write().

Example

```
#include <Wire.h> //include wire library

void setup() //this will run only once {
    Wire.begin(); // join i2c bus as master
}

short age = 0;

void loop() {
    Wire.beginTransmission(2);
    // transmit to device #2
    Wire.write("age is = ");
    Wire.write(age); // sends one byte
    Wire.endTransmission(); // stop transmitting
    delay(1000);
}
```

Slave Receiver

The following functions are used –

- **Wire.begin(address)** – Address is the 7-bit slave address.
- **Wire.onReceive(received data handler)** – Function to be called when a slave device receives data from the master.
- **Wire.available()** – Returns the number of bytes available for retrieval with Wire.read(). This should be called inside the Wire.onReceive() handler.

Example

```
#include <Wire.h> //include wire library

void setup() { //this will run only once
    Wire.begin(2); // join i2c bus with address #2
    Wire.onReceive(receiveEvent); // call receiveEvent when the
    master send any thing
    Serial.begin(9600); // start serial for output to print what
    we receive
}

void loop() {
    delay(250);
}

//-----this function will execute whenever data is received from
master-----//

void receiveEvent(int howMany) {
    while (Wire.available() > 1) // loop through all but the last {
```

```
char c = Wire.read(); // receive byte as a character
Serial.print(c); // print the character
}
}
```

Arduino - Random Numbers

To generate random numbers, you can use Arduino random number functions. We have two functions –

- randomSeed(seed)
- random()

randomSeed (seed)

The function randomSeed(seed) resets Arduino's pseudorandom number generator. Although the distribution of the numbers returned by random() is essentially random, the sequence is predictable. You should reset the generator to some random value. If you have an unconnected analog pin, it might pick up random noise from the surrounding environment. These may be radio waves, cosmic rays, electromagnetic interference from cell phones, fluorescent lights and so on.

Example

```
randomSeed(analogRead(5)); // randomize using noise from analog
pin 5
```

random()

The random function generates pseudo-random numbers. Following is the syntax.

random() Statements Syntax

```
long random(max) // it generate random numbers from 0 to max
long random(min, max) // it generate random numbers from min to
max
```

Example

```
long randNumber;

void setup() {
  Serial.begin(9600);
  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

void loop() {
  // print a random number from 0 to 299
  Serial.print("random1=");
  randNumber = random(300);
  Serial.println(randNumber); // print a random number from 0 to
299
  Serial.print("random2=");
```

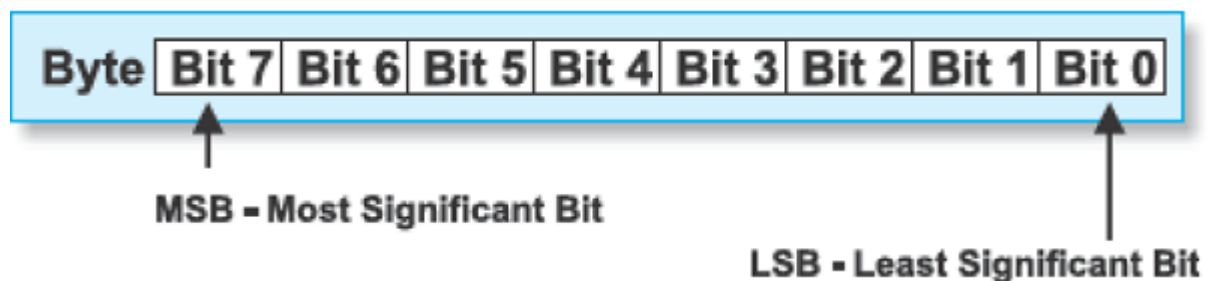
```
randNumber = random(10, 20); // print a random number from 10
to 19
Serial.println (randNumber);
delay(50);
}
```

Let us now refresh our knowledge on some of the basic concepts such as bits and bytes.

Bits

A bit is just a binary digit.

- The binary system uses two digits, 0 and 1.
- Similar to the decimal number system, in which digits of a number do not have the same value, the 'significance' of a bit depends on its position in the binary number. For example, digits in the decimal number 666 are the same, but have different values.



Bytes

A byte consists of eight bits.

- If a bit is a digit, it is logical that bytes represent numbers.
- All mathematical operations can be performed upon them.
- The digits in a byte do not have the same significance either.
- The leftmost bit has the greatest value called the Most Significant Bit (MSB).
- The rightmost bit has the least value and is therefore, called the Least Significant Bit (LSB).
- Since eight zeros and ones of one byte can be combined in 256 different ways, the largest decimal number that can be represented by one byte is 255 (one combination represents a zero).