

# UNIT-I

## Embedded System

Embedded system एक dedicated computer सिस्टम है जिसे सिर्फ एक विशिष्ट task (कार्य) को करने के लिए डिज़ाइन किया गया होता है। एक एम्बेडेड सिस्टम सॉफ्टवेयर तथा हार्डवेयर का एक combination (संयोजन) होता है।

जैसे ही इसके नाम से पता लग रहा है, embedded का अर्थ है कि कोई एक चीज किसी दूसरी चीज से जुड़ी हुई है। तो इस प्रकार हम कह सकते हैं कि embedded system एक कंप्यूटर हार्डवेयर सिस्टम है जिसमें software जुड़ा हुआ है।

एम्बेडेड सिस्टम एक independent system हो सकता है या फिर यह एक बहुत बड़े system का हिस्सा हो सकता है।

एक एम्बेडेड सिस्टम microcontroller या [microprocessor](#) पर आधारित सिस्टम होता है और इसे किसी विशेष task (कार्य) को करने के लिए design किया जाता है। **उदाहरण के लिए:-** एक fire alarm system एक एम्बेडेड सिस्टम है जो केवल smoke (धुँ) को ही sense कर सकता है।

एक embedded system के तीन components होते हैं:-

- 1. Hardwar:-** एम्बेडेड सिस्टम का हार्डवेयर माइक्रोकंट्रोलर पर आधारित होता है। यह एक छोटी चिप होती है, जो ठीक (Central Processing Unit) की तरह काम करती है। इसमें इनपुट तथा आउटपुट Devices, मेमोरी, प्रोसेसर इत्यादि सम्मिलित होते हैं।
- 2. Software Application:-** सॉफ्टवेयर एप्लीकेशन एम्बेडेड सिस्टम का एक अहम हिस्सा है, इसे Firmware भी कहा जाता है। एम्बेडेड सॉफ्टवेयर एप्लीकेशन को Non-Pc Devices में किसी खास Function को Perform करने के लिए Write किया जाता है।
- 3. RTOS:-** RTOS का Full Form है (Real Time Operating System) यह एम्बेडेड सिस्टम का एक महत्वपूर्ण हिस्सा है, जिसका कार्य एम्बेडेड सिस्टम Working की देख-रेख करना है। यह हार्डवेयर तथा सॉफ्टवेयर एप्लीकेशन के बीच Interface का काम करता है। RTOS एक प्रकार का ऑपरेटिंग सिस्टम है, जो टास्क की High Priority के अनुसार उनका Execution करता है। यह सेंट्रल प्रोसेसिंग यूनिट की टाइमिंग को Efficiently मैनेज करता है, और इसके द्वारा Multiple Tasks को Concurrently रन किया जा सकता है।

# Characteristics of Embedded System – एम्बेडेड सिस्टम की विशेषताएं

**1:- Single-functioned (केवल एक कार्य)** - यह सिस्टम आमतौर पर एक ही विशेष कार्य करता है और उस कार्य को बार-बार करता है। **उदाहरण के लिए:-** एक pager हमेशा पेजर की तरह कार्य करेगा.

**2:- Reactive और Real time**— बहुत सारें एम्बेडेड सिस्टम को सिस्टम के environment में changes (बदलाव) होने पर लगातार react करना चाहिए और किसी भी देरी के बिना real time में results को compute करना चाहिए।

उदाहरण के लिए हम car cruise controller को लेते हैं; यह लगातार speed और break sensor की निगरानी करता है और react (प्रतिक्रिया) करता है। इसे सीमित समय के भीतर बार-बार acceleration या de-acceleration को compute करना चाहिए। अगर compute करने में देरी हो जाती है तो car का control फेल हो सकता है.

**3:- Microprocessors based (माइक्रोप्रोसेसर पर आधारित)** - ये माइक्रोप्रोसेसर या माइक्रोकंट्रोलर पर आधारित होने चाहिए।

**4:- Memory** – इसमें एक मेमोरी होनी चाहिए, क्योंकि इसका सॉफ्टवेयर आमतौर पर [ROM](#) में embed रहता है। इसे कंप्यूटर में किसी भी secondary memory की जरूरत नहीं होती है।

**5:- Connected (जुड़ना)** - इसमें इनपुट और आउटपुट डिवाइस को कनेक्ट करने के लिए connected [peripheral](#) होना चाहिए।

**6:- हार्डवेयर-सॉफ्टवेर सिस्टम** - सॉफ्टवेयर का उपयोग अधिक features और flexibility के लिए किया जाता है तथा हार्डवेयर का उपयोग performance और security के लिए किया जाता है।

**7:-** इनके पास बहुत ही कम या कोई user interface (UI) नहीं होता है। एक पूरी तरह से automatic वाशिंग मशीन अपने आप ही काम करती है जब उसे set कर दिया जाता है. और कार्य समाप्त होने के बाद अपने आप बंद हो जाती है।

**8:-** ज्यादातर embedded system छोटे आकार के होते हैं, कम power के साथ काम कर सकते हैं और बहुत महंगे नहीं होते हैं।

**9:-** एंबेडेड सिस्टम को users के द्वारा change या upgrade नहीं किया जा सकता है। इसलिए ये reliable और stable होने चाहिए. और ये बिना किसी कठिनाई के लंबे समय तक कार्य करते रहने चाहिए। जिससे कि users को कोई मुश्किल ना आये.

### **advantage of embedded** एम्बेडेड सिस्टम के फायदे

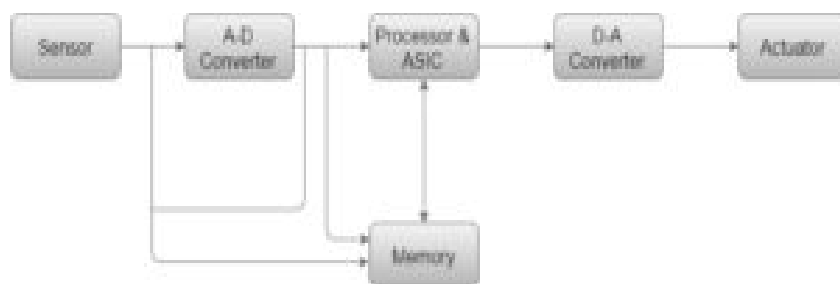
1. इसे आसानी से customize किया जा सकता है.
2. यह बहुत ही कम power को consume करता है.
3. इसका मूल्य (cost) बहुत ही कम होता है.
4. इसकी performance बहुत ही अच्छी होती है.
5. ये systems बहुत ही ज्यादा stable तथा reliable होते हैं.
6. एम्बेडेड सिस्टम size में बहुत ही छोटे होते हैं, इसलिए इन्हें कहीं भी ले जाया और लोड किया जा सकता है।
7. ये बहुत ही fast होते हैं.
8. ये product की quality को बेहतर बनाते हैं.

### **एम्बेडेड सिस्टम के नुकसान**

1. एक बार इन्हें configure करने के बाद बदला नहीं जा सकता है और इन्हें users खुद से upgrade या change नहीं कर सकते.
2. इन्हें maintain करना बहुत ही मुश्किल होता है और इन systems की files का backup लेना भी कठिन होता है.
3. इन systems के लिए troubleshooting बहुत difficult है. एक system से दूसरे system में data को ट्रान्सफर करना भी कठिन कार्य है.
4. चूँकि ये केवल एक विशेष कार्य के लिए design किये जाते हैं इसलिए इनका हार्डवेयर सिमित होता है.

### **structure of embedded system** – एम्बेडेड सिस्टम की संरचना

नीचे चित्र में इसका बेसिक structure दिया गया है:-



**Sensor (सेंसर):-** यह physical quantity को मापता है और इसे electrical signal में बदल देता है. इस electrical signal को observer के द्वारा या electrical यंत्र जैसे:- A2D Converter के द्वारा read किया जाता है. एक sensor मापी गयी quantity को memory में स्टोर करता है.

**A-D Converter:-** एक [analog to digital converter](#) सेंसर के द्वारा भेजी गयी analog signal को digital signal में बदल देता है.

**Processor & ASICs:-** प्रोसेसर, output को मापने के लिए data को प्रोसेस करता है तथा इसे memory में स्टोर करता है.

**D-A Converter:-** एक [digital to analog converter](#) डिजिटल डेटा को एनालॉग डेटा में परिवर्तित करता है।

**Actuator:-** एक Actuator जो है वह D-A कन्वर्टर द्वारा दिए गए आउटपुट की actual (वास्तविक) आउटपुट से तुलना करता है।

## **applications of embedded system– एम्बेडेड के अनुप्रयोग**

इनका प्रयोग निम्नलिखित real life जगहों पर किया जाता है:-

**Consumer electronics** – टेलीविजन, डिजिटल कैमरा, computer printers, विडियो गेम कंसोल, तथा PS4 आदि में इनका use किया जाता है.

**Household appliance में** - रेफ्रिजरेटर; वॉशिंग मशीन, माइक्रोवेव ओवन, एयर कंडीशनर आदि में.

**Medical Equipment में** - scanners जैसे:- MRI, CT स्कैन के लिए; ECG मशीन- रक्तचाप और दिल की धड़कन की निगरानी के लिए उपकरण।

**ऑटोमोबाइल में-** ईंधन इंजेक्शन प्रणाली, एंटी-लॉक ब्रेकिंग सिस्टम, संगीत और मनोरंजन प्रणाली, एयर-कंडीशनर को नियंत्रित करने आदि में इनका प्रयोग किया जाता है.

**industry में** - असेंबली लाइन्स, फीडबैक के लिए, डेटा कलेक्शन के लिए इन सिस्टम का use किया जाता है।

**Aerospace में:-** navigation, GPS आदि में.

**communication में:-** [routers](#), satellite में इनका प्रयोग होता है.

## Operating System in Embedded system:

एम्बेडेड ऑपरेटिंग सिस्टम एक विशेष प्रकार का ऑपरेटिंग सिस्टम (OS) है जिसे किसी ऐसे डिवाइस, जो कंप्यूटर नहीं है के लिए एक विशिष्ट कार्य करने के लिए डिज़ाइन किया गया है। एक एम्बेडेड OS का मुख्य काम उस कोड को चलाना है जो डिवाइस को अपना काम करने की अनुमति देता है। एम्बेडेड OS डिवाइस के हार्डवेयर को OS में चल रहे सॉफ्टवेयर के लिए भी सुलभ बनाता है।

एक एम्बेडेड ओएस अक्सर एक एम्बेडेड सिस्टम के भीतर काम करता है। एक एम्बेडेड सिस्टम एक कंप्यूटर है जो एक मशीन का समर्थन करता है। यह बड़ी मशीनों जैसे की कारों में कंप्यूटर सिस्टम, ट्रैफिक लाइट, डिजिटल टीवी, एटीएम, हवाई जहाज नियंत्रण, बिक्री के बिंदु (पीओएस) टर्मिनल, डिजिटल कैमरा, जीपीएस नेविगेशन सिस्टम, लिफ्ट और स्मार्ट मीटर आदि में एक विशेष काम करता है।

एम्बेडेड सिस्टम वाले उपकरणों के नेटवर्क से मिलकर इंटरनेट ऑफ थिंग्स (IoT) बनाता है। एम्बेडेड सिस्टम IoT उपकरणों के अंदर बुनियादी संचालन करते हैं, जैसे कि मानव संपर्क के बिना नेटवर्क पर डेटा स्थानांतरित करना।

एक एम्बेडेड ओएस एक एम्बेडेड डिवाइस को एक बड़े सिस्टम के भीतर अपना काम करने में सक्षम बनाता है। यह एक विशिष्ट कार्य करने के लिए एम्बेडेड सिस्टम के हार्डवेयर के साथ संचार करता है। उदाहरण के लिए, एक लिफ्ट में एक एम्बेडेड सिस्टम हो सकता है, जैसे कि माइक्रोप्रोसेसर या माइक्रोकंट्रोलर, जो यह समझने देता है कि यात्री कौन से बटन दबा रहा है। उस सिस्टम पर चलने वाला एम्बेडेड सॉफ्टवेयर एम्बेडेड ओएस है।

## Types of embedded OS एम्बेडेड OS के प्रकार:

**Multitasking operating system :** एक मल्टीटास्किंग ओएस एक साथ कई काम कर सकता है। यह बुनियादी कार्यों को करने के लिए जॉब शेड्यूलिंग का उपयोग करता है। उदाहरण के लिए, एक सेलफोन ओएस कई कार्यों के बीच सीपीयू संसाधनों को विभाजित करता है।

**Real-time operating system :** एक रीयल-टाइम OS को प्रतिक्रियाशील होने के लिए डिज़ाइन किया गया है। यह इनपुट प्राप्त होने पर संसाधित करता है और

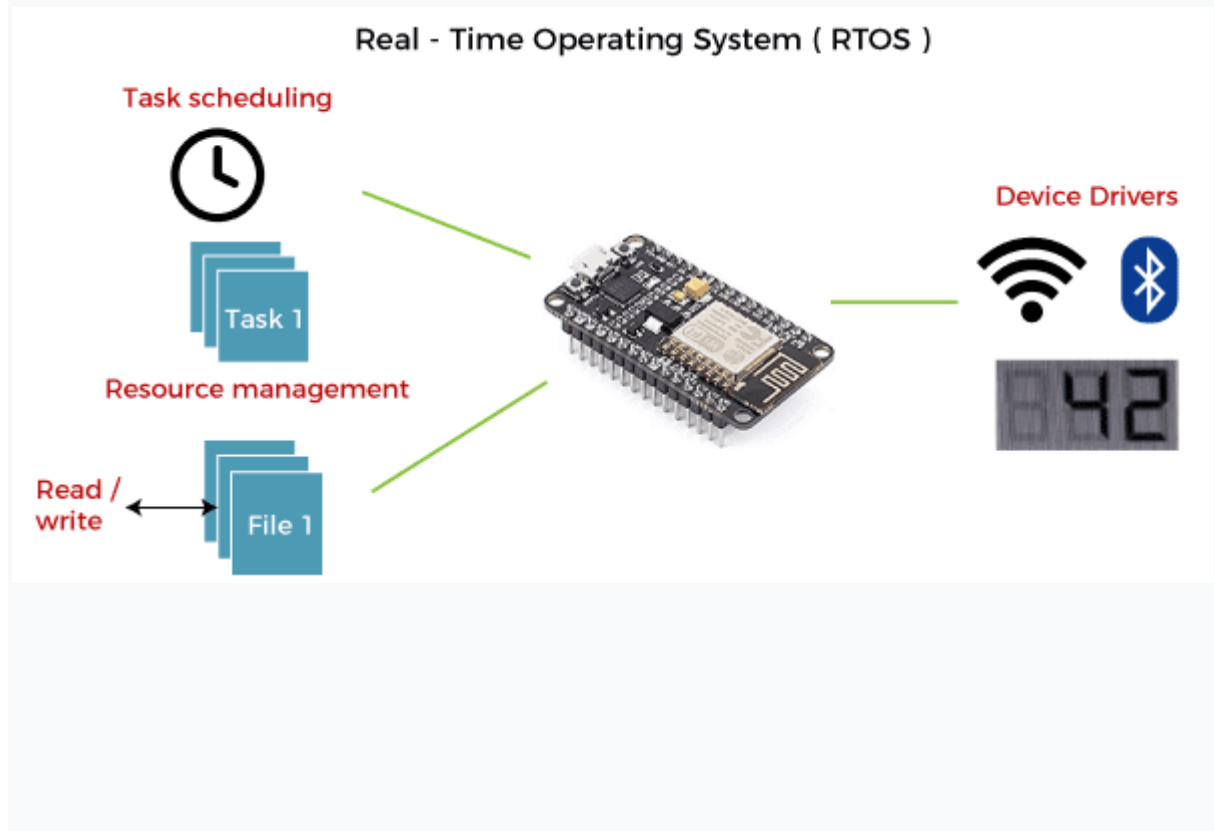
एक विशिष्ट समय सीमा के भीतर प्रतिक्रिया करता है। यदि प्रतिक्रिया समय निर्दिष्ट समय अवधि के बाहर आता है, तो सिस्टम विफल हो सकता है। रीयल-टाइम OS कभी-कभी रेट मोनोटोनिक शेड्यूलिंग का उपयोग करते हैं, जो कार्यों को प्राथमिकता देता है।

**Single loop control system:** इस प्रकार का एम्बेडेड OS एकल चर पर नियंत्रण रखता है। उदाहरण के लिए स्मार्ट घर में तापमान नियंत्रण करना। एक स्मार्ट थर्मोस्टेट घर में तापमान को मापता है और यदि यह उपयोगकर्ता द्वारा निर्धारित सीमा से अधिक हो जाता है, तो गर्मी बंद कर देता है।

**Characteristics of Real Time Operating System:** एक रीयल-टाइम ऑपरेटिंग सिस्टम (RTOS) कंप्यूटर में उपयोग किया जाने वाला एक विशेष-उद्देश्य वाला ऑपरेटिंग सिस्टम है जिसमें किसी भी कार्य को करने के लिए समय सीमा निर्धारित रहता है। यह ज्यादातर उन प्रणालियों में नियोजित होता है जिनमें गणना के परिणामों का उपयोग किसी प्रक्रिया को क्रियान्वित करते समय प्रभावित करने के लिए किया जाता है। जब भी कंप्यूटर के बाहर कोई घटना होती है, तो घटना की निगरानी के लिए उपयोग किए जाने वाले कुछ सेंसर की मदद से कंप्यूटर को इसकी सूचना दी जाती है। सेंसर द्वारा भेजा गया सिग्नल ऑपरेटिंग सिस्टम में इंटरप्ट की तरह काम करता है। एक इंटरप्ट प्राप्त होने पर, ऑपरेटिंग सिस्टम इंटरप्ट की सेवा के लिए एक विशिष्ट प्रक्रिया या प्रक्रियाओं के एक सेट को आमंत्रित करता है।

यह प्रक्रिया पूरी तरह से अबाधित है जब तक कि इसके निष्पादन के दौरान उच्च प्राथमिकता वाली बाधा उत्पन्न न हो। इसलिए, व्यवधानों (इंटरप्ट) के बीच प्राथमिकता का एक सख्त पदानुक्रम होना चाहिए। उच्चतम प्राथमिकता वाले इंटरप्ट को प्रक्रिया शुरू करने की अनुमति दी जानी चाहिए, जबकि कम प्राथमिकता वाले इंटरप्ट को एक बफर में रखा जाना चाहिए जिसे बाद में संभाला जाएगा। ऐसे ऑपरेटिंग सिस्टम में इंटरप्ट मैनेजमेंट महत्वपूर्ण है।

रीयल-टाइम ऑपरेटिंग सिस्टम विशेष-उद्देश्य वाले ऑपरेटिंग सिस्टम को नियोजित करते हैं क्योंकि पारंपरिक ऑपरेटिंग सिस्टम ऐसा प्रदर्शन प्रदान नहीं करते हैं।



# UNIT- II

## Microcontroller

Microcontrollers may be called computers on-chip. A combination of a controller, internal ROM, RAM, parallel and serial ports is a Microcontroller. Microcontrollers are dedicated devices embedded within an application. Ex:- as an engine controller in automobiles, as an exposure and focus controller in cameras. On-chip peripheral is selected depending on the specifics of the target application.

As we know that the Microcontrollers are powerful digital processors, the degree of control and programmability in which they provide significantly enhances the effectiveness of the application.

## Types of Microcontroller

(1) 8051 (2) PIC (3) AVR (4) ARM

### 8051 Microcontroller

Intel designed the first microcontroller and this is known as the 8051 microcontrollers. This microcontroller was introduced in the late 1970 and in 1981 it was developed by Intel.

8051 microcontroller is an 8-bit microcontroller. 8051 microcontroller has the ability to read, write and process 8-bit data. It is widely used in embedded systems, consumer electronics, robotics, etc. The peripherals of this microcontroller are integrated into a single chip, and the overall system cost is very low.

The size of the product is small as compared to the microcontroller-based system thus very handy. All features are available in 40 pin IC of the 8051 microcontrollers.

#### Applications of 8051 microcontrollers:

- (1) Home appliances like Microwave Oven, Washing machines, etc.
- (2) Light sensing and controlling devices like audio systems.
- (3) Temperature sensing and control devices.
- (4) Fire detection and safety devices such as home security systems.
- (5) Other devices like calculator, ATM machine, etc.

## PIC Microcontroller

PIC stands for Programmable Interface Controllers. It was developed by a Microchip average. It is fast and simple to implement program when we compare with other microcontrollers like 8051. The ease of programming and simple to interfacing with other peripherals PIC become successful microcontroller.



These are electronic circuits that can be programmed to carry out a vast range of tasks. It consists of memory structure, input/output ports, timers, A/D Converter, Oscillators, CCP module. PIC Microcontrollers are relatively cheap. PIC devices are familiar with both industrial developers and hobbyists due to their low cost, wide availability, large user base, extensive collection of application notes, serial programming, free development tools, and reprogrammable flash memory capability.



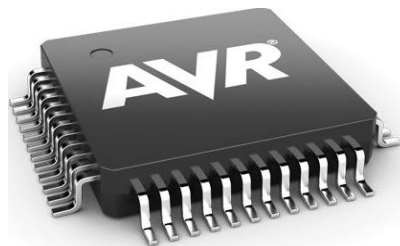
- (1) **Features of PIC Microcontroller**
- (2) Flash memory
- (3) Watchdog Timer
- (4) Sleep mode
- (5) EEPROM memory

#### **Applications of PIC Microcontroller: -**

- (1) Mobile Phones
- (2) Computer control systems
- (3) Alarm systems
- (4) Embedded systems

## **AVR Microcontroller**

AVR microcontroller was developed in the year of 1996 by Atmel Corporation. The structural design of AVR was developed by the Alf-Egil Bogen and Vegard Wollan. It is commonly accepted that AVR stands for Alf and Vegard's RISC microcontroller, which is also known as Advanced Virtual RISC. The AT90S8515 was the initial microcontroller which was based on the AVR architecture, though the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.



This microcontroller is the advanced version of a microcontroller. It contains a chip CPU, ROM, RAM, input/output unit, interrupts controller, etc. This microcontroller is used for high-speed signal processing operation which is connected inside an embedded system.

AVR Microcontrollers are Available in three categories: -

- 1) **TinyAVR:-** Less memory, small size, appropriate just for simpler applications
- 2) **MegaAVR:-** These are the mainly popular ones having a good quantity of memory (up to 256 KB), higher number of inbuilt peripherals and appropriate for modest to complex applications.
- 3) **XmegaAVR:-** Used in commercial for complex applications, which need large program memory and high speed.

#### **Features of AVR Microcontroller**

- 1) 32×8 general working function registers.
- 2) 32k bytes of in-system self-programmable flash program memory.
- 3) 2k bytes of internal SRAM.
- 4) 1024bytes of EEPROM.
- 5) Available in 40 pin DIP, 44lead QTFP, 44-pad QFN/MLF.
- 6) 32 programmable I/O lines.
- 7) 8 channel, 10bit ADC.
- 8) Harvard architecture.
- 9) UART, I2C, SPI protocol support

## Applications of AVR Microcontroller: -

- (1) AVR microcontroller is mainly used in an embedded system for the operation of high-speed signal processing.
- (2) These microcontrollers are used in touch screens, home automation, medical devices, defence, automobiles, etc.
- (3) This microcontroller can be used in many types of projects like data acquisition, motion control, For signal sensing, interface GPS, GSM, motors, displays on LCD, unmanned aerial vehicles development, etc.

## ARM Processor

Advanced RISC Machine (ARM) this microcontroller was introduced by Acron Computer Organization. It is manufactured by Apple, Qualcomm, Motorola, etc. The Processor of the ARM microcontroller belongs to the family of CPUs which are based on Reduced Instruction Set Computer (RISC) and also ARM Microprocessor. An ARM makes at 32-bit and 64-bit RISC multi-core processors. The speed of the ARM microcontroller is 1 clock cycle per machine cycle and the power consumption is low. The popular microcontroller of ARM includes ARM Cortex-M0 to ARM Cortex-M7, etc.



..

### Features of ARM Microcontroller:-

- 1) It must have the ability to control different types of software.
- 2) It is compatible with the sleep mode of operation.
- 3) Flash, EEPROM, SDRAM memory is used in ARM microcontroller.
- 4) ARM consists of an Arithmetic logic unit, booth multiplier, barrel shifter, control unit, register file.
- 5) It consists of a three-stage pipeline.

#### Three-stage pipeline

**S1 Fetch:** - The instruction is fetched from memory and placed in the instruction pipeline.

**S2. Decode:-** The instruction is decoded and the datapath control signals are prepared for the next cycle. In this stage, the instruction owns the decode logic but not the datapath.

**S3. Execute:-** The instruction owns the datapath; the register bank is read, an operand shifted, the ALU register generated and written back into a destination register.

### Applications Of ARM Microcontroller: -

- 1) ARM microcontroller is used in space and aerospace.
- 2) Used in medical devices such as MRI Machines, ultrasound machines.
- 3) Used in accelerators, nuclear reactors, and X-ray machines.
- 4) ARM processors are widely used in customer electronic devices such as smart phones, tablets, multimedia players and other mobile devices

## **Main Difference between features of AVR, ARM, 8051 and PIC Microcontrollers: -**

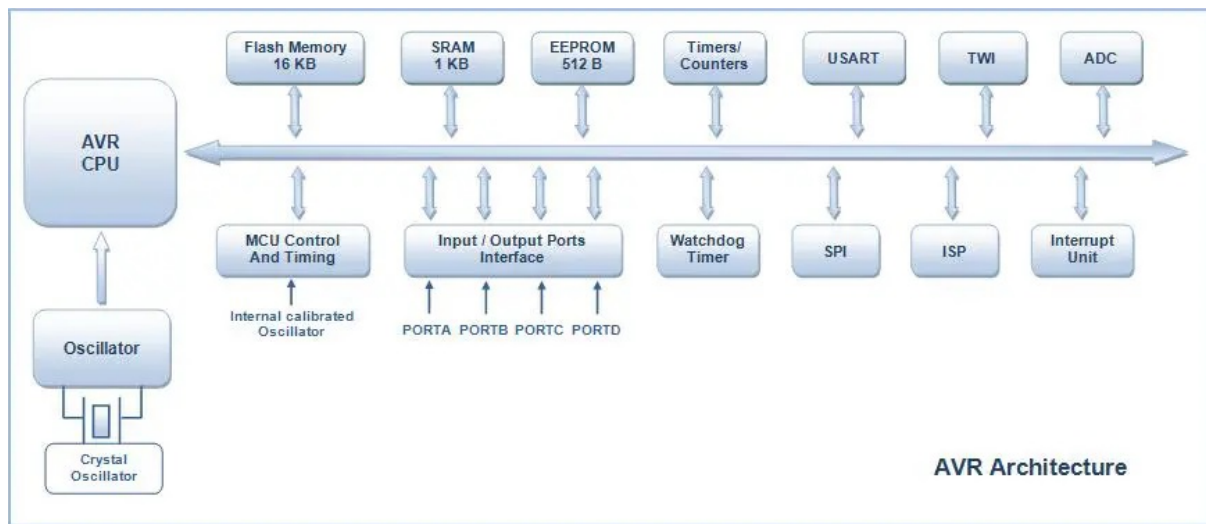
S.No.	Features	AVR	ARM	8051	PIC
1	Manufacturer	Atmel	Apple, Nvidia, Qualcomm, Samsung	NXP, Atmel, Silicon Labs, Dallas, Cypress, Infineon, etc.	Microchip Average
2	Bus Width	8/32-bit	32-bit mostly also available in 64-bit	8-bit for standard core	8/16/32-bit
3	Speed	1 clock/ instruction cycle	1 clock/ instruction cycle	12 Clock/instruction cycle	4 Clock/instruction cycle
4	Memory Architecture	Modified Harvard architecture	Modified Harvard architecture	Harvard architecture	Von Neumann architecture
5	Memory	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM	ROM, SRAM, FLASH	SRAM, FLASH
6	Instruction Set Architecture (ISA)	RISC	RISC	CISC	Some feature of RISC
7	Community	Very Good	Vast	Vast	Very Good
8	Power Consumption	Low	Low	Average	Low
9	Families	Tiny, Atmega, Xmega, special purpose AVR	ARMv4,5,6, 7 and series	8051 variants	PIC16, PIC17, PIC18, PIC24, PIC32
10	Popular Microcontrollers	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc	AT89C51, P89v51, etc.	PIC18fXX8, PIC16f88X, PIC32MXX

### **AVR Microcontroller Architecture**

The architecture of AVR microcontrollers is based on the advanced RISC & it includes 32 x 8-bit general-purpose registers. In a single CLK cycle, this microcontroller can get inputs from two registers to connect them to ALU for the requested operation & move back the result to an arbitrary register. Here, the ALU performs arithmetic & logical operations on the inputs from the register.

AVR can execute single cycle execution which means this microcontroller can perform 1 million instructions for each second if the frequency of the cycle is 1MHz. If the operating frequency of the controller is higher, then its processing speed will be higher. So the power consumption needs to optimize with processing speed & thus need to choose the operating frequency accordingly.

The architecture of the AVR microcontroller includes different building blocks and each block is explained in the AVR microcontroller block diagram shown below.



### *I/O Ports*

AVR microcontroller includes four 8-bit input-output ports like PORT-A, PORT-B, PORT-C & PORT-D.

### *Internal Calibrated Oscillator*

AVR microcontroller includes an internal oscillator used for driving its CLK. This microcontroller is set to work at a 1 MHz internal calibrated oscillator. So the maximum internal oscillator frequency is 8 MHz

### *ADC Interface*

This microcontroller includes an 8-channel ADC with a 10-bits resolution. The main function of this ADC is to read the analog input.

### *Timers/Counters*

The microcontroller includes two 8-bit & one 16-bit timer/counter. The main function of timers in this controller is to generate precision actions like time delays created in between two operations.

### *Watchdog Timer*

In this microcontroller, the watchdog timer is present with an internal oscillator. The main function of this is to monitor and reset the controller continuously if the code gets trapped while executing in a defined time interval.

### *Interrupts*

This microcontroller includes 21 interrupts where 16 interrupts are internal and the remaining interrupts are external. Here internal interrupts support different peripherals like ADC, USART, Timers, etc.

### *USART*

The term USART stands for "Universal Synchronous and Asynchronous Receiver" & interface of the transmitter is obtainable to interface with an external device that is capable of communicating serially.

## *General Purpose Registers*

This microcontroller has 32 general-purpose registers where these registers are connected with the ALU of the CPU directly.

## *Memory*

The memory of this microcontroller includes three different sections

### *Flash EEPROM*

This type of memory is helpful in storing the program dumped by the user into the AVR microcontroller. This program can be simply removed electrically like a single unit. This memory is non-volatile which means if the power is gone then the program will not erase. This microcontroller includes 16KB of in-system programmable Flash EEPROM.

### *Byte Addressable EEPROM*

Byte addressable EEPROM is a non-volatile memory that is mainly used for data storage. This microcontroller includes EEPROM- 512 bytes, so this memory can be simply helpful in storing the lock code if we are designing an electronic door lock application.

### *SRAM*

SRAM stands for Static Random Access Memory which is the volatile memory of the AVR microcontroller so the data will be lost once power is deactivated. This microcontroller includes 1KB – of internal SRAM. A small part of SRAM is reserved for general purpose registers which are used by the CPU & also some other peripheral subsystems.

### *ISP*

These microcontrollers include In-System Programmable Flash Memory or ISP that can be simply programmed without detaching the chip from the circuit; this allows for reprogramming of the microcontroller when it is within the application circuit.

### *SPI*

The term SPI stands for Serial Peripheral Interface, which is mainly used for serial communication between two different devices on a common CLK source. The SPI data transmission speed is high as compared to USART.

### *TWI*

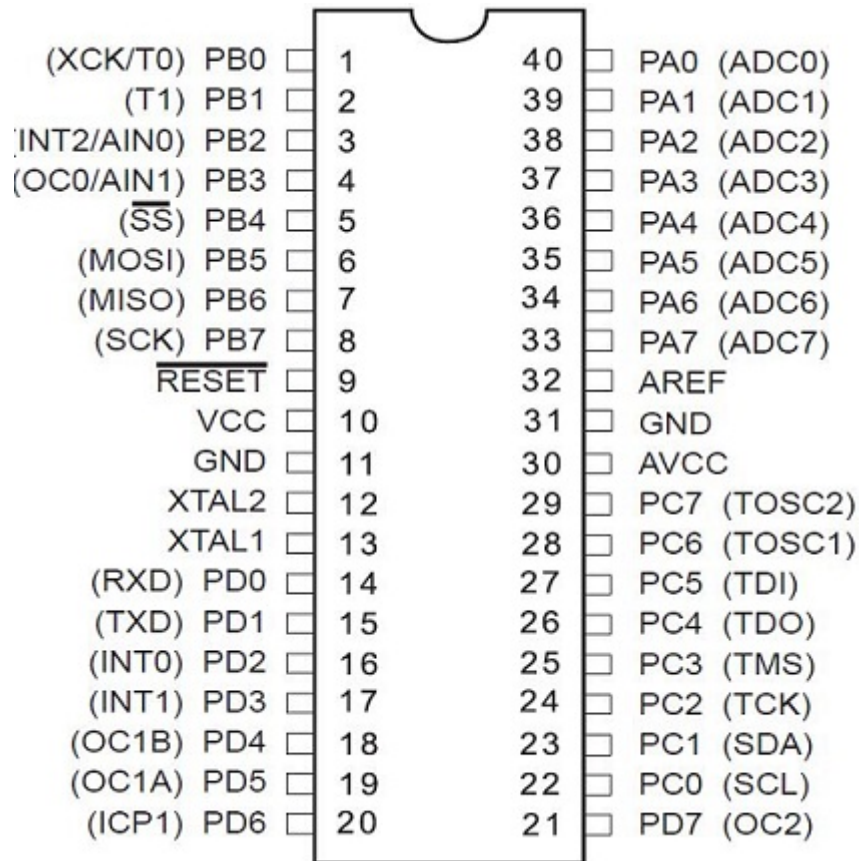
TWI is a Two-Wire Interface that can be used to connect a network for devices, so several devices can be simply connected above this interface to form this network so that the transmission of data can be done simultaneously by devices with their own unique address.

### *DAC*

The DAC or Digital to Analog Converter in the microcontroller is used to perform the reverse action of ADC. This converter is simply used whenever there is a requirement of changing a signal from digital to analog.

## AVR Microcontroller PinOut/PinDiagram

The AVR Atmega 32 microcontroller **pin configuration** is shown below. This microcontroller includes four ports port-A, port-B, port-C, and port-D. Port-A mainly includes pins from PA7to PA0, port-B includes PB7 to PB0, port-C includes from PC7 to PC0 and port-D includes from PD7 to PD0.



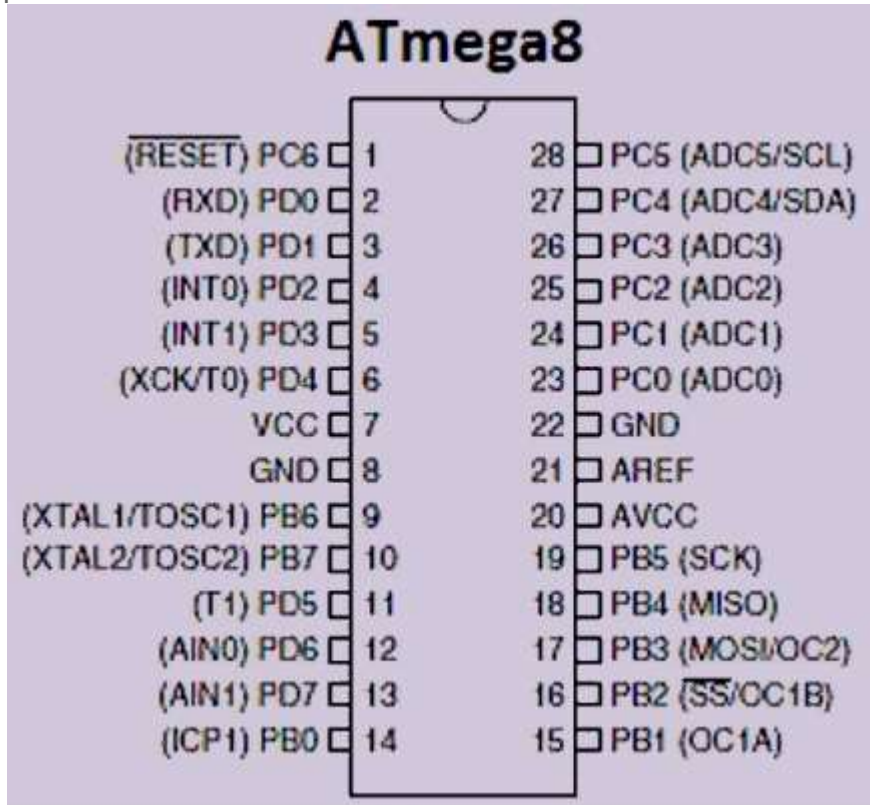
### What is an AVR Atmega8 Microcontroller?

In 1996, AVR Microcontroller was produced by the "Atmel Corporation". The Microcontroller includes the Harvard architecture that works rapidly with the RISC. The features of this Microcontroller include different features compared with other like sleep modes-6, inbuilt ADC (analog to digital converter), internal oscillator and serial data communication, performs the instructions in a single execution cycle. These Microcontrollers were very fast and they utilize low power to work in different power saving modes. There are different configurations of AVR microcontrollers are available to perform various operations like 8-bit, 16-bit, and 32-bit. AVR microcontrollers are available in three different categories such as TinyAVR, MegaAVR, and XmegaAVR

1. The Tiny AVR microcontroller is very small in size and used in many simple applications
2. Mega AVR microcontroller is very famous due to a large number of integrated components, good memory, and used in modern to multiple applications
3. The Xmega AVR microcontroller is applied in difficult applications, which require high speed and huge program memory.
- 4.

## Atmega8 Microcontroller Pin Description

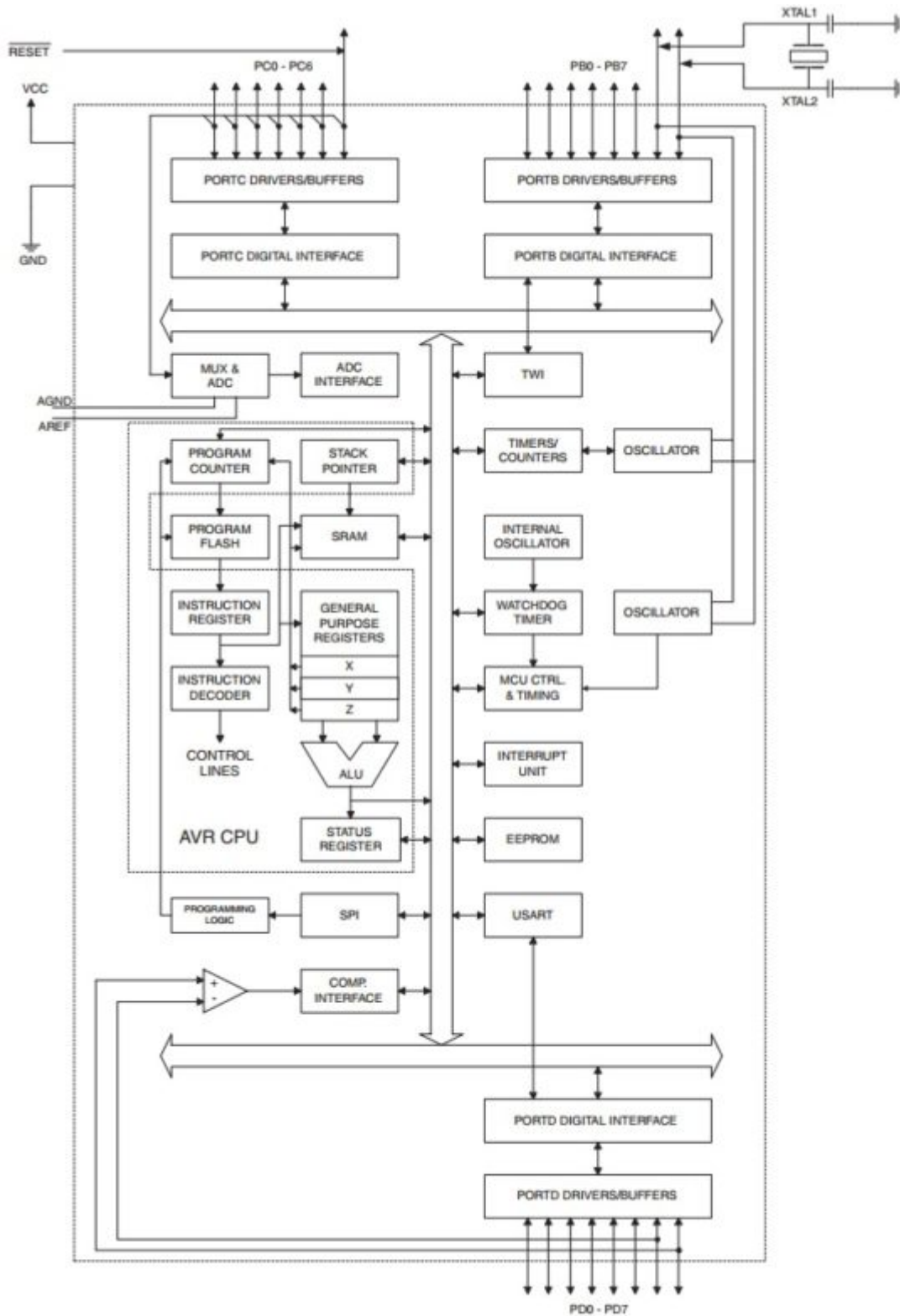
The main feature of Atmega8 Microcontroller is that all the pins of the Microcontroller support two signals except 5-pins. The Atmega8 microcontroller consists of 28 pins where pins 9,10,14,15,16,17,18,19 are used for port B, Pins 23,24,25,26,27,28 and 1 are used for port C and pins 2,3,4,5,6,11,12 are used for port D.



- Pin -1 is the RST (Reset) pin and applying a low-level signal for a time longer than the minimum pulse length will produce a RESET.
- Pin-2 and pin-3 are used in USART for serial communication.
- Pin-4 and pin-5 are used as an external interrupt. One of them will activate when an interrupt flag bit of the status register is set and the other will activate as long as the intrude condition succeeds.
- Pin-9 & pin-10 are used as a timer counters oscillators as well as an external oscillator where the crystal is associated directly with the two pins. Pin-10 is used for low-frequency crystal oscillator or crystal oscillator. If the internal adjusted RC oscillator is used as the CLK source & the asynchronous timer is allowed, these pins can be utilized as a timer oscillator pin.
- Pin-19 is used as a Master CLK o/p, slave CLK i/p for the SPI-channel.
- Pin-18 is used as Master CLK i/p, slave CLK o/p.
- Pin-17 is used as Master data o/p, slave data i/p for the SPI-channel. It is used as an i/p when empowered by a slave & is bidirectional when allowed by the master. This pin can also be utilized as an o/p compare with match o/p, which helps as an external o/p for the timer/counter.
- Pin-16 is used as a slave choice i/p. It can also be used as a timer or counter1 comparatively by arranging the PB2-pin as an o/p.
- Pin-15 can be used as an external o/p of the timer or counter compare match A.
- Pin-23 to Pins28 have used for ADC (digital value of analog input) channels. Pin-27 can also be used as a serial interface CLK & pin-28 can be used as a serial interface data
- Pin-12 and pin-13 are used as an Analog Comparator i/ps.
- Pin-6 and pin-11 are used as timer/counter sources.

# Atmega8 AVR Microcontroller Architecture

The Atmega AVR Microcontroller architecture includes the following blocks.



**Memory:** It has 1Kbyte Internal SRAM, 8 Kb of Flash program memory and 512 Bytes of EEPROM.

**I/O Ports:** It has three ports, namely port-B, port-C, and port-D and 23 I/O line can be attained from these ports.



**Interrupts:** The two Exterior Interrupt sources are located at port D. Nineteen dissimilar interrupts vectors supporting nineteen events produced by interior peripherals.

**Timer/Counter:** There are 3-Internal Timers are accessible, 8 bit-2, 16 bit-1, presenting numerous operating modes & supporting internal/external clocking.

**Serial Peripheral Interface (SPI):** ATmega8 microcontroller holds three integrated communication devices. One of them is an SPI, 4-pins are allocated to the Microcontroller to implement this system of communication.

**USART:** USART is one of the most powerful communication solutions. Microcontroller ATmega8 supports both synchronous & asynchronous data transmission schemes. It has three pins allocated for that. In many communication projects, the USART module is widely used for communication with PC-Microcontroller.

**Two-Wire Interface (TWI):** TWI is another communication device that is present in the ATmega8 microcontroller. It permits designers to set up a communication b/n two devices using two wires along with a mutual GND connection, As the o/p of the TWI is made using open collector o/ps, therefore external pull-up resistors are compulsory to make the circuit.

**Analog Comparator:** This module is incorporated in the integrated circuit that offers a contrast facility between two voltages linked to the two inputs of the comparator through External pins associated with the Microcontroller.

**ADC:** Inbuilt ADC (analog to digital converter) can alter an analog i/p signal into digital data of the 10-bit resolution. For a maximum of the low-end application, this much resolution is sufficient.

### ***Atmega8 Microcontroller Applications***

The Atmega8 microcontroller is used to build various electrical and electronic projects. Some of the AVR atmega8 Microcontroller projects are listed below.

1. AVR Microcontroller based LED Matrix Interfacing
2. UART communication between Arduino Uno and ATmega8
3. Interfacing of Optocoupler with ATmega8 Microcontroller
4. AVR Microcontroller based Fire Alarm System
5. Measurement of Light Intensity using AVR Microcontroller and LDR
6. AVR Microcontroller based 100mA Ammeter
7. ATmega8 Microcontroller based Anti-Theft Alarm System
8. AVR Microcontroller based Interfacing of Joystick
9. AVR Microcontroller based Interfacing of Flex Sensor
10. Stepper Motor Control using AVR Microcontroller

## **UNIT-III OPEN SOURCE EMBEDDED DEVELOPMENT**

### **BOARD(ARDUINO)**

#### **ARDUINO :-**

Arduino एक ओपन-सोर्स प्लेटफॉर्म है जिसका इस्तेमाल इलेक्ट्रॉनिक्स प्रोजेक्ट बनाने के लिए किया जाता है। Arduino में एक भौतिक प्रोग्रामेबल सर्किट बोर्ड (माइक्रोकंट्रोलर के रूप में) और एक सॉफ्टवेयर, या IDE (Integrated Development Environment) होता है, जो आपके कंप्यूटर पर चलता है, जिसका उपयोग भौतिक बोर्ड पर कंप्यूटर कोड लिखने और अपलोड करने के लिए किया जाता है।

Arduino प्लेटफॉर्म इलेक्ट्रॉनिक्स के क्षेत्र में लोगों के बीच अच्छे कारणों से काफी लोकप्रिय हो गया है। अधिकांश पिछले प्रोग्रामेबल सर्किट बोर्डों के विपरीत, बोर्ड पर नया कोड लोड करने के लिए Arduino को किसी दुसरे हार्डवेयर (प्रोग्रामर कहा जाता है) की आवश्यकता नहीं होती है - आप बस एक USB केबल का उपयोग कर सकते हैं। इसके अतिरिक्त, Arduino IDE C प्रोग्रामिंग के सरलीकृत संस्करण का उपयोग करता है, जिससे प्रोग्राम सीखना आसान हो जाता है। अंत में, Arduino एक मानक फॉर्म फैक्टर प्रदान करता है जो फंक्शन को तोड़ता है |

Arduino बटन, LED, मोटर, स्पीकर, GPS यूनिट, कैमरा, इंटरनेट और यहां तक कि आपके स्मार्ट-फोन या आपके टीवी के साथ इंटरैक्ट कर सकता है! यह लचीलापन इस तथ्य के साथ संयुक्त है कि Arduino सॉफ्टवेयर मुफ्त है, हार्डवेयर बोर्ड बहुत सस्ते हैं, और सॉफ्टवेयर और हार्डवेयर दोनों को सीखना आसान है, जिससे उपयोगकर्ताओं के एक बड़े समुदाय ने Arduino-आधारित प्रोजेक्ट के लिए कोड का योगदान दिया है और एक विशाल विविधता के लिए निर्देश जारी किए हैं।

Arduino वर्षों से रोज़मर्रा की वस्तुओं से लेकर जटिल वैज्ञानिक उपकरणों तक, हजारों परियोजनाओं का मस्तिष्क रहा है। निर्माताओं का एक विश्वव्यापी समुदाय - छात्र, शौकीन, कलाकार, प्रोग्रामर और प्रोफेशनल्स - इस ओपन-सोर्स प्लेटफॉर्म से जुड़े हुए हैं, उनके योगदान से इस प्लेटफॉर्म से संबंधित सुलभ ज्ञान अविश्वसनीय मात्रा में उपलब्ध है, जो नौसिखियों और विशेषज्ञों के लिए बहुत मददगार हो सकता है।

## **Why Arduino?**

इसके सरल और सुलभ उपयोगकर्ता अनुभव के कारण, Arduino का उपयोग हजारों विभिन्न परियोजनाओं और अनुप्रयोगों में किया गया है। Arduino सॉफ्टवेयर शुरुआती लोगों के लिए उपयोग में आसान है, एवं उन्नत उपयोगकर्ताओं के लिए भी काफी उपयोगी है। यह मैक, विंडोज और लिनक्स पर चलता है। शिक्षक और छात्र इसका उपयोग कम लागत वाले वैज्ञानिक उपकरण बनाने, रसायन विज्ञान और भौतिकी के सिद्धांतों को सिद्ध करने, या प्रोग्रामिंग और रोबोटिक्स के साथ आरंभ करने

के लिए करते हैं। डिजाइनर और आर्किटेक्ट इंटरैक्टिव प्रोटोटाइप बनाते हैं, संगीतकार और कलाकार इसे इंस्टॉलेशन के लिए और नए संगीत वाद्ययंत्रों के साथ प्रयोग करने के लिए उपयोग करते हैं। उदाहरण के लिए, मेकर्स, मेकर फेयर में प्रदर्शित कई परियोजनाओं के निर्माण के लिए इसका उपयोग करते हैं। नई चीजें सीखने के लिए Arduino एक महत्वपूर्ण टूल है। कोई भी - बच्चे, शौकीन, कलाकार, प्रोग्रामर - किट के चरण-दर-चरण निर्देशों का पालन करते हुए, या Arduino समुदाय के अन्य सदस्यों के साथ विचारों को ऑनलाइन साझा करना शुरू कर सकते हैं।

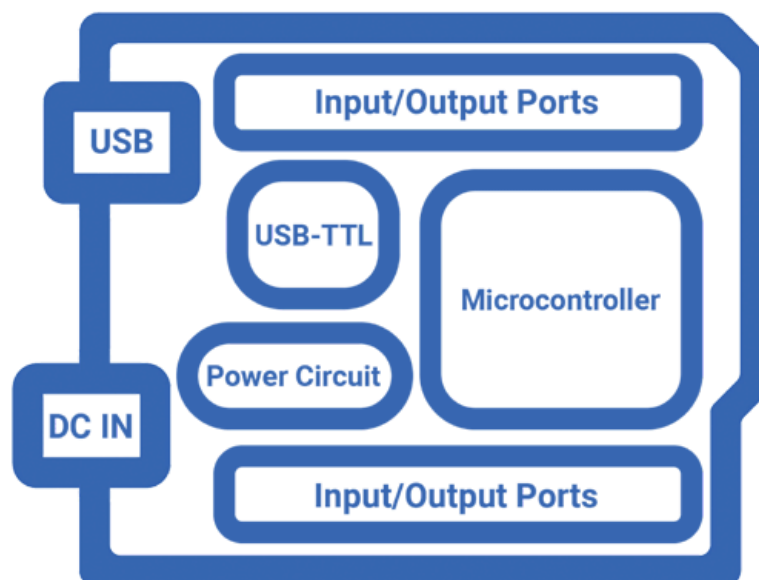
- सस्ता - Arduino बोर्ड अन्य माइक्रोकंट्रोलर प्लेटफॉर्म की तुलना में अपेक्षाकृत सस्ते हैं। Arduino मॉड्यूल को असेंबल करके सबसे कम खर्च में बनाया जा सकता है, और यहां तक कि पहले से असेंबलड Arduino मॉड्यूल की कीमत 600 रुपये से कम है।
- क्रॉस-प्लेटफॉर्म - Arduino सॉफ्टवेयर (IDE) Windows, Macintosh OSX और Linux ऑपरेटिंग सिस्टम पर चलता है। अधिकांश माइक्रोकंट्रोलर सिस्टम विंडोज तक ही सीमित हैं।
- सरल, स्पष्ट प्रोग्रामिंग वातावरण - Arduino Software (IDE) शुरुआती लोगों के लिए उपयोग में आसान है, फिर भी उन्नत उपयोगकर्ताओं के लिए भी इसका लाभ उठाने के लिए पर्याप्त लचीला है। शिक्षकों के लिए, यह प्रसंस्करण प्रोग्रामिंग वातावरण पर आसानी से आधारित है, इसलिए उस वातावरण में प्रोग्राम करना सीखने वाले छात्र Arduino IDE कैसे काम करते हैं, इससे परिचित होंगे।
- ओपन-सोर्स और एक्स्टेंसिबल सॉफ्टवेयर - Arduino सॉफ्टवेयर को ओपन-सोर्स टूल के रूप में प्रकाशित किया गया है, जो अनुभवी प्रोग्रामरों द्वारा विस्तार के लिए उपलब्ध है। भाषा को C++ लाइब्रेरीज़ के माध्यम से विस्तारित किया जा सकता है, और तकनीकी विवरण को समझने के इच्छुक लोग Arduino से AVR C प्रोग्रामिंग भाषा पर छलांग लगा सकते हैं, जिस पर यह आधारित है। इसी तरह, यदि आप चाहें तो सीधे अपने Arduino प्रोग्राम में AVR-C कोड जोड़ सकते हैं।
- ओपन सोर्स और एक्स्टेंसिबल हार्डवेयर - Arduino बोर्ड की योजनाएं क्रिएटिव कॉमन्स लाइसेंस के तहत प्रकाशित की जाती हैं, इसलिए अनुभवी सर्किट डिज़ाइनर मॉड्यूल का अपना संस्करण बना सकते हैं, इसे बढ़ा सकते हैं और इसमें सुधार कर सकते हैं। यहां तक कि अपेक्षाकृत अनुभवहीन उपयोगकर्ता मॉड्यूल के ब्रेडबोर्ड संस्करण को इसके कार्यप्रणाली समझने के लिए स्वयं बना सकते हैं एवं पैसा बचा सकते हैं।

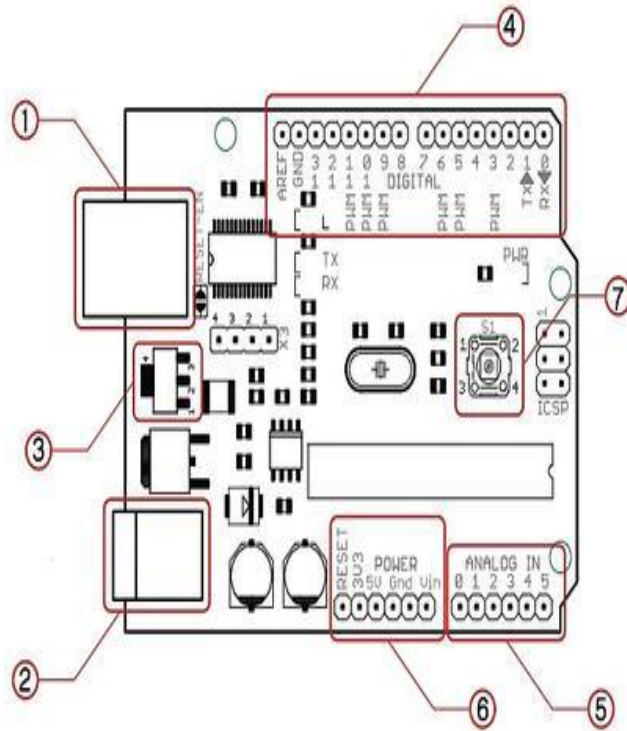
## Arduino Family: -

बाजार में विभिन्न प्रकार के Arduino बोर्ड मौजूद हैं जिनमें Arduino UNO, Arduino Nano, Red Board, LilyPad Arduino, Arduino Mega, Arduino Leonardo, Arduino Shields शामिल हैं। ये सभी Arduino बोर्ड विशिष्टताओं, सुविधाओं और उपयोगों में भिन्न हैं और विभिन्न प्रकार के इलेक्ट्रॉनिक्स प्रोजेक्ट में उपयोग किए जाते हैं।

## FUNCTIONAL BLOCK DIAGRAM OF ARDUINO BOARD: -

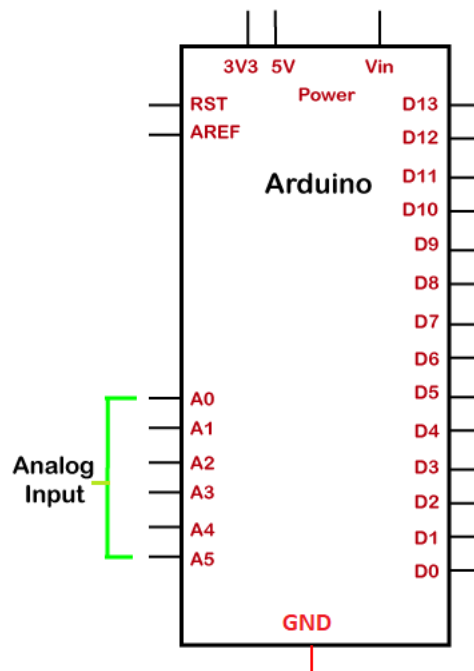
Arduino बोर्ड पर माइक्रोकंट्रोलर, डिजिटल इनपुट/आउटपुट पिन, USB इंटरफ़ेस और कनेक्टर, एनालॉग पिन, रीसेट बटन, पावर बटन, LED's, क्रिस्टल ऑसिलेटर और वोल्टेज रेगुलेटर जैसे विभिन्न कंपोनेंट्स मौजूद होते हैं। बोर्ड के प्रकार के आधार पर कुछ कंपोनेंट्स भिन्न हो सकते हैं।





The most important parts on the Arduino board high lighted in red:

- 1: USB connector
- 2: Power connector
- 3: Automatic power switch
- 4: Digital pins
- 5: Analog pins
- 6: Power pins
- 7: Reset switch



**Power USB:** Arduino बोर्ड को कंप्यूटर से USB केबल का उपयोग करके संचालित किया जा सकता है। इसके लिए सिर्फ USB केबल को USB कनेक्शन से कनेक्ट करना है ।

**Power (Barrel Jack):** Arduino बोर्डों को बैरल जैक से जोड़कर सीधे AC मेन बिजली आपूर्ति से संचालित किया जा सकता है।

**Voltage Regulator:** वोल्टेज रेगुलेटर का कार्य Arduino बोर्ड को दिए गए वोल्टेज को नियंत्रित करना और प्रोसेसर और अन्य तत्वों द्वारा उपयोग किए जाने वाले डीसी वोल्टेज को स्थिर करना है।

**Crystal Oscillator:** क्रिस्टल ऑसिलेटर Arduino को समय के मुद्दों से निपटने में मदद करता है। Arduino समय की गणना, क्रिस्टल ऑसिलेटर का उपयोग करके करता है। Arduino क्रिस्टल पर छपी संख्या 16.000H9H हमें बताता है कि आवृत्ति 16 मेगाहर्ट्ज है।

**Arduino Reset:** आप अपने Arduino बोर्ड को रीसेट कर सकते हैं, यानी अपने प्रोग्राम को शुरू से शुरू कर सकते हैं। आप यूएनओ बोर्ड को दो तरह से रीसेट कर सकते हैं। सबसे पहले, बोर्ड पर रीसेट बटन का उपयोग करके। दूसरा, आप एक बाहरी रीसेट बटन को RESET लेबल वाले Arduino पिन से कनेक्ट कर सकते हैं।

### **Pins (3.3, 5, GND, Vin):**

3.3V - 3.3 वोल्ट आउटपुट सप्लाई

5V - 5 वोल्ट आउटपुट सप्लाई

Arduino बोर्ड के साथ उपयोग किए जाने वाले अधिकांश घटक 3.3 वोल्ट और 5 वोल्ट के साथ ठीक काम करते हैं।

GND (ग्राउंड) - Arduino पर कई GND पिन हैं, जिनमें से किसी का भी उपयोग आपके सर्किट को ग्राउंड करने के लिए किया जा सकता है।

VIN - इस पिन का उपयोग Arduino बोर्ड को बाहरी शक्ति स्रोत से बिजली देने के लिए भी किया जा सकता है, जैसे AC मेन बिजली की आपूर्ति।

**Analog pins:** Arduino UNO बोर्ड में छह एनालॉग इनपुट पिन A0 से A5 हैं। ये पिन एनालॉग सेंसर जैसे ह्यूमिडिटी सेंसर या टेम्परेचर सेंसर से सिग्नल को पढ़ सकते हैं और इसे एक डिजिटल वैल्यू में बदल सकते हैं जिसे माइक्रोप्रोसेसर द्वारा पढ़ा जा सकता है।

**Main microcontroller:** प्रत्येक Arduino बोर्ड का अपना माइक्रोकंट्रोलर होता है। आप इसे अपने बोर्ड का दिमाग मान सकते हैं। Arduino पर मुख्य IC (इंटीग्रेटेड सर्किट) बोर्ड से बोर्ड में थोड़ा अलग है। माइक्रोकंट्रोलर आमतौर पर ATMEL कंपनी के होते हैं। Arduino IDE से एक नया प्रोग्राम लोड करने से पहले आपको पता होना चाहिए कि आपके बोर्ड के पास क्या IC है। यह जानकारी आईसी के शीर्ष पर उपलब्ध है। आईसी निर्माण और कार्यों के बारे में अधिक जानकारी के लिए, आप डेटा शीट का उल्लेख कर सकते हैं।

**ICSP PIN:** अधिकतर, ICSP एक AVR है, जो MOSI, MISO, SCK, RESET, VCC और GND से मिलकर Arduino के लिए एक छोटा प्रोग्रामिंग हेडर है। इसे अक्सर SPI (सीरियल पेरिफेरल इंटरफेस) के रूप में जाना जाता है, जिसे आउटपुट के "विस्तार" के रूप में माना जा सकता है। दरअसल, SPI बस के मास्टर आउटपुट डिवाइस को स्लेव करता है |

### **Power LED indicator:**

जब Arduino को एक पॉवर सोर्स में प्लग करते हैं, और बोर्ड सही ढंग से संचालित होता है तब यह एलईडी प्रकाश करना चाहिए। अगर यह लाइट चालू नहीं होती है, तो कनेक्शन में कुछ गड़बड़ है।

**TX and RX LEDs:** बोर्ड पर, दो लेबल होता है : TX (संचारित) और RX (प्राप्त)। वे Arduino UNO बोर्ड में दो स्थानों पर दिखाई देते हैं। सबसे पहले, डिजिटल पिन 0 और 1 पर सीरियल संचार के लिए जिम्मेदार पिनो को इंगित करने के लिए होते हैं | दूसरा, TX और RX LED, सीरियल डेटा भेजते समय TX एलईडी अलग-अलग गति से चमकती है। फ्लैशिंग की गति बोर्ड द्वारा उपयोग की जाने वाली बॉड दर पर निर्भर करती है। प्राप्त करने की प्रक्रिया के दौरान RX चमकता है।

**Digital I/O:** Arduino UNO बोर्ड में 14 डिजिटल (D0 से D13) I/O पिन हैं (जिनमें से 6 PWM (पल्स चौड़ाई मॉड्यूलेशन) आउटपुट प्रदान करते हैं। इन पिनो को लॉजिक वैल्यू (0 या 1) या डिजिटल के रूप में पढ़ने के लिए इनपुट डिजिटल पिन के रूप में काम करने के लिए कॉन्फिगर किया जा सकता है। एलईडी, रिले आदि जैसे विभिन्न मॉड्यूल को चलाने के लिए आउटपुट पिन का उपयोग होता है | PWM उत्पन्न करने के लिए "~" लेबल वाले पिन का उपयोग किया जा सकता है।

**AREF: एनालॉग रिफरेंस** यह कभी-कभी, बाहरी संदर्भ वोल्टेज (0 और 5 वोल्ट के बीच) को एनालॉग इनपुट पिन के लिए ऊपरी सीमा के रूप में सेट करने के लिए उपयोग किया जाता है।

## **What is open-source software?**

ओपन-सोर्स सॉफ्टवेयर (OSS) एक लाइसेंस के तहत प्रदान किया जाता है जो उपयोगकर्ताओं को उनके उद्देश्यों के लिए इसके सोर्स कोड को एक्सेस करने, बदलने और सुधारने की अनुमति देता है।

ऐसे सॉफ्टवेयर जिसमें सोर्स कोड शामिल होना चाहिए और सोर्स कोड के साथ-साथ संकलित रूप में उसी नाम और प्रारंभिक उत्पाद के लाइसेंस के तहत वितरण किया जा सके। एक प्रोग्राम को उद्योग या परियोजना की परवाह किए बिना किसी के द्वारा व्युत्पन्न कार्यों में संशोधित और उपयोग किया जा सकता है। उदाहरण के लिए, इंजीनियर प्रोग्राम की कार्यक्षमता बढ़ा सकते हैं, बग ठीक कर सकते हैं या सोर्स कोड के कुछ हिस्सों का उपयोग करके कम समय में एक नया सॉफ्टवेयर बना सकते हैं | डेवलपर किसी उत्पाद को मुफ्त में साझा कर सकते हैं या उसे बेच सकते हैं। लेकिन एक सीमा है:

डेवलपर्स को समान शर्तों के तहत एक ओपन-सोर्स प्रोग्राम के संशोधित भाग को वितरित करना होगा और स्रोत कोड प्रदान करना होगा।

## Benefits of using open-source software

ओपन-सोर्स लाइसेंसिंग में कॉपीराइट प्रतिबंध शामिल नहीं हैं। उपयोग की यह सापेक्ष स्वतंत्रता कई उपयोगकर्ताओं को ऐसे उत्पादों की ओर आकर्षित करती है। सार्वजनिक रूप से उपलब्ध सोर्स कोड वाले सॉफ्टवेयर के मुख्य लाभ हैं:

- **Flexibility (लचीलापन)** - सॉफ्टवेयर को विशिष्ट व्यावसायिक आवश्यकताओं को पूरा करने के लिए अनुकूलित किया जा सकता है। अतिरिक्त कार्यक्षमता जोड़ने के लिए इंजीनियर अधिक कोड लिख सकते हैं और इसके विपरीत - अनावश्यक भागों को हटा सकते हैं।
- **Stability (स्थिरता)** - आप इस उत्पाद का उपयोग दीर्घकालिक परियोजनाओं के लिए विश्वास के साथ कर सकते हैं क्योंकि यदि इसके लेखक इस पर काम करना बंद कर दें फिर भी यह बाजार से गायब नहीं होगा या पुराना नहीं होगा। उपयोगकर्ता समुदाय ओपन सोर्स सॉफ्टवेयर का ख्याल रखेगा।
- **Security & reliability (सुरक्षा और विश्वसनीयता)** - अलग-अलग कौशल स्तर वाले कई लोग एक ही सॉफ्टवेयर पर काम कर सकते हैं, जिससे कोड असंगति हो सकती है। तभी ओपन सोर्स की संस्कृति फायदेमंद होती है। दुनिया भर के अन्य डेवलपर इस कोड की समीक्षा कर सकते हैं, एरर ठीक कर सकते हैं और अपडेट कर सकते हैं। जितनी तेज़ कोड समीक्षा होगी, सॉफ्टवेयर उतना ही अधिक सुरक्षित और विश्वसनीय होगा। लेखक और उपयोगकर्ता एक समाधान में सुधार करते हैं क्योंकि अच्छा प्रदर्शन करने के लिए उन्हें इसकी आवश्यकता होती है।
- **Easier evaluation (आसान मूल्यांकन)** - सोर्स कोड की पूर्ण पारदर्शिता डेवलपर्स की टीम को किसी उत्पाद की क्षमताओं और खामियों के बारे में सीखने की जांच और मूल्यांकन करने की अनुमति देती है।
- **Better Support (बेहतर समर्थन)** - एक OSS उपयोगकर्ता के रूप में, आपके पास तकनीकी सलाह और समर्थन प्राप्त करने के अधिक तरीके हैं: एक विक्रेता से, इस सटीक उत्पाद में विशेषज्ञता रखने वाली परामर्श कंपनी से, या अन्य उपयोगकर्ताओं से जो अपने अनुभव और ज्ञान को मंचों या मेलिंग सूचियों में साझा करने के लिए तैयार हैं।

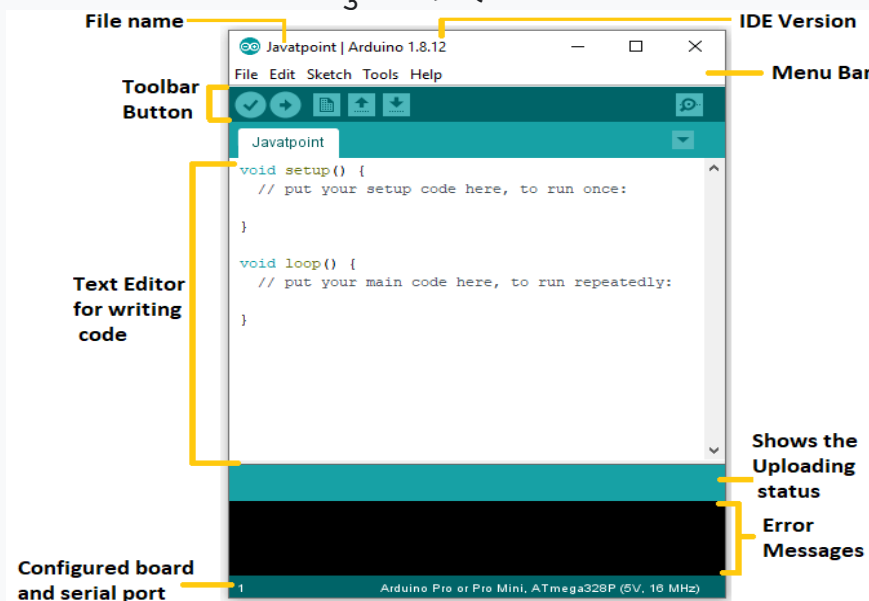


- **Possible Savings (बचत की संभावना)** - एक ओपन सोर्स सॉफ्टवेयर की खरीद लागत आमतौर पर एक कमर्शियल सॉफ्टवेयर की तुलना में कम होती है और फ्री ओपन सोर्स प्रोग्राम भी मौजूद होते हैं।
- **Possible Learning (संभावित सीख)** - इसके अलावा, जूनियर डेवलपर्स या छात्र ओपन-सोर्स कोड का उपयोग यह जानने के लिए कर सकते हैं कि कोड को बेहतर कैसे बनाया जाए।

## ARDUINO Integrated Development Environment (IDE):

Arduino इंटीग्रेटेड डेवलपमेंट एनवायरनमेंट - या Arduino Software (IDE) - में कोड लिखने के लिए टेक्स्ट एडिटर, मेसेज एरिया, टेक्स्ट कंसोल, सामान्य कार्यों के लिए बटन के साथ एक टूलबार और मेनू की एक श्रृंखला होती है। यह Arduino हार्डवेयर में प्रोग्राम अपलोड करने और उनके साथ संवाद करने के लिए उपयोग किया जाता है।

Arduino Software (IDE) का उपयोग करके लिखे गए प्रोग्राम को स्केच कहा जाता है। ये स्केच टेक्स्ट एडिटर में लिखे जाते हैं और फ़ाइल एक्सटेंशन .ino के साथ सेव होते हैं। एडिटर में टेक्स्ट को कॉपी/पेस्ट और सर्चिंग / रेप्लेसिंग की सुविधाएँ हैं। मेसेज एरिया सेव और एक्सपोर्ट करते समय प्रतिक्रिया देता है और एरर को भी प्रदर्शित करता है। कंसोल में Arduino Software (IDE) द्वारा टेक्स्ट आउटपुट प्रदर्शित होता है, जिसमें संपूर्ण एरर मेसेज और अन्य जानकारी शामिल होते हैं। विंडो का निचला दाहिना कोना कॉन्फ़िगर किए गए बोर्ड और सीरियल पोर्ट को प्रदर्शित करता है। टूलबार बटन आपको प्रोग्राम को सत्यापित करने और अपलोड करने, स्केच बनाने, ओपन करने, सेव करने और सीरियल मॉनिटर खोलने की अनुमति देते हैं।



## Arduino ISP:

Arduino ISP एक इन-सिस्टम-प्रोग्रामर है जिसका उपयोग AVR माइक्रोकंट्रोलर्स को प्रोग्राम करने के लिए किया जाता है। बूटलोडर की आवश्यकता के बिना AVR-आधारित Arduino बोर्डों पर सीधे स्केच अपलोड करने के लिए Arduino ISP का उपयोग किया जा सकता है। अन्यथा इसका उपयोग बूटलोडर को पुनर्स्थापित करने के लिए भी किया जा सकता है।

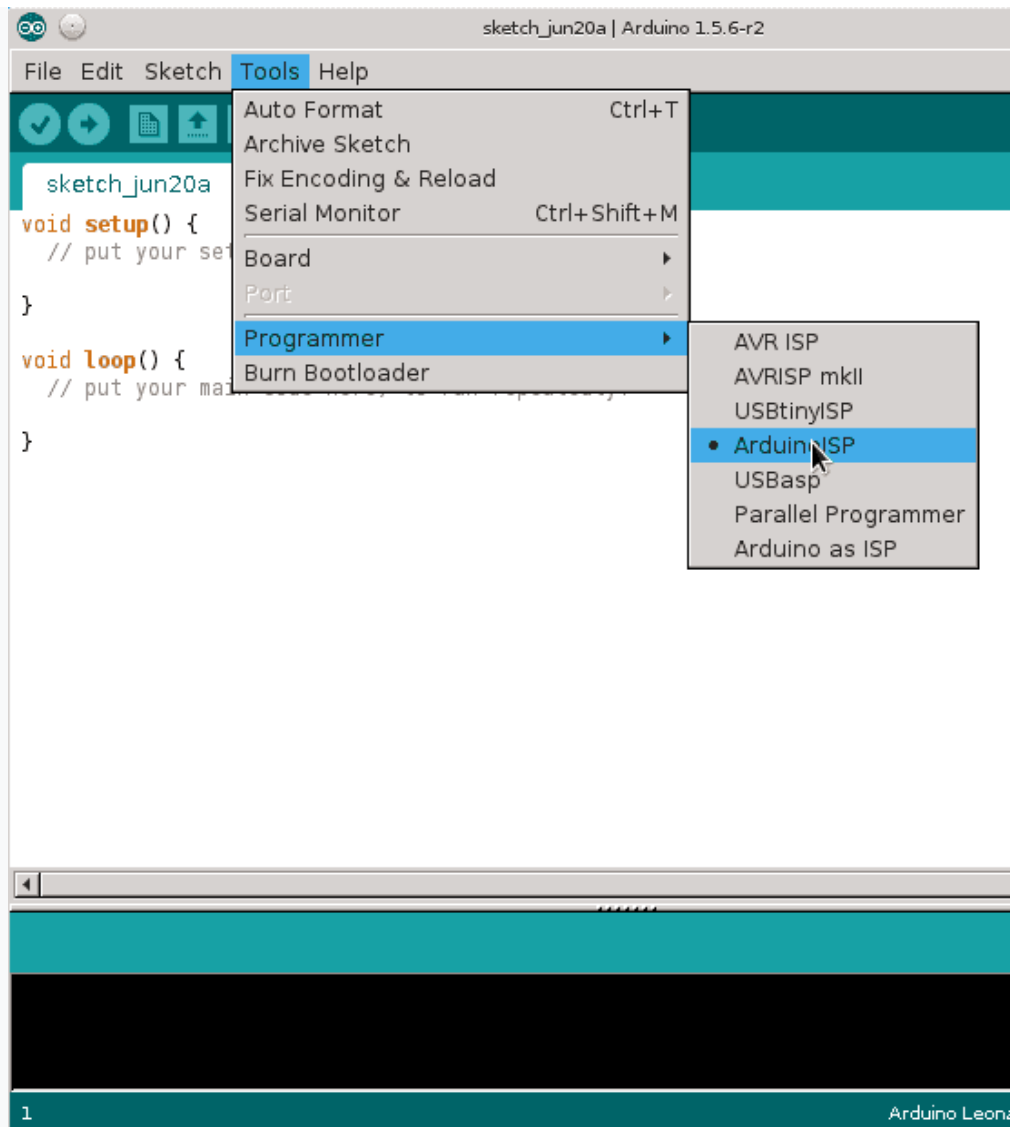
### How to connect the Arduino ISP?

Arduino ISP को Arduino बोर्ड के 6-पिन ICSP कनेक्टर पर प्लग करें जिसे आप प्रोग्राम करना चाहते हैं, जैसा कि चित्र में दिखाया गया है।



Arduino ISP को माइक्रो USB केबल से अपने कंप्यूटर से कनेक्ट करें, और टारगेट बोर्ड को पावर सोर्स (USB केबल या पावर जैक के साथ) से कनेक्ट करें। टारगेट बोर्ड को Arduino ISP से भी संचालित किया जा सकता है। इस सुविधा को सक्षम करने के लिए आपको सोल्डरिंग आयरन और सोल्डरिंग टिन की एक बूंद के साथ SJVCC जम्पर को बंद करना होगा।

आपके द्वारा सभी कनेक्शन बनाने के बाद आपको Arduino IDE में सही प्रोग्रामर का चयन करना होगा जैसा कि चित्र में दिखाया गया है।

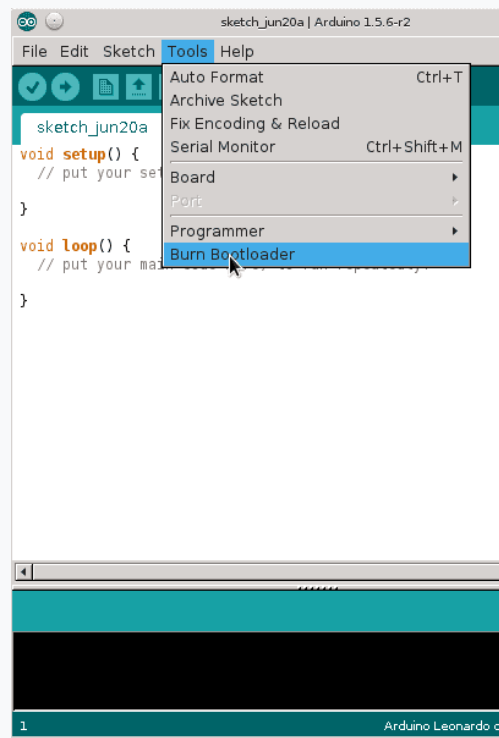


## Arduino Bootloader:

बूटलोडर एक प्रकार का सॉफ्टवेयर है जो स्केच अपलोड करते समय Arduino बोर्ड को Arduino IDE के साथ संचार करने की अनुमति देता है। आम तौर पर जब आप एक माइक्रोकंट्रोलर पर एक प्रोग्राम लोड करना चाहते हैं तो आपको Arduino ISP जैसे एक बाहरी प्रोग्रामर की आवश्यकता होती है | बूटलोडर बाहरी प्रोग्रामर की ज़रूरतों को समाप्त कर देता है, क्योंकि वह प्रोटोकॉल जो आपके कंप्यूटर को AVR की फ्लैश मेमोरी को प्रोग्राम करने की अनुमति देता है, बूटलोडर के अंदर समाहित है। सभी AVR-आधारित Arduino बोर्ड बूटलोडर के साथ पहले से इंस्टॉल आते हैं लेकिन कभी-कभी अपलोड प्रक्रिया या कुछ स्केच उस मेमोरी को दूषित कर सकते हैं जहां बूटलोडर भविष्य की अपलोड प्रक्रियाओं की विफलता का कारण बनता है। Arduino ISP के साथ बूटलोडर को फिर से बर्न करने से बूटलोडर को पुनर्स्थापित किया जा सकता है और फिर से USB पोर्ट का उपयोग करके अपने Arduino

को अपलोड करने के लिए वापस लाया जा सकता है। आप एकदम नए ATmega में बूटलोडर को बर्न करने के लिए Arduino ISP का उपयोग कर सकते हैं। यदि आप अपने Arduino पर ATmega माइक्रोकंट्रोलर को बदलते हैं, तो आपको स्केच को सामान्य तरीके से लोड करने के लिए बूटलोडर को बर्न करने की आवश्यकता होगी। आप इसे केवल Arduino ISP के साथ कर सकते हैं।

बूटलोडर को बर्न करना Arduino IDE द्वारा प्रदान की जाने वाली एक आसान सुविधा है। बूटलोडर को अपने बोर्ड पर अपलोड करने के लिए बस पहले बताए अनुसार सब कुछ कनेक्ट करें और टूल मेनू में बर्न बूटलोडर पर क्लिक करें।



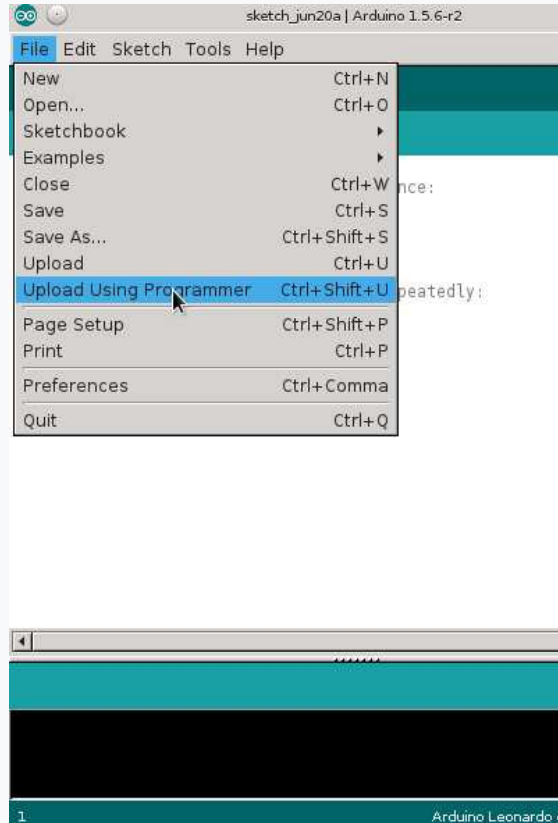
## Uploading a sketch:

Arduino ISP प्रोग्रामर का उपयोग AVR- आधारित Arduino बोर्डों पर या Arduino सॉफ्टवेयर द्वारा समर्थित अन्य AVR माइक्रोकंट्रोलर्स पर स्केच लोड करने के लिए भी किया जा सकता है।

मानक प्रक्रिया का उपयोग करके स्केच अपलोड करने के लिए बूटलोडर की उपस्थिति की आवश्यकता होती है। इसके बजाय, बाहरी प्रोग्रामर को स्केच अपलोड करने के विकल्प के रूप में चुनकर संपूर्ण फ्लैश मेमोरी स्पेस का उपयोग करके स्केच अपलोड किया जाएगा। बूटलोडर द्वारा उपयोग की जाने वाली मेमोरी का उपयोग करके स्केच के लिए अधिक स्थान की आवश्यकता होने पर यह उपयोगी हो सकता है।

**नोट:** याद रखें कि यदि आप बूटलोडर को ओवरराइट करते हैं तो आप Arduino IDE में अपलोड बटन पर क्लिक करके अन्य स्केच अपलोड नहीं कर पाएंगे। यदि आप अपने Arduino को पहले की तरह फिर से उपयोग करना चाहते हैं, तो आपको पहले बूटलोडर को बर्न करना होगा।

अपना स्केच समाप्त करने के बाद और सब कुछ सही ढंग से सेटअप हो जाने के बाद, फ़ाइल मेनू पर जाएं और प्रोग्रामर का उपयोग करके अपलोड पर क्लिक करें। वैकल्पिक रूप से, एक कीबोर्ड शॉर्टकट है। प्रोग्रामर का उपयोग करके अपलोड करने के लिए "CTRL+SHIFT+U" Key दबाएं।

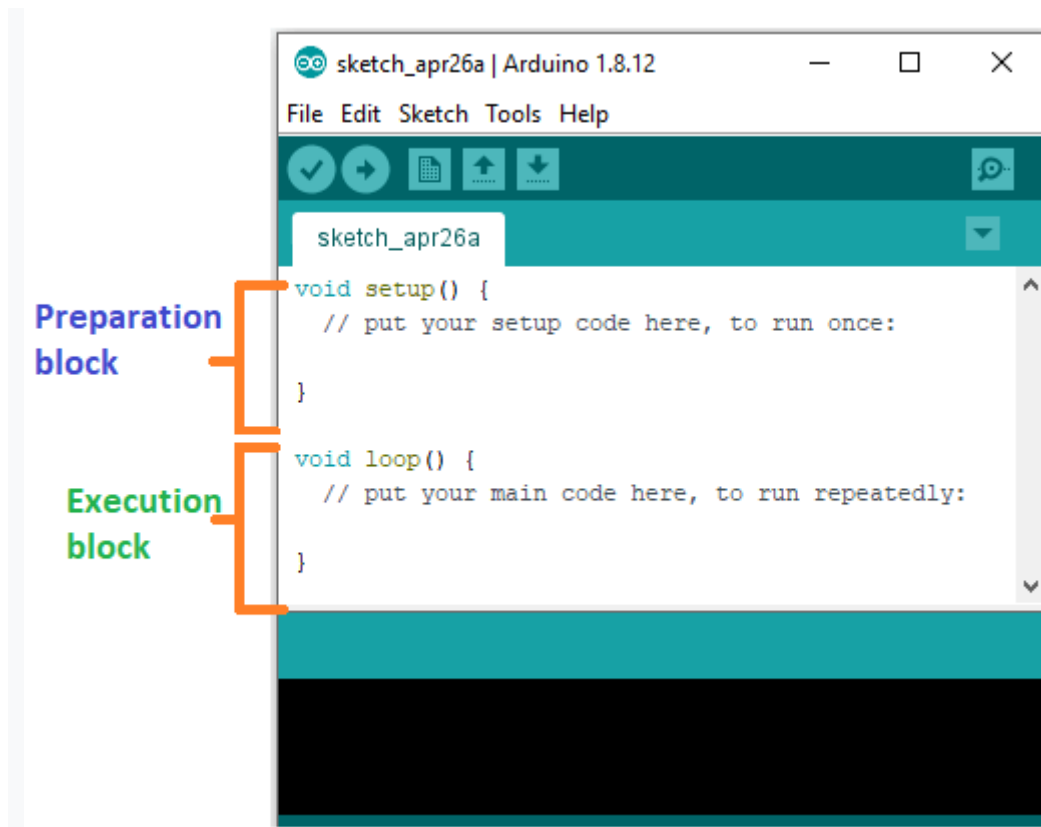


### Programming fuse bits on ATmega microcontrollers:

Arduino ISP का उपयोग ATmega माइक्रोकंट्रोलर्स पर फ़्यूज बिट्स को प्रोग्राम करने के लिए किया जा सकता है। प्रोग्रामिंग फ़्यूज आपको आंतरिक बाह्य उपकरणों और माइक्रोकंट्रोलर के व्यवहार को कॉन्फ़िगर करने की अनुमति देता है। उदाहरण के लिए, आप घड़ी की आवृत्ति चुन सकते हैं, वॉचडॉग टाइमर की प्रोग्रामिंग कर सकते हैं और बहुत कुछ। इसके लिए कुछ अनुभव और ध्यान देने की आवश्यकता है क्योंकि इन फ़्यूज को गलत तरीके से सेट करने से माइक्रोकंट्रोलर अब काम नहीं कर सकता है, और इसे ठीक करना बहुत मुश्किल हो सकता है।

## Arduino Programming:

कोडिंग स्क्रीन को दो ब्लॉक में बांटा गया है। सेटअप को प्रिपरेशन (तैयारी) ब्लॉक माना जाता है, जबकि लूप को execution (निष्पादन) ब्लॉक माना जाता है। इसे नीचे दिखाया गया है:



सेटअप और लूप ब्लॉक में स्टेटमेंट्स का सेट कर्ली ब्रैकेट्स के साथ संलग्न है। किसी विशेष प्रोजेक्ट के लिए कोडिंग आवश्यकताओं के आधार पर हम कई स्टेटमेंट लिख सकते हैं।

```
void setup ()  
{  
    Coding statement 1;  
    Coding statement 2;  
    .  
    .  
    .  
    Coding statement n;  
}
```

```

void loop ()
{
    Coding statement 1;
    Coding statement 2;
    .
    .
    .
    Coding statement n;
}

```

`setup()`: इसमें एकसिक्यूट किए जाने वाले कोड का प्रारंभिक भाग होता है। सेटअप सेक्शन में पिन मोड, लाइब्रेरी, वेरिएबल्स आदि को इनिशियलाइज़ किया जाता है। इसे केवल एक बार प्रोग्राम अपलोड करने के दौरान और Arduino बोर्ड को रीसेट या पावर अप करने के बाद निष्पादित किया जाता है।

शून्य सेटअप () प्रत्येक स्केच के शीर्ष पर रहता है। जैसे ही प्रोग्राम चलना शुरू होता है, `setup` फंक्शन के कर्ली ब्रैकेट के अंदर का कोड एकसिक्यूट होता है, और यह केवल एक बार ही एकसिक्यूट होता है।

`loop()` : लूप फंक्शन के कर्ली ब्रैकेट के अन्दर के स्टेटमेंट्स को बार बार एकसिक्यूट किया जाता है | वेरिएबल्स के वैल्यू के आधार पर `loop` फंक्शन के स्टेटमेंट्स को रिपीट किया जाता है |

Time in Arduino:

Arduino प्रोग्रामिंग में समय को एक मिलीसेकंड में मापा जाता है |

जहाँ, 1 सेकंड = 1000 मिलीसेकंड

हम मिलीसेकंड के अनुसार समय को समायोजित कर सकते हैं।

उदाहरण के लिए, 5-सेकंड की देरी के लिए, प्रदर्शित होने वाला समय 5000 मिलीसेकंड होगा।

### Decision making Techniques:

डिसिशन मेकिंग स्टेटमेंट्स प्रोग्राम की दिशा और प्रवाह तय करते हैं | उन्हें कंडीशनल स्टेटमेंट्स के रूप में भी जाना जाता है क्योंकि वे बूलियन एक्सप्रेशन के साथ शर्तों को निर्दिष्ट करते हैं जिनका मूल्यांकन सही (true) या गलत (false) बूलियन मान के लिए किया जाता है |

डिसिशन मेकिंग स्टेटमेंट्स निम्नलिखित हैं :

If statement

If ... else statement

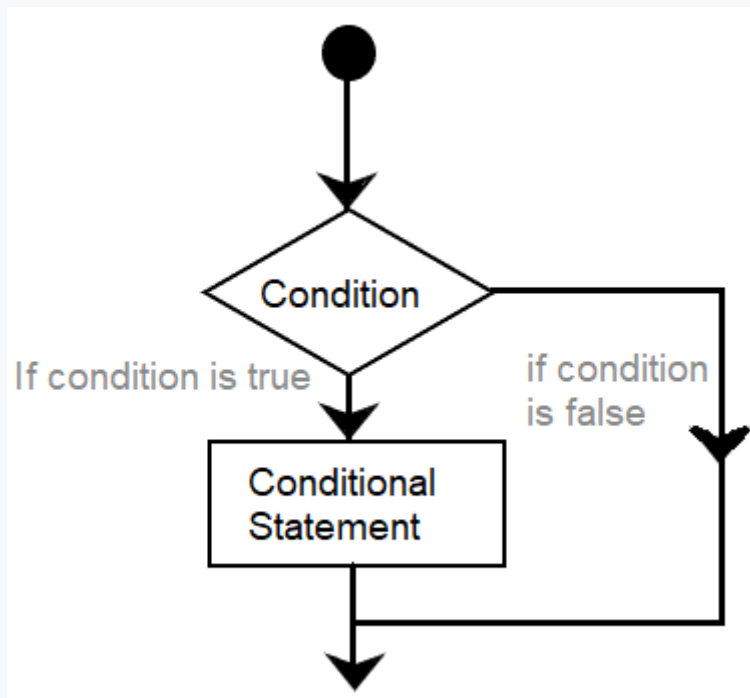
If ...else-if statement

Nested if statement

Switch statement

### if statement:

यदि कोड में स्थिति सही है, तो संबंधित कार्य एक्सीक्यूट किया जाता है। if स्टेटमेंट स्थिति की जाँच करता है और फिर एक स्टेटमेंट या स्टेटमेंट का एक सेट निष्पादित करता है।



दिए गए फ्लोचार्ट में यह स्पष्ट है कि यदि कंडीशन सही है तो स्टेटमेंट ब्लॉक एक्सीक्यूट होगा और यदि गलत है तो यह ब्लॉक एक्सीक्यूट नहीं होगा ।

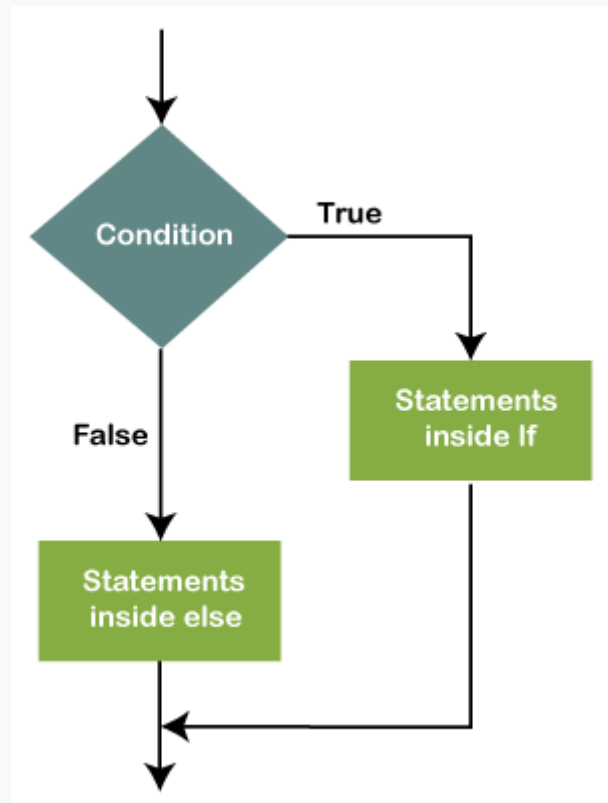
If स्टेटमेंट का सिंटैक्स :

```
if ( condition)
{
// include statements
// if the condition is true
// then performs the function or task specified inside the curly braces
}
```

**if else statement:** if else स्टेटमेंट में if कंडीशन सही होने पर if ब्लॉक एक्सीक्यूट होता है एवं कंडीशन गलत होने पर else ब्लॉक एक्सीक्यूट होता है ।



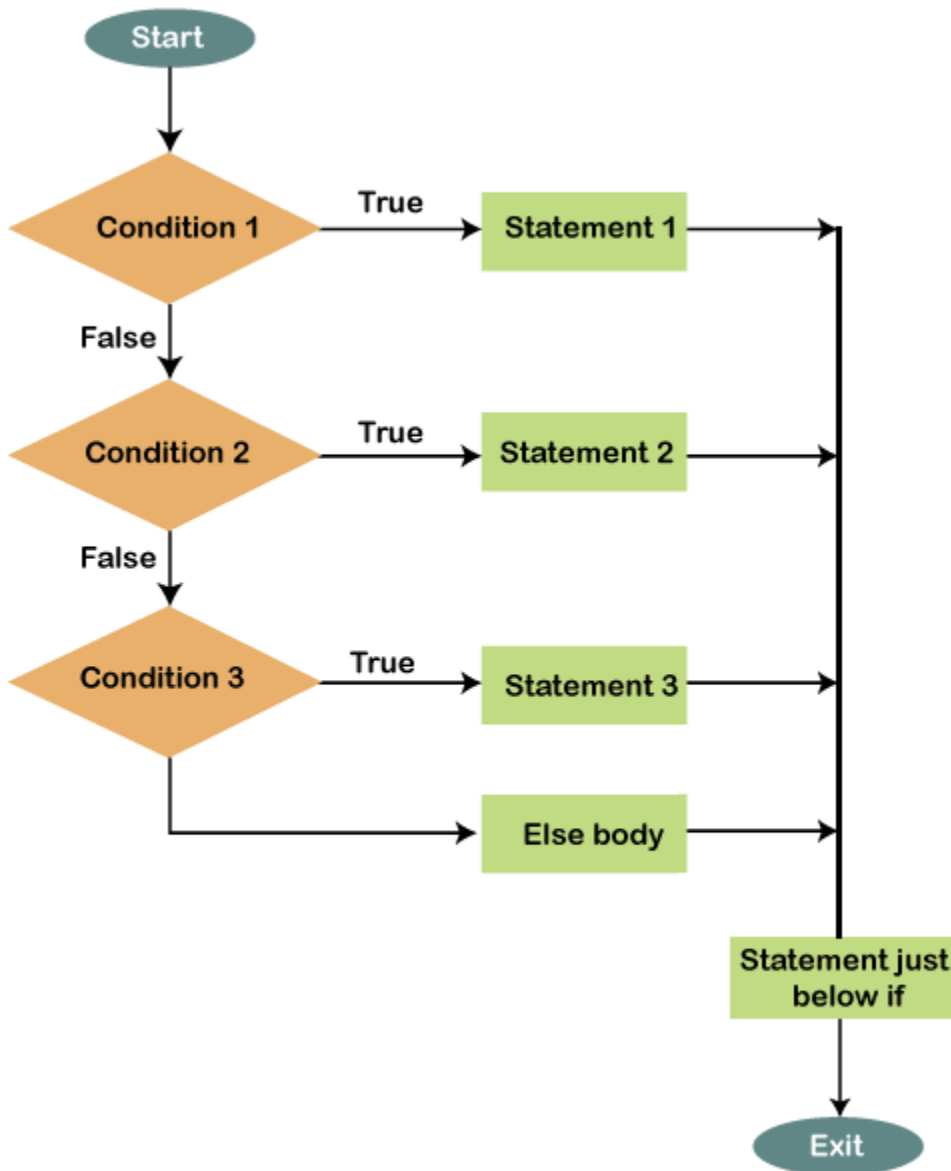
फ़्लोचार्ट नीचे दिखाया गया है:



If else सिंटेक्स:

```
if (condition)
{
// statements
}
else
{
//statements
}
```

else () कथन में अन्य if कथन भी शामिल हो सकते हैं। इसके कारण हम एक ही प्रोग्राम में कई Statement चला सकते हैं।



सही स्टेटमेंट मिलने तक स्टेटमेंट्स को एक-एक करके निष्पादित किया जाएगा। जब सही कथन मिल जाता है, तो यह कोड में अन्य सभी if और else कथनों को छोड़ देगा और कोड के संबंधित ब्लॉक को रन किया जायेगा ।

### **Looping Techniques**

**for Loop:** लूप के लिए कर्ली ब्रैकेट के अंदर दिए गए बयानों को निर्दिष्ट स्थिति के अनुसार बार-बार निष्पादित किया जाता है। लूप के लिए इंक्रीमेंट काउंटर का उपयोग लूप रिपीटिशन को बढ़ाने या घटाने के लिए किया जाता है।

फॉर स्टेटमेंट का उपयोग आमतौर पर दोहराए जाने वाले कार्य या संचालन के लिए या सरणियों के संयोजन में डेटा/पिन के समूह पर काम करने के लिए किया जाता है।

for loop सिंटेक्स :

**for** (initialization; condition; increment)

```
{  
  \\ statements  
}
```

**initialization:** इसे वेरिएबल के आरंभीकरण के रूप में परिभाषित किया गया है।

**condition:** प्रत्येक निष्पादन पर स्थिति का परीक्षण किया जाता है। यदि स्थिति सही है, तो यह दिए गए कार्य को निष्पादित करेगा। लूप तभी समाप्त होता है जब कंडीशन गलत हो जाती है।

**increment:** इसमें इंक्रीमेंट ऑपरेटर शामिल है, जैसे कि  $i++$ ,  $i--$ ,  $i+1$ , इत्यादि। इसे हर बार तब तक बढ़ाया जाता है जब तक कि कंडीशन गलत न हो जाय |

उदाहरण के लिए :

1. **for** ( $i = 0 ; i < 5 ; i + +$ )

उपरोक्त कथन लूप को पांच बार निष्पादित करेगा।  $i$  का मान 0 से 4 तक होगा।

2. **for** ( $i = 0 ; i <= 5 ; i + +$ )

उपरोक्त कथन लूप को छह बार निष्पादित करेगा।  $i$  का मान 0 से 5 तक होगा।

**while loop :** while लूप कंडीशनल लूप है जो कोष्ठक के अंदर कोड को निष्पादित करना जारी रखता है जब तक कि निर्दिष्ट स्थिति गलत नहीं हो जाती।

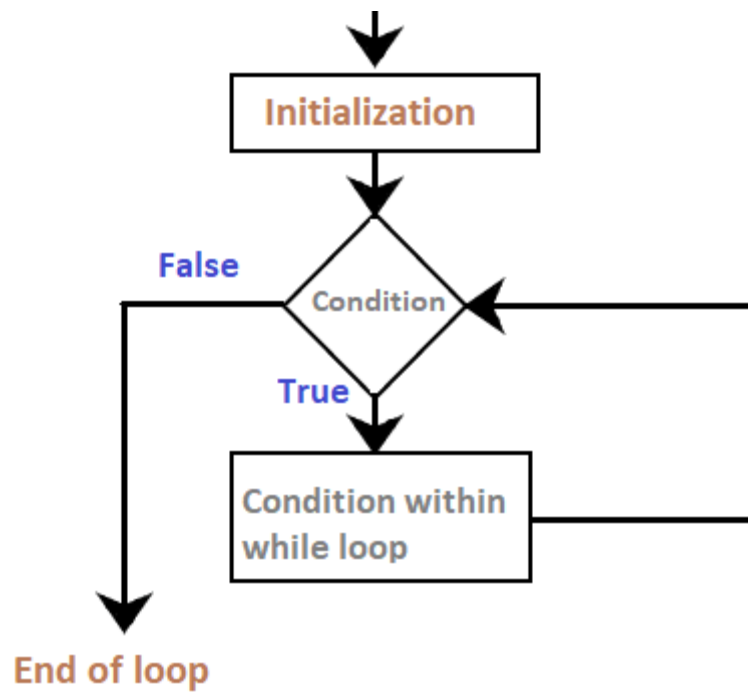
परीक्षण की स्थिति को बदलने या रोकने के लिए बनाए जाने तक while लूप कभी बाहर नहीं निकलेगा। Arduino में while लूप के सामान्य उपयोग में सेंसर परीक्षण, कैलिब्रेशन (सेंसर के इनपुट को कैलिब्रेट करना), वेरिएबल इंक्रीमेंट आदि शामिल हैं।

While loop सिंटैक्स :

```
while (condition)  
{  
  // code or set of statements  
}
```

**कंडीशन:** यह बूलियन एक्सप्रेशन को निर्दिष्ट करता है, जो कंडीशन के सही या गलत होने का निर्धारण करता है।

While loop फ्लोचार्ट :



### Do-while loop:

Do-while लूप की कार्यप्रणाली, while लूप के समान है। do-while के अंदर की कंडीशन कम से कम एक बार एक्जीक्यूट होगी। ऐसा इसलिए है क्योंकि शुरुआत के बजाय लूप के अंत में स्थिति का परीक्षण किया जाता है।

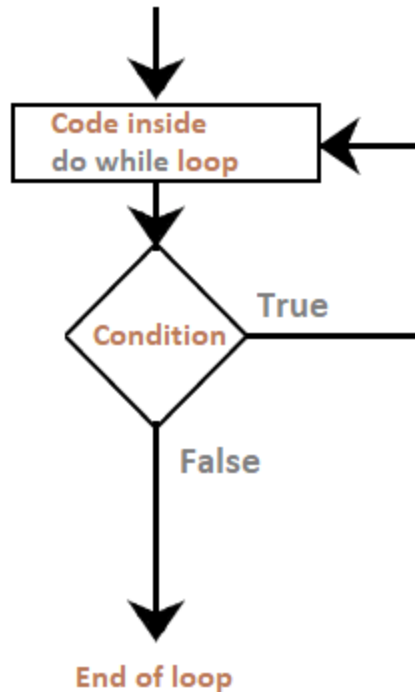
Do-while syntax:

```

do
{
  // code or set of statements
} while (condition);
  
```

कंडीशन: यह बूलियन एक्सप्रेशन को निर्दिष्ट करता है, जो कंडीशन के सही या गलत होने का निर्धारण करता है।

Flowchart of do-while loop



## **Concept of Input and Output port of embedded development**

### **board:**

Arduino बोर्ड पर पिन को इनपुट या आउटपुट के रूप में कॉन्फ़िगर किया जा सकता है। यह ध्यान रखना महत्वपूर्ण है कि अधिकांश Arduino एनालॉग पिन, डिजिटल पिन के समान ही कॉन्फ़िगर और उपयोग किए जा सकते हैं।

### **Pins Configured as INPUT:**

Arduino पिन डिफ़ॉल्ट रूप से इनपुट के रूप में कॉन्फ़िगर किए गए हैं, इसलिए जब आप उन्हें इनपुट के रूप में उपयोग कर रहे हों तो उन्हें pinMode() के साथ इनपुट के रूप में स्पष्ट रूप से घोषित करने की आवश्यकता नहीं है। इस तरह से कॉन्फ़िगर किए गए पिन को उच्च-प्रतिबाधा (high impedance) अवस्था में कहा जाता है।

इनपुट पिन के सामने 100 मेगाओम के श्रृंखला प्रतिरोधी के बराबर प्रतिरोध होने के कारण इनपुट पिन सैंपलिंग सर्किट पर बहुत कम मांग करते हैं |

इसका मतलब है कि इनपुट पिन को एक स्टेट से दूसरे स्टेट में स्विच करने में बहुत कम करंट लगता है। यह कैपेसिटिव टच सेंसर को लागू करने या एक एलईडी को फोटोडायोड के रूप में पढ़ने जैसे कार्यों के लिए पिन को उपयोगी बनाता है।

### **Pins Configured as OUTPUT:**

pinMode() के साथ OUTPUT के रूप में कॉन्फ़िगर किए गए पिन को कम-प्रतिबाधा अवस्था में कहा जाता है। इसका मतलब है कि वे अन्य सर्किटों को पर्याप्त मात्रा में करंट प्रदान कर सकते हैं। एटमेगा पिन अन्य उपकरणों/सर्किट को 40 mA (मिलीएम्पस) करंट स्रोत की तरह सकारात्मक विद्युत् प्रवाह प्रदान कर सकते हैं या सिंक की तरह नकारात्मक करंट प्रदान कर सकते हैं। यह एक एलईडी को उज्ज्वल रूप से रोशन करने के लिए या कई सेंसर को चलाने के लिए पर्याप्त करंट है, लेकिन रिले, सोलनॉइड्स या मोटर्स को चलाने के लिए पर्याप्त करंट नहीं है।

आउटपुट पिन से उच्च धारा उपकरणों को चलाने का प्रयास, पिन में आउटपुट ट्रांजिस्टर को नुकसान पहुंचा सकता है या नष्ट कर सकता है, या संपूर्ण एटमेगा चिप को नुकसान पहुंचा सकता है। अक्सर, इसके परिणामस्वरूप माइक्रोकंट्रोलर में "dead" पिन हो सकता है लेकिन शेष चिप्स अभी भी पर्याप्त रूप से कार्य करते हैं। इस कारण से, 470Ω या 1k रेसिस्टर्स के माध्यम से OUTPUT पिन को अन्य उपकरणों से कनेक्ट करना एक अच्छा विचार है, जब तक कि किसी विशेष एप्लिकेशन के लिए पिन से अधिकतम करंट की आवश्यकता न हो।

### **pinMode() Function:**

pinMode() फ़ंक्शन का उपयोग एक विशिष्ट पिन को इनपुट या आउटपुट के रूप में कॉन्फ़िगर करने के लिए किया जाता है। INPUT\_PULLUP मोड के साथ आंतरिक पुल-अप प्रतिरोधों को सक्षम करना संभव है। इसके अतिरिक्त, INPUT मोड आंतरिक पुल-अप को स्पष्ट रूप से अक्षम करता है।

pinMode function का syntax:

```
pinMode (pin, mode)
```

जहाँ, pin : पिन की संख्या जिसका मोड आप सेट करना चाहते हैं |

mode: हम संबंधित पिन नंबर के अनुसार मोड को INPUT या OUTPUT के रूप में सेट कर सकते हैं।

### **उदाहरण:**

```

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:**

DigitalWrite () फ़ंक्शन का उपयोग डिजिटल पिन पर उच्च या निम्न मान लिखने के लिए किया जाता है। यदि पिन को pinMode() के साथ एक OUTPUT के रूप में कॉन्फ़िगर किया गया है, तो इसका वोल्टेज हाई के लिए 5V (या 3.3V बोर्ड पर 3.3V) और LOW के लिए 0V (ग्राउंड) पर सेट किया जाएगा। यदि पिन को INPUT के रूप में कॉन्फ़िगर किया गया है, तो digitalWrite() इनपुट पिन पर आंतरिक पुलअप को इनेबल (HIGH) या डिसेबल (LOW) करेगा। इंटरनल पुलअप रेसिस्टर को इनेबल करने के लिए pinMode() को INPUT\_PULLUP पर सेट करने की अनुशंसा की जाती है।

यदि आप pinMode() को OUTPUT पर सेट नहीं करते हैं, और एक एलईडी को उस पिन से कनेक्ट करते हैं, तो digitalWrite(HIGH) कॉल करते समय, एलईडी मंद दिखाई दे सकती है। pinMode() को स्पष्ट रूप से सेट किए बिना, digitalWrite() ने आंतरिक पुल-अप प्रतिरोधी को इनेबल किया होगा, जो एक बड़े विद्युत्-सीमित प्रतिरोधी (करंट लिमिटिंग रेसिस्टर) की तरह कार्य करता है।

### **digitalWrite() फ़ंक्शन का सिंटैक्स :**

```

void loop() {
  digitalWrite (pin ,value);
}

```

pin : पिन की संख्या जिसका मोड आप सेट करना चाहते हैं |  
value: HIGH या LOW

उदाहरण:

```
int LED = 6; // LED connected to pin 6

void setup () {
  pinMode(LED, OUTPUT); // set the digital pin as output
}

void loop () {
  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, digitalRead() फ़ंक्शन के माध्यम से, यह पता लगा सकता है कि क्या उसके किसी एक पिन पर वोल्टेज लगाया गया है। एक ऑन/ऑफ सेंसर (जो किसी वस्तु की उपस्थिति का पता लगाता है) और एक एनालॉग सेंसर जिसका मूल्य लगातार बदलता रहता है, के बीच अंतर होता है। इस प्रकार के एनालॉग सेंसर को पढ़ने के लिए हमें एक अलग प्रकार के पिन की आवश्यकता होती है।

Arduino बोर्ड के निचले-दाएँ भाग में, आपको "एनालॉग इन" के रूप में चिह्नित छह पिन दिखाई देंगे। ये विशेष पिन न केवल यह बताते हैं कि उन पर कोई वोल्टेज लगाया गया है, बल्कि इसका मूल्य भी बताते हैं। analogRead() फ़ंक्शन का उपयोग करके, हम किसी एक पिन पर लागू वोल्टेज को पढ़ सकते हैं।

यह फ़ंक्शन 0 और 1023 के बीच की संख्या लौटाता है, जो 0 और 5 वोल्ट के बीच वोल्टेज का प्रतिनिधित्व करता है। उदाहरण के लिए, यदि पिन नंबर 0 पर 2.5 V का वोल्टेज लगाया जाता है, तो AnalogRead(0) 512 मान देता है।

analogRead() function का सिंटैक्स :

```
analogRead(pin);
```



pin: एनालॉग इनपुट पिन की संख्या (ज्यादातर बोर्ड पर 0 से 5, मिनी और नैनो पर 0 से 7, मेगा पर 0 से 15)

**उदाहरण:**

```
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
}
```

## **Interface serial port with Embedded development Board:**

### **Serial communication in Arduino:**

Arduino में, "सीरियल कम्युनिकेशन" का अर्थ श्रृंखला में डेटा को किसी अन्य डिवाइस में स्थानांतरित करना है। Arduino में, हम Arduino के USB प्लग और TX/RX पिन के माध्यम से कंप्यूटर या कुछ अन्य उपकरणों के साथ सीरियल संचार कर सकते हैं। Arduino में सीरियल संचार उन पिनो के माध्यम से किया जाता है जो इस उद्देश्य के लिए समर्पित हैं। सीरियल संचार डेटा के प्रत्येक बाइट को अन्य डिवाइस या कंप्यूटर पर स्थानांतरित करना सुनिश्चित करता है ।

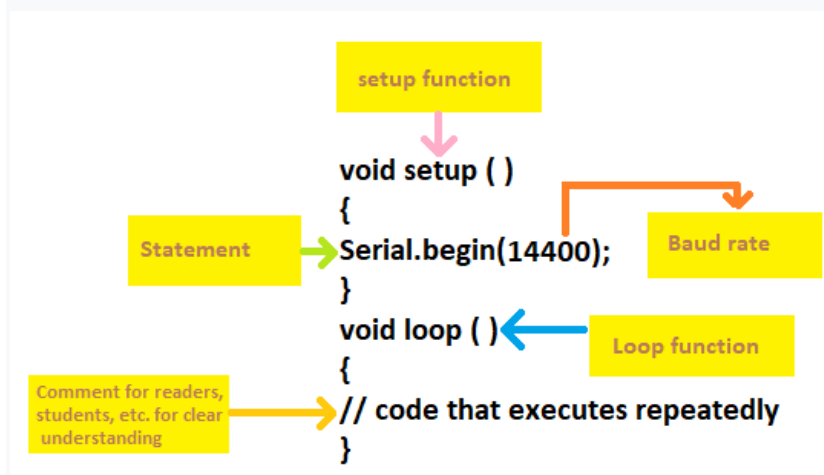
Arduino Uno में, दो पिन; पिन 0 और पिन 1 को UART (यूनिवर्सल एसिंक्रोनस रिसीवर ट्रांसमीटर) और USART (यूनिवर्सल सिंक्रोनस एसिंक्रोनस रिसीवर ट्रांसमीटर) के रूप में जाने जाने वाले सीरियल कम्युनिकेशन के लिए असाइन किया गया है और उन्हें Tx/Rx पिन के रूप में भी जाना जाता है। ये पिन 3.3 वोल्ट या 5 वोल्ट पर संचालित होते हैं, इसलिए उन्हें RS232 सीरियल पोर्ट से जोड़ने की अनुशंसा नहीं की जाती है क्योंकि यह 12 वोल्ट पर संचालित होता है जो Arduino बोर्ड को नुकसान पहुंचा सकता है, इसके अलावा, सीरियल कम्युनिकेशन यूएसबी प्लग की सहायता से कंप्यूटर के माध्यम से भी किया जा सकता है।

सीरियल संचार का आउटपुट सीरियल मॉनिटर पर देखा जा सकता है, जिसे टूल के ड्रॉप-डाउन मेनू में "सीरियल मॉनिटर" पर क्लिक करके "Arduino IDE" में एक्सेस किया जा सकता है। कंप्यूटर के साथ सीरियल संचार के लिए, Arduino को USB केबल के माध्यम से कंप्यूटर से कनेक्ट करें |

Arduino के विभिन्न बिल्ट-इन फंक्शन हैं लेकिन सीरियल संचार के लिए सबसे अधिक उपयोग किये जाने वाले फंक्शन निम्नलिखित हैं:

**Serial.begin(speed)** : इस फंक्शन का उपयोग विशिष्ट बॉड दर पर डेटा स्थानांतरित करने की गति को सेट करने के लिए किया जाता है

Arduino में डिफॉल्ट बॉड दर 9600 बीपीएस (बिट्स प्रति सेकंड) है। हम अन्य बॉड दरों को भी निर्दिष्ट कर सकते हैं, जैसे कि 4800, 14400, 38400, 28800, आदि।



**Serial.read()** : यह फंक्शन दुसरे कनेक्टेड मशीन से Arduino में आने वाले सीरियल डेटा को पढ़ता है। यहाँ int डेटा प्रकार का उपयोग किया जाता है। यह आने वाले सीरियल डेटा का पहला डेटा बाइट लौटाता है। सीरियल पोर्ट पर कोई डेटा उपलब्ध नहीं होने पर यह -1 भी लौटाता है।

**Serial.print()** : यह फंक्शन डेटा को ASCII में परिवर्तित करता है जो मनुष्यों द्वारा आसानी से पढ़ा जा सकता है और इसे सीरियल मॉनिटर पर प्रिंट करता है।

#### Example 1:

1. `Serial.print(15.452732)`

Output: 15.45

यह प्रिंटर को सिंगल कैरेक्टर के रूप में बाइट भेजता है। Arduino में, Serial.print() का उपयोग करने वाले स्ट्रिंग्स और कैरेक्टर जैसे हैं वैसे ही भेजे जाते हैं।

#### Example 2:

1. Serial.print("Hello Arduino")

Output: "Hello Arduino"

**Serial.println():** यह फ़ंक्शन प्रिंट () के समान काम करता है, लेकिन इसके अलावा, यह एक नई लाइन जोड़ता है।

**Serial.flush():** यह फ़ंक्शन आउटगोइंग सीरियल डेटा के प्रसारण को पूरा करना सुनिश्चित करता है।

#### उदहारण:

```
void setup ()
{
  Serial.begin ( 4800);
}
void loop ()
{
  Serial.print(" Hello");
  delay(1000);
  Serial.println("Arduino");
  delay ( 1500);
}
```

हम Arduino के USB प्लग के माध्यम से कंप्यूटर के साथ सीरियल संचार के लिए Serial.begin() फ़ंक्शन का उपयोग करेंगे, और डेटा को 4800 बॉड दर पर स्थानांतरित करने की गति निर्धारित करेंगे। फिर हम सीरियल मॉनिटर पर "Hello" टेक्स्ट को प्रिंट करने के लिए Serial.print("Hello") फ़ंक्शन का उपयोग करेंगे, फिर 1.5 सेकंड्स का डिले के लिये delay() फ़ंक्शन का उपयोग करेंगे | Serial.println("Arduino") फ़ंक्शन "Arduino" print करने के बाद एक नया लाइन जोड़ेगा, जब loop फ़ंक्शन जब अगली बार एकसीक्यूट होगा तो "Hello Arduino" नये लाइन में print होगा |

## Make a basic circuit of embedded development board:

### Blinking an LED:

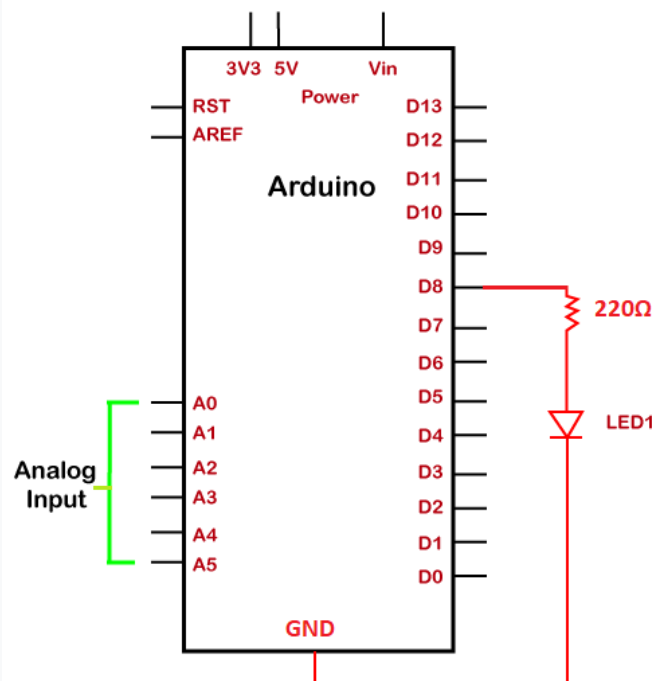
यह Arduino का उपयोग करके बनाई गई सरल बेसिक प्रोजेक्ट है। एलईडी (लाइट एमिटिंग डायोड) एक इलेक्ट्रॉनिक उपकरण है, जो अपने टर्मिनलों से करंट गुजरने पर प्रकाश उत्सर्जित करता है। एलईडी का उपयोग विभिन्न अनुप्रयोगों में किया जाता है। इसका उपयोग विभिन्न इलेक्ट्रॉनिक उपकरणों में ON/OFF संकेतक के रूप में भी किया जाता है।

इस परियोजना में, हम Arduino बोर्ड पर LED को डिजिटल पिन से जोड़ेंगे। एलईडी एक साधारण प्रकाश के रूप में काम करेगा जिसे निर्दिष्ट अवधि के लिए चालू और बंद किया जा सकता है।

एलईडी के ब्लिंकिंग में उपयोग किए जाने वाले घटक नीचे सूचीबद्ध हैं:

1. 1xArduino UNO board.
2. 1 x Breadboard
3. 2 x Jump wires
4. 1 x LED
5. 1 x Resistor of 220 Ohm

हम 470 ओम तक के किसी भी मान के प्रतिरोधक का उपयोग कर सकते हैं। हम अपनी सर्किट आवश्यकताओं के आधार पर प्रतिरोधकों के अन्य मूल्यों का भी उपयोग कर सकते हैं। आमतौर पर, मान स्वीकार्य फॉरवर्ड करंट से अधिक नहीं होना चाहिए।



सर्किट डायग्राम स्पष्ट रूप से UNO बोर्ड के पिनआउट को दर्शाती है। यह बोर्ड से जुड़े एलईडी और प्रतिरोध को भी प्रदर्शित करता है।

LED ब्लिंकिंग के लिए प्रोग्राम :

```
void setup ()
{
  pinMode ( 8, OUTPUT); // to set the OUTPUT mode of pin number 8.
}
void loop ()
{
  digitalWrite (8, HIGH);
  delay(1000); // 1 second = 1 x 1000 milliseconds
  digitalWrite (8, LOW);
  delay(500); // 0.5 second = 0.5 x 1000 milliseconds
}
```

### **Explain interfacing DC Motor with programming**

डीसी मोटर को सबसे सरल मोटर माना जाता है, जिसमें घरों से लेकर उद्योगों तक के विभिन्न अनुप्रयोग होते हैं। उदाहरण में कार, इलेक्ट्रिक वाहनों का इलेक्ट्रिक विंडो, लिफ्ट आदि शामिल है।

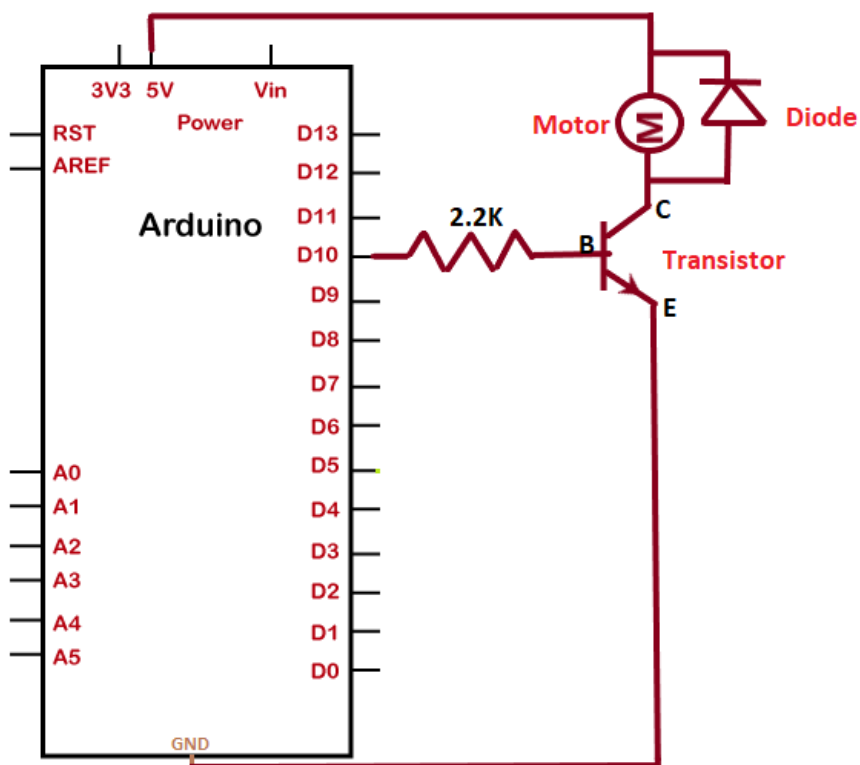
डीसी मोटर्स का सिद्धांत विद्युत चुम्बकीय प्रेरण पर आधारित है। इसका अर्थ है कि मोटर का घूर्णन चुम्बकीय क्षेत्र द्वारा उत्पन्न बल पर निर्भर करता है। यह विद्युत ऊर्जा को यांत्रिक ऊर्जा में परिवर्तित करता है। इन मोटरों को डायरेक्ट करंट से संचालित किया जा सकता है।

**डायोड, ट्रांजिस्टर और रेसिस्टर का उपयोग करके Arduino बोर्ड के साथ DC मोटर का सरल कनेक्शन:**

कनेक्शन के लिए आवश्यक कंपोनेंट्स :

- Arduino UNO R3 board
- Breadboard
- A Resistor of 2.2K Ohm
- Transistor (NPN)
- Diode
- DC Motor

- Jump wires



```
int PinOFmotor = 10;
// PIN 10 of the Arduino is initialized to the variable
// the pin must be a PWM pin
void setup()
{
  pinMode(PinOFmotor, OUTPUT);
}
void loop()
{
  digitalWrite(PinOFmotor, HIGH);
  delay(1000);
  digitalWrite(PinOFmotor, LOW);
  delay(1000);
}
```

कनेक्शन स्थापित करने के चरण नीचे सूचीबद्ध हैं:

1. Arduino बोर्ड के 10 (PWM) पिन करने के लिए रेसिस्टर के एक छोर को कनेक्ट करें।
2. रेसिस्टर के दूसरे छोर को ट्रांजिस्टर के मध्य पिन से कनेक्ट करें।
3. ट्रांजिस्टर के एक सिरे के टर्मिनल को Arduino के GND पिन से और दूसरे सिरे के टर्मिनल को डायोड से कनेक्ट करें।
4. डायोड के बैंड फेसिंग टर्मिनल को Arduino बोर्ड के 5V पिन से कनेक्ट करें।
5. डीसी मोटर के एक सिरे वाले टर्मिनल को डायोड के बैंड फेसिंग टर्मिनल से कनेक्ट करें।
6. डीसी मोटर के दूसरे सिरे के टर्मिनल को डायोड के दूसरे सिरे से जोड़ें।

**बोर्ड में कोड अपलोड करने के चरण:**

1. Arduino IDE खोलें।
2. Tools -> Board -> Arduino UNO से बोर्ड के प्रकार का चयन करें।
3. Tools -> Port -> COM से पोर्ट का चयन करें।
4. स्केच को कनेक्शन आरेख में अपलोड करें

## UNIT- IV

### VLSI Technology – Introduction and history

**VLSI full form - VERY LARGE-SCALE INTEGRATION (VLSI)** होता है जिसका हिंदी में मतलब "बहुत बड़े पैमाने पर एकीकरण (VLSI)" होता है

VLSI एक तरह का Integrated Circuit (IC) होता है, जिसमें काफी सारे transistors लगे होते हैं। आज से काफी साल पहले computer में जो ट्रांजिस्टर इस्तेमाल होते थे, उन transistor की संख्या काफी कम होती थी जिसकी वजह से computer उतना तेज़ काम नहीं करता था जितना आज के समय का computer कर पाता है। ज्यादा transistor के होने से processor ज्यादा operations कम समय में कर पाएगा और जिसकी वजह से कंप्यूटर की performance बढ़ जाती है।

एक VLSI में हजारों transistors को एक single silicon semiconductor chip में रखा जाता है। VLSI 1970 की दसक से बनने शुरू हुए थे और आज के समय भी VLSI का इस्तेमाल काफी ज्यादा किया जाता है।

VLSI technology के बाद भी एक और टेक्नोलॉजी आई थी जिसे ULSI कहा जाता है, जिसका पूरा नाम Ultra Large Scale Integration है और इसमें transistors की संख्या VLSI के मुकाबले काफी ज्यादा होती है। VLSI कंप्यूटर के चौथे generation में आता है।

VLSI की मदद से कंप्यूटर के processor, RAM, ROM जैसे Chips बनाये जाते हैं। जितने ज्यादा छोटे और ज्यादा transistors मौजूद होंगे उतनी ज्यादा इनकी power और calculations करने के speed ज्यादा होगी।

माइक्रोप्रोसेसर एक VLSI उपकरण का उदाहरण है।

पिछले कुछ दशकों में Electronics industry में बहुत तेजी से Growth देखने को मिली है जिसका मुख्य कारण तेजी से बढ़ी technologies और सिस्टम के द्वारा तैयार की गयी applications हैं जैसे ही Market में VLSI आया वैसे ही high-performance computing, controls, दूरसंचार, image and video processing, and consumer electronics में बहुत तेजी से बढ़त देखने को मिली।

VLSI की शुरुआत से पहले ज्यादातर काम ICs तक सीमित थे। एक Electronic circuit में एक CPU, ROM, RAM और अन्य glue logic शामिल होते थे।

आज से लगभग 45 साल पहले चिप में पाए जाने वाले Transistor की संख्या इसकी उच्चतम संख्या से 10,000 से कम थी।

VLSI के द्वारा आप इन ICs डिजाइनरों को एक chip में जोड़ सकते हैं।

VLSI में आपको Low bit-rate video और High-resolution, cellular communication,



Processing power और portability की सुविधा मिलती है. भविष्य में VLSI में और बढ़त देखने को मिल सकती है |

## VLSI का इतिहास

Transistor की खोज 1920 के दशक में हुई. इसकी खोज से पहले कई लोग ऐसा उपकरण बनाना चाहते थे जो current को ठोस अवस्था वाले Diode में बदलना चाहते थे फिर उसे Triode में बदलना चाहते थे। उनका ये काम Transistor ने किया.

जब Radar detectors के रूप में सिलिकॉन और जर्मेनियम क्रिस्टल के उपयोग के निर्माण और सिद्धांत में सुधार किया

इसे Second World War के बाद तेजी से पहचान और सफलता मिली.

1947 में Bell labs ने पहले Transistor के आविष्कार किया और साथ ही, Electronics का क्षेत्र वैक्यूम ट्यूबों से solid-state devices में बदल गया.

1950 के दशक के कई Electrical engineers ने Circuit के उजागर भविष्य की निर्माण की संभावनाओं को देखा लेकिन बाद में जैसे-जैसे Circuit की जटिलता बढ़ी, समस्याएं पैदा होना शुरू हो गयी.

जिसके कारण 1960 के दशक के प्रारंभ में small-scale integration (SSI) और 1960 के दशक के **medium-scale integration (MSI)** की खोज हुई.

VLSI का निर्माण MOS Transistor की जगह लेने के लिए किया गया था. इसका निर्माण 1959 में Mohamed M. Atalla और Dawon Kahng द्वारा Bell labs में किया गया था.

Mohamed M. Atalla ने 1960 में पहली बार MOS integrated circuit chip का idea सुझाया। 1961 ने Dawon Kahng द्वारा भी इस पर काम करना शुरू किया.

इसे integrated circuits के लिए Useful बनाया. पहले Semiconductor chips में दो Transistor होते थे. बाद में जरूरत के अनुसार अधिक Transistor जोड़े गए.

## Advantages of VLSI:

1. सर्किट के आकार को कम करता है।
2. उपकरणों की प्रभावी लागत को कम करता है।
3. सर्किट की ऑपरेटिंग गति को बढ़ाता है
4. असतत घटकों की तुलना में कम शक्ति की आवश्यकता होती है।
5. उच्च विश्वसनीयता
6. अपेक्षाकृत छोटे क्षेत्र पर कब्जा करता है।

## VLSI का उपयोग

आज के समय में VLSI चिप्स का व्यापक रूप से इंजीनियरिंग की विभिन्न शाखाओं में उपयोग किया जाता है जैसे:

1. Voice and Data Communication networks
2. Digital Signal Processing
3. Computers
4. Commercial Electronics
5. Automobiles
6. Medicine and many more.

## CMOS Fabrication Process:

एक जमाना था, जब कंप्यूटर इतने विशाल आकार के होते थे कि उन्हें स्थापित करने के लिए आसानी से एक कमरे की जगह की आवश्यकता होती थी। लेकिन आज वे इतने विकसित हैं कि हम उन्हें आसानी से नोटबुक के रूप में भी ले जा सकते हैं। यह इंटीग्रेटेड सर्किट के कारण संभव हो पाया। इंटीग्रेटेड सर्किट में, बड़ी संख्या में सक्रिय और निष्क्रिय तत्व अपने इंटरकनेक्शन के साथ एक छोटे सिलिकॉन वेफर पर विकसित होते हैं। इस तरह के सर्किट के उत्पादन के लिए अपनाई जाने वाली बुनियादी प्रक्रियाओं में एपिटैक्सियल ग्रोथ, मास्कड इम्प्युरिटी डीफ्यूजन, ऑक्साइड वृद्धि, और ऑक्साइड नक्काशी, पैटर्न बनाने के लिए फोटोलिथोग्राफी का उपयोग करना शामिल है।

वेफर के ऊपर के घटकों में रेसिस्टर्स, ट्रांजिस्टर, डायोड, कैपेसिटर आदि शामिल हैं... IC के निर्माण के लिए सबसे जटिल तत्व ट्रांजिस्टर है। ट्रांजिस्टर विभिन्न प्रकार के होते हैं जैसे CMOS, BJT, FET। हम आवश्यकताओं के आधार पर IC पर लागू की जाने वाली ट्रांजिस्टर तकनीक का प्रकार चुनते हैं।

कम बिजली अपव्यय आवश्यकता के लिए ट्रांजिस्टर को लागू करने के लिए CMOS तकनीक का उपयोग किया जाता है। यदि हमें तेज सर्किट की आवश्यकता है तो BJT का उपयोग

करके IC पर ट्रांजिस्टर लागू किए जाते हैं। IC के रूप में CMOS ट्रांजिस्टर का निर्माण तीन अलग-अलग तरीकों से किया जा सकता है।

1. **N-Well** - पी-टाइप सबस्ट्रेट पर एन-टाइप डिफ्यूजन किया जाता है
2. **P-Well** - एन-टाइप सबस्ट्रेट पर पी-टाइप डिफ्यूजन किया जाता है
3. **Twin Well** - NMOS और PMOS ट्रांजिस्टर एक सबस्ट्रेट के बजाय एक एपिटैक्सियल ग्रोथ बेस पर एक साथ प्रसार द्वारा वेफर पर विकसित होते हैं।

एन-वेल/पी-वेल तकनीक का उपयोग करते हुए निर्मित करते समय सीएमओएस फैब्रिकेशन प्रक्रिया प्रवाह बीस बुनियादी फैब्रिकेशन चरणों का उपयोग करके आयोजित किया जाता है।

### Making of CMOS using N well

Step 1. पहले हम निर्माण के लिए आधार के रूप में एक सबस्ट्रेट चुनते हैं। एन-वेल के लिए, एक पी-टाइप सिलिकॉन सबस्ट्रेट का चयन किया जाता है।



©Elprocus.com

Step 2. ऑक्सीकरण:- एन-प्रकार की अशुद्धियों का चयनात्मक प्रसार  $\text{SiO}_2$  का उपयोग एक बाधा के रूप में किया जाता है जो सबस्ट्रेट के संदूषण के खिलाफ वेफर के कुछ हिस्सों की रक्षा करता है।  $\text{SiO}_2$  को ऑक्सीकरण प्रक्रिया द्वारा निर्धारित किया जाता है, जिसमें सबस्ट्रेट को उच्च गुणवत्ता वाले ऑक्सीजन और हाइड्रोजन के साथ लगभग  $10000^\circ\text{C}$  पर ऑक्सीकरण कक्ष में उजागर किया जाता है।



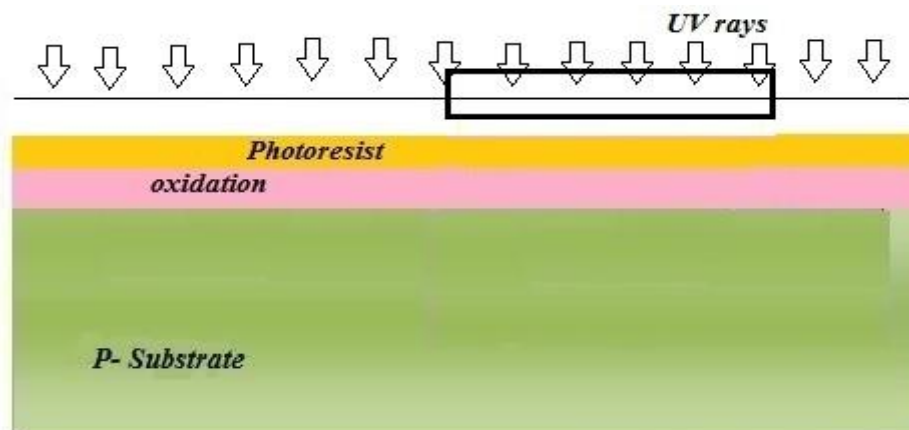
©Elprocus.com

Step 3. फोटोरेसिस्ट का बढ़ना:- इस स्तर पर चयनात्मक नक्काशी (Selective etching ) की अनुमति देने के लिए, SiO<sub>2</sub> परत को फोटोलिथोग्राफी प्रक्रिया के अधीन किया जाता है। इस प्रक्रिया में, वेफर को एक सहज पायस की एक समान फिल्म के साथ लेपित किया जाता है।



©Elprocus.com

Step 4. मास्किंग:- यह चरण फोटोलिथोग्राफी प्रक्रिया की निरंतरता है। इस चरण में, एक स्टैंसिल का उपयोग करके खुलेपन का वांछित पैटर्न बनाया जाता है। इस स्टैंसिल का उपयोग फोटोरेसिस्ट के ऊपर मास्क के रूप में किया जाता है। सब्सट्रेट को अब यूवी किरणों के संपर्क में लाया जाता है, जिससे मास्क के उजागर क्षेत्रों के नीचे मौजूद फोटोरेसिस्ट पॉलीमराइज़ हो जाता है।



©Elprocus.com

Step 5. अनएक्सपोज्ड फोटोरेसिस्ट को हटाना:- मास्क को हटा दिया जाता है और फोटोरेसिस्ट के अनएक्सपोज्ड क्षेत्र को ट्राइक्लोरोइथाइलीन जैसे रसायन का उपयोग करके वेफर विकसित करके गला दिया जाता है।



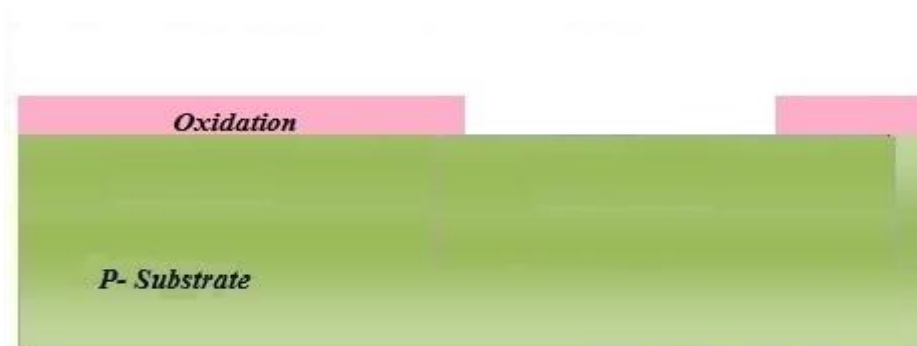
©Elprocus.com

Step 6. Etching: वेफर को हाइड्रोफ्लोरिक एसिड के एचिंग के घोल में डुबोया जाता है, जो उन क्षेत्रों से ऑक्साइड को हटा देता है जहाँ से डोपेंट को फैलाना होता है।



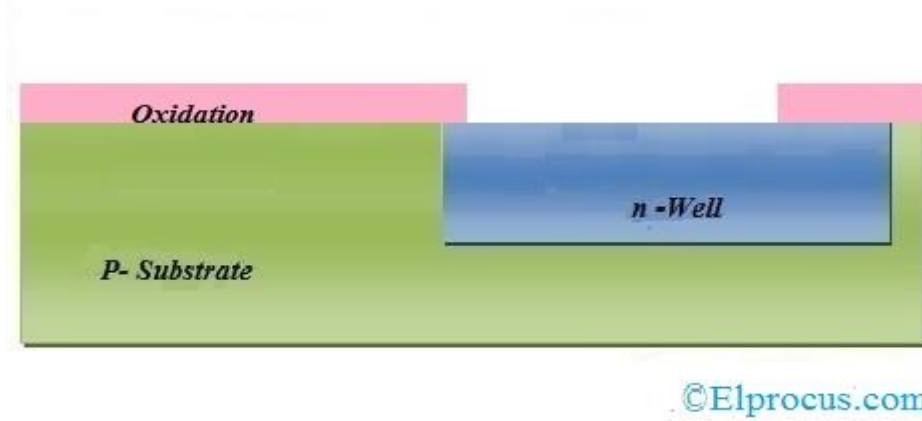
©Elprocus.com

Step 7. संपूर्ण फोटोरेसिस्ट परत को हटाना: एचिंग प्रक्रिया के दौरान,  $\text{SiO}_2$  के वे हिस्से जो फोटोरेसिस्ट परत द्वारा संरक्षित होते हैं, प्रभावित नहीं होते हैं। फोटोरेसिस्ट मास्क को अब एक रासायनिक विलायक (गर्म  $\text{H}_2\text{SO}_4$ ) से हटा दिया जाता है।

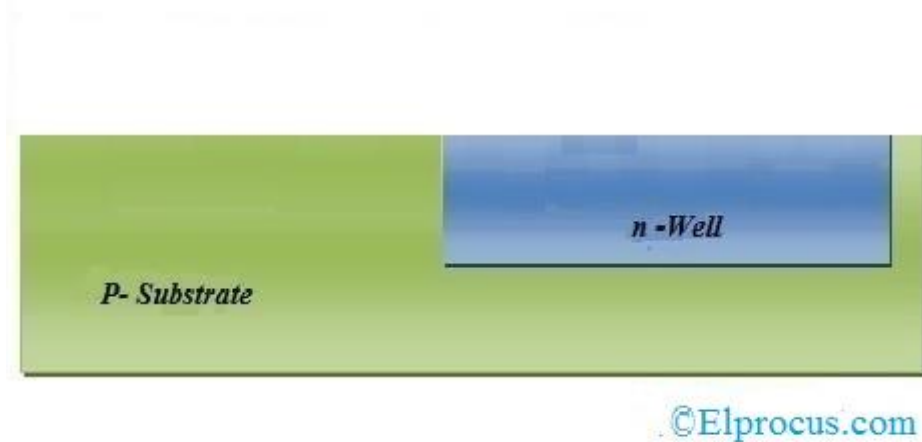


©Elprocus.com

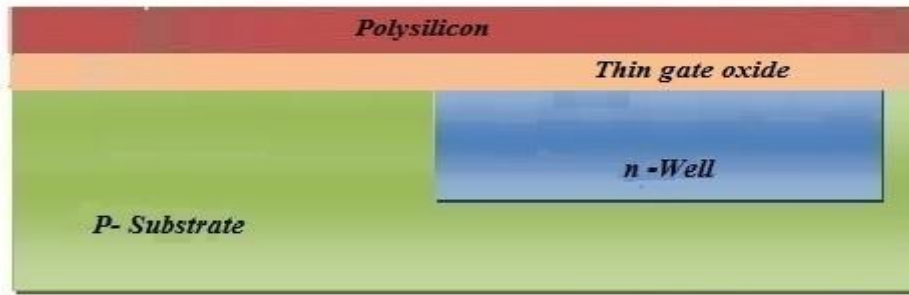
Step 8. एन-वेल का निर्माण: एन-टाइप अशुद्धियों को उजागर क्षेत्र के माध्यम से पी-टाइप सब्सट्रेट में फैलाया जाता है और इस प्रकार एन-वेल का निर्माण होता है।



Step 9. SiO<sub>2</sub> को हटाना: SiO<sub>2</sub> की परत अब हाइड्रोफ्लोरिक एसिड का उपयोग करके हटा दी जाती है।

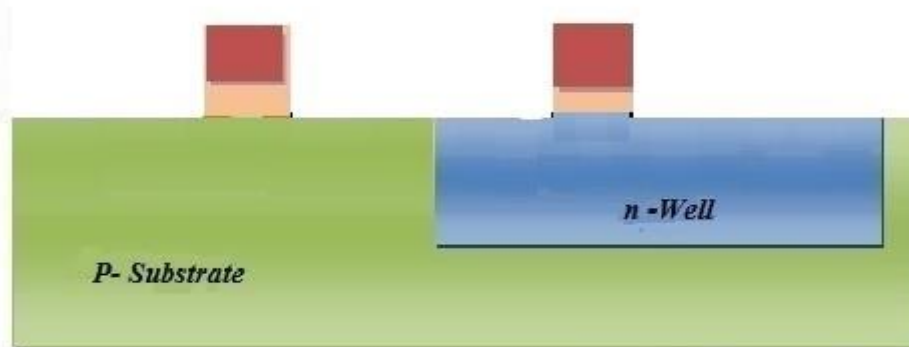


Step 10. पॉलीसिलिकॉन का जमाव: CMOS ट्रांजिस्टर के गेट के गलत संरेखण से अवांछित धारिता बनती है जो सर्किट को नुकसान पहुंचा सकती है। इसलिए इसे रोकने के लिए "स्व-संरेखित गेट प्रक्रिया" को प्राथमिकता दी जाती है जहां आयन आरोपण का उपयोग करके सोर्स और ड्रेन के निर्माण से पहले गेट क्षेत्र बनते हैं।



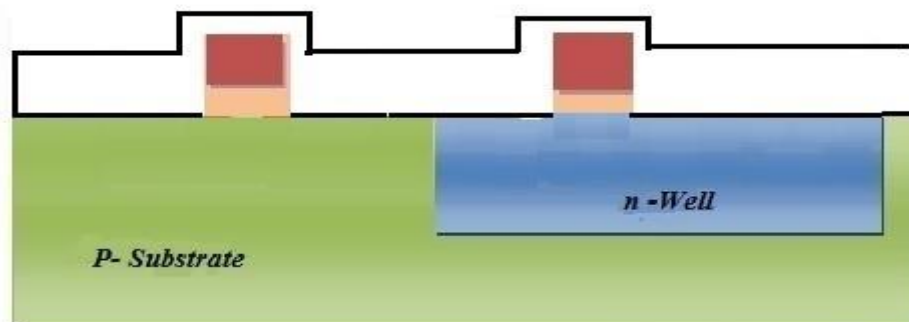
©Elprocus.com

Step 11. गेट क्षेत्र का गठन: NMOS और PMOS ट्रांजिस्टर के लिए गेट के गठन के लिए आवश्यक दो क्षेत्रों को छोड़कर पॉलीसिलिकॉन के शेष हिस्से को हटा दिया जाता है।



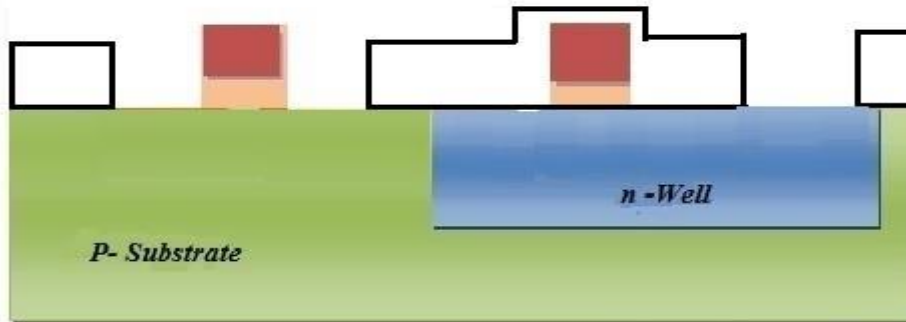
©Elprocus.com

Step 12. ऑक्सीकरण प्रक्रिया: वेफर के ऊपर एक ऑक्सीकरण परत जमा होती है जो आगे प्रसार और धातुकरण प्रक्रियाओं के लिए एक ढाल के रूप में कार्य करती है।



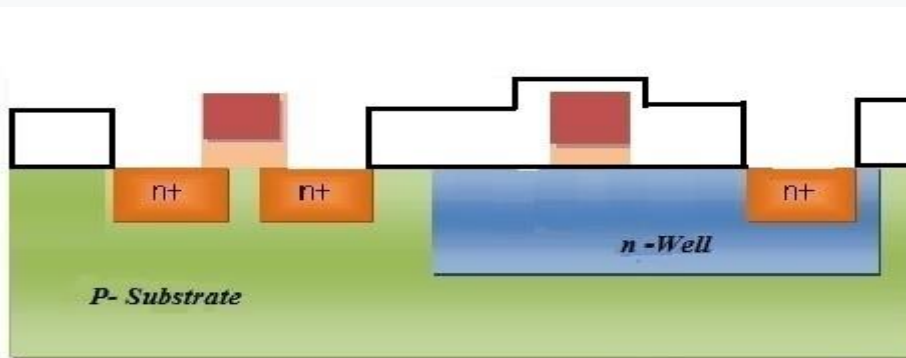
©Elprocus.com

Step 13. मास्किंग और डिफ्यूजन: मास्किंग प्रक्रिया का उपयोग करके एन-टाइप अशुद्धियों के प्रसार के लिए क्षेत्र बनाने के लिए छोटे अंतराल बनाए जाते हैं।



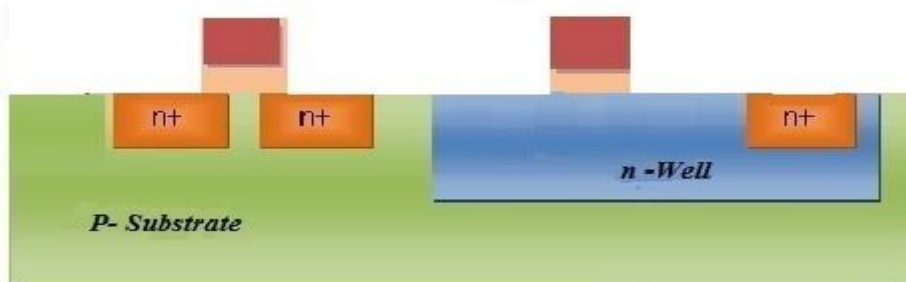
©Elprocus.com

डिफ्यूजन प्रक्रिया का उपयोग करते हुए NMOS के टर्मिनलों के निर्माण के लिए तीन n+ क्षेत्र विकसित किए गए हैं।



©Elprocus.com

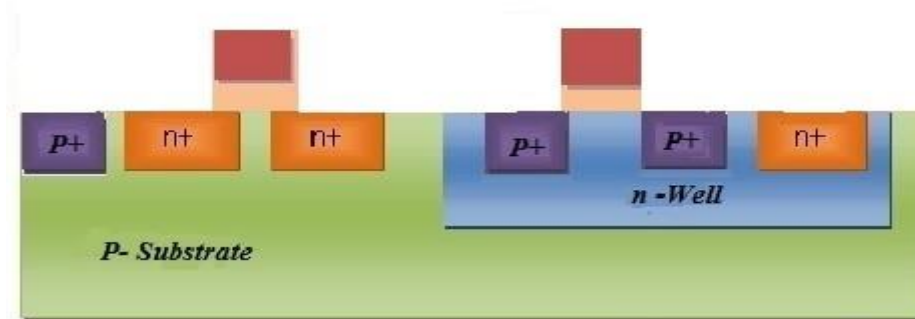
Step 14. ऑक्साइड को हटाना: ऑक्साइड परत को हटा दिया जाता है।



©Elprocus.com

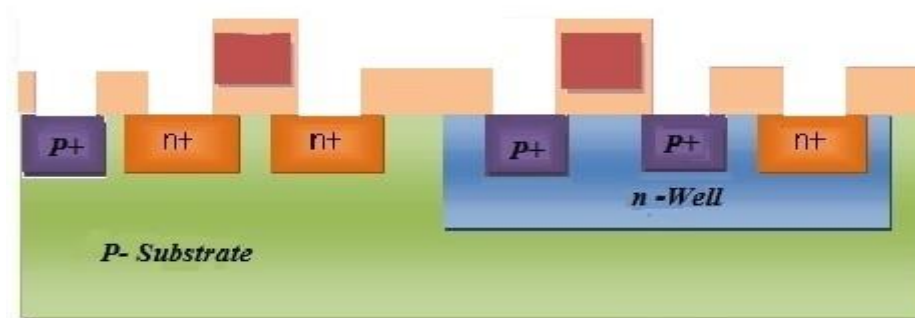


Step 15. पी-टाइप डिफ्यूजन: PMOS के टर्मिनल बनाने के लिए एन-टाइप डिफ्यूजन के समान पी-टाइप डिफ्यूजन किया जाता है।



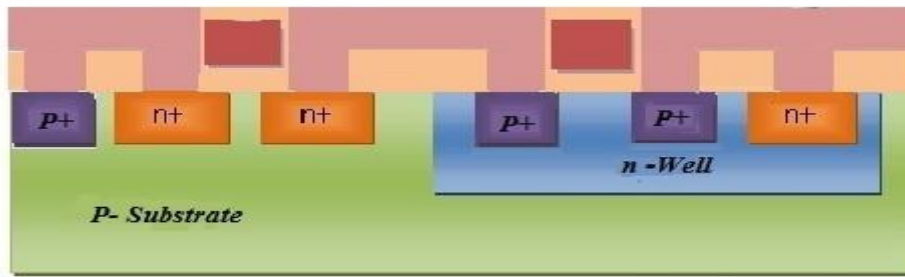
©Elprocus.com

Step 16. थिक फील्ड ऑक्साइड बिछाना: धातु के टर्मिनलों को बनाने से पहले वेफर के उन क्षेत्रों के लिए एक सुरक्षात्मक परत बनाने के लिए एक मोटी फील्ड ऑक्साइड बिछाई जाती है, जहां किसी टर्मिनल की आवश्यकता नहीं होती है।



©Elprocus.com

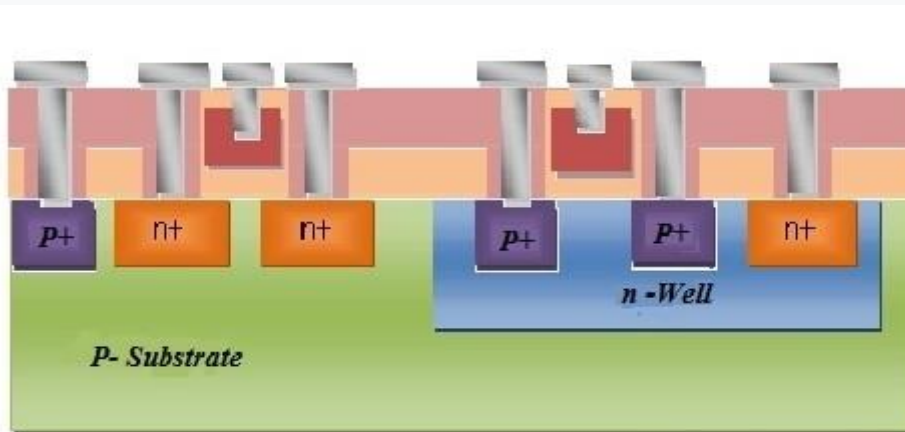
Step 17. धातुकरण: इस चरण का उपयोग धातु टर्मिनलों के निर्माण के लिए किया जाता है जो इंटरकनेक्शन प्रदान कर सकते हैं। एल्युमिनियम पूरी वेफर पर फैला हुआ है।



©Elprocus.com

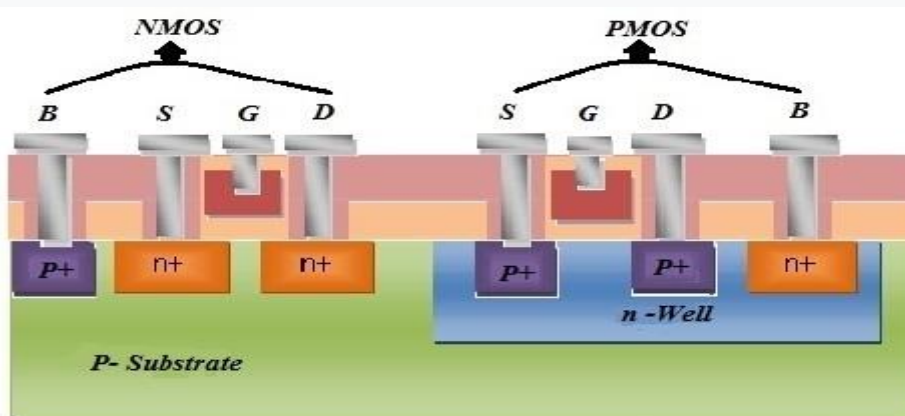
Step 18. अतिरिक्त धातु को हटाना: अतिरिक्त धातु को वेफर से हटा दिया जाता है।

Step 19. टर्मिनलों का निर्माण: अतिरिक्त धातु को हटाने के बाद बनने वाले अंतराल में इंटरकनेक्शन के लिए टर्मिनल बनते हैं।



©Elprocus.com

Step 20. टर्मिनल नाम निर्दिष्ट करना: NMOS और PMOS ट्रांजिस्टर के टर्मिनलों को नाम दिया जाता है।



©Elprocus.com

## Making of CMOS using P-well

पी-वेल प्रक्रिया एन वेल प्रक्रिया के समान है सिवाय इसके कि यहां एन-टाइप सबस्ट्रेट का उपयोग किया जाता है और पी-टाइप डिफ्यूजन किया जाता है। सादगी के लिए आमतौर पर, एन वेल प्रोसेस को प्राथमिकता दी जाती है।

## Twin Tube Fabrication of CMOS

ट्विन-ट्यूब प्रक्रिया का उपयोग करके कोई भी पी और एन-टाइप के उपकरणों के लाभ को नियंत्रित कर सकता है। ट्विन-ट्यूब विधि का उपयोग करके CMOS के निर्माण में शामिल विभिन्न चरण इस प्रकार हैं:

- एक हल्का डोप किया गया एन या पी-टाइप सबस्ट्रेट लिया जाता है और एपिटैक्सियल परत का उपयोग किया जाता है। एपिटैक्सियल परत चिप में लैच-अप समस्या की रक्षा करती है।
- मापी गई मोटाई और सटीक डोपेंट एकाग्रता के साथ उच्च शुद्धता वाली सिलिकॉन परतें बनाई जाती हैं।
- पी और एन वेल के लिए ट्यूबों का निर्माण।
- डिफ्यूजन प्रक्रियाओं के दौरान संदूषण से सुरक्षा के लिए पतला ऑक्साइड निर्माण।
- आयन आरोपण विधियों का उपयोग करके सोर्स और ड्रेन का निर्माण किया जाता है।
- धातु संपर्कों के लिए भाग बनाने के लिए कटौती की जाती है।
- धातु संपर्कों को खींचने के लिए धातुकरण किया जाता है

## Stick Diagram:

स्टिक डायग्राम एक प्रकार का डायग्राम होता है जिसका उपयोग ट्रांजिस्टर सेल के लेआउट की योजना बनाने के लिए किया जाता है। स्टिक डायग्राम में उपकरणों और कंडक्टरों का प्रतिनिधित्व करने के लिए "स्टिक" या रेखाओं का उपयोग किया जाता है।

स्टिक डायग्राम चिप लेआउट का कार्टून होता है। वे लेआउट के सटीक मॉडल नहीं होते हैं। कलर कोड के उपयोग के माध्यम से परत की जानकारी देने के लिए स्टिक डायग्राम का उपयोग किया जाता है

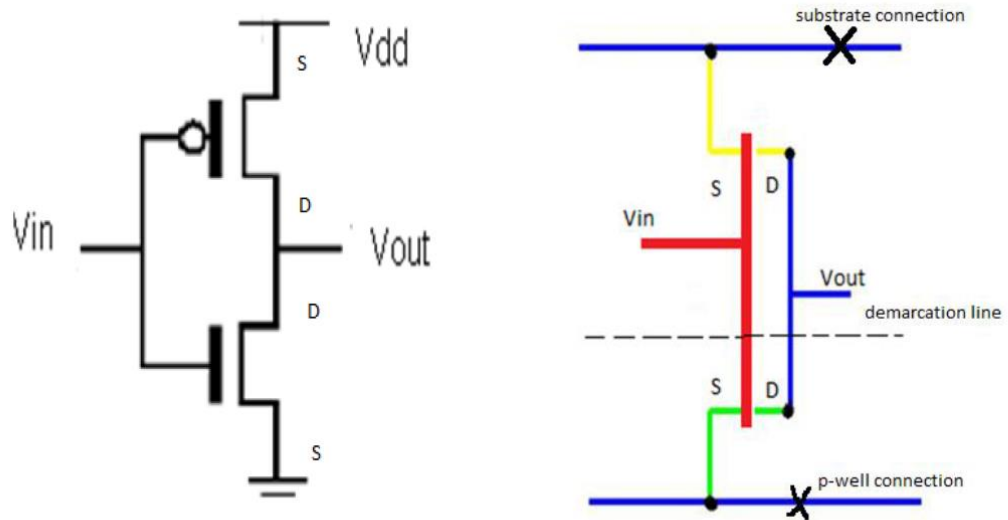
उदाहरण के लिए NMOS डिजाइन में।

- \* एन-डीफ्यूजन के लिए हरा रंग
- \* पॉली सिलिकॉन के लिए लाल रंग
- \* धातु के लिए नीला रंग
- \* प्रत्यारोपण के लिए पीला रंग
- \* संपर्क क्षेत्रों के लिए काला रंग

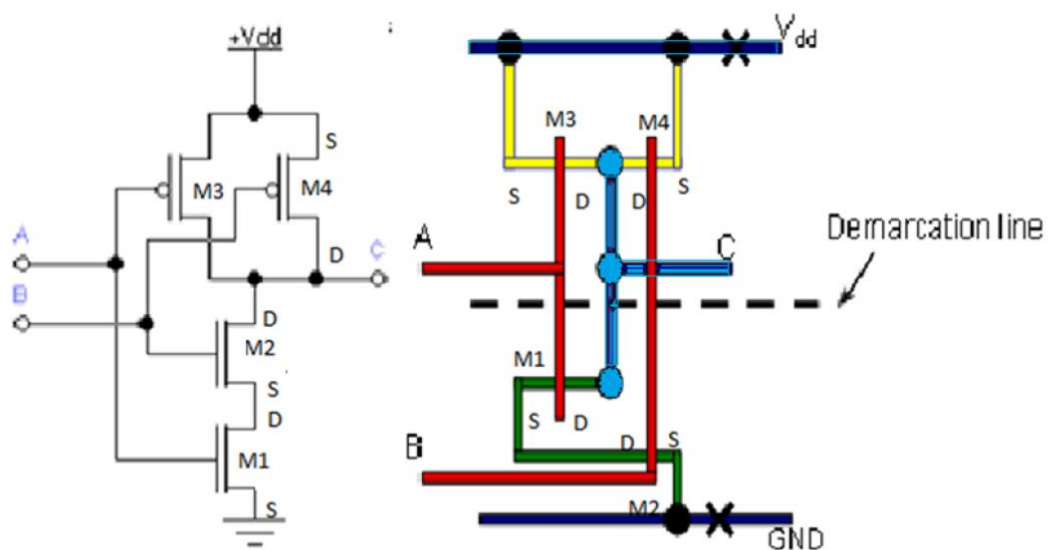
स्टिक डायग्राम बनाने के नियम:

1. जब एक ही प्रकार की दो या दो से अधिक स्टिक एक दुसरे को पार या स्पर्श करते हैं तो यह विद्युत संपर्क को दर्शाता है ।
2. जब अलग-अलग प्रकार की दो या दो से अधिक स्टिक दूसरे को पार करती हैं या स्पर्श करती हैं तो कोई विद्युत संपर्क नहीं होता है। (यदि संपर्क की आवश्यकता है तो स्पष्ट रूप से दिखाएं)
3. जब एक पॉली डीफ्यूजन को पार करती है तो यह MOSFET का प्रतिनिधित्व करती है। यदि संपर्क दिखाया गया है तो यह ट्रांजिस्टर नहीं है।
4. PMOS उपकरणों को पुल अप भाग (सीमांकन रेखा के ऊपर) और NMOS उपकरणों को CMOS डिजाइन शैली में पुल डाउन भाग (सीमांकन रेखा के नीचे) में रखा जाना चाहिए।
5. p और n-टाइप के उपकरणों के बीच अंतर करने के लिए, दो प्रकारों के बीच में सीमांकन रेखा का उपयोग किया जाता है।
6. PMOS उपकरणों और NMOS उपकरणों के लिए क्रमशः n-well और p-well संपर्क बनाने के लिए Vdd और Vss रेल पर क्रॉस मार्क का उपयोग किया जाता है।

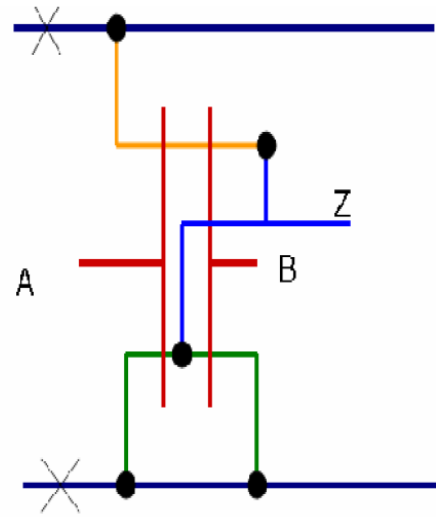
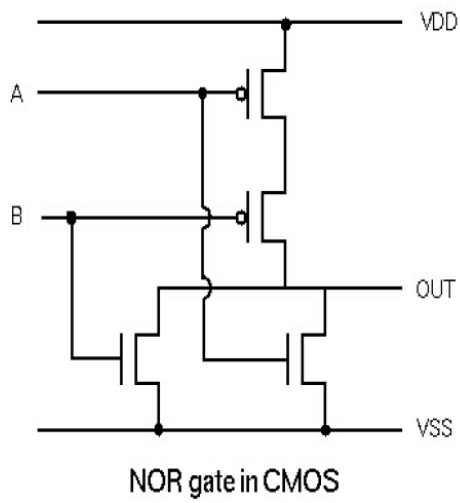
# CMOS INVERTER



# 2 INPUT – CMOS NAND GATE



# NOR gate



# UNIT- V

## VLSI Design - VHDL Introduction

VHDL stands for very high-speed integrated circuit hardware description language. It is a programming language used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC program.

### Describing a Design

In VHDL an entity is used to describe a hardware module. An entity can be described using,

- Entity declaration
- Architecture
- Configuration
- Package declaration
- Package body

Let's see what are these?

### Entity Declaration

It defines the names, input output signals and modes of a hardware module.

**Syntax –**

```
entity entity_name is
    Port declaration;
end entity_name;
```

An entity declaration should start with 'entity' and end with 'end' keywords. The direction will be input, output or inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source.

## Architecture –

Architecture can be described using structural, dataflow, behavioral or mixed style.

## Syntax –

```
architecture architecture_name of entity_name  
architecture_declarative_part;
```

```
begin
```

```
    Statements;
```

```
end architecture_name;
```

Here, we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the 'begin' and 'end' keyword. Architecture declarative part may contain variables, constants, or component declaration.

## Data Flow Modeling

In this modeling style, the flow of data through the entity is expressed using concurrent (parallel) signal. The concurrent statements in VHDL are WHEN and GENERATE.

Besides them, assignments using only operators (AND, NOT, +, \*, sll, etc.) can also be used to construct code.

Finally, a special kind of assignment, called BLOCK, can also be employed in this kind of code.

In concurrent code, the following can be used –

- Operators
- The WHEN statement (WHEN/ELSE or WITH/SELECT/WHEN);
- The GENERATE statement;
- The BLOCK statement

## Behavioral Modeling

In this modeling style, the behavior of an entity as set of statements is executed sequentially in the specified order. Only statements placed inside a PROCESS, FUNCTION, or PROCEDURE are sequential.

PROCESSES, FUNCTIONS, and PROCEDURES are the only sections of code that are executed sequentially.

However, as a whole, any of these blocks is still concurrent with any other statements placed outside it.

One important aspect of behavior code is that it is not limited to sequential logic. Indeed, with it, we can build sequential circuits as well as combinational circuits.



The behavior statements are IF, WAIT, CASE, and LOOP. VARIABLES are also restricted and they are supposed to be used in sequential code only. VARIABLE can never be global, so its value cannot be passed out directly.

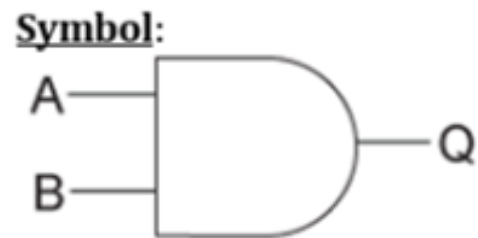
## Structural Modeling

In this modeling, an entity is described as a set of interconnected components. A component instantiation statement is a concurrent statement. Therefore, the order of these statements is not important. The structural style of modeling describes only an interconnection of components (viewed as black boxes), without implying any behavior of the components themselves nor of the entity that they collectively represent.

In Structural modeling, architecture body is composed of two parts – the declarative part (before the keyword begin) and the statement part (after the keyword begin).

## Logic Operation – AND GATE

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



VHDL **Code:**

```
Library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity and1 is
```

```
    port(x,y:in bit ; z:out bit);
```

```
end and1;
```

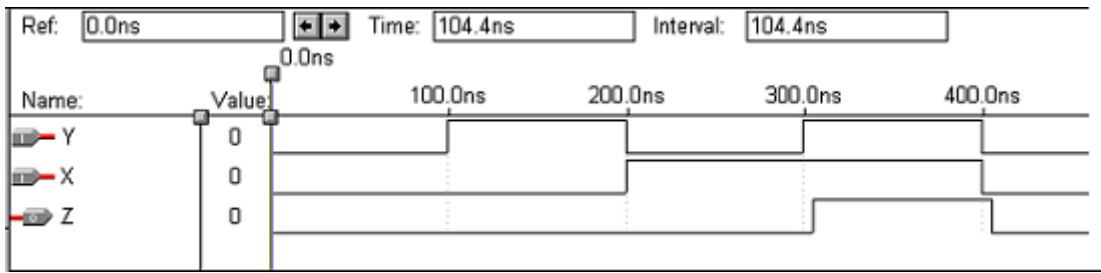
```
architecture virat of and1 is
```

```
begin
```

```
    z<=x and y;
```

```
end virat;
```

## Waveforms



## Logic Operation – OR Gate

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

### Symbol:



### VHDL Code:

```
Library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
entity or1 is
```

```
    port(x,y:in bit ; z:out bit);
```

```
end or1;
```

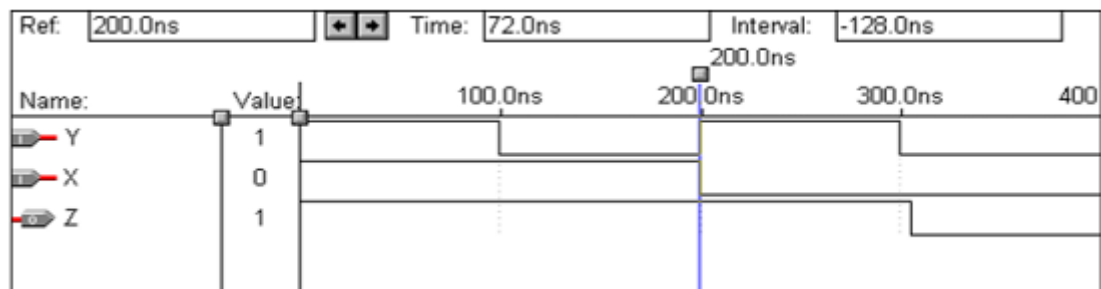
```
architecture virat of or1 is
```

```
begin
```

```
    z<=x or y;
```

```
end virat;
```

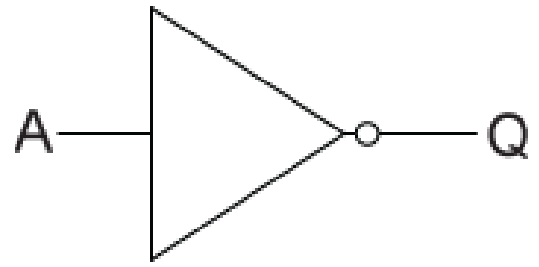
## Waveforms



## Logic Operation – NOT Gate

X	Y
0	1
1	0

Symbol :



VHDL Code:

```

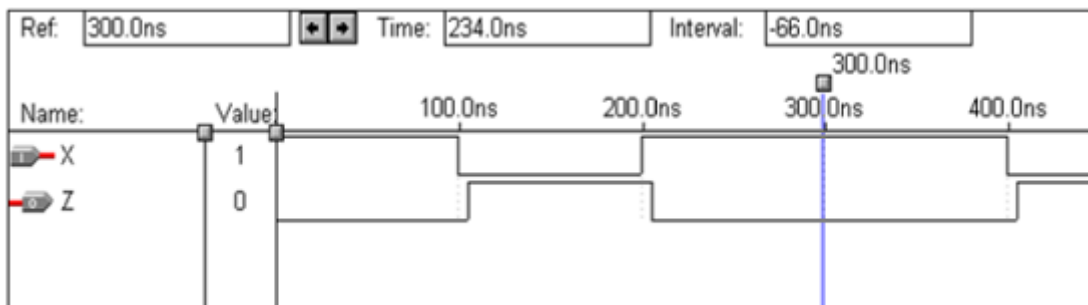
Library ieee;
use ieee.std_logic_1164.all;

entity not1 is
  port(x:in bit ; y:out bit);
end not1;

architecture virat of not1 is
begin
  y<=not x;
end virat;

```

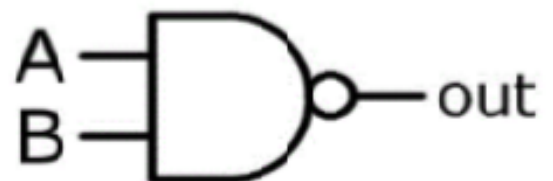
## Waveforms



## Logic Operation – NAND Gate

X	Y	z
0	0	1
0	1	1
1	0	1
1	1	0

Symbol:



VHDL **Code:**

```

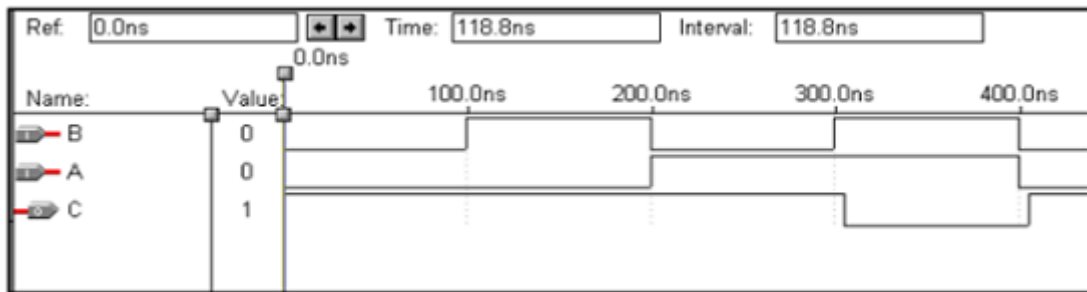
Library ieee;
use ieee.std_logic_1164.all;

entity nand1 is
  port(a,b:in bit ; c:out bit);
end nand1;

architecture virat of nand1 is
begin
  c<=a nand b;
end virat;

```

## Waveforms



## Logic Operation – NOR Gate

X	Y	z
0	0	1
0	1	0
1	0	0
1	1	0

Symbol:



VHDL **Code:**

```

Library ieee;
use ieee.std_logic_1164.all;

entity nor1 is
  port(a,b:in bit ; c:out bit);
end nor1;

architecture virat of nor1 is
begin

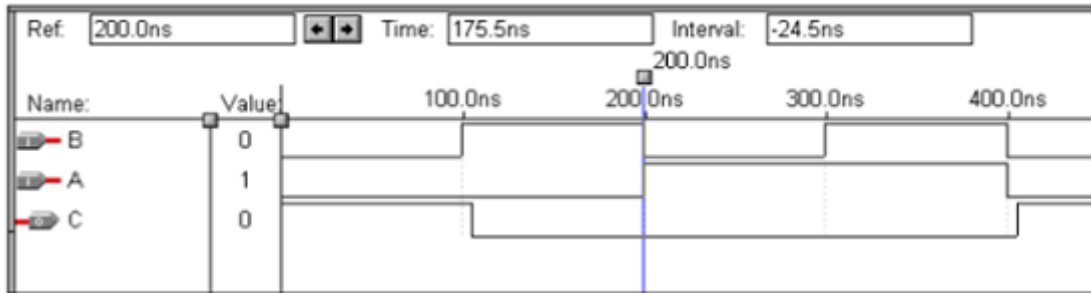
```

```

c<=a nor b;
end virat;

```

## Waveforms



## Logic Operation – XOR Gate

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

Symbol:



VHDL **Code:**

```

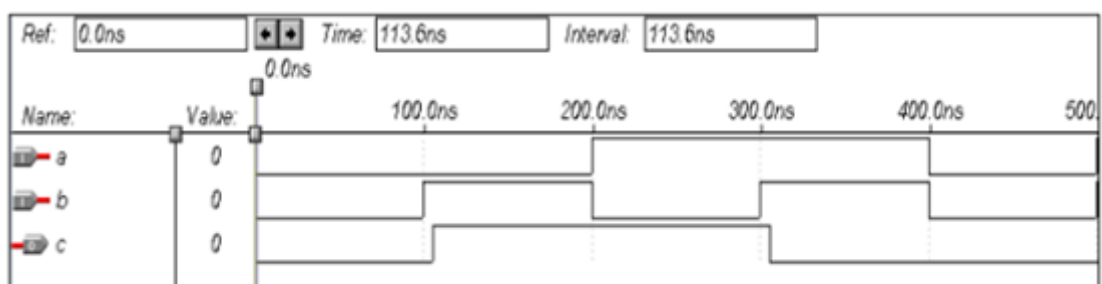
Library ieee;
use ieee.std_logic_1164.all;

entity xor1 is
    port(a,b:in bit ; c:out bit);
end xor1;

architecture virat of xor1 is
begin
    c<=a xor b;
end virat;

```

## Waveforms



## Logic Operation – X-NOR Gate

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

**Symbol:**



VHDL **Code:**

```

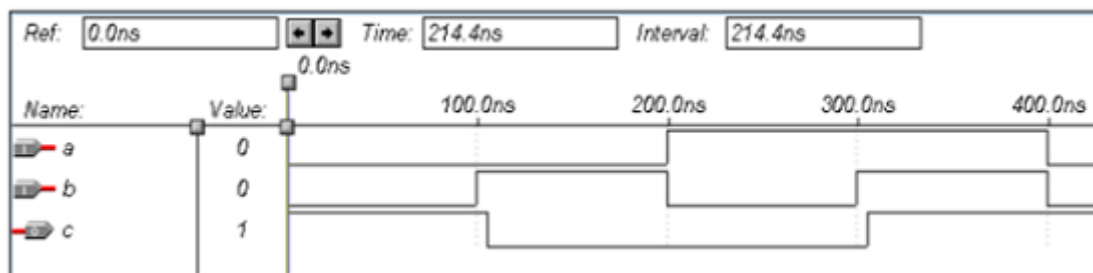
Library ieee;
use ieee.std_logic_1164.all;

entity xnor1 is
    port(a,b:in bit ; c:out bit);
end xnor1;

architecture virat of xnor1 is
begin
    c<=not(a xor b);
end virat;

```

## Waveforms



# VHDL Programming Combinational Circuits

This chapter explains the VHDL programming for Combinational Circuits.

## VHDL Code for a Half-Adder

VHDL **Code**:

```

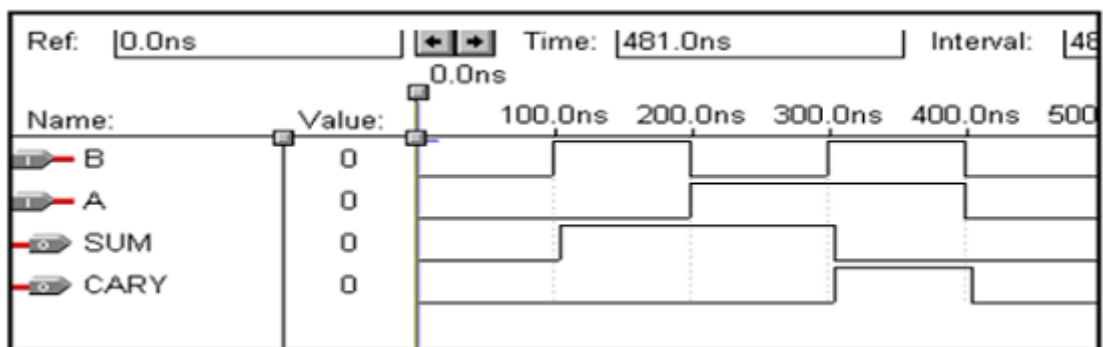
Library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port(a,b:in bit; sum,carry:out bit);
end half_adder;

architecture data of half_adder is
begin
  sum<= a xor b;
  carry <= a and b;
end data;

```

## Waveforms



## VHDL Code for a Full Adder

```

Library ieee;
use ieee.std_logic_1164.all;

entity full_adder is port(a,b,c:in bit; sum,carry:out bit);

```

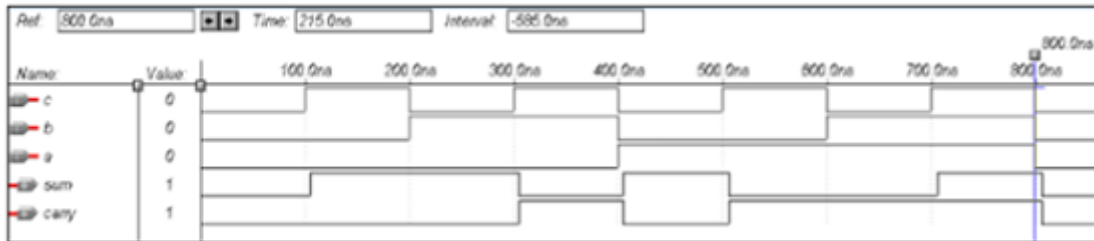
```

end full_adder;

architecture data of full_adder is
begin
    sum<= a xor b xor c;
    carry <= ((a and b) or (b and c) or (a and c));
end data;

```

## Waveforms



## VHDL Code for a Half-Subtractor

```

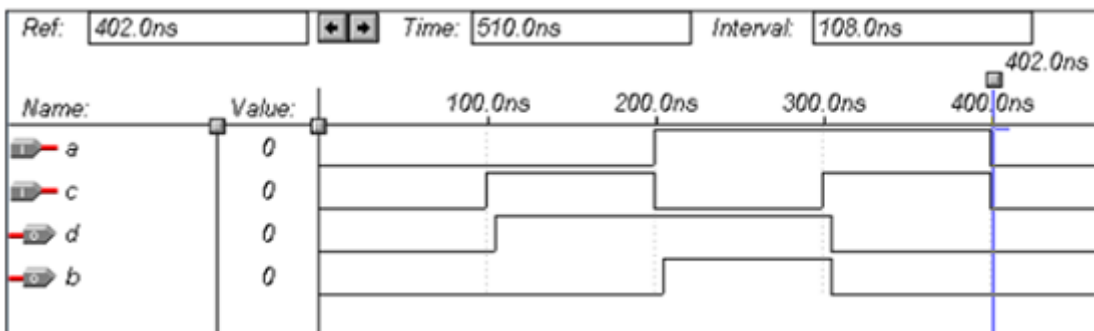
Library ieee;
use ieee.std_logic_1164.all;

entity half_sub is
    port(a,c:in bit; d,b:out bit);
end half_sub;

architecture data of half_sub is
begin
    d<= a xor c;
    b<= (a and (not c));
end data;

```

## Waveforms



## VHDL Code for a Full Subtractor

```

Library ieee;
use ieee.std_logic_1164.all;

```



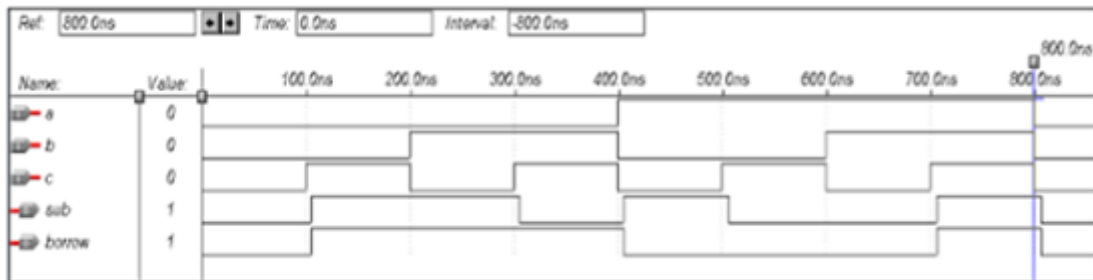
```

entity full_sub is
    port(a,b,c:in bit; sub,borrow:out bit);
end full_sub;

architecture data of full_sub is
begin
    sub<= a xor b xor c;
    borrow <= ((b xor c) and (not a)) or (b and c);
end data;

```

## Waveforms



## VHDL Code for a Multiplexer

```

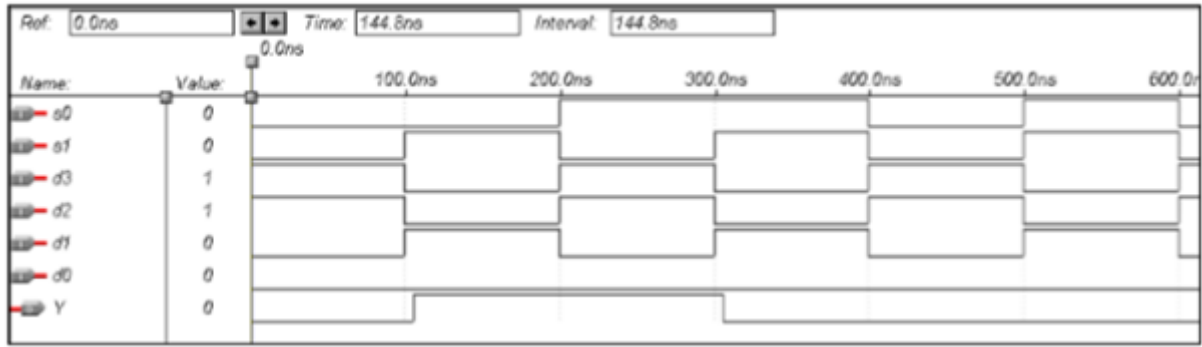
Library ieee;
use ieee.std_logic_1164.all;

entity mux is
    port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);
end mux;

architecture data of mux is
begin
    Y<= (not S0 and not S1 and D0) or
        (S0 and not S1 and D1) or
        (not S0 and S1 and D2) or
        (S0 and S1 and D3);
end data;

```

## Waveforms



## VHDL Code for a Demultiplexer

```

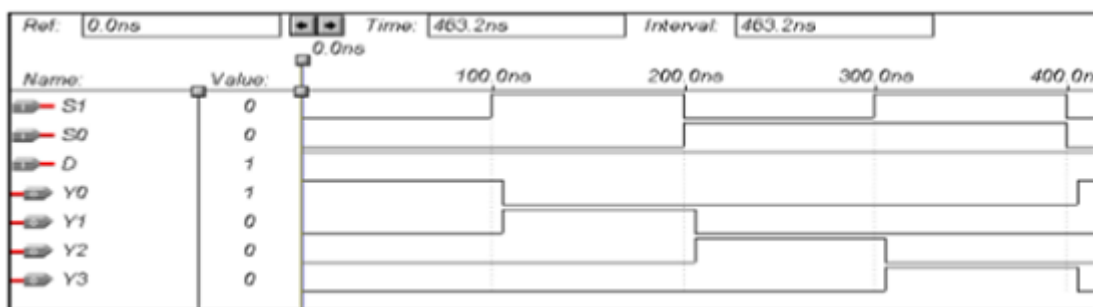
Library ieee;
use ieee.std_logic_1164.all;

entity demux is
    port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);
end demux;

architecture data of demux is
begin
    Y0<= ((Not S0) and (Not S1) and D);
    Y1<= ((Not S0) and S1 and D);
    Y2<= (S0 and (Not S1) and D);
    Y3<= (S0 and S1 and D);
end data;

```

## Waveforms



## VHDL Code for a 8 x 3 Encoder

```

library ieee;
use ieee.std_logic_1164.all;

entity enc is
    port(i0,i1,i2,i3,i4,i5,i6,i7:in bit; o0,o1,o2: out bit);
end enc;

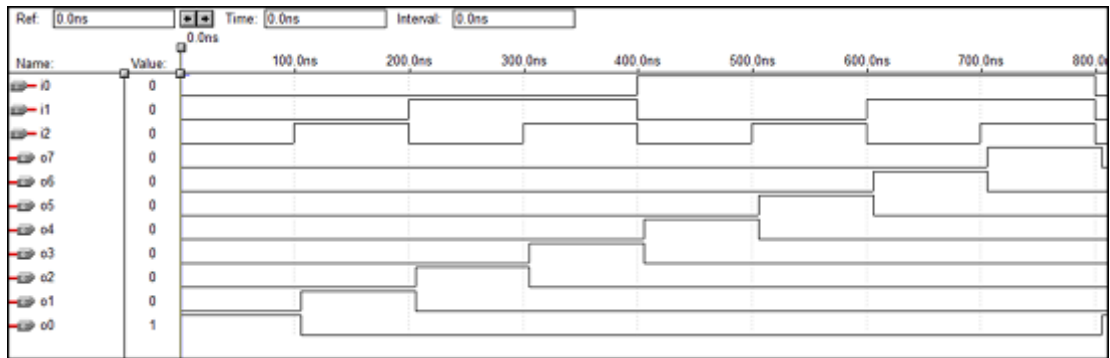
```

```

architecture vcgandhi of enc is
begin
    o0<=i4 or i5 or i6 or i7;
    o1<=i2 or i3 or i6 or i7;
    o2<=i1 or i3 or i5 or i7;
end vcgandhi;

```

## Waveforms



## VHDL Code for a 3 x 8 Decoder

```

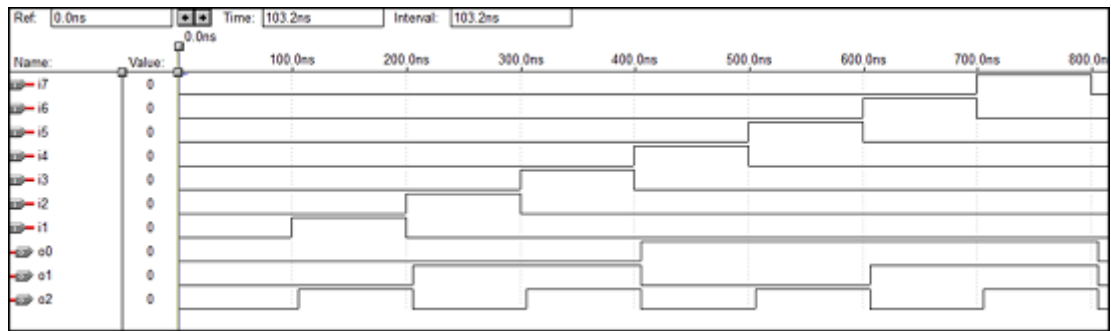
library ieee;
use ieee.std_logic_1164.all;

entity dec is
    port(i0,i1,i2:in bit; o0,o1,o2,o3,o4,o5,o6,o7: out bit);
end dec;

architecture vcgandhi of dec is
begin
    o0<=(not i0) and (not i1) and (not i2);
    o1<=(not i0) and (not i1) and i2;
    o2<=(not i0) and i1 and (not i2);
    o3<=(not i0) and i1 and i2;
    o4<=i0 and (not i1) and (not i2);
    o5<=i0 and (not i1) and i2;
    o6<=i0 and i1 and (not i2);
    o7<=i0 and i1 and i2;
end vcgandhi;

```

## Waveforms



## VHDL Code – 4 bit Parallel adder

```

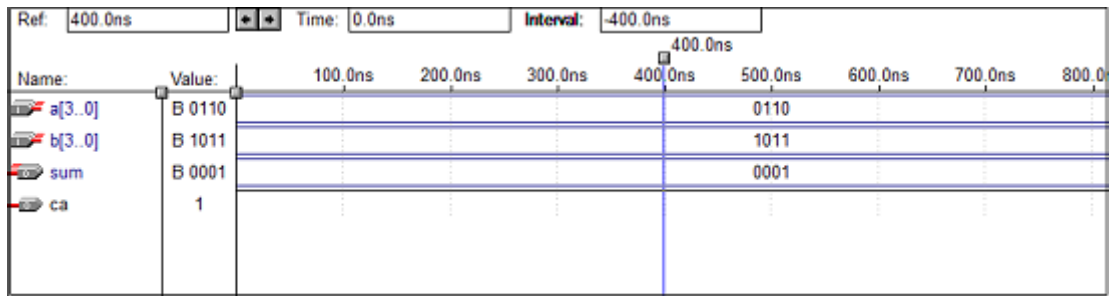
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity pa is
    port(a : in STD_LOGIC_VECTOR(3 downto 0);
          b : in STD_LOGIC_VECTOR(3 downto 0);
          ca : out STD_LOGIC;
          sum : out STD_LOGIC_VECTOR(3 downto 0)
    );
end pa;

architecture vcgandhi of pa is
    Component fa is
        port (a : in STD_LOGIC;
              b : in STD_LOGIC;
              c : in STD_LOGIC;
              sum : out STD_LOGIC;
              ca : out STD_LOGIC
        );
    end component;
    signal s : std_logic_vector (2 downto 0);
    signal temp: std_logic;
begin
    temp<='0';
    u0 : fa port map (a(0),b(0),temp,sum(0),s(0));
    u1 : fa port map (a(1),b(1),s(0),sum(1),s(1));
    u2 : fa port map (a(2),b(2),s(1),sum(2),s(2));
    ue : fa port map (a(3),b(3),s(2),sum(3),ca);
end vcgandhi;

```

## Waveforms



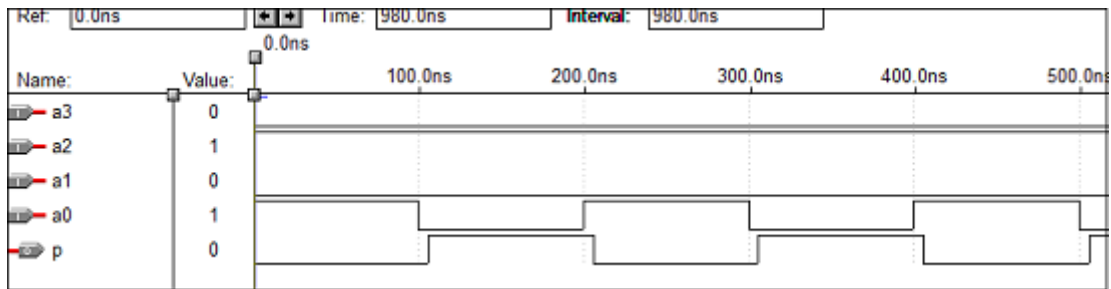
## VHDL Code – 4 bit Parity Checker

```
library ieee;
use ieee.std_logic_1164.all;

entity parity_checker is
    port (a0,a1,a2,a3 : in std_logic;
          p : out std_logic);
end parity_checker;

architecture vcgandhi of parity_checker is
begin
    p <= (((a0 xor a1) xor a2) xor a3);
end vcgandhi;
```

## Waveforms



## VHDL Code – 4 bit Parity Generator

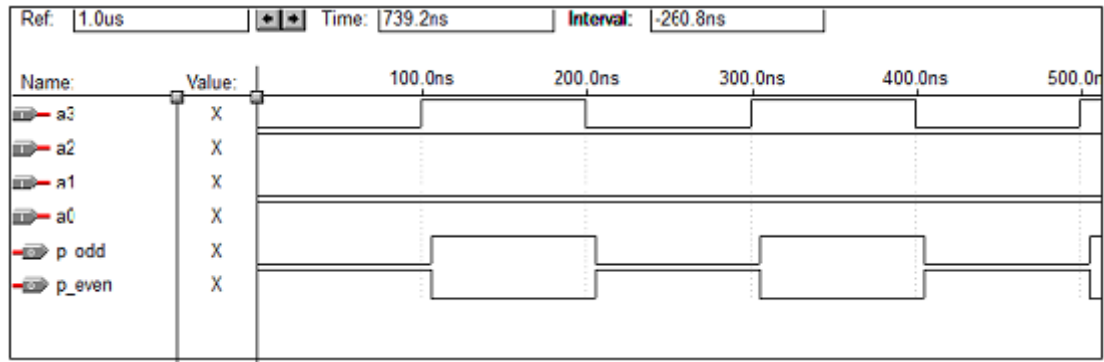
```
library ieee;
use ieee.std_logic_1164.all;

entity paritygen is
    port (a0, a1, a2, a3: in std_logic; p_odd, p_even: out std_logic);
end paritygen;

architecture vcgandhi of paritygen is
begin
    process (a0, a1, a2, a3)
```

```
if (a0 = '0' and a1 = '0' and a2 = '0' and a3 = '0')
  then odd_out <= "0";
  even_out <= "0";
else
  p_odd <= (((a0 xor a1) xor a2) xor a3);
  p_even <= not(((a0 xor a1) xor a2) xor a3);
end vcgandhi
```

## Waveforms



# VHDL Programming for Sequential Circuits

This chapter explains how to do VHDL programming for Sequential Circuits.

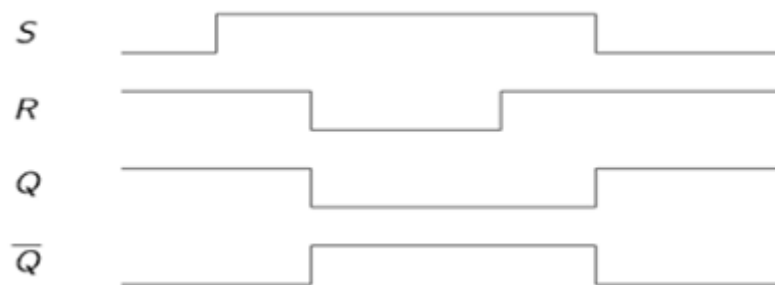
## VHDL Code for an SR Latch

```
library ieee;
use ieee.std_logic_1164.all;

entity srl is
  port(r,s:in bit; q,qbar:buffer bit);
end srl;

architecture virat of srl is
  signal s1,r1:bit;
begin
  q<= s nand qbar;
  qbar<= r nand q;
end virat;
```

## Waveforms



## VHDL Code for a D Latch

```
library ieee;
use ieee.std_logic_1164.all;

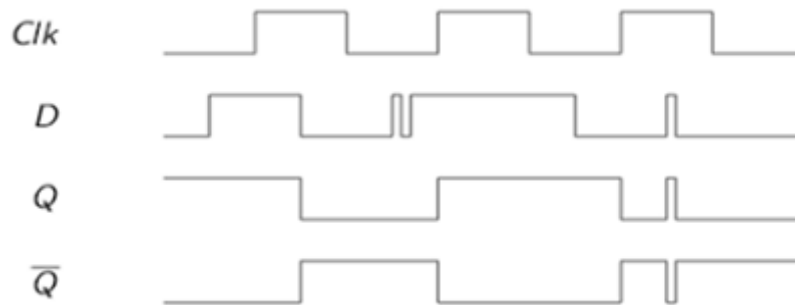
entity D1 is
  port(d:in bit; q,qbar:buffer bit);
end D1;
```

```

architecture virat of D1 is
    signal s1,r1:bit;
begin
    q<= d nand qbar;
    qbar<= d nand q;
end virat;

```

## Waveforms



## VHDL Code for an SR Flip Flop

```

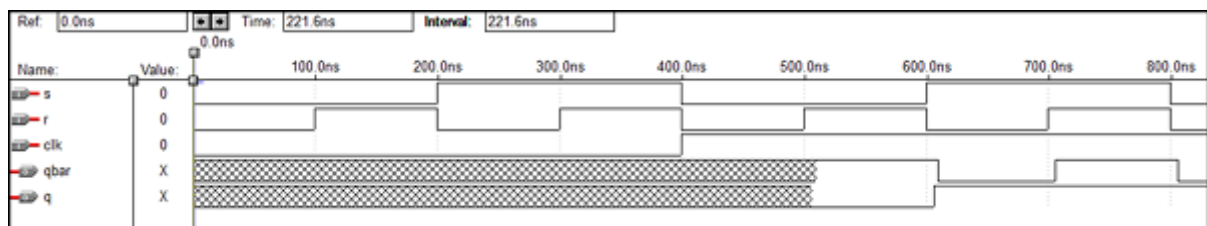
library ieee;
use ieee.std_logic_1164.all;

entity srflip is
    port(r,s,clk:in bit; q,qbar:buffer bit);
end srflip;

architecture virat of srflip is
    signal s1,r1:bit;
begin
    s1<=s nand clk;
    r1<=r nand clk;
    q<= s1 nand qbar;
    qbar<= r1 nand q;
end virat;

```

## Waveforms





## VHDL code for a JK Flip Flop

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity jk is
  port(
    j : in STD_LOGIC;
    k : in STD_LOGIC;
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    q : out STD_LOGIC;
    qb : out STD_LOGIC
  );
end jk;

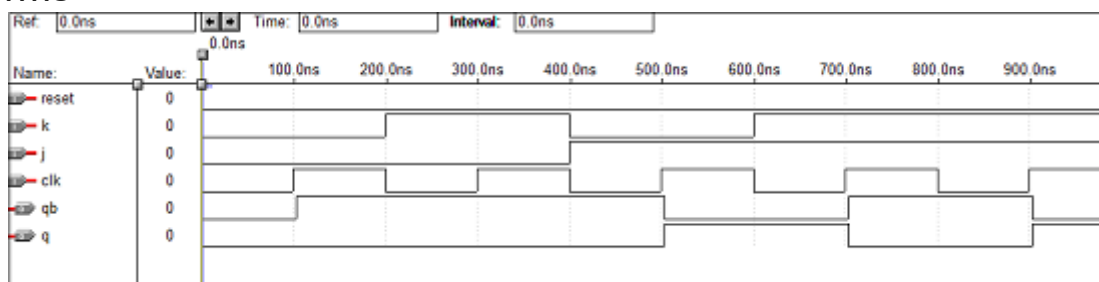
architecture virat of jk is
begin
  jkff : process (j,k,clk,reset) is
    variable m : std_logic := '0';

  begin
    if (reset = '1') then
      m := '0';
    elsif (rising_edge (clk)) then
      if (j/ = k) then
        m := j;
      elsif (j = '1' and k = '1') then
        m := not m;
      end if;
    end if;

    q <= m;
    qb <= not m;
  end process jkff;
end virat;

```

## Waveforms



## VHDL Code for a D Flip Flop

```

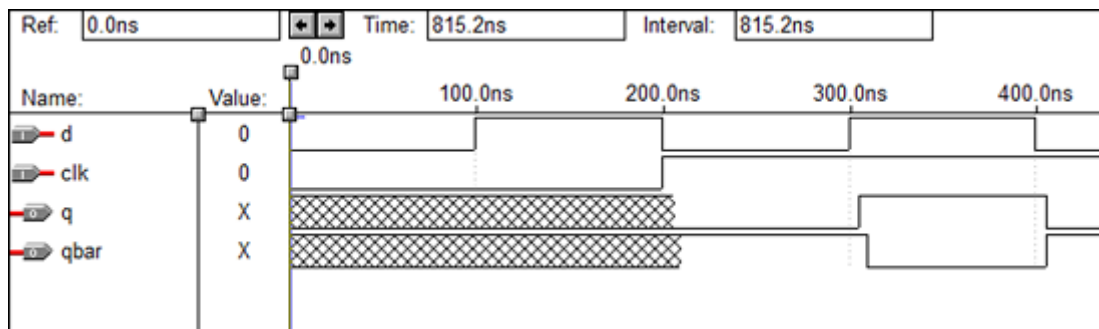
Library ieee;
use ieee.std_logic_1164.all;

entity dflip is
  port(d,clk:in bit; q,qbar:buffer bit);
end dflip;

architecture virat of dflip is
  signal d1,d2:bit;
begin
  d1<=d nand clk;
  d2<=(not d) nand clk;
  q<= d1 nand qbar;
  qbar<= d2 nand q;
end virat;

```

## Waveforms



## VHDL Code for a T Flip Flop

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Toggle_flip_flop is
  port(
    t : in STD_LOGIC;
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    dout : out STD_LOGIC
  );
end Toggle_flip_flop;

architecture virat of Toggle_flip_flop is
begin
  tff : process (t,clk,reset) is
    variable m : std_logic := '0';

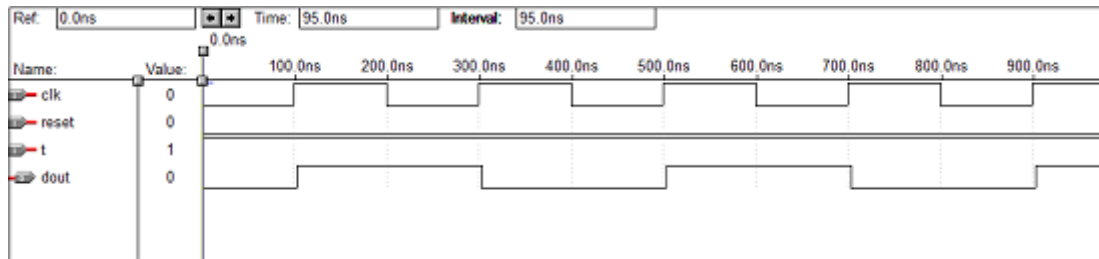
```

```

begin
  if (reset = '1') then
    m := '0';
  elsif (rising_edge (clk)) then
    if (t = '1') then
      m := not m;
    end if;
  end if;
  dout <= m;
end process tff;
end virat;

```

## Waveforms



## VHDL Code for a 4 - bit Up Counter

```

library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port(Clock, CLR : in std_logic;
        Q : out std_logic_vector(3 downto 0)
  );
end counter;

architecture virat of counter is
  signal tmp: std_logic_vector(3 downto 0);
begin
  process (Clock, CLR)

  begin
    if (CLR = '1') then
      tmp <= "0000";
    elsif (Clock'event and Clock = '1') then
      mp <= tmp + 1;
    end if;
  end process;
end virat;

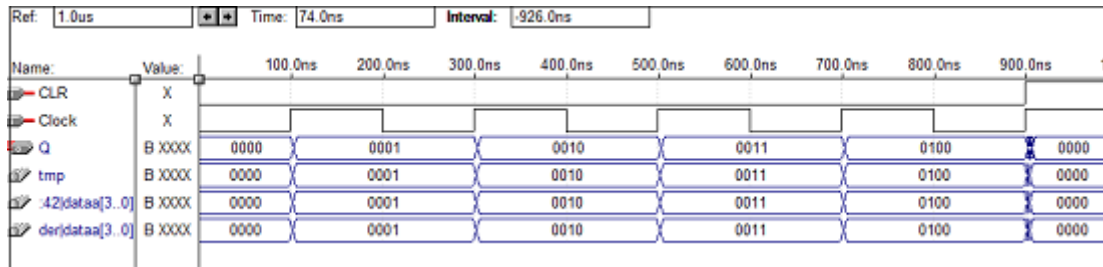
```

```

end process;
Q <= tmp;
end virat;

```

## Waveforms



## VHDL Code for a 4-bit Down Counter

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dcounter is
    port(Clock, CLR : in std_logic;
          Q : out std_logic_vector(3 downto 0));
end dcounter;

architecture virat of dcounter is
    signal tmp: std_logic_vector(3 downto 0);

begin
    process (Clock, CLR)
    begin
        if (CLR = '1') then
            tmp <= "1111";
        elsif (Clock'event and Clock = '1') then
            tmp <= tmp - 1;
        end if;
    end process;
    Q <= tmp;
end virat;

```

## Waveforms

