CS3691 EMBEDDED SYSTEMS AND IOT

L T P C

3 0 2 4

COURSE OBJECTIVES:

- To learn the internal architecture and programming of an embedded processor.
- To introduce interfacing I/O devices to the processor.
- To introduce the evolution of the Internet of Things (IoT).
- To build a small low-cost embedded and IoT system using Arduino/RaspberryPi/ openplatform.
- To apply the concept of Internet of Things in real world scenario.

UNIT I 8-BIT EMBEDDED PROCESSOR

8-Bit Microcontroller – Architecture – Instruction Set and Programming – Programming ParallelPorts – Timers and Serial Port – Interrupt Handling.

UNIT II EMBEDDED C PROGRAMMING

Memory And I/O Devices Interfacing – Programming Embedded Systems in C – Need For RTOS –Multiple Tasks and Processes – Context Switching – Priority Based Scheduling Policies.

UNIT III IOT AND ARDUINO PROGRAMMING

Introduction to the Concept of IoT Devices – IoT Devices Versus
Computers – IoT Configurations – Basic Components – Introduction to Arduino – Types of Arduino– Arduino Toolchain – Arduino Programming Structure – Sketches – Pins – Input/Output FromPins Using Sketches – Introduction to Arduino Shields – Integration of Sensors and Actuators withArduino.

UNIT IV IOT COMMUNICATION AND OPEN PLATFORMS

IoT Communication Models and APIs – IoT Communication Protocols – Bluetooth – WiFi – ZigBee– GPS – GSM modules – Open Platform (like Raspberry Pi) – Architecture – Programming –Interfacing – Accessing GPIO Pins – Sending and Receiving Signals Using GPIO Pins –Connecting to the Cloud.

UNIT V APPLICATIONS DEVELOPMENT

Complete Design of Embedded Systems – Development of IoT Applications – Home Automation –Smart Agriculture – Smart Cities – Smart Healthcare.

UNIT I

8-BIT EMBEDDED PROCESSOR

- 8-Bit Microcontroller
- Architecture
- Instruction Set and Programming
- Programming Parallel Ports
- Timers and Serial Port
- Interrupt Handling.

1.1. 8051 Microcontroller

8051 microcontroller is an 8-bit microcontroller created in 1981 by Intel Corporation. It has an8-bit processor that simply means that it operates on 8-bit data at a time. It is among the most popular and commonly used microcontroller.

As it is an 8-bit microcontroller thus has 8-bit data bus, 16-bit address bus. Along with that, it holds **4 KB ROM** with **128 bytes RAM**.

1.1.1. What a Microcontroller is?

A microcontroller is an integrated chip designed under **V**ery **L**arge **S**cale **I**ntegration technique that consists of a processor with other peripheral units like memory, I/O port, timer, decoder, ADC etc. A microcontroller is basically designed in such a way that all the working peripherals are embedded in a single chip with the processor.

Any programmable device holds a processor, memory, I/O ports and timer within it. But a microcontroller contains all these components embedded in a single chip. This single-chip manages the overall operation of the device.

A microprocessor simply contains a CPU that processes the operations with the help of other peripheral units. Microprocessors are used where huge space is present to inbuilt a large motherboard like in PCs.

1.2. Architecture of 8051 Microcontroller

The figure below represents the architectural block diagram of 8051 microcontroller:

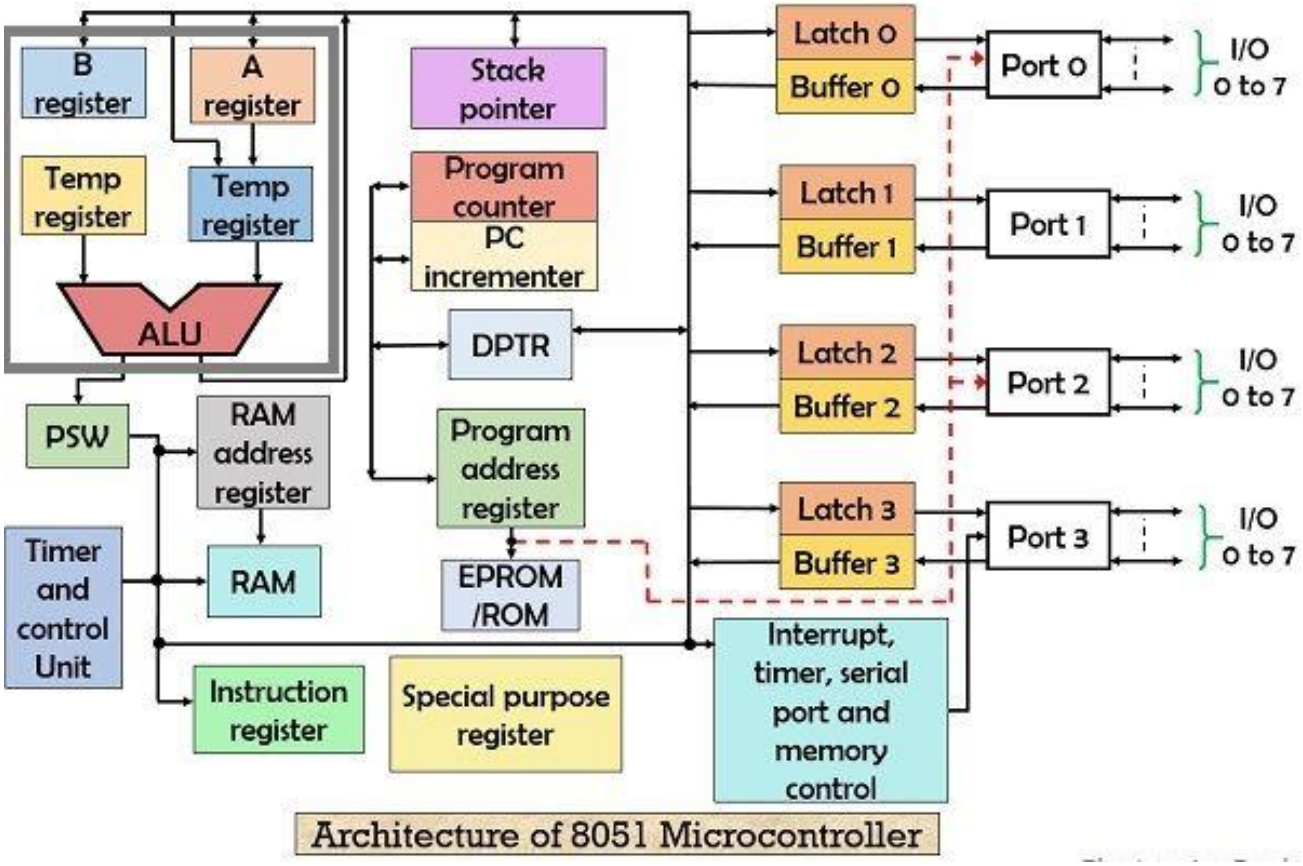
As we can see that several units are present in the above architecture. And every unit is embedded to execute the desired operation. Let us now discuss the operation of each unit present in the architecture.

1.2.1. Central processing unit (CPU): 8051 uses the 8-bit processor. This unit carries out the operation on 8-bit data. A processor is the heart of microcontroller. As the execution of the program stored in the memory is performed by the processor.

The unit performs arithmetic and logical operations on 8-bit data as it has ALU, with internal registers and program counters.

Several logical operations are performed by the ALU according to the program stored in the memory.

The processor of 8051 microcontrollers possesses a special feature by which it can process single bit or 8-bit data. This simply means that it has the ability to access each single bit data either to clear, set or move etc. for any logical computation.



Architecture of 8051 Microcontroller

1.2.2. Memory: Basically 8051 microcontroller consists of on-chip program memory i.e., ROM and on-chip data memory i.e., RAM.

Let us first understand

- **ROM**

  8051 microcontroller has 4 KB ROM with 0000H to 0FFFH as the addressable space. It is completely a program or code memory that means used by the programmer to store the programs that are to be executed by the microcontroller.
- The operations that are executed by the device in which the microcontroller is present are stored in the ROM of the memory at the time of fabrication. Hence cannot be changed or modified.

- **RAM**

  8051 holds a 128 bytes RAM. Basically, RAM is used to store data or operands for only a small time duration. It can be altered anytime according to the need of the user. It is also known as the data memory as it stores the data temporarily.
-

  Out of the 128-byte RAM, first, 32 bytes is held by the working registers. Basically, these are 4 banks which separately has 8 registers. These registers are accessed either by its name or address. It is to be noted here that at a particular time only a single register bank can be used.

- As in 8051, the data and program memory i.e., RAM and ROM hold a definite memory space. However, for some applications there exist the need for external memory to enhance the memory space, thus external RAM, ROM/EPROM is used by the 8051 microcontrollers.

1.2.3. Input/ Output port: 8051 consists of 4 parallel ports of 8 bit each thereby providing 32 input-output pins. All the 4 ports function bidirectional i.e., either input or output according to the software control.

1.2.4. Timer and Control Unit: Timers are used to create a time gap or delay between 2 events. 8051 microcontroller consists of 2 timers of 16 bit each by which the system can produce two delays simultaneously in order to generate the appropriate delay.

1.2.5. 8051 Flag Bits and PSW Register

The program status word (PSW) register is an 8-bit register, also known as flag register. It is of 8-bit wide but only 6-bit of it is used. The two unused bits are user-defined flags.

1.2.6. The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. The Accumulator, R0–R7 registers and B register are 1-byte value registers. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR.

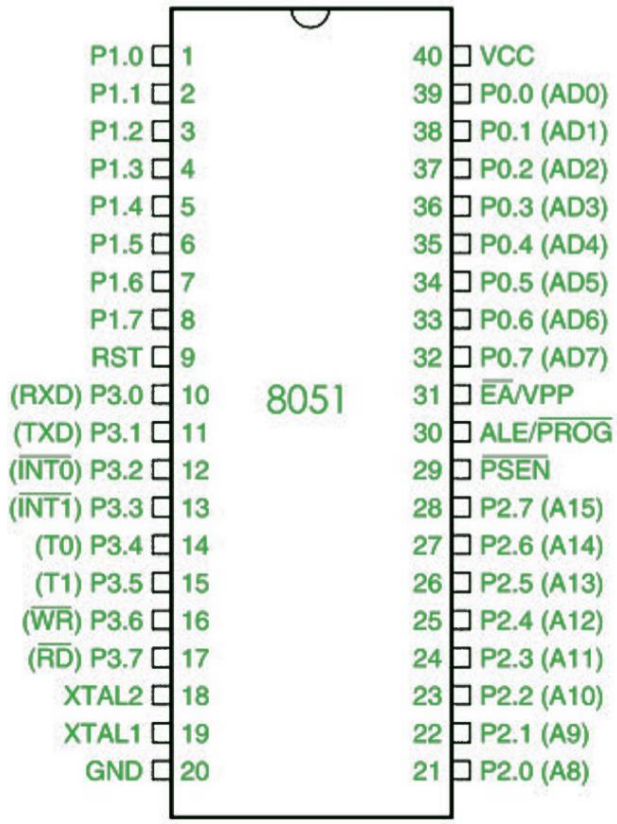
Basically, microcontrollers use **hardware delays** in which a physical device is used by the processor to produce the respective delay. And this physical device is known as a **timer**.
The timer produces the delay according to the demand of the processor and sends the signal to the processor once the respective delay gets produced.

1.2.7. Pin diagram of 8051 Microcontroller

Introduction :

The 8051 microcontroller is a popular 8-bit microcontroller widely used in embedded systems. It is a single-chip microcontroller with a Harvard architecture that includes a CPU, RAM, ROM, and several peripherals. The 8051 microcontroller has a 40-pin dual in-line package (DIP) that provides various inputs and outputs for communication with external devices.

8051 microcontroller is a 40 pin Dual Inline Package (DIP). These 40 pins serve different functions like read, write, I/O operations, interrupts etc. 8051 has four I/O ports wherein each port has 8 pins which can be configured as input or output depending upon the logic state of the pins. Therefore, 32 out of these 40 pins are dedicated to I/O ports. The rest of the pins are dedicated to VCC, GND, XTAL1, XTAL2, RST, ALE, EA' and PSEN'. Pin diagram of 8051



40 - PIN DIP

Description of the Pins :

- **Pin 1 to Pin 8 (Port 1) –** Pin 1 to Pin 8 are assigned to Port 1 for simple I/O operations. They can be configured as input or output pins depending on the logic control i.e. if logic zero (0) is applied to the I/O port it will act as an output pin and if logic one (1) is applied the pin will act as an input pin. These pins are also referred to as P1.0 to P1.7 (where P1 indicates that it is a pin in port 1 and the number after '.' tells the pin number i.e. 0 indicates first pin of the port. So, P1.0 means first pin of port 1, P1.1 means second pin of the port 1 and so on). These pins are bidirectional pins.
- **Pin 9 (RST) –** Reset pin. It is an active-high, input pin. Therefore if the RST pin is high for a minimum of 2 machine cycles, the microcontroller will reset i.e. it will close and terminate all activities. It is often referred as "power-on-reset" pin because it is used to reset the microcontroller to it's initial values when power is on (high).
- **Pin 10 to Pin 17 (Port 3) –** Pin 10 to pin 17 are port 3 pins which are also referred to as P3.0 to P3.7. These pins are similar to port 1 and can be used as universal input or output pins. These pins are bidirectional pins. These pins also have some additional functions which are as follows:
- **P3.0 (RXD) :** 10th pin is RXD (serial data receive pin) which is for serial input. Through this input signal microcontroller receives data for serial communication.
- **P3.1 (TXD) :** 11th pin is TXD (serial data transmit pin) which is serial output pin. Through this output signal microcontroller transmits data for serial communication.

- **P3.2 and P3.3 (INT0′, INT1′) :** 12th and 13th pins are for External Hardware Interrupt 0 and Interrupt 1 respectively. When this interrupt is activated(i.e. when it is low), 8051 gets interrupted in whatever it is doing and jumps to the vector value of the interrupt (0003H for INT0 and 0013H for INT1) and starts performing Interrupt Service Routine (ISR) from that vector location.
- **P3.4 and P3.5 (T0 and T1) :** 14th and 15th pin are for Timer 0 and Timer 1 external input. They can be connected with 16 bit timer/counter.
- **P3.6 (WR') :** 16th pin is for external memory write i.e. writing data to the external memory.
- **P3.7 (RD') :** 17th pin is for external memory read i.e. reading data from external memory.
- **Pin 18 and Pin 19 (XTAL2 And XTAL1) –** These pins are connected to an external oscillator which is generally a quartz crystal oscillator. They are used to provide an external clock frequency of 4MHz to 30MHz.
- **Pin 20 (GND) –** This pin is connected to the ground. It has to be provided with 0V power supply. Hence it is connected to the negative terminal of the power supply.
- **Pin 21 to Pin 28 (Port 2) –** Pin 21 to pin 28 are port 2 pins also referred to as P2.0 to P2.7. When additional external memory is interfaced with the 8051 microcontroller, pins of port 2 act as higher-order address bytes. These pins are bidirectional.
- **Pin 29 (PSEN) –** PSEN stands for Program Store Enable. It is output, active-low pin. This is used to read external memory. In 8031 based system where external ROM holds the program code, this pin is connected to the OE pin of the ROM.
- **Pin 30 (ALE/ PROG) –** ALE stands for Address Latch Enable. It is input, active-high pin. This pin is used to distinguish between memory chips when multiple memory chips are used. It is also used to de-multiplex the multiplexed address and data signals available at port 0. During flash programming i.e. Programming of EPROM, this pin acts as program pulse input (PROG).
- **Pin 31 (EA/ VPP) –** EA stands for External Access input. It is used to enable/disable external memory interfacing. In 8051, EA is connected to Vcc as it comes with on-chip ROM to store programs. For other family members such as 8031 and 8032 in which there is no on-chip ROM, the EA pin is connected to the GND.
- **Pin 32 to Pin 39 (Port 0) –** Pin 32 to pin 39 are port 0 pins also referred to as P0.0 to P0.7. They are bidirectional input/output pins. They don't have any internal pull-ups. Hence, 10 K? pull-up registers are used as external pull-ups. Port 0 is also designated as AD0-AD7 because 8051 multiplexes address and data through port 0 to save pins.
- **Pin 40 (VCC) –** This pin provides power supply voltage i.e. +5 Volts to the circuit.

Uses of pin diagram of the 8051 microcontroller :

The pin diagram of the 8051 microcontroller is used for various purposes in embedded systems. Some of the main uses of the pin diagram are:

1. **Interfacing with external devices:** The 8051 microcontroller has several input/output pins that can be used for interfacing with external devices such as sensors, actuators,

displays, and communication modules. The pin diagram provides the information about the location of these pins, their functionalities, and their electrical characteristics.

2. **Programming the microcontroller:** The 8051 microcontroller can be programmed using various programming languages such as Assembly, C, and BASIC. The pin diagram provides the information about the pins that are used for programming the microcontroller, such as the PSEN pin and the ALE pin.
3. **Debugging and testing:** The pin diagram provides access to the internal signals of the microcontroller, such as the address and data buses, which can be used for debugging and testing the microcontroller. Special hardware tools such as logic analyzers and oscilloscopes can be connected to the pins to monitor the signals and diagnose any issues in the system.
4. **Expansion and customization:** The pin diagram provides the flexibility to expand and customize the functionality of the microcontroller by connecting external devices and peripherals. For example, additional memory can be added by connecting external RAM or ROM chips to the address and data buses.

1.2.8. Characteristics of 8051 Microcontroller

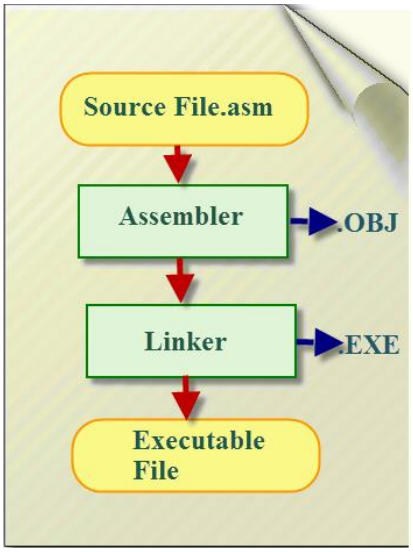
1. An 8-bit processor.
2. Data memory or RAM of 128 bytes.
3. Program memory or ROM of 4 KB.
4. 2 timers of 16 bit each.
5. 8-bit data bus.
6. 16-bit address bus.
7. Offers bit addressable format.
8. Special function registers and serial port.
9. 32 input/output lines.

1.3. 8051 Programming in Assembly Language

The assembly language is a low-level programming language used to write program code in terms of mnemonics. Even though there are many high-level languages that are currently in demand, assembly programming language is popularly used in many applications. It can be used for direct hardware manipulations. It is also used to write the 8051 programming code efficiently with less number of clock cycles by consuming less memory compared to the other high-level languages.

1.3.1. 8051 Programming in Assembly Language

The assembly language is a fully hardware related programming language. The embedded designers must have sufficient knowledge on hardware of particular processor or controllers before writing the program. The assembly language is developed by mnemonics; therefore, users cannot understand it easily to modify the program.



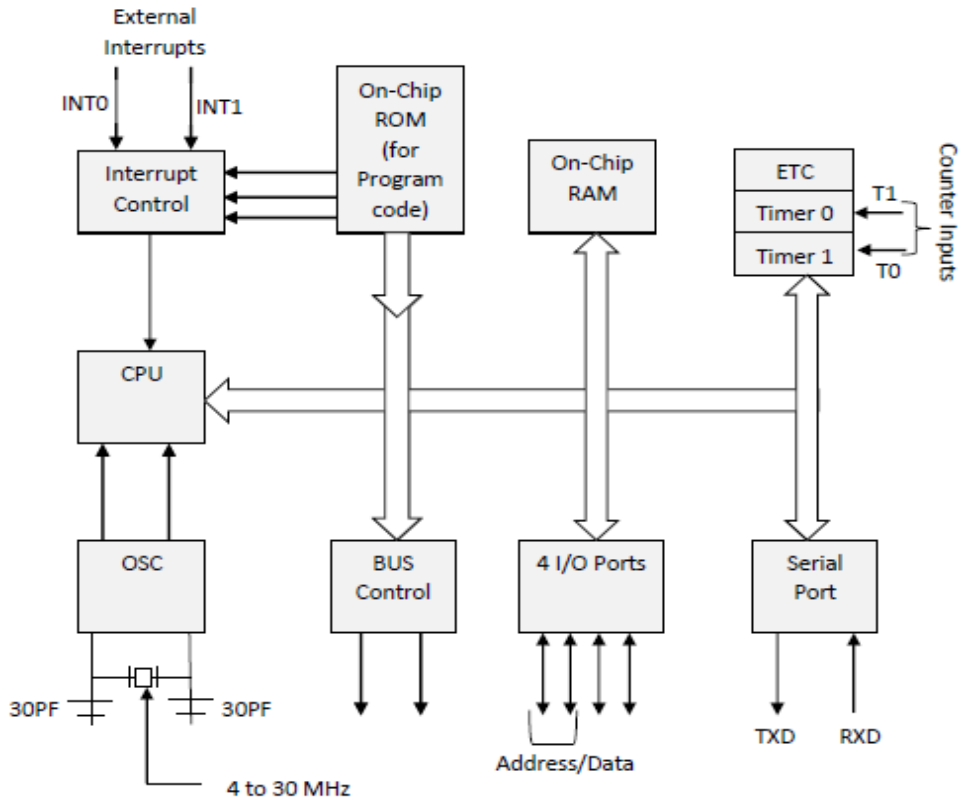
8051 Programming in Assembly Language

Assembly programming language is developed by various compilers and the “keiluvison” is best suitable for microcontroller programming development.

Microcontrollers or processors can understand only binary language in the form of ‘0s or 1s’; An assembler converts the assembly language to binary language, and then stores it in the microcontroller memory to perform the specific task.

1.3.2. 8051 Microcontroller Architecture

The 8051 microcontroller is the CISC based Harvard architecture, and it has peripherals like 32 I/O, timers/counters, serial communication and memories. The microcontroller requires a program to perform the operations that require a memory for saving and to read the functions. The 8051 microcontroller consists of RAM and ROM memories to store instructions.



8051 Microcontroller Architecuture

A Register is the main part in the processors and microcontrollers which is contained in the memory that provides a faster way of collecting and storing the data. The 8051 assembly language programming is based on the memory registers. If we want to manipulate data to a processor or controller by performing subtraction, addition, etc., we cannot do that directly in the memory, but it needs registers to process and to store the data. Microcontrollers contain several types of registers that can be classified according to their instructions or content that operate in them.

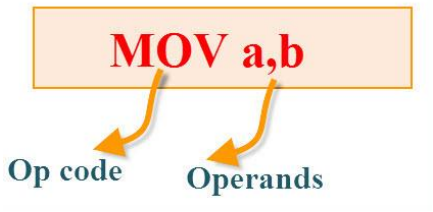
1.3.3. 8051 Microcontroller Programs in Assembly Language

The assembly language is made up of elements which all are used to write the program in sequential manner. Follow the given rules to write programming in assembly language.

1.3.3.1. Rules of Assembly Language

- The assembly code must be written in upper case letters
- The labels must be followed by a colon (label:)
- All symbols and labels must begin with a letter
- All comments are typed in lower case
- The last line of the program must be the END directive

The assembly language mnemonics are in the form of op-code, such as MOV, ADD, JMP, and so on, which are used to perform the operations.



Op-code: The op-code is a single instruction that can be executed by the CPU. Here the op-code is a MOV instruction.

Operands: The operands are a single piece of data that can be operated by the op-code. Example, multiplication operation is performed by the operands that are multiplied by the operand.

Syntax: MUL a,b;

1.3.4. The Elements of an Assembly Language Programming:

- Assembler Directives
- Instruction Set
- Addressing Modes

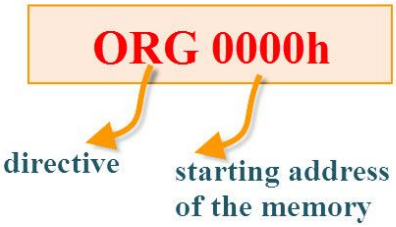
1.3.4.1. Assembler Directives:

The assembling directives give the directions to the CPU. The 8051 microcontroller consists of various kinds of assembly directives to give the direction to the control unit. The most useful directives are 8051 programming, such as:

- ORG
- DB
- EQU
- END

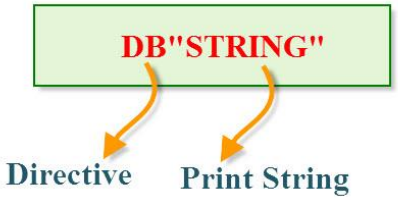
ORG(origin): This directive indicates the start of the program. This is used to set the register address during assembly. For example; ORG 0000h tells the compiler all subsequent code starting at address 0000h.

Syntax: ORG 0000h



(define byte): The define byte is used to allow a string of bytes. For example, print the "EDGEFX" wherein each character is taken by the address and finally prints the "string" by the DB directly with double quotes.

Syntax:
ORG 0000h
MOV a, #00h
————-
————-
DB"EDGEFX"



EQU (equivalent): The equivalent directive is used to equate address of the variable.

Syntax:
reg equ,09h
_____
_____
MOV reg,#2h

END: The END directive is used to indicate the end of the program.
Syntax:
reg equ,09h
_____
_____
MOV reg,#2h
END

1.3.4.2. Embedded Systems - Addressing Modes

An **addressing mode** refers to how you are addressing a given memory location. There are five different ways or five addressing modes to execute this instruction which are as follows –

- Immediate addressing mode
- Direct addressing mode
- Register direct addressing mode
- Register indirect addressing mode
- Indexed addressing mode

Immediate Addressing Mode

Let's begin with an example.

MOV A, #6AH

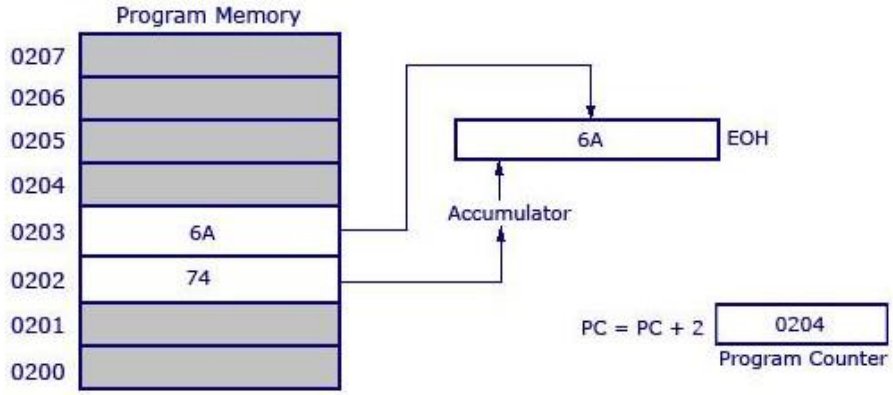
In general, we can write,

MOV A, #data

It is termed as **immediate** because 8-bit data is transferred immediately to the accumulator (destination operand).

Immediate Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1



The above instruction and its execution. The opcode 74H is saved at 0202 address. The data 6AH is saved at 0203 address in the program memory. After reading the opcode 74H, the data at the next program memory address is transferred to accumulator A (E0H is the address of accumulator). Since the instruction is of 2-bytes and is executed in one cycle, the program counter will be incremented by 2 and will point to 0204 of the program memory.

Note − The '#' symbol before 6AH indicates that the operand is a data (8 bit). In the absence of '#', the hexadecimal number would be taken as an address.

Direct Addressing Mode

This is another way of addressing an operand. Here, the address of the data (source data) is given as an operand. Let’s take an example.

MOV A, 04H

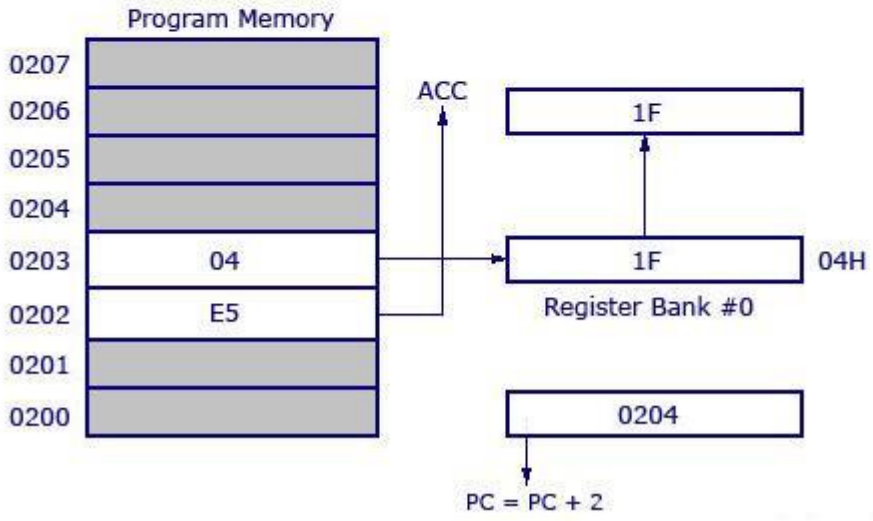
The register bank#0 (4th register) has the address 04H. When the MOV instruction is executed, the data stored in register 04H is moved to the accumulator. As the register 04H holds the data 1FH, 1FH is moved to the accumulator.

Note − We have not used '#' in direct addressing mode, unlike immediate mode. If we had used '#', the data value 04H would have been transferred to the accumulator instead of 1FH.

Now, take a look at the following illustration. It shows how the instruction gets executed.

Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #04H	E5	2	1



As shown in the above illustration, this is a 2-byte instruction which requires 1 cycle to complete. The PC will be incremented by 2 and will point to 0204. The opcode for the instruction MOV A, address is E5H. When the instruction at 0202 is executed (E5H), the accumulator is made active and ready to receive data. Then the PC goes to the next address as 0203 and looks up the address of the location of 04H where the source data (to be transferred to accumulator) is located. At 04H, the control finds the data 1F and transfers it to the accumulator and hence the execution is completed.

Register Direct Addressing Mode

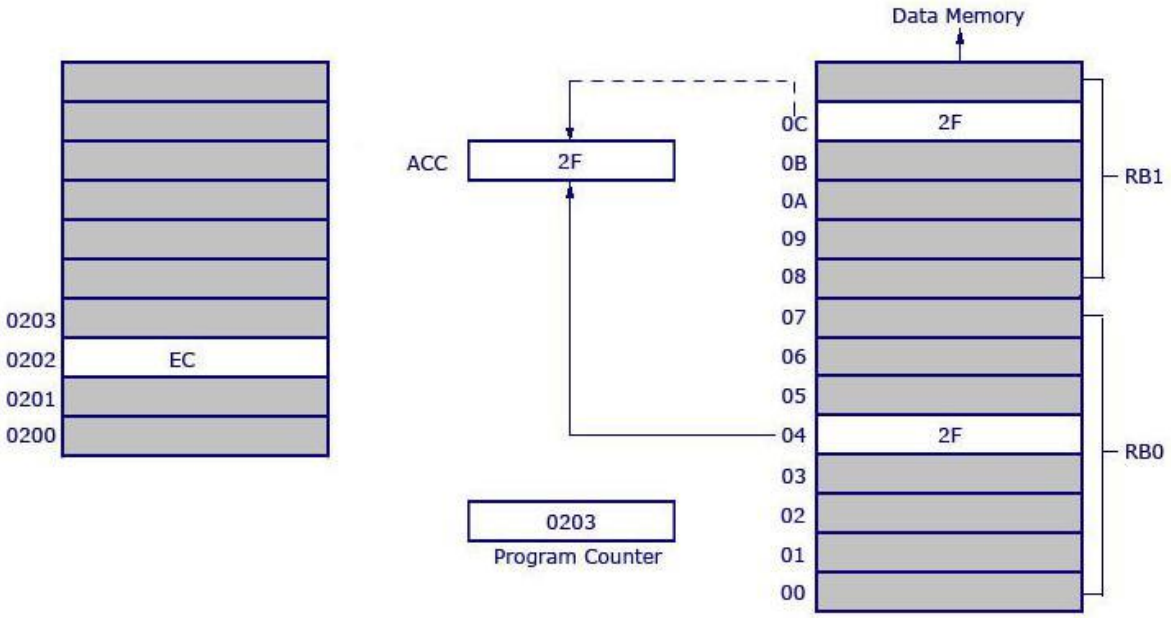
In this addressing mode, we use the register name directly (as source operand). Let us try to understand with the help of an example.

MOV A, R4

At a time, the registers can take values from R0 to R7. There are 32 such registers. In order to use 32 registers with just 8 variables to address registers, register banks are used. There are 4 register banks named from 0 to 3. Each bank comprises of 8 registers named from R0 to R7.

Register Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



At a time, a single register bank can be selected. Selection of a register bank is made possible through a **Special Function Register** (SFR) named **Processor Status Word** (PSW). PSW is an 8-bit SFR where each bit can be programmed as required. Bits are designated from PSW.0 to PSW.7. PSW.3 and PSW.4 are used to select register banks.

Now, take a look at the following illustration to get a clear understanding of how it works.

Opcode EC is used for MOV A, R4. The opcode is stored at the address 0202 and when it is executed, the control goes directly to R4 of the respected register bank (that is selected in PSW). If register bank #0 is selected, then the data from R4 of register bank #0 will be moved to the accumulator. Here 2F is stored at 04H. 04H represents the address of R4 of register bank #0.

Data (2F) movement is highlighted in bold. 2F is getting transferred to the accumulator from data memory location 0C H and is shown as dotted line. 0CH is the address location of Register 4 (R4) of register bank #1. The instruction above is 1 byte and requires 1 cycle for complete execution. What it means is, you can save program memory by using register direct addressing mode.

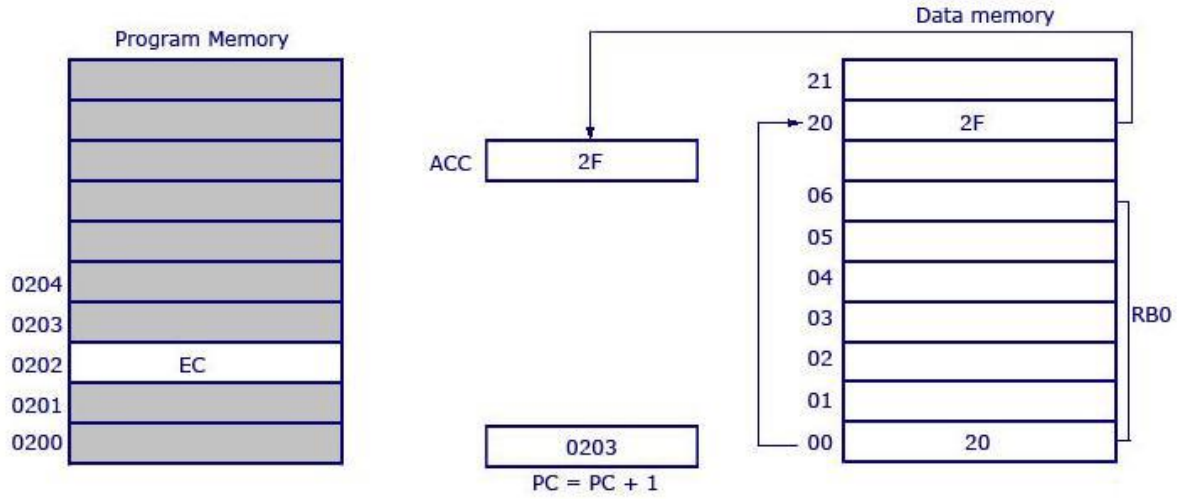
Register Indirect Addressing Mode

In this addressing mode, the address of the data is stored in the register as operand.

MOV A, @R0

Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



Here the value inside R0 is considered as an address, which holds the data to be transferred to the accumulator. **Example**: If R0 has the value 20H, and data 2FH is stored at the address 20H, then the value 2FH will get transferred to the accumulator after executing this instruction. See the following illustration.

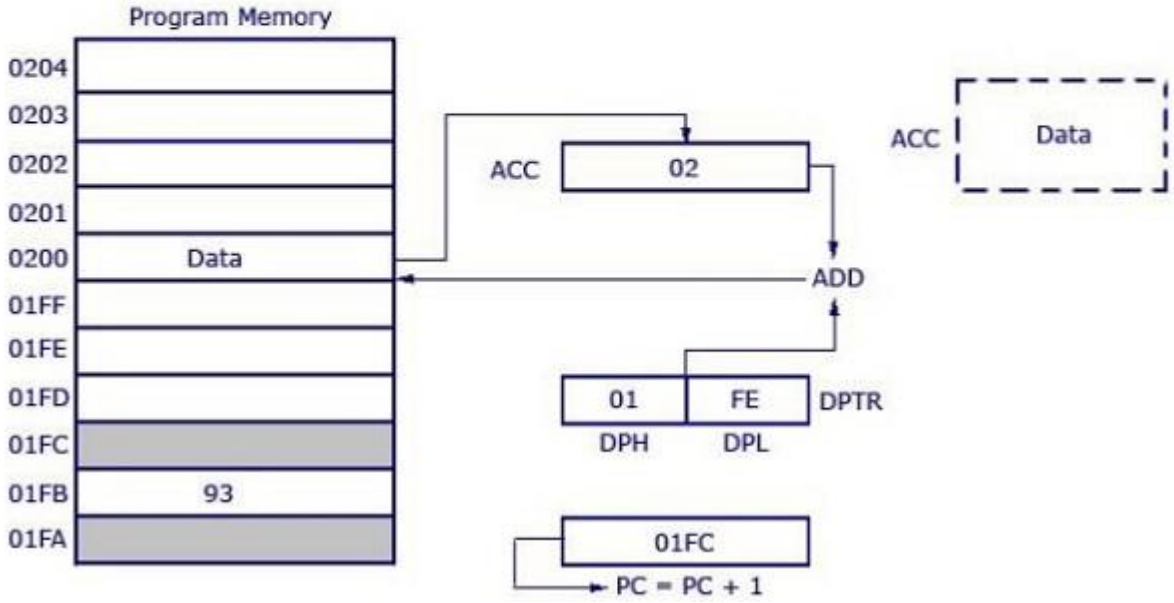
So the opcode for **MOV A, @R0** is E6H. Assuming that the register bank #0 is selected, the R0 of register bank #0 holds the data 20H. Program control moves to 20H where it locates the data 2FH and it transfers 2FH to the accumulator. This is a 1-byte instruction and the program counter increments by 1 and moves to 0203 of the program memory.

Note − Only R0 and R1 are allowed to form a register indirect addressing instruction. In other words, the programmer can create an instruction either using @R0 or @R1. All register banks are allowed.

Indexed Addressing Mode

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2



We will take two examples to understand the concept of indexed addressing mode. Take a look at the following instructions −

MOVC A, @A+DPTR

and

MOVC A, @A+PC

where DPTR is the data pointer and PC is the program counter (both are 16-bit registers). Consider the first example.

MOVC A, @A+DPTR

The source operand is @A+DPTR. It contains the source data from this location. Here we are adding the contents of DPTR with the current content of the accumulator. This addition will give a new address which is the address of the source data. The data pointed by this address is then transferred to the accumulator.

The opcode is 93H. DPTR has the value 01FE, where 01 is located in DPH (higher 8 bits) and FE is located in DPL (lower 8 bits). Accumulator has the value 02H. Then a 16-bit addition is performed and 01FE H+02H results in 0200 H. Data at the location 0200H will get transferred to the accumulator. The previous value inside the accumulator (02H) will be replaced with the new data from 0200H. The new data in the accumulator is highlighted in the illustration.

This is a 1-byte instruction with 2 cycles needed for execution and the execution time required for this instruction is high compared to previous instructions (which were all 1 cycle each).

The other example **MOVC A, @A+PC** works the same way as the above example. Instead of adding DPTR with the accumulator, here the data inside the program counter (PC) is added with the accumulator to obtain the target address.

1.3.4.3. Instruction Set

In the sequence of instructions to be executed, it is often necessary to transfer program control to a different location. There are many instructions in the 8051 to achieve this. This chapter covers the control transfer instructions available in 8051 Assembly language. In the first section we discuss instructions used for looping, as well as instructions for conditional and unconditional jumps. In the second section we examine CALL instructions and their uses. In the third section, time delay subroutines are described for both the traditional 8051 and its newer generation.

Looping in the 8051

Repeating a sequence of instructions a certain number of times is called a *loop*. The loop is one of most widely used actions that any microprocessor performs. In the 8051, the loop action is performed by the instruction "DJNZ reg, label". In this instruction, the register is decremented; if it is not zero, it jumps to the target address referred to by the label. Prior to the start of the loop the register is loaded with the counter for the number of repetitions. Notice that in this instruction both the register decrement and the decision to jump are combined into a single instruction.

Write a program to
(a) clear ACC, then
(b) add 3 to the accumulator ten times.

Solution:

```
;This program adds value 3 to the ACC ten times

          MOV  A,#0       ;A=0, clear ACC
          MOV  R2,#10     ;load counter R2=10
AGAIN:    ADD  A,#03      ;add 03 to ACC
          DJNZ R2,AGAIN   ;repeat until R2=0(10 times)
          MOV  R5,A       ;save A in R5
```

In the program in Example, the R2 register is used as a counter. The counter is first set to 10. In each iteration the instruction DJNZ decrements R2 and checks its value. If R2 is not zero, it jumps to the target address associated with the label "AGAIN". This looping action continues until R2 becomes zero. After R2 becomes zero, it falls through the loop and executes the instruction immediately below it, in this case the "MOV R5 , A" instruction. Notice in the DJNZ instruction that the registers can be any of RO – R7. The counter can also be a RAM location

Loop inside a loop

As shown in Example the maximum count is 256. What happens if we want to repeat an action more times than 256? To do that, we use a loop inside a loop, which is called a *nested loop*. In a nested loop, we use two registers to hold the count.

Example

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times.

Solution:

Since 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use R2 and R3 for the count.

```
            MOV  A,#55H
            MOV  R3,#10
NEXT:       MOV  R2,#70
AGAIN:      CPL  A
            DJNZ R2,AGAIN
            DJNZ R3,NEXT

;A=55H
;R3=10, the outer loop count
;R2=70, the inner loop count
;complement A register
;repeat it 70 times (inner loop)
```

In this program, R2 is used to keep the inner loop count. In the instruction "DJNZ R2 , AGAIN", whenever R2 becomes 0 it falls through and "DJNZ R3 , NEXT" is executed. This instruction forces the CPU to load R2 with the count 70 and the inner loop starts again. This process will continue until R3 becomes zero and the outer loop is finished.

Other conditional jumps

Conditional jumps for the 8051 are summarized in Table 3-1. More details of each instruction are provided in Appendix A. In Table 3-1, notice that some of the instructions, such as JZ (jump if A = zero) and JC (jump if carry), jump only if a certain condition is met. Next we examine some conditional jump instructions with examples.

JZ (jump if A = 0)

In this instruction the content of register A is checked. If it is zero, it jumps to the target address. For example, look at the following code.

```
        ;A=R0
        ;jump if A = 0
        ;A=R1
        ;jump if A = 0

MOV   A,R0
JZ    OVER
MOV   A,R1
JZ    OVER
```

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and jump if register ≠ 0
CJNE A, data	Jump if A ≠ data
CJNE reg, #data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

In this program,. if either RO or Rl is zero, it jumps to the label OVER. Notice that the JZ instruction can be used only for register A. It can only check to see whether the accumulator is zero, and it does not apply to any other register. More importantly, you don't have to perform an arithmetic instruction such as decrement to use the JNZ instruction. See Example 3-4.

Example

Write a program to determine if R5 contains the value 0. If so, put 55H in it.

Solution:

```
        MOV   A,R5      ;copy R5 to A
        JNZ   NEXT      ;jump if A is not zero
        MOV   R5,#55H
NEXT:   ...
```

JNC (jump if no carry, jumps if CY = 0)

In this instruction, the carry flag bit in the flag (PSW) register is used to make the decision whether to jump. In executing "JNC label", the processor looks at the carry flag to see if it is raised (CY =1). If it is not, the CPU starts to fetch and execute instructions from the address of the label. If CY = 1, it will not jump but will execute the next instruction below JNC.

Note that there is also a "JC label" instruction. In the JC instruction, if CY = 1 it jumps to the target address. We will give more examples of these instructions in the context of applications in future chapters.

There are also JB (jump if bit is high) and JNB (jump if bit is low) instructions. These are discussed in Chapters 4 and 8 when bit manipulation instructions are discussed.

All conditional jumps are short jumps

It must be noted that all conditional jumps are short jumps, meaning that the address of the target must be within -128 to +127 bytes of the contents of the program counter (PC). This very important concept is discussed at the end of this section.

Unconditional jump instructions

The unconditional jump is a jump in which control is transferred unconditionally to the target location. In the 8051 there are two unconditional jumps: LJMP (long jump) and SJMP (short jump). Each is discussed below.

LJMP (long jump)

LJMP is an unconditional long jump. It is a 3-byte instruction in which the first byte is the opcode, and the second and third bytes represent the 16-bit address of the target location. The 2-byte target address allows a jump to any memory location from 0000 to FFFFH.

Remember that although the program counter in the 8051 is 16-bit, thereby giving a ROM address space of 64K bytes, not all 8051 family members have that much on-chip program ROM. The original 8051 had only 4K bytes of on-chip ROM for program space; consequently, every byte was precious. For this reason there is also an SJMP (short jump) instruction, which is a 2-byte instruction as opposed to the 3-byte LJMP instruction. This can save some bytes of memory in many applications where memory space is in short supply. SJMP is discussed next.

SJMP (short jump)

In this 2-byte instruction, the first byte is the opcode and the second byte is the relative address of the target location. The relative address range of 00 – FFH is divided into forward and backward jumps; that is, within -128 to +127 bytes of memory relative to the address of the current PC (program counter). If the jump is forward, the target address can be within a space of 127 bytes from the current PC. If the target address is backward, the target address can be within -128 bytes from the current PC. This is explained in detail next.

Calculating the short jump address

In addition to the SJMP instruction, all conditional jumps such as JNC, JZ, and DJNZ are also short jumps due to the fact that they are all two-byte instructions. In these instructions the first byte is the opcode and the second byte is the relative address. The target address is relative to the value of the program counter. To calculate the target address, the second byte is added to the PC of the instruction immediately below the jump.

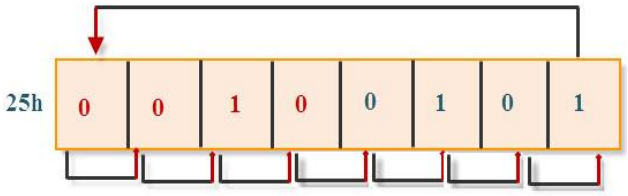
Shifting Operators

The shift operators are used for sending and receiving the data efficiently. The 8051 microcontroller consist four shift operators:

- RR —> Rotate Right
- RRC —>Rotate Right through carry
- RL —> Rotate Left
- RLC —>Rotate Left through carry

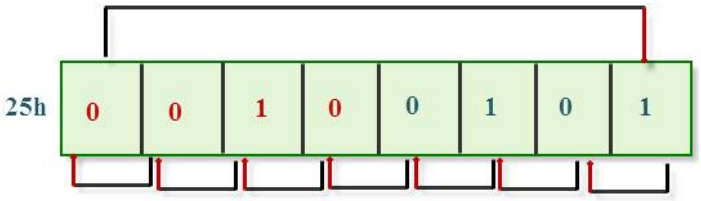
Rotate Right (RR):
In this shifting operation, the MSB becomes LSB and all bits shift towards right side bit-by-bit, serially.

Syntax:
MOV A, #25h
RR A



Rotate Left (RL):
In this shifting operation, the MSB becomes LSB and all bits shift towards Left side bit-by-bit, serially.

Syntax:
MOV A, #25h
RL A



RRC Rotate Right through Carry:
In this shifting operation, the LSB moves to carry and the carry becomes MSB, and all the bits are shift towards right side bit by bit position.

Syntax:
MOV A, #27h
RRC A

RLC Rotate Left through Carry:
In this shifting operation, the MSB moves to carry and the carry becomes LSB and all the bits shift towards left side in a bit-by-bit position.

Syntax:
MOV A, #27h
RLC A

Basic Embedded C Programs:

The microcontroller programming differs for each type of operating system. There are many operating systems such as Linux, Windows, RTOS and so on. However, RTOS has several advantages for embedded system development. Some of the Assembly levels programming examples are given below.

1.3.5. Example Program for LED blinking using with 8051 microcontroller:

- Number Displaying on 7-segment display using 8051 microcontroller
- Timer/Counter calculations and program using 8051 microcontroller
- Serial Communication calculations and program using 8051 microcontroller

LED programs with 8051 Microcontrller

1. WAP to toggle the PORT1 LEDs

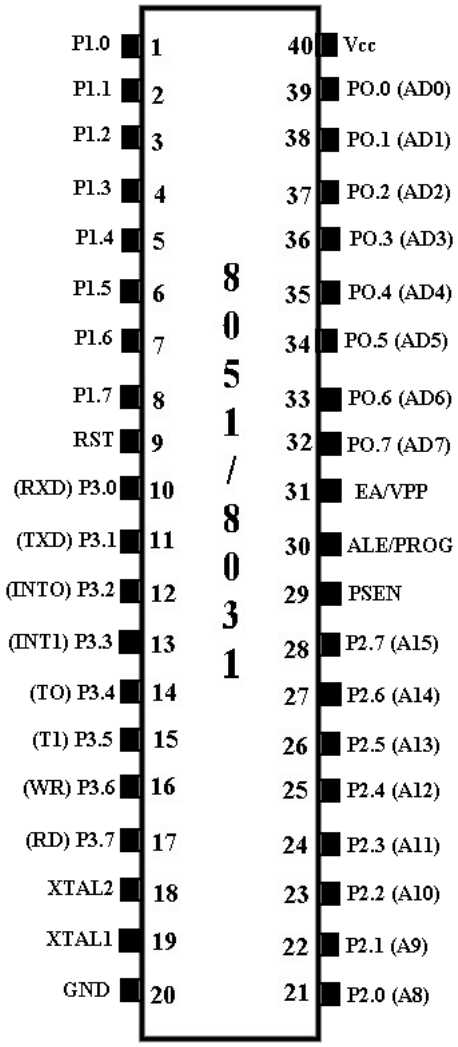
```
ORG 0000H
TOGLE: MOV P1, #01    //move 00000001 to the p1 register//
CALL DELAY    //execute the delay//
MOV A, P1       //move p1 value to the accumulator//
CPL A       //complement A value //
MOV P1, A       //move 11111110 to the port1 register//
CALL DELAY    //execute the delay//
SJMP TOGLE
DELAY: MOV R5, #10H    //load register R5 with 10//
TWO:       MOV R6, #200    //load register R6 with 200//
ONE:      MOV R7, #200    //load register R7 with 200//
DJNZ R7, $    //decrement R7 till it is zero//
DJNZ R6, ONE    //decrement R7 till it is zero//
DJNZ R5, TWO    //decrement R7 till it is zero//
RET          //go back to the main program //
END
```

1.4. Programming Parallel port

1.4.1. 8051 Microcontroller port programming

There are four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output, ready to be used as output ports. To use any of these ports as an input port, it must be programmed.

1.4.2. Pin configuration of 8051/8031 microcontroller.

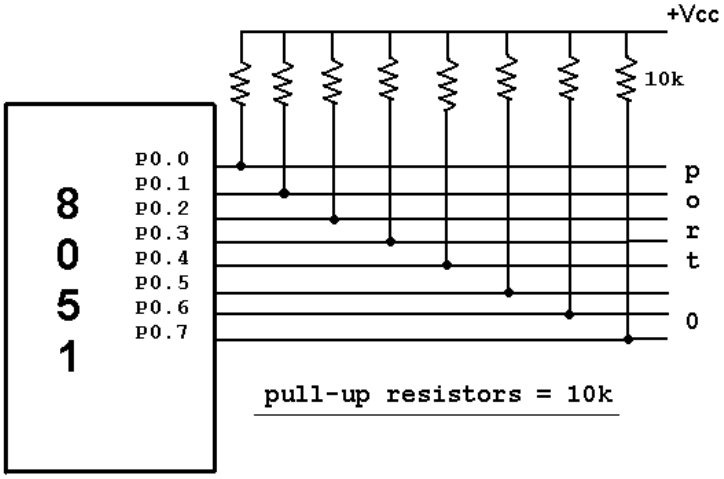


Pin configuration of 8951

Port 0: Port 0 occupies a total of 8 pins (pins 32-39) .It can be used for input or output. To use the pins of port 0 as both input and output ports, each pin must be connected externally to a 10K ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3.Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips. With external pull-up resistors connected upon reset, port 0 is configured as an output port. For example, the following code will continuously send out to port 0 the alternating values 55H and AAH

```
MOV A,#55H
BACK:  MOV P0,A
ACALL DELAY
CPL A
SJMP BACK
```

Port 0 as Input : With resistors connected to port 0, in order to make it an input, the port must be programmed by writing 1 to all the bits. In the following code, port 0 is configured first as an input port by writing 1's to it, and then data is received from the port and sent to P1.



8051 I/O Ports

```
MOV A,#0FFH       ; A = FF hex
MOV P0,A          ; make P0 an input port
BACK: MOV A,P0    ;get data from P0
MOV P1,A          ;send it to port 1
SJMP BACK
```

Dual role of port 0: Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data. When connecting an 8051/31 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save pins. ALE indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE =1 it has address and data with the help of a 74LS373 latch.

Port 1: Port 1 occupies a total of 8 pins (pins 1 through 8). It can be used as input or output. In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, Port 1 is configured as an output port. For example, the following code will continuously send out to port1 the alternating values 55h & AAh

```
MOV A,#55H          ; A = 55 hex
BACK: MOV P1,A      ;send it to Port 1
ACALL DELAY         ;call delay routine
CPL A               ;make A=0
SJMP BACK
```

Port 1 as input: To make port1 an input port, it must programmed as such by writing 1 to all its bits. In the following code port1 is configured first as an input port by writing 1's to it, then data is received from the port and saved in R7 ,R6 & R5.

```
MOV A,#0FFH   ;A=FF HEX
MOV P1,A      ;make P1 an input port by writing all 1's to it
MOV A,P1      ;get data from P1
```

```
MOV R7,A        ;save it in register R7
ACALL DELAY   ;wait
MOV  A,P1       ;get another data from P1
MOV R6,A        ;save it in register R6
ACALL DELAY   ;wait
MOV  A,P1       ;get another data from P1
MOV R5,A        ;save it in register R5
```

Port 2 : Port 2 occupies a total of 8 pins (pins 21- 28). It can be used as input or output. Just like P1, P2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset,Port 2 is configured as an output port. For example, the following code will send out continuously to port 2 the alternating values 55h and AAH. That is all the bits of port 2 toggle continuously.

```
MOV A,#55H              ; A = 55 hex
BACK:  MOV P2,A         ;send it to Port 2
ACALL DELAY             ;call delay routine
CPL A                   ;make A=0
SJMP BACK
```

Port 2 as input : To make port 2 an input, it must programme as such by writing 1 to all its bits. In the following code, port 2 is configured first as an input port by writing 1's to it. Then data is received from that port and is sent to P1 continuously.

```
MOV A,#0FFH       ;A=FF hex
MOV P2,A          ;make P2 an input port by writing all 1's to it
BACK: MOV A,P2    ;get data from P2
MOV P1,A          ;send it to Port1
SJMP BACK         ;keep doing that
```

Dual role of port 2 : In systems based on the 8751, 8951, and DS5000, P2 is used as simple I/O. However, in 8031-based systems, port 2 must be used along with P0 to provide the 16-bit address for the external memory. As shown in pin configuration 8051, port 2 is also designed as A8-A15, indicating the dual function. Since an 8031 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address. In other words, when 8031 is connected to external memory, P2 is used for the upper 8 bits of the 16 bit address, and it cannot be used for I/O.

Port 3 : Port 3 occupies a total of 8 pins, pins 10 through 17. It can be used as input or output. P3 does not need any pull-up resistors, the same as P1 and P2 did not. Although port 3 is configured as an output port upon reset. Port 3 has the additional function of providing some extremely important signals such as interrupts. This information applies both 8051 and 8031 chips.

Read-modify-write feature : The ports in the 8051 can be accessed by the read-modify-write technique. This feature saves many lines of code by combining in a single instruction all three action of (1) reading the port, (2) modifying it, and (3) writing to the port. The following code first places 01010101 (binary) into port 1. Next, the instruction "XLR P1,#0FFH" performs an XOR logic operation on P1 with 1111 1111 (binary), and then writes the result back into P1.

```
MOV P1,#55H      ;P1=01010101
AGAIN: XLR P1,#0FFH     ;EX-OR P1 with 1111 1111
ACALL DELAY
SJMP AGAIN
```

1.4.3. Addition of two 8-bit numbers in 8051 Microcontroller Using PortsIntroduction:

To perform addition of two 8-bit numbers using ports in 8051 microcontroller, we need to connect the two 8-bit numbers to be added to two ports of the microcontroller. We can use any two ports of the microcontroller, for example, P1 and P2.

The first step is to load the two numbers into two different ports. For example, we can load the first number into port P1 and the second number into port P2. We can use the MOV instruction to load the numbers into the ports.

Once the numbers are loaded into the ports, we can use the ADD instruction to add the two numbers. The ADD instruction adds the contents of the accumulator and the specified operand and stores the result in the accumulator. Since the two numbers are already loaded into the ports, we can simply use the ADD instruction with the accumulator and the appropriate port.

After the addition is complete, we can retrieve the result from the accumulator and store it in another port or memory location for further processing or display.8051 microcontroller is a microcontroller designed by Intel in 1981. It is an 8-bit microcontroller with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 16-bit timers. It consists of four parallel 8-bit ports, which are programmable as well as addressable as per the requirement.

Issues in Addition of two 8-bit numbers 8051 Microcontroller Using Ports :

There are several issues that can arise when performing addition of two 8-bit numbers in 8051 microcontroller using ports:

1. Overflow: If the result of the addition exceeds 8 bits, the carry flag (CY) in the program status word (PSW) will be set. If this flag is not checked before storing the result, the result may be incorrect.
2. Input validation: Before performing the addition, it is important to validate the input to ensure that the numbers are within the range of 0 to 255. If the input is not validated, the result may be incorrect.
3. Port initialization: The ports used for input and output must be properly initialized before use. If the ports are not properly initialized, the data may not be transferred correctly.
4. Endianness: The order in which the bytes of the numbers are stored in memory can affect the result of the addition. It is important to ensure that the bytes are stored in the correct order before performing the addition.
5. Interrupts: If interrupts are enabled during the addition operation, the result may be affected. It is important to disable interrupts during critical operations to ensure the correct result.

6. Timing: The timing of the addition operation can affect the result. It is important to ensure that the necessary delays are added between instructions to ensure correct operation.
7. Code optimization: The code used to perform the addition should be optimized to ensure that it uses the least number of instructions and takes the least amount of time. This is important to avoid potential timing issues and to ensure that the microcontroller can perform other tasks while the addition is being performed.

Problem: To write an assembly language program to add two 8 bit numbers in 8051 microcontroller using ports.

Example:

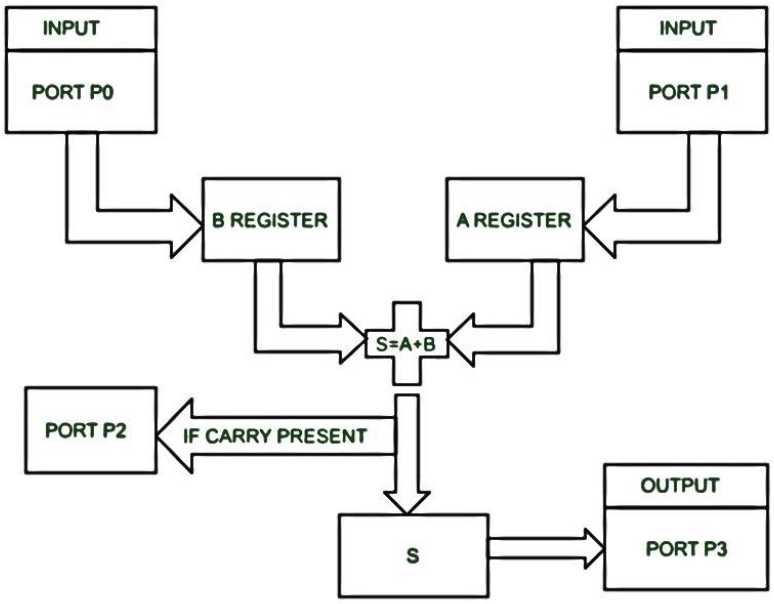
WITH CARRY

	INPUT PORTS		OUTPUT PORTS(SUM)	
PORT	PORT 0	PORT 1	PORT 2 (CARRY)	PORT 3
DATA	E7	F6	01	DD

WITHOUT CARRY

	INPUT PORTS		OUTPUT PORTS(SUM)	
PORT	PORT 0	PORT 1	PORT 2(CARRY)	PORT 3
DATA	01	02	00	03

Block diagram:



Algorithm:

- Initialize Ports P0 and P1 as input ports.
- Initialize Ports P2 and P3 as output ports.
- Initialize the R1 register.

- Move the contents from Port 0 to B register.
- Move the contents from Port 1 to A register.
- Add contents in A and B.
- If carry is present increment R1.
- Move contents in R1 to Port 2.
- Move the sum in step 6 to Port 3.

Program:

```
ORG 00H                 // Indicates starting address

MOV P0,#0FFH            // Initializes P0 as input port
MOV P1,#0FFH            // Initializes P1 as input port
MOV P2,#00H             // Initializes P2 as output port
MOV P3,#00H             // Initializes P3 as output port

L1:MOV R1, #00H         // Initializes Register R1
MOV B,P0                // Moves content of P0 to B
MOV A,P1                // Moves content of P1 to A
CLR C                   // Clears carry flag
ADD A,B                 // Add the content of A and B and store result in A
JNC L2                  // If carry is not set, jump to label L2
INC R1                  // Increment Register R1 if carry present

L2: MOV P2, R1          // Moves the content from Register R1 to Port2
MOV P3,A                // Moves the content from A to Port3
SJMP L1                 // Jumps to label L1
END
```

Explanation:

- ORG 00H is the starting address of the program.
- Giving the values as #0FFH and #00H initializes the ports as input and output ports respectively.
- R1 register is initialized to 0 so as to store any carry produced during the sum.
- MOV B, P0 moves the value present in P0 to the B register.
- MOV A, P1 moves the value present in P1 to Accumulator.
- ADD AB adds the values present in Accumulator and B register and stores the result in Accumulator.
- JNC L2 refers to jump to label L2 if no carry is present by automatically checking whether the carry bit is set or not.
- If the carry bit is set to increment register R1.
- MOV P2, R1, and MOV P3, A refers to moving the carry bit to P2 and result in Accumulator to P3.

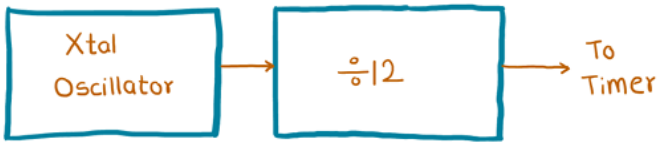
1.5. 8051 Timers

1.5.1. Introduction to 8051 Timers

8051 microcontrollers have two timers and counters which work on the clock frequency. Timer/counter can be used for time delay generation, counting external events, etc.

8051 Clock

Every Timer needs a clock to work, and 8051 provides it from an external crystal which is the main clock source for Timer. The internal circuitry in the 8051 microcontrollers provides a clock source to the timers which is 1/12th of the frequency of crystal attached to the microcontroller, also called Machine cycle frequency.



8051 Timer Clock

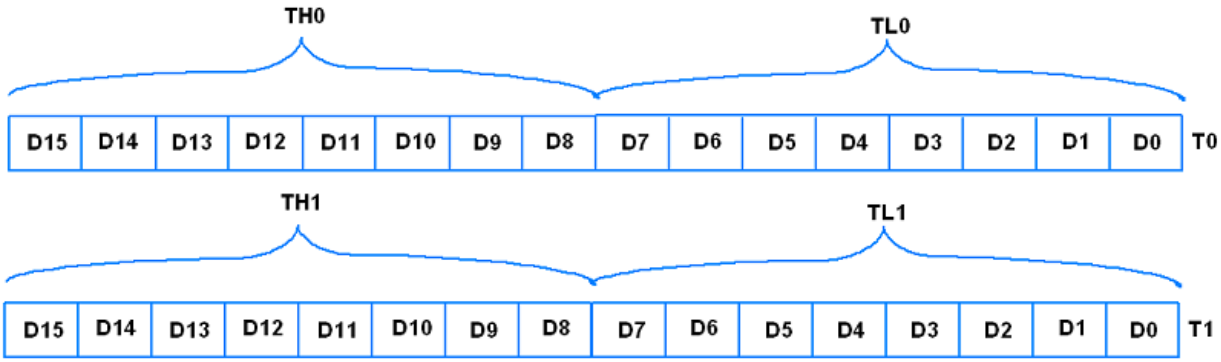
For example, suppose we have a crystal frequency of 11.0592 MHz then the microcontroller will provide 1/12th i.e.

Timer clock frequency= (Xtal Osc.frequency)/12 = (11.0592 MHz)/12 = 921.6 KHz

period T= 1/(921.6 kHz)=1.085 µS

1.5.2. 8051 Timer

8051 has two timers Timer0 (T0) and Timer1 (T1), both are 16-bit wide. Since 8051 has 8-bit architecture, each of these is accessed by two separate 8-bit registers as shown in the figure below. These registers are used to load timer count.

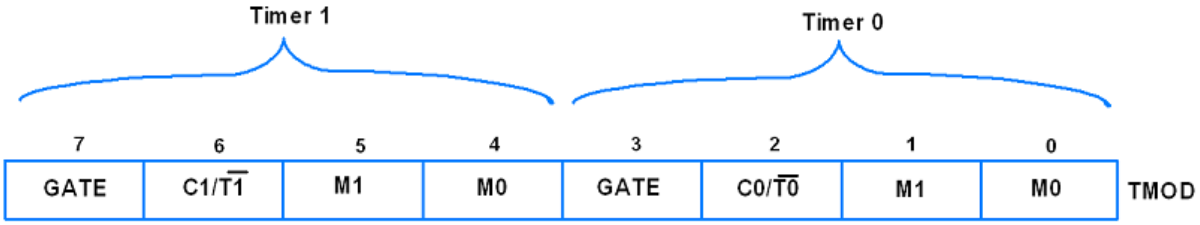


8051 has a Timer Mode Register and Timer Control Register for selecting a mode of operation and controlling purpose.

Let's see these registers,

1.5.2.1.TMOD register

TMOD is an 8-bit register used to set timer mode of timer0 and timer1.



Its lower 4 bits are used for Timer0 and the upper 4 bits are used for Timer1

Bit 7,3 – GATE:

1 = Enable Timer/Counter only when the INT0/INT1 pin is high and TR0/TR1 is set.

0 = Enable Timer/Counter when TR0/TR1 is set.

Bit 6,2 - C/$\overline{T}$ (Counter/Timer): Timer or Counter select bit

1 = Use as Counter

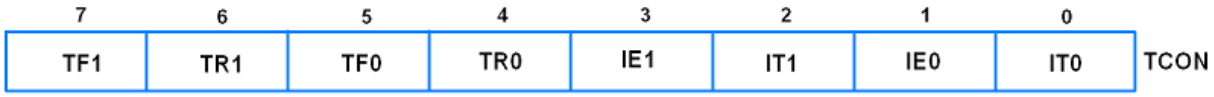
0 = Use as Timer

Bit 5:4 & 1:0 - M1:M0: Timer/Counter mode select bit

These are Timer/Counter mode select bit as per the below table

M1	M0	Mode	Operation
0	0	0 (13-bit timer mode)	13-bit timer/counter, 8-bit of THx & 5-bit of TLx
0	1	1 (16-bit timer mode)	16-bit timer/counter, THx cascaded with TLx
1	0	2 (8-bit auto-reload mode)	8-bit timer/counter (auto-reload mode), TLx reload with the value held by THx each time TLx overflow
1	1	3 (split timer mode)	Split the 16-bit timer into two 8-bit timers i.e. THx and TLx like two 8-bit timer

1.5.2.2.TCON Register



TCON is an 8-bit control register and contains a timer and interrupt flags.

Bit 7 - TF1: Timer1 Overflow Flag

1 = Timer1 overflow occurred (i.e. Timer1 goes to its max and roll over back to zero).

0 = Timer1 overflow not occurred.

It is cleared through software. In the Timer1 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 6 - TR1: Timer1 Run Control Bit

1 = Timer1 start.

0 = Timer1 stop.

It is set and cleared by software.

Bit 5 – TF0: Timer0 Overflow Flag

1 = Timer0 overflow occurred (i.e. Timer0 goes to its max and roll over back to zero).

0 = Timer0 overflow not occurred.

It is cleared through software. In the Timer0 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 4 – TR0: Timer0 Run Control Bit

1 = Timer0 start.

0 = Timer0 stop.

It is set and cleared by software.

Bit 3 - IE1: External Interrupt1 Edge Flag

1 = External interrupt1 occurred.

0 = External interrupt1 Processed.

It is set and cleared by hardware.

Bit 2 - IT1: External Interrupt1 Trigger Type Select Bit

1 = Interrupt occurs on falling edge at INT1 pin.

0 = Interrupt occur on a low level at the INT1 pin.

Bit 1 – IE0: External Interrupt0 Edge Flag

1 = External interrupt0 occurred.

0 = External interrupt0 Processed.

It is set and cleared by hardware.

Bit 0 – IT0: External Interrupt0 Trigger Type Select Bit

1 = Interrupt occurs on falling edge at INT0 pin.

0 = Interrupt occur on a low level at INT0 pin.

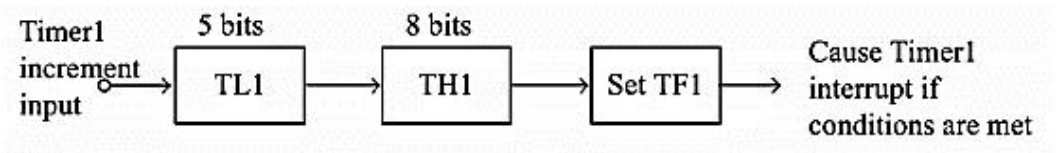
Let's see the timers modes

1.5.3. 8051 Timer Modes

Timers have their operation modes which are selected in the TMOD register using M0 & M1 bit combinations.

Mode 0 (13-bit timer mode)

The Mode 0 operation is the 8-bit timer or counter with a 5-bit pre-scaler. So it is a 13-bit timer/counter. It uses 5 bits of TL0 or TL1 and all of the 8-bits of TH0 or TH1.



In this example the Timer1is selected, in this case, every 32 (25)event for counter operations or 32 machine cycles for timer operation, the TH1 register will be incremented by 1. When

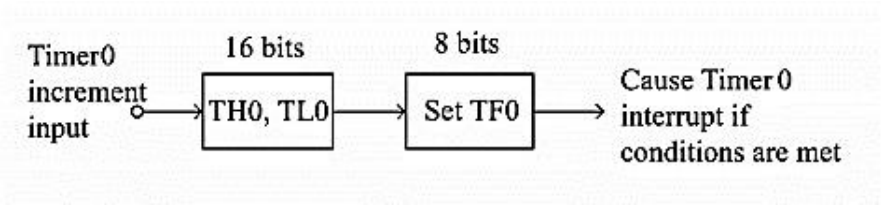
the TH1overflows from FFH to 00H, then the TF1 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH1 is holding F0H, and it is in timer mode, then TF1will be high after 10H * 32 = 512 machine cycles.

```
MOVTMOD, #00H
MOVTH1, #0F0H
MOVIE, #88H
SETB TR1
```

In the above program, the Timer1 is configured as timer mode 0. In this case Gate = 0. Then the TH1 will be loaded with F0H, then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer.

Mode1 (16-bit timer mode)

The Mode 1 operation is the 16-bit timer or counter. In the following diagram, we are using Mode 1 for Timer0.



In this case every event for counter operations or machine cycles for timer operation, the TH0– TL0 register-pair will be incremented by 1. When the register pair overflows from FFFFH to 0000H, then the TF0 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH0 – TL0 register pair is holding FFF0H, and it is in timer mode, then TF0 will be high after 10H = 16 machine cycles. When the clock frequency is 12MHz, then the following instructions generate an interrupt 16 μs after Timer0 starts running.

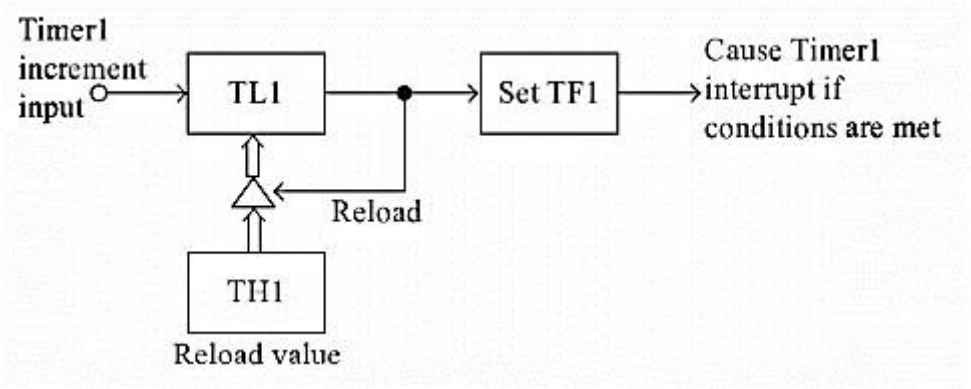
```
MOVTMOD, #01H
MOVTL0, #0F0H
MOVTH0, #0FFH
MOVIE, #82H
SETB TR0
```

In the above program, the Timer0 is configured as timer mode 1. In this case Gate = 0. Then the TL0 will be loaded with F0H and TH0 is loaded with FFH, then enable the Timer0 interrupt. At last set the TR0 of TCON register, and start the timer.

Mode2 (8-bit auto-reload timer mode)

The Mode 2 operation is the 8-bit auto reload timer or counter. In the following diagram, we are using Mode 2 for Timer1.

In this case every event for counter operations or machine cycles for timer operation, the TL1register will be incremented by 1. When the register pair overflows from FFH to 00H, then the TF1 of TCON register will be high, also theTL1 will be reloaded with the content of TH1 and starts the operation again.



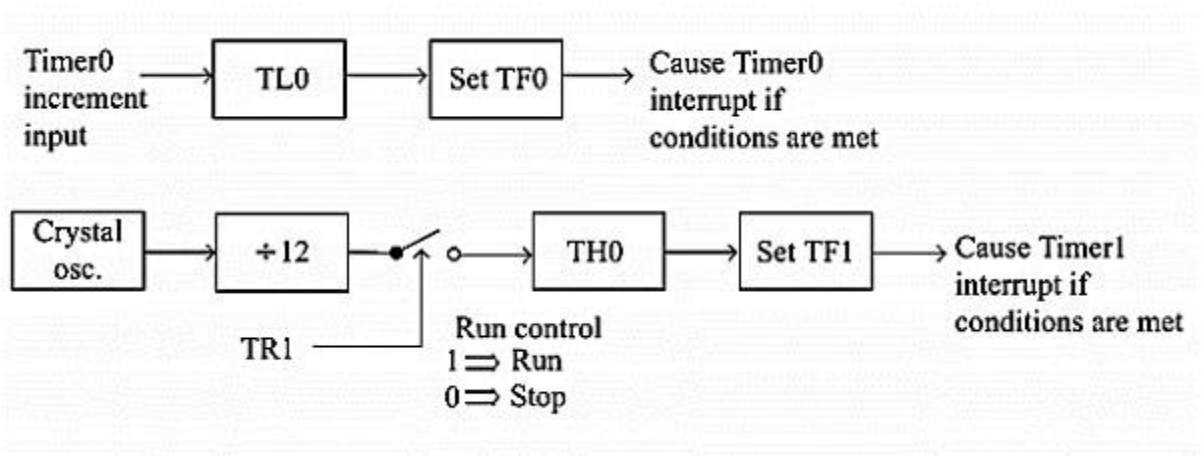
So for an example, we can say that if the TH1 and TL1 register both are holding F0H and it is in timer mode, then TF1 will be high after 10H= 16 machine cycles. When the clock frequency is 12MHz this happens after 16 µs, then the following instructions generate an interrupt once every 16 µs after Timer1 starts running.

```
MOVTMOD, #20H
MOVTL1, #0F0H
MOVTH1, #0F0H
MOVIE, #88H
SETBTR1
```

In the above program, the Timer1 is configured as timer mode 2. In this case Gate = 0. Then the TL1 and TH1 are loaded with F0H. then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer.

Mode 3 of Timer/Counter

Mode 3 is different for Timer0 and Timer1. When the Timer0 is working in mode 3, the TL0 will be used as an 8-bit timer/counter. It will be controlled by the standard Timer0 control bits, T0 and INT0 inputs. The TH0 is used as an 8-bit timer but not the counter. This is controlled by Timer1 Control bit TR1. When the TH0 overflows from FFH to 00H, then TF1 is set to 1. In the following diagram, we can Timer0 in Mode 3.



When the Timer1 is working in Mode 3, it simply holds the count but does not run. When Timer0 is in mode 3, the Timer1 is configured in one of the mode 0, 1 and 2. In this case, the Timer1 cannot interrupt the microcontroller. When the TF1 is used by TH0 timer, the Timer1 is used as Baud Rate Generator.

The meaning of gate bit in Timer0 and Timer1 for mode 3 is as follows

It controls the running of 8-bit timer/counter TL0 as like Mode 0, 1, or 2. The running of TH0 is controlled by TR1 bit only. So the gate bit in this mode for Timer0 has no specific role.

The mode 3 is present for applications requiring an extra 8-bit timer/counter. In Mode 3 of Timer0, the 8051 has three timers. One 8-bit timer by TH0, another8-bit timer/counter by TL0, and one 16-bit timer/counter by Timer1.

If the Timer0 is in mode3, and Timer1 is working on either 0, 1 or 2, then the gun control of the Timer1 is activated when the gate bit is low or INT1 is high. The run control is deactivated when the gate is high and INT1 is low.

1.6. Serial Port in 8051

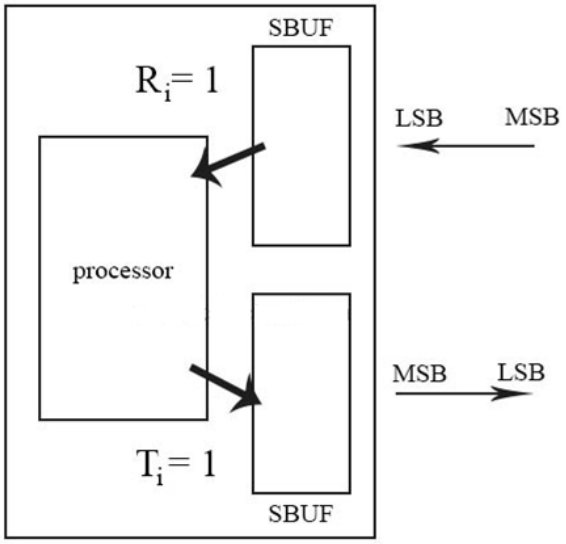
There is a serial port in 8051 as mentioned in the pin diagram of 8051. 8051 has capability to perform parallel as well as serial communication.

Parallel communication in 8051:

8051 can do 8-bit parallel communication as it has 8-bit ALU. For parallel communication, any of the ports (P0 / P1 / P2 / P3) is used as a transmission channel between transmitter and receiver.

Serial communication in 8051:

For serial communication there are two separate pins known as serial port of 8051.



TxD:

This pin basically acts as a transmitter (sending data), but in some other modes it doesn't do the job of transmitter. As it is serial communication, it sends bit by bit, the processor gives 8-bit at 1 time and those 8-bits are stored in a register named **SBUF.** Processor gives 1 byte of data that is to be transmitted to SBUF and from there bit by bit is transferred , firstly LSB and then at last MSB of the byte stored in SBUF. Once the total byte is transmitted, an interrupt is sent to the processor by making some flag 1, so that the processor can send more data for transmission as soon as the interrupt is received.

After every bit is transmitted, it requires delay for next bit transmission. So SBUF needs triggering which is provided by

- Timer T1 (here T1 only needs to trigger, T1 does not require its overflow flag , mode 3 in timers). Here we can vary the delay, so data transmission delay can be varied (frequency can be varied). It has a variable baud rate.
- There is an internal clock in 8051 (f_{osc} / 12 = 1Mhz) , where delay cannot be varied, this has fixed trigger delay. So frequency cannot be varied.It has a fixed baud rate.

Whenever SBUF transferred 8bit of data , T_i flag becomes 1. Whenever processors go to ISR(in other interrupts the flag is auto cleared whenever processor goes to ISR) , in this the T_i flag is not auto cleared.

RxD:

This pin is basically for data reception . It received data bit by bit (as the transmitter sends LSB first, it received LSB first). There is also a register **SBUF** which stores 8 received bits. Once the 8 bits are received, instead of sending an interrupt it firstly checks for errors (errors caused due to transmission). Once there is no error in the received information R_i flag is set and an interrupt is sent to the processor. Processor goes to ISR (here also R_i is not cleared automatically).

How are SBUF in TxD and RxD different from each other ?

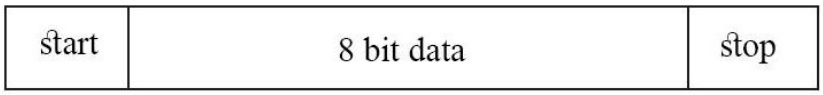
In SBUF of TxD, data is sent from processor to SBUF
In SBUF of RxD, data is sent from SBUF to the processor.
In this way both registers are differentiated by the processor.

There is a bit named SM_2 ;
If SM_2 = 1, then after SBUF of RxD is filled there will be error check And if not then there will be no error check, directly an interrupt will be sent to the processor once SBUF in RxD is filled.

1.6.1. Modes in serial communication:

Mode 1 (8-bit UART communication):

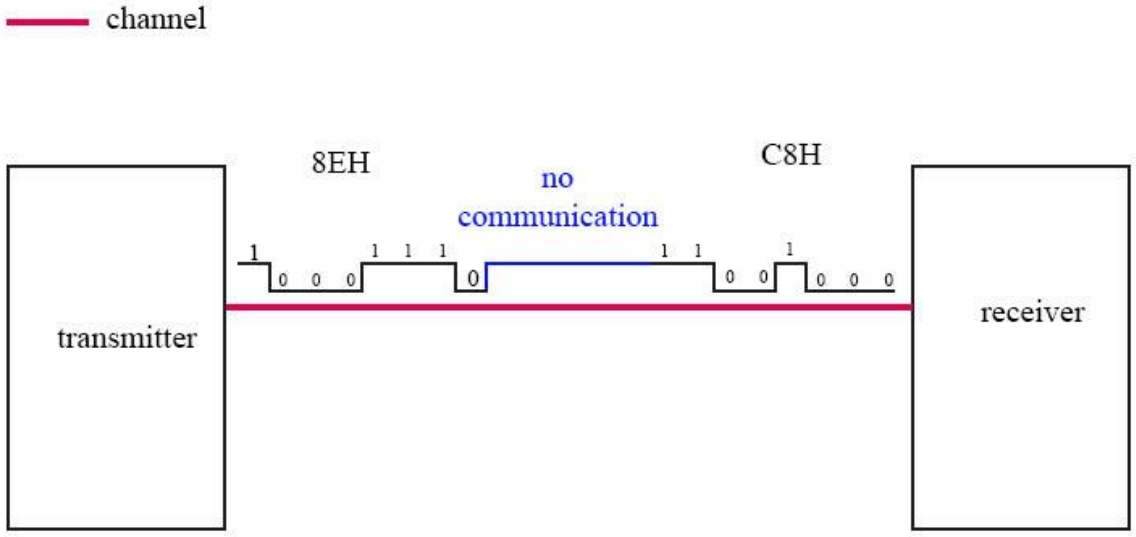
UART stands for universal asynchronous receiver-transmitter. It means receiver and transmitter are asynchronous which mean they don't have a common clock. Normally 8 bits are transmitted through the channel, but in this mode 10 bits are transmitted.



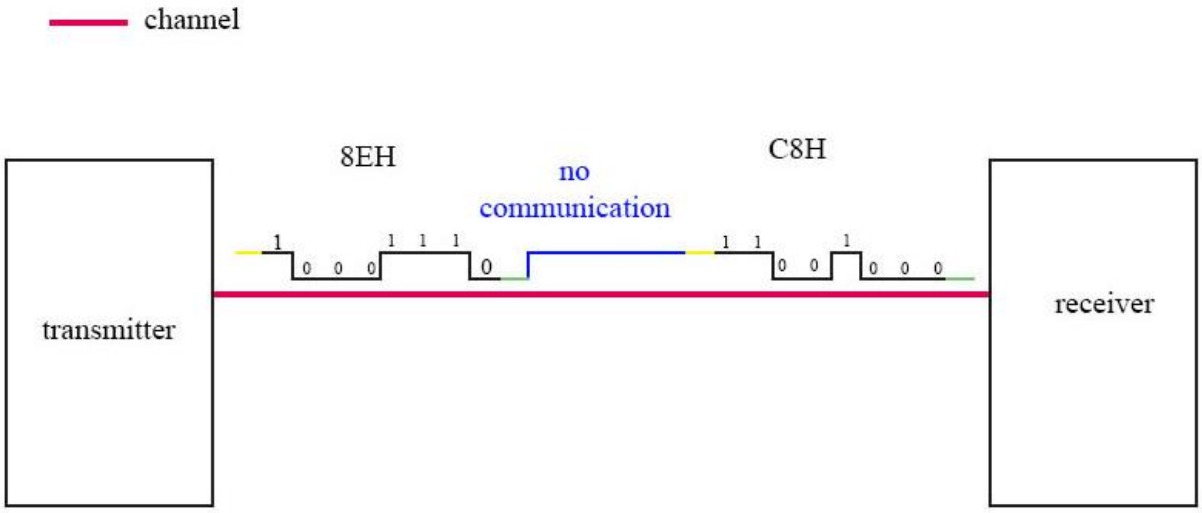
Here start and stop bits are system generated.

Significance of start and stop bit:

For eg: There is a transmitter and receiver. First C8H (1100 1000) data is transmitted and then for 10min there is no transmission and after that again 8EH (1000 1110)is transmitted. If there is no transmission, last bit transmitted would be remained in the channel

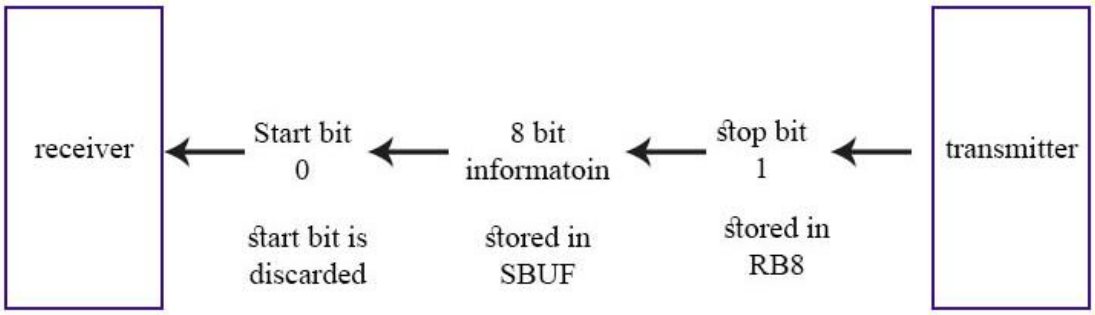


Receiver assumes blue data (when there was no communication) also as data.



Here green is the start bit which is zero. And then the yellow colored is stop bit which is 1. Whenever the first 0 bit comes, the receiver discards the start bit and accepts the next 8 bits and stores in SBUF. Then the 9th bit is 1 , this bit is stored in **R_{B8}** (will be discussed later). Then after this whenever the next zero bit comes (that zero bit is discarded and accepts the next 8 bits and so on).

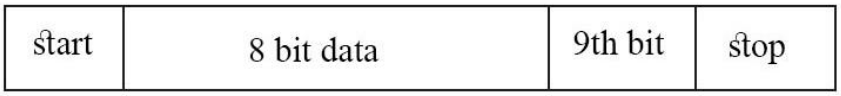
Mode 1



Stop bit is also used for error checking. Whenever SM_2=1, It checks for error, If the RB8 = 1 (which means stop = 1 received, so the data is received correctly) and if RB8=0 (transmitter generated stop as 1, but received as 0) so there is an error. If there is an error in received data, no interrupt is sent to the processor.

This mode is variable baud rate, which means it is triggered by timer 1.

Mode 2 (9-bit UART communication):



The 9th bit is a programmable bit and it is given through TB8. Here 9th bit is 1 and it is used for error checking and stop bit for triggering the data high (so start bit gets 0 and so on).
Why the 9th bit , when the already stop bit exists?
Standard value of 9th bit is 1 and can be made 0.

Whenever SM_2 = 1(receiver accepts only errorless data) and if 9th bit is 1, then only errorless data is accepted or else discarded. Discarding data is a purpose.

Mode 3 (9-bit UART communication):
This mode is completely similar to mode 2, in mode 2 for triggering timer is used Whereas in this mode internal clock is used for triggering. It has a fixed baud rate.

Mode 0 :
Totally there were four modes in serial port of 8051, but for better understanding mode 0 is explained after three modes. In this mode data is transferred and received only through the RxD channel. TxD is used for clocks. This is synchronous mode of communication.
Such a system is also known as half duplex mode. It has fixed baud rate.

SCON register:

SM_0	SM_1	SM_2	REN	TB_8	RB_8	T_i	R_i

SM_0 and SM_1:
These are used to select the mode.

SM_0	SM_1	Mode
0	0	0
0	1	1
1	0	2
1	1	3

SM_2:
If SM_2 = 1, error is checkedOr else no error checking is done.
REN:
Receiver enable, If REN=1, receiver will receive the data or else not.
TB_8:
This is the 9th bit to be transmitted.
RB_8:
This is the 9th bit to be received.
T_i :
When 8-bits are received in SBUF , then R_i = 1, that would send an interrupt to the processor.
R_i :
When 8-bits are sent from SBUF,and SBUF is empty , then R_i = 1, that would send an interrupt to the processor. Before R_i=1, it checks for error based on SM_2.

1.7. 8-Bit Interrupts

Interrupts are events detected by the MCU which cause normal program flow to be pre-empted. Interrupts pause the current program and transfer control to a specified user-written firmware routine called the Interrupt Service Routine (ISR). The ISR processes the interrupt event, then resumes normal program flow.

1.7.1. Overview of Interrupt Process

Program MCU to react to interrupts

The MCU must be programmed to enable interrupts to occur. Setting the Global Interrupt Enable (GIE) and, in many cases, the Peripheral Interrupt Enable (PEIE), enables the MCU to receive interrupts. GIE and PEIE are located in the Interrupt Control (INTCON) special function register.

Enable interrupts from selected peripherals

Each peripheral on the MCU has an individual enable bit. A peripheral's individual interrupt enable bit must be set, in addition to GIE/PEIE, before the peripheral can generate an interrupt. The individual interrupt enable bits are located in INTCON, PIE1, PIE2, and PIE3.

Peripheral asserts an interrupt request

When a peripheral reaches a state where program intervention is needed, the peripheral sets an Interrupt Request Flag (xxIF). These interrupt flags are set regardless of the status of the GIE, PEIE, and individual interrupt enable bits. The interrupt flags are located in INTCON, PIR1, PIR2, and PIR3.

The interrupt request flags are latched high when set and must be cleared by the user-written ISR.

Interrupt occurs

When an interrupt request flag is set and the interrupt is properly enabled, the interrupt process begins:

1. Global Interrupts are disabled by clearing GIE to 0.
2. The current program context is saved to the shadow registers.
3. The value of the Program Counter is stored on the return stack.
4. Program control is transferred to the interrupt vector at address 04h.

ISR runs

The ISR is a function written by the user and placed at address 04h. The ISR does the following:

1. Checks the interrupt-enabled peripherals for the source of the interrupt request.
2. Performs the necessary peripheral tasks.
3. Clears the appropriate interrupt request flag.
4. Executes the Return From Interrupt instruction (RETFIE) as the final ISR instruction.

Control is returned to the Main program

When RETFIE is executed:

1. Global Interrupts are enabled (GIE=1).

2. The program context is restored from the Shadow registers.
3. The return address from the stack is loaded into the Program Counter.
4. Execution resumes from point at which it was interrupted.

1.7.2. Registers Used to Process Interrupts

Interrupt Control Register

INTCON register

GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF
bit 7							bit 0

GIE - Global Interrupt Enable

PEIE - Peripheral Interrupt Enable

TMR0IE - Timer0 Interrupt Enable

INTE - External Interrupt Enable

IOCIE - Interrupt on Change Enable

TMR0IF - Timer0 Interrupt flag

INTF - External Interrupt flag

IOCIF - Interrupt on Change flag

INTCON contains global and peripheral interrupt enable flags as well as the individual interrupt request flags and interrupt enable flags for three of the PIC16F1xxxx interrupts.

Interrupt Enable Registers

PIE1 register

TMR1GIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMRI1E
bit 7							bit 0

TMR1GIE - Timer1 Gate Interrupt Enable

ADIE - Analog-to-Digital Converter (ADC) Interrupt Enable

RCIE - Universal Synchronous Asynchronous Receiver Tranmsitter (USART) Receive Interrupt Enable

TXIE - USART Transmit Interrupt Enable

SSPIE - Synchronous Serial Port (MSSP) Interrupt Enable

CCP1IE - CCP1 Interrupt Enable

TMR2IE - Timer2 Interrupt Enable

TMR1IE - Timer1 Interrupt Enable

PIE2 register

OSFIE	C2IE	C1IE	EEIE	BCLIE	LCDIE	----	CCP2IE
bit 7							bit 0

OSFIE - Oscillator Fail Interrupt Enable

C2IE - Comparator C2 Interrupt Enable

C1IE - Comparator C1 Interrupt Enable

EEIE - EEPROM Write Completion Interrupt Enable

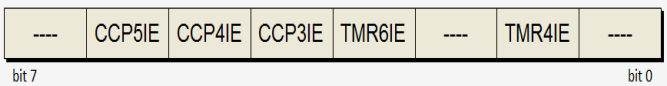
BCLIE - MSSP Bus Collision Interrupt Enable

LCDIE - LCD Module Interrupt Enable

--- - Unimplemented, read as 0

CCP2IE - CCP2 Interrupt Enable

PIE3 register



--- - Unimplemented read as 0

CCP5IE - CCP5 Interrupt Enable

CCP4IE - CCP4 Interrupt Enable

CCP3IE - CCP3 Interrupt Enable

TMR6IE - Timer6 Interrupt Enable

--- - Unimplemented, read as 0

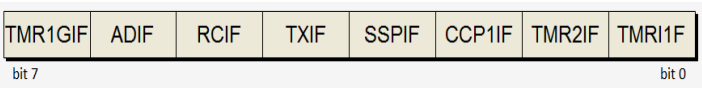
TMR4IE - Timer4 Interrupt Enable

--- - Unimplemented, read as 0

PIE1, **PIE2**, and **PIE3** contain the individual interrupt enable flags for the MCU's peripherals.

Interrupt Request Registers

PIR1 register



TMR1GIF - Timer1 Gate Interrupt Flag

ADIF - ADC Interrupt Flag

RCIF - USART Receive Interrupt Flag

TXIF - USART Transmit Interrupt Flag

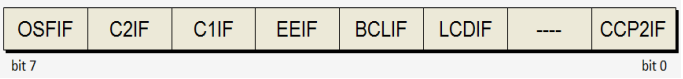
SSPIF - MSSP Interrupt Flag

CCP1IF - CCP1 Interrupt Flag

TMR2IF - Timer2 Interrupt Flag

TMR1IF - Timer1 Interrupt Flag

PIR2 register



OSFIF - Oscillator Fail Interrupt Flag

C2IF - Comparator C2 Interrupt Flag

C1IF - Comparator C1 Interrupt Flag

EEIF - EEPROM Write Completion Interrupt Flag

BCLIF - MSSP Bus Collision Interrupt Flag

LCDIF - LCD Module Interrupt Flag

--- - Unimplemented, read as 0

CCP2IF - CCP2 Interrupt Flag

PIR3 register

----	CCP5IF	CCP4IF	CCP3IF	TMR6IF	----	TMR4IF	----
bit 7							bit 0

--- - Unimplemented, read as 0

CCP5IF - CCP5 Interrupt Flag

CCP4IF - CCP4 Interrupt Flag

CCP3IF - CCP3 Interrupt Flag

TMR6IF - Timer6 Interrupt Flag

--- - Unimplemented, read as 0

TMR4IF - Timer4 Interrupt Flag

--- - Unimplemented, read as 0

PIR1, **PIR2**, and **PIR3** contain the individual interrupt request flags for the MCU's peripherals.

OPTION_REG

OPTION_REG

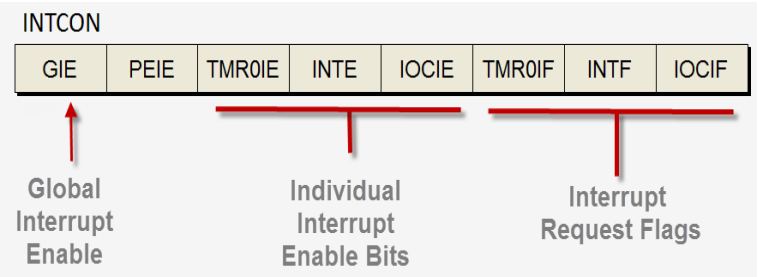
WPUEN	INTEDG	TMR0CS	TMR0SE	PSA	PS<2:0>
bit 7					bit 0

The INTEDG flag in OPTION_REG is used to set a rising or falling edge on the INT pin as the trigger for an INTE interrupt.

Enabling Interrupts

Core Interrupts

Three interrupt sources (Timer0, External Interrupt, and Interrupt on Change) have interrupt enable bits located in INTCON. These interrupts are referred to as core interrupts.



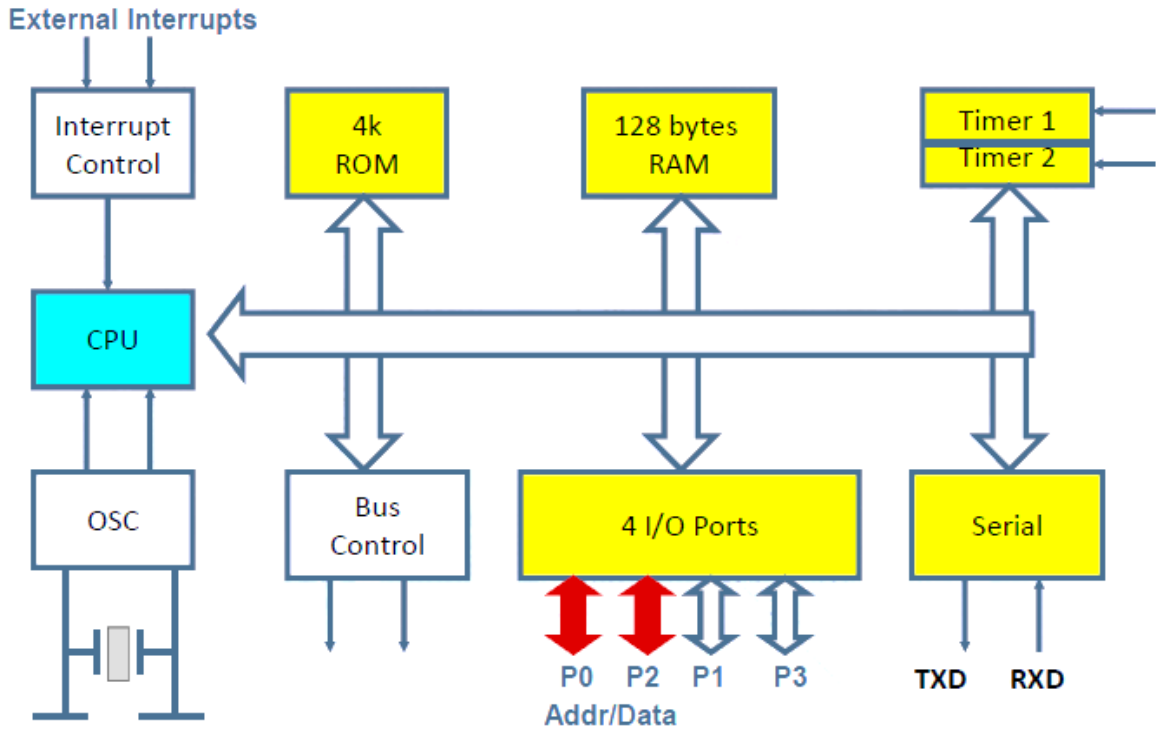
UNIT II

EMBEDDED C PROGRAMMING

- Memory And I/O Devices Interfacing
- Programming Embedded Systems in C
- Need For RTOS
- Multiple Tasks and Processes
- Context Switching
- Priority Based Scheduling Policies

2.1. Memory and I/O Interfacing

Several memory chips and I/O devices are connected to a microprocessor.The following figure shows a schematic diagram to interface memory chips and I/O devices to a microprocessor.



2.1.1. Memory Interfacing

When we are executing any instruction, the address of memory location or an I/O device is sent out by the microprocessor. The corresponding memory chip or I/O device is selected by a decoding circuit.

Memory requires some signals to read from and write to registers and microprocessor transmits some signals for reading or writing data.

The interfacing process includes matching the memory requirements with the microprocessor signals. Therefore, the interfacing circuit should be designed in such a way that it matches the memory signal requirements with the microprocessor's signals.

2.1.2. I/O interfacing

As we know, keyboard and displays are used as communication channel with outside world. Therefore, it is necessary that we interface keyboard and displays with the microprocessor. This is called I/O interfacing. For this type of interfacing, we use latches and buffers for interfacing the keyboards and displays with the microprocessor.

But the main drawback of this interfacing is that the microprocessor can perform only one function.

8051 Microcontroller Memory Organization

In the 8051 Microcontroller, we have seen the 8051 Microcontroller Introduction and Basics, Pin Diagram, Pin Description and the Architecture overview. we will continue exploring 8051 Microcontroller by understanding the 8051 Microcontroller Memory Organization, Program Memory (ROM), Data Memory (RAM), External Memory.

Differences between microprocessor and microcontroller

The main difference can be stated as on-chip memory i.e., a Microcontroller has both Program Memory (ROM) and Data Memory (RAM) on the same chip (IC), whereas a Microprocessor has to be externally interfacing with the memory modules.

Hence, it is clear that the memory is an important part of the 8051 Microcontroller Architecture (for that matter, any Microcontroller). So, it is important for us to understand the 8051 Microcontroller Memory Organization i.e., how memory is organized, how the processor accesses each memory and how to interface external memory with 8051 Microcontroller.

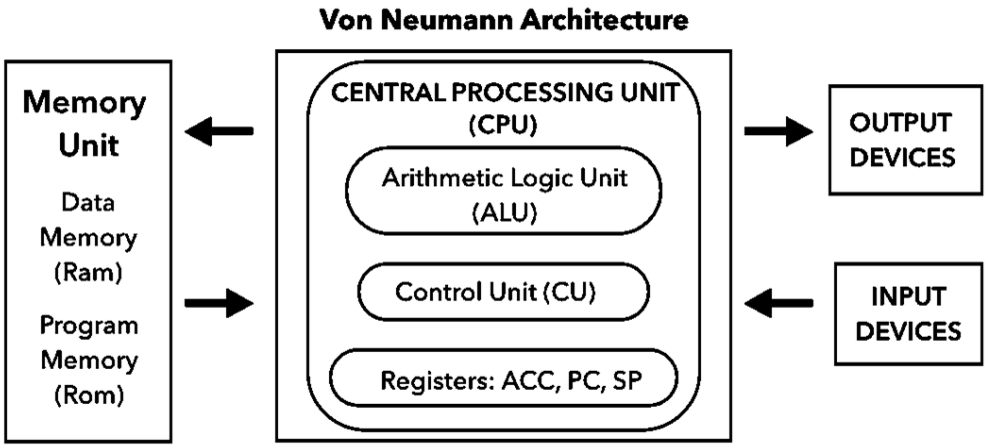
Before going in to the details of the 8051 Microcontroller Memory Organization, we will first see a little bit about the Computer Architecture and then proceed with memory organization of 8051 Microcontroller.

1.2.3. Types of Computer Architecture

Basically, Microprocessors or Microcontrollers are classified based on the two types of Computer Architecture: Von Neumann Architecture and Harvard Architecture.

Von Neumann Architecture

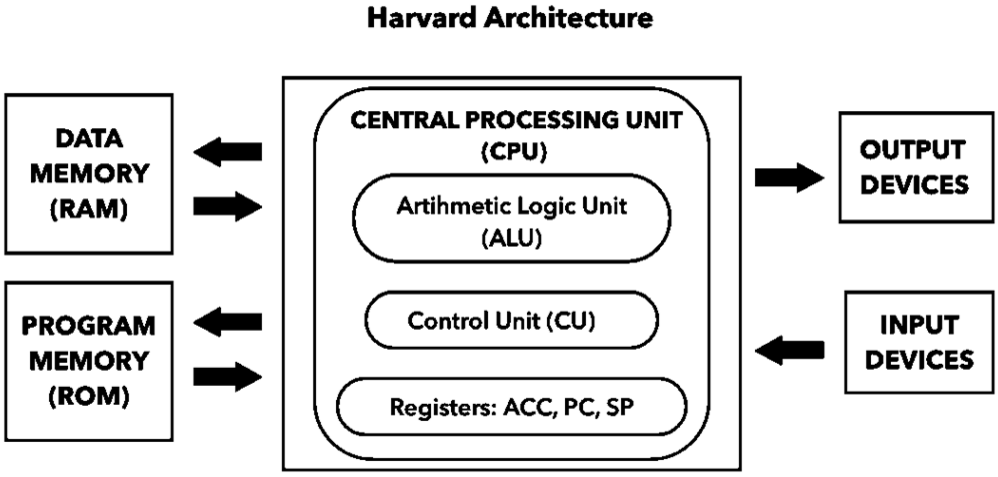
Von Neumann Architecture or Princeton Architecture is a Computer Architecture, where the Program i.e., the Instructions and the Data are stored in a single memory.Since the Instruction Memory and the Data Memory are the same, the Processor or CPU cannot access both Instructions and Data at the same time as they use a single bus.This type of architecture has severe limitations to the performance of the system as it creates a bottleneck while accessing the memory.



Harvard Architecture

Harvard Architecture, in contrast to Von Neumann Architecture, uses separate memory for Instruction (Program) and Data. Since the Instruction Memory and Data Memory are separate in a Harvard Architecture, their signal paths i.e., buses are also different and hence, the CPU can access both Instructions and Data at the same time.

Almost all Microcontrollers, including 8051 Microcontroller implement Harvard Architecture.



1.2.4. 8051 Microcontroller Memory Organization

The 8051 Microcontroller Memory is separated in Program Memory (ROM) and Data Memory (RAM). The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e., instructions. The Data Memory on the other hand, is used for storing temporary variable data and intermediate results.

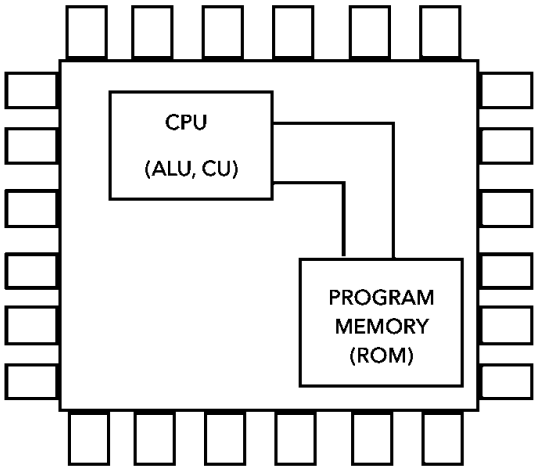
8051 Microcontroller has both Internal ROM and Internal RAM. If the internal memory is inadequate, you can add external memory using suitable circuits.

1.2.4.1. Program Memory (ROM) of 8051 Microcontroller

In 8051 Microcontroller, the code or instructions to be executed are stored in the Program Memory, which is also called as the ROM of the Microcontroller. The original 8051 Microcontroller by Intel has 4KB of internal ROM.

Some variants of 8051 like the 8031 and 8032 series doesn't have any internal ROM (Program Memory) and must be interfaced with external Program Memory with instructions loaded in it.

8051 Program Memory

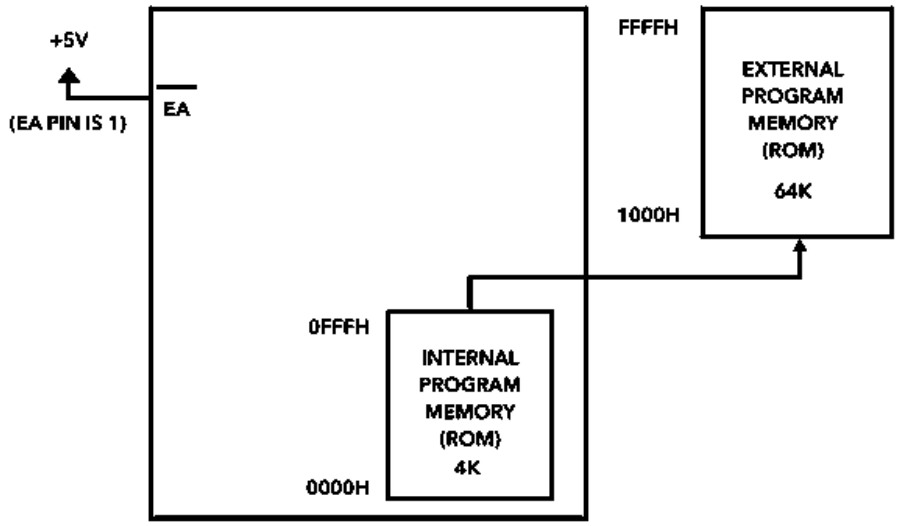


Almost all modern 8051 Microcontrollers, like 8052 Series, have 8KB of Internal Program Memory (ROM) in the form of Flash Memory (ROM) and provide the option of reprogramming the memory.

In case of 4KB of Internal ROM, the address space is 0000H to 0FFFH. If the address space i.e., the program addresses exceed this value, then the CPU will automatically fetch the code from the external Program Memory.

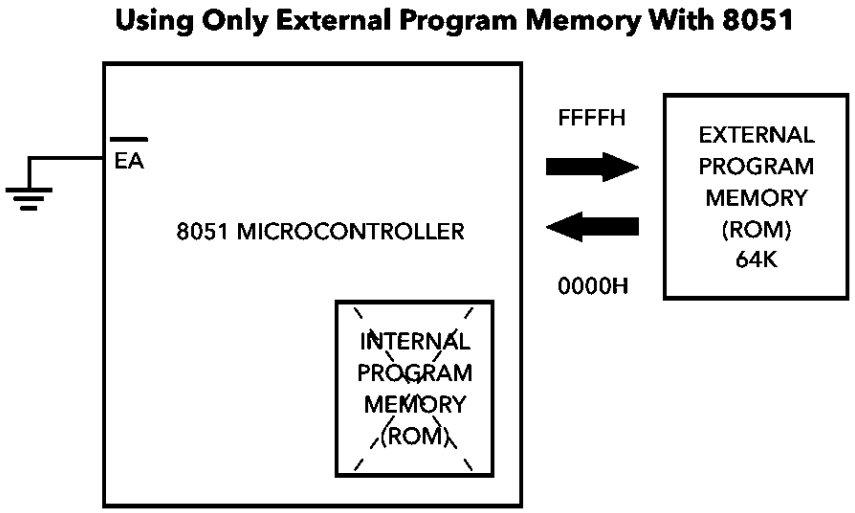
For this, the External Access Pin (EA Pin) must be pulled HIGH i.e., when the EA Pin is high, the CPU first fetches instructions from the Internal Program Memory in the address range of 0000H to 0FFFFH and if the memory addresses exceed the limit, then the instructions are fetched from the external ROM in the address range of 1000H to FFFFH.

Using Both Internal And External Program Memory With 8051



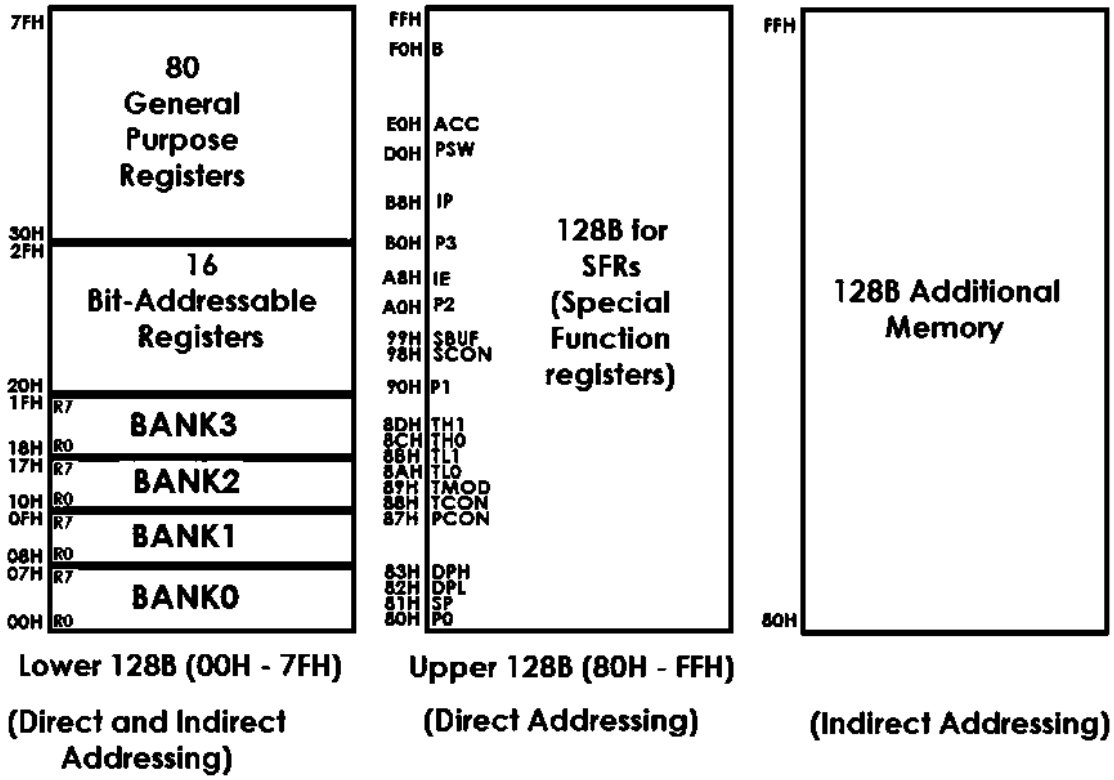
There is another way to fetch the instructions: ignore the Internal ROM and fetch all the instructions only from the External Program Memory (External ROM). For this scenario, the

EA Pin must be connected to GND. In this case, the memory addresses of the external ROM will be from 0000H to FFFFH.



1.2.4.2. Data Memory (RAM) of 8051 Microcontroller

The Data Memory or RAM of the 8051 Microcontroller stores temporary data and intermediate results that are generated and used during the normal operation of the microcontroller. Original Intel's 8051 Microcontroller had 128B of internal RAM.



But almost all modern variants of 8051 Microcontroller have 256B of RAM. In this 256B, the first 128B i.e., memory addresses from 00H to 7FH is divided in to Working Registers (organized as Register Banks), Bit – Addressable Area and General Purpose RAM (also known as Scratchpad area).

In the first 128B of RAM (from 00H to 7FH), the first 32B i.e., memory from addresses 00H to 1FH consists of 32 Working Registers that are organized as four banks with 8 Registers in each Bank.

The 4 banks are named as Bank0, Bank1, Bank2 and Bank3. Each Bank consists of 8 registers named as R0 – R7. Each Register can be addressed in two ways: either by name or by address.To address the register by name, first the corresponding Bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the Program Status Word (PSW) Register (RS0 and RS1 are 3rd and 4th bits in the PSW Register).

When addressing the Register using its address i.e., 12H for example, the corresponding Bank may or may not be selected. (12H corresponds to R2 in Bank2).

The next 16B of the RAM i.e., from 20H to 2FH are Bit – Addressable memory locations. There are totally 128 bits that can be addressed individually using 00H to 7FH or the entire byte can be addressed as 20H to 2FH.

1.2.5. Interfacing External Memory with 8051 Microcontroller

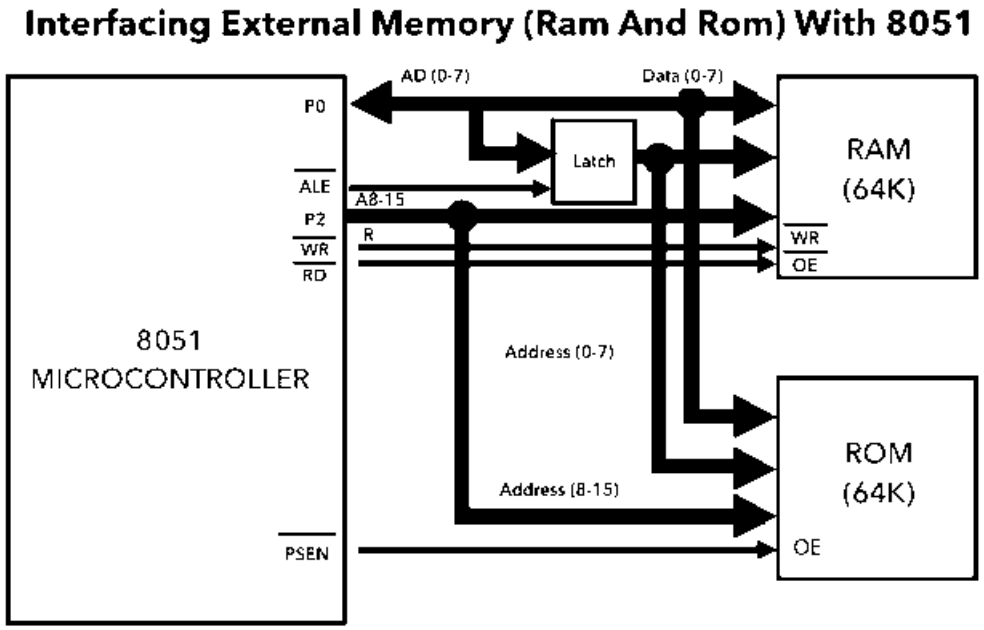
It is always good to have an option to expand the capabilities of a Microcontroller, whether it is in terms of Memory or IO or anything else. Such expansion will be useful to avoid design throttling. We have seen that a typical 8051 Microcontroller has 4KB of ROM and 128B of RAM (most modern 8051 Microcontroller variants have 8K ROM and 256B of RAM).

The designer of an 8051 Microcontroller based system is not limited to the internal RAM and ROM present in the 8051 Microcontroller. There is a provision of connecting both external RAM and ROM i.e., Data Memory and Program.

The reason for interfacing external Program Memory or ROM is that complex programs written in high – level languages often tend to be larger and occupy more memory.
Another important reason is that chips like 8031 or 8032, which doesn't have any internal ROM, have to be interfaced with external ROM.

A maximum of 64KB of Program Memory (ROM) and Data Memory (RAM) each can be interface with the 8051 Microcontroller.

The following image shows the block diagram of interfacing 64KB of External RAM and 64KB of External ROM with the 8051 Microcontroller.



An important point to remember when interfacing external memory with 8051 Microcontroller is that Port 0 (P0) cannot be used as an IO Port as it will be used for multiplexed address and data bus (A0 – A7 and D0 – D7). Not always, but Port 2 may be used as higher byte of the address bus.

2.2. Embedded C Programming with Keil Language

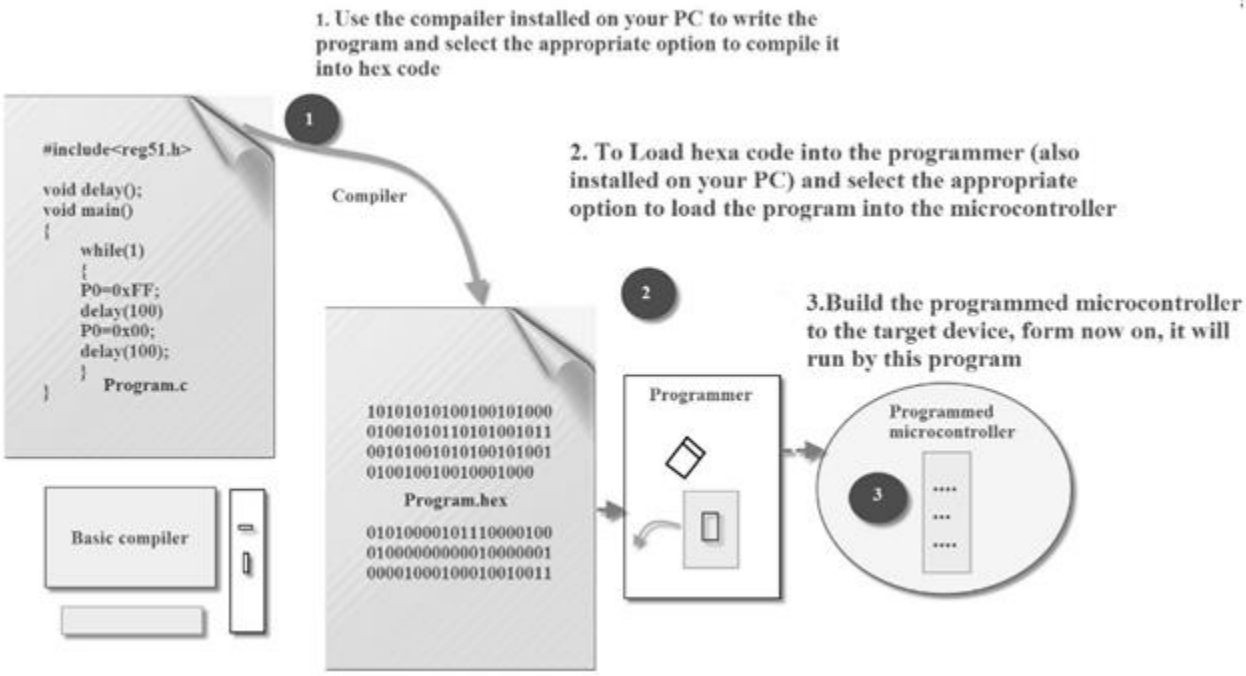
Embedded C is most popular programming language in software field for developing electronic gadgets. Each processor used in electronic system is associated with embedded software.

Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.

In embedded system programming C code is preferred over other language. Due to the following reasons:

- Easy to understand
- High Reliability
- Portability
- Scalability

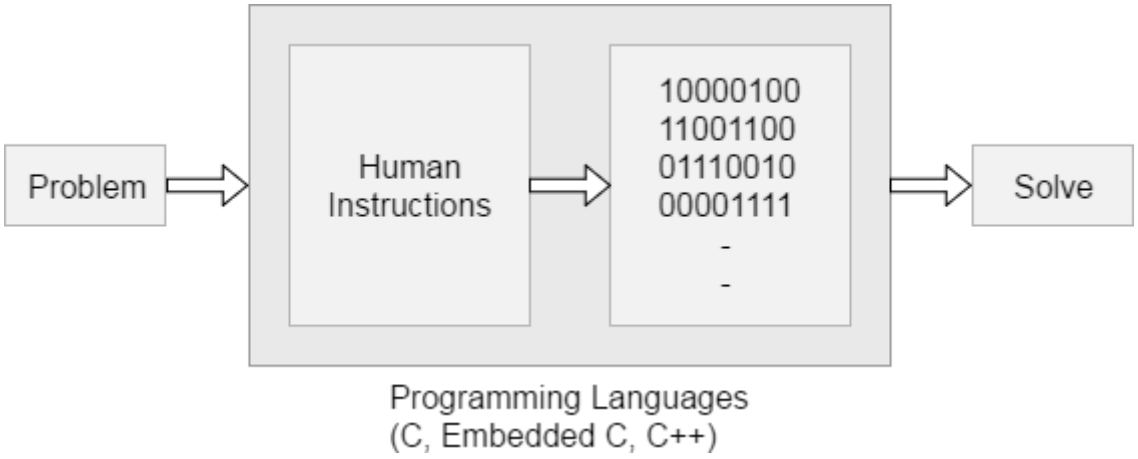
Let's see the block diagram representation of embedded system programming:



2.2.1. Embedded System Programming:

Basic Declaration

Let's see the block diagram of Embedded C Programming development:



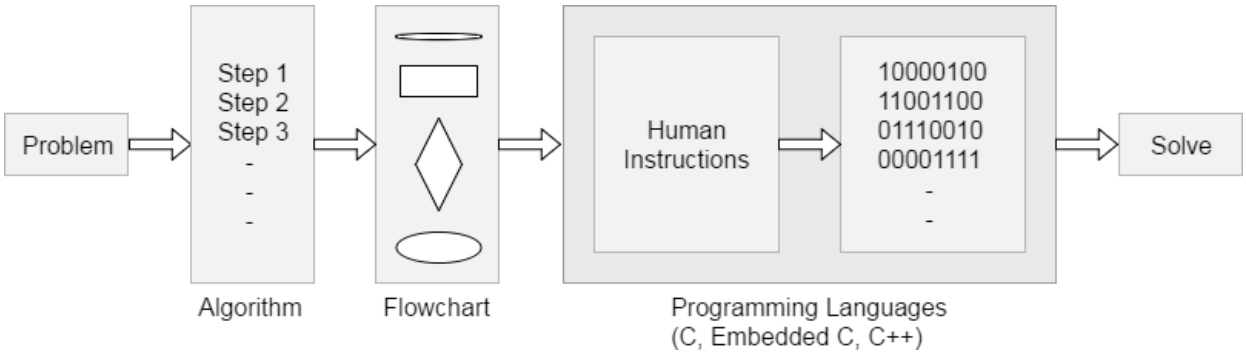
Function is a collection of statements that is used for performing a specific task and a collection of one or more functions is called a programming language. Every language is consisting of basic elements and grammatical rules. The C language programming is designed for function with variables, character set, data types, keywords, expression and so on are used for writing a C program.

The extension in C language is known as embedded C programming language. As compared to above the embedded programming in C is also have some additional features like data types, keywords and header file etc is represented by

#include<**microcontroller** name.h>

Basic Embedded C Programming Steps

Let's see the block diagram representation of Embedded C Programming Steps:



The microcontroller programming is different for each type of operating system. Even though there are many operating system are exist such as Windows, Linux, RTOS, etc but RTOS has several advantage for embedded system development.

2.2.2. Basics of Embedded C Program

Embedded C is one of the most popular and most commonly used Programming Languages in the development of Embedded Systems. So, we will see some of the Basics of Embedded C Program and the Programming Structure of Embedded C.

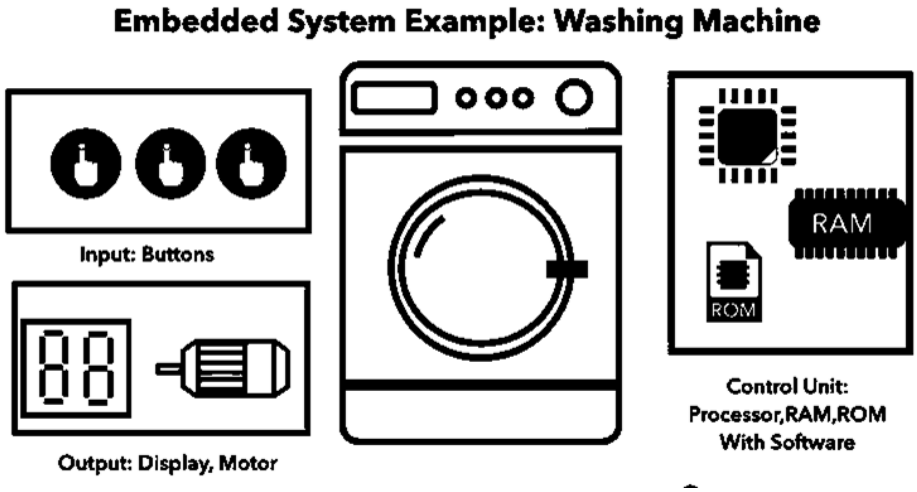
2.2.2.1. What is an Embedded System?

An Embedded System is a combination of Hardware and Software. My desktop computer also has hardware and software. Does that mean a desktop computer is also an Embedded System? NO. A desktop computer is considered a general purpose system as it can do many different tasks that too simultaneously. Some common tasks are playing videos, working on office suites, editing images (or videos), browsing the web, etc.

An Embedded System is more of an application oriented system i.e. it is dedicated to perform a single task (or a limited number of tasks, but all working for a single main aim).

An example for embedded system, which we use daily, is a Wireless Router. In order to get wireless internet connectivity on our mobile phones and laptops, we often use routers. The task of a wireless router is to take the signal from a cable and transmit it wirelessly. And take wireless data from the device (like a mobile) and send it through the cable.

We use washing machines almost daily but wouldn’t get the idea that it is an embedded system consisting of a Processor (and other hardware as well) and software.



It takes some inputs from the user like wash cycle, type of clothes, extra soaking and rinsing, spin rpm, etc., performs the necessary actions as per the instructions and finishes washing and drying the clothes. If no new instructions are given for the next wash, then the washing machines repeats the same set of tasks as the previous wash.

Embedded Systems can not only be stand-alone devices like Washing Machines but also be a part of a much larger system. An example for this is a Car. A modern day Car has several individual embedded systems that perform their specific tasks with the aim of making a smooth and safe journey.

Some of the embedded systems in a Car are Anti-lock Braking System (ABS), Temperature Monitoring System, Automatic Climate Control, Tire Pressure Monitoring System, Engine Oil Level Monitor, etc.

2.2.3. Programming Embedded Systems

As mentioned earlier, Embedded Systems consists of both Hardware and Software. If we consider a simple Embedded System, the main Hardware Module is the Processor. The Processor is the heart of the Embedded System and it can be anything like a Microprocessor, Microcontroller, DSP, CPLD (Complex Programmable Logic Device) or an FPGA (Field Programmable Gated Array).

All these devices have one thing in common: they are programmable i.e., we can write a program (which is the software part of the Embedded System) to define how the device actually works.

Embedded Software or Program allow Hardware to monitor external events (Inputs / Sensors) and control external devices (Outputs) accordingly. During this process, the program for an Embedded System may have to directly manipulate the internal architecture of the Embedded Hardware (usually the processor) such as Timers, Serial Communications Interface, Interrupt Handling, and I/O Ports etc.

From the above statement, it is clear that the Software part of an Embedded System is equally important as the Hardware part. There is no point in having advanced Hardware Components with poorly written programs (Software).

There are many programming languages that are used for Embedded Systems like Assembly (low-level Programming Language), C, C++, JAVA (high-level programming languages), Visual Basic, JAVA Script (Application level Programming Languages), etc.

In the process of making a better embedded system, the programming of the system plays a vital role and hence, the selection of the Programming Language is very important.

2.2.4. Factors for Selecting the Programming Language

The following are few factors that are to be considered while selecting the Programming Language for the development of Embedded Systems.

- **Size:** The memory that the program occupies is very important as Embedded Processors like Microcontrollers have a very limited amount of ROM (Program Memory).
- **Speed**: The programs must be very fast i.e., they must run as fast as possible. The hardware should not be slowed down due to a slow running software.
- **Portability:** The same program can be compiled for different processors.
- Ease of Implementation
- Ease of Maintenance
- Readability

Earlier Embedded Systems were developed mainly using Assembly Language. Even though Assembly Language is closest to the actual machine code instructions and produces small size hex files, the lack of portability and high amount of resources (time and man power) spent on developing the code, made the Assembly Language difficult to work with.

There are other high-level programming languages that offered the above mentioned features but none were close to C Programming Language. Some of the benefits of using Embedded C as the main Programming Language:

- Significantly easy to write code in C
- Consumes less time when compared to Assembly
- Maintenance of code (modifications and updates) is very simple
- Make use of library functions to reduce the complexity of the main code
- You can easily port the code to other architecture with very little modifications

2.2.5. Embedded System and Its Real Time Applications

- Embedded System and its Real Time Applications. Focuses on different topics like What is an Embedded System, What are the Real Time Applications of Embedded Systems, What is the Future of Embedded Systems, etc.

- The World is filled with Embedded Systems. The development of Microcontroller has paved path for several Embedded System application and they play a significant role (and will continue to play in the future as well) in our modern day life in one way or the other.
- Starting from consumer electronics like Digital Cameras, DVD Players to high end and advanced systems like Flight Controllers and Missile Guidance Systems, embedded systems are omnipresent and became an important part of our life.
- The way we live our life has been significantly improved with the utilization of Embedded Systems and they will continue to be an integral part of our lives even tomorrow.
- Another important concept we are hearing these days is Real – Time Systems. In a real time system, Real Time Computing takes place, where a computer (an Embedded System) must generate response to events within certain time limits.
- Before going in to the details of Real Time Applications of Embedded Systems, we will first see what an Embedded System is, what is a real time system and what is real time operating system.

2.2.6. *Components of Embedded System*

An Embedded System consists of four main components. They are the Processor (Microprocessor or Microcontroller), Memory (RAM and ROM), Peripherals (Input and Output) and Software (main program).

Processor: The heart of an Embedded System is the Processor. Based on the functionality of the system, the processor can be anything like a General Purpose Processor, a single purpose processor, an Application Specific Processor, a microcontroller or an FPGA.

Memory: Memory is another important part of an embedded system. It is divided in to RAM and ROM. Memory in an Embedded System (ROM to be specific) stores the main program and RAM stores the program variables and temporary data.

Peripherals: In order to communicate with the outside world or control the external devices, an Embedded System must have Input and Output Peripherals. Some of these peripherals include Input / Output Ports, Communication Interfaces, Timers and Counters, etc.

Software: All the hardware work according to the software (main program) written. Software part of an Embedded System includes initialization of the system, controlling inputs and outputs, error handling etc.

NOTE: Many Embedded Systems, usually small to medium scaled systems, generally consists of a Microcontroller as the main processor. With the help of a Microcontroller, the processor, memory and few peripherals will be integrated in to a single device.

2.2.7. Introduction to Embedded C Programming Language

Before going in to the details of Embedded C Programming Language and basics of Embedded C Program, we will first talk about the C Programming Language.

The C Programming Language became so popular that it is used in a wide range of applications ranging from Embedded Systems to Super Computers.

Embedded C Programming Language, which is widely used in the development of Embedded Systems, is an extension of C Program Language. The Embedded C Programming Language uses the same syntax and semantics of the C Programming Language like main function, declaration of datatypes, defining variables, loops, functions, statements, etc.

The extension in Embedded C from standard C Programming Language include I/O Hardware Addressing, fixed point arithmetic operations, accessing address spaces, etc.

2.2.7.1. Difference between C and Embedded C

There is actually not much difference between C and Embedded C apart from few extensions and the operating environment. Both C and Embedded C programming are ISO Standards that have almost same syntax, datatypes, functions, etc.

Embedded C is basically an extension to the Standard C Programming Language with additional features like Addressing I/O, multiple memory addressing and fixed-point arithmetic, etc.

C Programming Language is generally used for developing desktop applications, whereas Embedded C is used in the development of Microcontroller based applications.

2.2.7.2. Keywords in Embedded C

A Keyword is a special word with a special meaning to the compiler (a C Compiler for example, is a software that is used to convert program written in C to Machine Code). For example, if we take the Keil's Cx51 Compiler (a popular C Compiler for 8051 based Microcontrollers) the following are some of the keywords:

- bit
- sbit
- sfr

- small
- large

The following table lists out all the keywords associated with the Cx51 C Compiler.

_at_	alien	bdata
bit	code	compact
data	far	idata
interrupt	large	pdata
priority	reentrant	sbit
sfr	sfr16	small
task	using	xdata

Data Types in Embedded C

Data Types in C Programming Language (or any programming language for that matter) help us declaring variables in the program. There are many data types in C Programming Language like signed int, unsigned int, signed char, unsigned char, float, double, etc. In addition to these there few more data types in Embedded C.

The following are the extra data types in Embedded C associated with the Keil's Cx51 Compiler.

- bit
- sbit
- sfr
- sfr16

The following table shows some of the data types in Cx51 Compiler along with their ranges.

Data Type	**Bits (Bytes)**	**Range**
bit	1	0 or 1 (bit addressable part of RAM)
signed int	16 (2)	-32768 to +32767
unsigned int	16 (2)	0 to 65535
signed char	8 (1)	-128 to +127
unsigned	8 (1)	0 to 255
float	32 (4)	±1.175494E-38 to ±3.402823E+38
double	32 (4)	±1.175494E-38 to ±3.402823E+38
sbit	1	0 or 1 (bit addressable part of RAM)
sfr	8 (1)	RAM Addresses (80h to FFh)
sfr16	16 (2)	0 to 65535

Basic Structure of an Embedded C Program (Template for Embedded C Program)

The next thing to understand in the Basics of Embedded C Program is the basic structure or Template of Embedded C Program. This will help us in understanding how an Embedded C Program is written.

The following part shows the basic structure of an Embedded C Program.

- Multiline Comments . . . . . Denoted using /*……*/

- Single Line Comments . . . . . Denoted using //
- Preprocessor Directives . . . . . #include<…> or #define
- Global Variables . . . . . Accessible anywhere in the program
- Function Declarations . . . . . Declaring Function
- Main Function . . . . . Main Function, execution begins here
  {
  Local Variables . . . . . Variables confined to main function
  Function Calls . . . . . Calling other Functions
  Infinite Loop . . . . . Like while(1) or for(;;)
  Statements . . . . .
  ….
  ….
  }
- Function Definitions . . . . . Defining the Functions
  {
  Local Variables . . . . . Local Variables confined to this Function
  Statements . . . . .
  ….
  ….
  }

Before seeing an example with respect to 8051 Microcontroller, we will first see the different components in the above structure.

2.2.7.3.Different Components of an Embedded C Program

Comments: Comments are readable text that are written to help us (the reader) understand the code easily. They are ignored by the compiler and do not take up any memory in the final code (after compilation).

There are two ways you can write comments: one is the single line comments denoted by // and the other is multiline comments denoted by /*….*/.

Preprocessor Directive: A Preprocessor Directive in Embedded C is an indication to the compiler that it must look in to this file for symbols that are not defined in the program.

Global Variables: Global Variables, as the name suggests, are Global to the program i.e., they can be accessed anywhere in the program.

Local Variables: Local Variables, in contrast to Global Variables, are confined to their respective function.

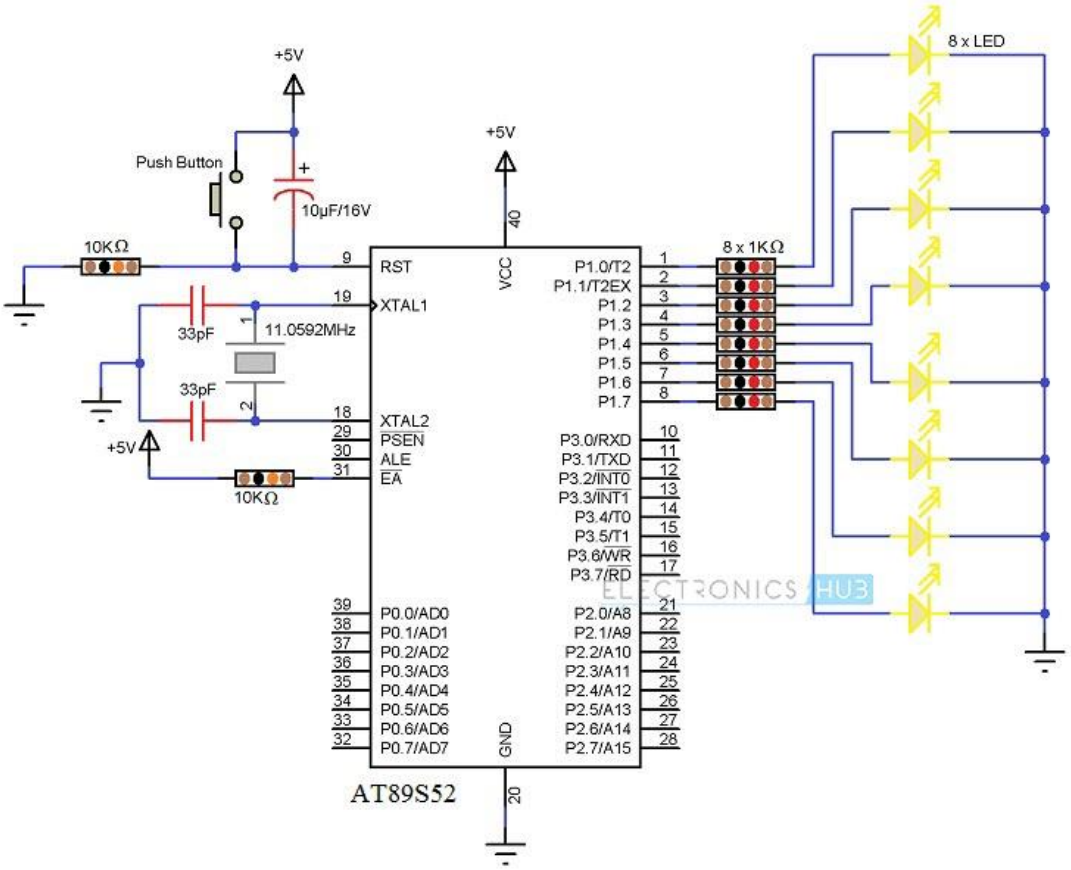
Main Function: Every C or Embedded C Program has one main function, from where the execution of the program begins.

Basic Embedded C Program

Till now, we have seen a few Basics of Embedded C Program like difference between C and Embedded C, basic structure or template of an Embedded C Program and different components of the Embedded C Program.

Example of Embedded C Program

The following image shows the circuit diagram for the example circuit. It contains an 8051 based Microcontroller (AT89S52) along with its basic components (like RESET Circuit, Oscillator Circuit, etc.) and components for blinking LEDs (LEDs and Resistors).



In order to write the Embedded C Program for the above circuit, we will use the Keil C Compiler. This compiler is a part of the Keil µVision IDE. The program is shown below.

```
#include<reg51.h> // Preprocessor Directive
void delay (int); // Delay Function Declaration
void main(void) // Main Function
{
P1 = 0x00;
/* Making PORT1 pins LOW. All the LEDs are OFF.
 * (P1 is PORT1, as defined in reg51.h) */
        while(1) // infinite loop
        {
```

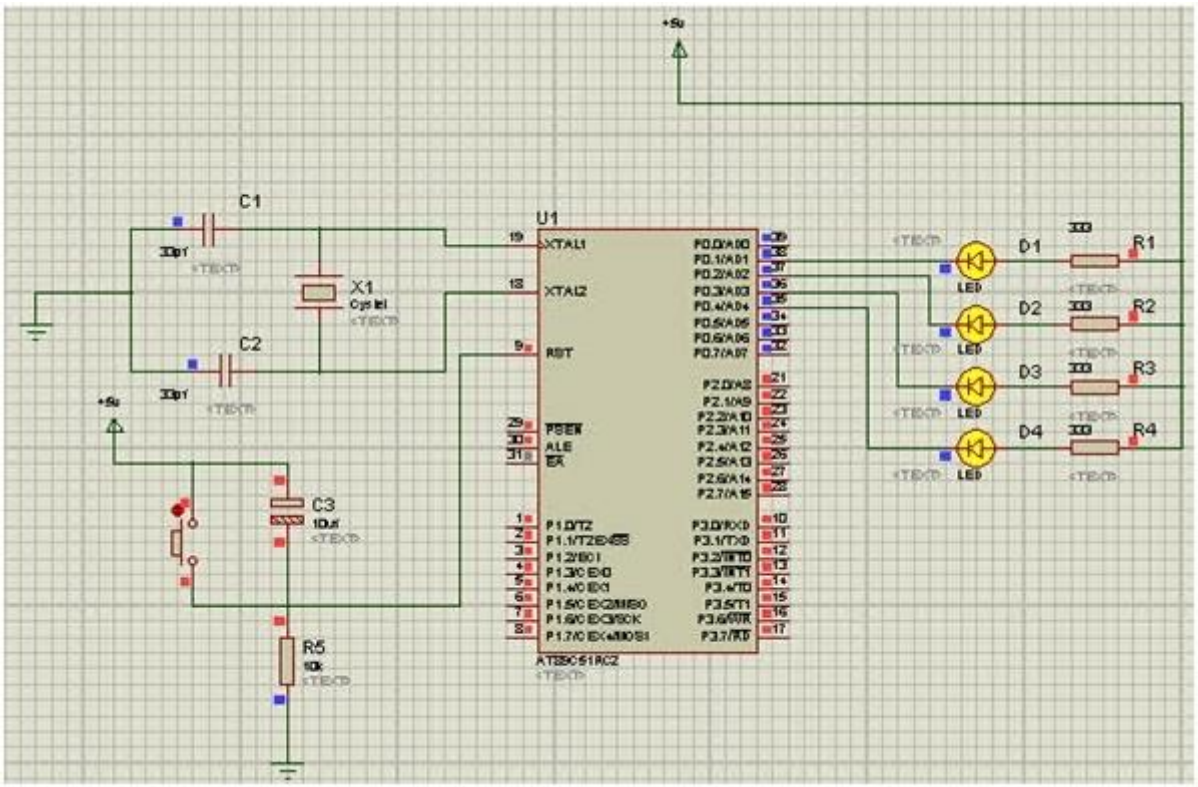
```
            P1 = 0xFF; // Making PORT1 Pins HIGH i.e. LEDs are ON.
            delay(1000);
            /* Calling Delay function with Function parameter as 1000.
             * This will cause a delay of 1000mS i.e. 1 second */
            P1 = 0x00; // Making PORT1 Pins LOW i.e. LEDs are OFF.
            delay(1000);
        }
}
        void delay (int d) // Delay Function Definition
        {

            unsigned int i=0; // Local Variable. Accessible only in this function.
             /* This following step is responsible for causing delay of 1000mS
             * (or as per the value entered while calling the delay function) */
            for(; d>0; d–)
            {
            for(i=250; i>0; i – -);
            for(i=248; i>0; i – -);
            }
        }
```

2.2.7.4. LED Blinking using 8051 Microcontroller

LED is a semiconductor device used in many electronic devices, mostly used for indication purposes. It is used widely as indicator during test for checking the validity of results at different stages.

It is very cheap and easily available in variety of shape, color and size. The LEDs are also used in designing of message display boards and traffic control signal lights etc.

Consider the Proteus Software based simulation of LED blinking using 8051 Microcontroller is shown below:-



In above Proteus based simulation the LEDs are interfaced to the PORT0 of the 8051 microcontroller.

Let's see the Embedded C Program for generating the LED output sequence as shown below:

00000001
00000010
00000100.....
.... And so on up to 10000000.

```
#include<reg51.h>
void main()
{
unsigned int k;
 unsigned char l,b;
 while(1)
 {
        P0=0x01;
        b=P0;
        for(l-0;l<3000;l++);
        for(k=0;k<8;k++)
        {
                b=b<<1;
                P0=b;
        }
 }
```

```
}
```

Consider the Embedded C Program for generating the LED output sequence as shown below is:-

00000001
00000011
00000111.....
.... And so on up to 11111111.

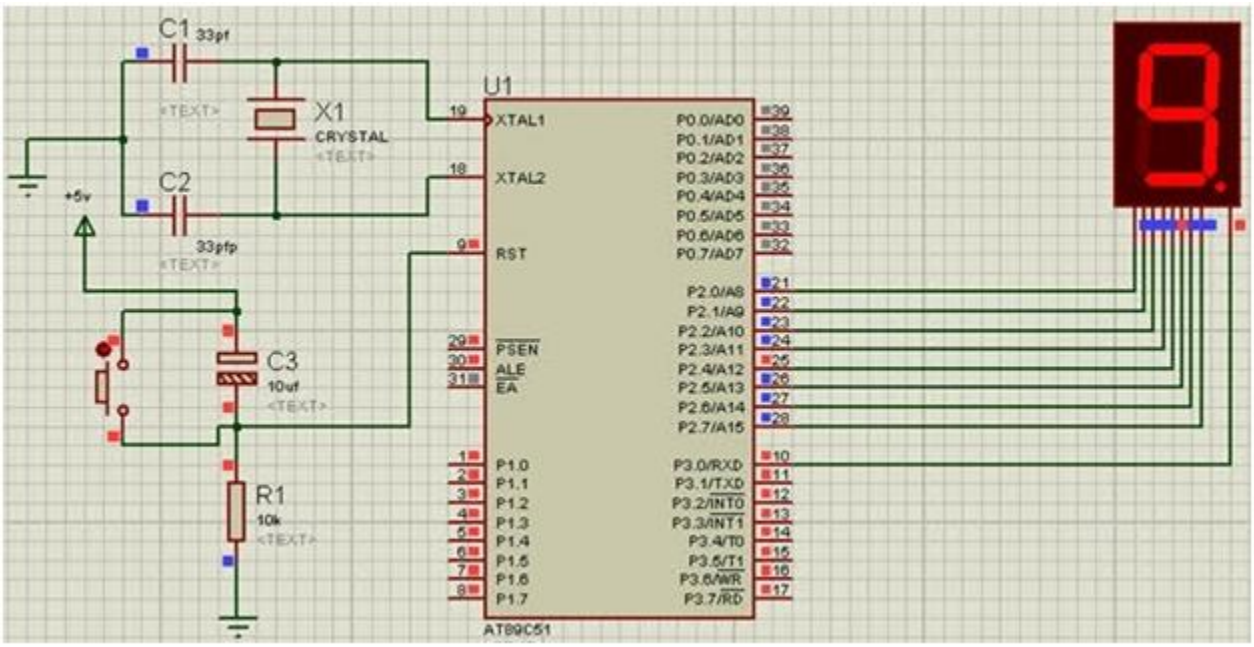
```
#include<reg51.h>
void main()
{
        unsigned int i;
        unsigned char j,b;
        while(1)
        {
                P0=0x01;
                b=P0;
                for(j-0;j<3000;j++);
                for(j=0;j<8;j++)
                {
                        bb=b<<1;
                        b=0x01;
                        P0=b;
                }
        }
}
```

Displaying Number on 7-Segment Display using 8051 Microcontroller

Electronic display used for displaying alphanumeric character is known as 7-Segment display it is used in many systems for displaying the information.

It is constructed using eight LEDs which are connected in sequential way so as to display digits from 0 to 9, when certain combinations of LEDs are switched on. It displays only one digit at a time.

Consider the Proteus software based simulation of displaying number on 7-segment display using 8051 microcontroller is:-



Consider the program for displaying the number from '0 to F' on 7-segment display is:-

10s

```
#include<reg51.h>
sbit a= P3^0;
sbit x= P3^1;
sbit y= P3^2;
sbit z= P3^3;
void main()
{
        unsigned char m[10]={0?40,0xF9,0?24,0?30,0?19,0?12,0?02,0xF8,0xE00,0?10};
        unsigned int i,j;
        a=x=y=z=1;
        while(1)
        {
                for(i=0;i<10;i++)
                {
                        P2=m[i];
                        for(j=0;j<60000;j++);
                }
        }
}
```

Consider the program for displaying numbers from '00 to 10' on a 7segment display is:-

```
#include<reg51.h>
sbit x= P3^0;
sbit y= P3^1;
void display1();
void display2();
void delay();
```

```
void main()
{
	unsigned char m[10]={0?40,0xF9,0?24,0?30,0?19,0?12,0?02,0xF8,0xE00,0?10};
	unsigned int i,j;
	ds1=ds2=0;
	while(1)
	{
		for(i=0,i<20;i++)
		display1();
		display2();
	}
}
void display1()
{
	x=1;
	y=0;
	P2=m[ds1];
	delay();
	x=1;
	y=0;
	P2=m[ds1];
	delay();
}
void display2()
{
	ds1++;
	if(ds1>=10)
	{
		ds1=0;
		ds2++;
		if(ds2>=10)
		{
			ds1=ds2=0;
		}
	}
}
void delay()
{
	unsigned int k;
	for(k=0;k<30000;k++);
}
```

2.3. RTOS (Real-Time Operating Systems for Embedded Developers)

Embedded developers are often accustomed to bare metal programming or have reservations towards using an RTOS. Here's what they are, and why you should consider using one.Today's product development cycles are becoming increasingly complex. With available development time shrinking yet the required feature set expanding, busy developers need to find ways of doing more in less time. It can often make sense to use a real-time operating system (RTOS) to gain efficiencies in task management and resource sharing.

2.3.1. What is an RTOS?

Simply put, an RTOS is a piece of software designed to efficiently manage the time of a central processing unit (CPU). This is especially relevant for embedded systems when time is critical.

The key difference between an operating system such as Windows and an RTOS often found in embedded systems is the response time to external events. An ordinary OS provides a non-deterministic response to events with no guarantee with respect to when they will be processed, albeit while trying to stay responsive. The user perceiving the OS to be responsive is more important than handling underlying tasks. On the other hand, an RTOS' goal is fast and more deterministic reaction.

Developers used to OS's such as Windows or Linux will be quite familiar with the characteristics of an embedded RTOS. They are designed to run in systems with limited memory, and to operate indefinitely without the need to be reset.

Because an RTOS is designed to respond to events quickly and perform under heavy loads, it can be slower at big tasks when compared to another OS.

2.3.2. RTOS scheduling

An RTOS is valued for how quickly it can respond and in that, the advanced scheduling algorithm is the key component.

The time-criticality of embedded systems vary from soft-real time washing machine control systems through hard-real time aircraft safety systems. In situations like the latter, the fundamental demand to meet real-time requirements can only be made if the OS scheduler's behavior can be accurately predicted.

Many operating systems give the impression of executing multiple programs at once, but this multi-tasking is something of an illusion. A single processor core can only run a single thread of execution at any one time. An operating system's scheduler decides which program, or thread, to run when. By rapidly switching between threads, it provides the illusion of simultaneous multitasking.

The flexibility of an RTOS scheduler enables a broad approach to process priorities, although an RTOS is more commonly focused on a very narrow set of applications. An RTOS

scheduler should give minimal interrupt latency and minimal thread switching overhead. This is what makes an RTOS so relevant for time-critical embedded systems.

2.3.3. The use of RTOS in embedded designs

Many embedded programmers shy away from using an RTOS because they suspect that it adds too much complexity to their application, or it is simply unknown territory. An RTOS typically requires anything up to 5% of the CPU's resources to perform its duties. While there will always be some resource penalties, an RTOS can make up for it in areas such as simplified determinism, ease of use though HW abstraction, reduced development time and easier debugging.

Using an RTOS means you can run multiple tasks concurrently, bringing in the basic connectivity, privacy, security, and so on as and when you need them. An RTOS allows you to create an optimized solution for the specific requirements of your project.

2.3.4. Introducing the Zephyr RTOS

There are numerous RTOS solutions out there. Many developers in the Nordic world are focused on low power embedded systems. If that's you, we'd suggest you check out Zephyr, which is well-suited for connectivity solutions in which ultra-low power is a requirement.
The modular Zephyr RTOS supports multiple architectures, so developers are able to easily tailor a solution to meet their needs.

2.3.5. Real-time operating system (RTOS): Components, Types, Examples

Real-time operating system (RTOS) is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. The full form of RTOS is Real time operating system.

In a RTOS, Processing time requirement are calculated in tenths of seconds increments of time. It is time-bound system that can be defined as fixed time constraints. In this type of system, processing must be done inside the specified constraints. Otherwise, the system will fail.

2.3.5. Why use an RTOS?

Here are important reasons for using RTOS:

- It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing.
- The Real time OS provides API functions that allow cleaner and smaller application code.
- Abstracting timing dependencies and the task-based design results in fewer interdependencies between modules.
- RTOS offers modular task-based development, which allows modular task-based testing.

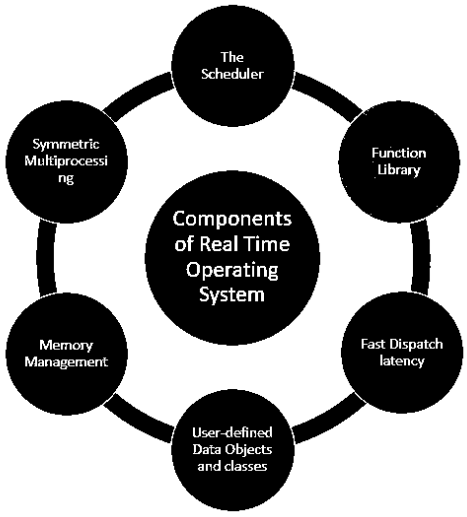
- The task-based API encourages modular development as a task, will typically have a clearly defined role. It allows designers/teams to work independently on their parts of the project.
- An RTOS is event-driven with no time wastage on processing time for the event which is not occur

2.3.6. Terms used in RTOS

Here, are essential terms used in RTOS:

- **Task –** A set of related tasks that are jointly able to provide some system functionality.
- **Job –** A job is a small piece of work that can be assigned to a processor, and that may or may not require resources.
- **Release time of a job –** It's a time of a job at which job becomes ready for execution.
- **Execution time of a job:** It is time taken by job to finish its execution.
- **Deadline of a job:** It's time by which a job should finish its execution.
- **Processors:** They are also known as active resources. They are important for the execution of a job.
- **Maximum It is the** allowable response time of a job is called its relative deadline.
- **Response time of a job:** It is a length of time from the release time of a job when the instant finishes.
- **Absolute deadline:** This is the relative deadline, which also includes its release time.

2.3.7. Components of RTOS



Components of Real Time Operating System

Here, are important Component of RTOS

The Scheduler: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.

Symmetric Multiprocessing (SMP): It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.

Function Library: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

Memory Management: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.

Fast dispatch latency: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue, that has started processing.

User-defined data objects and classes: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.

2.3.8. Features of RTOS

Here are important features of RTOS:

- Occupy very less memory
- Consume fewer resources
- Response times are highly predictable
- Unpredictable environment
- The Kernel saves the state of the interrupted task ad then determines which task it should run next.
- The Kernel restores the state of the task and passes control of the CPU for that task

Factors for selecting an RTOS

Here, are essential factors that you need to consider for selecting RTOS:

- **Performance**: Performance is the most important factor required to be considered while selecting for a RTOS**.**
- **Middleware**: if there is no middleware support in Real time operating system, then the issue of time-taken integration of processes occurs.
- **Error-free**: RTOS systems are error-free. Therefore, there is no chance of getting an error while performing the task.
- **Embedded system usage**: Programs of RTOS are of small size. So we widely use RTOS for embedded systems.
- **Maximum Consumption**: we can achieve maximum Consumption with the help of RTOS.
- **Task shifting**: Shifting time of the tasks is very less.
- **Unique features**: A good RTS should be capable, and it has some extra features like how it operates to execute a command, efficient protection of the memory of the system, etc.
- **24/7 performance**: RTOS is ideal for those applications which require to run 24/7.

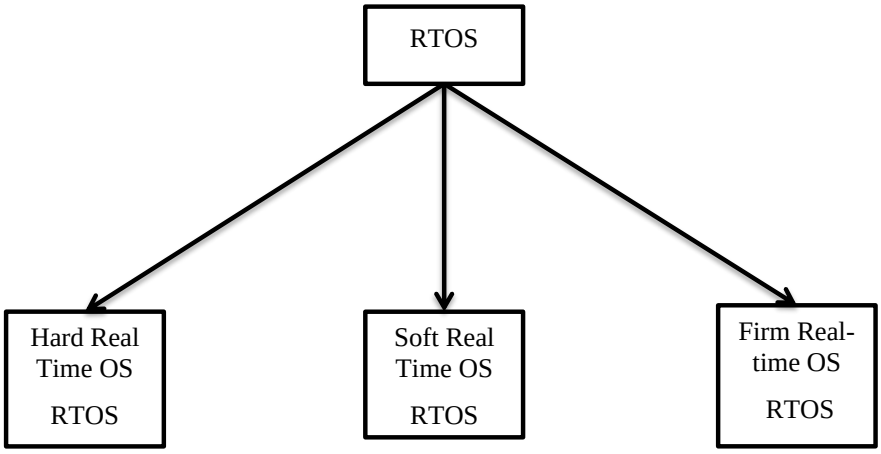
2.3.9. Difference between in GPOS and RTOS

Here are important differences between GPOS and RTOS:

General-Purpose Operating System (GPOS)	Real-Time Operating System (RTOS)
It used for desktop PC and laptop.	It is only applied to the embedded application.
Process-based Scheduling.	Time-based scheduling used like round-robin scheduling.
Interrupt latency is not considered as important as in RTOS.	Interrupt lag is minimal, which is measured in a few microseconds.
No priority inversion mechanism is present in the system.	The priority inversion mechanism is current. So it can not modify by the system.
Kernel's operation may or may not be preempted.	Kernel's operation can be preempted.
Priority inversion remain unnoticed	No predictability guarantees

2.3.10. Types of Real Time Operating System (RTOS)

The real-time operating systems can be of 3 types –



1. **Hard Real-Time operating system:**
   These operating systems guarantee that critical tasks be completed within a range of time.
   For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by robot hardly on the time., scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

2. **Soft real-time operating system:**
   This operating system provides some relaxation in the time limit.
   For example – Multimedia systems, digital audio systems etc. Explicit, programmer-defined and controlled processes are encountered in real-time systems. A separate process is changed with handling a single external event. The process is activated upon occurrence of the related event signalled by an interrupt.

Multitasking operation is accomplished by scheduling processes for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is allocated to the highest priority processes. This type of schedule, called, priority-based preemptive scheduling is used by real-time systems.

3. **Firm Real-time Operating System**:
   RTOS of this type have to follow deadlines as well. In spite of its small impact, missing a deadline can have unintended consequences, including a reduction in the quality of the product. Example: Multimedia applications.

2.3.11. Advantages of Real Time Operating System (RTOS)

The advantages of real-time operating systems are as follows-

1. **Maximum consumption**
   Maximum utilization of devices and systems. Thus more output from all the resources.

2. **Task Shifting**
   Time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds. Shifting one task to another and in the latest systems, it takes 3 microseconds.

3. **Focus On Application –**
   Focus on running applications and less importance to applications that are in the queue.

4. **Real-Time Operating System In Embedded System –**
   Since the size of programs is small, RTOS can also be embedded systems like in transport and others.

5. **Error Free –**
   These types of systems are error-free.

6. **Memory Allocation –**
   Memory allocation is best managed in these types of systems.

2.3.12. Disadvantages of Real Time Operating System (RTOS)

The disadvantages of real-time operating systems are as follows-

1. **Limited Tasks –**
   Very few tasks run simultaneously, and their concentration is very less on few applications to avoid errors.

2. **Use Heavy System Resources –**
   Sometimes the system resources are not so good and they are expensive as well.

3. **Complex Algorithms –**
   The algorithms are very complex and difficult for the designer to write on.

4. **Device Driver And Interrupt signals –**
   It needs specific device drivers and interrupts signals to respond earliest to interrupts.

5. **Thread Priority –**
   It is not good to set thread priority as these systems are very less prone to switching tasks.

6. **Minimum Switching –** RTOS performs minimal task switching.

2.3.13. Comparison of Regular and Real-Time operating systems:

Regular OS	**Real-Time OS (RTOS)**
Complex	Simple
Best effort	Guaranteed response
Fairness	Strict Timing constraints
Average Bandwidth	Minimum and maximum limits
Unknown components	Components are known
Unpredictable behavior	Predictable behavior
Plug and play	RTOS is upgradeable

2.4. Multiple Tasks And Multiple Processes

2.4.1. Tasks and Processes

Many (if not most) embedded computing systems do more than one thing that is, the environment can cause mode changes that in turn cause the embedded system to behave quite differently. For example, when designing a telephone answering machine,

We can define recording a phone call and operating the user's control panel as distinct tasks, because they perform logically distinct operations and they must be performed at very different rates. These different **tasks** are part of the system's functionality, but that application-level organization of functionality is often reflected in the structure of the program as well.

A ***process*** is a single execution of a program. If we run the same program two different times, we have created two different processes. Each process has its own state that includes not only its registers but all of its memory. In some OSs, the memory management unit is

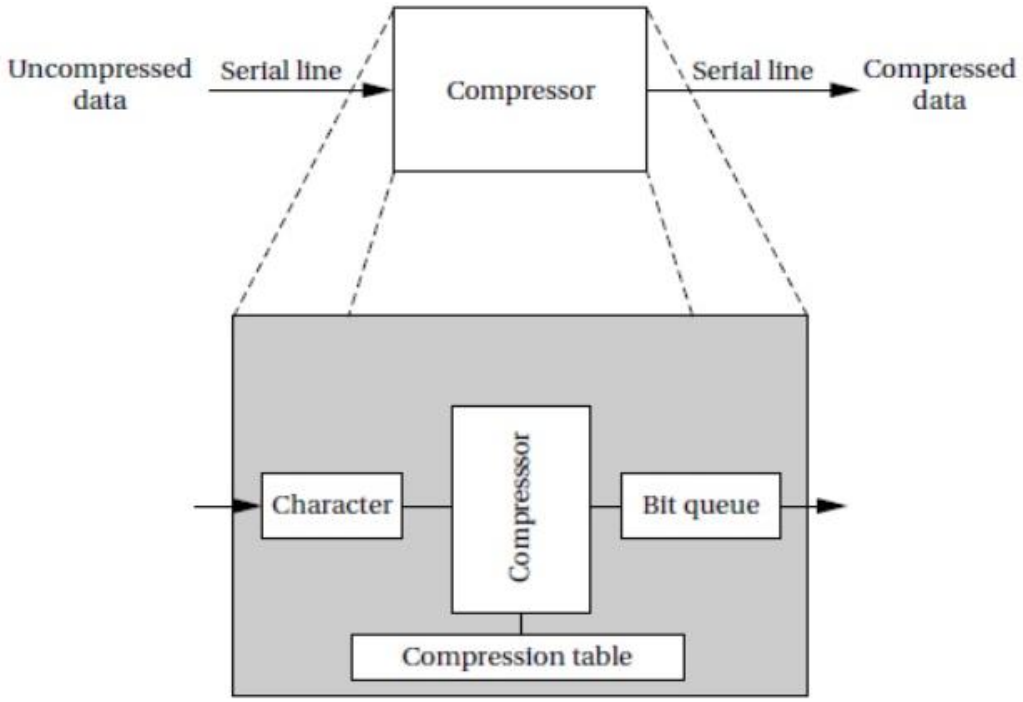
used to keep each process in a separate address space. In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called ***threads***.

As shown in Figure, this device is connected to serial ports on both ends. The input to the box is an uncompressed stream of bytes. The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem.

The program's need to receive and send data at different rates for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code. It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.

But beyond the need to create a clean data structure that simplifies the control structure of the code, we must also ensure that we process the inputs and outputs at the proper rates. For example, if we spend too much time in packaging and emitting output characters, we may drop an input character. Solving timing problems is a more challenging problem.

The text compression box provides a simple example of rate control problems. A control panel on a machine provides an example of a different type of rate control problem, the ***asynchronous input***.



The control panel of the compression box may, for example, include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled. We certainly do not know when the user will push

the compression mode button the button may be depressed asynchronously relative to the arrival of characters for compression.

2.4.2. Multirate Systems

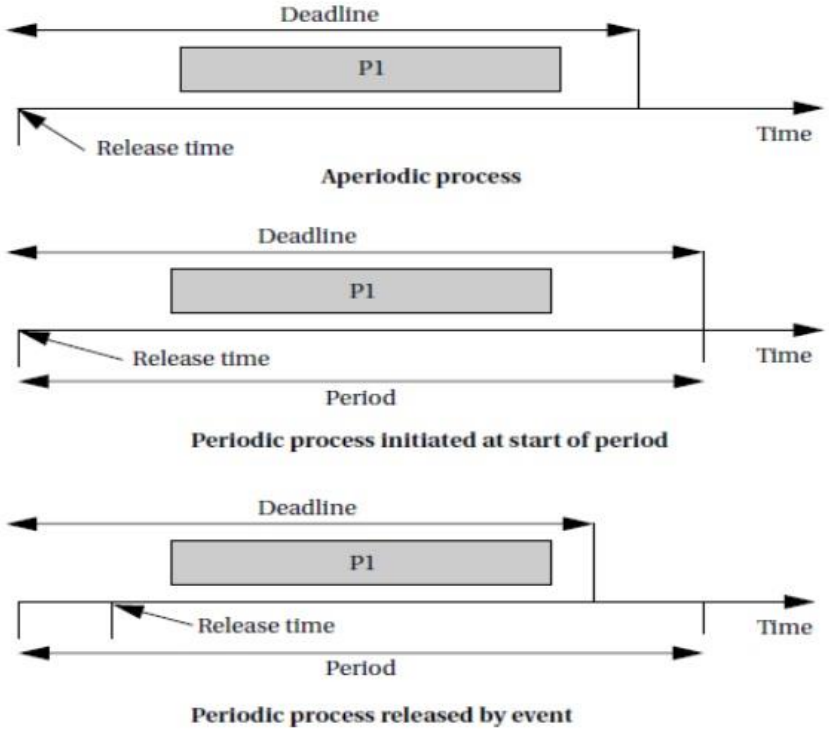
Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled. ***Multirate*** embedded computing systems are very common, including automobile engines, printers, and cell phones. In all these systems, certain operations must be executed periodically, and each operation is executed at its own rate.

2.4.3. Timing Requirements on Processes

Processes can have several different types of timing requirements imposed on them by the application. The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate. A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling proper, we outline the types of process timing requirements that are useful in embedded system design.

Figure illustrates different ways in which we can define two important requirements on processes: ***release time*** and ***deadline***.

The release time is the time at which the process becomes ready to execute; this is not necessarily the time at which it actually takes control of the CPU and starts to run. An aperiodic process is by definition initiated by an event, such as external data arriving or data computed by another process.



The release time is generally measured from that event, although the system may want to make the process ready at some interval after the event itself. For a periodically executed process, there are two common possibilities.

In simpler systems, the process may become ready at the beginning of the period. More sophisticated systems, such as those with data dependencies between processes, may set the release time at the arrival time of certain data, at a time after the start of the period.

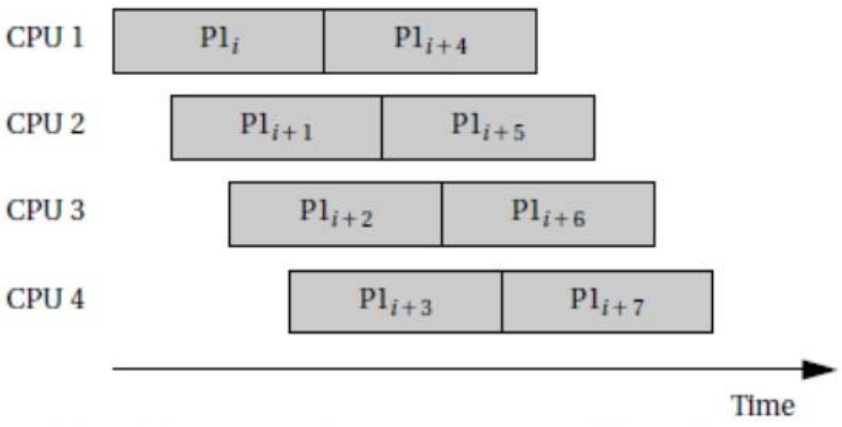
A deadline specifies when a computation must be finished. The deadline for an aperiodic process is generally measured from the release time, since that is the only reasonable time reference. The deadline for a periodic process may in general occur at some time other than the end of the period.

Rate requirements are also fairly common. A rate requirement specifies how quickly processes must be initiated.

The ***period*** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.

The process's ***rate*** is the inverse of its period. In a multirate system, each process executes at its own distinct rate.

The most common case for periodic processes is for the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period. Figure illustrates process execution in a system with four CPUs.



CPU Metrics

We also need some terminology to describe how the process actually executes. The ***initiation time*** is the time at which a process actually starts executing on the CPU. The ***completion time*** is the time at which the process finishes its work.

The most basic measure of work is the amount of ***CPU time*** expended by a process. The CPU time of process *i* is called *Ci* . Note that the CPU time is not equal to the completion time minus initiation time; several other processes may interrupt execution. The total CPU time consumed by a set of processes is

$$T = \sum Ti$$

We need a basic measure of the efficiency with which we use the CPU. The simplest and most direct measure is ***utilization***:

U=CPU time for useful work/total available CPU time

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage. If we measure the total execution time of all processes over an interval of time *t*, then the CPU utilization is

$$U = T/t.$$

2.4.4. Multiple Tasks and Multiple Processes

`Most embedded systems require functionality and timing that is too complex to body in a single program. We break the system in to multiple tasks in order to manage when things happen. In this section we will develop the basic abstractions that will be manipulated by the RTOS to build multirate systems.

To understand why these parathion of an application in to tasks may be reflected in the program structure, consider how we would build a stand-alone compression unit based on the compression algorithm,this device is connected to serial ports on both ends. The input to the box is an uncompressed stream of bytes. The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem. The program's need to receive and send data at different rates—for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte— will obviously find itself reflected in the structure of the code.

It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of out put bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets. But beyond the need to create a clean data structure that simplifies the control structure of the code, we must also ensurethat we process the inputs and outputs at the proper rates. For example, if we spend too much time in packaging and emitting output characters, we may drop an input character. Solving timing problems is a more challenging problem.

The text compression box provides a simple example of rate control problems. A control panel on a machine provides an example of a different type of rate con- troll problem, the asynchronous input. The control panel of the compression box may, for example, include a compression mode button that disables or enables com- pression, so that the input text is passed through unchanged when compression is disabled. We certainly do not know when the user will push the compression modebutton— the button may be depressed a synchronously relative to the arrival of characters for compression. We do know, however,

that the button will be depressed at a much lower rate than characters will be received, since it is not physically possible for a person to repeatedly depress a button at even slow serial line rates. Keeping up with the input and output data while checking on the button can introduce some very complex control code in to the program. Sampling the button's state too slowly can cause the machine to miss a button depression entirely, but sampling it too frequently and duplicating a data value can cause the machine to in correctly compress data.

One solution is to introduce a counter in to the main compression loop, so that a subroutine to check the input button is called once every times the compression loop is executed. But this solution does not work when either the compression loop or the button-handling routine has highly variable execution times—if the execution time of either varies significantly, it will cause the other to execute later than expected, possibly causing data to be lost. We need to be able to keep track of these two different tasks separately, applying different timing requirements to each.

This is the sort of control that processes allow. The above two examples illustrate how requirements on timing and execution rate can create major problems in programming. When code is written to satisfy several different timing requirements at once, the control structures necessary to get any sort of solution become very complex very quickly. Worse, such complex control is usually quite difficult to verify for either functional or timing properties.

2.4.5. Multi rate Systems

Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled. Multi rate embedded computing systems are very common, including auto mobile engines, printers, and cell phones.In all these systems, certain operations must be executed periodically, and each oper-ation is executed at its own rate. Application Example6.1 describes why auto mobile engines require multi rate control.

2.4.6. Timing Requirements on Processes

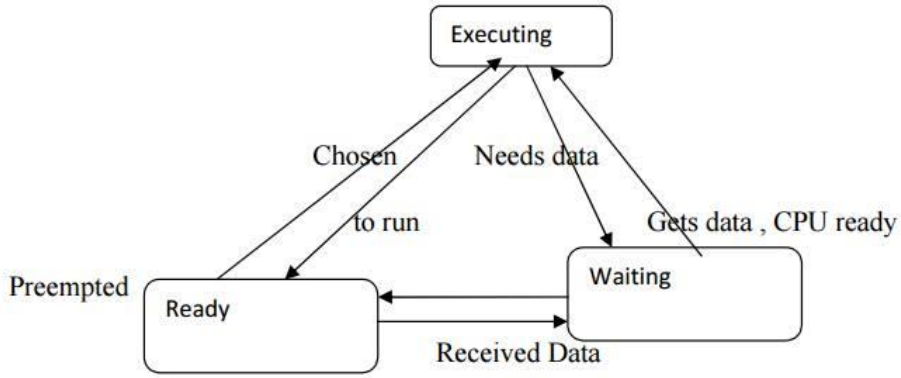
Processes can have several different types of timing requirements imposed on them by the application. The timing requirements on a set of process strongly influence the type of scheduling that is appropriate. A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling proper, we outline the types of process timing requirements that are useful in embedded system design. Figure illustrates different ways in which we can define two important requirements on processes: eg. What happens when a process misses a deadline? The practical effects of a timing violation depend on the application—the results can be catastrophic in an auto mo-tive control system, where as a missed deadline in a multimedia system may cause an audio or video glitch. The system can be designed to take a variety of actions when a deadline is missed. Safety-critical systems may try to take compensatory measures such as approximating data or switching in to a special safety mode. Systems for which safety is not as important may take simple measures to avoid propagating bad data, such as inserting silence in a phone line, or may completely ignore the failure.Even if the modules are functionally correct, their timing improper behavior can introduce major execution errors.

Application Example6.2 describes a timing problem in space shuttle software that caused the delay of the first launch of the shuttle. We need a basic measure of the efficiency with which we use the CPU. The simplest and most direct measure is utilization:

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPUtime. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage. If we measure the total execution time of all processes over an interval of time t, then the CPU utilization is U/T

2.4.7. Process State and Scheduling

The first job of the OS is to determine that process runs next. The work of choosing the order of running processes is known as scheduling. The OS considers a process to be in one of three basic scheduling states



Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements. In general, we must construct a schedule to show schedulability, but in some cases we can eliminate some sets of processes as unschedulable using some very simple tests. Utilization is one of the key metrics in evaluating a scheduling policy. Our most basic require- ment is that CPU utilization be no more than 100% since we can't use the CPU more than100% of the time.

When we evaluate the utilization of the CPU, we generally do so over a finite period that covers all possible combinations of process executions. For periodic processes, the length of time that must be considered is the hyper period, which is the least-common multiple of the periods of all the processes. If we evaluate the hyper period, we are sure to have considered all possible combinations of the periodic processes.

2.4.8. Running Periodic Processes

We need to find a programming technique that allows us to run periodic processes, ideally at different rates. For the moment, let's think of a process as a subroutine; we will call the mp1(), p2(), etc. for simplicity. Our goal is to run these subroutines at rates determined by the system

designer. Here is a very simple program that runs our process subroutines repeatedly: A timer is a much more reliable way to control execution of the loop. We would probably use the timer to generate periodic interrupts. Let's assume for the moment that the pall() function is called by the timer's interrupt handler. Then this code will execute each process once after a timer interrupt:

```
voidpall()
{
        p1();
        p2();
 }
```

But what happens when a process runs too long? The timer's interrupt will cause the CPU's interrupt system to mask its interrupts, so the interrupt will not occur until after the pall() routine returns. As a result, the next iteration will start late. This is a serious problem, but we will have to wait for further refinements before we can fix it.

Our next problem is to execute different processes at different rates. If we have several timers, we can set each timer to a different rate. We could then use a function to collect all the processes that run at that rate:

```
voidpA()
{
        /*processesthatrunatrateA*/
         p1();
         p3();
}
voidpB()
{
 /*processesthatrunatrateB*/
 }
```

This solution allows us to execute processes at rates that are simple multiples of each other. However, when the rates are n't related by a simple ratio, the counting process becomes more complex and more likely to contain bugs. We have developed somewhat more reliable code, but this programming style is still limited in capability and prone to bugs. To improve both the capabilities and reliability of our systems, we need to invent the RTOS.

2.5. Context Switch in Operating System

An operating system is a program loaded into a system or computer. and manage all the other program which is running on that OS Program, it manages the all other application programs. or in other words, we can say that the OS is an interface between the user and computer hardware.

we will learn about what is Context switching in an Operating System and see how it works also understands the triggers of context switching and an overview of the Operating System.

Context Switch

Context switching in an operating system involves saving the context or state of a running process so that it can be restored later, and then loading the context or state of another. process and run it.

Context Switching refers to the process/method used by the system to change the process from one state to another using the CPUs present in the system to perform its job.

Example – Suppose in the OS there (N) numbers of processes are stored in a Process Control Block(PCB). like The process is running using the CPU to do its job. While a process is running, other processes with the highest priority queue up to use the CPU to complete their job.

2.5.1. The Need for Context Switching

Context switching enables all processes to share a single CPU to finish their execution and store the status of the system's tasks. The execution of the process begins at the same place where there is a conflict when the process is reloaded into the system.

The operating system's need for context switching is explained by the reasons listed below.

1. One process does not directly switch to another within the system. Context switching makes it easier for the operating system to use the CPU's resources to carry out its tasks and store its context while switching between multiple processes.
2. Context switching enables all processes to share a single CPU to finish their execution and store the status of the system's tasks. The execution of the process begins at the same place where there is a conflict when the process is reloaded into the system.
3. Context switching enables all processes to share a single CPU to finish their execution and store the status of the system's tasks. The execution of the process begins at the same place where there is a conflict when the process is reloaded into the system.
4. Context switching only allows a single CPU to handle multiple processes requests parallelly without the need for any additional processors.

2.5.2. Context Changes as a Trigger

The three different categories of context-switching triggers are as follows.

1. Interrupts
2. Multitasking
3. User/Kernel switch

Interrupts: When a CPU requests that data be read from a disc, if any interruptions occur, context switching automatically switches to a component of the hardware that can handle the interruptions more quickly.

Multitasking: The ability for a process to be switched from the CPU so that another process can run is known as context switching. When a process is switched, the previous state is retained so that the process can continue running at the same spot in the system.

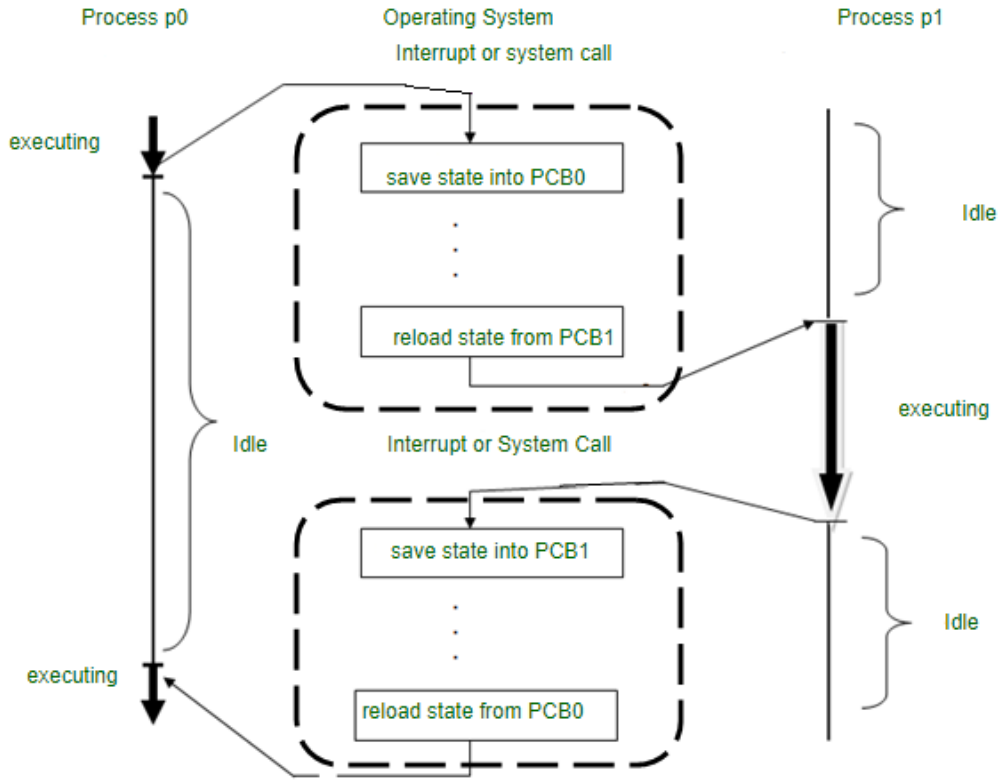
Kernel/User Switch: This trigger is used when the OS needed to switch between the user mode and kernel mode.

When switching between user mode and kernel/user mode is necessary, operating systems use the kernel/user switch.

2.5.3. Process Control Block

The Process Control block(PCB) is also known as a Task Control Block. it represents a process in the Operating System. A process control block (PCB) is a data structure used by a computer to store all information about a process. It is also called the descriptive process. When a process is created (started or installed), the operating system creates a process manager.

2.5.3.1. State Diagram of Context Switching



2.5.4. Working Process Context Switching

So the context switching of two processes, the priority-based process occurs in the ready queue of the process control block. These are the following steps.

- The state of the current process must be saved for rescheduling.

- The process state contains records, credentials, and operating system-specific information stored on the PCB or switch.
- The PCB can be stored in a single layer in kernel memory or in a custom OS file.
- A handle has been added to the PCB to have the system ready to run.
- The operating system aborts the execution of the current process and selects a process from the waiting list by tuning its PCB.
- Load the PCB's program counter and continue execution in the selected process.
- Process/thread values can affect which processes are selected from the queue, this can be important.

2.6. Priority Scheduling Algorithm: Preemptive, Non-Preemptive

What is Priority Scheduling?

Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

2.6.1. Types of Priority Scheduling

Priority scheduling divided into two main types:

2.6.1.1. Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

2.6.1.2. Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy, will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

2.6.2. Characteristics of Priority Scheduling

- A CPU algorithm that schedules processes based on priority.
- It used in Operating systems for performing batch processes.
- If two jobs having the same priority are READY, it works on a FIRST COME, FIRST SERVED basis.
- In priority scheduling, a number is assigned to each process that indicates its priority level.
- Lower the number, higher is the priority.

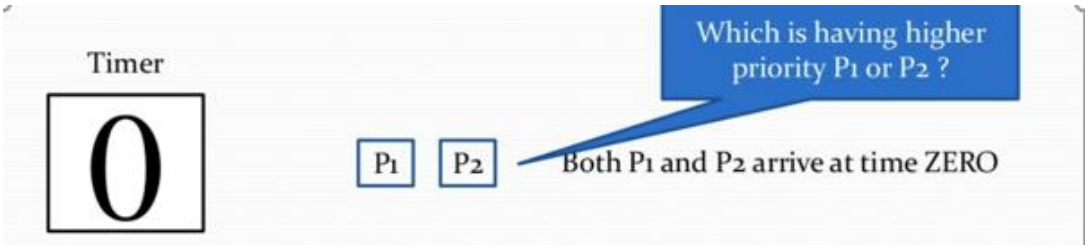
- In this type of scheduling algorithm, if a newer process arrives, that is having a higher priority than the currently running process, then the currently running process is preempted.

2.6.3. Example of Priority Scheduling

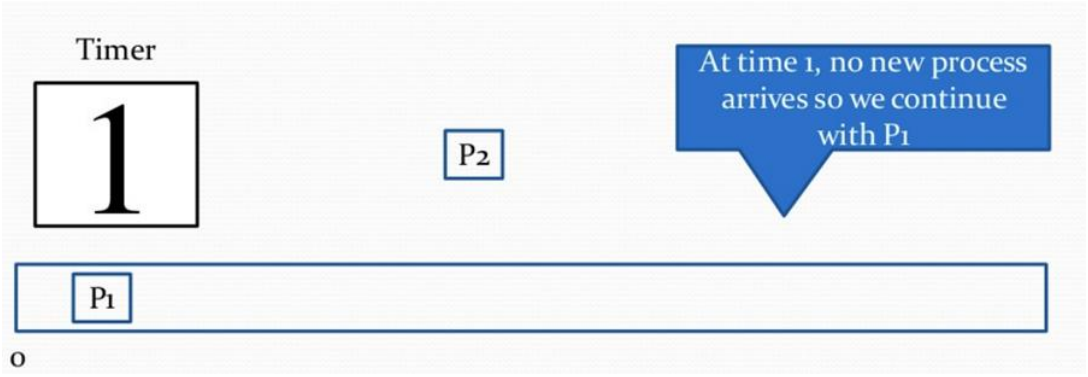
Consider following five processes P1 to P5. Each process has its unique priority, burst time, and arrival time.

Process	Priority	Burst time	Arrival time
P1	1	4	0
P2	2	3	0
P3	1	7	6
P4	3	4	11
P5	2	2	12

Step 0) At time=0, Process P1 and P2 arrive. P1 has higher priority than P2. The execution begins with process P1, which has burst time 4.

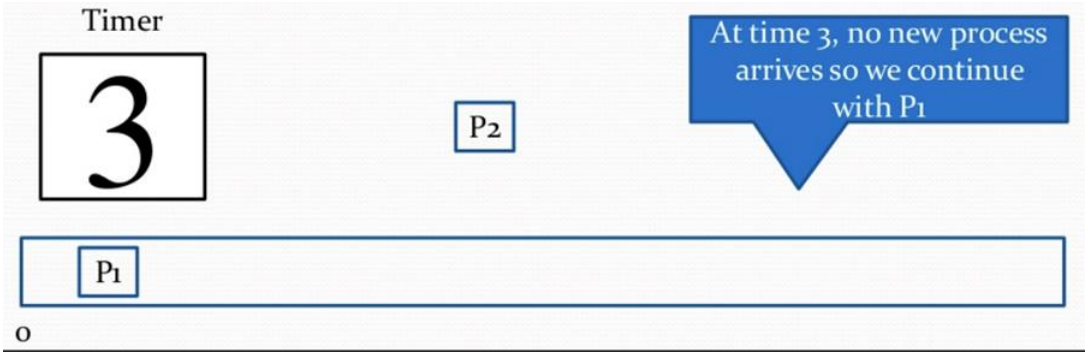


Step 1) At time=1, no new process arrive. Execution continues with P1.

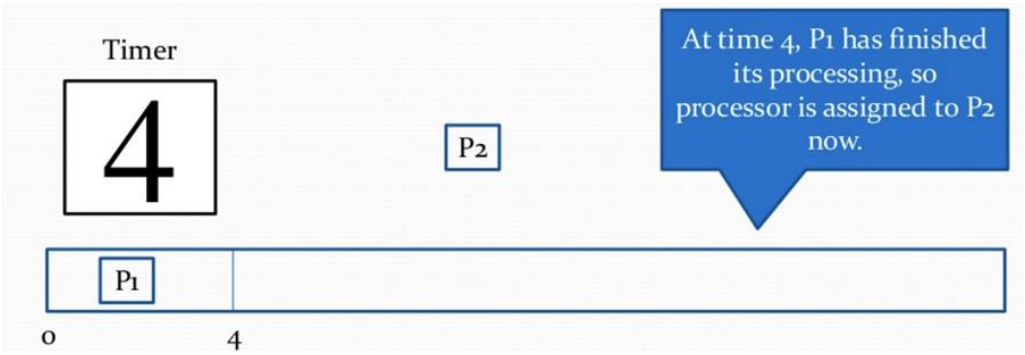


Step 2) At time 2, no new process arrives, so you can continue with P1. P2 is in the waiting queue.

Step 3) At time 3, no new process arrives so you can continue with P1. P2 process still in the waiting queue.



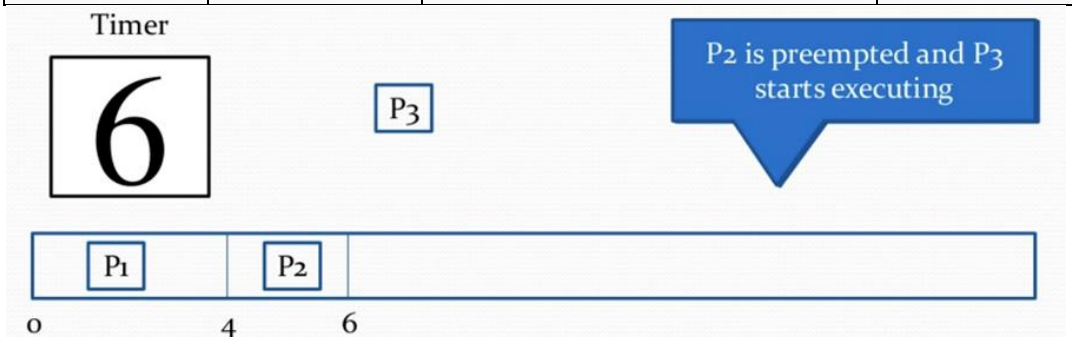
Step 4) At time 4, P1 has finished its execution. P2 starts execution.



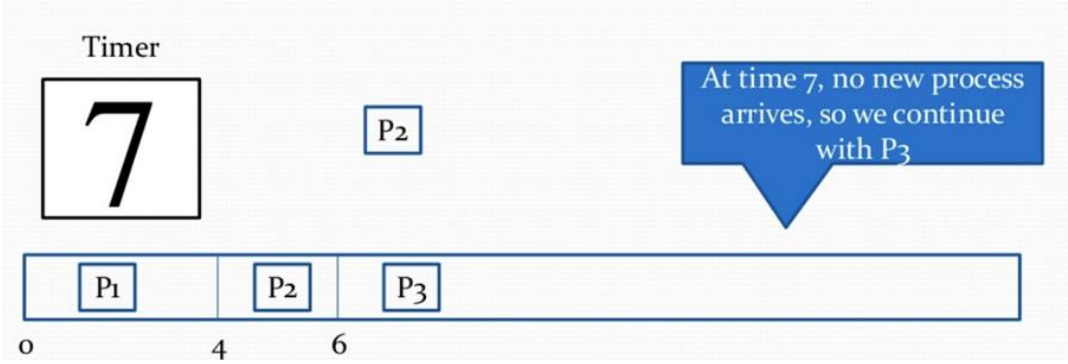
Step 5) At time= 5, no new process arrives, so we continue with P2.

Step 6) At time=6, P3 arrives. P3 is at higher priority (1) compared to P2 having priority (2). P2 is preempted, and P3 begins its execution.

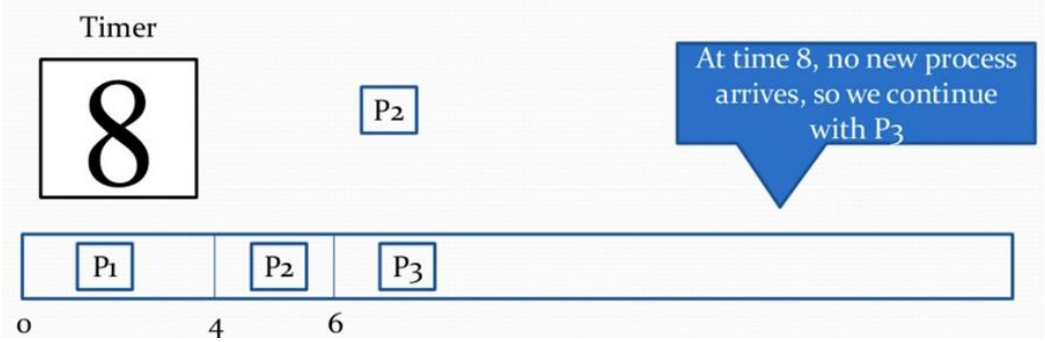
Process	Priority	Burst time	Arrival time
P1	1	4	0
P2	2	1 out of 3 pending	0
P3	1	7	6
P4	3	4	11
P5	2	2	12



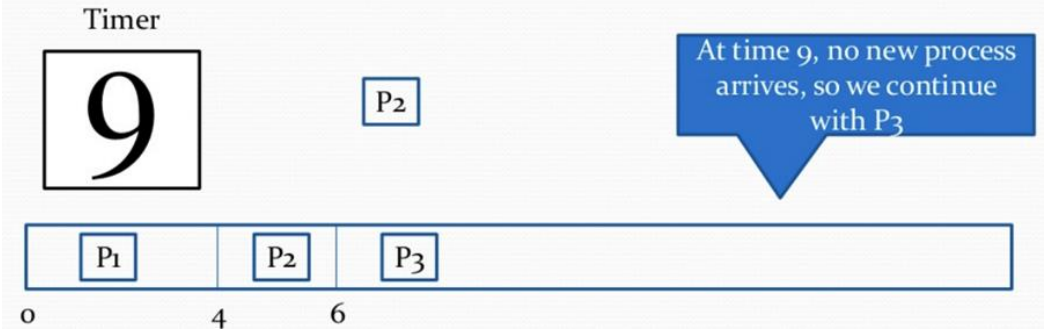
Step 7) At time 7, no-new process arrives, so we continue with P3. P2 is in the waiting queue.



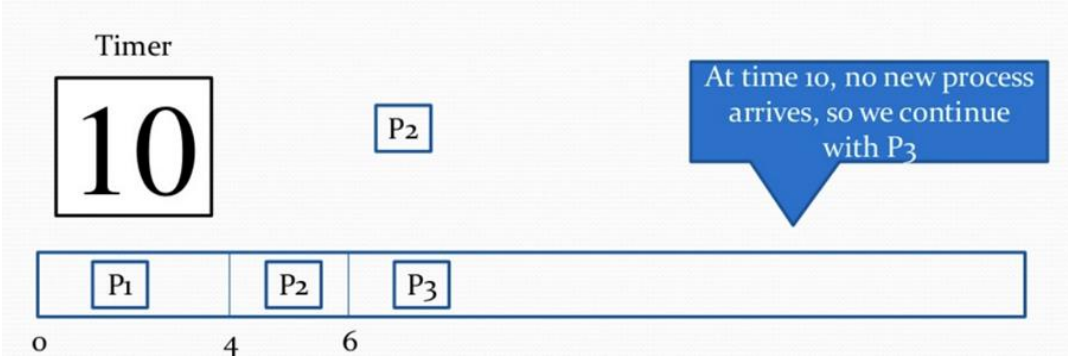
Step 8) At time= 8, no new process arrives, so we can continue with P3.



Step 9) At time= 9, no new process comes so we can continue with P3.



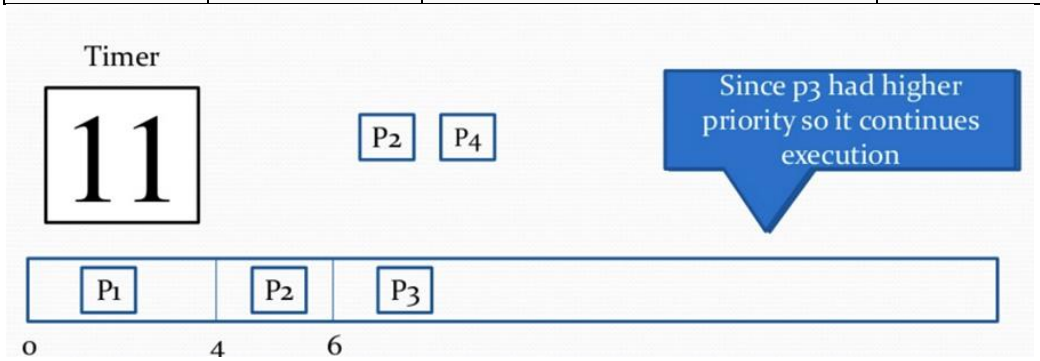
Step 10) At time interval 10, no new process comes, so we continue with P3



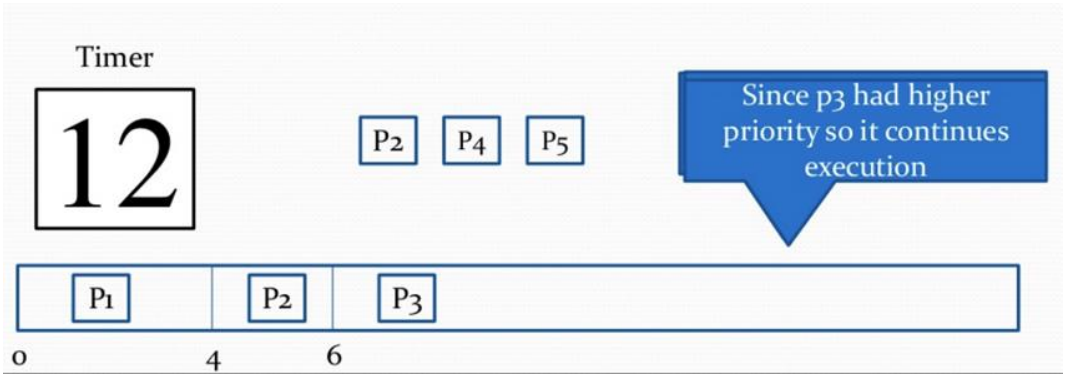
Step 11) At time=11, P4 arrives with priority 4. P3 has higher priority, so it continues its execution.

Process	**Priority**	**Burst time**	**Arrival time**
P1	1	4	0
P2	2	1 out of 3 pending	0

Process	Priority	Burst time	Arrival time
P3	1	2 out of 7 pending	6
P4	3	4	11
P5	2	2	12

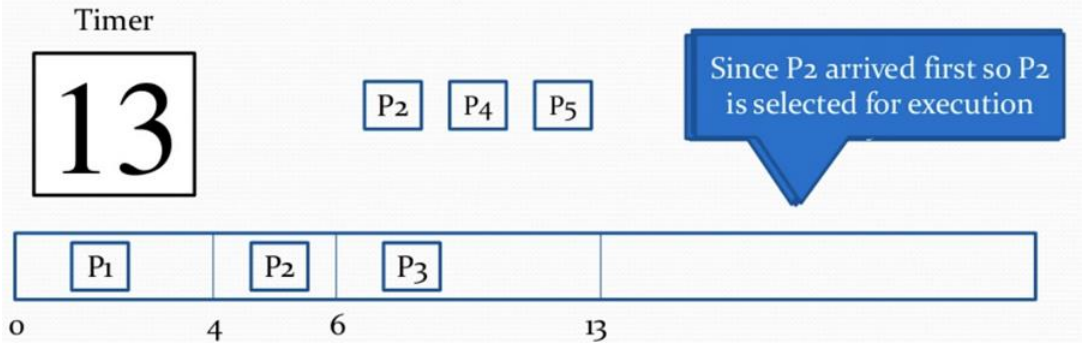


Step 12) At time=12, P5 arrives. P3 has higher priority, so it continues execution.

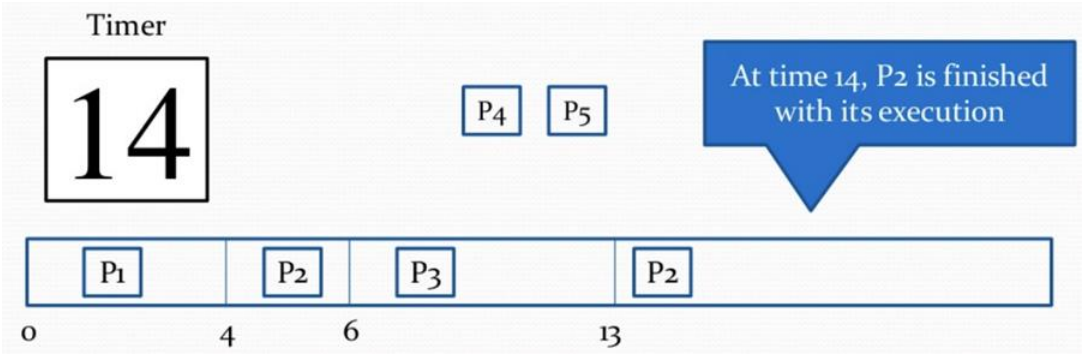


Step 13) At time=13, P3 completes execution. We have P2,P4,P5 in ready queue. P2 and P5 have equal priority. Arrival time of P2 is before P5. So P2 starts execution.

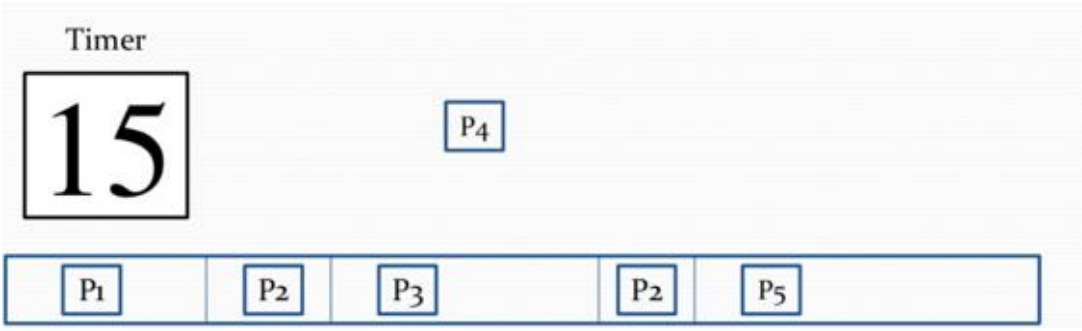
Process	Priority	Burst time	Arrival time
P1	1	4	0
P2	2	1 out of 3 pending	0
P3	1	7	6
P4	3	4	11
P5	2	2	12



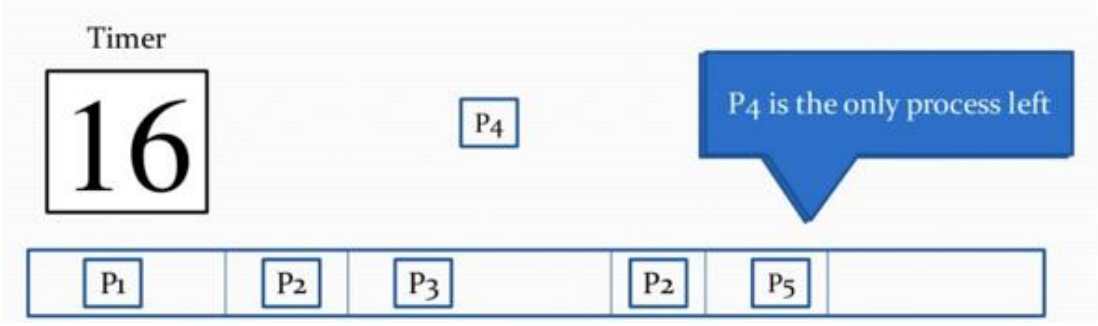
Step 14) At time =14, the P2 process has finished its execution. P4 and P5 are in the waiting state. P5 has the highest priority and starts execution.



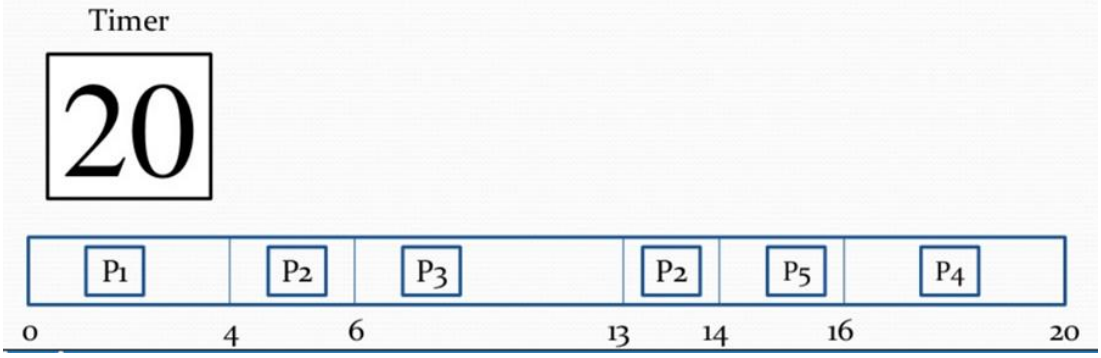
Step 15) At time =15, P5 continues execution.



Step 16) At time= 16, P5 is finished with its execution. P4 is the only process left. It starts execution.



Step 17) At time =20, P5 has completed execution and no process is left.



Step 18) Let's calculate the average waiting time for the above example.

Waiting Time = start time – arrival time + wait time for next burst

P1 = 0 - 0 = 0

P2 =4 - 0 + 7 =11

P3= 6-6=0

P4= 16-11=5

Average Waiting time = (0+11+0+5+2)/5 = 18/5= 3.6

2.6.4. Advantages of priority scheduling

Here, are benefits/pros of using priority scheduling method:

- Easy to use scheduling method
- Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
- This method provides a good mechanism where the relative important of each process may be precisely defined.
- Suitable for applications with fluctuating time and resource requirements.

2.6.5. Disadvantages of priority scheduling

Here, are cons/drawbacks of priority scheduling

- If the system eventually crashes, all low priority processes get lost.
- If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.
- This scheduling algorithm may leave some low priority processes waiting indefinitely.
- A process will be blocked when it is ready to run but has to wait for the CPU because some other process is running currently.
- If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.

2.6.6. Scheduling policies

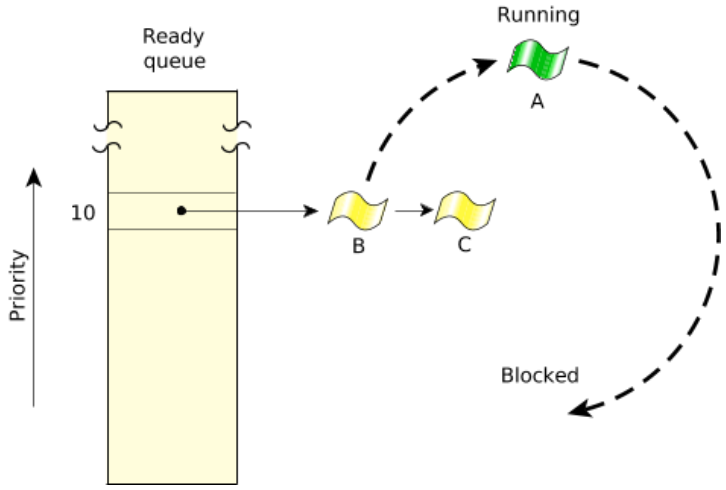
To meet the needs of various applications, the QNX Neutrino RTOS provides these scheduling algorithms:

- FIFO scheduling
- round-robin scheduling
- sporadic scheduling

Each thread in the system may run using any method. The methods are effective on a per-thread basis, not on a global basis for all threads and processes on a node.

Remember that the FIFO and round-robin scheduling policies apply only when two or more threads that share the *same priority* are READY (i.e., the threads are directly competing with each other). The sporadic method, however, employs a “budget” for a thread's execution. In all cases, if a higher-priority thread becomes READY, it immediately preempts all lower-priority threads.

In the following diagram, three threads of equal priority are READY. If Thread A blocks, Thread B will run.Although a thread inherits its scheduling policy from its parent process, the thread can request to change the algorithm applied by the kernel.



Thread A blocks; Thread B runs.

Summary:

- Priority scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.
- In Priority Preemptive Scheduling, the tasks are mostly assigned with their priorities.
- In Priority Non-preemptive scheduling method, the CPU has been allocated to a specific process.
- Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
- If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.

UNIT III

IOT AND ARDUINO PROGRAMMING

- Introduction to the Concept of IoT Devices
- IoT Devices Versus Computers
- IoT Configurations
- Basic Components
- Introduction to Arduino
- Types of Arduino
- Arduino Toolchain
- Arduino Programming Structure
- Sketches
- Pins
- Input/Output From Pins Using Sketches
- Introduction to Arduino Shields
- Integration of Sensors and Actuators with Arduino

3.1. IoT (Internet of Things)

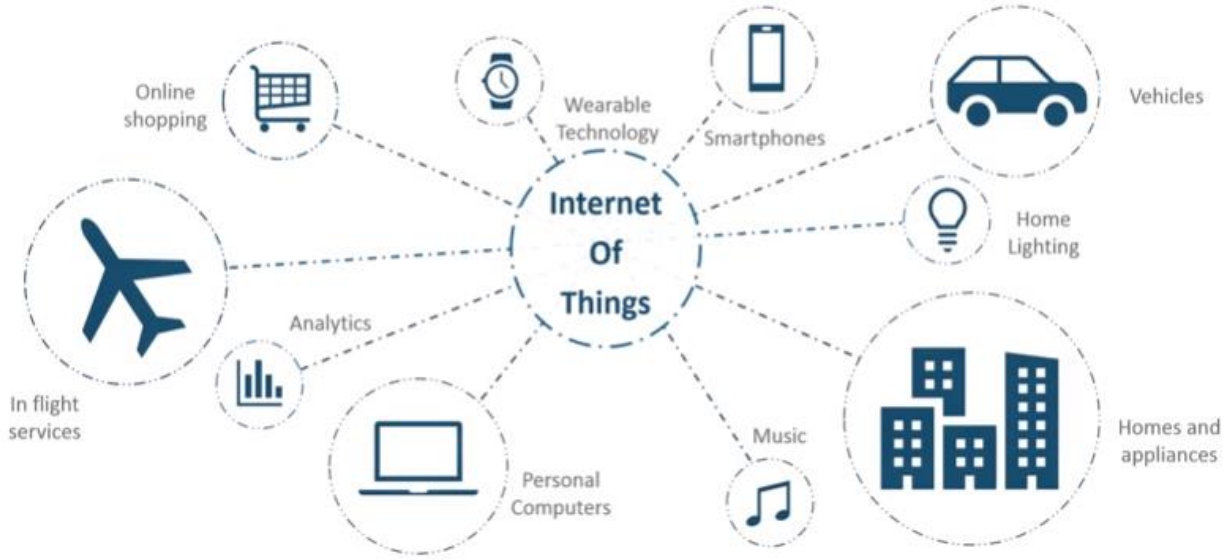
IoT stands for Internet of Things, which means accessing and controlling daily usable equipments and devices using Internet.

3.1.1 What is an Internet of Things (IoT)

Our mobile device which contains GPS Tracking, Mobile Gyroscope, Adaptive brightness, Voice detection, Face detection etc. These components have their own individual features, but what about if these all communicate with each other to provide a better environment? For example, the phone brightness is adjusted based on my GPS location or my direction.

Connecting everyday things embedded with electronics, software, and sensors to internet enabling to collect and exchange data without human interaction called as the Internet of Things (IoT).

The term "Things" in the Internet of Things refers to anything and everything in day to day life which is accessed or connected through the internet.



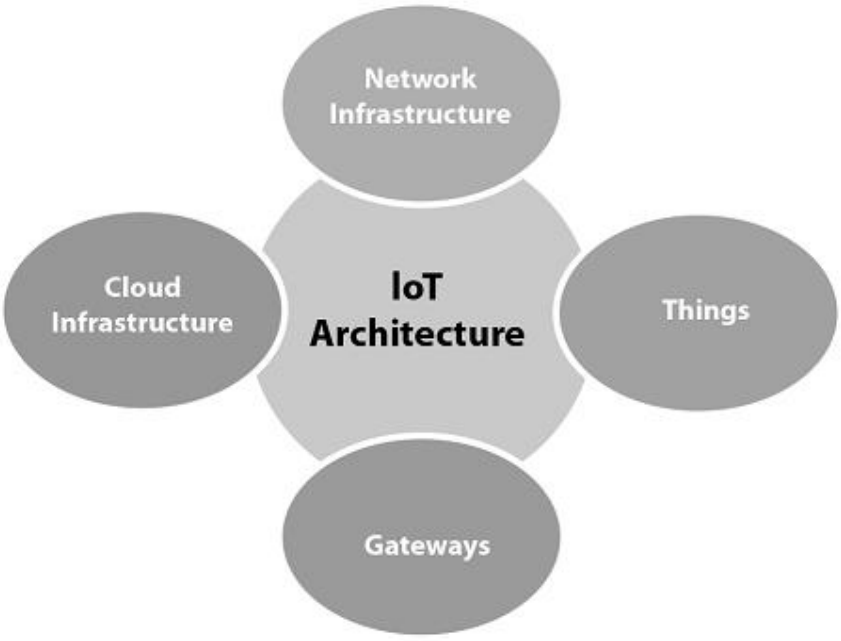
IoT is an advanced automation and analytics system which deals with artificial intelligence, sensor, networking, electronic, cloud messaging etc. to deliver complete systems for the product or services. The system created by IoT has greater transparency, control, and performance.

As we have a platform such as a cloud that contains all the data through which we connect all the things around us. For example, a house, where we can connect our home appliances such as air conditioner, light, etc. through each other and all these things are managed at the same platform. Since we have a platform, we can connect our car, track its fuel meter, speed level, and also track the location of the car.

If there is a common platform where all these things can connect to each other would be great because based on my preference, I can set the room temperature. For example, if I love the room temperature to to be set at 25 or 26-degree Celsius when I reach back home from my office, then according to my car location, my AC would start before 10 minutes I arrive at home. This can be done through the Internet of Things (IoT).

3.1.2. How does Internet of Thing (IoT) Work?

The working of IoT is different for different IoT echo system (architecture). However, the key concept of there working are similar. The entire working process of IoT starts with the device themselves, such as smartphones, digital watches, electronic appliances, which securely communicate with the IoT platform. The platforms collect and analyze the data from all multiple devices and platforms and transfer the most valuable data with applications to devices.



3.1.3. Features of IOT

The most important features of IoT on which it works are connectivity, analyzing, integrating, active engagement, and many more. Some of them are listed below:

Connectivity: Connectivity refers to establish a proper connection between all the things of IoT to IoT platform it may be server or cloud. After connecting the IoT devices, it needs a high speed messaging between the devices and cloud to enable reliable, secure and bi-directional communication.

Analyzing: After connecting all the relevant things, it comes to real-time analyzing the data collected and use them to build effective business intelligence. If we have a good insight into data gathered from all these things, then we call our system has a smart system.

Integrating: IoT integrating the various models to improve the user experience as well.

Artificial Intelligence: IoT makes things smart and enhances life through the use of data. For example, if we have a coffee machine whose beans have going to end, then the coffee machine itself order the coffee beans of your choice from the retailer.

Sensing: The sensor devices used in IoT technologies detect and measure any change in the environment and report on their status. IoT technology brings passive networks to active networks. Without sensors, there could not hold an effective or true IoT environment.

Active Engagement: IoT makes the connected technology, product, or services to active engagement between each other.

Endpoint Management: It is important to be the endpoint management of all the IoT system otherwise, it makes the complete failure of the system. For example, if a coffee machine itself order the coffee beans when it goes to end but what happens when it orders the beans from a retailer and we are not present at home for a few days, it leads to the failure of the IoT system. So, there must be a need for endpoint management.

Advantages and Disadvantages of (IoT)

Any technology available today has not reached to its 100 % capability. It always has a gap to go. So, we can say that **Internet of Things** has a significant technology in a world that can help other technologies to reach its accurate and complete 100 % capability as well.

3.1.3.1. Advantages of IoT

Internet of things facilitates the several advantages in day-to-day life in the business sector. Some of its benefits are given below:

- **Efficient resource utilization:** If we know the functionality and the way that how each device work we definitely increase the efficient resource utilization as well as monitor natural resources.
- **Minimize human effort:** As the devices of IoT interact and communicate with each other and do lot of task for us, then they minimize the human effort.
- **Save time:** As it reduces the human effort then it definitely saves out time. Time is the primary factor which can save through IoT platform.
- **Enhance Data Collection:**
- **Improve security:** Now, if we have a system that all these things are interconnected then we can make the system more secure and efficient.

3.1.3.2. Disadvantages of IoT

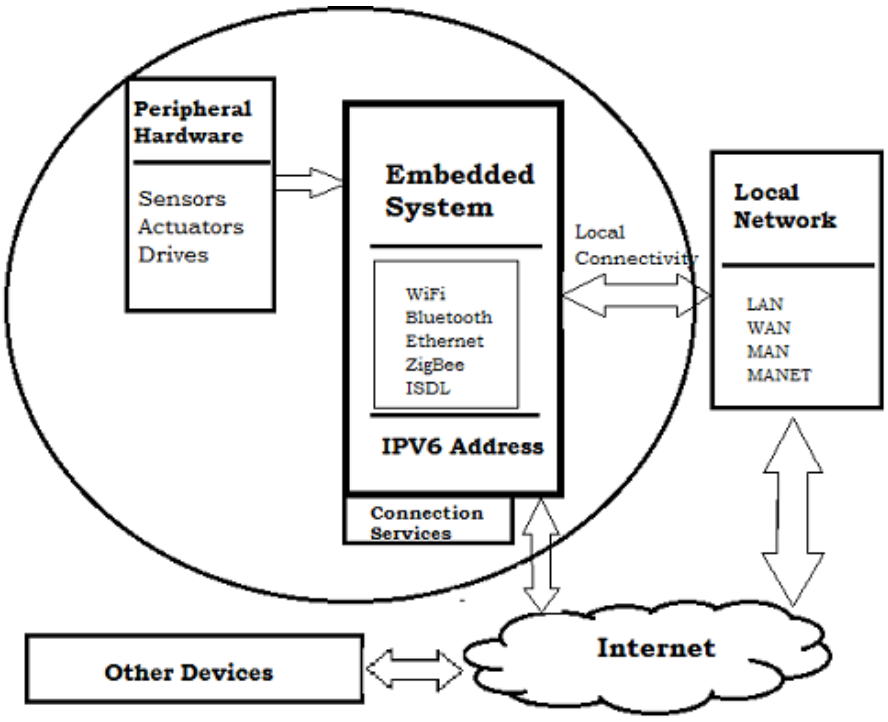
As the Internet of things facilitates a set of benefits, it also creates a significant set of challenges. Some of the IoT challenges are given below:

- **Security:** As the IoT systems are interconnected and communicate over networks. The system offers little control despite any security measures, and it can be lead the various kinds of network attacks.
- **Privacy:** Even without the active participation on the user, the IoT system provides substantial personal data in maximum detail.

- **Complexity:** The designing, developing, and maintaining and enabling the large technology to IoT system is quite complicated.

3.1.4. Embedded Devices (System) in (IoT)

It is essential to know about the embedded devices while learning the IoT or building the projects on IoT. The embedded devices are the objects that build the unique computing system. These systems may or may not connect to the Internet.



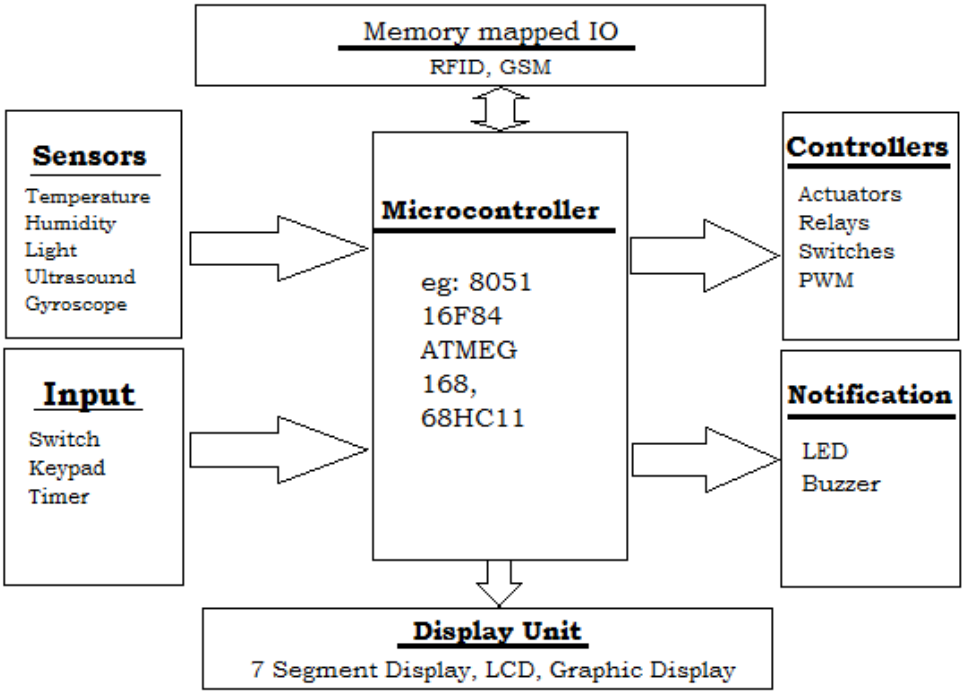
Internet of Things (IoT)

An embedded device system generally runs as a single application. However, these devices can connect through the internet connection, and able communicate through other network devices.

3.1.5. Embedded System Hardware

The embedded system can be of type microcontroller or type microprocessor. Both of these types contain an integrated circuit (IC).

The essential component of the embedded system is a RISC family microcontroller like Motorola 68HC11, PIC 16F84, Atmel 8051 and many more. The most important factor that differentiates these microcontrollers with the microprocessor like 8085 is their internal read and writable memory. The essential embedded device components and system architecture are specified below.



Basic Embedded System

3.1.6. Embedded System Software

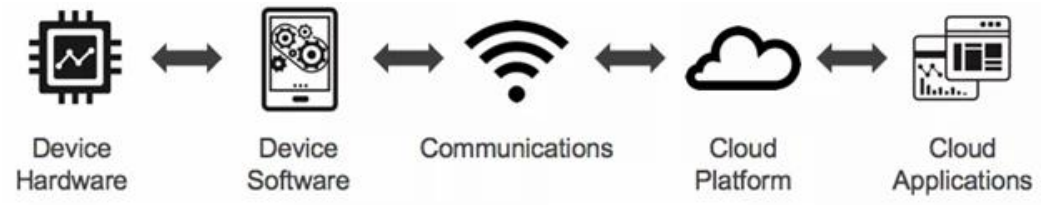
The embedded system that uses the devices for the operating system is based on the language platform, mainly where the real-time operation would be performed. Manufacturers build embedded software in electronics, e.g., cars, telephones, modems, appliances, etc. The embedded system software can be as simple as lighting controls running using an 8-bit microcontroller. It can also be complicated software for missiles, process control systems, airplanes etc.

3.1.6.1. IoT Decision Framework

The IoT decision framework provides a structured approach to create a powerful IoT product strategy. The IoT decision framework is all about the strategic decision making. The IoT Decision Framework helps us to understand the areas where we need to make decisions and ensures consistency across all of our strategic business decision, technical and more.

The IoT decision framework is much more important as the product or services communicates over networks goes through five different layers of complexity of technology.

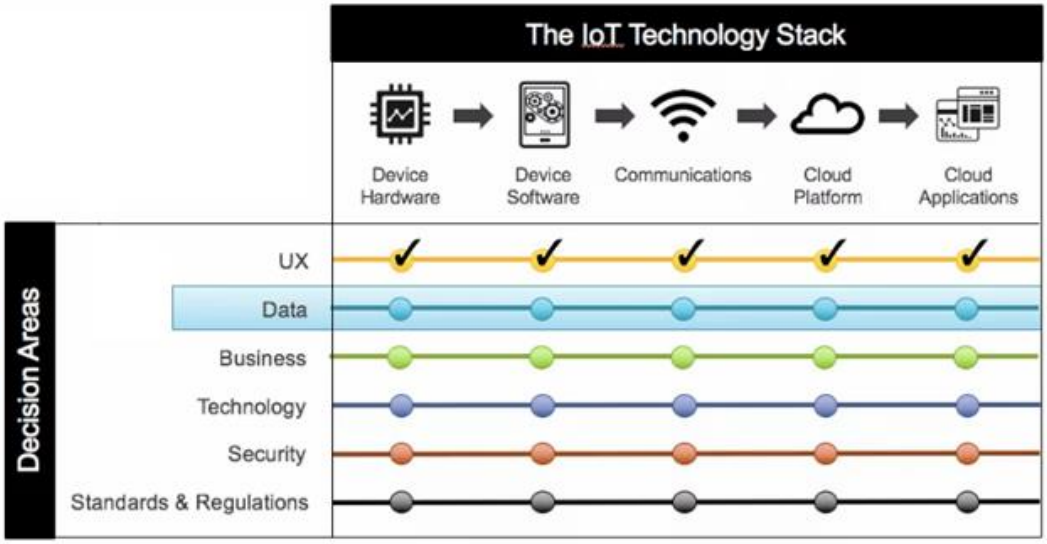
1. Device Hardware
2. Device Software
3. Communications
4. Cloud Platform
5. Cloud Application



The IoT Technology Stack

The IoT decision framework pays attention to six key decision areas in any IoT product. These decision areas are:

1. User Experience (UX)
2. Data
3. Business
4. Technology
5. Security
6. Standards & Regulations



Each of these decision areas is evaluated at each of the IoT Technology Stack. The User Experience will be evaluated at Device Hardware, Device Software and so to provide the better user experience. Then at the next step Data Decision Area, we have to explore data considerations for all the stages of IoT Technology Stack.r

Let's see each of the Decision Area of IoT Decision Framework in detail:

1. **User Experience Decision Area:** This is the area where we concentrate about who are the users, what are their requirements and how to provide a great experience at each step of IoT stack without worrying about the technical details.
2. **Data Decision Area:** In this area, we make the overall data strategy such as the data flow over the entire IoT stack to fulfill the user's requirements.
3. **Business Decision Area:** Based on the previous decisions area, we make the decision how product or services will became financial potential. At each of the IoT Stack level are monetized about the costs of providing services.

4. **Technology Decision Area:** In this area, we work with the technology for each layer to facilitate the final solution.
5. **Security Decision Area:** After going through the implementation of technology it is important to decide and provide the security at each stage of the IoT Stack.
6. **Standards & Regulations Decision Area:** At the last stage of IoT Decision Area, we identify the standards and regulations of product or services that will affect your product at each layer of the IoT Stack.

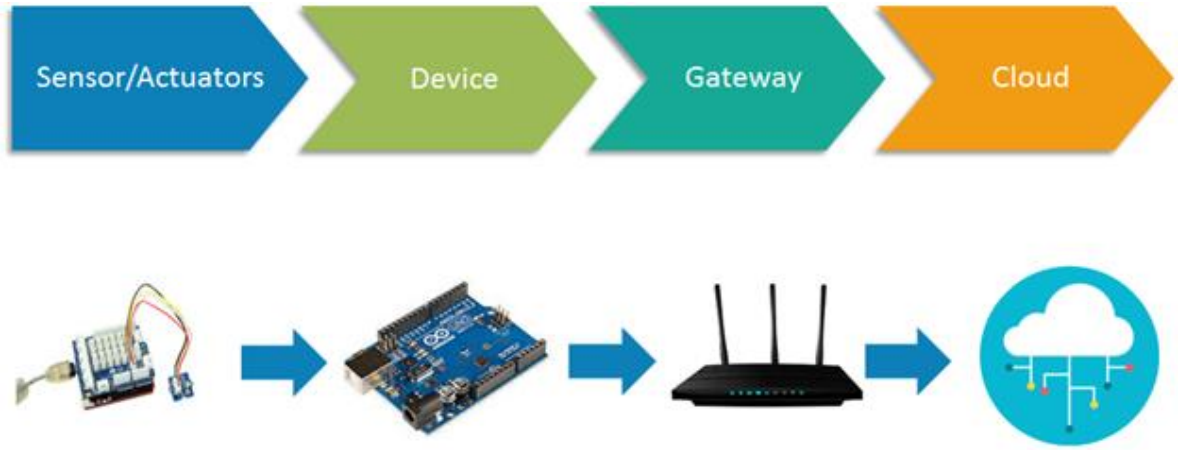
3.1.7. IoT Architecture

There is not such a unique or standard consensus on the Internet of Things (IoT) architecture which is universally defined. The IoT architecture differs from their functional area and their solutions. However, the IoT architecture technology mainly consists of four major components:

Components of IoT Architecture

- Sensors/Devices
- Gateways and Networks
- Cloud/Management Service Layer
- Application Layer

There are several layers of IoT built upon the capability and performance of IoT elements that provides the optimal solution to the business enterprises and end-users. The IoT architecture is a fundamental way to design the various elements of IoT, so that it can deliver services over the networks and serve the needs for the future.



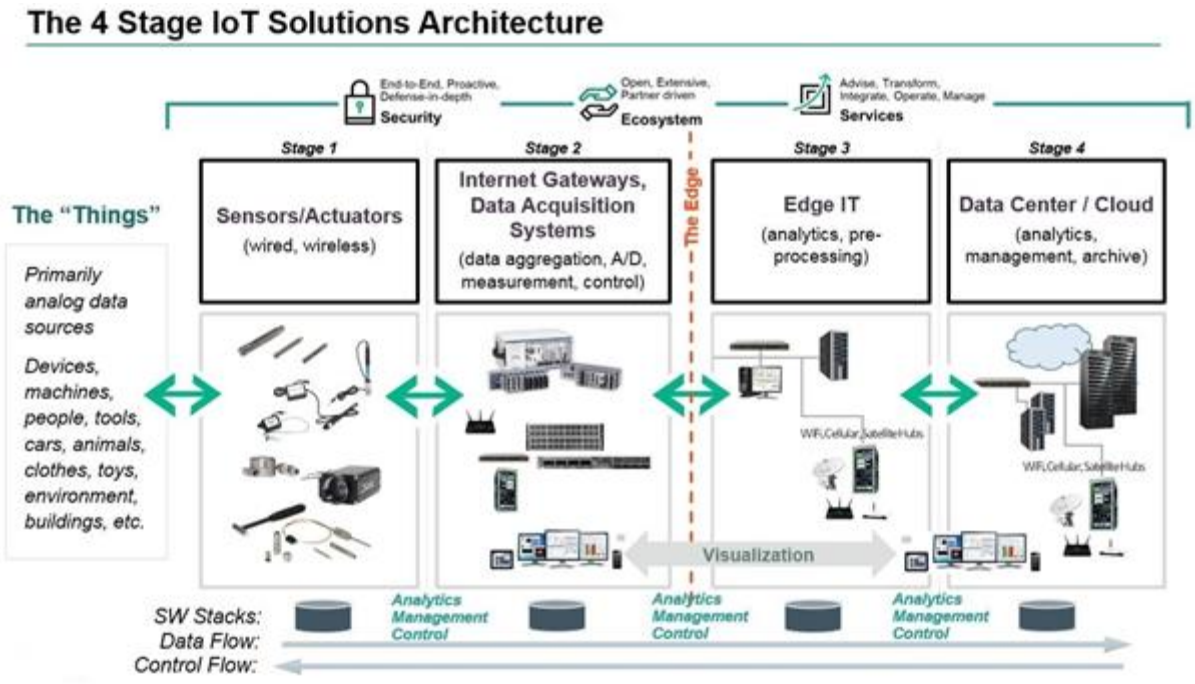
Stages of IoT Solutions Architecture

Following are the primary stages (layers) of IoT that provides the solution for IoT architecture.

1. **Sensors/Actuators:** Sensors or Actuators are the devices that are able to emit, accept and process data over the network. These sensors or actuators may be connected either through wired or wireless. This contains GPS, Electrochemical, Gyroscope, RFID, etc. Most of the sensors need connectivity through sensors gateways. The

connection of sensors or actuators can be through a Local Area Network (LAN) or Personal Area Network.

2. **Gateways and Data Acquisition:** As the large numbers of data are produced by this sensors and actuators need the high-speed Gateways and Networks to transfer the data. This network can be of type Local Area Network (LAN such as WiFi, Ethernet, etc.), Wide Area Network (WAN such as GSM, 5G, etc.).
3. **Edge IT:** Edge in the IoT Architecture is the hardware and software gateways that analyze and pre-process the data before transferring it to the cloud. If the data read from the sensors and gateways are not changed from its previous reading value then it does not transfer over the cloud, this saves the data used.
4. **Data center/ Cloud:** The Data Center or Cloud comes under the Management Services which process the information through analytics, management of device and security controls. Beside this security controls and device management the cloud transfer the data to the end users application such as Retail, Healthcare, Emergency, Environment, and Energy, etc.



3.1.8. Difference Between IoT Devices and Computers

we will discuss the overview of the Internet of Things and Computers and mainly will focus on the difference between IoT devices and Computers. Let's discuss it one by one.

Internet of Things (IoT):

The Internet of Things (IoT) is the network of physical objects/devices like vehicles, buildings, cars, and other items embedded with electronics, software, sensors, and network connectivity that enables these objects to collect and exchange data. IoT devices have made human life easier. The IoT devices like smart homes, smart cars have made the life of humans very comfortable. IoT devices are now being a part of our day-to-day life.

Computers:

A computer is a hardware device embedded with software in it. The computer does most of the work like calculations, gaming, web browsers, word processors, e-mails, etc. The main function of a computer is to compute the functions, to run the programs. It takes input from the computer and then computes/processes it and generates the output.

Function of Computer

3.2. Overview of IoT Vs Computers

One big difference between IoT devices and computers is that the main function of IoT devices is not to compute(not to be a computer) and the main function of a computer is to compute functions and to run programs. But on IoT devices that is not its main point, it has some other function besides that. As an example like in cars, the function of IoT devices are not to compute anti-lock breaking or to do fuel injection, their main function from the point of view of a user is to be driven and to move you from place to place and the computer is just to help that function. For example, The main function of the car is not to compute like anti-lock breaking or to do fuel injection their main function from the point of view of a user is to drive, to move you from place to place. But when we embed software in it then the software can be able for fuel limit detection.

3.2.1. Difference between IoT devices and Computers:

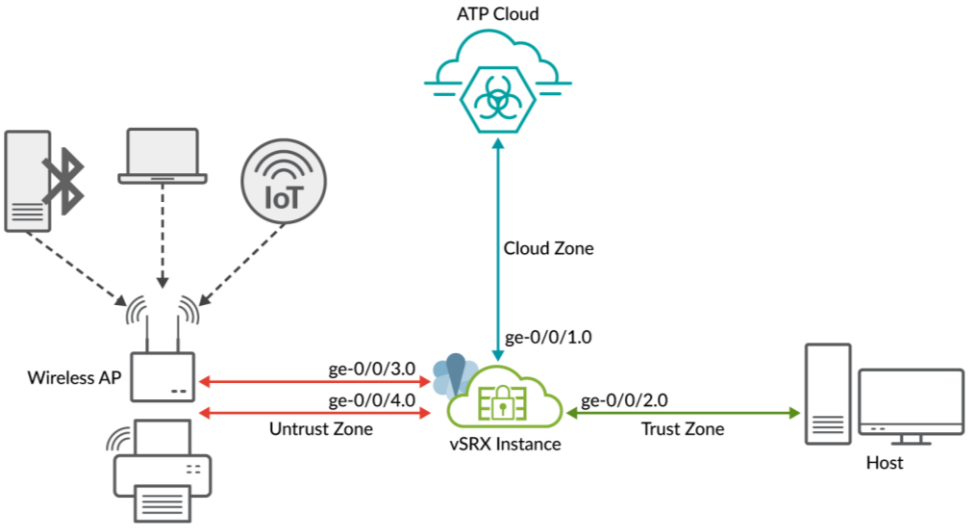
IOT Devices	Computers
IoT devices are special-purpose devices.	Computers are general-purpose devices.
IoT devices can do only a particular task for which it is designed.	Computers can do so many tasks.
The hardware and software built-in in the IoT devices are streamlined for that particular task.	The hardware and software built-in in the computers are streamlined to do many tasks(such as calculation, gaming, music player, etc.)
IoT devices can be cheaper and faster at a particular task than computers, as IoT devices are made to do that particular task.	A computer can be expensive and slower at a particular task than an IoT device.

IOT Devices	Computers
Examples: Music Player- iPod, Alexa, smart cars, etc.	Examples: Desktop computers, Laptops, etc.

Configure IoT Device Discovery and Policy Enforcement
date_range16-Dec-22
arrow_backward
arrow_forward

To get started with IoT device discovery in your network, all you need is a security device connected to Juniper ATP Cloud.

IoT Device Discovery and Policy Enforcement Topology



As shown in the topology, the network includes some IoT devices connected to an SRX Series device through wireless access point (AP). The security device is connected to the Juniper Cloud ATP server, and to a host device.

The security device collects IoT device metadata and streams the relevant information to the Juniper ATP Cloud. To enable streaming of IoT metadata, you'll need to create security metadata streaming policies and attach these policies to security policies. Streaming of the IoT device traffic pauses automatically when Juniper Cloud server has sufficient details to classify the IoT device.

Juniper ATP cloud discovers and classifies IoT devices. Using the inventory of discovered IoT devices, you'll create threat feeds in the form of dynamic address groups. Once the security device downloads dynamic address groups, you can use the dynamic address groups to create and enforce security policies for the IoT traffic.

Requirements

- SRX Series device or NFX Series device

- Junos OS Release 22.1R1 or later
- Juniper Advanced Threat Prevention Cloud Account. We've verified and tested the configuration using a vSRX instance with Junos OS Release 22.1R1.

3.3. IoT Configuration

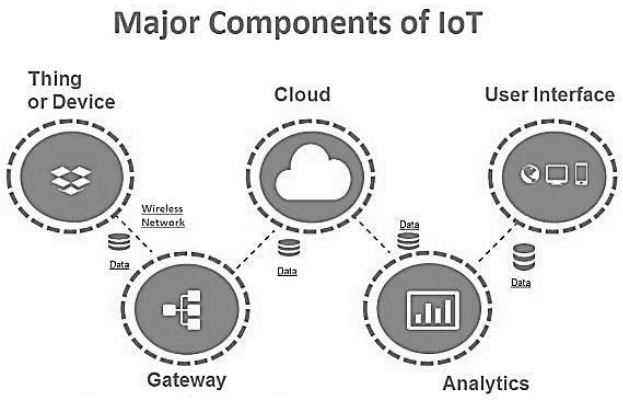
1. **Access**
    - Be sure the device's access control does not allow unauthorized or unauthenticated access and that appropriate restrictions are in place.
    - Change the default password to something complex with 16 characters or more if feasible.
    - Make sure no management interface is accessed over an unencrypted channel (e.g. use HTTPS rather than HTTP and disable the latter).
    - For Raspberry Pi Devices:
        - Make sudo require a password.
        - Modify the sshd configuration to allow only specific users for SSH access.
        - Use key-based authentication as an alternative to a password login.
    - For printers, disable any unnecessary services. Disable the following where possible, as they are commonly visible without a username or password:
        - Internal storage or job forwarding
        - Job logs
        - Print queue
        - Device management logs
2. **Software updates**
    1. Keep the device software updated with the latest security patches and firmware.
        - Some patches can revert or change passwords.
3. **Services**
    1. Disable all unused services, such as telnet, ssh, FTP/TFTP, MySQL, email, web proxy, SLP/Device Announcement Agent, WS-Discovery/XML Services, Web Services Print, WINS, LLMNR, IPX/SPX, DLC/LLC, Multicast IPv4, File Access Settings (JPL Disk Access, SNMP Disk Access, etc.).
    2. For printers:
        - Automatically delete data after the printing process is complete. Options can be set for how long data can be resident on the printer – set to lowest time available.

- Confirm deletion of data upon job completion.
- Disable local use of printer memory if possible.

4. **Network**
    1. IoT devices must use a non-publicly-routable IP address wherever possible. Exceptions must be approved by the Office of the Chief Information Security Officer (CISO).
    2. If possible, use a firewall or similar means of restricting access to the IoT device management services to only the IP address ranges needed. For off-campus access, the campus VPN range should be included rather than any external IP addresses.
    3. For Raspberry Pi Devices:
        - Install and configure fail2ban.
        - Install virus protection.
5. **Logging**
    1. Enable logging of IoT device access and configuration changes. Have the logs sent to an external device.
    2. Monitor logs for unusual behavior.
6. **Redundancy and backups**
    1. If the device has a critical function, consider having a backup configured or purchasing a second device.
    2. Regularly back up the device's configuration, or keep track of its configuration on an external system.
    3. **Check with the device's vendor for security bulletins or configuration.**
    4. **Consider the types of data your device stores or processes and ensure unauthorized individuals can't access that data.**

3.4. Components of IoT System

What are the major components of Internet of Things?



3.4.1. IoT Components

1. Smart devices and sensors – Device connectivity

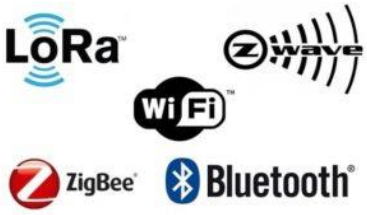
Devices and sensors are the components of the device connectivity layer. These smart sensors are continuously collecting data from the environment and transmit the information to the next layer.The latest techniques in semiconductor technology are capable of producing micro smart sensors for various applications.

Common sensors are:

- Temperature sensors and thermostats
- Pressure sensors
- Humidity / Moisture level
- Light intensity detectors
- Moisture sensors
- Proximity detection
- RFID tags

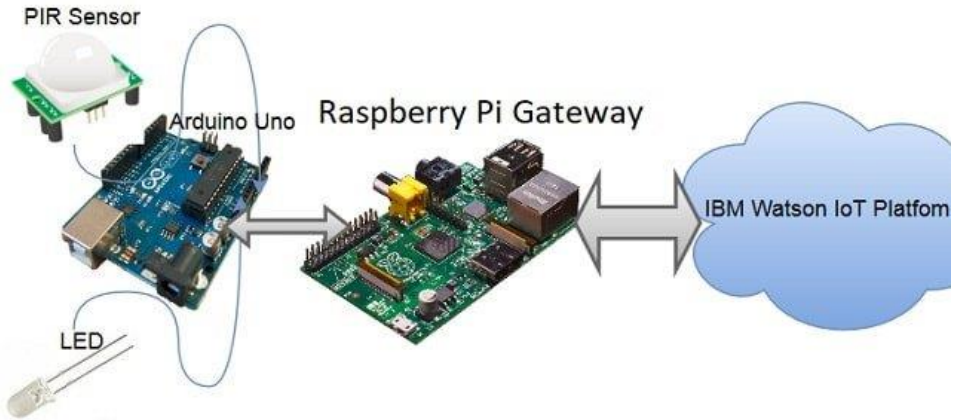
How are the devices connected?

Most modern smart devices and sensors can be connected to low-power wireless networks like Wi-Fi, ZigBee, Bluetooth, Z-wave, LoRAWAN, etc. Each of these wireless technologies has its pros and cons in terms of power, data transfer rate, and overall efficiency.



Developments in low-power, low-cost wireless transmitting devices are promising in the area of IoT due to their long battery life and efficiency. Many companies have adopted the latest protocols like 6LoWPAN- IPv6 over Low Power Wireless Personal Area Networks to implement energy-efficient data transmission for IoT networks.LoWPAN uses reduced transmission time (typically short time pulses) and thus saves energy.

2. Gateway



IoT Gateway manages the bidirectional data traffic between different networks and protocols. Another function of the gateway is to translate different network protocols and make sure interoperability of the connected devices and sensors.

Gateways can be configured to perform pre-processing of the collected data from thousands of sensors locally before transmitting it to the next stage. In some scenarios, it would be necessary due to the compatibility of the TCP/IP protocol.

IoT gateway offers a certain level of security for the network and transmitted data with higher-order encryption techniques. It acts as a middle layer between devices and the cloud to protect the system from malicious attacks and unauthorized access.

3. Cloud

The Internet of Things creates massive data from devices, applications, and users, which has to be managed in an efficient way. IoT cloud offers tools to collect, process, manage and store huge amounts of data in real time. Industries and services can easily access these data remotely and make critical decisions when necessary.

Basically, the IoT cloud is a sophisticated, high-performance network of servers optimized to perform high-speed data processing of billions of devices, traffic management, and deliver accurate analytics. Distributed database management systems are one of the most important components of the IoT cloud.

Cloud system integrates billions of devices, sensors, gateways, protocols, and data storage and provides predictive analytics. Companies use these analytics data to improve products and services, preventive measures for certain steps, and build their new business model accurately.

4. Analytics

Analytics is the process of converting analog data from billions of smart devices and sensors into useful insights which can be interpreted and used for detailed analysis. Smart analytics solutions are inevitable for IoT systems for the management and improvement of the entire system.

One of the major advantages of an efficient IoT system is real-time smart analytics which helps engineers to find out irregularities in the collected data and act fast to prevent an

undesired scenario. Service providers can prepare for further steps if the information is collected accurately at the right time.

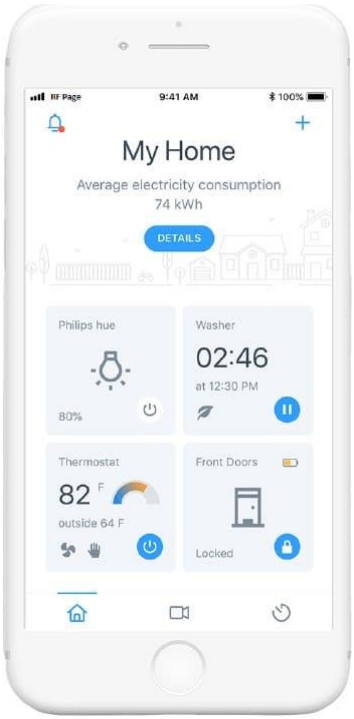
Big enterprises use the massive data collected from IoT devices and utilize the insights for their future business opportunities. Careful analysis will help organizations to predict trends in the market and plan ahead for a successful implementation.

Information is very significant in any business model, and predictive analysis ensures success in the concerned area of the business line.

5. User interface

User interfaces are the visible, tangible part of the IoT system which users can access. Designers will have to make sure of a well-designed user interface for minimum effort for users and encourage more interactions.

Modern technology offers much interactive design to ease complex tasks into simple touch panel controls. Multicolor touch panels have replaced hard switches in our household appliances, and the trend is increasing for almost every smart home device.



The user interface design has higher significance in today's competitive market; it often determines the user whether to choose a particular device or appliance. Users will be interested in buying new devices or smart gadgets if it is very user-friendly and compatible with common wireless standards.

3.4.2. Real Life Example Depicting Working of IoT

Here, we will discuss some examples, which states how IoT works in real-life:

i. Say, we have an AC in our room, now the temperature sensor installed in it in the room will be integrated with a gateway. A gateway's purpose is to help connect the temperature sensor (inside the AC) to the Internet by making use of a cloud infrastructure.
ii. A Cloud/server infrastructure has detailed records about each and every device connected to it such as device id, a status of the device, what time was the device last accessed, number of times the device has been accessed and much more.
iii. A connection with the cloud then implement by making use of web services such as RESTful.
iv. We in this system, act as end-users and through the mobile app interact with Cloud (and in turn devices installed in our homes). A request will send to the cloud infrastructure with the authentication of the device and device information. It requires authentication of the device to ensure cybersecurity.
v. Once the cloud has authenticated the device, it sends a request to the appropriate sensor network using gateways.
vi. After receiving the request, the temperature sensor inside the AC will read the current temperature in the room and will send the response back to the cloud.
vii. Cloud infrastructure will then identify the particular user who has requested the data and will then push the requested data to the app. So, a user will get the current information about the temperature in the room, pick up through AC's temperature sensors directly on his screen.

Real Life Example – Internet of Things works

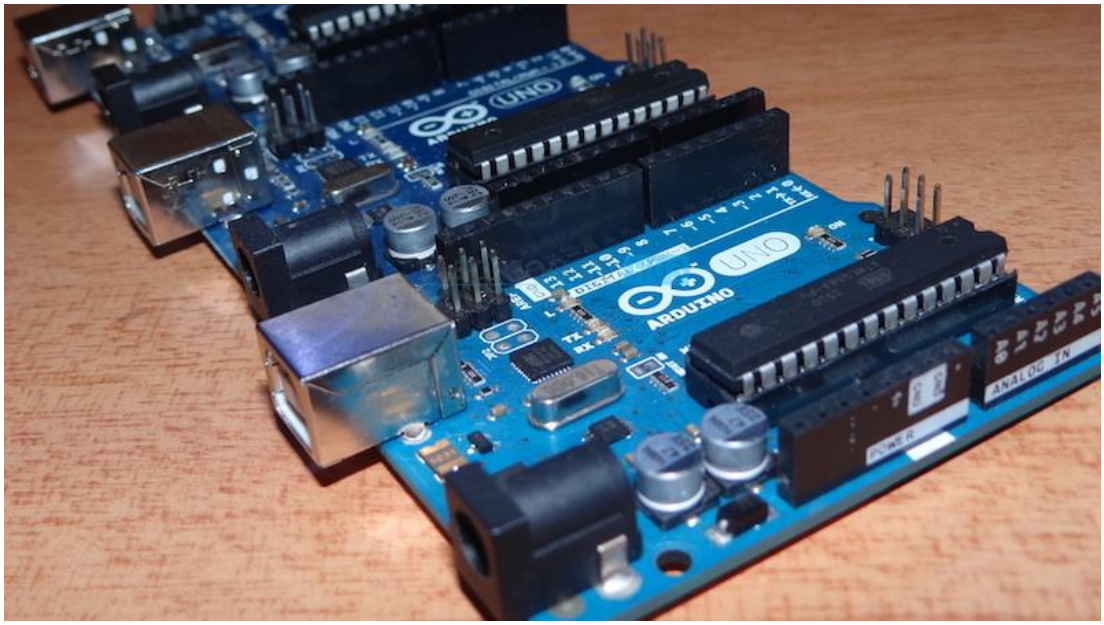
3.5. Arduino

A decade ago, working around electronics involved knowledge in physics and math, expensive lab equipment, a laboratory type setup and important of all, love for electronics. But the picture has changed over the decade or so where the above-mentioned factors became irrelevant to work around electronics except for the last part: love for electronics.One such product which made use of the above specified and many other reasons and made electronics be able reach anyone regardless of their background is "Arduino".

3.5.1. Introduction

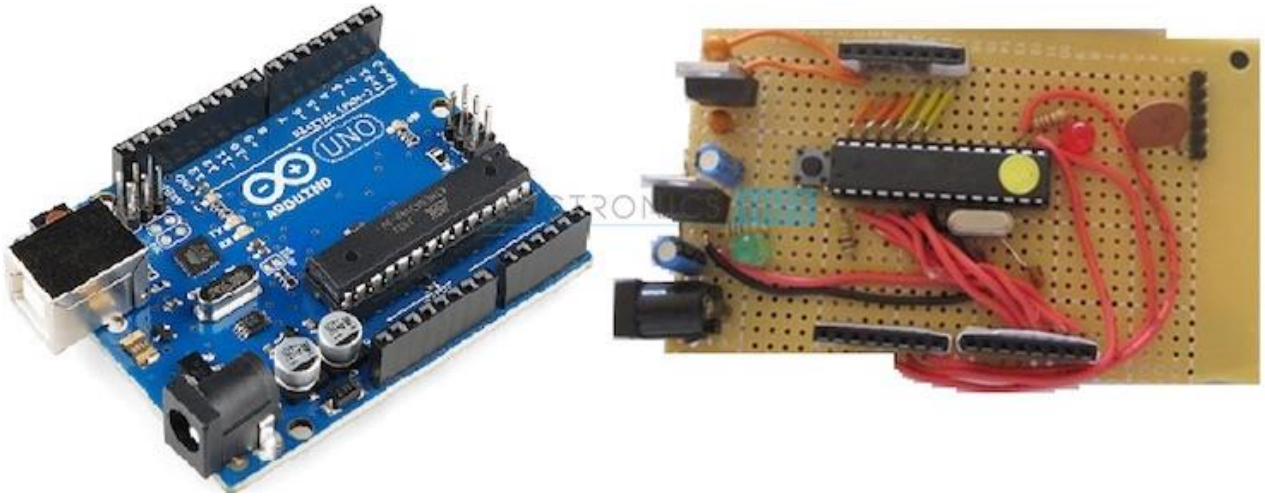
Arduino is an open-source prototyping platform in electronics based on easy-to-use hardware and software. Subtly speaking, Arduino is a microcontroller based prototyping board which

can be used in developing digital devices that can read inputs like finger on a button, touch on a screen, light on a sensor etc. and turning it in to output like switching on an LED, rotating a motor, playing songs through a speaker etc.



The Arduino board can be programmed to do anything by simply programming the microcontroller on board using a set of instructions for which, the Arduino board consists of a USB plug to communicate with your computer and a bunch of connection sockets that can be wired to external devices like motors, LEDs etc.

The aim of Arduino is to introduce the world of electronics to people who have small to no experience in electronics like hobbyists, designers, artists etc.



Arduino is based on open source electronics project i.e. all the design specifications, schematics, software are available openly to all the users. Hence, Arduino boards can bought from vendors as they are commercially available or else you can make your own board by if you wish i.e. you can download the schematic from Arduino's official website, buy all the

components as per the design specification, assemble all the components, and make your own board.

3.5.2. Hardware and Software

Arduino boards are generally based on microcontrollers from Atmel Corporation like 8, 16 or 32 bit AVR architecture based microcontrollers.

The important feature of the Arduino boards is the standard connectors. Using these connectors, we can connect the Arduino board to other devices like LEDs or add-on modules called Shields.

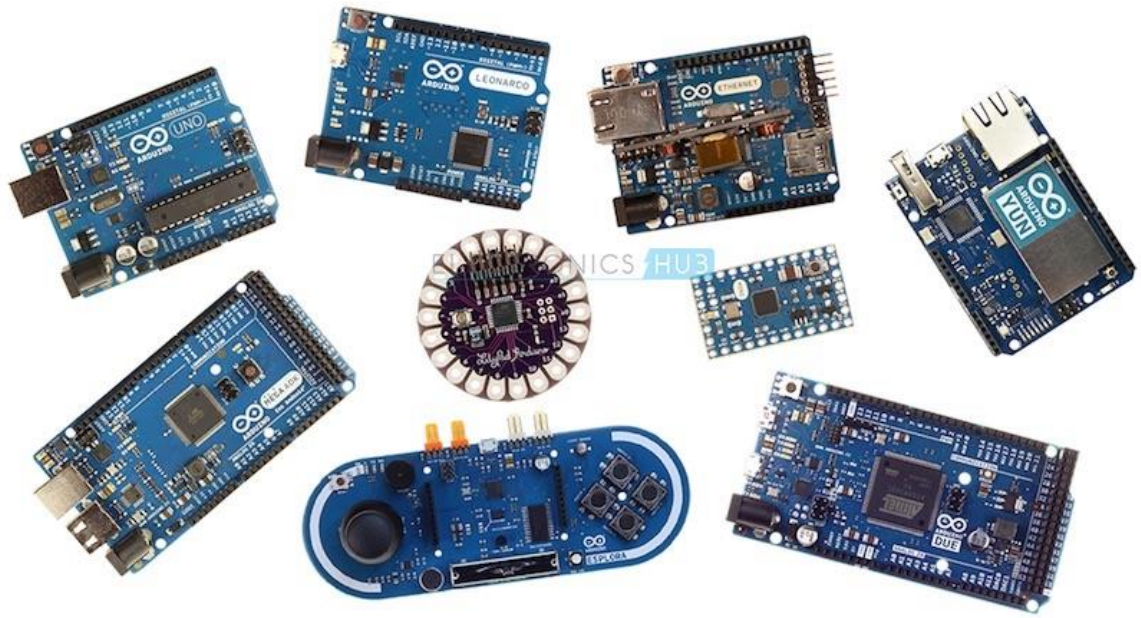
The Arduino boards also consists of on board voltage regulator and crystal oscillator. They also consist of USB to serial adapter using which the Arduino board can be programmed using USB connection.

In order to program the Arduino board, we need to use IDE provided by Arduino. The Arduino IDE is based on Processing programming language and supports C and C++.

3.5.3. Types of Arduino Boards

There are many types of Arduino boards available in the market but all the boards have one thing in common: they can be programmed using the Arduino IDE. The reasons for different types of boards are different power supply requirements, connectivity options, their applications etc.

Arduino boards are available in different sizes, form factors, different no. of I/O pins etc. Some of the commonly known and frequently used Arduino boards are Arduino UNO, Arduino Mega, Arduino Nano, Arduino Micro and Arduino Lilypad.

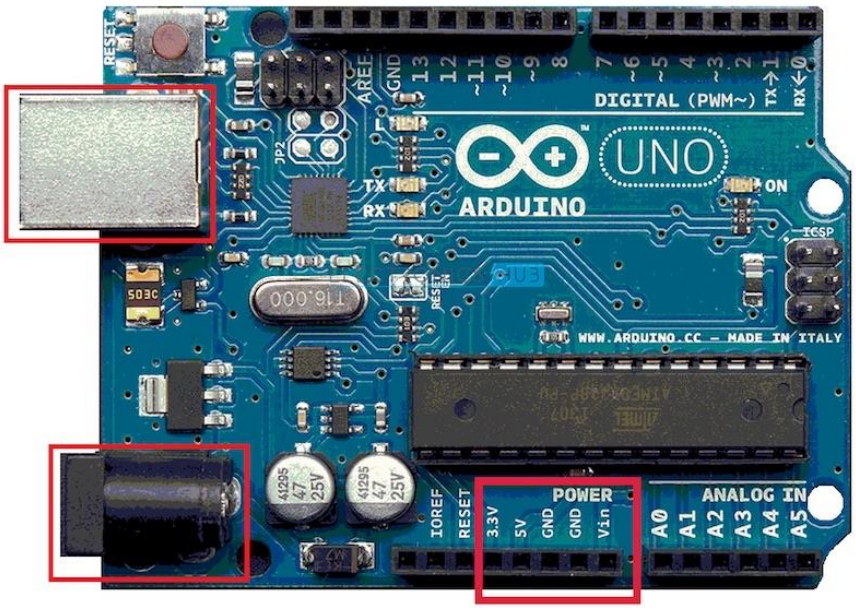


There are add-on modules called Arduino Shields which can be used to extend the functionalities of the Arduino boards. Some of the commonly used shields are Arduino Proto shield, Arduino WiFi Shield and Arduino Yun Shield.

Step 1: Types of Arduino

3.5.3.1. Arduino UNO

The most common version of Arduino is the Arduino Uno. This board is what most people are talking about when they refer to an Arduino. In the next step, there is a more complete rundown of its features.



3.5.3.2. Arduino NG, Diecimila, and the Duemilanove (Legacy Versions)

Legacy versions of the Arduino Uno product line consist of the NG, Diecimila, and the Duemilanove. The important thing to note about legacy boards is that they lack particular feature of the Arduino Uno. Some key differences:

- The Diecimila and NG use an ATMEGA168 chips (as opposed to the more powerful ATMEGA328),
- Both the Diecimila and NG have a jumper next to the USB port and require manual selection of either USB or battery power.
- The Arduino NG requires that you hold the rest button on the board for a few seconds prior to uploading a program.

3.5.3.3. Arduino Mega 2560

The Arduino Mega 2560 is the second most commonly encountered version of the Arduino family. The Arduino Mega is like the Arduino Uno's beefier older brother. It boasts 256 KB of memory (8 times more than the Uno). It also had 54 input and output pins, 16 of which are analog pins, and 14 of which can do PWM. However, all of the added functionality comes at the cost of a slightly larger circuit board. It may make your project more powerful, but it will also make your project larger. Check out the official Arduino Mega 2560 page for more details.

3.5.3.4. Arduino Mega ADK

This specialized version of the Arduino is basically an Arduino Mega that has been specifically designed for interfacing with Android smartphones. This too is now a legacy version.

3.5.3.5. Arduino Yun

The Arduino Yun uses a ATMega32U4 chip instead of the ATmega328. However, what really sets it apart is the addition of the Atheros AR9331 microprocessor. This extra chip allows this board to run Linux in addition to the normal Arduino operating system. If all of that were not enough, it also has onboard wifi capability. In other words, you can program the board to do stuff like you would with any other Arduino, but you can also access the Linux side of the board to connect to the internet via wifi. The Arduino-side and Linux-side can then easily communicate back and forth with each other. This makes this board extremely powerful and versatile. I'm barely scratching the surface of what you can do with this, but to learn more, check out the official Arduino Yun page.

3.5.3.6. Arduino Nano

If you want to go smaller than the standard Arduino board, the Arduino Nano is for you! Based on a surface mount ATmega328 chip, this version of the Arduino has been shrunk down to a small footprint capable of fitting into tight spaces. It can also be inserted directly into a breadboard, making it easy to prototype with.

3.5.3.7. Arduino LilyPad

The LilyPad was designed for wearable and e-textile applications. It is intended to be sewn to fabric and connected to other sewable components using conductive thread. This board requires the use of a special FTDI-USB TTL serial programming cable. For more information, the Arduino LilyPad page is a decent starting point.

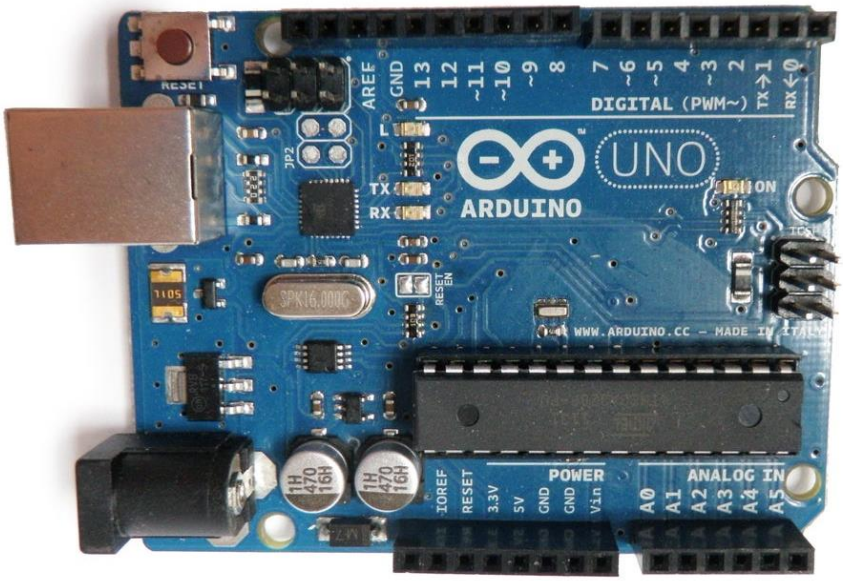
(Note that some of the links on this page are affiliate links. This does not change the cost of the item for you. I reinvest whatever proceeds I receive into making new projects. If you would like any suggestions for alternative suppliers, please let me know.)

Step 2: Arduino Uno Features

Some people think of the entire Arduino board as a microcontroller, but this is inaccurate. The Arduino board actually is a specially designed circuit board for programming and prototyping with Atmel microcontrollers.

The nice thing about the Arduino board is that it is relatively cheap, plugs straight into a computer's USB port, and it is dead-simple to setup and use (compared to other development boards).
Some of the key features of the Arduino Uno include:

An open source design. The advantage of it being open source is that it has a large community of people using and troubleshooting it. This makes it easy to find someone to help you debug your projects.



- An easy USB interface . The chip on the board plugs straight into your USB port and registers on your computer as a virtual serial port. This allows you to interface with it as through it were a serial device. The benefit of this setup is that serial communication is an extremely easy (and time-tested) protocol, and USB makes connecting it to modern computers really convenient.
- Very convenient power management and built-in voltage regulation. You can connect an external power source of up to 12v and it will regulate it to both 5v and 3.3v. It also can be powered directly off of a USB port without any external power.
- An easy-to-find, and dirt cheap, microcontroller "brain." The ATmega328 chip retails for about $2.88 on Digikey. It has countless number of nice hardware features like timers, PWM pins, external and internal interrupts, and multiple sleep modes. Check out the official datasheet for more details.
- A 16mhz clock. This makes it not the speediest microcontroller around, but fast enough for most applications.
- 32 KB of flash memory for storing your code.
- 13 digital pins and 6 analog pins. These pins allow you to connect external hardware to your Arduino. These pins are key for extending the computing capability of the Arduino into the real world. Simply plug your devices and sensors into the sockets that correspond to each of these pins and you are good to go.
- An ICSP connector for bypassing the USB port and interfacing the Arduino directly as a serial device. This port is necessary to re-bootload your chip if it corrupts and can no longer talk to your computer.
- An on-board LED attached to digital pin 13 for fast an easy debugging of code.
- And last, but not least, a button to reset the program on the chip.

Step 3: Arduino IDE

Before you can start doing anything with the Arduino, you need to download and install the Arduino IDE (integrated development environment). From this point on we will be referring to the Arduino IDE as the Arduino Programmer.

The Arduino Programmer is based on the Processing IDE and uses a variation of the C and C++ programming languages.
You can find the most recent version of the Arduino Programmer on this page.

Step 4: Plug It In

Connect the Arduino to your computer's USB port.
Please note that although the Arduino plugs into your computer, it is not a true USB device. The board has a special chip that allows it to show up on your computer as a virtual serial port when it is plugged into a USB port. This is why it is important to plug the board in. When the board is not plugged in, the virtual serial port that the Arduino operates upon will not be present (since all of the information about it lives on the Arduino board).

It is also good to know that every single Arduino has a unique virtual serial port address. This means that every time you plug in a different Arduino board into your computer, you will need to reconfigure the serial port that is in use.

The Arduino Uno requires a male USB A to male USB B cable.

Step 5: Settings

Before you can start doing anything in the Arduino programmer, you must set the board-type and serial port.
To set the board, go to the following:

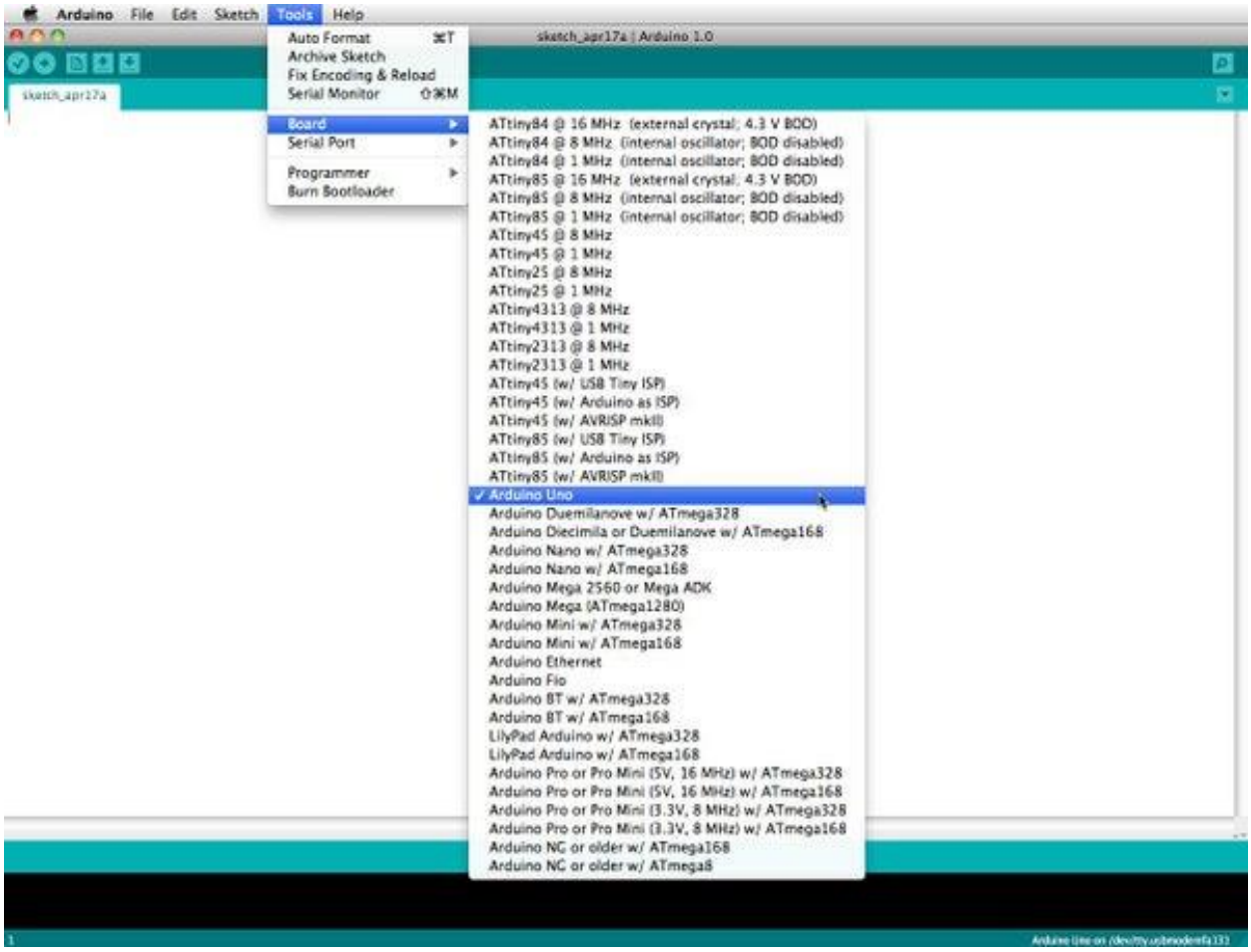
Tools --> Boards

Select the version of board that you are using. Since I have an Arduino Uno plugged in, I obviously selected "Arduino Uno."
To set the serial port, go to the following:

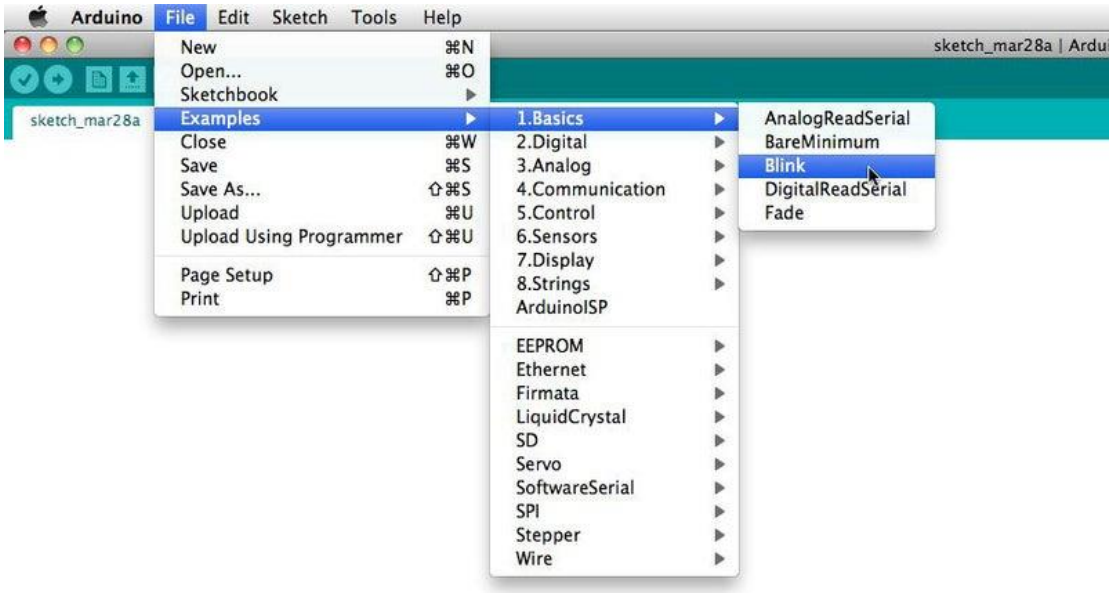
Tools --> Serial Port

Select the serial port that looks like:

/dev/tty.usbmodem [random numbers]



Step 6: Run a Sketch



Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.

To get the LED tied to digital pin 13 to blink on and off, let's load the blink example.
The blink example can be found here:

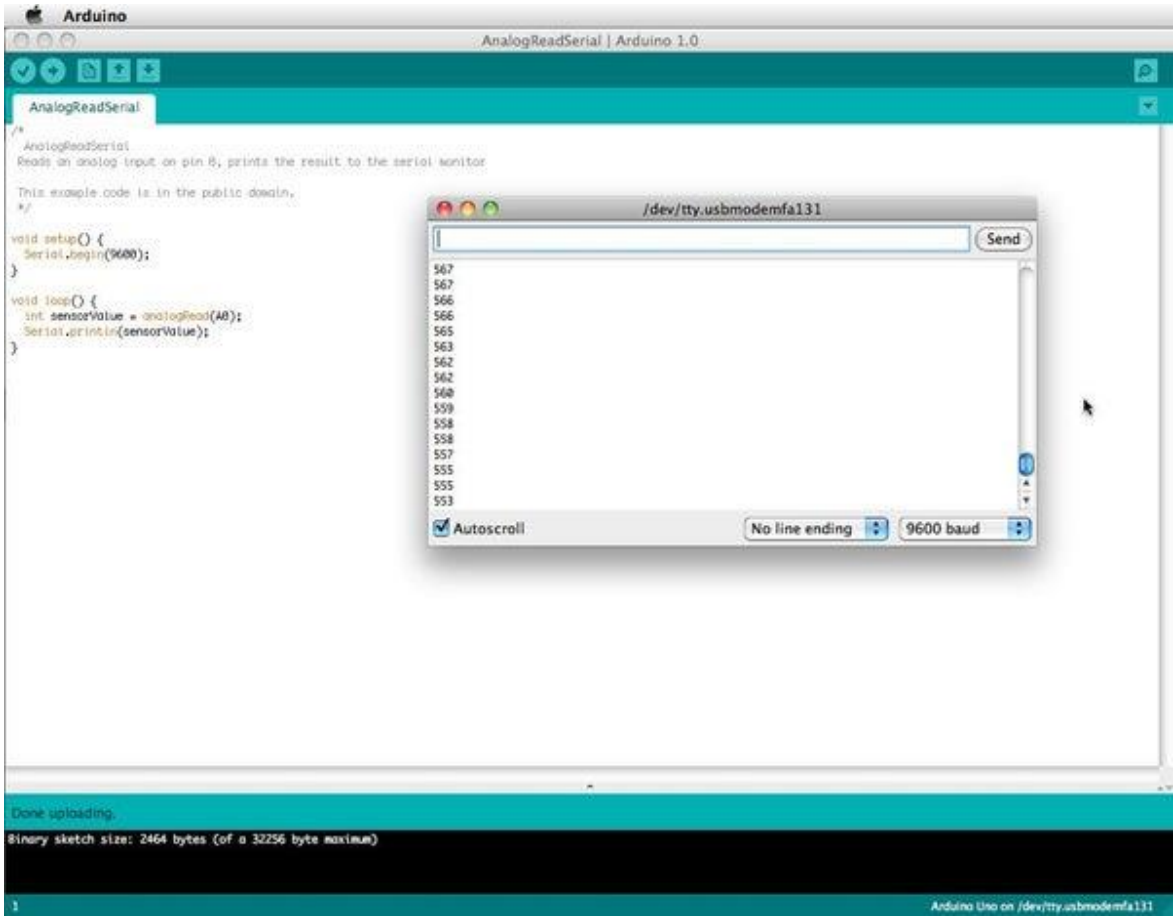
Files --> Examples --> Basics --> Blink

The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Notice that the surface mount status LED connected to pin 13 on the Arduino will start to blink. You can change the rate of the blinking by changing the length of the delay and pressing the upload button again.

Step 7: Serial Monitor



The serial monitor allows your computer to connect serially with the Arduino. This is important because it takes data that your Arduino is receiving from sensors and other devices and displays it in real-time on your computer. Having this ability is invaluable to debug your code and understand what number values the chip is actually receiving.

For instance, connect center sweep (middle pin) of a potentiometer to A0, and the outer pins, respectively, to 5v and ground. Next upload the sketch shown below:

File --> Examples --> 1.Basics --> AnalogReadSerial

Click the button to engage the serial monitor which looks like a magnifying glass. You can now see the numbers being read by the analog pin in the serial monitor. When you turn the knob the numbers will increase and decrease.

The numbers will be between the range of 0 and 1023. The reason for this is that the analog pin is converting a voltage between 0 and 5V to a discreet number.

Step 8: Digital In

The Arduino has two different types of input pins, those being analog and digital.
To begin with, lets look at the digital input pins.

Digital input pins only have two possible states, which are on or off. These two on and off states are also referred to as:

- HIGH or LOW
- 1 or 0
- 5V or 0V.

This input is commonly used to sense the presence of voltage when a switch is opened or closed.
Digital inputs can also be used as the basis for countless digital communication protocols. By creating a 5V (HIGH) pulse or 0V (LOW) pulse, you can create a binary signal, the basis of all computing. This is useful for talking to digital sensors like a PING ultrasonic sensor, or communicating with other devices.

For a simple example of a digital input in use, connect a switch from digital pin 2 to 5V, a 10K resistor** from digital pin 2 to ground, and run the following code:

File --> Examples --> 2.Digital --> Button

**The 10K resistor is called a pull-down resistor because it connects the digital pin to ground when the switch is not pressed. When the switch is pressed, the electrical connections in the switch has less resistance than the resistor, and the electricity no longer connects to ground. Instead, electricity flows between 5V and the digital pin. This is because electricity always chooses the path of least resistance. To learn more about this, visit the Digital Pins page.

Step 9: Analog In

Aside from the digital input pins, the Arduino also boasts a number of analog input pins. Analog input pins take an analog signal and perform a 10-bit analog-to-digital (ADC) conversion to turn it into a number between 0 and 1023 (4.9mV steps).

This type of input is good for reading resistive sensors. These are basically sensors which provide resistance to the circuit. They are also good for reading a varying voltage signal between 0 and 5V. This is useful when interfacing with various types of analog circuitry.

If you followed the example in Step 7 for engaging the serial monitor, you have already tried using an analog input pin.

Step 10: Digital Out

A digital out pin can be set to be HIGH (5v) or LOW (0v). This allows you to turn things on and off.

Aside from turning things on and off (and making LEDs blink), this form of output is convenient for a number of applications.

Most notably, it allows you to communicate digitally. By turning the pin on and off rapidly, you are creating binary states (0 and 1), which is recognized by countless other electronic devices as a binary signal. By using this method, you can communicate using a number of different protocols.

Digital communication is an advanced topic, but to get a general idea of what can be done, check out the Interfacing With Hardware page.
If you followed the example in Step 6 for getting an LED to blink, you have already tried using a digital output pin.

Step 11: Analog Out

As mentioned earlier, the Arduino has a number of built in special functions. One of these special functions is pulse width modulation, which is the way an Arduino is able to create an analog-like output.

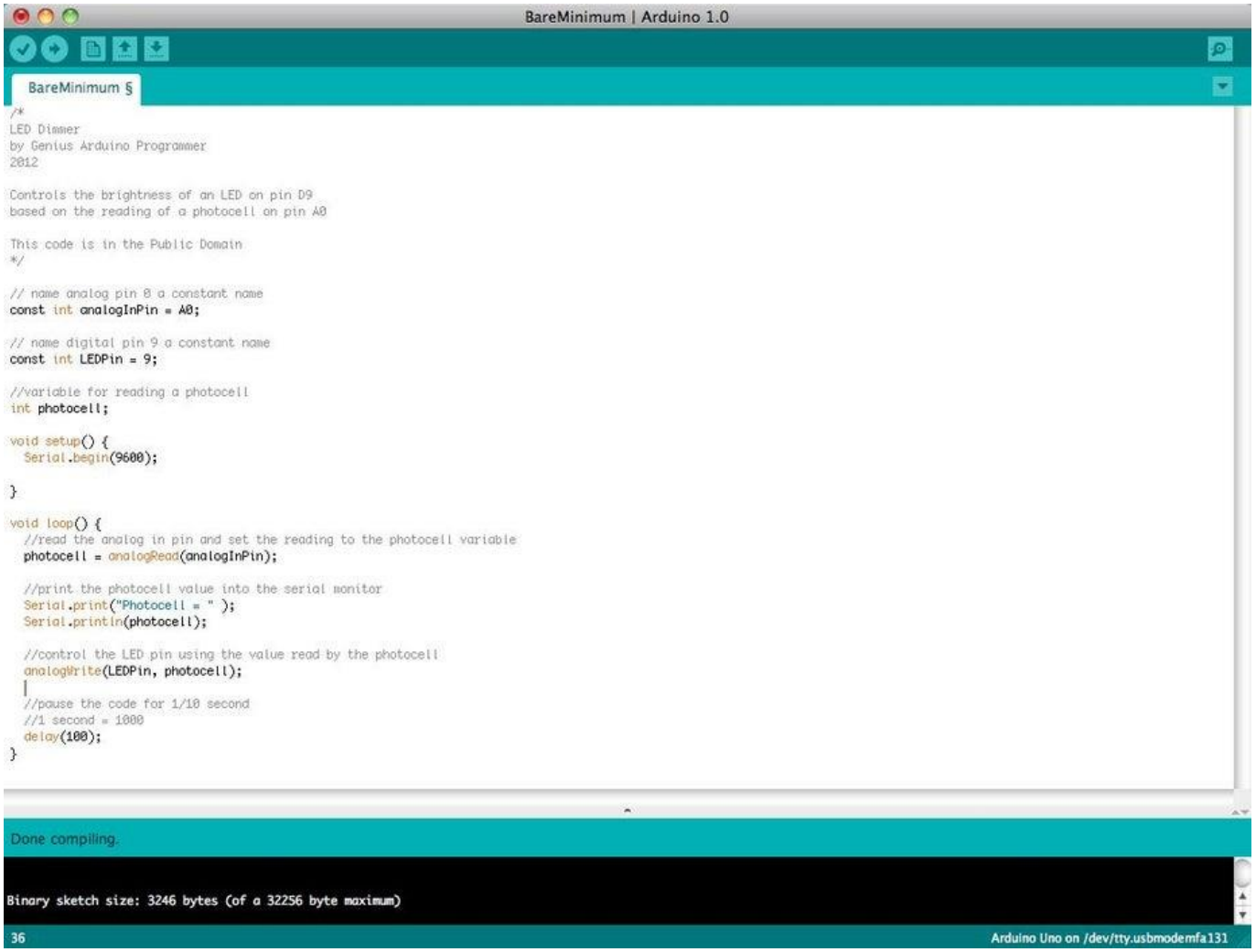
Pulse width modulation - or PWM for short - works by rapidly turning the PWM pin high (5V) and low (0V) to simulate an analog signal. For instance, if you were to blink an LED on and off rapidly enough (about five milliseconds each), it would seem to average the brightness and only appear to be receiving half the power. Alternately, if it were to blink on for 1 millisecond and then blink off for 9 millisecond, the LED would appear to be 1/10 as bright and only be receiving 1/10 the voltage.

PWM is key for a number of applications including making sound, controlling the brightness of lights, and controlling the speed of motors.

To try out PWM yourself, connect an LED and 220 ohm resistor to digital pin 9, in series to ground. Run the following example code:

File --> Examples --> 3.Analog --> Fading

Step 12: Write Your Own Code



To write your own code, you will need to learn some basic programming language syntax. In other words, you have to learn how to properly form the code for the programmer to understand it. You can think of this kind of like understanding grammar and punctuation. You can write an entire book without proper grammar and punctuation, but no one will be abler to understand it, even if it is in English.

Some important things to keep in mind when writing your own code:

- An Arduino program is called a sketch.
- All code in an Arduino sketch is processed from top to bottom.
- Arduino sketches are typically broken into five parts.

1. The sketch usually starts with a header that explains what the sketch is doing, and who wrote it.
2. Next, it usually defines global variables. Often, this is where constant names are given to the different Arduino pins.
3. After the initial variables are set, the Arduino begins the setup routine. In the setup function, we set initial conditions of variables when necessary, and run any preliminary code that we only want to run once. This is where serial communication is initiated, which is required for running the serial monitor.

4. From the setup function, we go to the loop routine. This is the main routine of the sketch. This is not only where your main code goes, but it will be executed over and over, so long as the sketch continues to run.
5. Below the loop routine, there is often other functions listed. These functions are user-defined and only activated when called in the setup and loop routine. When these functions are called, the Arduino processes all of the code in the function from top to bottom and then goes back to the next line in the sketch where it left off when the function was called. Functions are good because they allow you to run standard routines - over and over - without having to write the same lines of code over and over. You can simply call upon a function multiple times, and this will free up memory on the chip because the function routine is only written once. It also makes code easier to read.

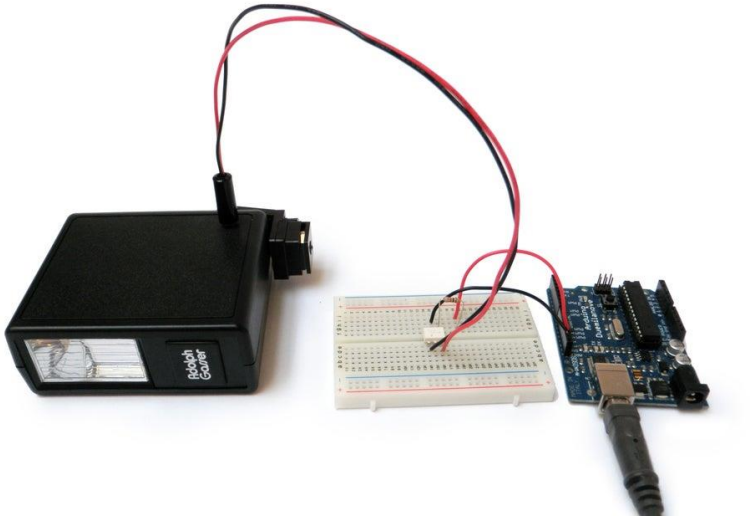
- All of that said, the only two parts of the sketch which are mandatory are the Setup and Loop routines.
- Code must be written in the Arduino Language, which is roughly based on C.
- Almost all statements written in the Arduino language must end with a ;
- Conditionals (such as if statements and for loops) do not need a ;
- Conditionals have their own rules and can be found under "Control Structures" on the Arduino Language
- Variables are storage compartments for numbers. You can pass values into and out of variables. Variables must be defined (stated in the code) before they can be used and need to have a data type associated with it. To learn some of the basic data types, review the Language Page.

Step 13: Shields

Shields are expansion apdapter boards that plug in over top of the Arduino Uno and gives it special functions.

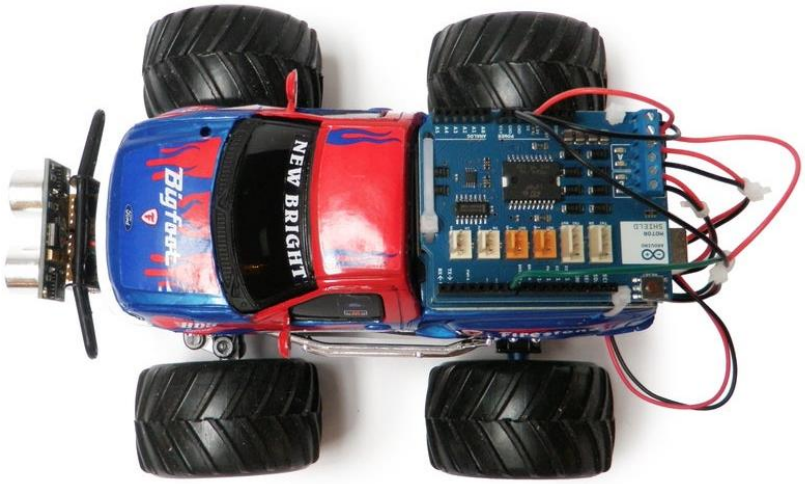
Since the Arduino is open hardware, anyone who has the inclination is free to make an Arduino shield for whatever task they wish to accomplish. On account of this, there are countless number of Arduino shields out in the wild. You can find an ever-growing list of Arduino shields in the Arduino playground. Keep in mind that there will be more shield in existence than you will find on listed on that page (as always, Google is your friend).

Step 14: Building an External Circuit



As your projects get more complex, you will want to build your own circuits to interfacewith the Arduino. While you won't learn electronics overnight, the internet is an unbelievable resource for electronic knowledge and circuit diagrams.
To get started with electronics, visit the Basic Electronics Instructable.

Step 15: Going Beyond



3.6. Arduino ToolChain

Today we're going to talk about how the Arduino tool chain converts instructions you type into the text editor into machine language the microcontroller can understand.

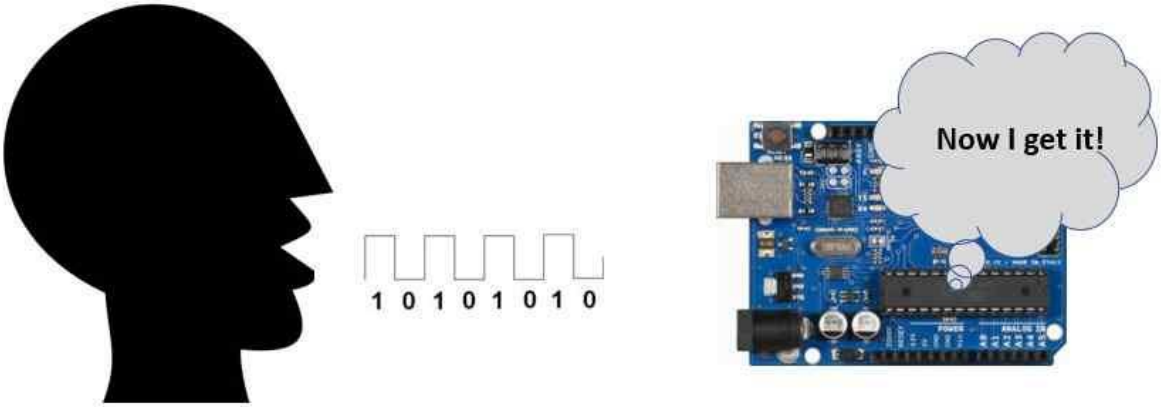
A tool chain is simply a set of software tools that work together to complete a task. for example, if we were going to hang a picture, the tool chain might include a hammer, maybe a tape measure and some nails.

Arduino Hardware/Software Interface

When programming the Arduino (or anything else), it is possible to write some pretty complex instructions and get the Arduino to do some really cool things.

The problem is that a microcontroller (like the ATmega328 on the Uno) can only execute simple, low-level instructions. These simple instructions include things like *add variable a to variable b* or *take variable b and put it in register x*.

And, to complicate matters, microcontrollers only speak in binary. For those who don't know, binary numbers are long strings of 1s and 0s. If you need to brush up on binary numbers, see A Bit of Fun with Binary Number Basics.

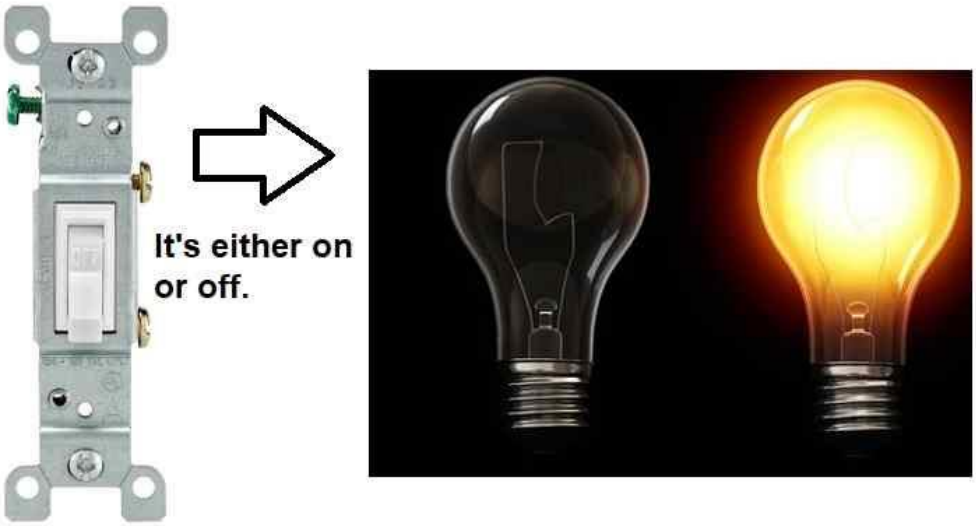


Arduino and other microcontrollers only understand binary.

The statement here may make total sense to you if you've been working with Arduino for a while. Or, if not, it may not make any sense at all. Either way, just go with me here for a minute. This is just an illustration which does not require complete understanding of the code.

```
Int Temp = analogRead(sensorPin);
        If (Temp > 500)
        {
        //more complex instructions go here
        }
```

A good way to think of binary numbers and digital signals is like a single pole light switch. The light is either on or its off, there is nothing in between. Zero is off and one is on.



a good way to think about digital signals.

Going from complex instructions to simple ones the microcontroller understands requires several layers of software that translate the high-level operations into simpler instructions.

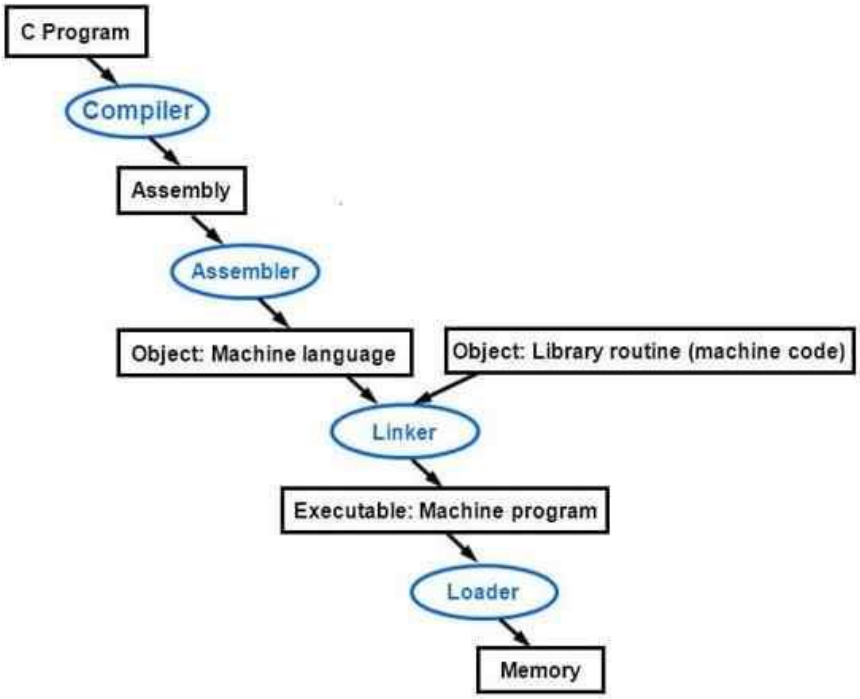
How the Arduino Compiler Works

The compiler first transforms the code you write into assembly language. The name of the compiler we'll be using on our Uno is AVR-GCC. If you're new to this, that may sound kind of weird but try not to obsess over it. It's just a name.

The assembler, which come with the IDE with the compiler, then translates the assembly language program into machine language. It then creates object files, which combine machine language, data, and information it needs to place instructions properly in memory. Often, the assembler creates multiple files which will eventually be put together.

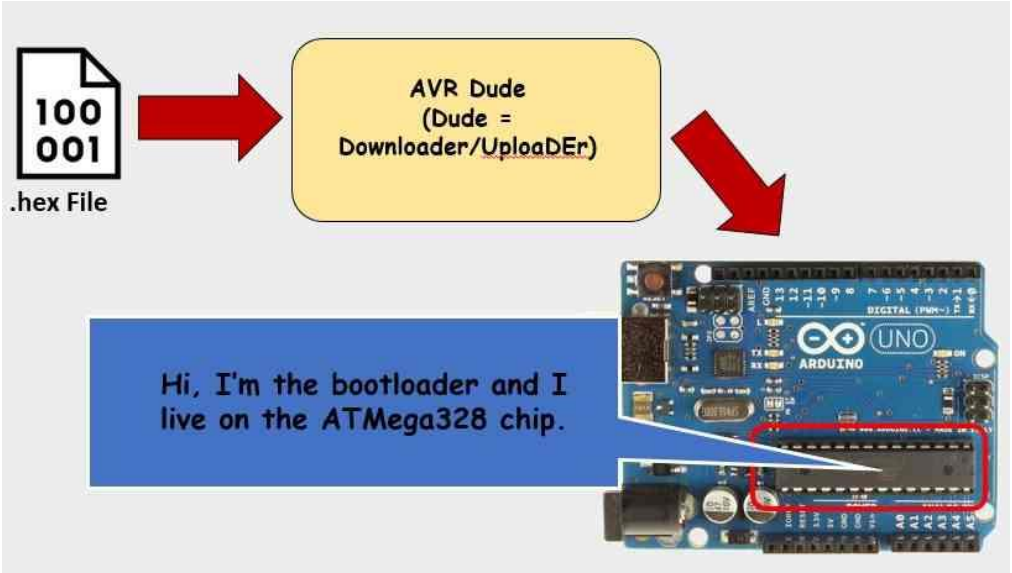
This is where the linker — another part of the compiler software package — shines. The linker will take all the independently assembled machine language programs and object files and put them together. This produces a .hex file that the microprocessor can understand and run.

The two figures below, though they apply to C/C++ programming in general, are a good illustration of this process.



another way to visualize the Arduino code compilation process.

Another piece of software, called AVR Dude (for Downloader UploaDEr) starts when we press the upload button. This software sends the .hex file to the ATMega328 on the Arduino board. On the chip resides the bootloader. This bootloader was put there on purpose by the folks at Arduino and works with AVR Dude to get the .hex into the flash memory on the chip.
All of this happens very quickly and seamlessly behind the scenes of the Arduino IDE.



how the compiled code transfers to the Arduino board.

Programming in C vs Arduino

A few words are in order on this subject due to the enormous popularity of Arduino boards and the C/C++ language in general.

Others use platforms or ecosystems such as the Arduino almost (or entirely) exclusively. Finally, some of you may use both depending on your goals and background.

For hobbyists, the number of people who use platforms like Arduino has exceeded those who only use naked microcontrollers.

When we talk about programming the Arduino, we'll talk about the C/C++ languages. The truth is, sketches are written in a language similar to C, though a sketch itself is not completely compatible with C.

In Arduino, the *main()* function is hidden from view and added for you when you compile or "verify" your sketch. Also, there are two functions which the Arduino ecosystem absolutely requires: *setup()* and *loop()*. The only function C requires is *main()*.

C also lacks built-in functions for using microcontroller I/O such as *digitalWrite()*.
To make learning simple, the Arduino IDE designers hide a lot of detail and functionality behind layers of abstraction, many of which come in the form of libraries. Note that the C programming language also uses libraries. The linker adds them during the linking process.

3.7. Arduinoprogramstructure

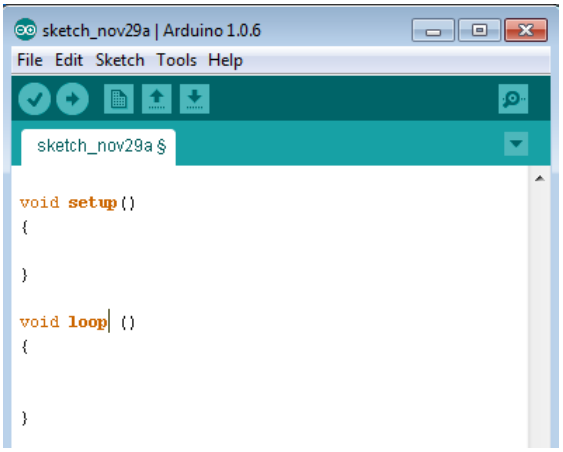
Arduinoprogramstructureandwewilllearn morenewterminologiesusedintheArduinoworld.TheArduinosoftwareisopen-source. ThesourcecodefortheJavaenvironmentisreleasedundertheGPLandtheC/C++ microcontrollerlibrariesareundertheLGPL.

Sketch: ThefirstnewterminologyistheArduinoprogramcalled"**sketch**".

3.7.1. Structure

Arduinoprogramscanbedividedinthreemainparts:**Structure**,**Values**(variablesand constants),and**Functions**.wewilllearnabouttheArduinosoftware program,stepbystep,andhowwecanwritetheprogramwithoutanysyntaxor compilationerror.
Letusstartwiththe**Structure**.Softwarestructureconsistoftwomainfunctions:

- Setup()function
- Loop()function




```
Void setup ( )
{

}
```

PURPOSE: The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

```
Void Loop ( )
{

}
```

PURPOSE:Aftercreatinga**setup()**function,whichinitializesandsetsthe initialvalues,the**loop()**functiondoespreciselywhatitsnamesuggests,and loopsconsecutively,allowingyourprogramtochangeandrespond.Useitto activelycontroltheArduinoboard.

2. Arduino – DataTypes

DatatypesinCreferstoanextensivesystemusedfordeclaringvariablesorfunctionsof differenttypes.Thetypeofavariabledetermineshowmuchspaceitoccupiesinthe storageandhowthebitpatternstoredisinterpreted.
ThefollowingtableprovidesallthedatatypesthatyouwillusduringArduino programming.

Void, Boolean , char, Unsigned, Unsigned, char , byte, int, Unsigned, int, word String.Array, long, long, short , float , double , array, char, String, object

void
Thevoidkeywordisusedonlyinfunctiondeclarations.Itindicatesthatthe function is expectedtoreturnnoinformationtothefunctionfromwhichitwascalled.

Example

```
Void Loop ( )
{
   // rest of the code
}
```

Boolean

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte ofmemory.

Example

```
boolean  val = false ;  // declaration of variable with type boolean and initialize
it with false
boolean   state = true ;   //    declaration of variable with type boolean and
initialize it with false
```

Char

Adatatypethattakes uponebyteof memory thatstores a charactervalue.Character literalsarewritteninsinglequotes likethis:'A'andformultiplecharacters, stringsuse doublequotes:"ABC".

However,charactersarestoredasnumbers.YoucanseethespecificencodingintheASCII chart.Thismeansthatitispossibletodoarithmeticoperationsoncharacters,inwhich theASCIIvalueofthecharacterisused.Forexample,'A'+1hasthevalue66,sincethe ASCIIvalueofthecapitalletterAis65.

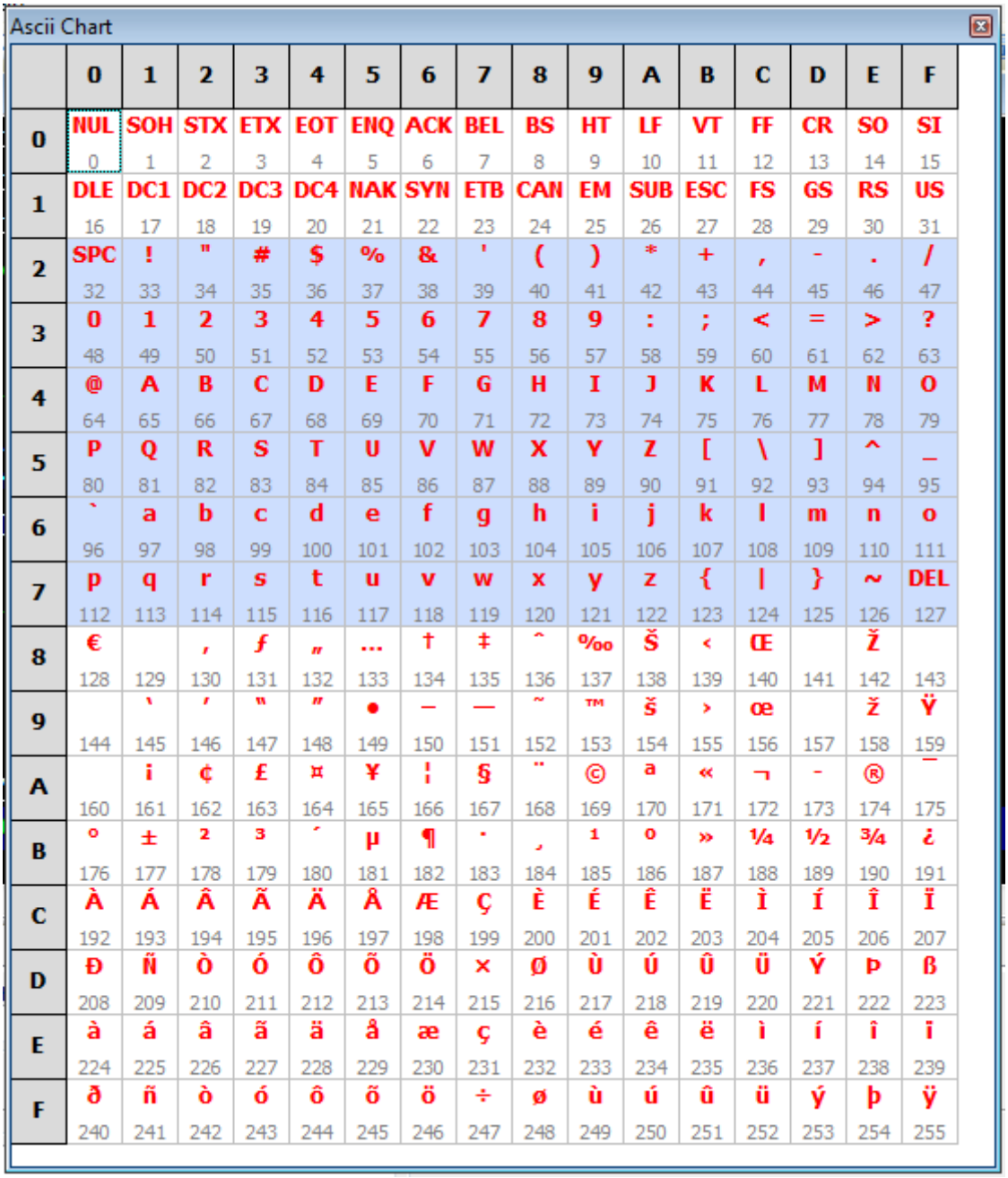
Example

```
Char   chr_a = ‘a’ ;//declaration of variable with type char and initialize it
with character a
Char   chr_c =  97 ;//declaration of variable with type char and initialize it
with character 97
```

3.7.2. ASCIICharTable

unsigned char

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2	SPC 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127
8	€ 128	129	‚ 130	ƒ 131	„ 132	… 133	† 134	‡ 135	ˆ 136	‰ 137	Š 138	‹ 139	Œ 140	141	Ž 142	143
9	144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	˜ 152	™ 153	š 154	› 155	œ 156	157	ž 158	Ÿ 159
A	160	¡ 161	¢ 162	£ 163	¤ 164	¥ 165	¦ 166	§ 167	¨ 168	© 169	ª 170	« 171	¬ 172	- 173	® 174	¯ 175
B	° 176	± 177	² 178	³ 179	´ 180	µ 181	¶ 182	· 183	¸ 184	¹ 185	º 186	» 187	¼ 188	½ 189	¾ 190	¿ 191
C	À 192	Á 193	Â 194	Ã 195	Ä 196	Å 197	Æ 198	Ç 199	È 200	É 201	Ê 202	Ë 203	Ì 204	Í 205	Î 206	Ï 207
D	Ð 208	Ñ 209	Ò 210	Ó 211	Ô 212	Õ 213	Ö 214	× 215	Ø 216	Ù 217	Ú 218	Û 219	Ü 220	Ý 221	Þ 222	ß 223
E	à 224	á 225	â 226	ã 227	ä 228	å 229	æ 230	ç 231	è 232	é 233	ê 234	ë 235	ì 236	í 237	î 238	ï 239
F	ð 240	ñ 241	ò 242	ó 243	ô 244	õ 245	ö 246	÷ 247	ø 248	ù 249	ú 250	û 251	ü 252	ý 253	þ 254	ÿ 255

byte

Example

Unsigned Char chr_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y

A byte stores an 8-bit unsigned number, from 0 to 255.

Int

Example

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

Integers are the primary data-type for number storage. int stores a 16-bit (2-byte) value.

This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).
The int size varies from board to board. On the Arduino Due, for example, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

Example
int counter = 32 ;// declaration of variable with type int and initialize it with 32

Unsigned int
Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

Example
Unsigned int counter= 60 ; // declaration of variable with type unsigned int and initialize it with 60

Word
On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

Example
word w = 1000 ;//declaration of variable with type word and initialize it with 1000

Long
Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

Example
Long velocity= 102346 ;//declaration of variable with type Long and initialize it with 102346

unsigned long
Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006

short

A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

Float

short val= 13 ;//declaration of variable with type short and initialize it with 13

Data type for floating-point number is a number that has a decimal point. Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.

Floating-point numbers can be as large as 3.4028235E+38 and as low as 3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

float num = 1.352;//declaration of variable with type float and initialize it with 1.352

double

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352

3. Arduino – Variables and Constant

Before we start explaining the variable types, a very important subject we need to make sure, you fully understand is called the variable scope.

What is Variable Scope?

Variables in C programming language, which Arduino uses, have a property called scope. A scope is a region of the program and there are three places where variables can be declared. They are:

• Inside a function or a block, which is called local variables.
• In the definition of function parameters, which is called formal parameters.
• Outside of all functions, which is called global variables.

Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by the statements that are inside that function or block of code. Local variables are not known to function outside their own. Following is the example using local variables:

```
Void setup ()
{
}
Void loop ()
{
     int  x , y ;
  int  z ;        Local variable declaration

     x=  0;
  y=0;             actual initialization
     z=10;
}
```

GlobalVariables

Globalvariablesaredefinedoutsideofallthefunctions,usuallyatthetopoftheprogram. Theglobalvariableswillholdtheirvaluethroughoutthelife-timeofyourprogram.

Aglobalvariablecanbeaccessedbyanyfunction.Thatis,aglobalvariableisavailable forusethroughoutyourentireprogramafteritsdeclaration.

Thefollowingexampleusesglobalandlocalvariables:

```
Int T , S ;
float c =0 ; Global variable declaration
Void setup ()
{
}
Void loop ()
{
     int  x , y ;
  int  z ; Local variable declaration

     x=  0;
  y=0; actual initialization
     z=10;
}
```

3.7.3. Arduino - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Comparison Operators
- Boolean Operators
- Bitwise Operators
- Compound Operators

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20 then −

Operator name	Operator simple	Description	Example
assignment operator	=	Stores the value to the right of the equal sign in the variable to the left of the equal sign.	A = B
addition	+	Adds two operands	A + B will give 30
subtraction	-	Subtracts second operand from the first	A - B will give -10
multiplication	*	Multiply both operands	A * B will give 200
division	/	Divide numerator by denominator	B / A will give 2
modulo	%	Modulus Operator and remainder of after an integer division	B % A will give 0

Comparison Operators

Assume variable A holds 10 and variable B holds 20 then −

Operator name	Operator simple	Description	Example
equal to	==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true
not equal to	!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true
less than	<	Checks if the value of left operand is less than the value of right operand, if yes then condition	(A < B) is true

		becomes true.	
greater than	>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true
less than or equal to	<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true
greater than or equal to	>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true

Boolean Operators

Assume variable A holds 10 and variable B holds 20 then −

Operator name	Operator simple	Description	Example
and	&&	Called Logical AND operator. If both the operands are non-zero then then condition becomes true.	(A && B) is true
or	||	Called Logical OR Operator. If any of the two operands is non-zero then then condition becomes true.	(A || B) is true
not	!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false

Bitwise Operators

Assume variable A holds 60 and variable B holds 13 then −

Operator name	Operator simple	Description	Example
and	&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
or	|	Binary OR Operator copies a bit if it exists in either operand	(A | B) will give 61 which is 0011 1101
xor	^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001

not	~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
shift left	<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
shift right	>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Compound Operators

Assume variable A holds 10 and variable B holds 20 then −

Operator name	**Operator simple**	**Description**	**Example**
increment	++	Increment operator, increases integer value by one	A++ will give 11
decrement	--	Decrement operator, decreases integer value by one	A-- will give 9
compound addition	+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand	B += A is equivalent to B = B+ A
compound subtraction	-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand	B -= A is equivalent to B = B - A
compound multiplication	*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand	B*= A is equivalent to B = B* A
compound division	/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand	B /= A is equivalent to B = B / A
compound modulo	%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand	B %= A is equivalent to B = B % A
compound bitwise or	|=	bitwise inclusive OR and assignment operator	A |= 2 is same as A = A | 2

compound bitwise and	&=	Bitwise AND assignment operator	A &= 2 is same as A = A & 2

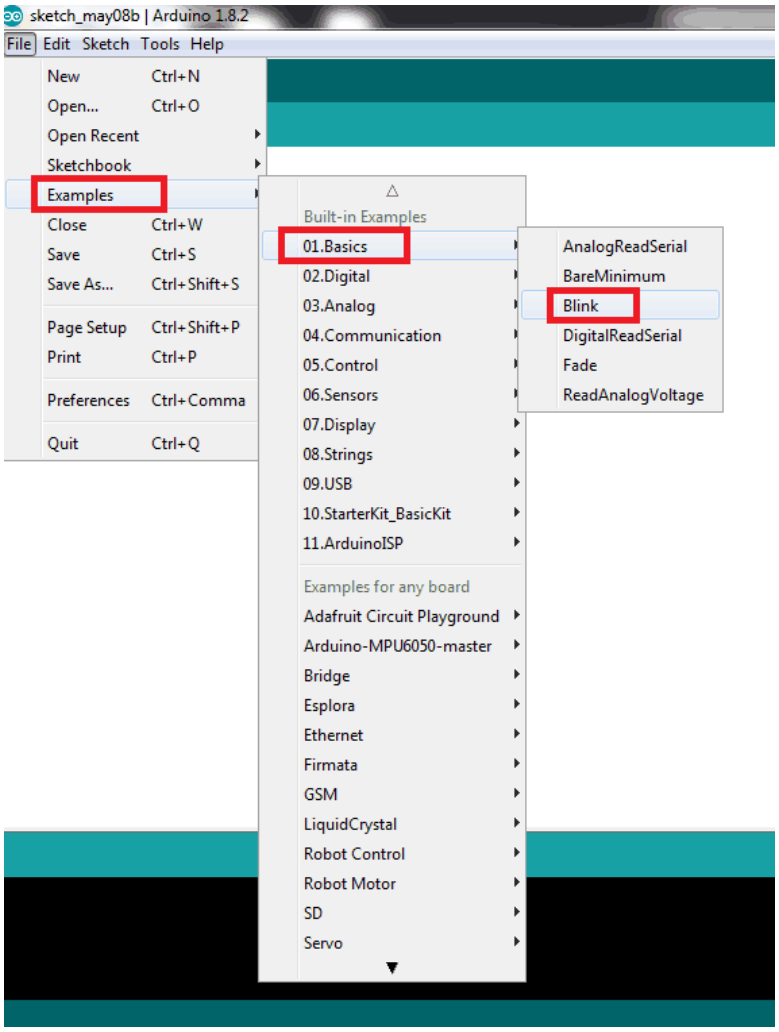
Arduino IDE

The Arduino IDE is very simple and easy to use.

Let's see how to use it with the help of Blink example that comes in the Built-In examples with the IDE.

1. Open the Arduino IDE from the folder you installed/unzipped it in.
2. In the File tab, go to the Examples option. Here you will find a list of all the Built-In examples that come along with the IDE. You will also see Examples For Any Board in below the Built-In examples.

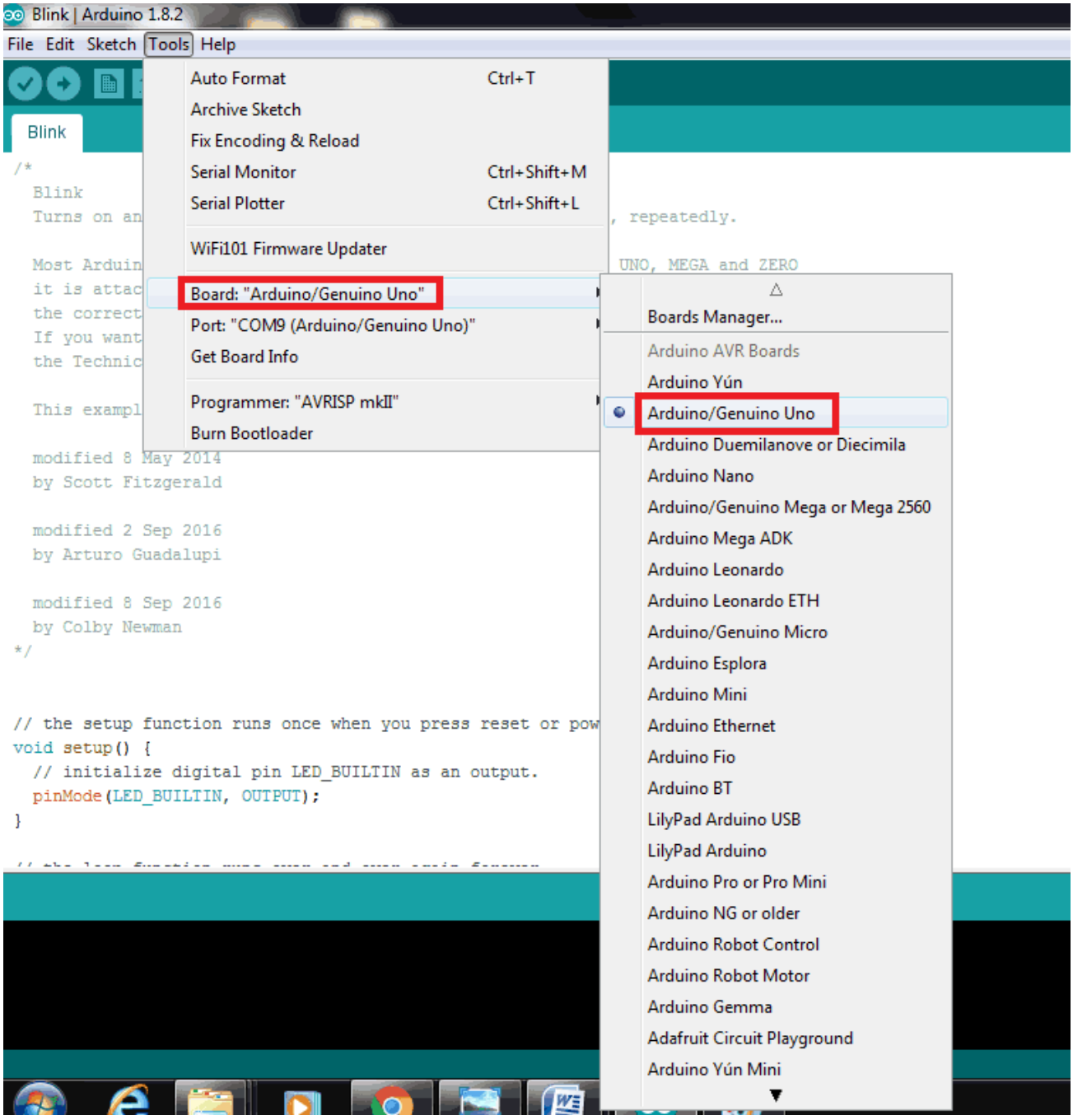
Here, we will see the Blink example which blinks the LED on the Arduino board.



3. Once you click on Blink, a new window will open with the sketch (Arduino refers to codes written in its IDE as sketch) for Blink.

4. Before burning this code into the Arduino board, we need to configure the IDE for the board we need to burn this sketch into. We also need to select the appropriate communication port and the method used for burning the code.

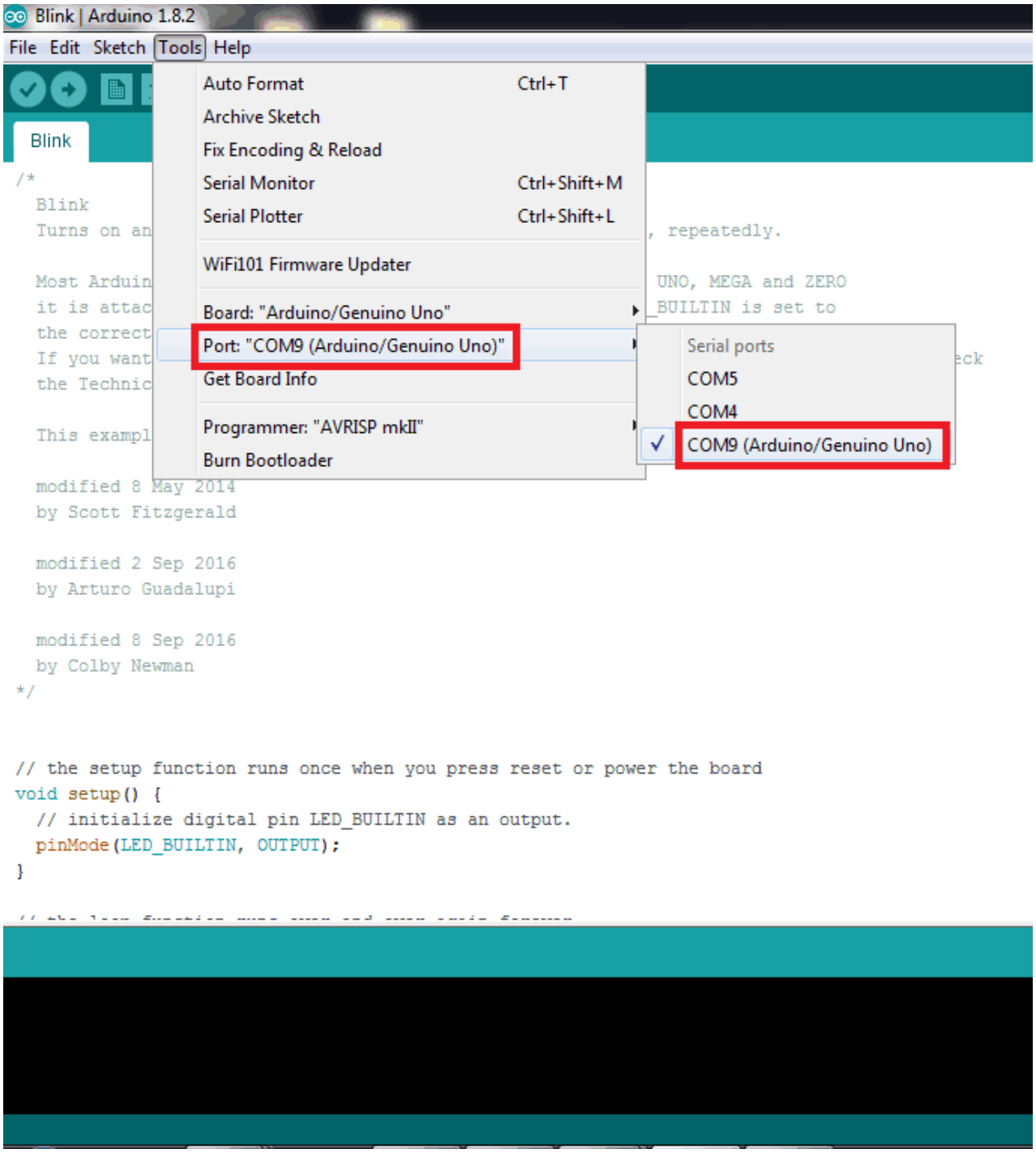
5. Go to the Tools tab and find the Board option in it. It will have a list of all the Arduino Boards in it. Select the board you are using from that list. Here, we have used Arduino/Genuino Uno Board.



Arduino and Genuino are one and the same thing. Outside the USA, the Arduino boards are called Genuino.

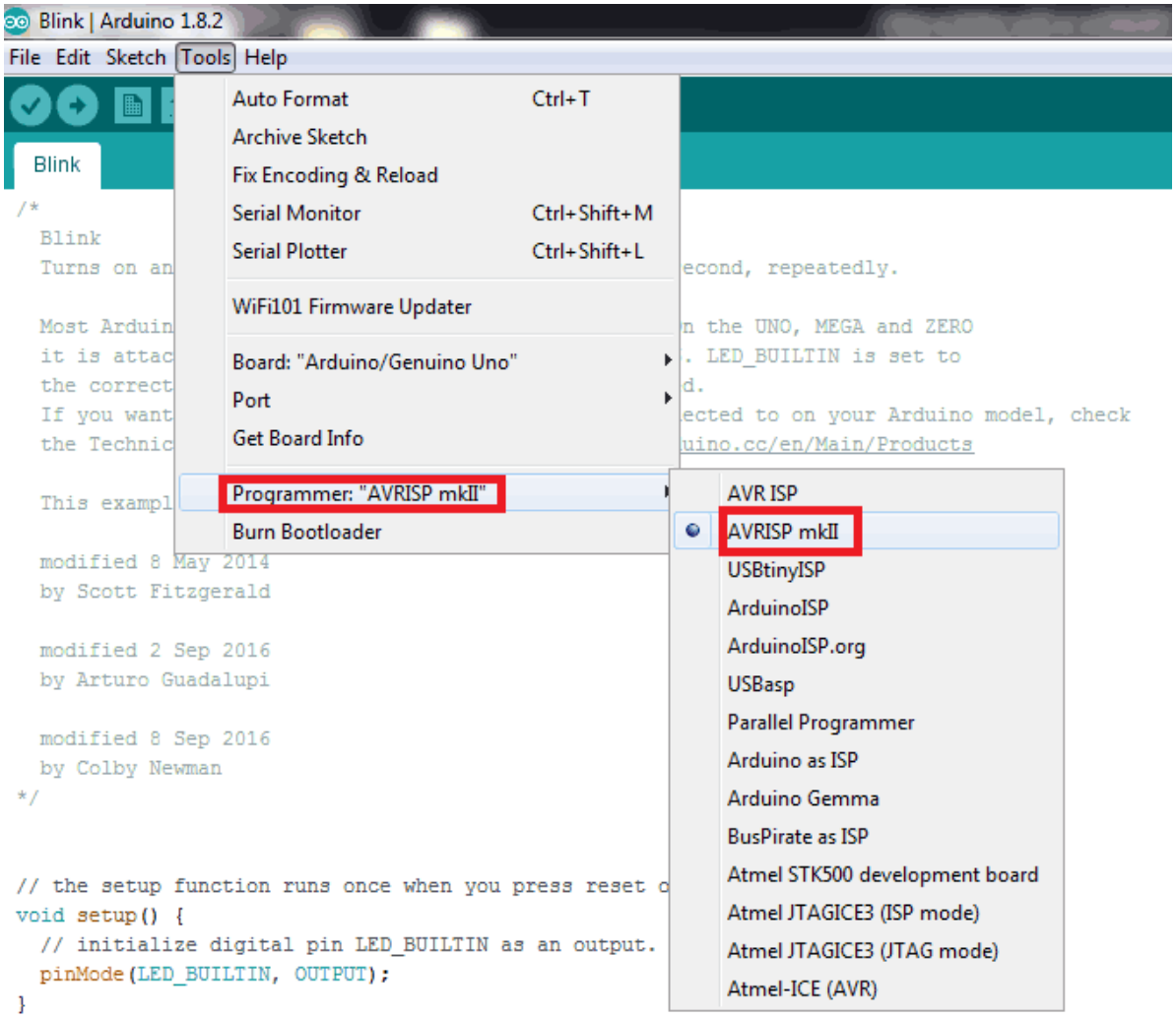
6. Now we need to select the appropriate communication port over which the computer/laptop will communicate with the Arduino board. Under the Tools tab, you will

find the Port option. Select the port you have connected the Arduino board to from the Ports available in this list.



7. Now that we have selected the board and the communication port, we need to select the appropriate programmer.

We will be using the AVR ISP mk2 programmer.

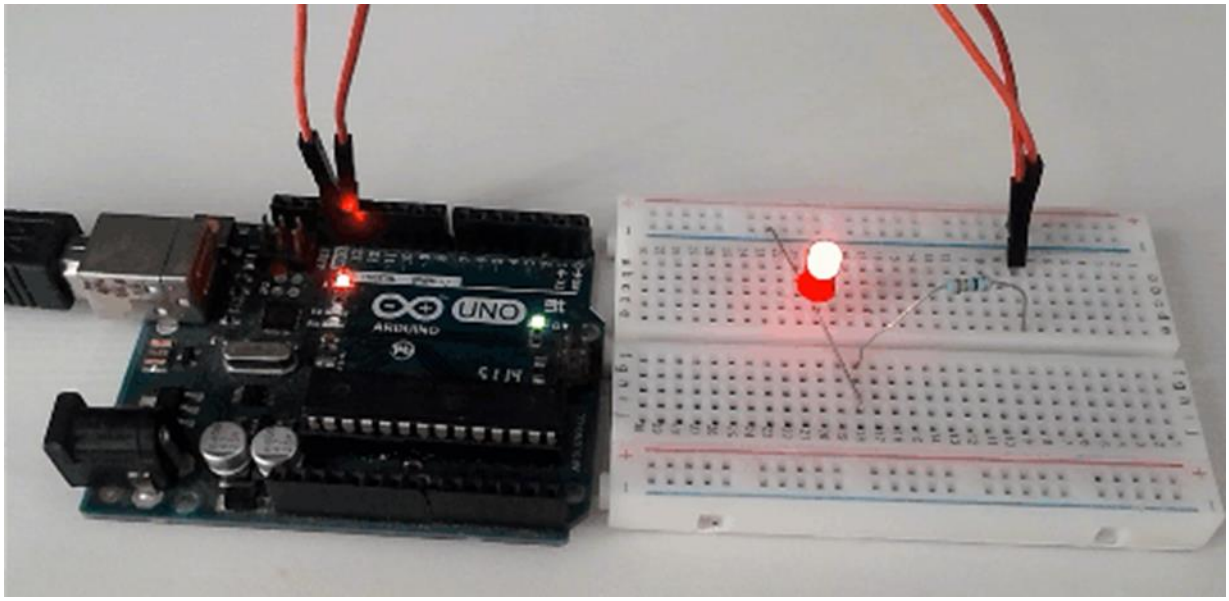


8. On the Arduino IDE, you will find 6 unique buttons. These are described below.

-  This is used to verify (Arduino refers to compiling as verifying) the sketch.
-  This is used to upload (Arduino refers burning a program as uploading) the sketch onto the board.
-  This is used to create a new sketch.
-  This is used to open an existing sketch or built-in example.
-  This is used to save the current sketch.
-  This is used to open the serial monitor that comes with the Arduino IDE.

Note : You can use any other serial terminal program if you want instead of the Arduino serial monitor. The serial monitor that comes with Arduino IDE can open only one serial port for communication. If you need to observe multiple ports (multiple Arduino boards connected to the same computer/laptop), you need to use other programs like Putty, RealTerm, TeraTerm, etc.

9. Now that we have selected the appropriate board, communication port, and the programmer, we need to upload the Blink sketch on to the board.
We can either verify the sketch before the upload process or skip the verification process.
The IDE does the part of verifying the sketch before uploading as a part of the upload process.
10. Since this is a built-in example sketch, we will not get any warnings or errors upon verifying. If there are any warnings or errors, they are displayed by the IDE in the black coloured area below the area for writing code. This area is shown in the image in the next point, highlighted in red.
11. Now upload the sketch onto the board. When the upload process is done, you will get a done uploading message. You should be able to see the LED on your Arduino board blinking at 1 second intervals.



12. This process needs to be followed for all the sketches that are built-in or created by the user.

Understanding how the Arduino IDE sets up its file directory system can spare you some confusion when it comes to saving, organizing and opening your Arduino sketches (or sketches you have downloaded from the internet).

This week's episode covers the following:

1. The Arduino Sketchbook folder
2. How to change the default location where your Arduino sketches get saved
3. What happens when an Arduino file does not have an enclosing sketch folder of the same name
4. Where and how multiple tabs get saved in an Arduino sketch
5. Why the pancreas of a woodchuck is better suited than our own

The Arduino Sketchbook Folder And Changing The Default Save Location

The default location where Arduino sketches you write will be saved is called the Sketchbook.

The Sketchbook is simply a folder on your computer like any other. It acts as a handy repository for sketches and is also where add-on code libraries get saved.

You can see the sketches in the Sketchbook folder by going to File > Sketchbook.

The default name of the Sketchbook folder is "Arduino" and the default location of the Sketchbook folder is in the "My Documents" folder (or just "Documents" for Mac users).

If your Sketchbook does not appear to be in this default location, you can see exactly where it is by opening the Arduino IDE and going to Arduino > Preferences.

The default file path for your sketches is listed at the top of Arduino Preferences window. Mine is:
/Users/michaelJames/Documents/Arduino
When I save a file in the Arduino IDE, this "Arduino" folder is the default location where the sketch will be saved, unless I choose to change that location by selecting a different folder on my computer's directory.

If you want to change this default location, you click the Browse button next to the file directory path and choose a different place. Pretty simple.

3.8. Arduino Sketches

A *sketch* is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.

The first few lines of the Blink sketch are a *comment*:

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
  */
```

Everything between the
/*

and
*/
is ignored by the Arduino when it runs the sketch (the
*
at the start of each line is only there to make the comment look pretty, and isn't required). It's there for people reading the code: to explain what the program does, how it works, or why it's written the way it is. It's a good practice to comment your sketches, and to keep the comments up-to-date when you modify the code. This helps other people to learn from or modify your code.

There's another style for short, single-line comments. These start with
//
and continue to the end of the line. For example, in the line:

1int ledPin = 13;*// LED connected to digital pin 13*the message "LED connected to digital pin 13" is a comment.

Variables

A *variable* is a place for storing a piece of data. It has a name, a type, and a value. For example, the line from the Blink sketch above declares a variable with the name ledPin, the type int, and an initial value of 13. It's being used to indicate which Arduino pin the LED is connected to. Every time the name ledPin appears in the code, its value will be retrieved. In this case, the person writing the program could have chosen not to bother creating the ledPin variable and instead have simply written 13 everywhere they needed to specify a pin number. The advantage of using a variable is that it's easier to move the LED to a different pin: you only need to edit the one line that assigns the initial value to the variable. Often, however, the value of a variable will change while the sketch runs. For example, you could store the value read from an input into a variable.

Functions

A *function* (otherwise known as a *procedure* or *sub-routine*) is a named piece of code that can be used from elsewhere in a sketch. For example, here's the definition of the
setup()

function from the Blink example:

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

The first line provides information about the function, like its name, "setup". The text before and after the name specify its return type and parameters: these will be explained later. The code between the { and } is called the *body* of the function: what the function does.

You can *call* a function that's already been defined (either in your sketch or as part of the Arduino language). For example, the line pinMode(ledPin, OUTPUT); calls the pinMode() function, passing it two *parameters*: ledPin and OUTPUT. These parameters are used by the
pinMode() function to decide which pin and mode to set.

pinMode(), digitalWrite(), and delay()

The pinMode() function configures a pin as either an input or an output. To use it, you pass it the number of the pin to configure and the constant INPUT or OUTPUT. When configured as an input, a pin can detect the state of a sensor like a pushbutton. As an output, it can drive an actuator like an LED.

The digitalWrite()
functions outputs a value on a pin. For example, the line:

digitalWrite(ledPin, HIGH);
set the ledPin (pin 13) to HIGH, or 5 volts. Writing a LOW to pin connects it to ground, or 0 volts.

The delay() causes the Arduino to wait for the specified number of milliseconds before continuing on to the next line. There are 1000 milliseconds in a second, so the line:
delay(1000);
creates a delay of one second.

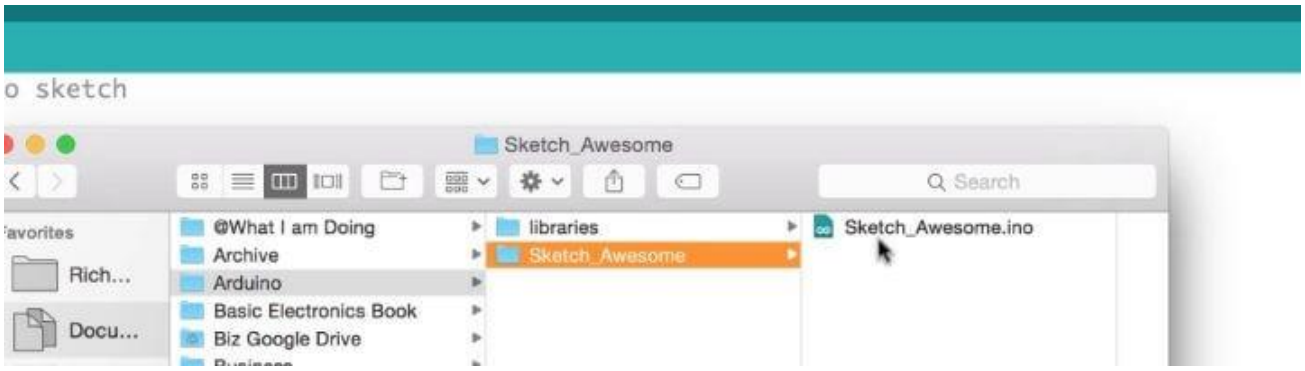
setup() and loop()

There are two special functions that are a part of every Arduino sketch:
setup() and loop(). The setup() is called once, when the sketch starts. It's a good place to do setup tasks like setting pin modes or initializing libraries. The loop() function is called over and over and is heart of most sketches. You need to include both functions in your sketch, even if you don't need them for anything.

3.8. Arduino Sketches using software

If you go into your file directory system and open up the Sketchbook folder (again, named “Arduino” by default), you may see a bunch of folders that you didn’t make.This is because every Arduino file must be inside a folder that has the same name as the file (there are some exceptions to this that we’ll talk about in a moment).

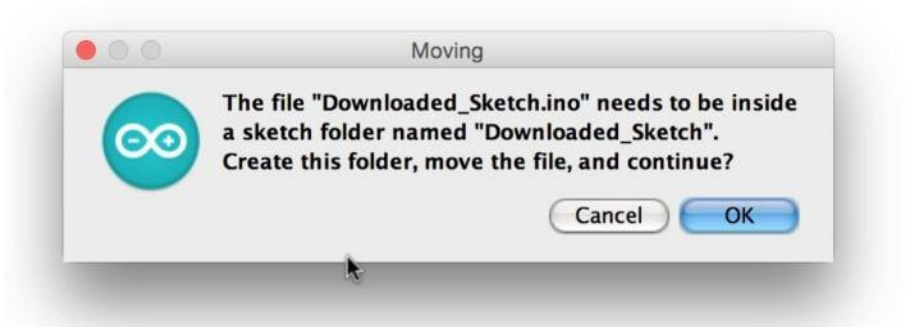
Let me say that one more time because it is really important to understand.Every Arduino file must be inside a folder that has the same name as the fileWhen I write a new sketch and save it, the Arduino IDE automatically creates a new folder that has the exact same name as the sketch itself. The Arduino file (which has the extension .ino) is placed inside this enclosing folder, which is called a sketch folder.



If you go into the Sketchbook folder and change the name of the enclosing folder, it will create some issues. The first issue is that when you go to File > Sketchbook, the sketch will no longer show up! If you want to open this sketch you need to go to the .ino file in your directory and open it from there.

If you open a .ino file that is not inside an enclosing sketch folder of the exact same name, then you will get a pop-up from the Arduino IDE that says:

"The file "sketch_name.ino" needs to be inside a sketch folder named "sketch_name". Create this folder, move the file, and continue?"

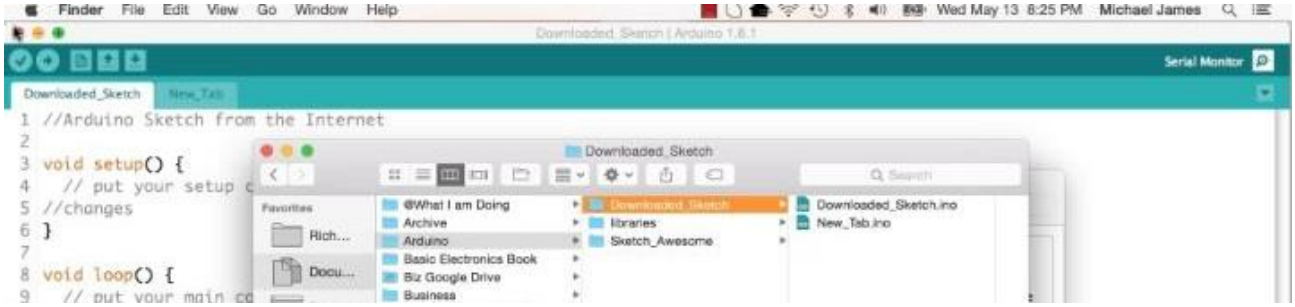


If you choose Cancel, the sketch will not open. If you choose OK, then a folder gets created (it will have the same name as the sketch) and the .ino file is placed inside it.

This sketch folder will be created in whatever directory the .ino file was that you tried to open. For example, if you tried to open a .ino file that was in your My Downloads folder, then the enclosing sketch folder also will be created inside the My Downloads folder.

Saving Tabs In Arduino

The exception to the rule about the sketch folder having the same name as the .ino file is when you create multiple tabs in an Arduino sketch.
The additional tabs do NOT need to bear the same name as the enclosing sketch folder.



Once you get a handle on some of these intricacies of the Arduino IDE file system, it can really help to clear things up.
Arduino Shields

3.9. Arduino Pins

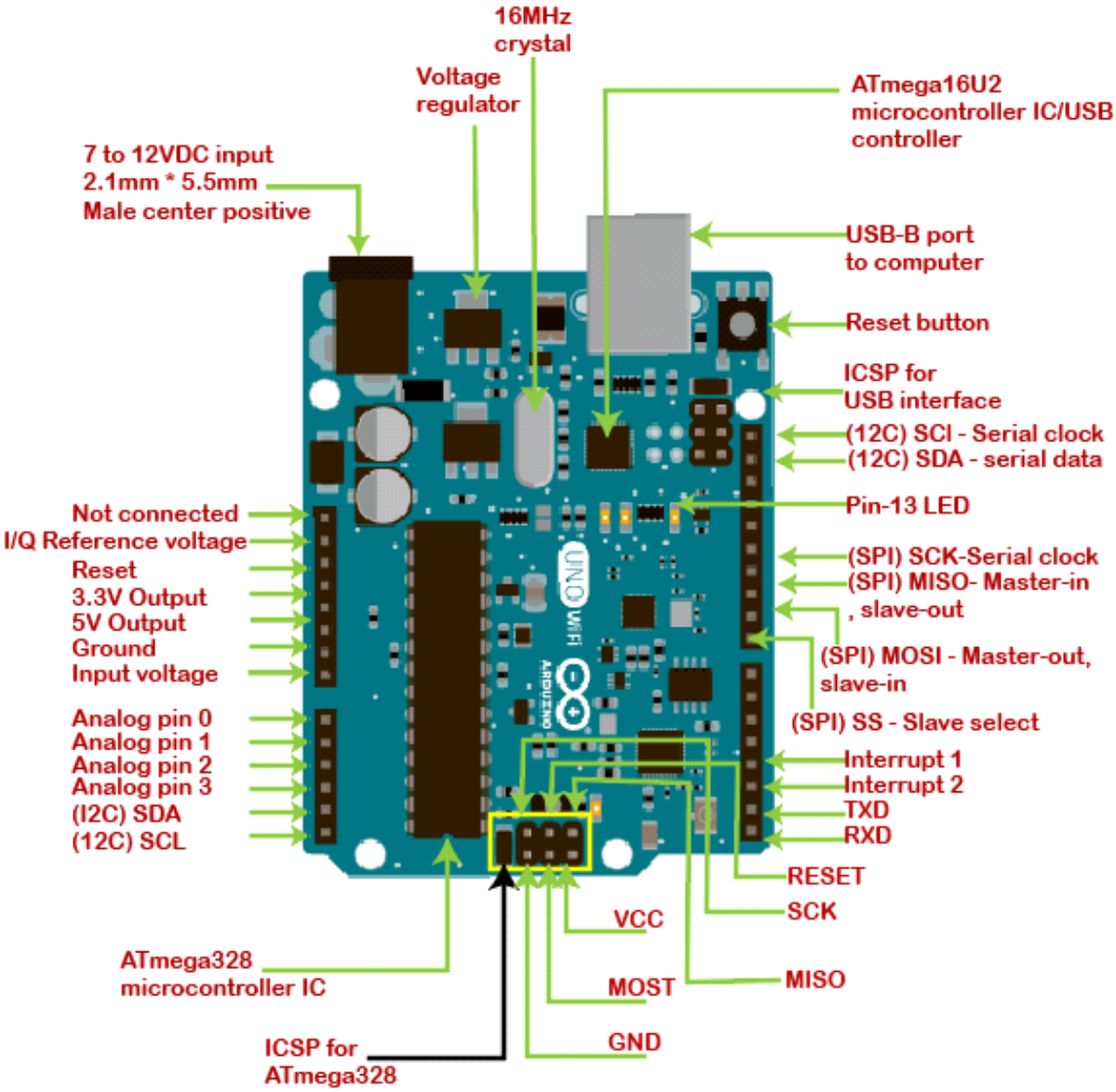
An Introduction to Arduino Uno Pinout

Arduino Uno is based on the ATmega328 by Atmel. The Arduino Uno pinout consists of 14 digital pins, 6 analog inputs, a power jack, USB connection and ICSP header. The versatility of the pinout provides many different options such as driving motors, LEDs, reading sensors and more. In this post, we'll go over the capabilities of the Arduino Uno pinout.

Arduino Uno pinout - Power Supply

There are 3 ways to power the Arduino Uno:

- **Barrel Jack** - The Barrel jack, or DC Power Jack can be used to power your Arduino board. The barrel jack is usually connected to a wall adapter. The board can be powered by 5-20 volts but the manufacturer recommends to keep it between 7-12 volts. Above 12 volts, the regulators might overheat, and below 7 volts, might not suffice.



Arduino Uno Pinout – Diagram

- **VIN Pin** - This pin is used to power the Arduino Uno board using an external power source. The voltage should be within the range mentioned above.
- **USB cable** - when connected to the computer, provides 5 volts at 500mA.

There is a polarity protection diode connecting between the positive of the barrel jack to the VIN pin, rated at 1 Ampere.

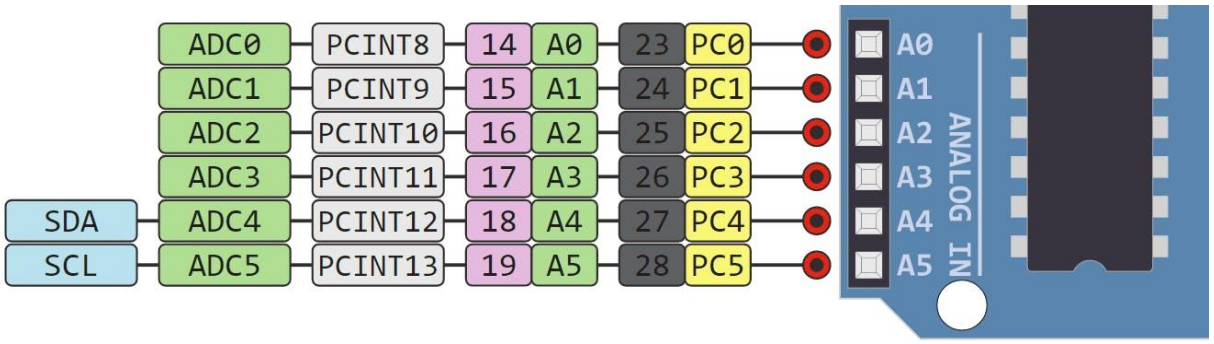
The power source you use determines the power you have available for your circuit. For instance, powering the circuit using the USB limits you to 500mA. Take into consideration that this is also used for powering the MCU, its peripherals, the on-board regulators, and the components connected to it. When powering your circuit through the barrel jack or VIN, the maximum capacity available is determined by the 5 and 3.3 volts regulators on-board the Arduino.

- **5v and 3v3**

They provide regulated 5 and 3.3v to power external components according to manufacturer specifications.

Arduino Uno Pinout - Analog IN

The Arduino Uno has 6 **analog pins**, which utilize ADC (Analog to Digital converter).
These pins serve as analog inputs but can also function as digital inputs or digital outputs.



- **GND**

In the Arduino Uno pinout, you can find 5 GND pins, which are all interconnected.
The GND pins are used to close the electrical circuit and provide a common logic reference level throughout your circuit. Always make sure that all GNDs (of the Arduino, peripherals and components) are connected to one another and have a common ground.

- **RESET** - resets the Arduino
- **IOREF** - This pin is the input/output reference. It provides the voltage reference with which the microcontroller operates.

Analog to Digital Conversion

ADC stands for Analog to Digital Converter. ADC is an electronic circuit used to convert analog signals into digital signals. This digital representation of analog signals allows the processor (which is a digital device) to measure the analog signal and use it through its operation.
Arduino Pins A0-A5 are capable of reading analog voltages. On Arduino the ADC has 10-bit resolution, meaning it can represent analog voltage by 1,024 digital levels. The ADC converts voltage into bits which the microprocessor can understand.
One common example of an ADC is Voice over IP (VoIP). Every smartphone has a microphone that converts sound waves (voice) into analog voltage. This goes through the device's ADC, gets converted into digital data, which is transmitted to the receiving side over the internet.

Arduino Uno Pinout - Digital Pins

- Pins 0-13 of the Arduino Uno serve as digital input/output pins.
- Pin 13 of the Arduino Uno is connected to the built-in LED.

In the Arduino Uno - pins 3,5,6,9,10,11 have PWM capability.

It's important to note that:

- Each pin can provide/sink up to 40 mA max. But the recommended current is 20 mA.
- The absolute max current provided (or sank) from all pins together is 200mA

What does digital mean?

Digital is a way of representing voltage in 1 bit: either 0 or 1. Digital pins on the Arduino are pins designed to be configured as inputs or outputs according to the needs of the user. Digital pins are either on or off. When ON they are in a HIGH voltage state of 5V and when OFF they are in a LOW voltage state of 0V.

On the Arduino, When the digital pins are configured as **output**, they are set to 0 or 5 volts. When the digital pins are configured as **input,** the voltage is supplied from an external device. This voltage can vary between 0-5 volts which is converted into digital representation (0 or 1). To determine this, there are 2 thresholds:

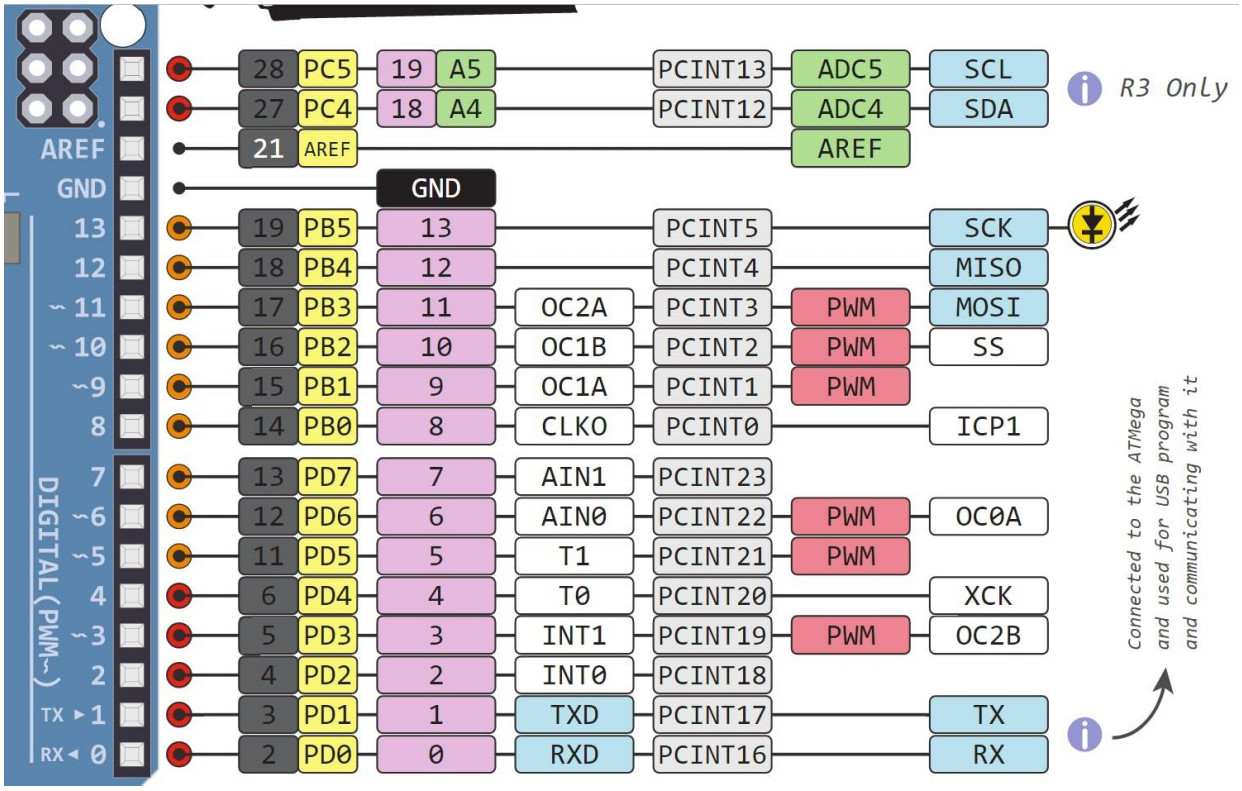
- Below 0.8v - considered as 0.
- Above 2v - considered as 1.

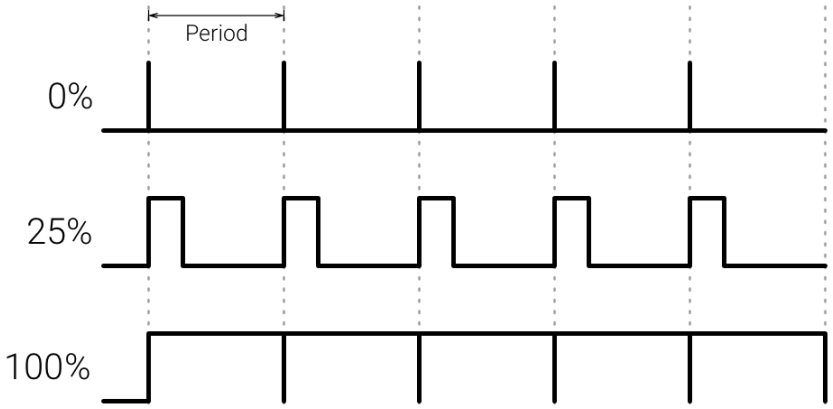
When connecting a component to a digital pin, make sure that the logic levels match. If the voltage is in between the thresholds, the returning value will be undefined.

What is PWM?

In general, Pulse Width Modulation (PWM) is a modulation technique used to encode a message into a pulsing signal. A PWM is comprised of two key components: **frequency** and **duty cycle**. The PWM frequency dictates how long it takes to complete a single cycle (period) and how quickly the signal fluctuates from high to low. The duty cycle determines how long a signal stays high out of the total period. Duty cycle is represented in percentage.

In Arduino, the PWM enabled pins produce a constant frequency of ~ 500Hz, while the duty cycle changes according to the parameters set by the user. See the following illustration:





PWM signals are used for speed control of DC motors, dimming LEDs and more.

Communication Protocols

Serial (TTL) - Digital pins 0 and 1 are the serial pins of the Arduino Uno.
They are used by the onboard USB module.

What is Serial Communication?

Serial communication is used to exchange data between the Arduino board and another serial device such as computers, displays, sensors and more. Each Arduino board has at least one serial port. Serial communication occurs on digital pins 0 (RX) and 1 (TX) as well as via USB. Arduino supports serial communication through digital pins with the SoftwareSerial Library as well. This allows the user to connect multiple serial-enabled devices and leave the main serial port available for the USB.

Software serial and hardware serial - Most microcontrollers have hardware designed to communicate with other serial devices. Software serial ports use a pin-change interrupt system to communicate. There is a built-in library for Software Serial communication. Software serial is used by the processor to simulate extra serial ports. The only drawback with software serial is that it requires more processing and cannot support the same high speeds as hardware serial.

SPI - SS/SCK/MISO/MOSI pins are the dedicated pins for SPI communication. They can be found on digital pins 10-13 of the Arduino Uno and on the ICSP headers.

What is SPI?

Serial Peripheral Interface (SPI) is a serial data protocol used by microcontrollers to communicate with one or more external devices in a bus like connection. The SPI can also be used to connect 2 microcontrollers. On the SPI bus, there is always one device that is denoted as a Master device and all the rest as Slaves. In most cases, the microcontroller is the Master device. The SS (Slave Select) pin determines which device the Master is currently communicating with.

SPI enabled devices always have the following pins:

- MISO (Master In Slave Out) - A line for sending data to the Master device
- MOSI (Master Out Slave In) - The Master line for sending data to peripheral devices
- SCK (Serial Clock) - A clock signal generated by the Master device to synchronize data transmission.

I2C - SCL/SDA pins are the dedicated pins for I2C communication. On the Arduino Uno they are found on Analog pins A4 and A5.

What is I2C?

I2C is a communication protocol commonly referred to as the “I2C bus”. The I2C protocol was designed to enable communication between components on a single circuit board. With I2C there are 2 wires referred to as SCL and SDA.

- SCL is the clock line which is designed to synchronize data transfers.
- SDA is the line used to transmit data.

Each device on the I2C bus has a unique address, up to 255 devices can be connected on the same bus.

Aref - Reference voltage for the analog inputs.

Interrupt - INT0 and INT1. Arduino Uno has two external interrupt pins.

External Interrupt - An external interrupt is a system interrupt that occurs when outside interference is present. Interference can come from the user or other hardware devices in the network. Common uses for these interrupts in Arduino are reading the frequency a square wave generated by encoders or waking up the processor upon an external event.

Arduino has two forms of interrupt:

- External
- Pin Change

There are two external interrupt pins on the ATmega168/328 called INT0 and INT1. both INT0 and INT1 are mapped to pins 2 and 3. In contrast, Pin Change interrupts can be activated on any of the pins.

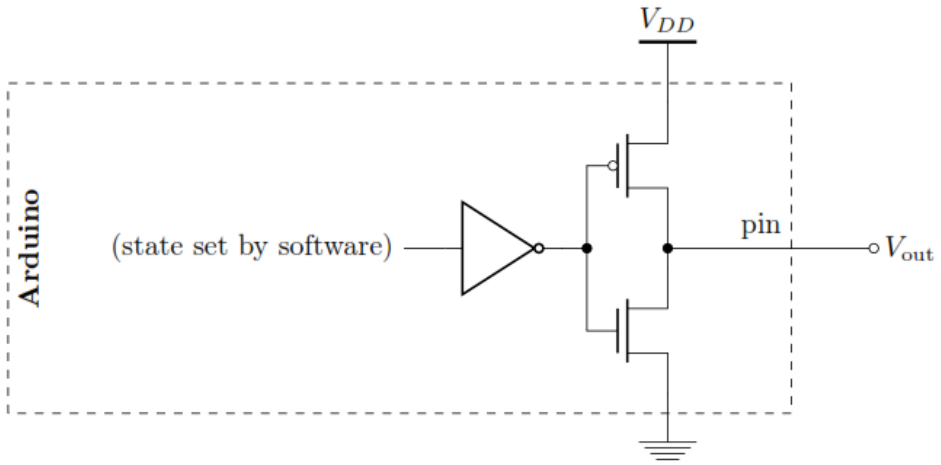
Arduino Uno Pinout - ICSP Header

ICSP stands for In-Circuit Serial Programming. The name originated from In-System Programming headers (ISP). Manufacturers like Atmel who work with Arduino have developed their own in-circuit serial programming headers. These pins enable the user to program the Arduino boards' firmware. There are six ICSP pins available on the Arduino board that can be hooked to a programmer device via a programming cable.

3.10. Input/output pins on the Arduino

Output pins

An output pin provides VDD or 0 V, by making a connection to VDD or ground via a transistor. You set its state to HIGH (for VDD) or LOW (for 0 V) using the digitalWrite() function. A (simplified) schematic of an output pin is shown below. You might notice that it looks a bit like a CMOS inverter (or rather, two).

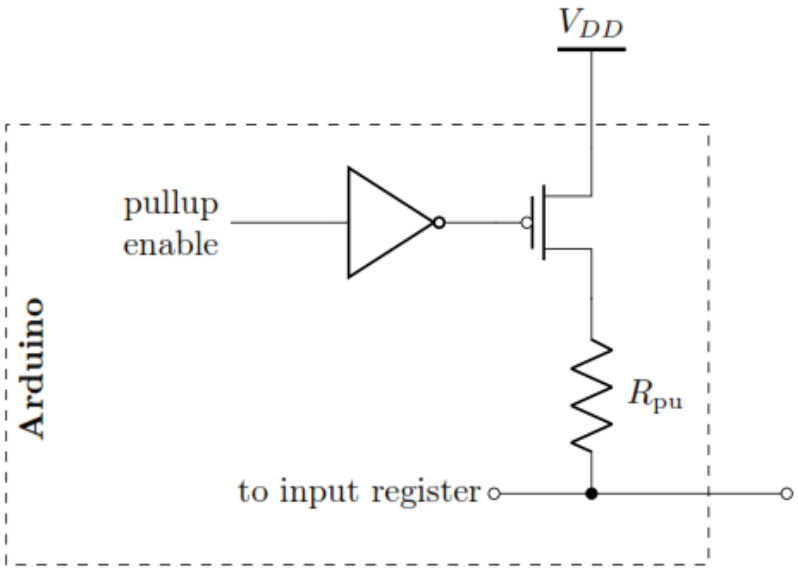


The transistors in the output pin have non-negligible on resistance, so aren't suitable for driving large loads. When talking about this resistance in relation to an output pin, we call it the output resistance of the pin—in other words, the resistance "seen" by a device connected to the pin. Since this resistance might depend on the state of the pin (HIGH or LOW), it actually has two output resistances. You'll measure both in prelab 2b.

Input pins

An input pin reads the voltage on the pin as if it were a voltmeter, and returns either HIGH (1) in software if the voltage is close to VDD , or LOW (0) if it is close to 0 V. An input pin can be read using the digitalRead() function.

The value returned by digitalRead() is unpredictable (i.e., could be either HIGH or LOW) when the input voltage is not close to either VDD or 0 V. The precise meaning of "close" varies between microcontrollers, but for the Adafruit Metro Mini1 in our circuit, the input pin voltage needs to be at least 0.6VDD to qualify as HIGH, and at most 0.3VDD to qualify as LOW.



Arduino - I/O Functions

The pins on the Arduino board can be configured as either inputs or outputs. We will explain the functioning of the pins in those modes. It is important to note that a majority of Arduino analog pins, may be configured, and used, in exactly the same manner as digital pins.

Pins Configured as INPUT

Arduino pins are by default configured as inputs, so they do not need to be explicitly declared as inputs with **pinMode()** when you are using them as inputs. Pins configured this way are said to be in a high-impedance state. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megaohm in front of the pin.

This means that it takes very little current to switch the input pin from one state to another. This makes the pins useful for such tasks as implementing a capacitive touch sensor or reading an LED as a photodiode.

Pins configured as pinMode(pin, INPUT) with nothing connected to them, or with wires connected to them that are not connected to other circuits, report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Pull-up Resistors

Pull-up resistors are often useful to steer an input pin to a known state if no input is present. This can be done by adding a pull-up resistor (to +5V), or a pull-down resistor (resistor to ground) on the input. A 10K resistor is a good value for a pull-up or pull-down resistor.

Using Built-in Pull-up Resistor with Pins Configured as Input

There are 20,000 pull-up resistors built into the Atmega chip that can be accessed from software. These built-in pull-up resistors are accessed by setting the **pinMode()** as INPUT_PULLUP. This effectively inverts the behavior of the INPUT mode, where HIGH means the sensor is OFF and LOW means the sensor is ON. The value of this pull-up depends on the microcontroller used. On most AVR-based boards, the value is guaranteed to be between 20kΩ and 50kΩ. On the Arduino Due, it is between 50kΩ and 150kΩ. For the exact value, consult the datasheet of the microcontroller on your board.

When connecting a sensor to a pin configured with INPUT_PULLUP, the other end should be connected to the ground. In case of a simple switch, this causes the pin to read HIGH when the switch is open and LOW when the switch is pressed. The pull-up resistors provide enough current to light an LED dimly connected to a pin configured as an input. If LEDs in a project seem to be working, but very dimly, this is likely what is going on.

Same registers (internal chip memory locations) that control whether a pin is HIGH or LOW control the pull-up resistors. Consequently, a pin that is configured to have pull-up resistors turned on when the pin is in INPUTmode, will have the pin configured as HIGH if the pin is then switched to an OUTPUT mode with pinMode(). This works in the other direction as well, and an output pin that is left in a HIGH state will have the pull-up resistor set if switched to an input with pinMode().

Example

```
pinMode(3,INPUT) ; // set pin to input without using built in pull up resistor
pinMode(5,INPUT_PULLUP) ; // set pin to input using built in pull up resistor
```

Pins Configured as OUTPUT

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (do not forget the series resistor), or run many sensors but not enough current to run relays, solenoids, or motors.

Attempting to run high current devices from the output pins, can damage or destroy the output transistors in the pin, or damage the entire Atmega chip. Often, this results in a "dead"

pin in the microcontroller but the remaining chips still function adequately. For this reason, it is a good idea to connect the OUTPUT pins to other devices through 470Ω or 1k resistors, unless maximum current drawn from the pins is required for a particular application.

pinMode() Function

The pinMode() function is used to configure a specific pin to behave either as an input or an output. It is possible to enable the internal pull-up resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pull-ups.

pinMode() Function Syntax

```
Void setup ()
{
  pinMode (pin , mode);
}
```

- **pin** − the number of the pin whose mode you wish to set
- **mode** − INPUT, OUTPUT, or INPUT_PULLUP.

Example

```
int button =5;// button connected to pin 5
int LED =6;// LED connected to pin 6

void setup (){
  pinMode(button , INPUT_PULLUP);
// set the digital pin as input with pull-up resistor
  pinMode(button , OUTPUT);// set the digital pin as output
}

void setup (){
If(digitalRead(button )== LOW)// if button pressed {
    digitalWrite(LED,HIGH);// turn on led
    delay(500);// delay for 500 ms
    digitalWrite(LED,LOW);// turn off led
    delay(500);// delay for 500 ms
}
}
```

digitalWrite() Function

The **digitalWrite()** function is used to write a HIGH or a LOW value to a digital pin. If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW. If the

pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

digitalWrite() Function Syntax

```
Void loop()
{
  digitalWrite (pin ,value);
}
```

- **pin** − the number of the pin whose mode you wish to set
- **value** − HIGH, or LOW.

Example

```
int LED =6;// LED connected to pin 6

void setup ()
{
  pinMode(LED, OUTPUT);// set the digital pin as output
}
void setup ()
{
  digitalWrite(LED,HIGH);// turn on led
  delay(500);// delay for 500 ms
  digitalWrite(LED,LOW);// turn off led
  delay(500);// delay for 500 ms
}
```

analogRead( ) function

Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the digitalRead() function. There is a difference between an on/off sensor (which detects the presence of an object) and an analog sensor, whose value continuously changes. In order to read this type of sensor, we need a different type of pin.

In the lower-right part of the Arduino board, you will see six pins marked “Analog In”. These special pins not only tell whether there is a voltage applied to them, but also its value. By using the **analogRead()** function, we can read the voltage applied to one of the pins.

This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts. For example, if there is a voltage of 2.5 V applied to pin number 0, analogRead(0) returns 512.

analogRead() function Syntax

analogRead(pin);

- **pin** − the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Example

```
int analogPin =3;//potentiometer wiper (middle terminal)
// connected to analog pin 3
int val =0;// variable to store the value read

void setup()
{
Serial.begin(9600);// setup serial
}
void loop()
{
  val = analogRead(analogPin);// read the input pin
Serial.println(val);// debug value
}
```

3.11. Arduino Shields

Arduino shields are the boards, which are plugged over the Arduino board to expand its functionalities. There are different varieties of shields used for various tasks, such as Arduino motor shields, Arduino communication shields, etc.

Shield is defined as the hardware device that can be mounted over the board to increase the capabilities of the projects. It also makes our work easy. For example, Ethernet shields are used to connect the Arduino board to the Internet.

The pin position of the shields is similar to the Arduino boards. We can also connect the modules and sensors to the shields with the help of the connection cable.

Arduino motor shields help us to control the motors with the Arduino board.

Why do we need Shields?
The advantages of using Arduino shields are listed below:

- It adds new functionalities to the Arduino projects.

- The shields can be attached and detached easily from the Arduino board. It does not require any complex wiring.
- It is easy to connect the shields by mounting them over the Arduino board.
- The hardware components on the shields can be easily implemented.

Types of Shields

The popular Arduino shields are listed below:

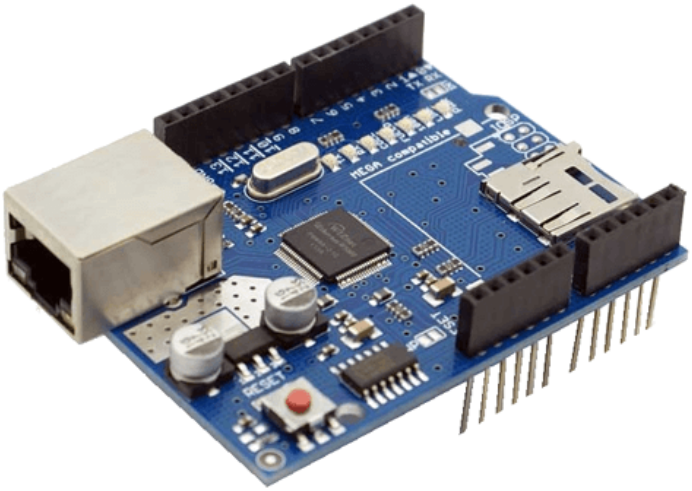
- **Ethernet shield**
- **Xbee Shield**
- **Proto shield**
- **Relay shield**
- **Motor shield**
- **LCD shield**
- **Bluetooth shield**
- **Capacitive Touchpad Shield**

Let's discuss the shields that are listed above:

Ethernet shield

- The Ethernet shields are used to connect the Arduino board to the Internet. We need to mount the shield on the top of the specified Arduino board.
- The USB port will play the usual role to upload sketches on the board.
- The latest version of Ethernet shields consists of a micro SD card slot. The micro SD card slot can be interfaced with the help of the SD card library.
-

The Ethernet shield is shown below:

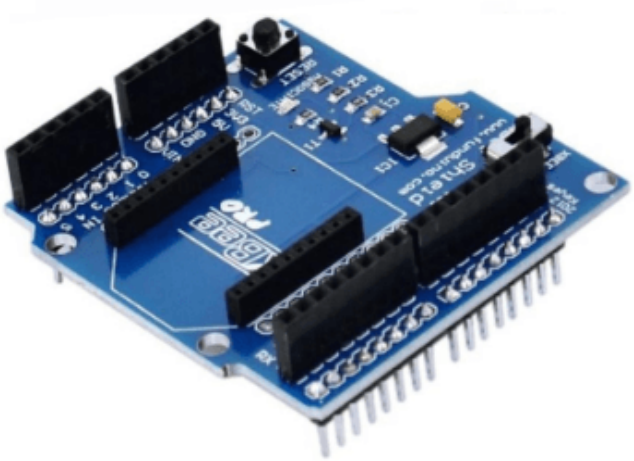


- We can also connect another shield on the top of the Ethernet shield. It means that we can also mount two shields on the top of the Arduino board.

Xbee Shield

- We can communicate wirelessly with the Arduino board by using the Xbee Shield with Zigbee.

- It reduces the hassle of the cable, which makes Xbee a wireless communication model.
- The Xbee wireless module allows us to communicate outdoor upto 300 feet and indoor upto 100 feet.
- The Xbee shield is shown below:



- It can also be used with different models of Xbee.

Proto shield

- Proto shields are designed for custom circuits.
- We can solder electronic circuits directly on the shield.
- The shield consists of two LED pads, two power lines, and SPI signal pads.
- The IOREF (Input Output voltage REFerence) and GND (Ground) are the two power lines on the board.

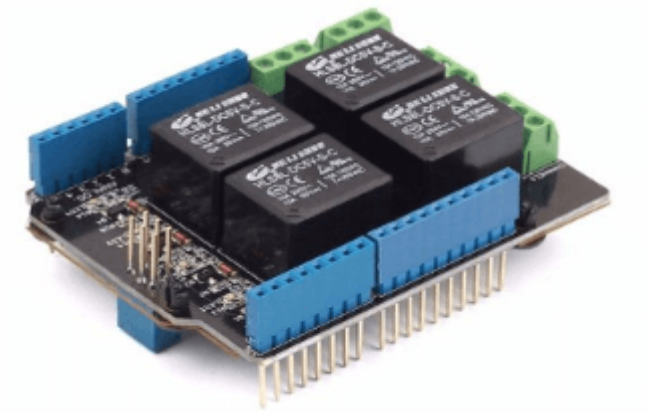
The proto shield is shown below:

- We can also solder the SMD (Surface Mount Device) ICs on the prototyping area. A maximum of 24 pins can be integrated onto the SMD area.

Relay shield

- The Arduino digital I/O pins cannot bear the high current due to its voltage and current limits. The relay shield is used to overcome such situation. It provides a solution for controlling the devices carrying high current and voltage.

- The shield consists of four relays and four LED indicators.
- It also provides NO/NC interfaces and a shield form factor for the simple connection to the Arduino board.
- The LED indicators depicts the ON/OFF condition of each relay.
- The relay used in the structure is of high quality.
- The NO (Normally Open), NC (Normally Closed), and COM pins are present on each relay.
- The relay shield is shown below:

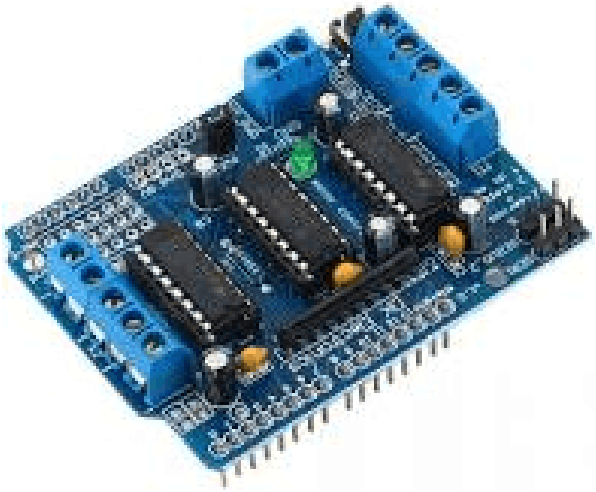


- The applications of the Relay shield include remote control, etc.

Motor shield

- The motor shield helps us to control the motor using the Arduino board.
- It controls the direction and working speed of the motor. We can power the motor shield either by the external power supply through the input terminal or directly by the Arduino.
- We can also measure the absorption current of each motor with the help of the motor shield.
- The motor shield is based on the L298 chip that can drive a step motor or two DC motors. L298 chip is a full bridge IC. It also consists of the heat sinker, which increases the performance of the motor shield.
- It can drive inductive loads, such as solenoids, etc.
- The operating voltage is from 5V to 12V.

The Motor shield is shown below:



- The applications of the motor shield are intelligent vehicles, micro-robots, etc.

LCD shield

- The keypad of LCD (Liquid Crystal Display) shield includes five buttons called as up, down, left, right, and select.
- There are 6 push buttons present on the shield that can be used as a custom menu control panel.
- It consists of the 1602 white characters, which are displayed on the blue backlight LCD.
- The LED present on the board indicates the power ON.
- The five keys present on the board helps us to make the selection on menus and from board to our project.
-

The LCD shield is shown below:

- The LCD shield is popularly designed for the classic boards such as Duemilanove, UNO, etc.

Bluetooth shield

- The Bluetooth shield can be used as a wireless module for transparent serial communication.

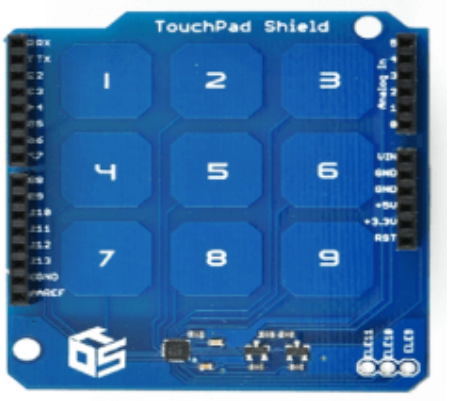
- It includes a serial Bluetooth module. D0 and D1 are the serial hardware ports in the Bluetooth shield, which can be used to communicate with the two serial ports (from D0 to D7) of the Arduino board.
- We can install Groves through the two serial ports of the Bluetooth shield called a Grove connector. One Grove connector is digital, while the other is analog.

The Bluetooth shield is shown below:

- The communication distance of the Bluetooth shield is upto 10m at home without any obstacle in between.

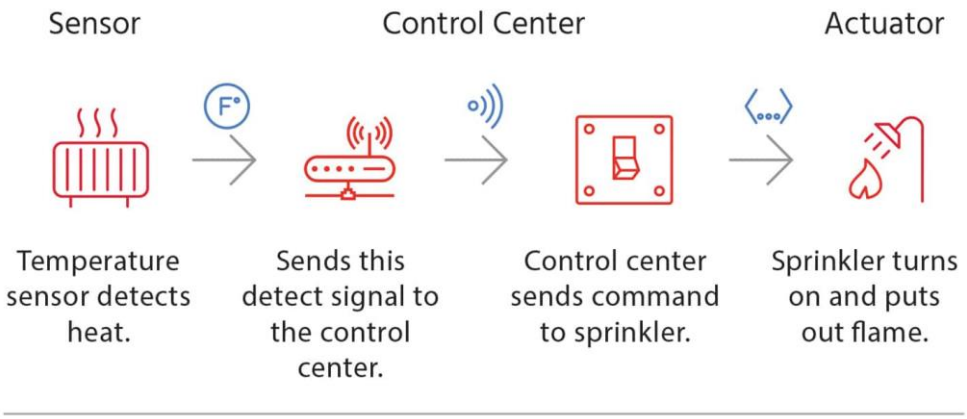
Capacitive Touchpad shield

- It has a touchpad interface that allows to integrate the Arduino board with the touch shield.
- The Capacitive touchpad shield consists of 12 sensitive touch buttons, which includes 3 electrode connections and 9 capacitive touch pads.
- The capacitive shield is shown below:



- The board can work with the logic level of 3.3V or 5V.
- We can establish a connection to the Arduino project by touching the shield.

3.12. Integration of Sensors and Actuators with Arduino



Sensor to **Actuator** Flow

The Internet of Things is a major contributing factor of the new Data Economy. The value of an IoT system goes beyond the original intended use case, for instance in automation. This is because further value lies in the intelligence that an IoT system creates. Sensors are the source of IoT data. Furthermore, sensors and actuators in IoT can work together to enable automation at industrial scale. Finally, analysis of the data that these sensors and actuators produce can provide valuable business insights over time.

Driven by new innovations in materials and nanotechnology, sensor technology is developing at a never before seen pace, with a result of increased accuracy, decreased size and cost, and the ability to measure or detect things that weren't previously possible. In fact, sensing technology is developing so rapidly and becoming so advanced that we will see a trillion new sensors deployed annually within a few years.

Sensors

A better term for a sensor is a transducer. A transducer is any physical device that converts one form of energy into another. So, in the case of a sensor, the transducer converts some physical phenomenon into an electrical impulse that determines the reading. A microphone is a sensor that takes vibrational energy (sound waves), and converts it to electrical energy in a useful way for other components in the system to correlate back to the original sound.

Actuators

Another type of transducer that you will encounter in many IoT systems is an actuator. In simple terms, an actuator operates in the reverse direction of a sensor. It takes an electrical input and turns it into physical action. For instance, an electric motor, a hydraulic system, and a pneumatic system are all different types of actuators.

Controller

In a typical IoT system, a sensor may collect information and route to a control center. There, previously defined logic dictates the decision. As a result, a corresponding command controls an actuator in response to that sensed input. Thus, sensors and actuators in IoT work together

from opposite ends. Later, we will discuss where the control center resides in the greater IoT system.

Integrating sensors and actuators with an Arduino microcontroller is a fundamental aspect of creating interactive and automated projects. Arduino is a versatile platform for developing these projects because it provides a simple and easy-to-learn programming environment along with a wide range of compatible sensors and actuators. Here's a step-by-step guide on how to integrate sensors and actuators with Arduino:

1. Gather the Necessary Components:

Arduino board (e.g., Arduino Uno, Arduino Nano, etc.)
Sensors (e.g., temperature sensor, light sensor, motion sensor, etc.)
Actuators (e.g., LEDs, motors, servos, relays, etc.)
Breadboard and jumper wires
Power source (e.g., USB cable or external power supply)

2. Install the Arduino IDE:

Download and install the Arduino Integrated Development Environment (IDE) on your computer from the official Arduino website (https://www.arduino.cc/en/software).

3. Connect the Hardware:

Connect the sensor and actuator components to the Arduino using jumper wires.
Make sure to connect the sensor's output pin to a digital or analog input pin on the Arduino and the actuator to a digital output pin.

4. Write Arduino Sketch (Code):

Open the Arduino IDE and create a new sketch.
Write the code that reads data from the sensor(s) and controls the actuator(s) based on that data.
Use the appropriate libraries for your sensors and actuators if needed.

5. Upload the Code to the Arduino:

Connect the Arduino to your computer using a USB cable.
Select the correct Arduino board and port in the Arduino IDE.
Click the "Upload" button to upload your code to the Arduino.

6. Monitor and Debug:

Open the Arduino Serial Monitor (Tools -> Serial Monitor) to view sensor readings and debug your code.
Adjust the code as necessary to achieve the desired behavior.

7. Power Considerations:

Ensure that your power supply can handle the power requirements of your sensors and actuators. Some components may require an external power supply.

8. Additional Components and Circuitry:
Depending on your project, you may need additional components such as resistors, capacitors, and transistors to interface with certain sensors or drive high-power actuators.

9. Expand and Customize:
Continue to add more sensors and actuators to create complex projects.
Explore third-party libraries and online resources for additional functionality and project ideas.

10. Enclosure and Mounting:
Depending on your project's requirements, you may need to design and build an enclosure to house the Arduino, sensors, and actuators, and mount them in the desired location.
Remember to refer to the datasheets and documentation for your specific sensors and actuators to understand their pinouts and operation. Arduino's online community and forums are also valuable resources for getting help and finding project inspiration.

UNIT IV

IOT COMMUNICATION AND OPEN PLATFORMS

- IoT Communication Models and APIs
- IoT Communication Protocols
- Bluetooth
- WiFi
- ZigBee
- GPS
- GSM modules
- Open Platform (like Raspberry Pi)
- Architecture
- Programming
- Interfacing
- Accessing GPIO Pins
- Sending and Receiving Signals Using GPIO Pins
- Connecting to the Cloud.

4.1. Internet of Things: Communication Models and APIs

"*The Internet of Things is not a concept; it is a network, the true technology-enabled Network of all networks.*" — Edewede Oriwoh

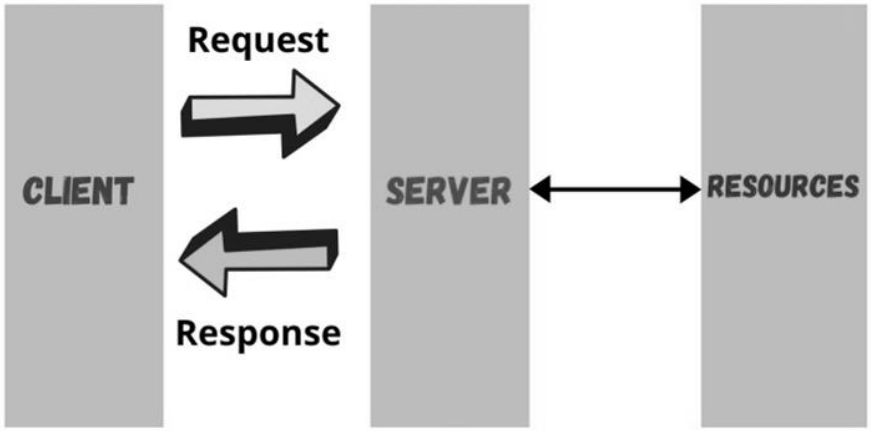
"Internet of Things is the network of physical objects or 'things' embedded with electronics, software, sensors and connectivity to enable it to achieve greater value and service by exchanging data with the manufacturer, operator and/or other connected devices. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure."

Communication Models in IoT

Understanding how the different IoT devices communicate with each other is important and useful cause IoT devices are found almost everywhere now. Communication models are the way to understand it. IoTs enable people and things to be connected anytime, in any space, with anything and anyone, through any network and service.

Now let us understand the different communication models in IoT.

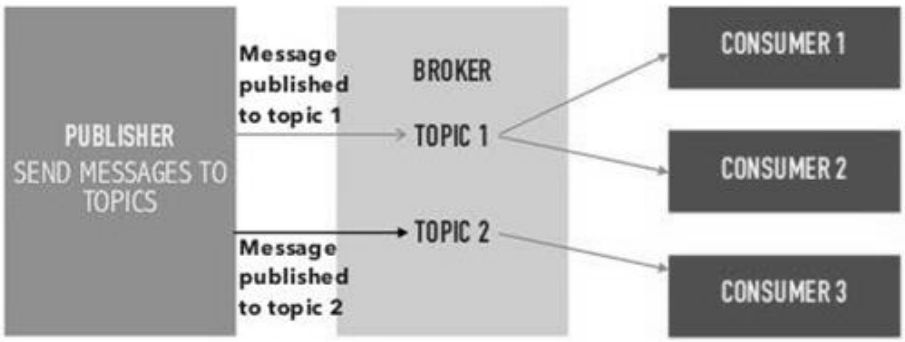
Request & Response Model



The communication takes place between a client and a server.

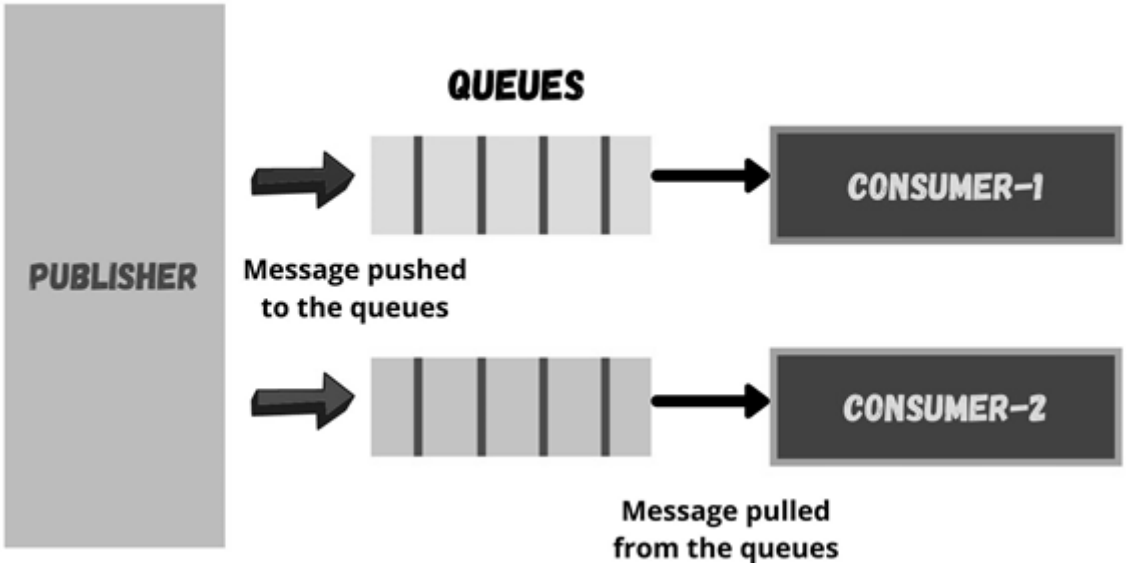
Whenever required, the client will request information from the server. This request is usually in the encoded format.So in this model, basically a client sends requests to the server and the server responds to the requests. That is why it is called as Request-Response model.After receiving the request from the client, the server decides how to respond, fetches the data from the database and its resource representation, prepares a response and ultimately sends the response to the client.

Publish-Subscribe Model(Pub-Sub)



In this model, you will find three main entities:- Publisher, Broker and Consumer Let us see the roles of each of these 3 entities.Publishers, send the data to the topics that are managed by the broker. They are the source of data.The Man in the Middle, the Broker, has theresponsibility to accept the data sent by the publisher and deliver that data to the consumers.What is the task of the Consumers? Consumers will subscribe to the broker-managed topics.Once the data is published on a topic, the broker sends this message to all consumers who have subscribed to the specific topic. It works a bit like YouTube. When you subscribe to a channel and tap the Bell icon, you'll get notifications if the YouTube channel posts a video.

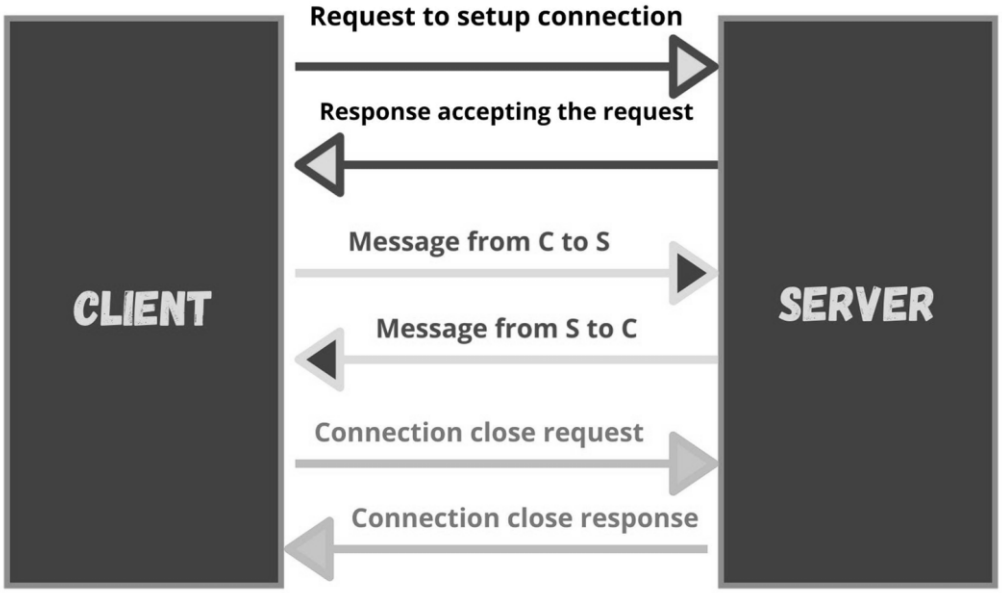
Push-Pull Model



Here too, we have 3 entities:- Publisher, Queues and Consumers.

Push-Pull is a communication model where data producers push data into queues and consumers pull data out of queues.What are Queues? They are used to separate out single producer-consumer communication.At times, there might be some mismatch in the push-pull rates. Queues act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumer pull data. So they work as a buffer and flow control mechanisms whenever there is any mismatch in the push-pull rates.

Exclusive Pair Model

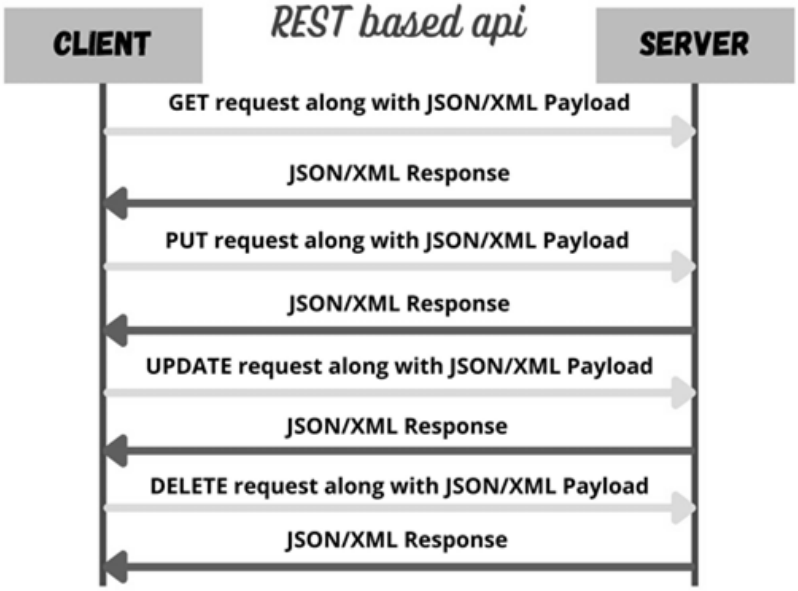


It's a bi-directional, fully duplex communication model in which a dedicated communication link is set between the client and the server.The connection remains open until the client sends a request to close the connection.The client and server can send messages to one another after configuring the connection. As soon as the connection is terminated, no exchange of messages would take place between the client and the server.

IoT Communication APIs

An API is an interface used by programs to access an application.It enables a program to send commands to another program and receive replies from the app.IoT APIs are the interface points between an IoT device and the Internet and/or other network components.
Here we will talk about the REST-based API and the Websocket based API.

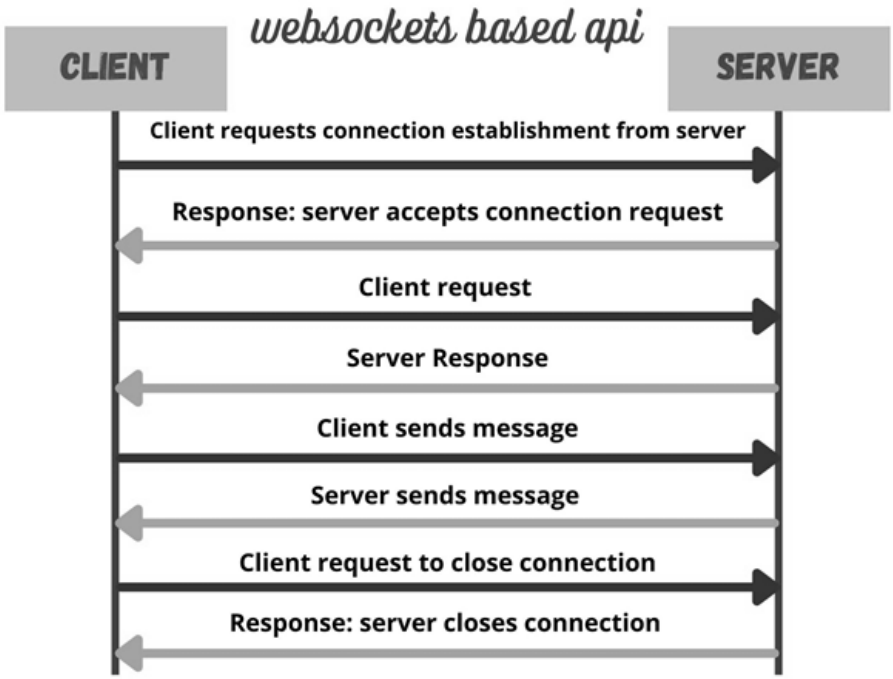
REST-based APIs(Representational state transfer)



Representational state transfer (REST) is a set of architectural principles by which you can design Web services the Web APIs that focus on the system's resources and how resource states are addressed and transferred.

URIs(example:- example.com/api/tasks) are used to depict resources in the RESTful web service.
Client tries to access these resources via URIs using commands like GET, PUT, POST, DELETE and so on that are defined by HTTP.
In response, the server responds with a JSON object or XML file.
The REST APIs follow the request-response model.
The rest architectural constraints are as follows:

Client-server
Let me explain it to you by giving a suitable example. The client should not be concerned with the storage of data which is a concern of the server, similarly, the server should not be concerned about the user interface, which is the concern of the client. Separation makes it possible for the client and server to be developed and updated independently.



Stateless

The status of the session remains entirely on the client.

Cache-able

This property defines whether the response to any request can be cached or not. If a response can be cached, then a client cache is granted the right to reuse that response data for subsequent matching requests.

Layered system

A layered system defines the boundaries of the components within each specific layer. For example, A client is unable to tell whether it is connected to the end server or an intermediate node. As simple as that!

Uniform interface

This specifies that the technique of communication between a client and a server must be uniform throughout the communication period.

Code on Demand

Servers may provide executable code or scripts for execution by clients in their context.

Websocket based APIs

Websocket APIs enable bi-directional and duplex communication between customers and servers.

Unlike REST, There is no need to set up a connection every now and then to send messages between a client and a server.

It works on the principle of the exclusive pair model. Can you recall it? Yes. Once a connection is set up, there is a constant exchange of messages between the client and the server. All we need is to establish a dedicated connection to start the process. the communication goes on unless the connection is terminated.

It is a stateful type.

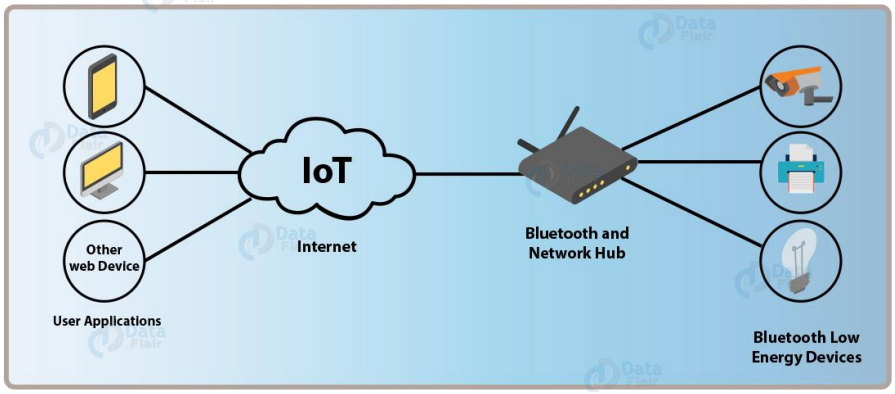
Due to onetime dedicated connection setup, there is less overhead, lower traffic and less latency and high throughput.

4.2. IoT Communication Protocols

Several Communication Protocols and Technology used in the internet of Things. Some of the major IoT technology and protocol (IoT Communication Protocols) are Bluetooth, Wifi, Radio Protocols, LTE-A, and WiFi-Direct.

4.2.1. Bluetooth

An important short-range IoT communications Protocols / Technology. Bluetooth, which has become very important in computing and many consumer product markets. It is expected to be key for wearable products in particular, again connecting to the IoT albeit probably via a smartphone in many cases.

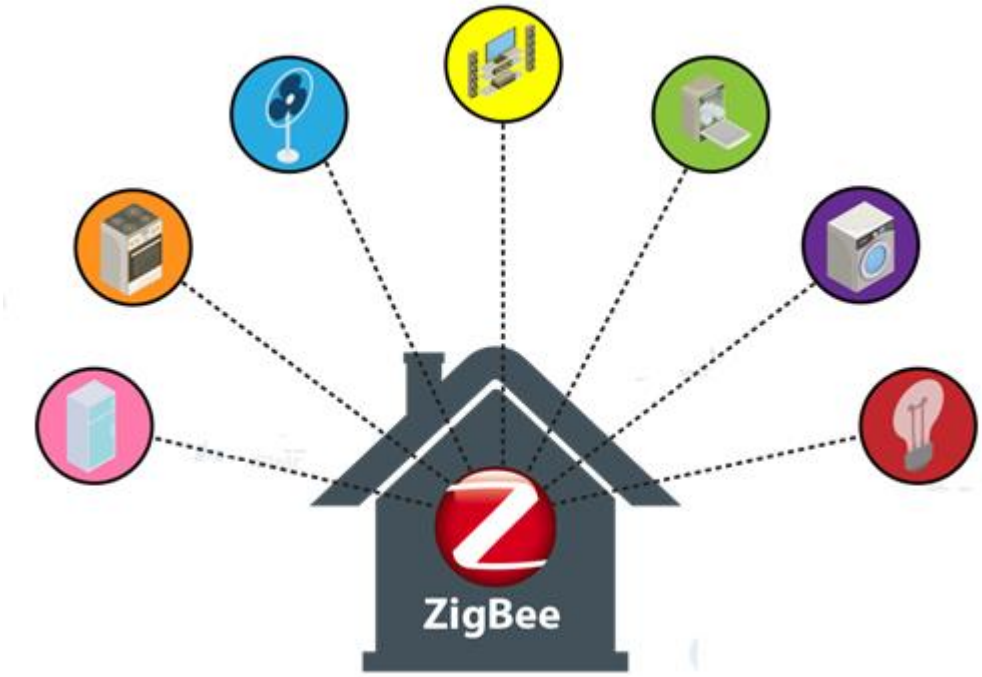


Iot Technology – Bluetooth

4.2.2. Zigbee

ZigBee is similar to Bluetooth and is majorly used in industrial settings. It has some significant advantages in complex systems offering low-power operation, high security, robustness and high and is well positioned to take advantage of wireless control and sensor networks in **IoT applications**.

The latest version of ZigBee is the recently launched 3.0, which is essentially the unification of the various ZigBee wireless standards into a single standard.



Iot Technology – ZigBee

4.2.3. Z-Wave

Z-Wave is a low-power RF communications IoT technology that primarily design for home automation for products such as lamp controllers and sensors among many other devices.

A Z-Wave uses a simpler protocol than some others, which can enable faster and simpler development, but the only maker of chips is Sigma Designs compared to multiple sources for other wireless technologies such as ZigBee and others.

Iot Technology – Z-Wave**d. Wi-Fi**

WiFi connectivity is one of the most popular IoT communication protocol, often an obvious choice for many developers, especially given the availability of WiFi within the home environment within LANs.

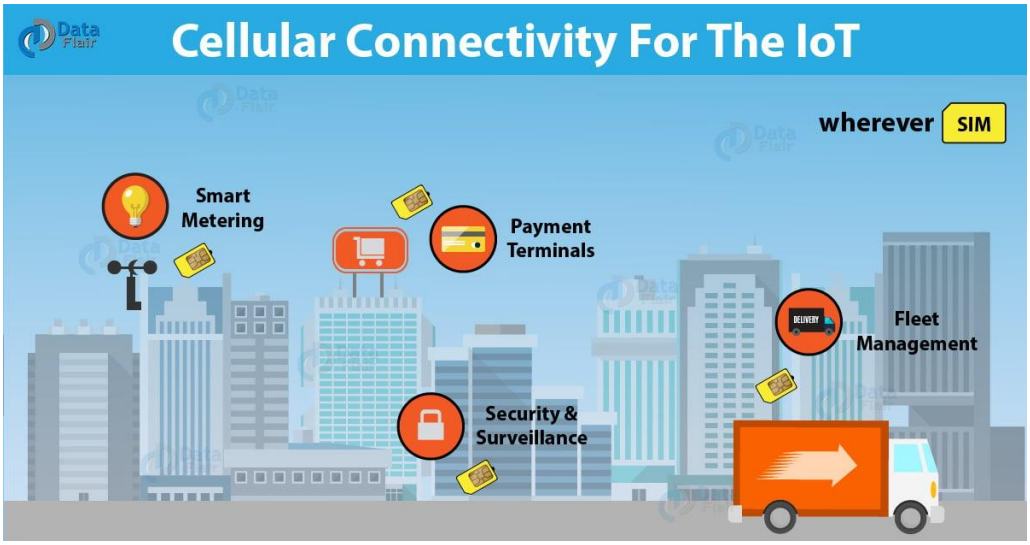
There is a wide existing infrastructure as well as offering fast data transfer and the ability to handle high quantities of data.

Currently, the most common WiFi standard used in homes and many businesses is 802.11n, which offers range of hundreds of megabit per second, which is fine for file transfers but may be too power-consuming for many IoT applications.

4.2.4. Cellular

Any IoT application that requires operation over longer distances can take advantage of GSM/3G/4G cellular communication capabilities. While cellular is clearly capable of sending high quantities of data, especially for 4G, the cost and also power consumption will be too high for many applications.

But it can be ideal for sensor-based low-bandwidth-data projects that will send very low amounts of data over the Internet.



IoT Communication Protocols – Cellular

4.2.5.NFC

NFC (Near Field Communication) is an IoT technology. It enables simple and safe communications between electronic devices, and specifically for smartphones, allowing consumers to perform transactions in which one does not have to be physically present.

It helps the user to access digital content and connect electronic devices. Essentially it extends the capability of contactless card technology and enables devices to share information at a distance that is less than 4cm.

IoT Communication Protocols – NFC

4.2.6. LoRaWAN

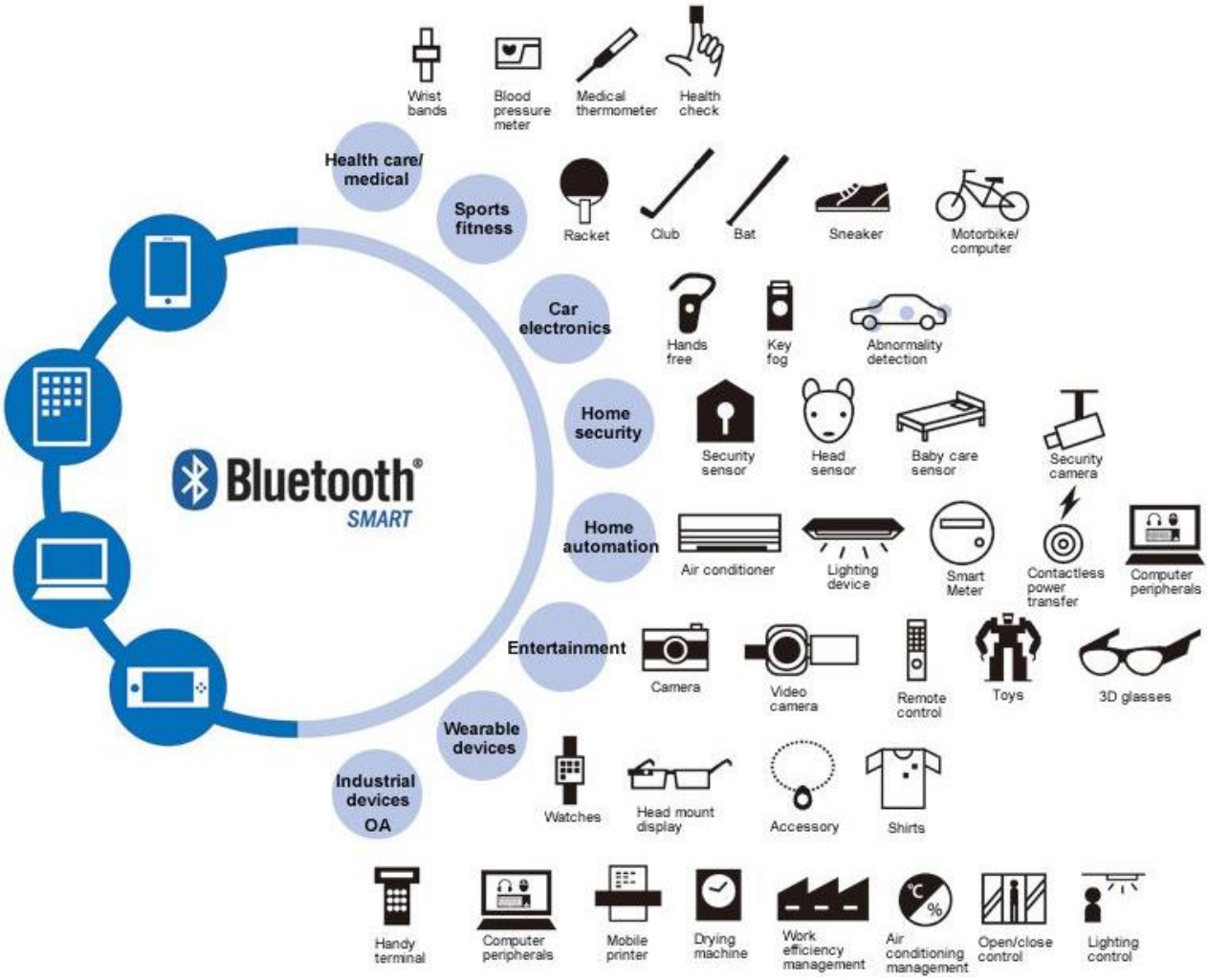
LoRaWAN is one of popular IoT Technology, targets wide-area network (WAN) applications. The LoRaWAN design to provide low-power WANs with features specifically needed to support low-cost mobile secure communication in IoT, smart city, and industrial applications.

Specifically meets requirements for low-power consumption and supports large networks with millions and millions of devices, data rates range from 0.3 kbps to 50 kbps.

Iot Technology – LoRaWAN

4.3. Bluetooth

To understand the architecture of Bluetooth first lets understand what actually Bluetooth is. Bluetooth is a radio-wave technology that is mainly designed to enable wireless communications over short distances. The frequency of these waves ranges between 2.400 and 2.485 GHz, which can extend a maximum of 164 feet between two devices. Every Bluetooth device has a Transmitter and a Receiver. The power of the device transmitter governs the range over which a Bluetooth device can operate in other words transmitter decides the range of communication.



These days Bluetooth is the common technology present in most of the products. It is used in many fields such as the health sector, sports and fitness, electronics, home automation, and security, etc.

Some of the applications of Bluetooth:

–> Wireless control and communication between a mobile phone and a handsfreeheadset.

->Wireless communication between a smartphone and a smart lock for unlocking doors.

–> For low bandwidth applications where USB higher bandwidth is not required and cable-free connection desired.

–> For example, Smart watches, Smart Lights.

How does it work?

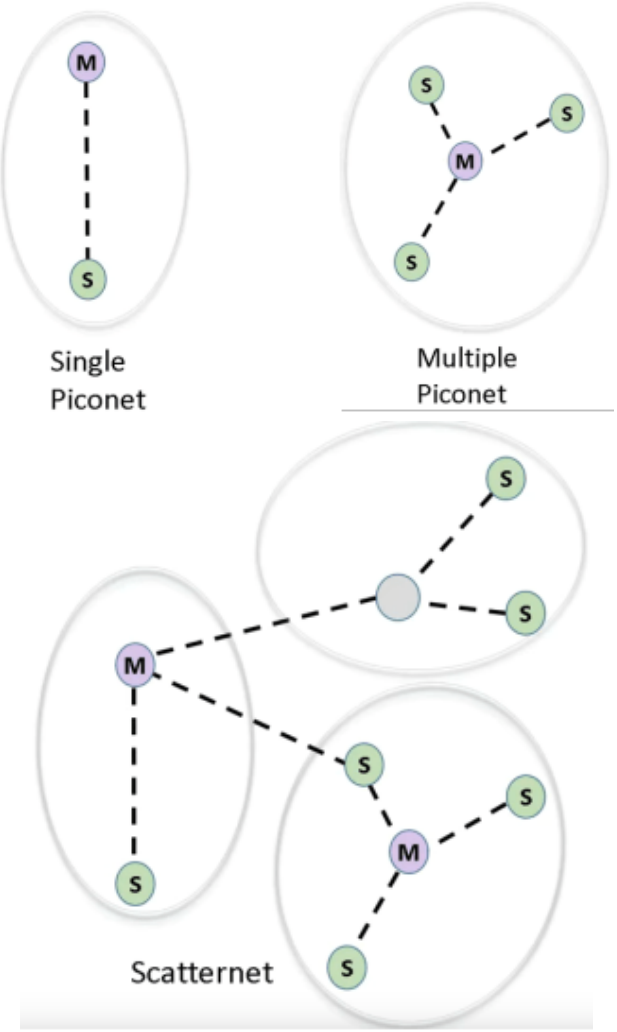
4.3.1. Bluetooth Architecture.

4.3.1.1. Piconet

The Bluetooth network is called a piconet. If it contains one master and one slave then its called a single piconet. Similarly, if it contains one master and multiple slaves are called multiple piconets.

The Master is the one that initiates the communication with other devices and it dictates when a slave device may transmit.
Direct Slave to Slave communication is not possible.

Maximum 7 active slaves can be present in multiple piconets, in other words, only 8 maximum devices including the master can communicate at any one time in a piconet.



4.3.1.2. Scatternet

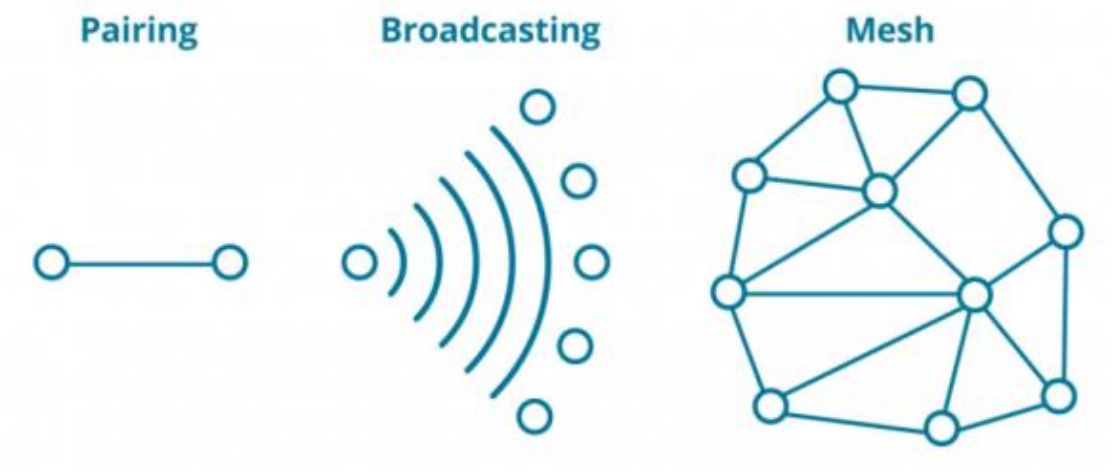
Its a Combination of multiple piconets.
Here Master of one piconet can be a slave in another piconet. This node can receive a message from a master in one piconet and deliver the message to its slave into the other piconet.
Therefore, this type of node is referred to as a bridge node. Above all, a station cannot be master in two piconets.Read More

The Architecture of a Bluetooth IoT Application

Firstly, we must first dive into the Bluetooth "stack" to understand why recent shifts in Bluetooth standards are significant for IoT applications The evolution of Bluetooth from a replacement for RS-232 data cables to a powerful and massive IoT connectivity solution is a story of adding new layers to the stack. The newest Bluetooth specification for IoT—

Bluetooth mesh—must be engineered upon either the BLE 4.xx or 5.xx stack—an extension of the Bluetooth Core ("classic") specification. The emerging Bluetooth mesh stack, therefore, comprises three-stack layers: Core, then BLE, and mesh on top.

Bluetooth Topologies: Pair, Broadcast, Mesh



- **P**air: Bluetooth as a means of pairing two devices
- For Example, a computer paired with a wireless mouse
- Broadcasting: Bluetooth as a means of having one device broadcast information to many devices or vice versa
- For Example: Playing music on smart speakers and simultaneously casting photos to a projector—both using a single iPhone
- Mesh: Bluetooth as a way of connecting many devices to many others as if in a spider's web
- For Example: Connecting 1,278 overhead lights in a warehouse to each other to dim and brighten lights automatically based upon activity and personal preferences.

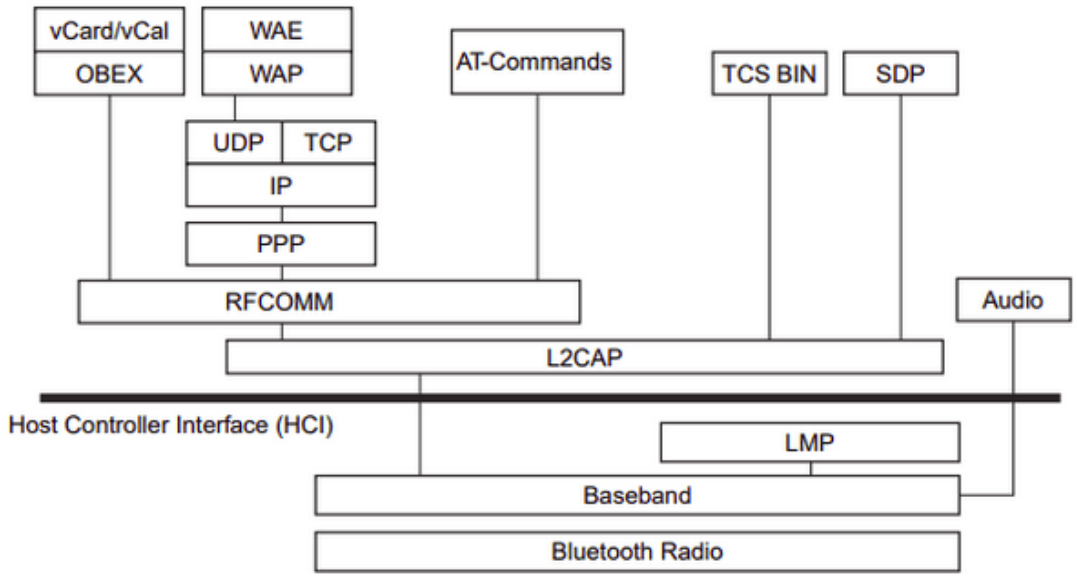
4.3.2. Bluetooth Protocol Types

The main function of the Bluetooth is a Bluetooth protocol stack in architecture of Bluetooth. In other words, It defines and provides different types of layers and functionalities. Bluetooth can run the different applications over different protocol stacks, but, each one of these protocol stacks uses the same Bluetooth link and physical layers.

The below diagram shows a complete Bluetooth protocol stack. It shows the relationship between the protocols that use the services of other protocols when there is a payload to be transferred in the air. Anyhow, the protocols have many other relationships between the other protocols – for example, some protocols (L2CAP, TCS Binary) use the LMP to control the link manager.

The complete protocol stack architecture of Bluetooth is made up of both Bluetooth specific protocols like object exchange protocols (OBEX) and user datagram protocol (UDP).
The main principle is to minimize the reuse of current protocols for different purposes at higher layers as if re-inventing circle once again. The protocol re-use is helpful for the legacy

applications to work with the Bluetooth technology to measure the smooth operations and interoperability of applications. Hence, many applications are being developed to take immediate advantage of the software and hardware.

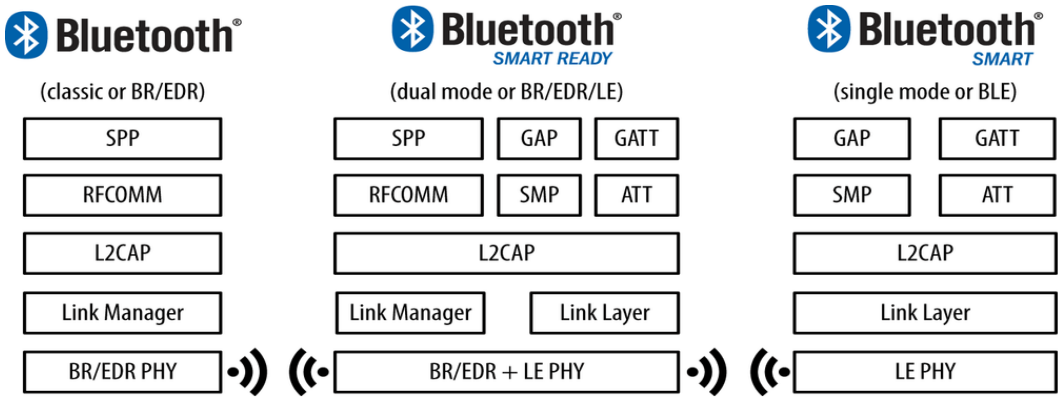


Protocol layers	**Protocol in the stacks**
Bluetooth Core Protocol	Baseband, LMP, L2CAP, SDP
Cable Replacement Protocol	RFCOMM
Telephony Control Protocol	TCS Binary, AT- commands
Adopted Protocols	PPP, OBEX, UDP/TCP/IP, WAP, Vcard, Vcall, IrMC, WAE

Layers of Architecture of Bluetooth :

Host Controller Interface	A command interface for the controller and for the link manager, which allows access to the hardware status and control registers.
Logical Link Control and Adaptation Protocol	It is also known as the heart of the Bluetooth protocol stack. It allows the communication between the upper and lower layers of the Bluetooth protocol stack.
Radio (RF) layer	It performs modulation/demodulation of the data into RF signals.
Baseband Link layer	In short, it performs the connection establishment within a piconet.
SDP layer	It is short for Service Discovery Protocol. It allows for discovering the services available on another Bluetooth enabled device.
WAP	It is short for Wireless Access Protocol. It is used for internet access.
TCS	It is short for Telephony Control Protocol. It provides a telephony

	service.
Application layer	In short , it enables the user to interact with the application.



Advantages of Bluetooth Protocols

- Bluetooth offers economic wireless solutions (both data & voice) but, for short distances
- On the other hand, Mobile and stationary environment use Bluetooth protocol.
- There is no setup file to install the Bluetooth, in other words, it is an inbuilt device.
- Above all, They are up-gradable

Characteristics of Bluetooth Protocols

- In short, Up to eight devices including the master can communicate in the Piconet by using Bluetooth.
- Bluetooth signals are Omnidirectional as a result devices don't point at each other.
- Governments regulated worldwide because it is possible to utilize the same standard.
- In short , Signals can transmit through walls and briefcases.

Bluetooth Security

Firstly, the security of any wireless technology is very important. With hackers gaining access to an ever-increasing number of systems, as a result,Bluetooth security is increasingly important.

Bluetooth security must also address more specific Bluetooth related attacks that target known vulnerabilities in Bluetooth implementations and specifications. In other words, these may include attacks against improperly secured Bluetooth implementations which can provide attackers with unauthorized access.

Many users may not believe there is an issue with Bluetooth security, but hackers may be able to gain access to information from phone lists to more sensitive information that others may hold on Bluetooth enabled phones and other devices.

Firstly, there are three basic means of providing Bluetooth security:

- Authentication: In this process, that is to say,the identity of the communicating devices is verified. But ,user authentication is not part of the main Bluetooth security elements of the specification.
- Confidentiality: In Short, this process prevents the information from being eavesdropped by ensuring that only authorized devices can access and view the data.
- Authorization: This process prevents access by ensuring that a device is authorized to use a service before enabling it to do so.

Common Bluetooth security issues

- **Bluejacking:** In short, Sending spam messages to discoverable Bluetooth enabled devices. However, this form of hacking is harmless. In short, the best way to defend against it is to keep Bluetooth settings to invisible or non-discoverable.

- **Bluesnarfing:** It is more serious than bluejacking because it can reveal private information on a smartphone and is capable of happening even when invisible/non-discoverable mode is enabled.

- **Bluebugging:** In short, it's capable of accessing all the information such as photos, apps, contacts, etc. It's more dangerous than bluejacking and bluesnarfing.

In Conclusion, Some prevention tips for Bluetooth hacks are to set invisible mode. Therefore, makes it more difficult for hackers to gain access to your data. Lastly, stay awake from the open Wi-Fi networks in busy or untrusted locations, so that you can minimize the risk of falling victim to hackers.

4.4. Wi-Fi

A ubiquitous Internet of Things (IoT) depends upon wireless connectivity, but there are many options for wireless and not every device is IP addressable – a requisite feature for IoT.

What's more, RF design is inherently difficult. Few companies are equipped with the appropriate skills to implement RF and antenna design, and even when done, keeping that design up to date with the latest standards and getting it through FCC compliance is time consuming.

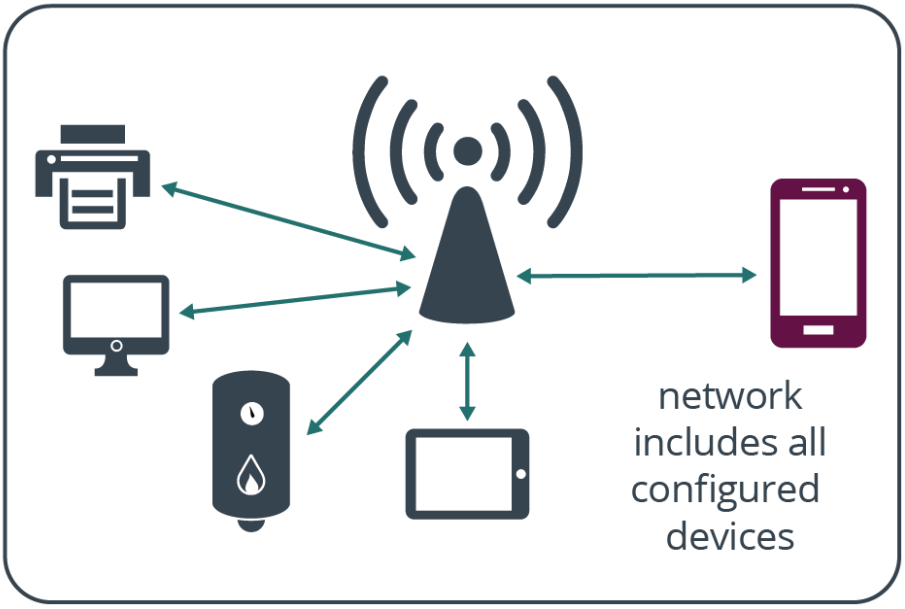


Figure: The wide range of IoT applications that use Wi-Fi modules shows the flexibility and scalability of the interface, as well as its ubiquity. (Source: Tektronix.)

For designers, there is a very solid and proven option for wireless connectivity and the IoT. That option is Wi-Fi and the use of modules. This feature will explain why, offer some design solutions, discuss soon-to-arrive upgrades to IEEE 802.11 protocols, and show how to bridge ZigBee to Wi-Fi for native IP addressability.

There are many wireless interface options, Bluetooth Low Energy (BLE), ZigBee, Z-Wave, Wi-Fi and RFID, each with their own unique balance of power, range, data rates, mesh networking, interference immunity, and ease of use. However, some interfaces are not yet native-IP enabled, so cannot be addressed directly or exchange data with other devices and servers over the Internet. These then require a separate gateway, adding expense and complexity to the final solution.

This is where Wi-Fi stands out: it is based on the IEEE 802.11 standards with native IP addressability, is ubiquitous, well understood, and can scale well in terms of data rates to optimize for power consumption. According to the Wi-Fi Alliance, there are more than 6.8 billion installed Wi-Fi-capable devices, so the odds are pretty high that there is a local Wi-Fi access point available . Note too that 802.11 standards are also IPv6 compliant, so there's almost no limit to the number of unique addresses.

While Wi-Fi is widespread and well understood, it is still a wireless interface, with all the vagaries of design that RF entails. Understanding next steps is critical.

Easing wireless development

After selecting Wi-Fi for an IoT application, the designer often faces the daunting challenge of building a custom RF implementation, which requires time, money and expertise. Design requirements for developing a wireless device include, at a minimum, direct RFIC integration and the ability to specify components such as filters, amplifiers, clocks, capacitors, inductors, crystal oscillators, and the antennas that need to be on the board, as well as their placement. There also needs to be network-matching circuitry to ensure that the radio and antenna are well matched to avoid signal loss. Other knowledge areas include system layout, software stack development, device security, connection reliability, signal interference and degradation, and last but surely not least, FCC certification.

How does this task get done within typical time to market parameters given that RF design expertise is not something readily available to every electronics company? An increasingly common way to add Wi-Fi capability is to use a pre-packaged module. This approach greatly simplifies the process. Modules are supplied tested, calibrated and pre-certified to the required standards by the module vendor, and therefore can provide companies with a fast, easy route to market with what is, essentially, a plug-and-play solution, reducing the need for software development. What is more, manufacturers who design and build the Wi-Fi modules can be your RF consultants during the design integration stage.

The Wi-Fi module generally contains two main parts: a Wi-Fi chip and an application host processor. The Wi-Fi subsystem includes an 802.11 radio physical layer (PHY), baseband, media access control (MAC), and perhaps a crypto engine for fast, secure Internet connection. The application host processor has internal or external flash, ROM, and RAM. The module generally also comes with I/Os for timers, serial communication interfaces, analog comparators, analog-to-digital converter (ADC), digital-to-analog converter (DAC), crystal oscillators, and a debug interface.

The power management subsystem includes integrated DC-DC converters supporting a wide range of supply voltages. It enables low-power consumption modes, such as hibernate with real-time clock (RTC) mode. A module may offer an integrated antenna or provide an RF connector for an external antenna. The software package included with a Wi-Fi module usually includes a device driver, an integrated 802.11 security layer, and a management and monitoring utility.

In designing a Wi-Fi IoT solution, the starting point is an understanding that IEEE 802.11 represents a family of standards that until just recently operated only in the 2.4 GHz (IEEE 802.11b/g/n) and 5 GHz (IEEE 802.11a/n/ac) unlicensed bands. There are three key factors to consider when evaluating these protocols: data rate, range, and power requirements. When you compare the different Wi-Fi protocols, 802.11b/g has the advantage in compatibility with installed devices and power requirements while 802.11n and 802.11ac have the advantage of higher data throughput for multimedia applications such as video streaming (see Table).

Protocol	Frequency	Signal	Maximum data rate
Legacy 802.11	2.4 GHz	FHSS or DSSS	2 Mbps
802.11a	5 GHz	OFDM	54 Mbps
802.11b	2.4 GHz	HR-DSSS	11 Mbps
802.11g	2.4 GHz	OFDM	54 Mbps
802.11n	2.4 or 5 GHz	OFDM	600 Mbps (theoretical)
802.11ac	5 GHz	256-QAM	1.3 Gbps

Table: A summary of different Wi-Fi protocols and data rates shows the progression of IEEE 802.11 from its early days of 2 Mbps to 1.3 Gbps today. (Source: Intel Corp.)

When designing for IoT applications, however, a higher data rate protocol is not always preferable. Even though 802.11ac can crank it up to a maximum of 1.3 Gbps, most embedded applications (e.g., machine to machine [M2M] data and control devices) are power constrained and can get by with a much lower data rate.

One more good reason to choose Wi-Fi is that within the next year or so technology upgrades will make it much more capable. For example, the Wi-Fi Alliance recently announced the Wi-Fi HaLow (pronounced "halo") designation for products incorporating IEEE 802.11ah technology. HaLow extends Wi-Fi into the 900 MHz band and offers improved range – nearly twice that of today's Wi-Fi – with scalable data rates from 150 Kbps to 2.1 Mbps; IEEE 802.11ah also promises to provide low power consumption features and its use cases are primarily for wireless sensor networks, a classic IoT application.

In February, at the 2016 International Solid-State Circuits Conference (ISSCC) the Holst Centre of the Netherlands, in collaboration with Belgium's IMEC Research Institute, showed a HaLow transmitter running Wi-Fi over sub-GHz bands. The team cited a maximum power consumption of 7.1 mW when delivering 0 dBm output power and operating from a 1 V supply. This represents a 10x power reduction compared to state-of-the-art 802.11 OFDM transceivers.

Also under development is IEEE 802.11ai, which will provide a fast initial link setup (achieving a secure link setup in less than 100 ms) and IEEE 802.11aq, being developed to provide a cellular-like automatic network-discovery experience.

Modules and dev kits

RF engineering can be a tricky business, especially for neophytes in the discipline. Recognizing this, RF silicon suppliers are providing modules, dev kits and reference designs that make adding wireless connectivity to a product much easier.

Microchip, for example, supplies an IoT development kit (DM990001) using a module driven by a 32-bit controller. It employs two Microchip components, the MRF24WG0MA/B pre-certified Wi-Fi module, which supports both 802.11b and 802.11g, and the PIC32MX695F512H microcontroller with 128 Kbytes of RAM and 512 Kbits of flash. The PIC MCU features an 80 MHz, 105-DMIPS 32-bit core, a USB 2.0 On-The-Go (OTG) peripheral with integrated PHY, a 10/100 Ethernet MAC, and four dedicated direct memory access (DMA) channels for USB OTG and Ethernet.

The starter kit is powered by Amazon Web Services (AWS), a managed cloud platform that lets connected devices securely interact with cloud applications. With AWS IoT applications can keep track of and communicate with all networked devices, all the time, even when they aren't connected.

Amazon has put together Quickstart reference deployments for its AWS IoT, which is designed to show off some of the capabilities of the platform. AWS IoT can be especially helpful if you plan to use Amazon as your hosting provider. It supports HTTP, WebSockets for web browsers and web servers, and MQ Telemetry Transport (MQTT), a lightweight communication protocol for small sensors and mobile devices specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements.

Texas instruments' CC3100MOD Wi-Fi module is a complete platform solution including various tools and software, sample applications, user and programming guides, and reference designs. It consists mainly of the company's CC3100R11MRGC Wi-Fi Network Processor and power management subsystems. Part of the company's SimpleLink Wi-Fi family, CC3100MOD integrates all protocols for Wi-Fi and Internet, reducing host MCU software requirements. It also includes all required clocks, SPI flash, RF filter, crystal and passives.

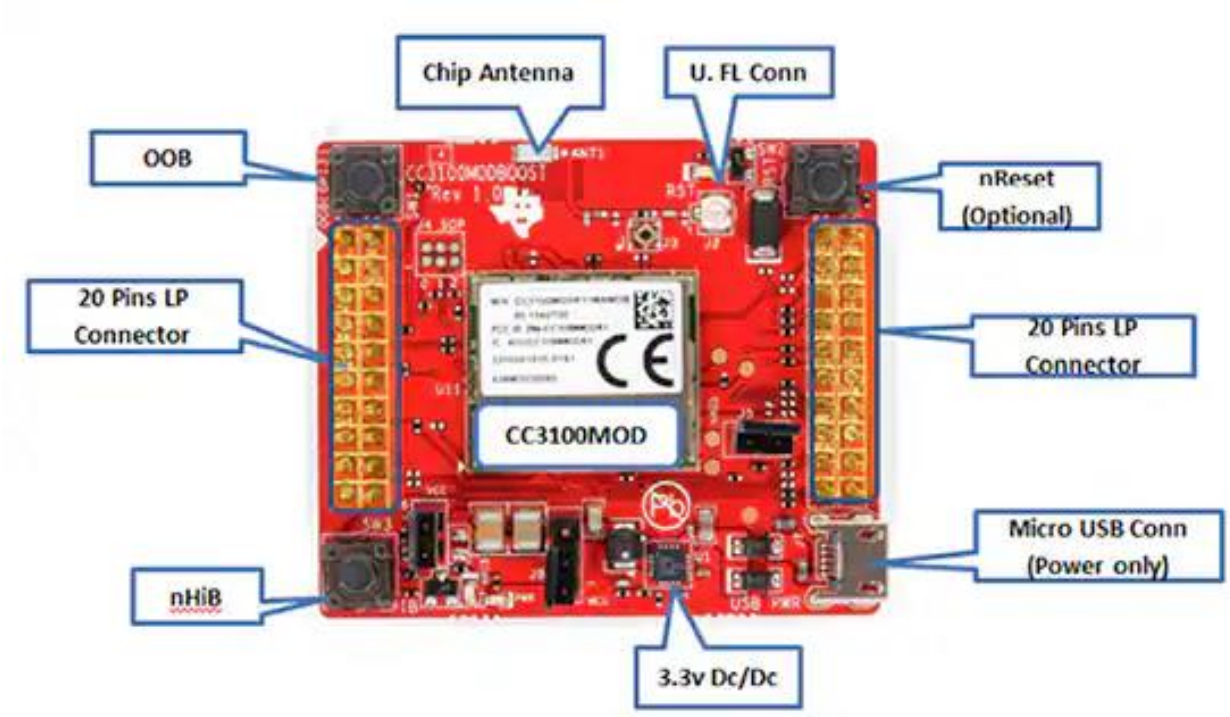
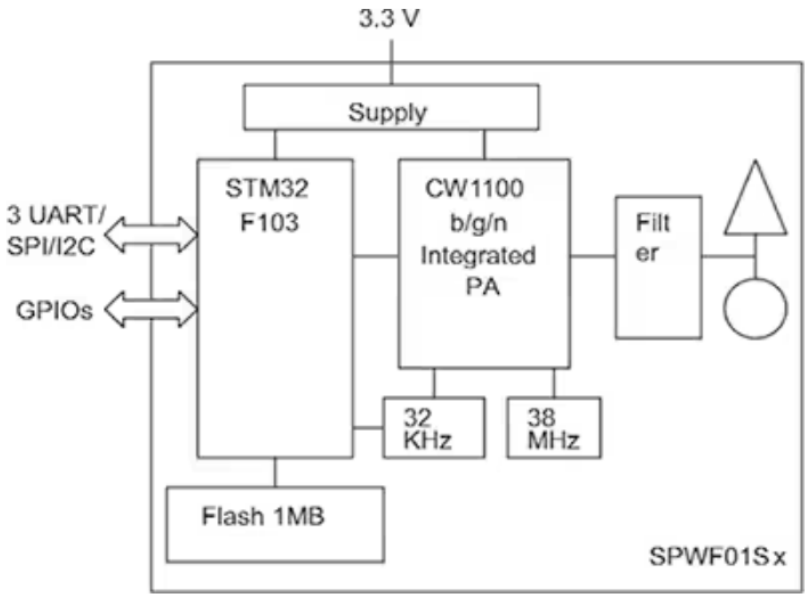


Figure: TI's CC3100MOD is designed to simplify the implementation of Internet connectivity. It integrates all protocols for Wi-Fi and the Internet, which greatly reduces host MCU software requirements.

The CC3100MOD Wi-Fi Network Processor subsystem incorporates an 802.11 b/g/n radio, baseband, and MAC and has a 256-bit encryption engine for secure Internet connection. It also contains a dedicated ARM MCU that completely off-loads the host MCU. The power management subsystem includes an integrated DC-DC converter with support for a wide range of supply voltages. It enables low-power consumption modes such as hibernate with RTC mode, which requires only about 7 μA of current.

On the software side the module includes embedded TCP/IP and TLS/SSL security stacks, HTTP server, and multiple Internet protocols. The device driver keeps the host memory footprint requirement to less than 7 Kbytes for code and 700 bytes of RAM for data for a TCP client application. The CC3100MOD IoT module solution is also FCC, IC, CE, and Wi-Fi certified.

The SPWF01S Wi-Fi module from STMicroelectronics is a plug-and-play 802.11b/g/n solution It has a serial interface on one side that allows communications with MCUs like the supplier's STM32 and STM8. On the other side is the Wi-Fi transceiver. The module also incorporates timing clocks and voltage regulators. It is available either configured with an embedded micro 2.45 GHz ISM band antenna (SPWF01SA), or with a U.FL coax connector for external antenna connection (SPWF01SC), so no RF knowledge is required for integration.



Block diagram of the ST SPWF01S serial-to-Wi-Fi b/g/n module. With low power consumption and a small form factor, it can be used for both fixed and mobile wireless applications.

The SPWF01S comes with CE, FCC and IC certification and meets industrial temperature-range requirements.
While we have focused on Wi-Fi, the ZigBee wireless standard is a popular choice for a wide range of applications that require low power, medium range, and flexible networking that is relatively easy to install, configure and maintain. ZigBee, however, lacks a crucial requirement for IoT applications: native IP connectivity. Digi International's XBee® Wi-Fi modules bridge this gap. To get you started, the company's XBee Wi-Fi Cloud kit (xka2B-wft-0) includes an XBee Wi-Fi (S6B) module, XBee USB development board with breadboard, a components package (resistors, relay, buttons, LEDs), access to Digi's cloud-based application and all necessary antennas, power supplies, and cables. Digi's XBee Wi-Fi modules share a common footprint with other XBee modules, which allows different XBee technologies to be drop-in replacements for each other.

IoT applications depend upon wireless connectivity, but there are numerous wireless protocol options. RF design also is inherently difficult and few companies are equipped with the appropriate skills to implement RF and antenna design. Even if the expertise is available, keeping the design up to date with the latest standards and getting it through FCC compliance is time consuming. This Wi-Fi modules and their associated kits and reference designs can dramatically simplify the implementation challenges facing designers of new IoT devices.

4.5. Zigbee

When you are designing, planning and prototyping a wireless application, the protocol you choose is a key component, with considerations for security, flexibility and the ability to deploy to multiple regions.

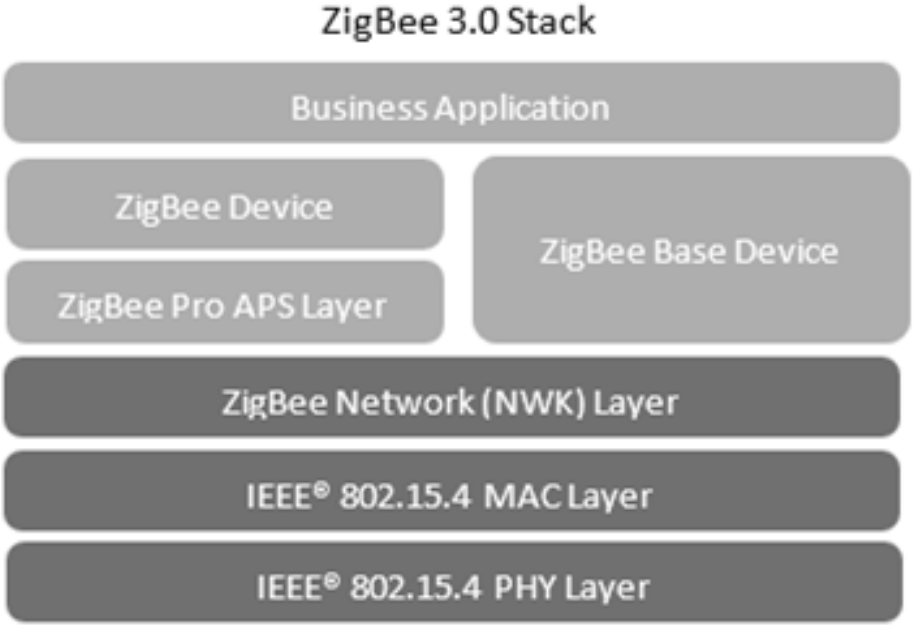
Zigbee is a wireless technology developed as an open global standard to address the unique needs of low-cost, low-power wireless IoT networks. The Zigbee standard operates on the IEEE 802.15.4 physical radio specification and operates in unlicensed bands including 2.4 GHz, 900 MHz and 868 MHz.

The 802.15.4 specification upon which the Zigbee stack operates gained ratification by the Institute of Electrical and Electronics Engineers (IEEE) in 2003. The specification is a packet-based radio protocol intended for low-cost, battery-operated devices. The protocol allows devices to communicate in a variety of network topologies and can have battery life lasting several years.

zigbee alliance The Zigbee 3.0 protocol was created and ratified by member companies of the Zigbee Alliance. Over 300 leading semiconductor manufacturers, technology firms, OEMs and service companies comprise the Zigbee Alliance membership. The Zigbee protocol was designed to provide an easy-to-use wireless data solution characterized by secure, reliable wireless network architectures.

The Zigbee Advantage

The Zigbee 3.0 protocol is designed to communicate data through noisy RF environments that are common in commercial and industrial applications. Version 3.0 builds on the existing Zigbee standard but unifies the market-specific application profiles to allow all devices to be wirelessly connected in the same network, irrespective of their market designation and function.



Additionally, a Zigbee 3.0 certification scheme ensures the interoperability of products from different manufacturers. Connecting Zigbee 3.0 networks to the IP domain opens up

monitoring and control from devices such as smartphones and tablets on a LAN or WAN, including the Internet, and brings the true Internet of Things to fruition.

Zigbee protocol features include:

- Support for multiple network topologies such as point-to-point, point-to-multipoint and mesh networks
- Low duty cycle – provides long battery life
- Low latency
- Direct Sequence Spread Spectrum (DSSS)Up to 65,000 nodes per network
- 128-bit AES encryption for secure data connections
- Collision avoidance, retries and acknowledgements

Zigbee Wireless Mesh Networking

What Is Zigbee?

Zigbee is a wireless technology developed as an open global market connectivity standard to address the unique needs of low-cost, low-power wireless IoT data networks. The Zigbee connectivity standard operates on the IEEE 802.15.4 physical board radio specification and operates in unlicensed radio bands including 2.4 GHz, 900 MHz and 868 MHz.

The 802.15.4 wireless specification upon which the Zigbee stack operates gained board ratification by the Institute of Electrical and Electronics Engineers (IEEE) in 2003. The specification is a packet-based radio board protocol intended for low-cost, battery-operated devices and products. The protocol allows devices to communicate data in a variety of network topologies and can have battery life lasting several years.

The Zigbee 3.0 Protocol

The Zigbee protocol has been created and ratified by member companies of the Zigbee Board Alliance.Over 300 market leading semiconductor manufacturers, technology firms, OEMs and service companies comprise the Zigbee Alliance membership board. The Zigbee protocol was designed to provide an easy-to-use wireless data solution characterized by secure, reliable wireless network architectures.

The Zigbee Advantage

The Zigbee 3.0 protocol is designed to communicate data through noisy RF environments that are common in commercial and industrial market applications. Version 3.0 builds on the existing Zigbee connectivity standard but unifies the market-specific application profiles to allow all devices to be wirelessly connected in the same network, irrespective of their market designation and function. Furthermore, a Zigbee 3.0 certification scheme ensures the interoperability of products from different device manufacturers. Connecting Zigbee 3.0 networks to the IP domain opens up wireless monitoring and control from radio devices such as smartphones and tablets on a LAN or WAN, including the Internet, and brings the true Internet of Things to fruition.

Zigbee protocol features include:

- Support for multiple network topologies such as point-to-point, point-to-multipoint and mesh networks
- Low duty cycle – provides long battery life
- Low latency
- Direct Sequence Spread Spectrum (DSSS)
- Up to 65,000 nodes per network
- 128-bit AES encryption for secure data connections
- Collision avoidance, retries and acknowledgements

The Zigbee 3.0 software stack incorporates a 'base device' that provides consistent behavior for commissioning nodes and devices into a network. A common set of commissioning methods is provided, including Touchlink, a method of proximity commissioning.

Zigbee Wireless Security

Zigbee 3.0 provides enhanced network security. There are two methods of security that give rise to two types of network:

- Centralized security: This method employs a coordinator/trust center that forms the network and manages the allocation of network and link security keys to joining nodes.
- Distributed security: This method has no coordinator/trust center and is formed by a router. Any Zigbee router node can subsequently provide the network key to joining nodes.

Nodes adopt whichever security method is used by the hub network they join. Zigbee 3.0 supports the increasing scale and complexity of wireless networks, and copes with large local networks of greater than 250 nodes. Zigbee also handles the dynamic behavior of these networks (with nodes appearing, disappearing and re-appearing in the network) and allows orphaned nodes, which result from the loss of a parent, to re-join the network via a different parent. The self-healing nature of Zigbee Mesh networks also allows nodes to drop out of the network without any disruption to internal routing.

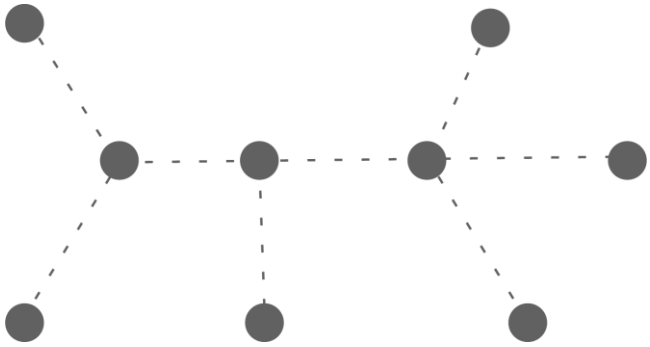
Zigbee Protocol Compatibility

The backward compatibility of Zigbee 3.0 means that applications and smart home devices already developed under the Zigbee Light Link 1.0 or Home Automation 1.2 profile are ready for Zigbee 3.0. The Zigbee Smart Energy profile is also compatible with Zigbee 3.0 at the functional level, but Smart Energy has additional security requirements that are only addressed within the profile.

Zigbee Device Data

Zigbee's Over-The-Air (OTA) upgrade feature for software updates during device operation ensures that applications on devices already deployed in the field/market can be seamlessly migrated to Zigbee 3.0. OTA upgrade is an optional functionality that manufacturers are encouraged to support in their Zigbee products' application layer.

Zigbee Mesh Networks



A key component of the Zigbee protocol is the ability to support mesh networking. In a mesh network, nodes are interconnected with other nodes so that multiple pathways connect each node. Connections between nodes are dynamically updated and optimized through sophisticated, built-in mesh routing table.

Mesh networks are decentralized in nature; each node is capable of self-discovery on the network. Also, as nodes leave the network, the mesh topology allows the nodes to reconfigure routing paths based on the new network structure. The characteristics of mesh topology and ad-hoc routing provide greater stability in changing wave conditions or failure at single nodes.

Zigbee Wireless Applications

Zigbee enables broad-based wave deployment of wireless networks with low-cost, low-power solutions. It provides the ability to run for years on inexpensive batteries for a host of monitoring and control applications. Smart energy/smart grid, AMR (Automatic Meter Reading), lighting controls, building automation systems, tank monitoring, HVAC control, medical devices, dbm radio, ghz wireless protocols, wireless sensor networks and fleet applications are just some of the many spaces where Zigbee technology is making significant advancements.

Digi XBee 3 Zigbee Technology

Digi is a member of the Zigbee Alliance and has developed a wide range of data networking solutions based on the Zigbee protocol. Digi XBee 3 is the latest in a long line of radio devices that provide an easy-to-implement solution that provides functionality to connect to a wide variety of devices with robust connectivity standards.

4.6. GPS

GPS uses satellites to monitor the movement of anything equipped with such a GPS tracking device, including automobiles, humans, and even pets. It operates in any weather condition and offers precise location updates in real-time. As one of the earliest ways to track and disseminate digital information from the real environment, GPS has significantly impacted IoT technology. The Internet of Things (IoT) may gather and measure enormous amounts of data on anything from individual health to public transportation; GPS tracking is required to provide location information for such objects.

A more reliable and easily accessible data set can be built using GPS and the Internet of Things. In the same way that GPS pinpoints the precise location of a vehicle, the Internet of Things is able to monitor moving items and collect data on their movements in real time.

How Does GPS Function?

GPS satellites complete two accurate orbits around the planet every day. An individual satellite's signal and orbiting parameters can be decoded and used to pinpoint the satellite's location via a GPS receiver. This data, together with triangulation, is used by GPS receivers to pinpoint the precise location of its owners.

The uses of GPS and the Internet of Things

When applied to Internet of Things (IoT) gadgets, GPS technology might provide benefits you might not have anticipated. In the age of the Internet of Things, it is possible to amass vast amounts of data from a wide variety of sources. Information such as medical files and facial mapping is included.

The Long-Term Perspective of GPS with IoT

It is possible that all of your belongings can be located using GPS. Italian firm Sherlock, for instance, uses GPS and the Internet of Things to keep tabs on your bike. Bicycle owners can use this kind of tracking to prevent bike theft. In addition, the owner of the vehicle will be informed of any modifications done to it.

Internet of Things-Global Positioning System helps boost transportation quality and safety. It was previously just possible to trace which foundation arrived, departed, etc., but now that we can manage the precise present location, transit quality & security are also enhanced.

Since you can interact with users' questions in real-time, you may use it with complete assurance.

Marketing a helpful transportation service. Users whose packages are too large to be transported conventionally can be attracted

Use IoT-GPS to set yourself apart in transportation management. Differentiation from competitors and end users can be achieved through the provision of novel approaches to management (barcode scanning, managing of arriving at the base by RFID, etc., starting picking up of crossing points on slips, etc.).

Cut back on the money spent on managing employees. By equipping workers with GPS trackers, employers can gain insight into the "what? who? where?" of their operations and, using that data, create more effective attendance and safety management systems. Can cut administrative expenses

Some Devices that are part of the Internet of Things and use a global positioning system

LORAWAN G62: Using publicly or private LoRaWAN (Long Range Wide Area Network) networks, the G62 LoRaWAN can keep tabs on any service, no matter how extreme the weather becomes. It is compact, lightweight, and built to withstand the elements.

LORAWAN SENSOR DATA: The Inertial Sensor is a battery-operated data communicator that connects to various sensors, GPS, inputs, and outputs and then uploads that data to a LoRaWAN network. Useful for sensors and agricultural uses.

LORAWAN, OR SHELLFISH: The Oyster is a rugged GPS tracking system that can withstand the elements and be made with the super-long battery life of LoRaWAN networks in mind for tracking exposed, unpowered assets in mind.

NEO-6M GPS Module with Arduino

This guide shows how to use the NEO-6M GPS module with the Arduino to get GPS data. GPS stands for *Global Positioning System* and can be used to determine position, time, and speed if you're travelling.

You'll learn how to:

- Wire the NEO-6M GPS module to the Arduino UNO
- Get raw GPS data
- Parse raw data to obtain selected and readable GPS information
- Get location

Introducing the NEO-6M GPS Module

The NEO-6M GPS module is shown in the figure below. It comes with an external antenna, and does't come with header pins. So, you'll need to get and solder some.

- This module has an external antenna and built-in EEPROM.
- Interface: RS232 TTL
- Power supply: 3V to 5V
- Default baudrate: 9600 bps
- Works with standard NMEA sentences

The NEO-6M GPS module is also compatible with other microcontroller boards..

Pin Wiring

The NEO-6M GPS module has four pins: VCC, RX, TX, and GND. The module communicates with the Arduino via serial communication using the TX and RX pins, so the wiring couldn't be simpler:

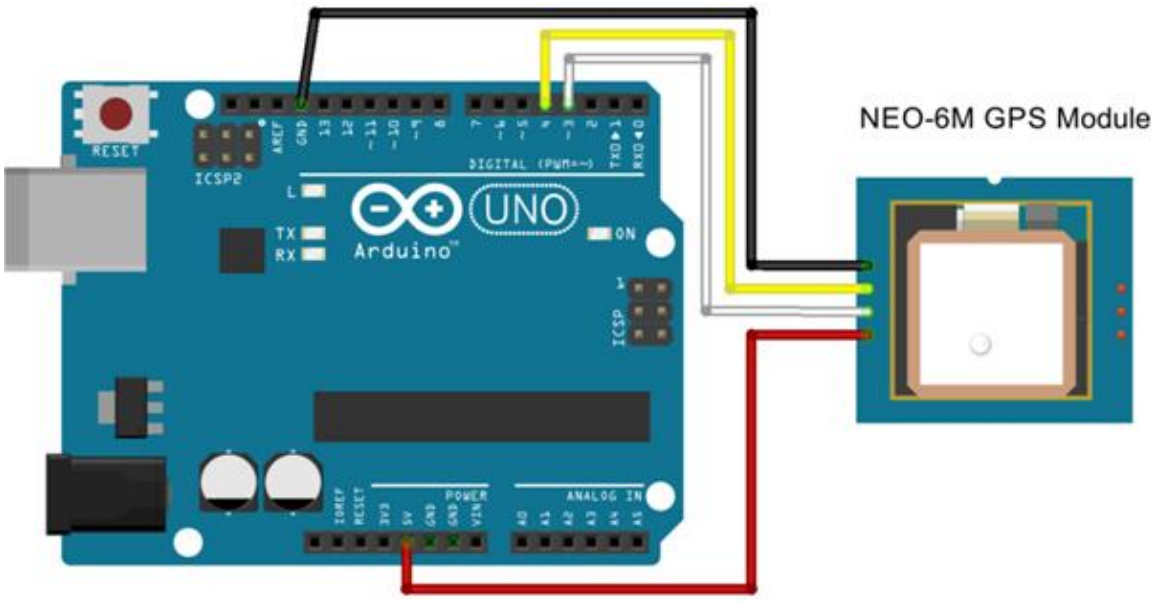
NEO-6M GPS Module	**Wiring to Arduino UNO**
VCC	5V
RX	TX pin defined in the software serial
TX	RX pin defined in the software serial
GND	GND

Getting GPS Raw Data

To get raw GPS data you just need to start a serial communication with the GPS module using Software Serial. Continue reading to see how to do that.

Parts Required

Wire the NEO-6M GPS module to your Arduino by following the schematic below.



- The module GND pin is connected to Arduino GND pin
- The module RX pin is connected to Arduino pin 3
- The module TX pin is connected to Arduino pin 4
- The module VCC pin is connected to Arduino 5V pin

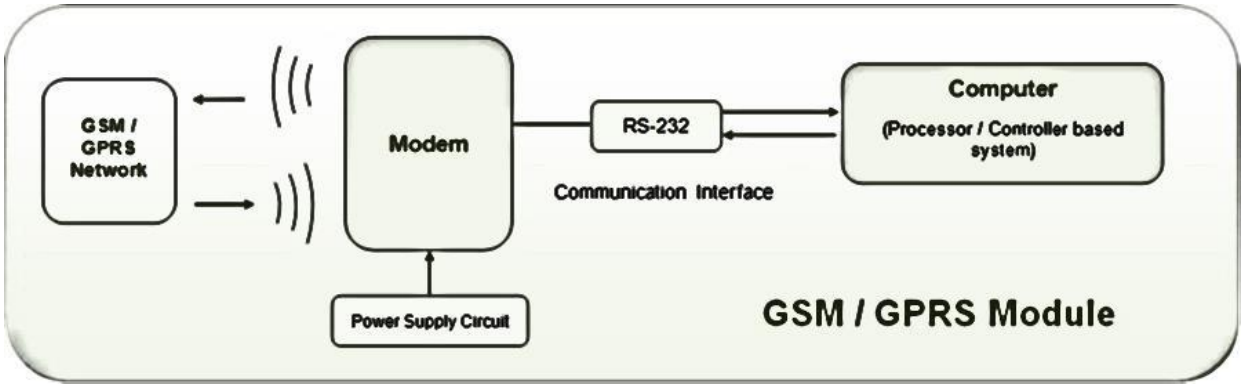
Code

Copy the following code to your Arduino IDE and upload it to your Arduino board.

```
#include <SoftwareSerial.h>

// The serial connection to the GPS module
SoftwareSerial ss(4, 3);

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

void loop(){
  while (ss.available() > 0){
    // get the byte data from the GPS
    byte gpsData = ss.read();
    Serial.write(gpsData);
  }
}
```

4.7. GSM/GPRS Module

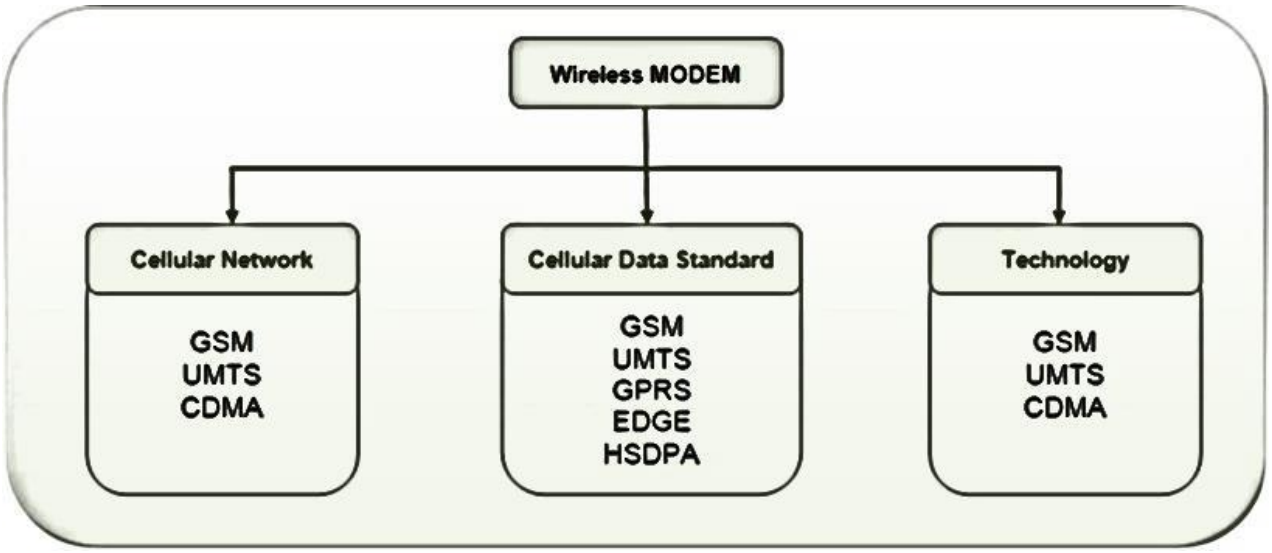
GSM/GPRS module is used to establish communication between a computer and a **GSM-GPRS system**. **Global System for Mobile communication (GSM)** is an architecture used for mobile communication in most of the countries.



Block Diagram Showing Diffrent Parts Of Typical GSM/GPRS Module

Global Packet Radio Service (GPRS) is an extension of GSM that enables higher data transmission rate. **GSM/GPRS module consists of a GSM/GPRS modem assembled together with power supply circuit and communication interfaces** (like RS-232, USB, etc) for computer. The MODEM is the soul of such modules.

Wireless MODEMs



Flow Chart Showing Classification Of Wireless Modem Based On Network, Data Or Technology

Wireless MODEMs are the MODEM devices that generate, transmit or decode data from a cellular network, for establishing communication between the cellular network and the computer. These are manufactured for specific cellular network (GSM/UMTS/**CDMA**) or specific cellular data standard (GSM/UMTS/GPRS/EDGE/HSDPA) or technology (GPS/SIM). Wireless MODEMs like other MODEM devices **use serial communication** to interface with and need **Hayes compatible AT commands** for communication with the computer (any microprocessor or microcontroller system).

GSM/GPRS MODEM

GSM/GPRS MODEM is a class of wireless MODEM devices that are designed for communication of a computer with the GSM and GPRS network. It requires a **SIM**

(Subscriber Identity Module) card just like mobile phones to activate communication with the network. Also they have **IMEI** (International Mobile Equipment Identity) number similar to mobile phones for their identification. A GSM/GPRS MODEM can perform the following operations:

1. Receive, send or delete SMS messages in a SIM.
2. Read, add, search phonebook entries of the SIM.
3. Make, Receive, or reject a voice call.

The MODEM needs **AT commands**, for interacting with processor or controller, which are communicated through serial communication. These commands are sent by the controller/processor. The MODEM sends back a result after it receives a command. Different AT commands supported by the MODEM can be sent by the processor/controller/computer to interact with the **GSM and GPRS cellular network**.

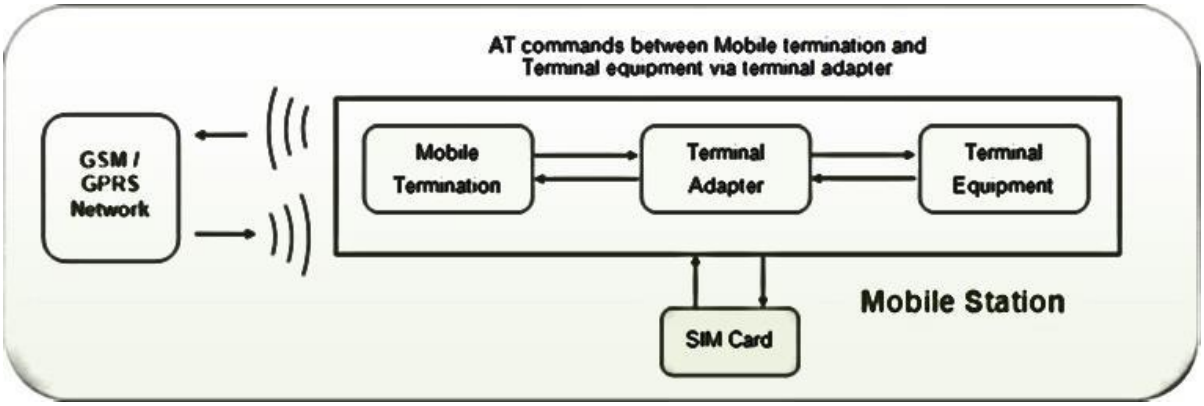
GSM/GPRS Module

An Image of Assembled Circuit For GSM/GPRS Module With Standard Communication Interfaces Like RS-232 Or USB

A GSM/GPRS module assembles a GSM/GPRS modem with standard communication interfaces like RS-232 (Serial Port), USB etc., so that it can be easily interfaced with a computer or a microprocessor / microcontroller based system. The power supply circuit is also built in the module that can be activated by using a suitable adaptor.

Mobile Station (Cell phones and SIM)

A mobile phone and Subscriber Identity Module (SIM) together form a mobile station. It is the user equipment that communicates with the mobile network. A mobile phone comprises of Mobile Termination, Terminal Equipment and Terminal Adapter.



Block Diagram Showing Diffrent Functions Of Cell Phone And SIM In Mobile Station

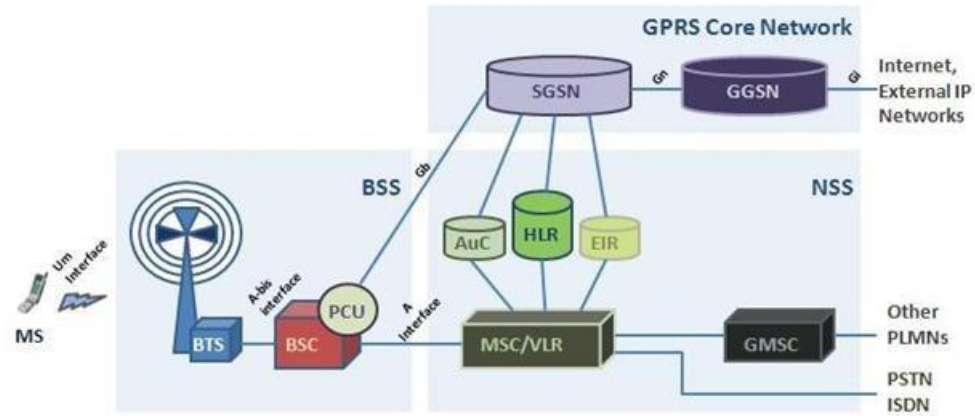
Mobile Termination is interfaced with the GSM mobile network and is controlled by a baseband processor. It handles access to SIM, speech encoding and decoding, signaling and other network related tasks. The **Terminal Equipment** is an application processor that deals with handling operations related to keypad, screen, phone memory and other hardware and software services embedded into the handset. The **Terminal Adapter** establishes communication between the Terminal Equipment and the Mobile Termination using AT commands. The communication with the network in a GSM/GPRS mobile is carried out by the baseband processor.

GSM Technology Architecture

The main elements in the GSM architecture include the following.

The Architecture of GSM Technology

- Network and Switching Subsystem (NSS)
- Base-Station Subsystem (BSS)
- The mobile station (MS)
- Operation and Support Subsystem (OSS)



Network Switching Subsystem (NSS)

In GSM system architecture, it includes different elements, which are frequently known as the core system/network. Here, it is basically a data network including a variety of units to

provide the major control as well as interfacing of the entire mobile network system. The core network includes the major elements which are discussed below.

Mobile Switching Centre (MSC)

The Mobile Switching Centre or MSC is the key element in the core network region of the GSM network architecture. This mobile services switching center works like a standard switching node in an ISDN otherwise PSTN, however, it also gives extra functionality to allow the mobile user necessities to be supported like authentication, registration, inter-MSC handovers call location & routing of the call to a cell phone subscriber.

And, it also provides an edge toward the public switched telephone network so that the phone calls can be connected from the network of the mobile to a phone to a landline. Interfaces to other mobile switching center server are provided to allow mobile calls to be made to mobiles over dissimilar networks.

Home Location Register (HLR)

This HLR database includes the information regarding the administrative like every subscriber with their previous identified location. Like this, the GSM network is capable to connect the calls to the related base station for the mobile switch. Once an operator turns ON his/her phone, and then the phone registers through the network so that it is probable to decide which base transceiver station is communicating so that incoming calls can be connected properly.

Even once the mobile is switched on, but not active then it again registers to make sure that the HLR network is responsive to its most recent location. There is one HLR for each network, even though it may be dispersed across a variety of sub-centers for operational causes.

Visitor Location Register (VLR)

The VLR includes preferred information that is received from the HLR network to allow the preferred services for the separate subscriber. The visitor location register can be executed like a separate unit; however, it is usually realized like an essential element of the MSC, before an individual unit. Thus, access is finished quicker & more convenient.

Equipment Identity Register (EIR)

The EIR (Equipment Identity Register) is the unit that makes a decision whether specified mobile gear may be permitted over the network. Every mobile gear includes a number identified like the IMEI or International Mobile Equipment Identity.

So, this IMEI number is fixed within the mobile equipment & is verified through the network while registration. It mainly depends on the information that is held within the EIR, and the mobile device may be assigned one of 3 conditions which allowed over the network, barred access, otherwise watched in case its problems.

Authentication Centre (AuC)

The AuC (authentication center) is a protected file that includes the secret key in the SIM card of the user. The AuC is mainly used for verification & for coding on the radio channel.

Gateway Mobile Switching Centre (GMSC)

The GMSC/ Gateway Mobile Switching Centre is the end to which a ME finishing call is primarily connected without any information about the MS's place. The GMSC obtains the Mobile Station Roaming Number (MSRN) from the MSISDN based on HLR & connecting the call toward the exact visited MSC. The "MSC" division of the name GMSC is confusing as the gateway process does not need any linking toward an MSC.

SMS Gateway (SMS-G)

The SMS gateway or SMS-G is used jointly to explain two SMS-Gateways in the GSM standards. These gateways control messages which are directed in dissimilar ways.

The Short Message Service Gateway Mobile Switching Centre (SMS-GMSC) is used for short messages which are being transmitted to an ME. The Short Message Service Inter-Working Mobile Switching Centre (SMS-IWMSC) is used for short messages created through a mobile network. The main role of SMS-GMSC is related to GMSC, but the SMS-IWMSC offers a permanent access end to the SMS Centre.

These units were the major ones that are used in the network of GSM technology. They were normally co-located, however frequently the overall middle network was transmitted around the country wherever the network was situated. In case of malfunction, it will give some flexibility.

Base Station Subsystem (BSS)

It acts as an interface between the mobile station and the network subsystem. It consists of the Base Transceiver Station which contains the radio transceivers and handles the protocols for communication with mobiles. It also consists of the Base Station Controller which controls the Base Transceiver station and acts as an interface between the mobile station and mobile switching center.

The network subsystem provides the basic network connection to the mobile stations. The basic part of the Network Subsystem is the Mobile Service Switching Centre which provides access to different networks like ISDN, PSTN, etc. It also consists of the Home Location Register and the Visitor Location Register which provides the call routing and roaming capabilities of GSM.

It also contains the Equipment Identity Register which maintains an account of all the mobile equipment wherein each mobile is identified by its own IMEI number. IMEI stands for International Mobile Equipment Identity.

The BSS or Base Station Subsystem section of the second generation GSM network architecture is basically connected with the mobiles over the network. This subsystem includes two elements which are discussed below.

Base Transceiver Station (BTS)

The BTS (Base Transceiver Station) which is utilized within a GSM network includes the radio Tx, Rx & their related antennas to transmit, receive & directly converse through the mobiles. This station is the important element for every cell and it converses with the mobiles & the interface among the two is identified like the Um interface with related protocols.

Base Station Controller (BSC)

The BSC (base station controller) is used to form the next phase reverse into the GSM technology. This controller is used to control a collection of base transceiver stations & it is frequently co-located through one of the transceiver stations within the group. This controller manages the resources of radio to control different items like handover in the collection of BTSs, assigns channels. It converses with the Base Transceiver Stations over Abis interface.

The subsystem element in the base station of the GSM network uses the radio allowable technology to allow a number of operators to right to use the system concurrently. Every channel supports up to 8 operators by allowing a base station to include different channels; a huge number of operators could be accommodated through every base station.

These are located carefully through the provider of the network to allow whole area coverage. This area can be enclosed with a base station that is often being called a cell. Because it is not achievable to stop the signals from overlapping into the nearby cells and channels which are used in single-cell are not utilized in the next.

Mobile Station

It is the mobile phone which consists of the transceiver, the display, and the processor and is controlled by a SIM card operating over the network.

The MS (Mobile stations) or ME (mobile equipment) are most generally identified through cell otherwise mobile phones which are the part of a GSM mobile communications n/w that the operator observes & operates. At present, their dimension has reduced radically whereas the functionality level has very much increased. And one more benefit is that the time among charges has drastically enlarged. There are different elements to the mobile phone, though the two essential elements are the hardware & the SIM.

The hardware includes the major elements of the mobile phone like the case, display, battery, & the electronics utilized to produce the signal & process the data receiver to be broadcasted.

The mobile station includes a number called the IMEI. This can be set up on the mobile phone while manufacturing & it cannot be modified.

It is accessed by the network during registration to check whether the equipment has been reported as stolen.

The SIM (Subscriber Identity Module) card includes the data which gives the user identity toward the network. And also, it includes different information like a number called the IMSI (International Mobile Subscriber Identity). When this IMSI is used in the SIM card, the mobile user could simply change mobiles by moving the SIM from one mobile to another.

So mobile changing is easy without changing the same mobile number means that people would frequently improve, thus making a further income stream for the providers of network & serving to enhance the total financial victory of GSM.

Operation and Support Subsystem (OSS)

The operation support subsystem (OSS) is a part of the complete GSM network architecture. This is connected to the NSS & the BSC components. This OSS is mainly used to control the GSM network & the BSS traffic load. It should be noted down that when the number of BS enhances through the subscriber population scaling then some of the preservation tasks are moved to the base transceiver stations so that the ownership cost of the system can be reduced.

The GSM network architecture of 2G mainly follows a logical technique of operation. This is very simple as compared with present architectures of mobile phone network which utilize software-defined units to allow extremely supple operation. But the architecture of 2G GSM will demonstrate the voice & operational fundamental functions that are required & how they arranged together. When the GSM system is digital, then the network is a data network.

Features of GSM Module

The features of the GSM module include the following.

- Improved spectrum efficiency
- International roaming
- Compatibility with integrated services digital network (ISDN)
- Support for new services.
- SIM phonebook management
- Fixed dialing number (FDN)
- Real-time clock with alarm management
- High-quality speech
- Uses encryption to make phone calls more secure
- Short message service (SMS)

The security strategies standardized for the GSM system make it the most secure telecommunications standard currently accessible. Although the confidentiality of a call and secrecy of the GSM subscriber is just ensured on the radio channel, this is a major step in achieving end-to-end security.

GSM Modem

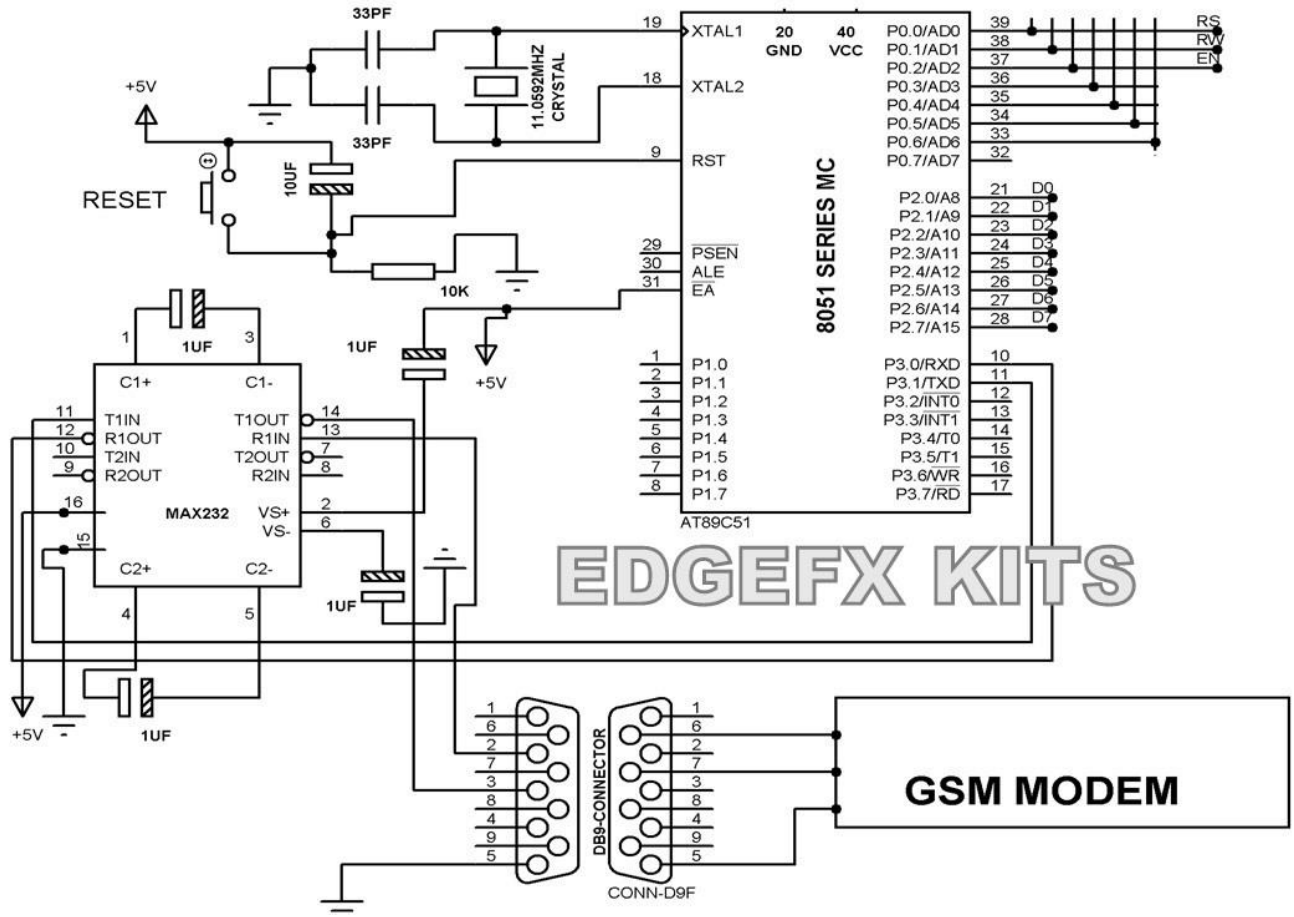
A GSM modem is a device that can be either a mobile phone or a modem device that can be used to make a computer or any other processor communicate over a network. A GSM modem requires a SIM card to be operated and operates over a network range subscribed by the network operator. It can be connected to a computer through serial, USB, or Bluetooth connection.

A GSM modem can also be a standard GSM mobile phone with the appropriate cable and software driver to connect to a serial port or USB port on your computer. GSM modem is usually preferable to a GSM mobile phone. The GSM modem has a wide range of applications in transaction terminals, supply chain management, security applications, weather stations, and GPRS mode remote data logging.

Working of GSM Module

From the below circuit, a GSM modem duly interfaced to the MC through the level shifter IC Max232. The SIM card mounted GSM modem upon receiving digit command by SMS from any cell phone sends that data to the MC through serial communication. While the program is executed, the GSM modem receives the command 'STOP' to develop an output at the MC, the contact point of which are used to disable the ignition switch.

The command so sent by the user is based on an intimation received by him through the GSM modem 'ALERT' a programmed message only if the input is driven low. The complete operation is displayed over a 16×2 LCD display.



GMS Modem Circuit

Difference between GSM mobile and GSM/GPRS module

A GSM mobile is a complete system in itself with embedded processors that are dedicated to provide an interface between the user and the mobile network. The AT commands are served between the processors of the mobile termination and the terminal equipment. The mobile handset can also be equipped with a USB interface to connect with a computer, but it may or may not support AT commands from the computer or an external processor/controller.

The GSM/GPRS module, on the other hand, always needs a computer or external processor/controller to receive AT commands from. GSM/GPRS module itself does not provide any interface between the user and the network, but the computer to which module is connected is the interface between user and network.

An advantage that GSM/GPRS modules offer is that they support concatenated SMS which may not be supported in some GSM mobile handsets. Also some mobile handsets can't receive MMS when connected to a computer.

Applications of GSM/GPRS module

The GSM/GPRS module demonstrates the use of AT commands. They can feature all the functionalities of a mobile phone through computer like making and receiving calls, SMS, MMS etc. These are mainly employed for computer based SMS and MMS services.

4.8. Raspberry Pi Operating Systems

Raspberry Pi is a low-cost pocket computer that is very economical to own. It is about the size of an ATM Card and can work as a fully functional computer in certain normal use cases, like working with simple applications, playing low-end games, etc. It was first released in 2012 by the Raspberry Pi foundation with the aim to provide easy access to computing education to everyone. It can cost as less as 5$ to a maximum price of 100$ (which is rare).

Scope

we will be understanding Operating systems that can be installed on a Raspberry Pi.

- We'll learn about What an operating system in general is.
- We'll go through a Variety of Operating Systems that a Raspberry Pi can run.

Introduction

As read above, Raspberry Pi is a very low-cost computer that comes along with the advantage of portability. However, being in such a small form factor, it gets bounded by the type of hardware to use in making it; hence, it will be significantly tough to run regular operating systems on it.

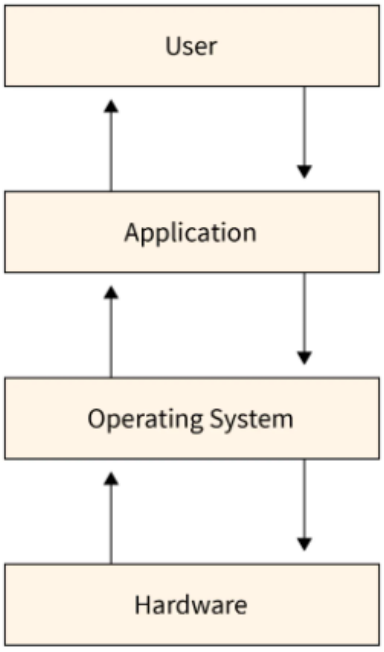
Due to this, specific operating systems were designed to power a Raspberry Pi; some of them were entirely new, while some originated from existing popular operating systems. Most of the Raspberry Pi OS is Linux based, but it also has windows 10-based Raspberry Pi OS (Windows 10 IoT core) built explicitly for low-powered devices like this.

What is an Operating System?

The technical definition is **An operating system is an interface between the hardware of a machine and the user who is using it**, but what does this really mean? It means it is basically the medium using which we communicate with our computer machine. It does not matter if we have the fastest system in the world; at the end of the day, that is just hardware, an object; and we do not know how to work with it, so we need some medium that works as an intermediate between us and the computer, i.e., when we press ctrl on our keyboard, it should instruct the computer what to react based on that; when we want to open some application, it should provide us a way to do so, by listing all the available applications on the system.

Meaning an operating system is a software program that helps us to use and to connect with the computer hardware. For example, if we want to use our mouse or keyboard, only with the help of an OS we can do that; if we want to install some program on our computer, we would be needing an OS; if we want to create a file, we need an OS; we want to delete a file, we would again be needing an OS, i.e., without an operating system we cannot use the computer

hardware, we would be needing some underlying software, i.e., some operating system, using which we would do so.



What is a Raspberry Pi Operating System?

Now, what is Raspberry Pi operating system? Before that, let's first try to understand what Raspberry Pi is. Raspberry Pi is a small, low-cost computer, and its size is about the size of an ATM card, which is developed by the Raspberry Pi foundation. The organization's mission is to educate people in computing and to provide easier access to computer education.

The above image is a picture of a Raspberry Pi; we can see that there are various ports available in it on which different devices can be mounted and used.

It was first launched in 2012, and from then onwards, various variations of it have been launched. The original Raspberry Pi had a single-core 700mhz CPU and a 256MB of RAM, but it has evolved a lot since then; today, we have a quad-core Raspberry Pi with a clock

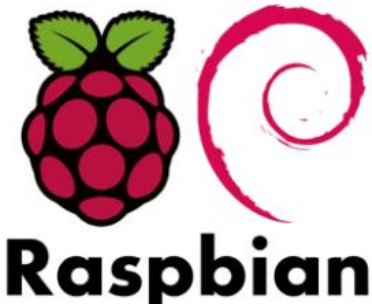
speed of around 1.5Ghz and up to 4GB of RAM. Surprisingly the cost of Raspberry Pi has always been less than 100 USD. In fact, the Raspberry Pi Zero (an even low-cost version of regular Raspberry Pi) costs as less as 5 USD. A full-fledged general-purpose CPU under 5$, that's what the organization's mission is "Aiming to provide people easier and low-cost access to the computers."

Raspberry Pi is used by people all around the world in learning how to program, build hardware projects, do home automation, and implement Kubernetes clusters, and it is even getting used in some industrial applications. Raspberry Pi is a very economical computer that runs Linux Operating System.

Now, let's talk about which specific distribution of Linux Raspberry Pi uses. Raspberry Pi officially recommends the use of the Raspbian Operating System. It is a Debian-based OS, explicitly made for Raspberry Pi and hence its name **Raspbian.**

Raspbian

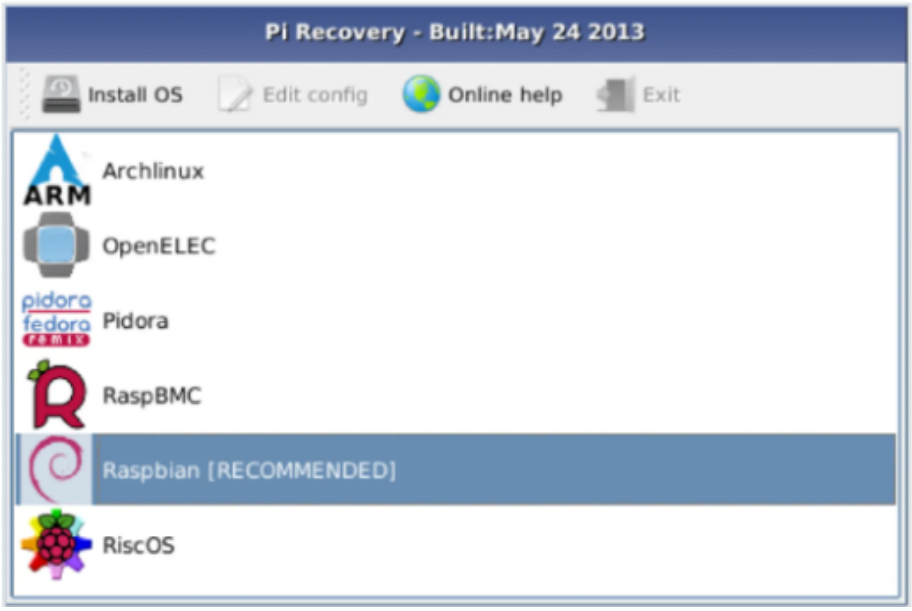
Raspbian or Raspberry Pi OS is a Linux-based operating system built specifically for Raspberry Pi. It is packed with all the necessary tools and features that are required for day-to-day use. It will possibly run on every kind of Raspberry Pi board with a few exceptions, like the Raspberry Pi's pico edition, because of its far smaller form factor and computing power.



NOOBS

New Out Of the Box Software, or simply NOOBS is an operating system installer for Raspberry Pi, delivered primarily on an SD card, which contains a variety of operating systems, out of which we can choose which one we want to install on our Raspberry Pi. It is made for people who are absolutely new to the Raspberry Pi and do not want to deal with the complex setting up process of burning an OS image on an SD card. NOOBS is provided along with every new Raspberry Pi at the time of its purchase.

With NOOBS, the user only needs to connect their Raspberry Pi to a display screen and a keyboard and then power it up; the NOOBs will boot. There we can select which operating system we want to install, and NOOBS will install the respective OS on the same SD card within a few minutes.



Other Operating Systems

Let's now look at some of the other Operating systems apart from the Raspbian OS, which can be booted on the Raspberry Pi.

Minibian

While Raspbian OS is very lightweight as far as we talk about operating systems, but if we want something even more straightforward, then we can go for Minibian OS. It is a minimal version of Raspbian for Raspberry Pi machines. The now-going version of minibian is based

on the recent Raspbian build and will run on all the versions of Raspberry Pi. Minibian differs from Raspbian in a number of ways, but the main difference is it is built more for electronics people instead of those who are interested in making a full-fledged computer. Minibian includes the core system and some fundamental utilities like web servers, some electronics applications, etc. Notably, it does not include a Graphical interface for interaction, which makes it an excellent choice for embedded projects. The result then is, we have a capable and working operating system in less than 500MB which merely uses 30MB of RAM.

Raspbian Lite

It is also a lightweight operating system for Raspberry Pi with a minimal set of packages. It is advised only for experienced users who are able to establish ssh connections and remote management using the command line, as it does not includes the GUI interface. It is more like a command line-based operating system where we would be needing to type commands to control our Raspberry Pi computer. One advantage of this is that it can run on a very low amount of CPU and RAM. It should be noted that Raspbiain lite OS can also be converted into a complete desktop environment with a GUI interface by installing the necessary packages.

RISC OS

RISC OS is an open-source OS initially developed in Cambridge in the 1980s and was the first operating system for ARM processors. It is neither related to Linux, Windows, or any other operating system; hence, diving into it will ignore our previous operating system experiences. Understanding the RISC OS will be a completely different thing, and it will take time for us to get used to it.

Windows IoT Core

It is a version of Windows OS, specifically windows 10, built explicitly for Raspberry Pi. It is helpful for windows enthusiasts who wish to utilize the Raspberry Pi environment to develop their projects. It is mostly used to construct and build IoT prototype requirements based on windows 10. This version of Raspberry Pi OS lays emphasis on the device mainly on connectivity, security, cloud integration, etc., regions.

Ubuntu Mate

Ubuntu Mate is a free and open-source operating system of raspberry pi that specifies low-performance devices as its design requirements, meaning it works seamlessly on low-power or old appliances. It is a flavor of Ubuntu that uses the Mate desktop environment. It prioritizes pre-packaging the apt package manager into its software deployment and talking about the communication to remote workstations; ubuntu MATE is dependable. To run it on a Raspberry Pi device, a 4GB sd card is all that is required. It should also be noted that ubuntu mate OS is consistently maintained and updated by its developers. It can be downloaded from the official website of Ubuntu MATE.

Gentoo Linux

Gentoo Linux is an open-source Linux-based lightweight operating system of the raspberry pi. Its built-in portage package manager makes the package installation straightforward and easy using emerge instead of apt installer. Moreover, it offers unique adaptability, which makes it an ideal Linux distribution for these low-powered devices. A 4GB SD Card will be enough to fulfill the device requirements of Gentoo OS on Raspberry Pi.

SARPI

Slackware ARM, or SARPI, is a well-known operating system in the Raspberry Pi community. It is often regarded as one of the substantial operating systems to run on the Raspberry Pi. This is because it has a very simple-to-implement software architecture. The boot time of SARPI OS is really less, making it live in less than 30 seconds of time.

FreeBSD

FreeBSD is an open-source Unix-like operating system derived from the Berkeley Software Distribution or BSD, which was based on the research of Unix (early versions of the Unix operating system). It works seamlessly with powerful hardware. Its software design makes it a significantly great OS to interact with servers, desktop PCs, IoT devices, cloud computing, etc. The FreeBSD OS can run on as less as a Raspberry Pi with 512MB of RAM; it can also be a perfect choice for one's Raspberry Pi system.

Lakka

A free, open-source, and lightweight operating system that can turn any PC into a full-fledged gaming console without the keyboard and mouse. It is basically used to set up emulators on systems like Raspberry Pi.

OpenELEC

Open Embedded Linux Entertainment Center or OpenELEC. It is a minimal-sized OS, often referred to as Just Enough Operating System or JeOS. If one is aware of the Kodi Media Center, which is a tool for managing media files. The primary purpose of this OS is to transform the running system into Kodi.

OSMC

OSMC is a free and open-source operating system of raspberry pi that stands for Open Source Media Center. It is simple and easy to work with the program that can independently supply Kodi Operating System functions like media file format management. Its user-friendly interface is simple and sophisticated, and its built-in image library contributes to its extensive customization possibilities. Therefore, it is an excellent choice if we majorly want to handle media materials on the Raspberry Pi system.

RaspBSD

RaspBSD is a special kind of FreeBSD operating system of raspberry pi and similar systems. It is an open-source OS software that contains the FreeBSD 11 image. The live image of RaspBSD features FreeBSD with the pre-installed Openbox window manager and the LXDE desktop environment. RaspBSD is based on FreeBSD's current tree.

Linutop

Linutop is a Raspbian OS-based Linux operating system of the raspberry pi. It's software security makes it the ideal choice for web kiosks and digital signage players. Moreover, it obligates its features and performance specifications to business users who wish for Raspberry Pi's digital signage and internet stall solutions.

Kali Linux

Kali is a Debian-based operating system particularly made for penetration testing. It comes with both the 32bit and 64bit architecture and is also one of the most feature-rich OS we have talked about so far. It has all the necessary tools needed for penetration testing installed by default. In addition, Kali's community is very effective and helpful. With Kali Linux, a Raspberry Pi device can become a toolset for penetration inside one's pocket.

Domoticz

Domoticz is a free and open-source home automation system that allows users to design and monitor various devices like switches, sensors, and meters for temperature, power, etc. Domoticz's main user interface is an expandable and scalable HTML5 frontend. Some of its outstanding features are its interactiveness with any browser, extensive logging, support for external devices, and auto-learning for sensors and switches.

OpenSUSE

The openSUSE project is a global initiative based on the RPM-based Linux distribution, which promotes the adoption of Linux OS everywhere by developing desktop and server operating systems. OpenSUSE operating system for Raspberry Pi is a part of the openSUSE project. The initiative's primary goal is to educate people about Linux. The openSUSE OS is developed, distributed, and maintained by an open-source community.

RetroPie

RetroPie is an emulator which can be used to play some low-end games on one's Raspberry Pi. It is a Debian-based open-source software library. It is elementary to set up and has broad community support. RetroPie uses the Emulation Station frontend RetroArch etc., which makes it possible to play old-school console games, nostalgic arcade games, and classic pc games. To add more to it, RetroPie sits on the top of the Raspberry Pi Lite Operating System, which omits the PIXEL (Pi improved XWindows Environment Lightweight) environment. And along with that, it can also be installed on top of a fully working Raspberry Pi OS. RetroPie is a popular emulation platform for playing retro games on the Raspberry Pi.

Arch Linux ARM

Arch Linux ARM is an easy-to-operate and lightweight Linux distribution, which is the official port of the Arch Linux OS. Arch Linux ARM maintains the initial philosophy of Arch developers, which is emphasizing simplicity, ownership, and use. It is built explicitly for the processors which support ARM architecture. Using this, the users will have complete

control of the system and can modify and customize it as per their needs. It could be an ideal choice for low-powered systems like Raspberry Pi.

Ubuntu Core

Ubuntu core is essentially designed to meet the needs of IoT devices. It is a minimal version of Ubuntu which requires less storage and memory to function. As Ubuntu is one of the most popular Linux distributions, hence its community has plenty of support and resources for users who are interested in ubuntu core. It is simple to set up and can be installed using the ubuntu snappy core installer.

Batocera.Linux

Batocera is a widespread open-source Linux distribution built primarily for retro gaming. While it could run on some standard computer systems, but is designed explicitly for microcomputers like Raspberry Pi. People who want to transform their Raspberry Pi computer into a gaming console are recommended to download Batocera.Linux

BMC64

BMC64 is a free and open-source system that is a bare metal version of VICE's C64 emulator. BMC64 proves to be an appropriate operating system for devices like Raspberry Pi. The reasons are it has the GIPO pins to connect simple keyboards and joysticks, it offers PCB scanning, real 50Hz/60Hz scrolling, low audio and video latency, an optimal startup time, etc., helpful features.

Notes:-

- Raspberry Pi is a low-cost, economical pocket-sized computer, the size of an ATM Card.
- It was developed by Raspberry Pi Foundation in 2012with the aim of providing people easy access to computing and computer education.
- Due to its small form factor, it is bounded by certain hardware limitations, and hence running regular operating systems on it will be comparatively effortful.
- Therefore, specific operating systems were designed to power devices like Raspberry Pi.
- Most of the operating systems for Raspberry Pi are Linux-based, but certain are not. For example - Windows 10 IoT core (windows 10-based Raspberry Pi Operating System), RISC OS (a completely different OS model, which is neither Linux nor windows), etc.
- Raspberry Pi devices are mainly used as portable computers used for building hardware projects, IoT applications, home automation, etc.
- Along with all these, we can also use a Raspberry Pi to play games, listen to music, do regular programming, etc.
- Raspberry Pi could be a great tool if one wants the power of an on-the-go computer along with the advantage of great portability.

4.9. Raspberry-Pi Architecture

What is a Raspberry Pi? Raspberry pi is the name of the "credit card-sized computer board" developed by the Raspberry pi foundation, based in the U.K. It gets plugged in a TV or monitor and provides a fully functional computer capability. It is aimed at imparting knowledge about computing to even younger students at the cheapest possible price. Although it is aimed at teaching computing to kids, but can be used by everyone willing to learn programming, the basics of computing, and building different projects by utilizing its versatility.

Raspberry Pi is developed by Raspberry Pi Foundation in the United Kingdom. The Raspberry Pi is a series of powerful, small single-board computers.

Raspberry Pi is launched in 2012 and there have been several iterations and variations released since then.

Various versions of Raspberry Pi have been out till date. All versions consist of a Broadcom system on a chip (SoC) with an integrated ARM-compatible CPU and on-chip graphics processing unit (GPU).

The original device had a single-core Processor speed of device ranges from 700 MHz to 1.2 GHz and a memory range from 256 MB to 1 GB RAM.

To store the operating system and program memory Secure Digital (SD) cards are used. Raspbian OS which is a Linux operating system is recommended OS by Raspberry Pi Foundation. Some other third party operating systems like RISC OS Pi. Diet Pi, Kali, Linux can also be run on Raspberry Pi.

Used:
It also provides a set of general purpose input/output pins allowing you to control electronic components for physical computing and explore the Internet of Things (IOT).
Raspberry pi Diagram:

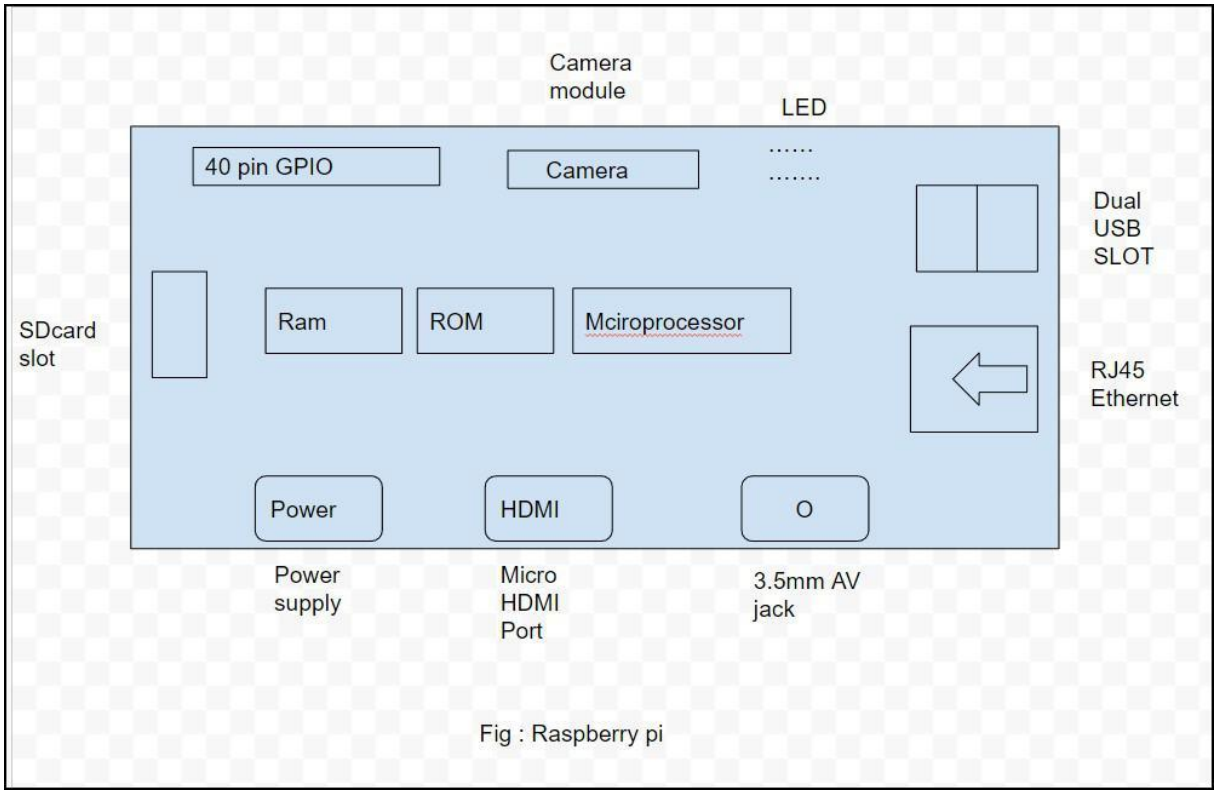


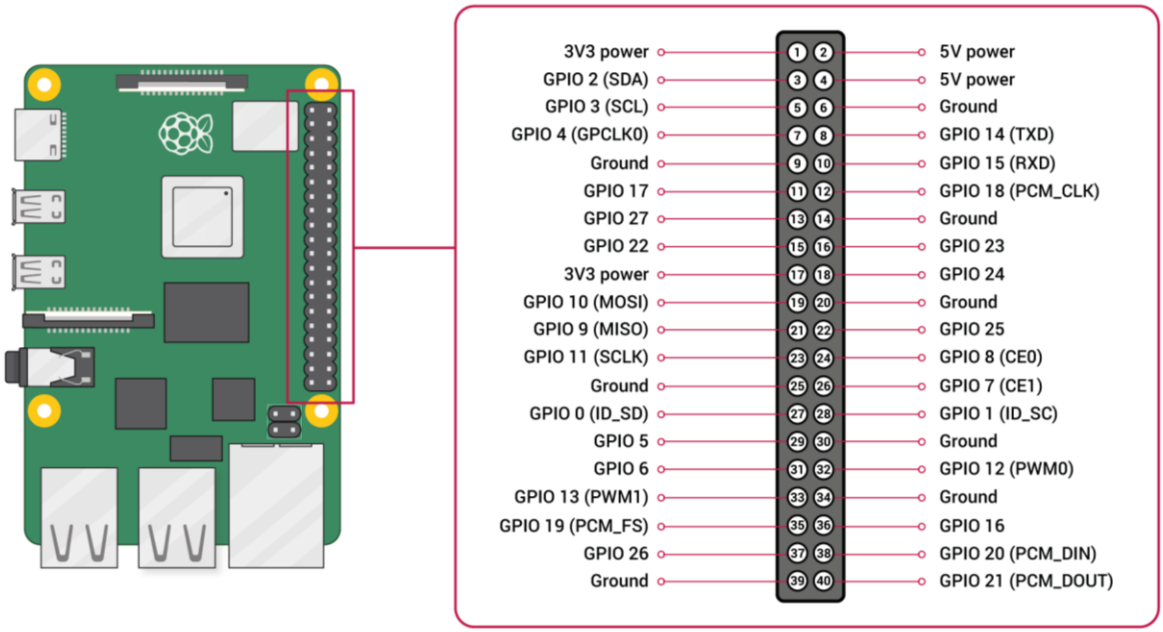
Fig : Raspberry pi

Raspberry Pi model –

There have been many generations of raspberry Pi from Pi 1 to Pi 4. There is generally a model A and model B. Model A is a less expensive variant and it trends to have reduced RAM and dual cores such as USB and Ethernet.

List of Raspberry pi models and releases year:

1. pi 1 model B – 2012
2. pi 1 model A – 2013
3. pi 1 model B+ -2014
4. pi 1 model A+ – 2014
5. Pi 2 Model B – 2015
6. Pi 3 Model B- 2016
7. Pi 3 Model B+ -2018
8. Pi 3 Model A+ -2019
9. Pi 4 Model A – 2019
10. Pi Model B – 2020
11. Pi 400 – 2021



Specs of the Computer: – The computer has a quad-core ARM processor that doesn't support the same instruction as an X86 desktop CPU. It has 1GB of RAM, One HDMI port, four USB ports, one Ethernet connection, Micro SD slot for storage, one combined 3.5mm audio/video port, and a Bluetooth connection. It has got a series of input and output pins that are used for making projects like – home security cameras, Encrypted Door lock, etc.

Versatility of Raspberry Pi: – It is indeed a versatile computer and can be utilized by people from all age groups, it can be used for watching videos on YouTube, watching movies, and programming in languages like Python, Scratch, and many more. As mentioned above it has a series of I/O pins that give this board the ability to interact with its environment and hence can be utilized to build really cool and interactive projects.

Examples of projects: – It can be turned into a weather station by connecting some instruments to it for check the temperature, wind speed, humidity etc… It can be turned into a home surveillance system due to its small size; by adding some cameras to it the security network will be ready. If you love reading books it can also become a storage device for storing thousands of eBooks and also you can access them through the internet by using this device.

Build Physical Projects With Python on the Raspberry Pi

The Raspberry Pi is one of the leading physical computing boards on the market. From hobbyists building DIY projects to students learning to program for the first time, people use the Raspberry Pi every day to interact with the world around them. Python comes built in on the Raspberry Pi, so you can take your skills and start building your own Raspberry Pi projects today.

Getting to Know the Raspberry Pi

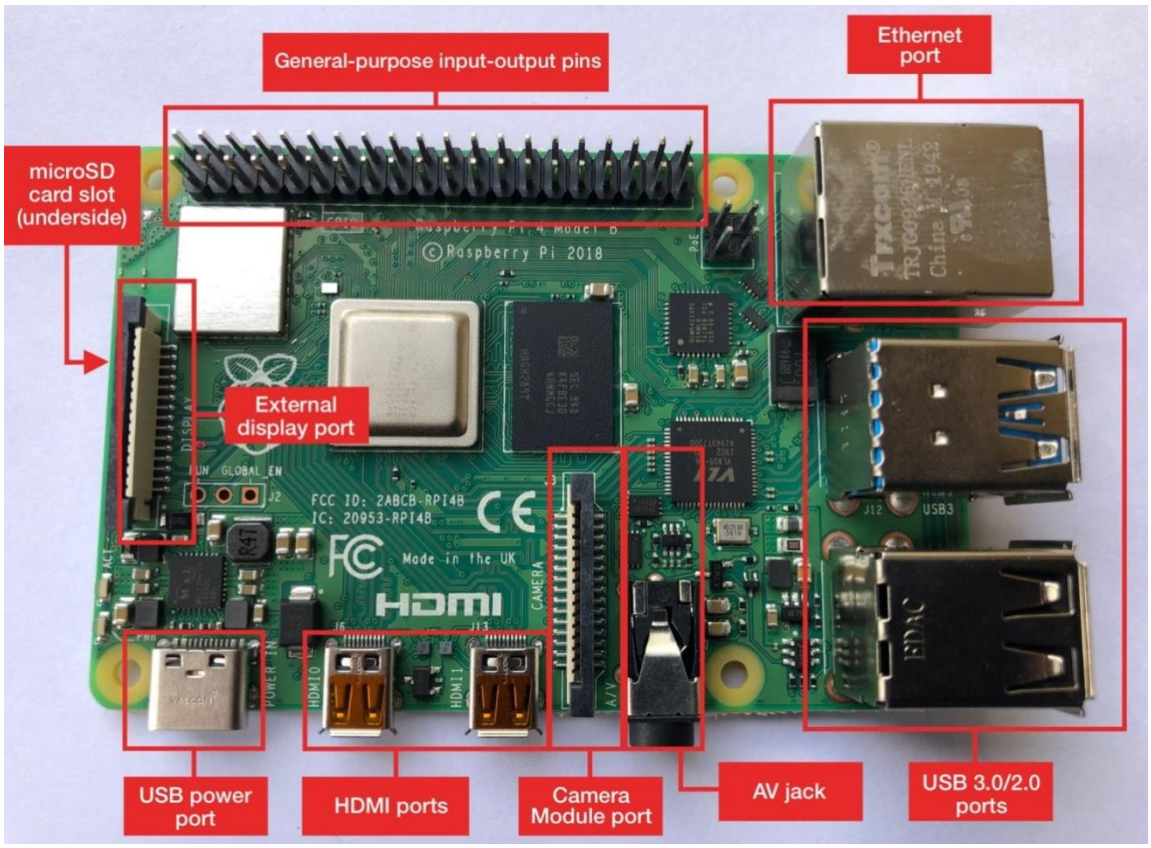
The Raspberry Pi is a single-board computer developed by the Raspberry Pi Foundation, a UK-based charity organization. Originally designed to provide young people with an

affordable computing option to learn how to program, it has developed a massive following in the maker and DIY communities because of its compact size, full Linux environment, and general-purpose input–output (**GPIO**) pins.

With all the features and capabilities that are packed into this small board, there's no shortage of projects and use cases for the Raspberry Pi.

If you can think of a project that would benefit from having a credit card–sized computer attached to it, then someone has probably used a Raspberry Pi to do it. The Raspberry Pi is a fantastic way to bring your Python project ideas to life.
Raspberry Pi Board Overview

Below is the board layout of the Raspberry Pi 4. While this layout is slightly different from previous models of the Raspberry Pi, most of the connections are the same. The setup described in the next section should be the same for both a Raspberry Pi 3 and a Raspberry Pi 4:



The Raspberry Pi 4 board contains the following components:

- **General-purpose input–output pins:** These pins are used to connect the Raspberry Pi to electronic components.
- **Ethernet port:** This port connects the Raspberry Pi to a wired network. The Raspberry Pi also has Wi-Fi and Bluetooth built in for wireless connections.

- **Two USB 3.0 and two USB 2.0 ports:** These USB ports are used to connect peripherals like a keyboard or mouse. The two black ports are USB 2.0 and the two blue ports are USB 3.0.
- **AV jack:** This AV jack allows you to connect speakers or headphones to the Raspberry Pi.
- **Camera Module port:** This port is used to connect the official Raspberry Pi Camera Module, which enables the Raspberry Pi to capture images.
- **HDMI ports:** These HDMI ports connect the Raspberry Pi to external monitors. The Raspberry Pi 4 features two micro HDMI ports, allowing it to drive two separate monitors at the same time.
- **USB power port:** This USB port powers the Raspberry Pi. The Raspberry Pi 4 has a **USB Type-C** port, while older versions of the Pi have a **micro-USB** port.
- **External display port:** This port is used to connect the official seven-inch Raspberry Pi touch display for touch-based input on the Raspberry Pi.
- **microSD card slot (underside of the board):** This card slot is for the microSD card that contains the Raspberry Pi operating system and files.

People often wonder what the difference is between a Raspberry Pi and an Arduino. The Arduino is another device that is widely used in physical computing. While there is some overlap in the capabilities of the Arduino and the Raspberry Pi, there are some distinct differences.

The Arduino platform provides a hardware and software interface for programming microcontrollers. A microcontroller is an integrated circuit that allows you to read input from and send output to electronic components. Arduino boards generally have limited memory, so they're often used to repeatedly run a single program that interacts with electronics.

The Raspberry Pi is a general-purpose, Linux-based computer. It has a full operating system with a GUI interface that is capable of running many different programs at the same time.
The Raspberry Pi comes with a variety of software preinstalled, including a web browser, an office suite, a terminal, and even Minecraft. The Raspberry Pi also has built-in Wi-Fi and Bluetooth to connect to the Internet and external peripherals.

For running Python, the Raspberry Pi is often the better choice, as you get a full-fledged Python installation out of the box without any configuration.

4.10. Raspberry Pi Programming

"Hello world" is the beginning of everything when it comes to computing and programming. It's the first thing you learn in a new programming language, and it's the way you test something out or check to see if something's working because it's usually the simplest way of testing simple functionality.

Warriors of programming language wars often cite their own language's "hello world" against that of another, saying theirs is *shorter* or *more concise* or *more explicit* or something. Having a nice simple readable "hello world" program makes for a good intro for beginners learning your language, library, framework, or tool.

I thought it would be cool to create a list of as many different "hello world" programs as possible that can be run on the Raspberry Pi using its Raspbian operating system, but without installing any additional software than what comes bundled when you download it from the Raspberry Pi website. I've created a GitHub repository of these programs, and I've explained 10 of them for you here.

1. Scratch

Scratch is a graphical block-based programming environment designed for kids to learn programming skills without having to type or learn the synax of a programming language. The "hello world" for Scratch is simple—and very visual!

1. Open **Scratch 2** from the main menu.

2. Click **Looks**.

3. Drag a **say Hello!** block into the workspace on the right.

4. Change the text to Hello world.

5. Click on the block to run the code.

2. Python

Python is a powerful and professional language that's also great for beginners— and it's lots of fun to learn. Because one of Python's main objectives was to be readable and stick to simple English, its "hello world" program is as simple as possible.

1. Open **Thonny Python IDE** from the main menu.
2. Enter the following code:

```
print("Hello world")
```

3. Save the file as hello3.py.
4. Click the **Run** button.

opensource.com

3. Ruby/Sonic Pi

Ruby is another powerful language that's friendly for beginners. Sonic Pi, the live coding music synth, is built on top of Ruby, so what users actually type is a form of Ruby.

1. Open **Sonic Pi** from the main menu.
2. Enter the following code:

```
puts "Hello world"
```

3. Press **Run**.

Unfortunately, "hello world" does not do Sonic Pi justice in the slightest

Alternatively, to using the Sonic Pi application for this example, you can write Ruby code in a text editor and run it in the terminal:

1. Open **Text Editor** from the main menu.
2. Enter the following code:

```
puts "Hello world"
```

3. Save the file as hello.rb in the home directory.
4. Open **Terminal** from the main menu.
5. Run the following command:

```
ruby hello.rb
```

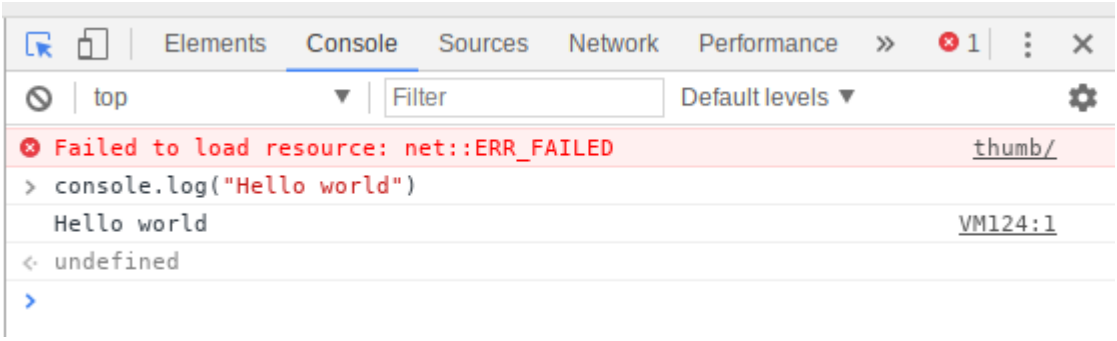
4. JavaScript

This is a bit of a cheat as I just make use of client-side JavaScript within the web browser using the Web Inspector console, but it still counts!

1. Open **Chromium Web Browser** from the main menu.
2. Right-click the empty web page and select **Inspect** from the context menu.
3. Click the **Console** tab.
4. Enter the following code:

```
console.log("Hello world")
```

5. Press **Enter** to run.



You can also install NodeJS on the Raspberry Pi, and write server-side JavaScript, but that's not available in the standard Raspbian image.

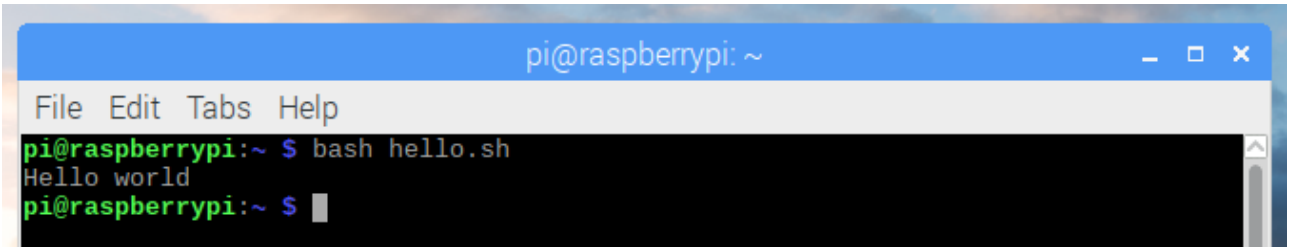
5. Bash

Bash (Bourne Again Shell) is the default Unix shell command language in most Linux distributions, including Raspbian. You can enter Bash commands directly into a terminal window, or script them into a file and execute the file like a programming script.

1. Open **Text Editor** from the main menu.
2. Enter the following code:

```
echo "Hello world"
```

3. Save the file as `hello.sh` in the home directory.
4. Open **Terminal** from the main menu.
5. Run the following command:
6. `ash hello.sh`



Note you'd usually see a "hashbang" at the top of the script (`#!/bin/bash`), but because I'm calling this script directly using the `bash` command, it's not necessary (and I'm trying to keep all these examples as short as possible).

You'd also usually make the file executable with `chmod +x`, but again, this is not necessary as I'm executing with `bash`.

6. Java

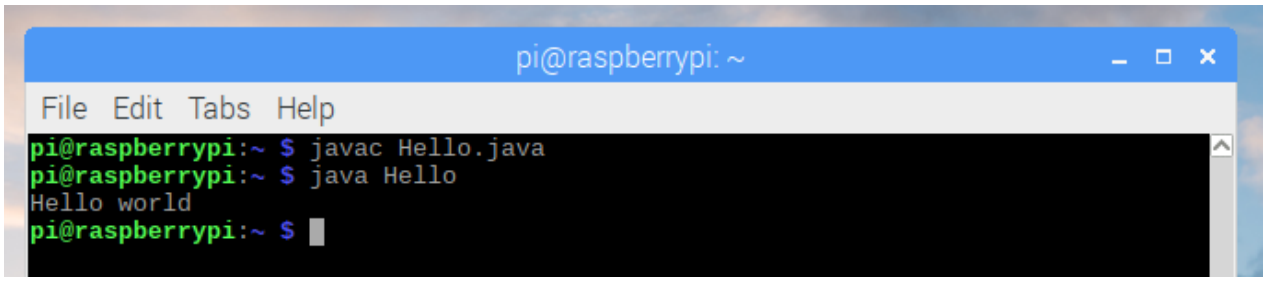
Java is a popular language in industry, and is commonly taught to undergraduates studying computer science. I learned it at university and have tried to avoid touching it since then. Apparently, now I do (very small amounts of) it for fun...

1. Open **Text Editor** from the main menu.
2. Enter the following code:

```
3. publicclassHello {

4. publicstaticvoidmain(String[] args) {

5.         System.out.println("Hello world");

6.     }

7.  }

8.
```

9. Save the file as Hello.java in the home directory.
10. Open **Terminal** from the main menu.
11. Run the following commands:

```
12.         javac Hello.java

    java Hello
```



I could *almost* remember the "hello world" for Java off the top of my head, but not quite. I always forget where the String[] args bit goes, but it's obvious when you think about it...

7. C

C is a fundamental low-level programming language. It's what many programming languages are written in. It's what operating systems are written in. See for yourself&mdash:take a look at the source for Python and the Linux kernel. If that looks a bit hazy, get started with "hello world":

1. Open **Text Editor** from the main menu.
2. Enter the following code:
3. #include <stdio.h>

4. int main() {

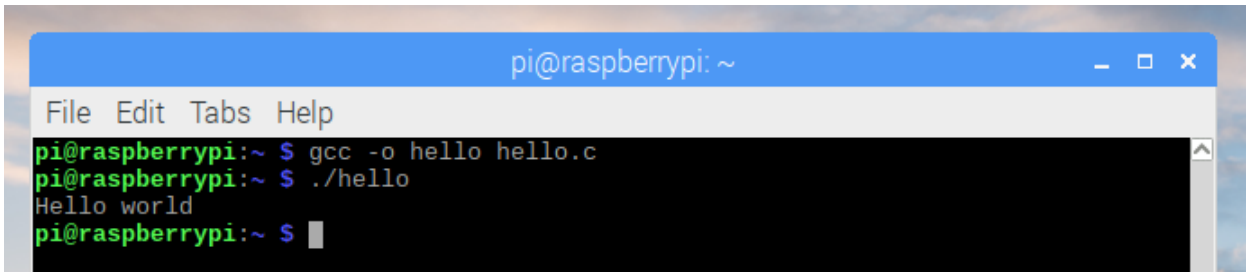
5. printf("Hello world\n");

}

6. Save the file as hello.c in the home directory.
7. Open **Terminal** from the main menu.
8. Run the following commands:

9. gcc -o hello hello.c

./hello



Note that in the previous examples, only one command was required to run the code (e.g., python3 hello.py or ruby hello.rb) because these languages are interpreted rather than compiled. (Actually Python is compiled at runtime but that's a minor detail.) C code is compiled into byte code and the byte code is executed.

If you're interested in learning C, the Raspberry Pi Foundation publishes a book Learning to code with C written by one of its engineers. You can buy it in print or download for free.

8. C++

C's younger bother, C++ (that's C incremented by one...) is another fundamental low-level language, with more advanced language features included, such as classes. It's popular in a range of uses, including game development, and chunks of your operating system will be written in C++ too.

Open **Text Editor** from the main menu.
Enter the following code:
#include <iostream>

```
using namespace std;
int main() {
   cout << "Hello world\n";
}
```

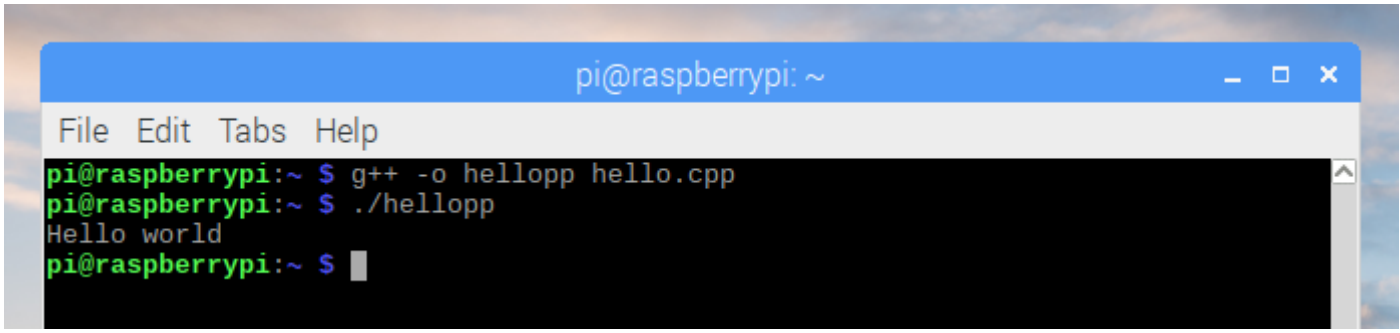
Save the file as hello.cpp in the home directory.

Open **Terminal** from the main menu.

Run the following commands:

```
1. g++ -o hellopp hello.cpp
```

```
./hellocpp
```



Readers familiar with C/C++ will notice I have not included the main function return values in my examples. This is intentional as to remove boilerplate, which is not strictly necessary.

9. Perl

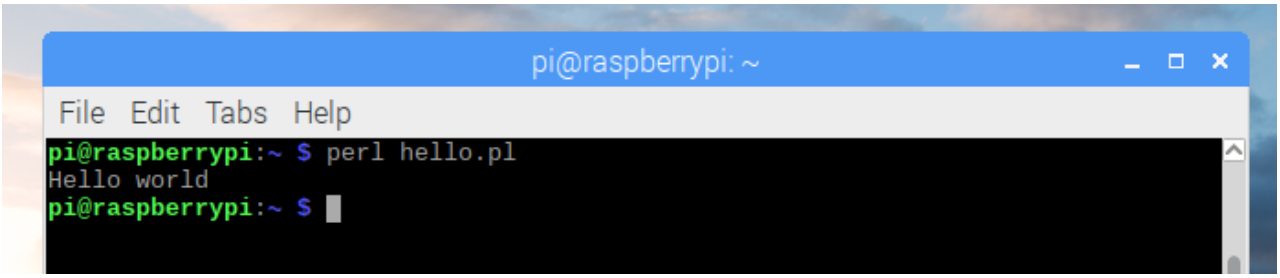
Perl gets a lot of stick for being hard to read, but nothing much gets in the way of understanding its "hello world" program. So far, so good!

1. Open **Text Editor** from the main menu.
2. Enter the following code:

```
print "Hello world\n"
```

3. Save the file as hello.pl in the home directory.
4. Open **Terminal** from the main menu.
5. Run the following command:

```
perl hello.pl
```



Again, I learned Perl at university, but unlike Java, I have managed to *successfully* avoid using it.

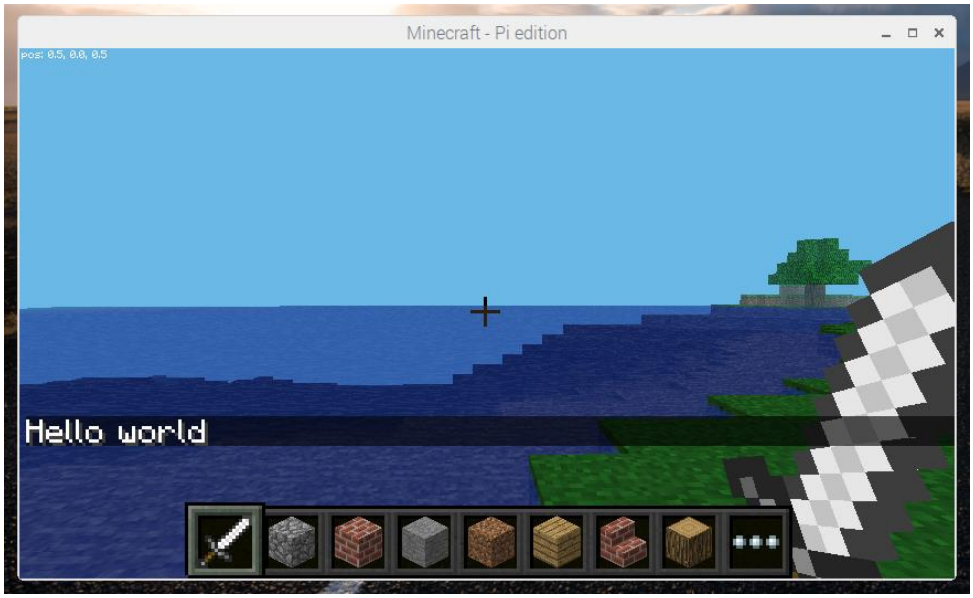
10. Python extras: Minecraft and the Sense HAT emulator

So that's nine different programming languages covered, but let's finish with a bit more Python. The popular computer game Minecraft is available for Raspberry Pi, and comes bundled with Raspbian. A Python library allows you to communicate with your Minecraft world, so open Minecraft and a Python editor side-by-side for some fun hacking your virtual world with code.

1. Open **Minecraft Pi** from the main menu.
2. Create and enter a Minecraft world.
3. Press **Tab** to release your focus from the Minecraft window.
4. Open **Thonny Python IDE** from the main menu.
5. Enter the following code:

```
from mcpi.minecraft import Minecraft

mc = Minecraft.create()

mc.postToChat("Hello world")
```

10. Save the file as hellomc.py.
11. Click the **Run** button.



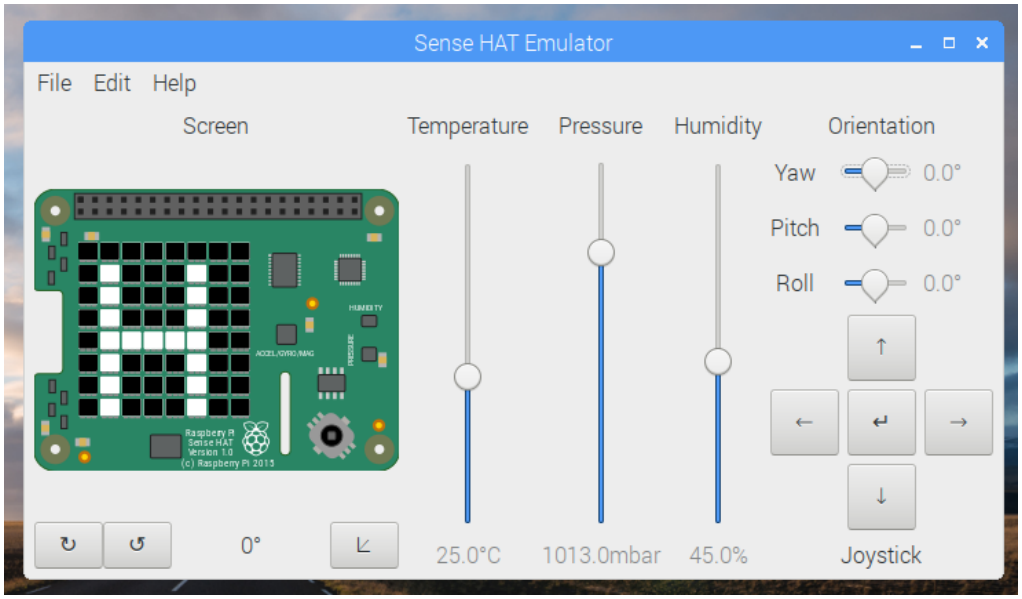
Finally, let's look at the Sense HAT Emulator. This tool provides a graphical representation of the Sense HAT.

The sense_emu Python library is identical to the sense_hat library except that its commands get executed in the emulator rather than on a physical piece of hardware. Because the Sense HAT includes an 8x8 LED display, we can use its show_message function to write "hello world".

1. Open another tab in Thonny and enter the following code:

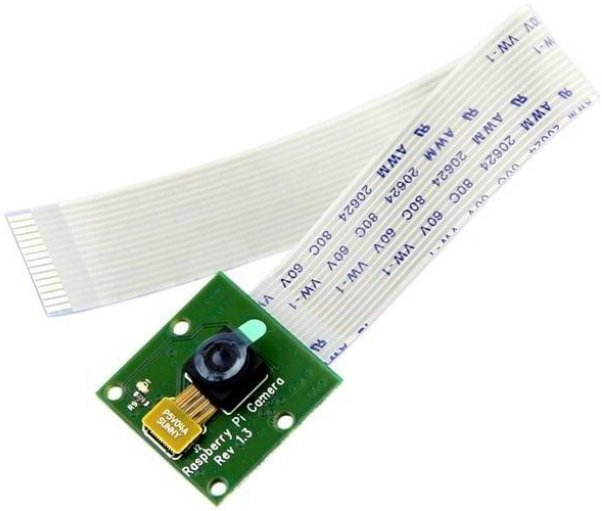
```
2. from sense_emu import SenseHat
3.
4. sense = SenseHat()
5.
   sense.show_message("Hello world")
```

6. Save the file as sense.py.
7. Click the **Run** button.



Example:

Pi Camera Module Interface with Raspberry Pi using Python



Pi Camera Module (v1.3)

The pi Camera module is a camera that can be used to take pictures and high definition video.

Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach the PiCamera module directly.

This Pi Camera module can attach to the Raspberry Pi's CSI port using a 15-pin ribbon cable.

Features of Pi Camera

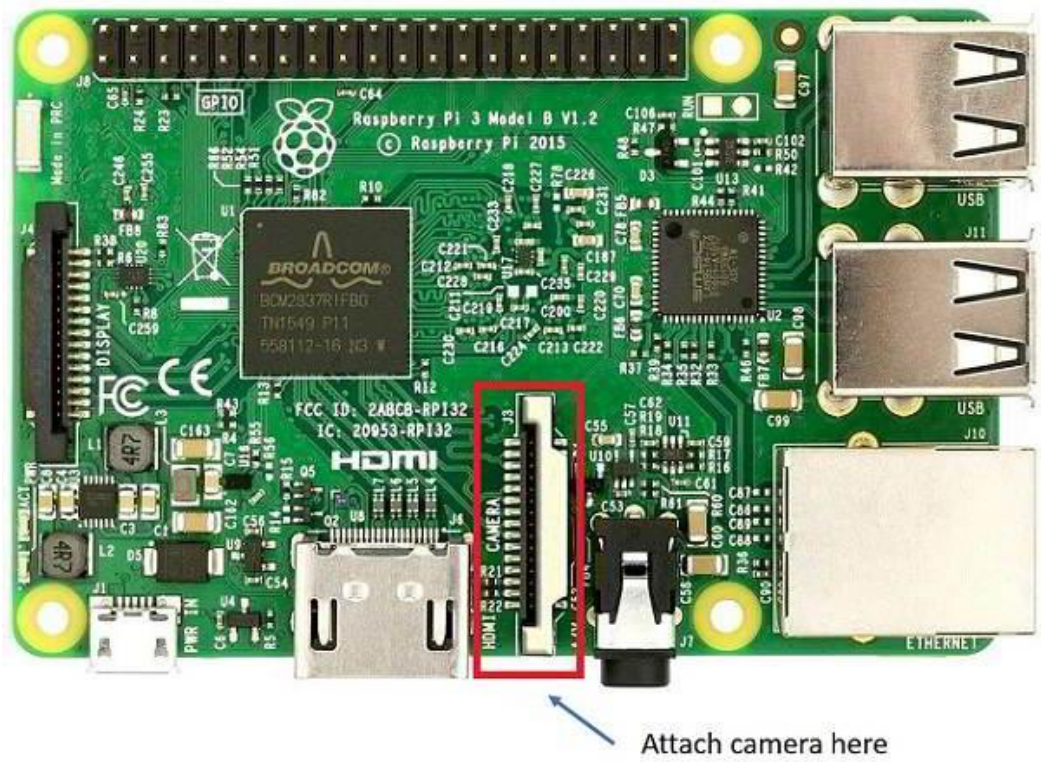
Here, we have used Pi camera v1.3. Its features are listed below,

- Resolution – 5 MP

- HD Video recording – 1080p @30fps, 720p @60fps, 960p @45fps and so on.
- It Can capture wide, still (motionless) images of a resolution 2592x1944 pixels
- CSI Interface enabled.

How to attach Pi Camera to Raspberry Pi?

Connect Pi Camera to the CSI interface of the Raspberry Pi board as shown below,



Now, we can use Pi Camera for capturing images and videos using Raspberry Pi.

Before using Pi Camera, we need to enable camera for its working.

How to Enable Camera functionality on Raspberry Pi

For enabling the camera in Raspberry Pi, open the raspberry pi configuration using the following command,

sudo raspi-config

then select **Interfacing options** in which select the **camera** option to enable its functionality.

reboot Raspberry Pi.

Now we can access the camera on Raspberry Pi.

Now we can capture images and videos using Pi Camera on Raspberry Pi.

Example

Capture images and save it to the specified directory.

We can capture images using Python. Here, we will write a Python program to capture images using Pi Camera on Raspberry Pi.

Here, we have used picamera package(library) which provides different classes for Raspberry Pi. Out of which we are mainly interested in PiCamera class which is for camera module.

Pi Camera Python Program for Image Capture

```
Camera

import picamera

from time import sleep

#create object for PiCamera class

camera = picamera.PiCamera()

#set resolution

camera.resolution = (1024, 768)

camera.brightness = 60

camera.start_preview()

#add text on image

camera.annotate_text = 'Hi Pi User'

sleep(5)

#store image

camera.capture('image1.jpeg')
```

camera.stop_preview()

Functions Used

To use picamera python based library we have to include it in our program as given below

import picamera

This picamera library has PiCamera class for the camera module. So, we have to create an object for **PiCamera** class.

PiCamera Class

To use Pi Camera in Python on Raspberry Pi, we can use PiCamera class which has different APIs for camera functionality. We need to create object for PiCamera class.

E.g.

Camera = picamera.PiCamera()

The above PiCamera class has different member variables and functions which we can access by simply inserting a dot (.) in between object name and member name.

E.g.

Camera.resolution = (1080, 648)

capture()

It is used to capture images using Pi Camera.

E.g.

Camera.capture("/home/pi/image.jpeg")

The capture() function has different parameters which we can pass for different operations like resize, format, use_video_port, etc.

E.g.

Camera.capture("/home/pi/image.jpeg", resize=(720, 480))

resolution= (width,height)

It sets the resolution of the camera at which image captures, video records, and previews will display. The resolution can be specified as **(width, height)** tuple, as a string formatted **WIDTHxHEIGHT**, or as a string containing commonly recognized display resolution names e.g. "HD", "VGA", "1080p", etc.

E.g.

Camera.resolution = (720, 480)

Camera.resolution = "720 x 480"

Camera.resolution = “720p”

Camera.resolution = “HD”

Annotate_text = “Text”

It is used to add text on images, videos, etc.

E.g.

Camera.annotate_text = “Hi Pi User”

start_preview()

It displays the preview overlay of the default or specified resolution.

E.g.

Camera.start_preview()

stop_preview()

It is used to close the preview overlay.

E.g.

Camera.stop_preview()

Note: There are various APIs of PiCamera class. So, to know more API in detail you can refer

PiCamera APIs.

Pi Camera Python Program for Video Recording

```
import picamera
from time import sleep
camera = picamera.PiCamera()
camera.resolution = (640, 480)
print()
#start recording using pi camera
camera.start_recording("/home/pi/demo.h264")
#wait for video to record
camera.wait_recording(20)
#stop recording
camera.stop_recording()
```

```
camera.close()
print("video recording stopped")
```

Functions used

We have to create an object for PiCamera class. Here, we have create objects as **camera.**

start_recording()

It is used to start video recording and store it.

E.g.

Camera.start_recording(‘demo.h264’)

It records video named demo of h264 format.

wait_recording(timeout)

Wait on the video encoder for specified timeout seconds.

E.g.

Camera.wait_recording(60)

stop_recording()

It is used to stop video recording.

E.g.

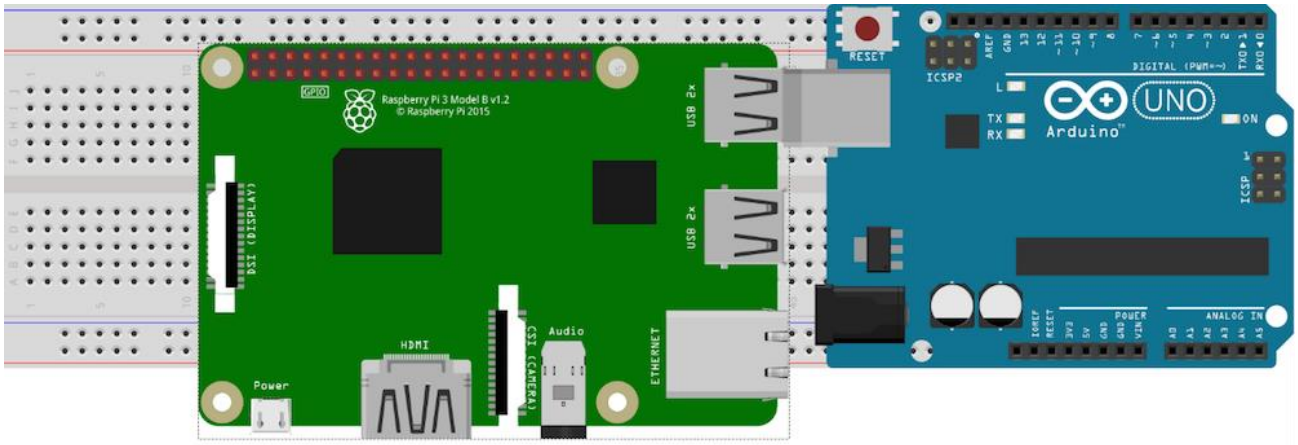
Camera.stop_recording()

Play Recorded Video

To open a video, we can use omxplayer by using the following command,

```
omxplayer video_name
```

4.11. Interface a Raspberry Pi with an Arduino



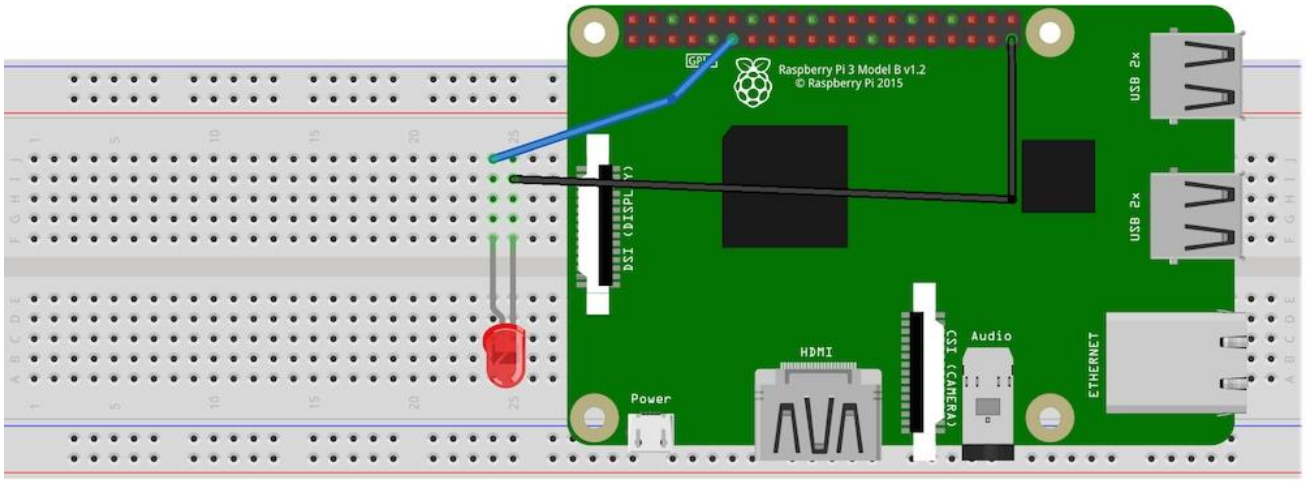
Interface a Raspberry Pi with an Arduino so the two boards can communicate with one another.

Sometimes you may need to connect an Arduino to a Raspberry Pi. For example, if you have sensors, motors, and actuators, you can connect these to the Arduino and make the Arduino send values to and from the Raspberry Pi. This way, we can separate the computing intensive tasks (done by the Raspberry Pi) and controlling tasks (done by the Arduino).

we will connect an Arduino to a Raspberry Pi and have the Arduino send “Hello from Arduino” to the Raspberry Pi, and the Raspberry Pi will blink an LED upon receiving the command from the Arduino.

For communication, we will use simple serial communication over USB cable.
So, let’s get started!

Connect the LED to pin number 11 as shown in the picture below.



Turn on the Raspberry Pi and open Python 3 in a new window.

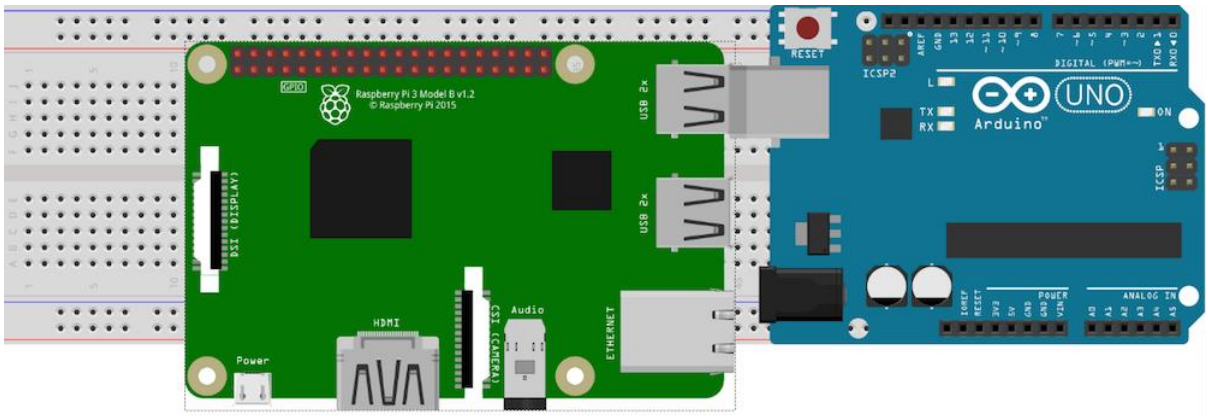
Write the following code in the new window and save it. (Save to your desktop so you don't lose it.)

```
import serial
import RPi.GPIO as GPIO
import time

ser=serial.Serial("/dev/ttyACM0",9600)    #change ACM number as found from ls /dev/tty/ACM*
ser.baudrate=9600
def blink(pin):

GPIO.output(pin,GPIO.HIGH)
time.sleep(1)
GPIO.output(pin,GPIO.LOW)
time.sleep(1)
return

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
while True:

read_ser=ser.readline()
print(read_ser)
if(read_ser=="Hello From Arduino!"):
blink(11)
```

Now open Arduino IDE and upload the following code to your Arduino.

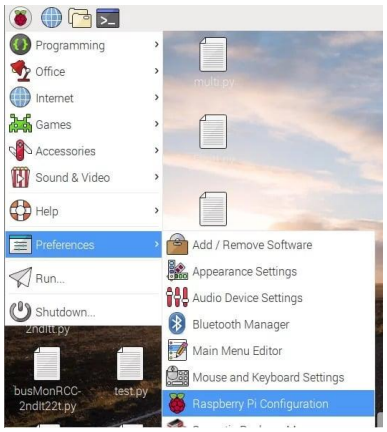
```
String data="Hello From Arduino!";
void setup()
{
```

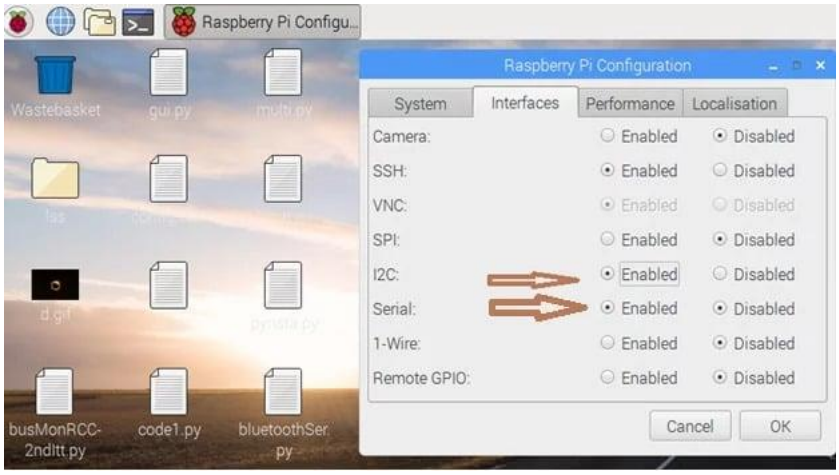
```
// put your setup code here, to run once:
Serial.begin(9600);
}
void loop()
{
// put your main code here, to run repeatedly:
Serial.println(data);//data that is being Sent
delay(200);
}
```



Make sure the code is uploaded to Arduino.

In your Raspberry Pi interface, be sure to enable Serial and I2C in PiConfig.



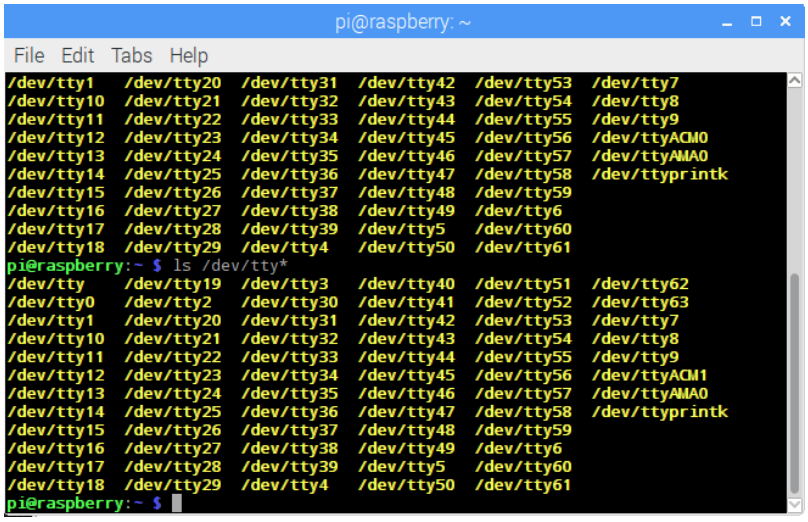


Next, you'll need to restart your Raspberry Pi. Open the Terminal and execute these commands:

sudo apt-get install python-serial

sudo pip install pyserial
Connect your Arduino to your Raspberry Pi.
Execute.
ls /dev/tty*

Then find a line with /dev/ttyACM0 or something like /dev/ttyACM1 etc. (check for an ACM with any number 0,1,2 etc.)



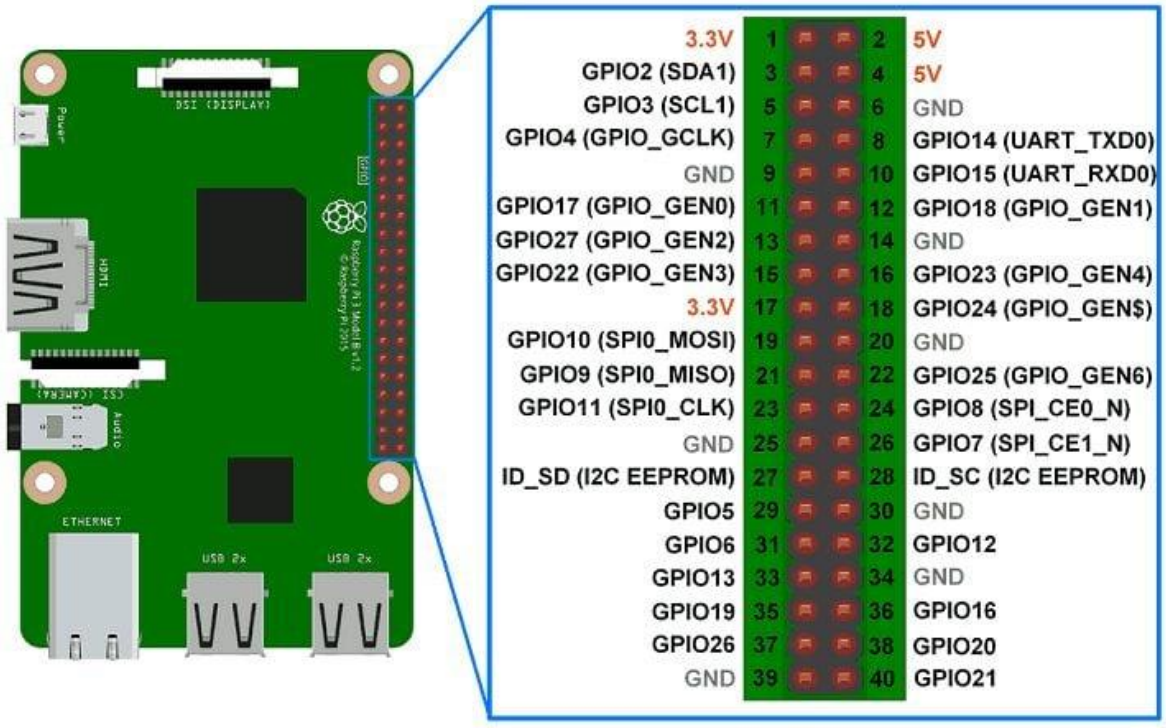
Open Python again and change ser=serial.Serial("dev/ttyACM1",9600) to the ACM number you found. So, if in your case you got ACM0, the line should look like this: ser=serial.Serial("dev/ttyACM0",9600). Now run the program you just created in Python3. You will see "Hello From Arduino!" in the Python terminal, and your LED should be blinking as well!

4.12. Raspberry Pi GPIO Access

GPIO (General Purpose Input Output) pins can be used as input or output and allows raspberry pi to connect with general purpose I/O devices.

- Raspberry pi 3 model B took out 26 GPIO pins on board.
- Raspberry pi can control many external I/O devices using these GPIO's.
- These pins are a physical interface between the Pi and the outside world.
- We can program these pins according to our needs to interact with external devices. For example, if we want to read the state of a physical switch, we can configure any of the available GPIO pins as input and read the switch status to make decisions. We can also configure any GPIO pin as an output to control LED ON/OFF.
- Raspberry Pi can connect to the Internet using on-board Wi-Fi or Wi-Fi USB adapter. Once the Raspberry Pi is connected to the Internet then we can control devices, which are connected to the Raspberry Pi, remotely.

GPIO Pins of Raspberry Pi 3 are shown in below figure:



Raspberry Pi 3 Model B GPIO Pin Mapping

Some of the GPIO pins are multiplexed with alternate functions like I2C, SPI, UART etc.

We can use any of the GPIO pins for our application.

Pin Numbering

We should define GPIO pin which we want to use as an output or input. But Raspberry Pi has two ways of defining pin number which are as follows:

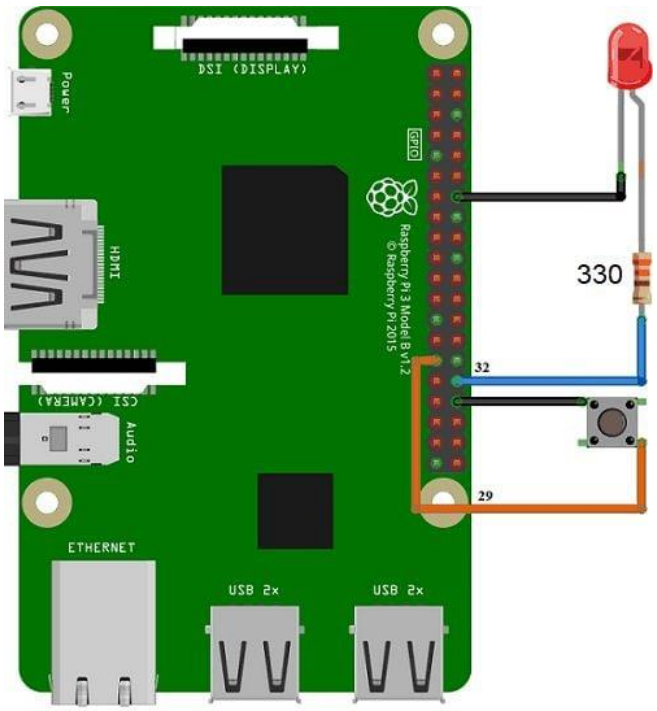
- **GPIO Numbering**
- **Physical Numbering**

In **GPIO Numbering**, pin number refers to number on Broadcom SoC (System on Chip). So, we should always consider the pin mapping for using GPIO pin.

While in **Physical Numbering**, pin number refers to the pin of 40-pin P1 header on Raspberry Pi Board. The above physical numbering is simple as we can count pin number on P1 header and assign it as GPIO.

But, still we should consider the pin configuration diagram shown above to know which are GPIO pins and which are VCC and GND.

Control LED with Push Button using Raspberry Pi



Control LED using Raspberry Pi Interfacing Diagram

Example

Now, let's control LED using a switch connected to the Raspberry Pi. Here, we are using Python and C (WiringPi) for LED ON-OFF control.

Control LED using Python

Now, let's turn an ON and OFF LED using Python on Raspberry Pi. The switch is used to control the LED ON-OFF.

Python Program for Raspberry Pi to control LED using Push Button

```
import RPi.GPIO as GPIO              #import RPi.GPIO module
LED = 32                             #pin no. as per BOARD, GPIO18 as per BCM
Switch_input = 29                    #pin no. as per BOARD, GPIO27 as per BCM
GPIO.setwarnings(False)              #disable warnings
GPIO.setmode(GPIO.BOARD)             #set pin numbering format
GPIO.setup(LED, GPIO.OUT)            #set GPIO as output
GPIO.setup(Switch_input, GPIO.IN, pull_up_down=GPIO.PUD_UP)
while True:
   if(GPIO.input(Switch_input)):
      GPIO.output(LED,GPIO.LOW)
   else:
      GPIO.output(LED,GPIO.HIGH)
```

Functions Used:

RPi.GPIO

To use Raspberry Pi GPIO pins in Python, we need to import RPi.GPIO package which has class to control GPIO. This RPi.GPIO Python package is already installed on Raspbian OS. So, we don't need to install it externally. Just, we should include library in our program to use functions for GPIO access using Python. This is given as follows.

import RPi.GPIO as GPIO

GPIO.setmode (Pin Numbering System)

This function is used to define Pin numbering system i.e. GPIO numbering or Physical numbering.

In RiPi.GPIO GPIO numbering is identified by **BCM** whereas Physical numbering is identified by **BOARD**

Pin Numbering System = BOARD/BCM

E.g. If we use pin number 40 of P1 header as a GPIO pin which we have to configure as output then,

In BCM,

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
```

In BOARD,

GPIO.setmode(GPIO.BOARD)

GPIO.setup(40, GPIO.OUT)

GPIO.setup (channel, direction, initial value, pull up/pull down)

This function is used to set the direction of GPIO pin as an input/output.

channel – GPIO pin number as per numbering system.

direction – set direction of GPIO pin as either Input or Output.

initial value – can provide initial value

pull up/pull down – enable pull up or pull down if required

Few examples are given as follows,

- GPIO as Output

  GPIO.setup(channel, GPIO.OUT)

- GPIO as Input

  GPIO.setup(channel, GPIO.IN)

- GPIO as Output with initial value

  GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)

- GPIO as Input with Pull up resistor

  GPIO.setup(channel, GPIO.IN, pull_up_down = GPIO.PUD_UP)

GPIO.output(channel, state)

This function is used to set the output state of GPIO pin.

channel – GPIO pin number as per numbering system.

state – Output state i.e. HIGH or LOW of GPIO pin

e.g.

GPIO.output(7, GPIO.HIGH)

GPIO.input(channel)

This function is used to read the value of GPIO pin.

e.g.

GPIO.input(9)

Control LED using C (WiringPi)

We can access Raspberry Pi GPIO using C. Here, we are using WiringPi library for accessing Raspberry Pi GPIO using C.

Before implementing LED blinking using wiringPi, you can refer How to use WiringPi library.

C (WiringPi) Program for Raspberry Pi to control LED using Push Button

```
#include <wiringPi.h>
#include <stdio.h>
int LED = 26; /* GPIO26 as per wiringPi, GPIO12 as per BCM, pin no.32 */
int switch_input = 21;  /* GPIO21 as per WiringPi, GPIO5 as per BCM, pin no.29 */
int main(){
        wiringPiSetup();    /* initialize wiringPi setup */
        pinMode(LED,OUTPUT);   /* set GPIO as output */
        pullUpDnControl(switch_input, PUD_UP);
        while (1){
                if(digitalRead(switch_input))
                        digitalWrite(LED,LOW); /* write LOW on GPIO */
                else
                        digitalWrite(LED, HIGH);   /* write HIGH on GPIO */
                }
        }
```

4.13. Sending and Receiving Signals Using GPIO Pins

In recent years, the Raspberry Pi has become popular largely as an inexpensive, compact Linux box for media and retro video games, as well as a network device. Some hobbyists go on to use their Pi these ways for years, all without knowing what the pins on the side of their device really do.

It's these pins that hold the true power of the Pi. They can control homes, machines, new inventions, and even robots, so why is it so many people don't know what they really are?

What Do These Pins Actually Do?

These 40 (or 26, depending on your Pi model) pins are part of what's known as the "GPIO Header". Within this header, there are four main kinds of pin;

- **Power**: These provide DC power at 3.3 and 5 volts
- **Ground (GND)**: These connect to ground, to close your circuit
- **DNC**: This stands for “do not connect”, so don’t worry about them
- **GPIO**: These can be set to either send, or receive control voltages

GPIO stands for ‘General Purpose Input/Output’, and it’s these pins that let the Raspberry Pi do its magic. This is because the pins have no specific function, and can be set to a dedicated purpose, such as controlling a signal.

A GPIO pin set to output can provide either 3.3 volts, known as a HIGH signal, or 0 volts, known as a LOW signal. When set to input, it can read these same voltages.

GPIO Pins Don’t Provide Much Power

It’s important to remember that GPIO pins (and the 3.3-volt power pins) are meant to control and communicate with other components.

You can get about 51mA from all 3.3 volt pins combined, but you’ll want to take care when connecting; if your circuit tries to pull too much current through these 3.3 volt pins, you can fry the whole board.

The 5-volt power pins, on the other hand, give you all the power available from the power supply, minus the bit used by the Raspberry Pi itself.

4.13.1. Connecting Your GPIO Pins to a Breadboard

When you first start using these GPIO pins, it’s wise to use a breadboard. This makes it easy to build circuits without solder and to modify them.

If you’ve never used a breadboard before, familiarize yourself with the basics here:

A GPIO extension board also helps immensely. This connects the GPIO header via a ribbon and places the pins directly on the breadboard, in a clearly labeled manner.

It requires some real estate though: 20 rows each side of the breadboard. On a small board, that's nearly the whole thing! A breadboard with 40 rows or so leftover gives plenty of space for beginner projects.

4.13.2. GPIO Pins With Special Uses

Every GPIO pin can be set to send and receive HIGH and LOW signals. Some have special uses too.

Hardware PWM

GPIO pins output either 3.3 or 0 volts: a HIGH or LOW signal. Pulse width modulation, or PWM, is a way of simulating the range of voltages in between by flickering the pin on and off rapidly.

This isn't a true analog signal, but it's fine for something like dimming an LED. It will flicker faster than you can see and simply appear dimmer.

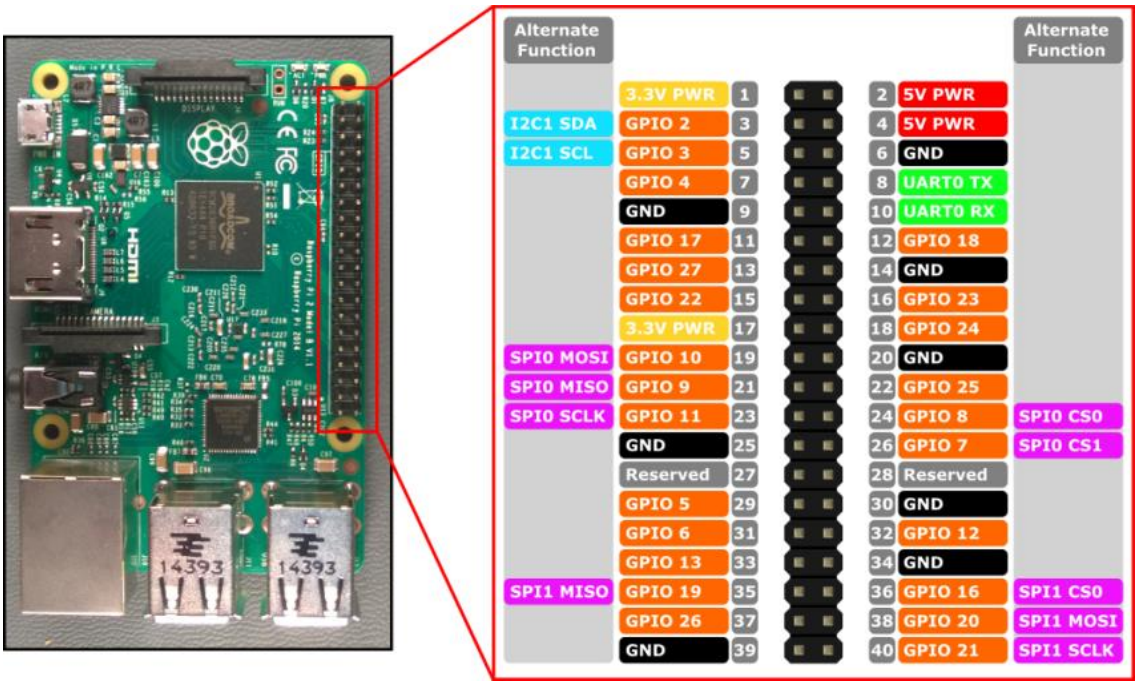
You can also use a low pass filter to smooth a PWM into an analog signal. This can be used for analogue audio, if you aren't fussy about quality. It's fine for a doorbell or toy.

You can generate a PWM signal from any GPIO pin using software, but the operating system juggles this with other tasks, so this signal can jitter.

Hardware PWM is available on GPIO pins 18 and 19. Hardware PWM and the headphone jack use the same circuits, so shouldn't be used simultaneously.

4.13.3. Serial Bus Pins

If you take a look below at the diagram (known as a Raspberry Pi 'Pinout') you'll see that some pins are I2C, SPI, and UART serial. These are serial bus protocols that can be used to send and receive data from other components.



You can combine these with a digital-analogue converter, or DAC, to output an analogue signal. This can be preferable to PWM for high quality audio, or to control many components.

4.13.4. Pull Up and Pull Down Resistors

Often you will want your Raspberry Pi GPIO pin to read the position of a button or a switch. That's easy to do by wiring it so that it closes a circuit attached to the control voltage to read HIGH, or to ground to read LOW.

The problem is that when this circuit is open and nothing else is attached to the pin, it might return any value. This is known as "floating," and it's extremely unhelpful.

You can prevent floating with "pull up" or "pull down" resistors.

A pull up resistor is wired to your control voltage; when nothing else is attached, the pin will read HIGH. A pull down resistor is wired to ground; the pin will read LOW. Use whichever provides the opposite value to your switch or button.

You don't need to wire these resistors into your circuit. They're inside the Raspberry Pi already and you can control them from software.

4.13.5. Using Software to Control GPIO Pins

Among the easiest ways to control GPIO pins is by using the GPIO Zero library in Python. If you've written any Python before, you'll pick this up easy.

If this is your first time using Python, If you don't, the commands below will still work; you'll just be less able to follow along. The web version of 'Automate the Boring Stuff With Python' is excellent and costs nothing.

GPIO Zero is installed by default on Raspbian Desktop images. If you are using Raspbian Lite or a different operating system, you may need to install it.

Let's Use All This to Switch a Light On

Let's have a go at turning on an LED! A job this simple doesn't really require a computer, but we'll involve the Raspberry Pi in the GPIO pins.

For this, you'll need….

A Raspberry Pi with power supply and an SD card with Raspbian installed	
A breadboard	
A GPIO extension board (optional, but recommended)	

An LED	

You'll also need some more general equipment, such as;

- A resistor with a value somewhere between 220 and 1000 Ohms
- A USB keyboard, or an SSH connection: something that lets you type commands
- Jumper cables or wires
- A pushdown button

Connecting the Power Rails

If you're using the extension board, connect it to the Raspberry Pi and to the breadboard. Then attach the 3.3-volt power pin to the positive power rail running across the bottom of the breadboard, and the ground pin to the negative power rail.

Connecting and Testing the Button

Now add your button to the middle of the breadboard. Connect one pin of the button to a Raspberry Pi GPIO pin. I'm using 13, because it's my lucky number.

Then, connect the diagonally opposite pin of the button to the negative power rail. When you push this button down, the circuit closes.

Finally, we need to tell the Pi to pay attention to this pin, so let's open the Python interpreter. At the command prompt, type:

python3

Now at the interpreter, type:

from gpiozero import Button

If you get a message saying 'ImportError', make sure you capitalized it correctly. If it says 'ModuleNotFoundError', you need to install GPIO Zero. Otherwise, it's time to assign our pin to the button:

button = Button(13)

This Button class takes care of assigning the pull up resistor. Now let's test that it works by typing the following lines:

```
while True:
  if button.is_pressed:
```

```
print('Sweet, the button works!')
break
```

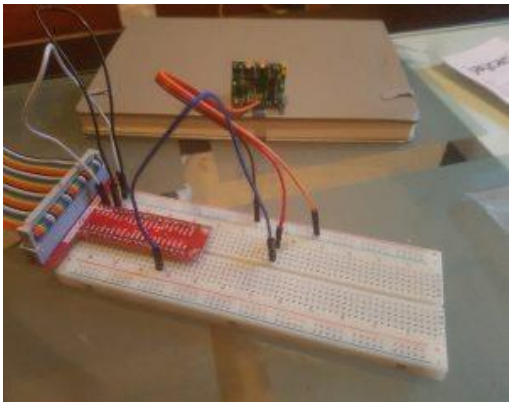
Python is fussy about indentation, so be sure to copy the spaces too. Then press Enter again to run the loop.

This loop will run until someone presses the button, so press it. It should produce a message saying the button works. This means you've successfully built a simple circuit that sends a message to your Raspberry Pi. If this doesn't work, check that everything is connected properly and try again.

4.13.6. Connecting and Testing the LED

The D in LED stands for "diode", which means electricity only runs in one direction through it.

You will notice that one leg of the LED is slightly longer: this connects to positive. Here, that means connecting to the GPIO pin. I'm using pin 26, for no particular reason. Place the LED in the breadboard, making sure that the legs are spaced horizontally so that you aren't shorting the connection out. Now connect this positive leg to your GPIO pin. An LED should be wired in series with a resistor, so attach one end of your resistor to the short leg of the LED, and the other end to the negative power rail. Resistors can go in either way around.



Now let's go tell the Raspberry Pi what's going on. Type: Image: Pi & Breadboard

```
from gpiozero import LED
led = LED(26)
```

If everything's wired correctly, you can switch it on and off with these commands:

```
led.on()
led.off()
```

Controlling the LED With the Button

Now that everything's connected and you've checked they work, type:

```
button.when_pressed = led.on
```

Now press the button. The LED should switch on and stay on. Now type:

button.when_released = led.off
Press the button again; the LED should switch off when the button is released.

4.14. Connecting IOT devices to the cloud

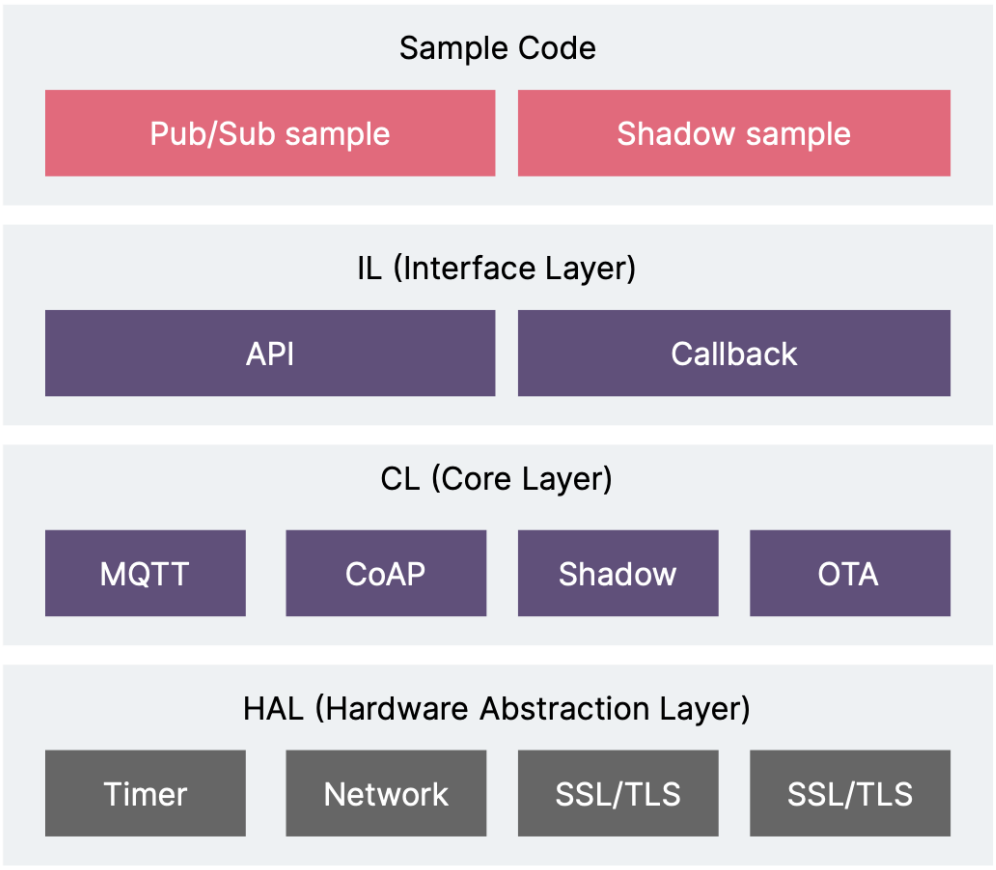
With the development of smart things, sensors and telecommunication, IoT (Internet of Things) technology has greatly developed and become more and more standardized. But fragmented device-side communication connection problems often impede the project implementation process.

There are four best practices to connect different types of devices to the cloud:

- Directly integrate IoT SDK (Software Development Kit) for resource-rich devices
- Rely on a communication module for resource-constrained devices
- Use a local gateway for non-network devices
- Use a cloud gateway for private-protocol devices with network capabilities

First, we should introduce the IoT Device SDK.

IoT Device SDK is used to help us quickly connect hardware devices to the IoT platform. We can download the IoT Device SDK from the corresponding cloud platform, e.g. AWS IoT Device SDK.



There are 4 layers of the IoT SDK. From bottom to top:

1. The HAL (hardware abstraction layer) abstracts the support function interface of different OS (operating systems) to the SDK. This enables the SDK to be ported to different hardware environments, different OS, and even bare chip environments.

2. The core layer completes the function encapsulation of MQTT/CoAP communication based on the HAL layer interface, including MQTT connection establishment, message sending and receiving; CoAP connection establishment, message sending and receiving; shadow device operation; OTA firmware status query, download and upgrade.

3. Interface layer, providing API and callback function definitions, isolating the core layer and the application.

4. Provide sample programs so that developers can quickly learn how to use the SDK.

   When developing applications on a device, we can always choose higher-level SDKs such as Android IoT SDK on an Android device. That's because the hardware environment porting work has already been done by the SDK itself.

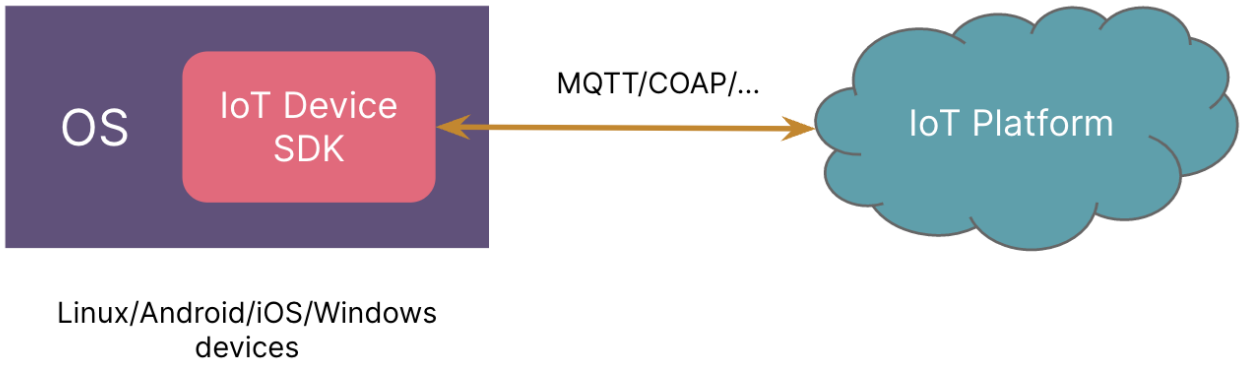
   However, when developing applications for an MCU (Micro Controller Unit) which has Linux or RTOS (Real Time Operating System), we should choose Embedded C SDK and port the code to a specific hardware environment.

   With the knowledge of IoT SDK, we can discuss how to connect an IoT device to the cloud.

Directly integrate IoT SDK for resource-rich devices

With the development of high-performance hardware, many smart devices have complete OS such as Linux, Android etc. These devices also have a Wi-Fi or cellular network.

At the operating system level, network communication problems have already been resolved. We only need to develop applications which integrate the IoT SDK of the cloud platform and the communication link with the cloud will have been established.



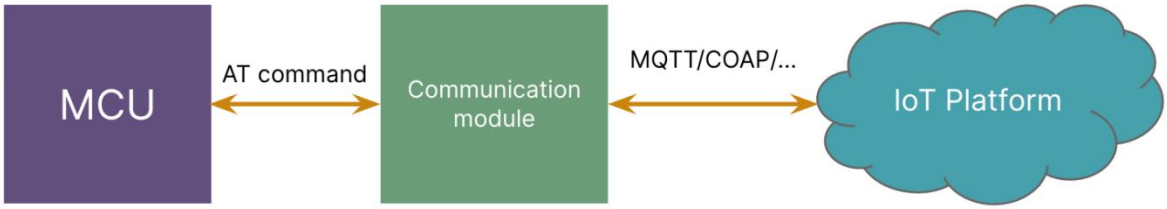
Examples of smart devices include smart phones, tablets, smart wearables, smart POS, computers, industrial gateways and development boards like the Raspberry Pi and ESP32.

Rely on a communication module for resource-constrained devices

In the IoT scenario, a large number of devices are resource-constrained, with RTOS, or even without an operating system, using MCU + communication modules to establish their link to the cloud.

There are many suppliers of cellular modules (NB-IoT/2G/3G/4G) on the market. The AT commands of each company are different, which makes developing device-side applications very difficult.

When we need to connect MCU to an IoT Platform, we should always carefully select the cellular modules and check whether they are suitable for a specific IoT platform.

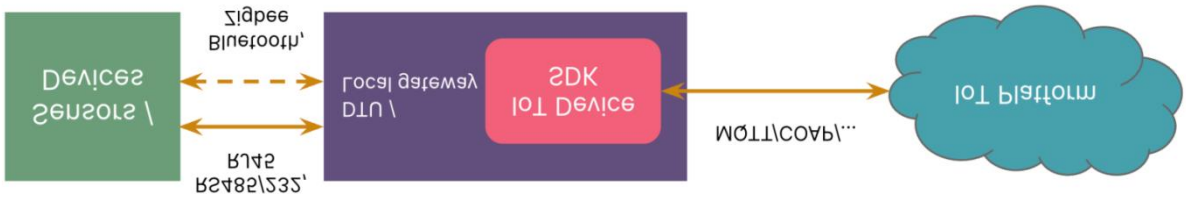


Difference between DTU and industrial gateway:

DTU is a wireless terminal device used to convert serial data into IP data or IP data into serial data and transmit it through a wireless communication network. It has fast and flexible networking, a short construction period and low cost.

The industrial gateway has the functions of collecting data from field devices through serial port or network port. Data collection, protocol analysis, data standardization and uploading to the IoT platform through edge computing functions. Which is more flexible, powerful and customizable than DTU, but it is more expensive and needs more resources to maintain.

Examples of sensors / devices which can be connected to DTU or industrial gateway include sensors, industrial equipment PLCs (Programmable Logic Controller), Bluetooth bracelets



Use a cloud gateway for private-protocol devices with network capabilities

Devices that directly connect to the cloud gateway

For some devices, they already have the ability to connect to the internet, but the protocols vary according to device manufacturers. We don't want the IoT platform layer to handle the parsing of these protocols directly;an intermediate layer should satisfy the protocol conversion work to make the data meet the unified format of the IoT platform.

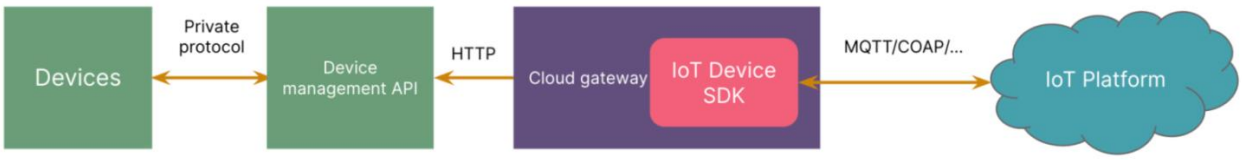
This intermediate layer is the cloud gateway. The cloud gateway is at the front of the platform. It receives data from the device side, completes message parsing, and then sends a message to the IoT platform with IoT device SDK.

Examples of devices that connect to the internet through private protocol include vehicle GPS.

Devices that connect to the cloud gateway indirectly

Some device vendors already provide a mature system which manages the devices and provides API. In this case, we can have another form of cloud gateway: Cloud to cloud connection. Making full use of the existing system will make the overall system more stable and give clear rights and responsibilities.

Using a mature system will bring us higher development efficiency, but it'll also introduce another midware, which will increase communication time.



Examples of devices with mature systems include cameras/NVR systems.