Basics of Microcontrollers

G.Mallikarjuna Rao

Microcontroller

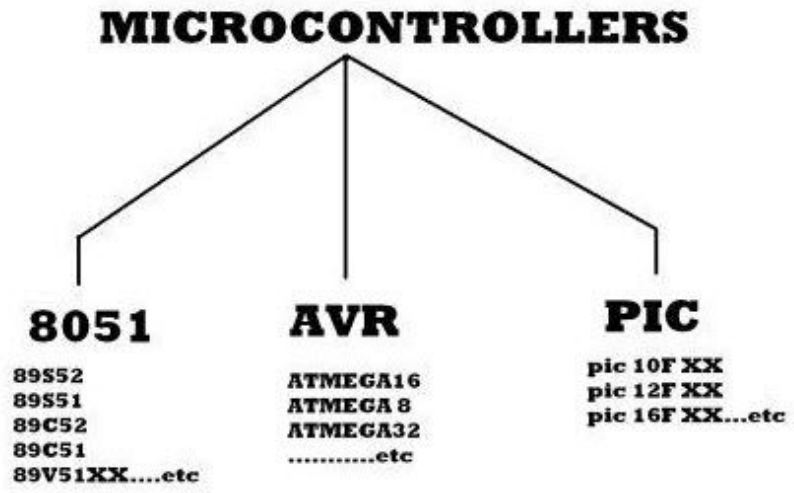
- A microcontroller is a small, low-cost and self contained computer-on-a-chip that can be used as an embedded system.
- A microcontroller is a system on a chip (SOC) it has CPU, I/O ports, timers, RAM, ROM on a single chip or it is like integrated on a single chip.
- Microcontrollers usually low powered (battery-operated) and are used in many consumer electronics, car engines, computer peripherals and test or measurement equipment

Basic contents of Microcontrollers

- An 8 or 16 bit microprocessor.
- A little measure of RAM.
- Programmable ROM and flash memory.
- Parallel and serial I/O.
- Timers and signal generators.
- Analog to Digital and Digital to Analog conversion

Classification

- The microcontrollers are characterized regarding bus-width, instruction set, and memory structure.



Classification

Classification According to Bits

- In **8-bit** microcontroller, internal bus is 8-bit and ALU performs 8-bit arithmetic and logic operations.
  - ❑ The examples of 8-bit microcontrollers are Intel 8031/8051, PIC1x and Motorola MC68HC11 families.
- The 16-bit microcontroller performs greater precision and performance as compared to 8-bit. It can automatically operate on two 16 bit numbers.
  - ❑ Some examples of 16-bit microcontroller are 16-bit MCUs are extended 8051XA, PIC2x, Intel 8096 and Motorola MC68HC12 families.

Classification

- The **32-bit** microcontroller uses the 32-bit instructions to perform the arithmetic and logic operations.
- These are used in automatically controlled devices including implantable medical devices, engine control systems, office machines, appliances and other types of embedded systems.
  - ❑ Some examples are Intel/Atmel 251 family, PIC3x.

Classification

Classification According to Instruction Set

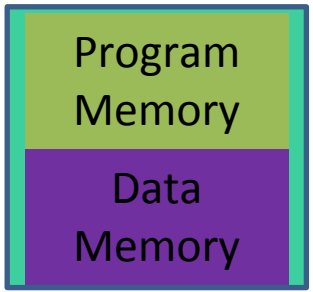
- **CISC**: CISC is a Complex Instruction Set Computer. It allows the programmer to use one instruction in place of many simpler instructions.
  - Ex: Renesas Rx, 8051
- **RISC**: The RISC is stands for Reduced Instruction Set Computer, this type of instruction sets reduces the design of microprocessor for industry standards.
  - Ex: AVR microcontrollers.

Classification

Classification According to Memory Architecture

- **Harvard Memory Architecture Microcontroller**: Here the microcontroller unit has a dissimilar memory address space for the program and data memory.

- **Princeton Memory Architecture Microcontroller**: The point when a microcontroller has a common memory address for the program memory and data memory, the microcontroller has Princeton memory architecture in the processor.



Types of Microcontrollers

- 8051 Microcontrollers typically 8-bit CISC architecture
- Renesas Rx controllers typically 32-bit controllers
- AVR Microcontrollers typically 8-bit, 32-bit controllers of RISC type, Atmel 8, Atmel 16 and Atmel328
- PIC : Peripheral Interconnect Microcontrollers typically 8-bit,16-bit and 32-bit are used for industrial and gaming applications
- ARM (Advanced RISC Machines) typically 32/64 bit controllers

	8051	PIC	AVR	ARM
Bus width	8-bit for standard core	8/16/32-bit	8/32-bit	32-bit mostly also available in 64-bit
Communication Protocols	UART, USART,SPI,I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet)	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA
Speed	12 Clock/instruction cycle	4 Clock/instruction cycle	1 clock/ instruction cycle	1 clock/ instruction cycle
Memory	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
ISA	CISC	Mostly CISC	RISC	RISC
Popular Microcontrollers	AT89C51, P89v51, etc.	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc.

Applications

Microcontroller has many applications electronic equipment’s

- Mobile Phones
- Auto Mobiles
- Washing Machines
- Cameras
- Security Alarms

Any Quires…?
Thank You

Check Your knowledge

ATmege 328 controller

G.Mallikarjuna Rao

ATmega328 & Arduino

www.TheEngineeringProjects.com

Contents

Atmega 328 controller

- Introduction
- Pin Diagram
- Architectural Diagram
- Addressing Modes

Introduction

- It is the Microchip product of 8-bit AVR Micro controller with RISC Architecture having 28-pins and 32KB Flash Memory. It has an EEPROM memory of 1KB and its SRAM memory is of 2KB.
- It has internally 10-bit ADC.
- It has **Three builtin** timers of which 2-are 8bits and one is 16-bits.

Features

- It operates ranging from 3.3V to 5.5V but normally we use 5V as a standard.
- Its excellent features include the cost efficiency, low power dissipation, programming lock for security purposes, real timer counter with separate oscillator.
- It's normally used in Embedded Systems applications.

Feature Summary

Snap of Atmel 328 Features	
No. of Pins	28
CPU	RISC 8-Bit AVR
Operating Voltage	1.8 to 5.5 V
Program Memory	32KB
Program Memory Type	Flash
SRAM	2048 Bytes
EEPROM	1024 Bytes
ADC	10-Bit
Number of ADC Channels	8
PWM Pins	6
USI (Universal Serial Interface)	Yes

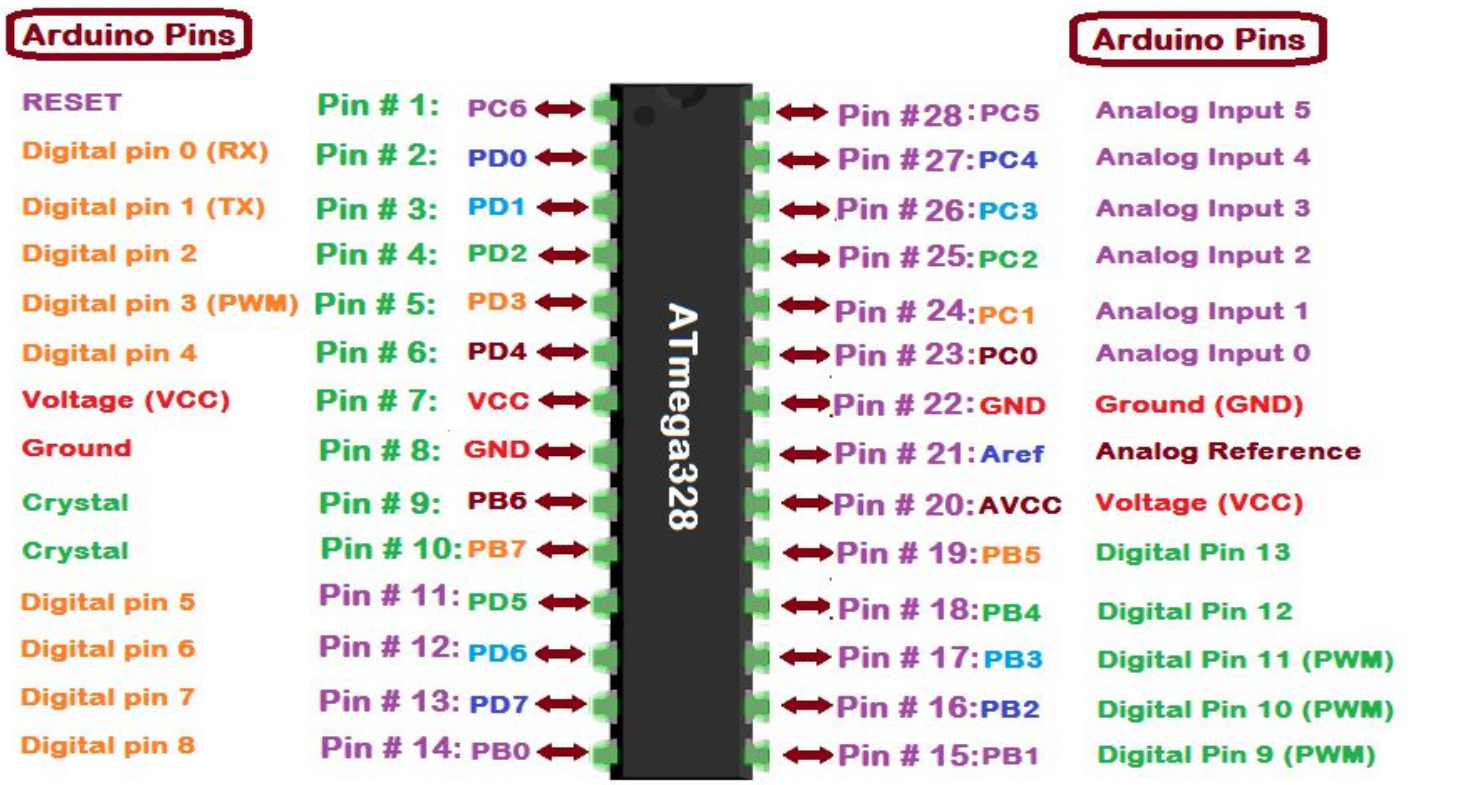
Snap of Atmel 328 Features (operating range -40 C to +85 C)	
Oscillator	up to 20 MHz
Timer (3)	8-Bit x 2 & 16-Bit x 1
Enhanced Power on Reset	Yes
Power Up Timer	Yes
I/O Pins	23
Manufacturer	Microchip
SPI	Yes
I2C	Yes
Watchdog Timer	Yes
Brown out detect (BOD)	Yes
Reset	Yes

Pin Diagram

- ATmega-328 is an AVR Microcontroller having twenty eight (28) pins in total (14-Digital and 6-Analog pins. ,1 Analog Reference, 2- clock and 5-power supply)
- ATmega-328 pins are divided into three ports (Port B(8) and Port C(7) &port D(8))
- 4-pins Digital VCC & GND, Analog AVCC and GND and 1-Analog Ref

ATmega328 Pins			
Pin Number	Pin Name	Pin Number	Pin Name
1	PC6	15	PB1
2	PD0	16	PB2
3	PD1	17	PB3
4	PD2	18	PB4
5	PD3	19	PB5
6	PD4	20	AVCC
7	Vcc	21	Aref
8	GND	22	GND
9	PB6	23	PC0
10	PB7	24	PC1
11	PD5	25	PC2
12	PD6	26	PC3
13	PD7	27	PC4
14	PB0	28	PC5

ATmega328 Pinout



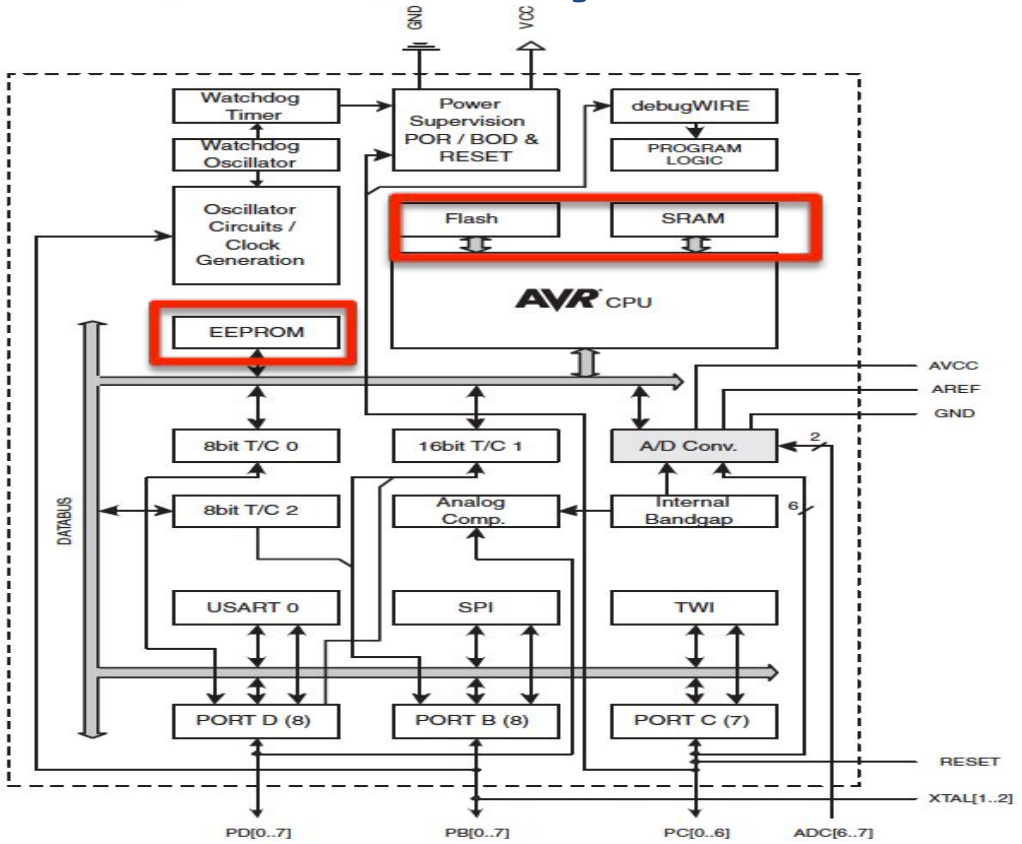
www.TheEngineeringProjects.com

Pin Functions

- VCC is a digital voltage supply. AVCC is a supply voltage pin for analog to digital converter. GND denotes Ground and it has a 0V.
- Port C consists of the pins from PC0 to PC5. These pins serve as analog input to analog to digital converters. If analog to digital converter is not used, as bidirectional input/output port.
- Port B consists of the pins from PB0 to PB7. This port is an 8 bit bidirectional port having an internal pull-up resistor.
- Port D consists of the pins from PD0 to PD7. It is also an 8 bit input/output port having an internal pull-up resistor.

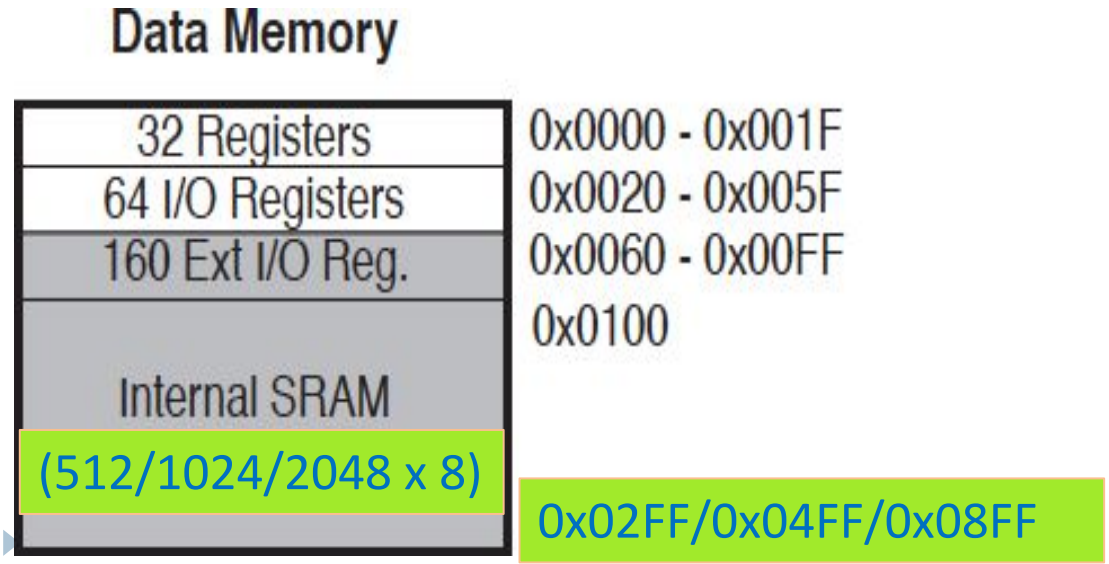
AVR Architecture : Memory

- Harvard architecture
- **32KB non-volatile** Flash (program memory)
  - Allocate data to Flash using PROGMEM keyword
- **2KB volatile** SRAM (data memory for stack &Temporary values)
- 1KB EEPROM
  - For long-term data
  - On I/O data bus



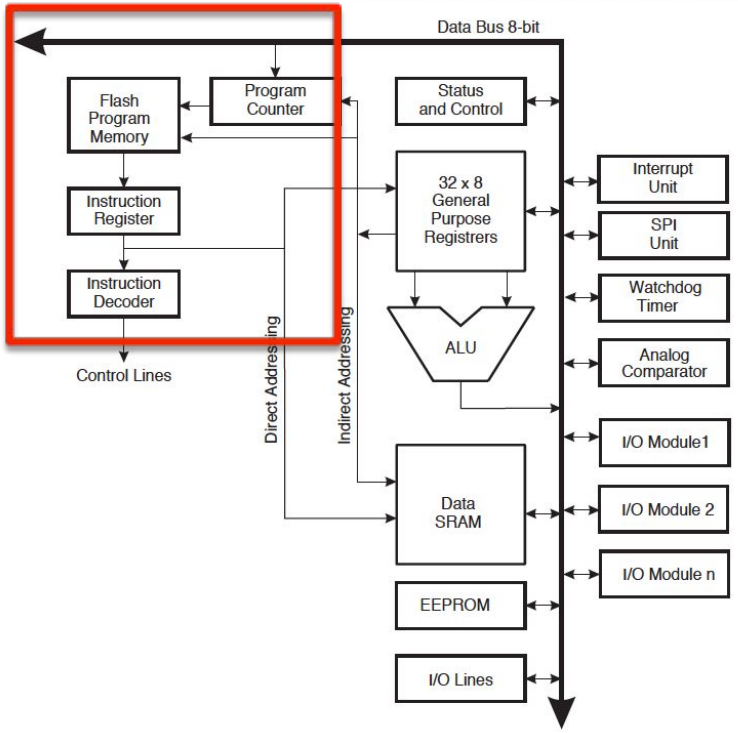
AVR M

- Program memory – Flash



AVR Architecture : IO Unit

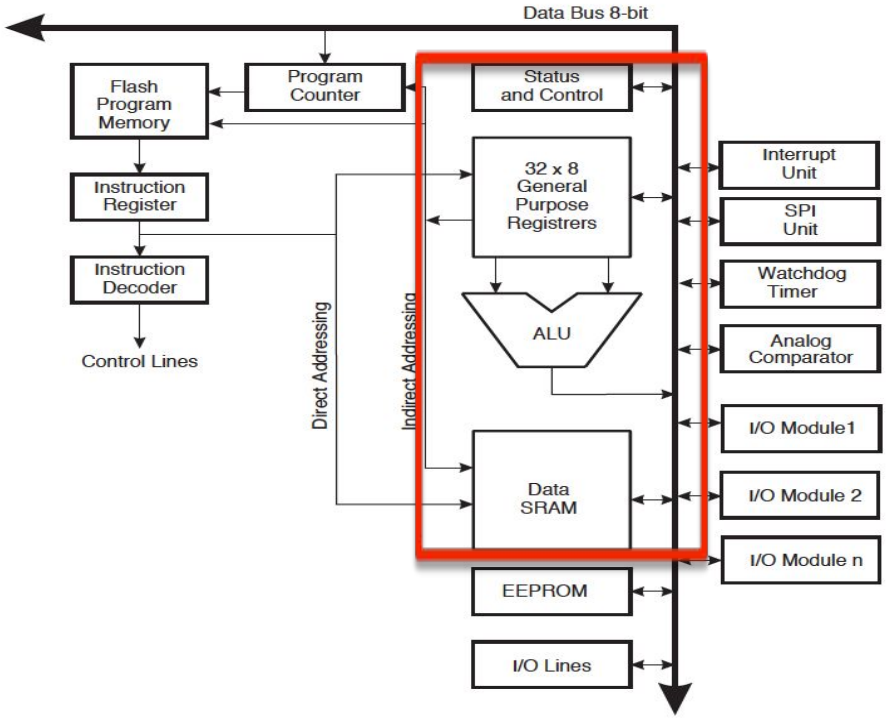
- Instruction Fetch and Decode



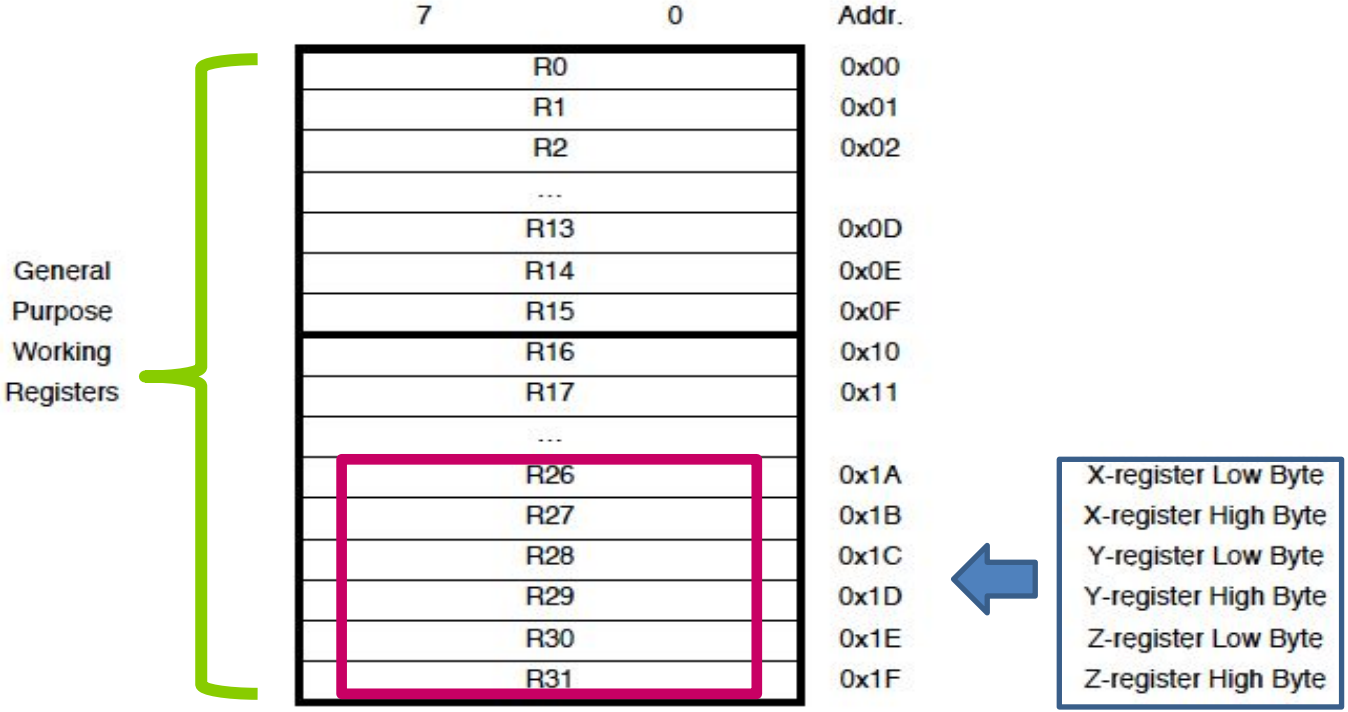
AVR Architecture : ALU

- ALU ,General Purpose Registers(part of SRAM)
- 32- 8-Bit Registers (part of SRAM), 6 of them can be used as 16-bit registers

.

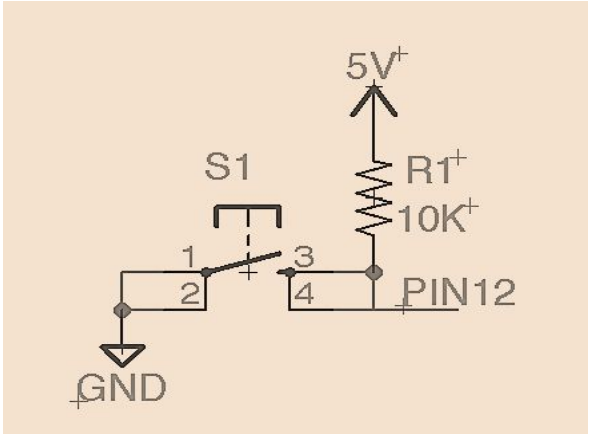


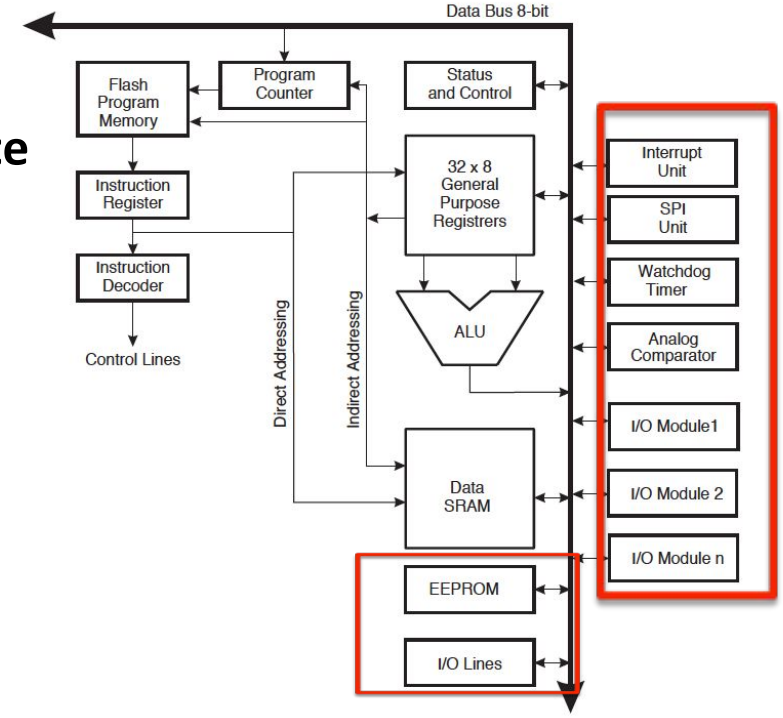
AVR Regist

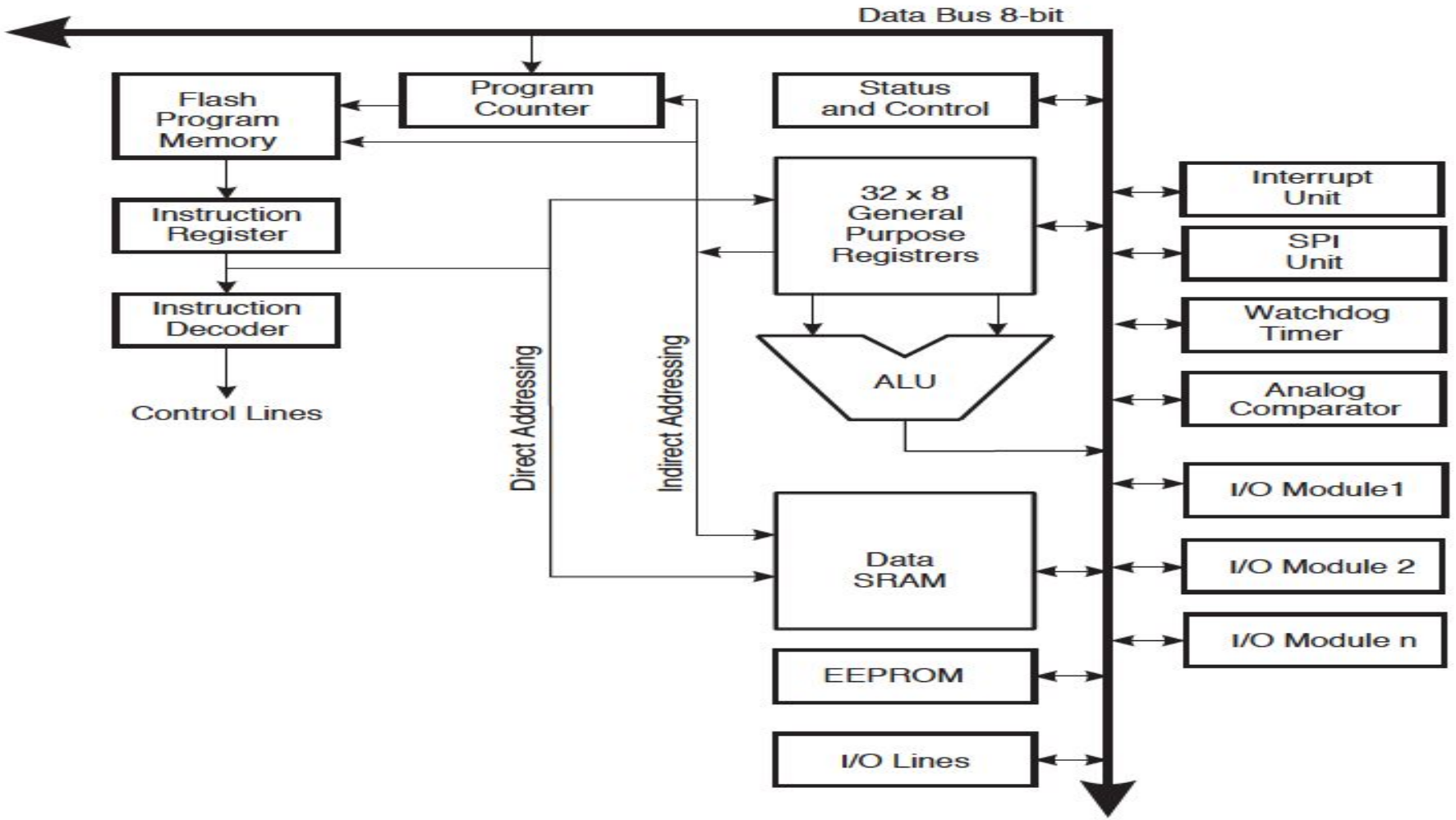

7
0
Addr.
R0
R1
R2
...
R13
R14
R15
R16
R17
...
R26
R27
R28
R29
R30
R31
0x00
0x01
0x02
0x0D
0x0E
0x0F
0x10
0x11
0x1A
0x1B
0x1C
0x1D
0x1E
0x1F
General Purpose Working Registers
X-register Low Byte
X-register High Byte
Y-register Low Byte
Y-register High Byte
Z-register Low Byte
Z-register High Byte

IO and Special functions

- Each I/O port pin has internal Pull-up resistor , i.e., floating state is HIGH, Switched state is LOW
- Serial Communication
- Interrupts






Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Data SRAM
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n

Observe the difference

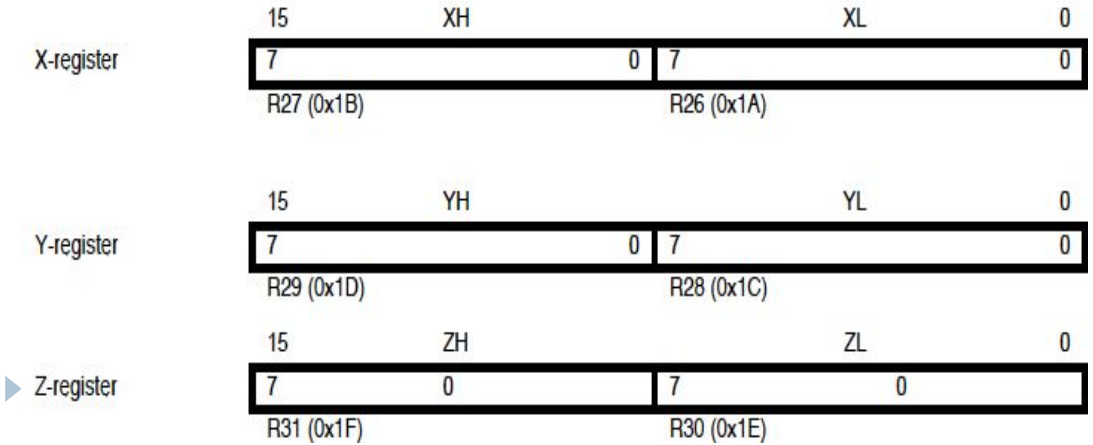
```
void setup() {
  pinMode(13, OUTPUT); // Use Built-In LED for Indication
  pinMode(12, INPUT);   // Push-Button On Bread Board
}

void loop() {
  bool buttonState = digitalRead(12);  // store current state of pin 12
  digitalWrite(13, buttonState);
}
```

```
1   void setup() {
2     pinMode(13, OUTPUT);     // Use Built-In LED for Indication
3     /* INPUT_PULLUP enables the Arduino Internal Pull-Up Resistor */
4     pinMode(12, INPUT_PULLUP);      // Push-Button On Bread Board
5   }
6
7   void loop() {
8     bool buttonState = digitalRead(12);  // store current state of pin 12
9     digitalWrite(13, buttonState);
10  }
```

Special Addressing Registers

- X,Y and Z registers
  - 16-bit registers made using registers 26 – 31
- Support indirect addressing

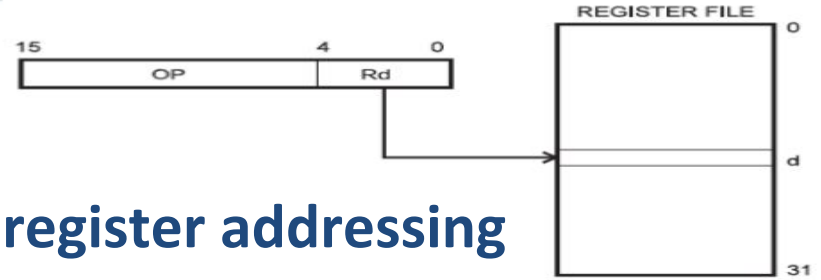
X, Y, Z Registers



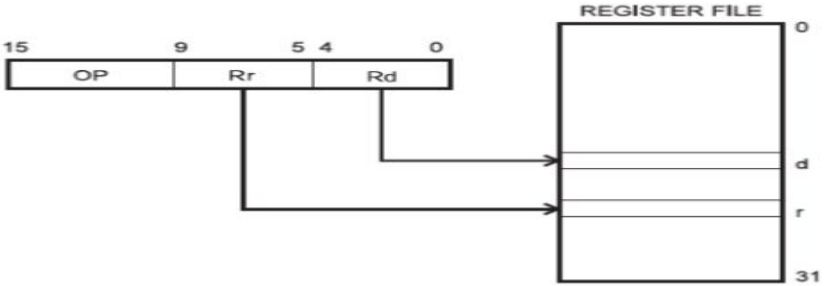
Addressing Modes

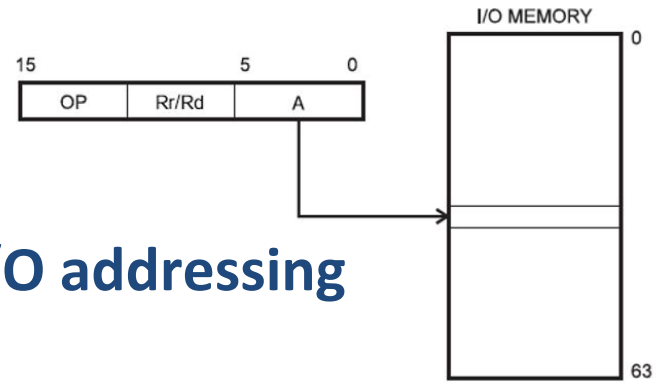
- Direct register addressing
- Direct I/O addressing
- Data memory addressing
  - Direct data memory addressing
  - Direct data memory with displacement addressing
  - Indirect data memory addressing
  - Indirect data memory addressing with pre-decrement
  - Indirect data memory addressing with post-increment
- Program memory addressing (constant data)

Addressing Modes



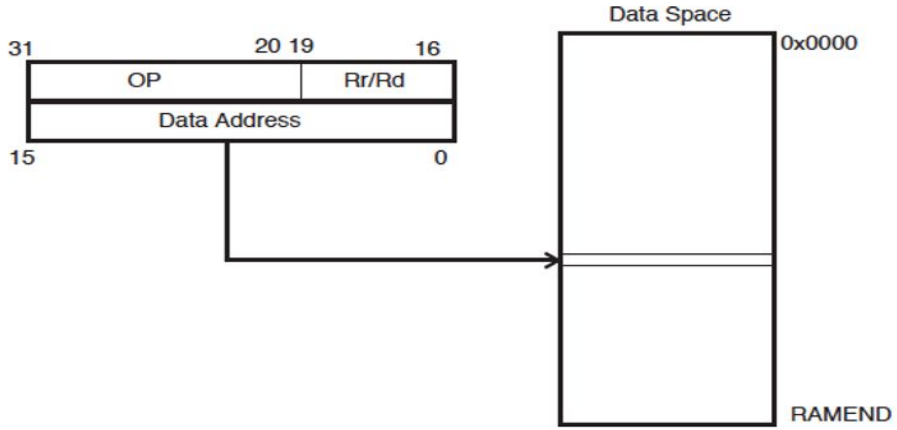
Direct register addressing

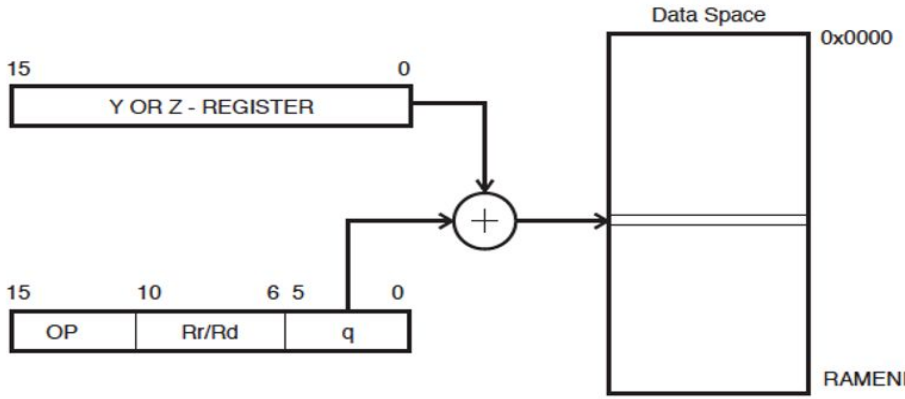


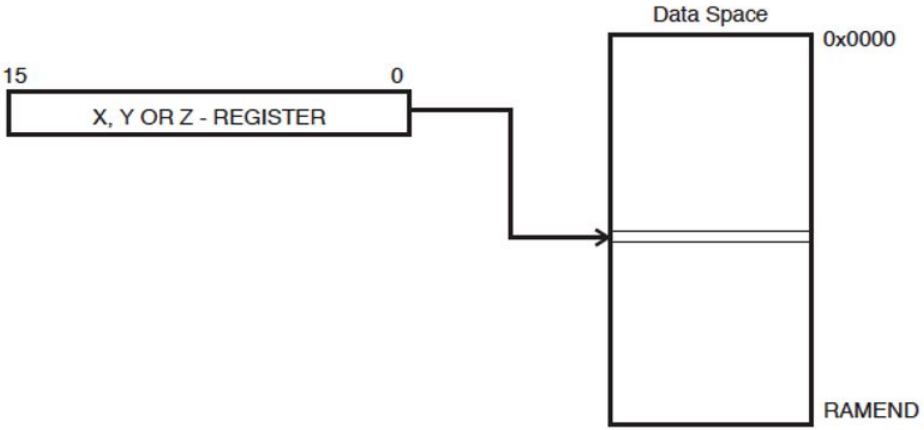


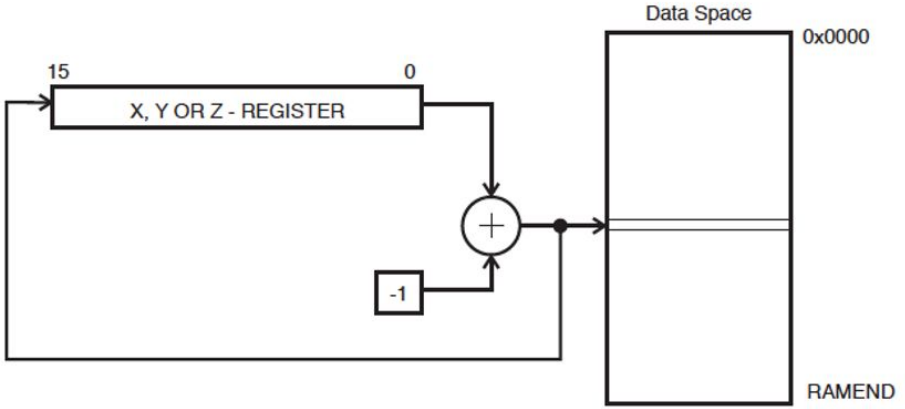
Direct I/O addressing

Data memory addressing

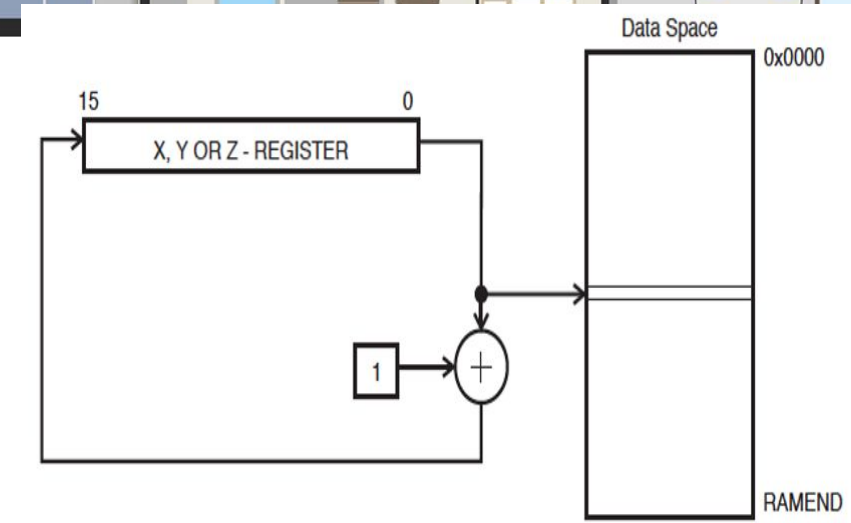




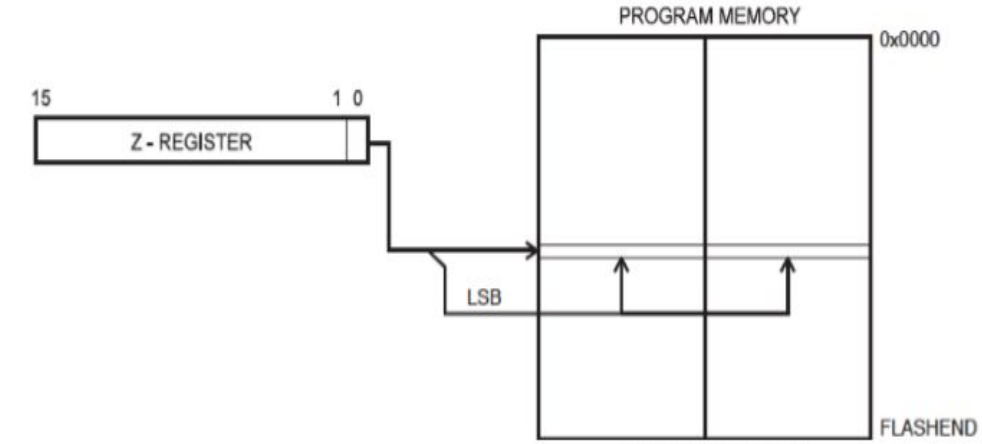


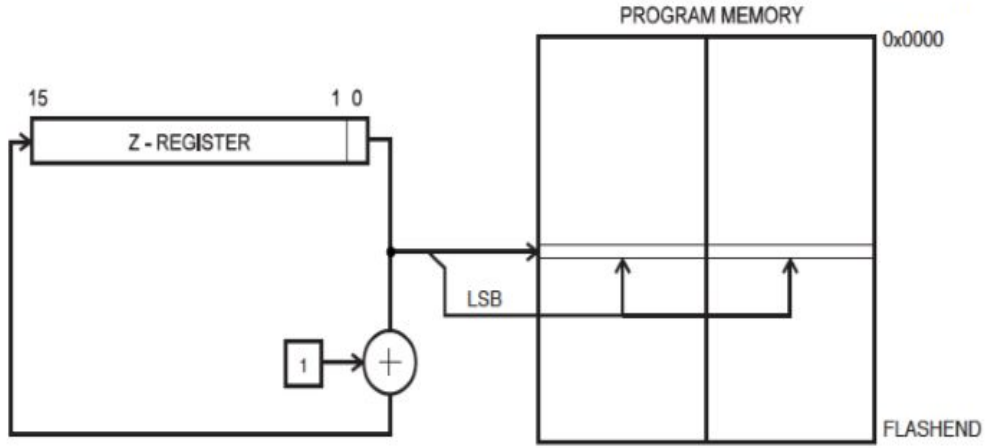


Indirect Data memory with Post increment



Program memory addressing (constant data)





Assembly programming for turing on LEDS (4no) connected Even pins of port D.

```
ldi  r18, 01010101
out DDRD,r18
out portD,r18
```

LED are connected to
D0,D2,D4,D6 digital pins

Any Quires…?
Thank You

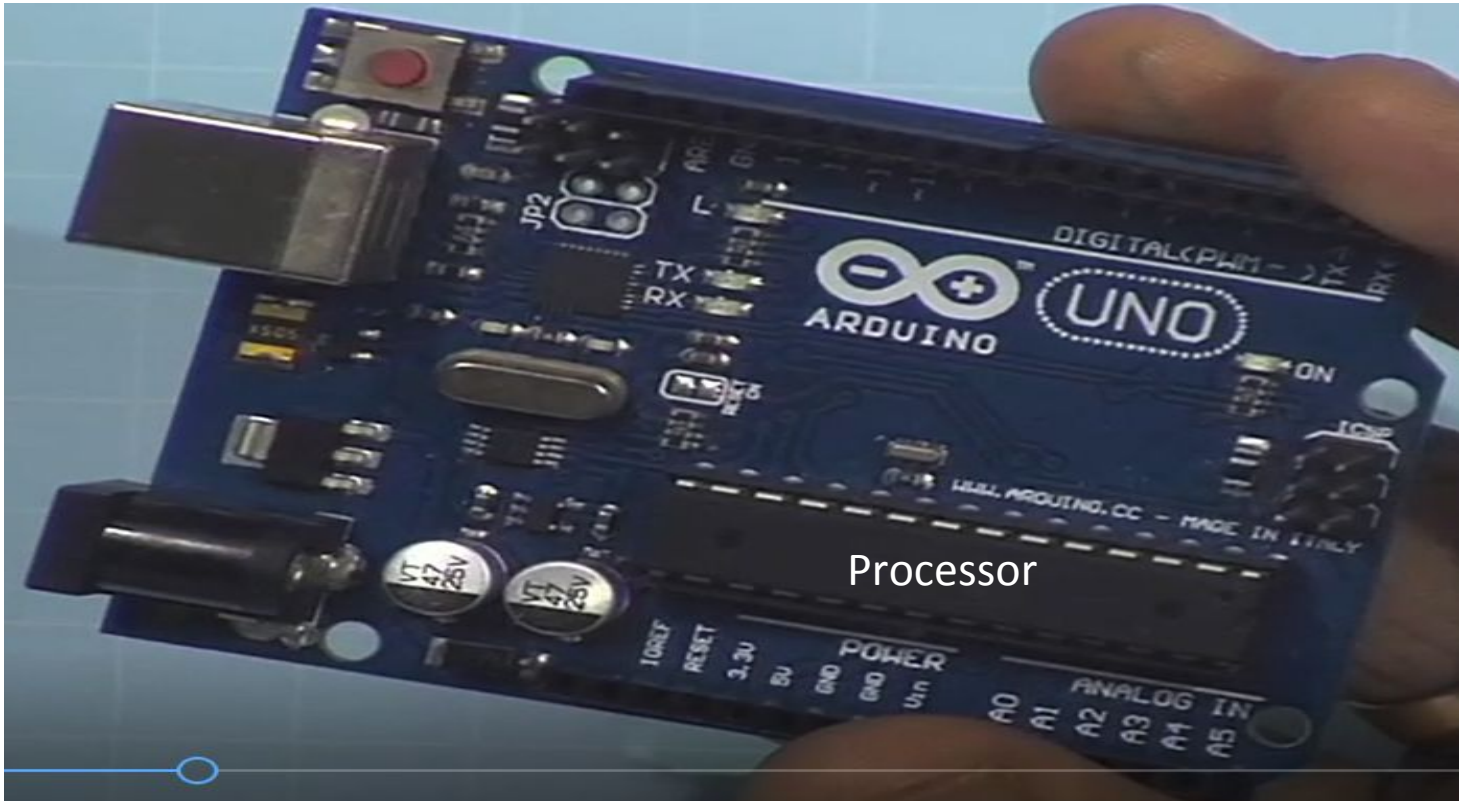
Check Your knowledge

Arduino Controller

G.Mallikarjuna Rao

Assembly programming for turing on LEDS (4no) connected Even pins of port D.

```
ldi  r18, 01010101
out DDRD,r18
out portD,r18
```


Processor
ARDUINO
UNO
DIGITALPWM
ON
ICSP
POWER
ANALOG IN

Objectives - Outcomes

Course Objectives: The objective of this course is to

1. Expose to 8 bit/16 bit microcontrollers.
2. Understand ATMEGA 328 Controller architecture.
3. Learn sensors and their controlling operations.
4. Understand raspberry pi architecture.
5. Understand Python web application frame work for IoT.

Course Outcomes: At the end of this course student will be able to

1. Use AVR Controllers.
2. Explore architecture of ATMEGA 328 controllers for programming.
3. Explore programming the sensor devices.
4. Develop IoT Projects.
5. Explore python web application framework for cloud IoT services.

Syllabus

Unit 1

Introduction: Introduction to microprocessors and micro controllers, differences between micro processor and micro controllers, AVR ATMEGA 328Controller: architecture of ATMEGA 328. ARDUINO: Introduction, Arduino Functions Libraries: Input and output functions, operators, control statements, loops, arrays, strings.

Unit II

Integration of Sensors and Actuators with Arduino: Sensors: Temperature, Compass, Light, Sound, Accelerometer, DHT, Distance Sensor Actuators: Servomotor, Stepper Motor, Solenoid, Relay, DC Motor Communication Devices: Bluetooth, RF433, Wi Fi Module

Syllabus

Unit III

Introduction To Internet of Things: Introduction, Physical Design of IoT, Logical Design of IoT, IoT enabling Technologies, IoT Levels and Deployment Templates Domain Specific IoTs: Introduction, Home Automation, Smart Cities, Environment, Energy, Retail, Logistics, Agriculture, Industry, Health and LifeStyle

UNIT IV

IoT Systems- Logical Design Using Python: Introduction, Installing python, Python data types and data structures, Control Flow, Functions, Modules, Packages, File Handling, Date/ Time operations, Classes, Python Packages of Interest for IoT.

IoT Physical Devices And End Points: IoT Device, Exemplary Device: Raspberry Pi, About the board, Linux on Raspberry Pi, Raspberry Pi Interfaces, Programming Raspberry Pi with Python

Syllabus

UNIT V

IoT Physical Server and Cloud Offerings: Introduction to cloud storage models and communication APIs, WAMP-AutoBahn for IoT, Xively Cloud for IoT, Python web application framework, Designing a RESTful Web API Case Studies Illustrating IoT Design: Home Automation, Cities, Environment, Agriculture

Text Books: 1. ArshdeepBahga, Vijay Madisetti " Internet of Things(A hands on approach)" 1ST edition, VPIpublications,2014 2. Embedded Controllers using C and Arduino/2E by Jmes M.Fiore

Board Details

- Power Supply: USB or power barrel jack
- Voltage Regulator
- LED Power Indicator
- Tx-Rx LED Indicator
- Output power, Ground
- Analog Input Pins
- Digital I/O Pins

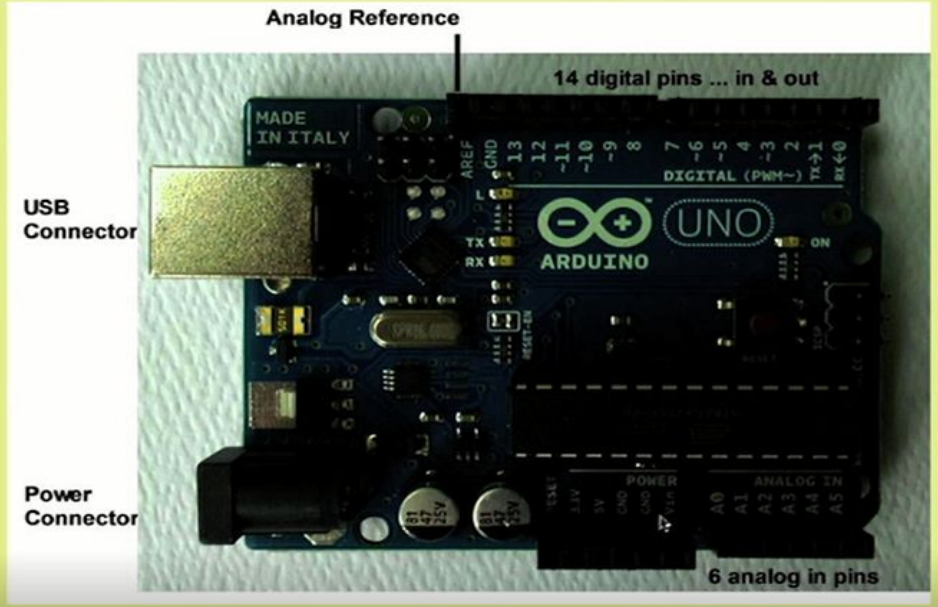


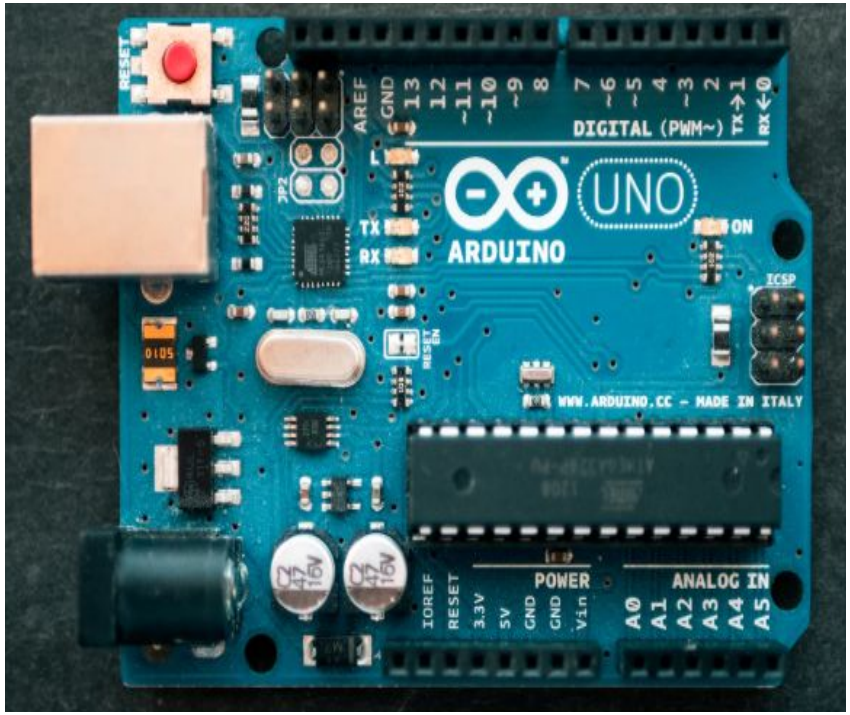
Image source: https://upload.wikimedia.org/wikipedia/commons/9/9d/UnoConnections.jpg

Arduino IDE

- Arduino IDE is an open source software that is used to program the Arduino controller board
- Based on variations of the C and C++ programming language
- It can be downloaded from Arduino's official website and installed into PC

ARDUINO UNO

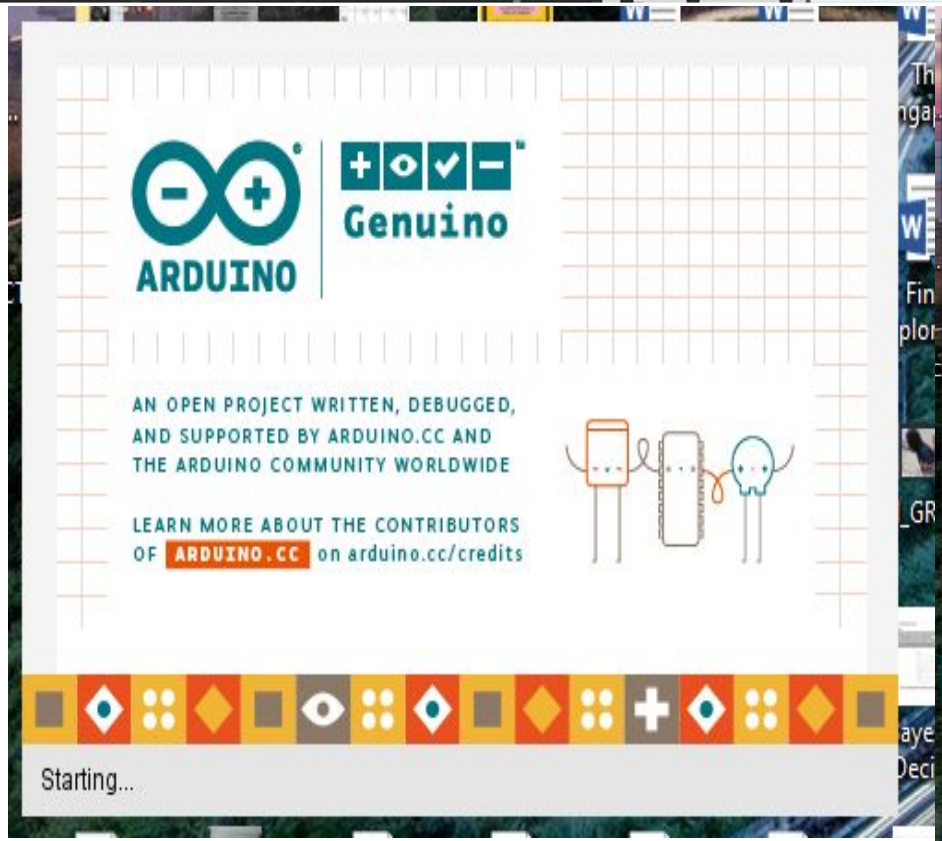
Feature	Value
Operating Voltage	5V
Clock Speed	16MHz
Digital I/O	14
Analog Input	6
PWM	6
UART	1
Interface	USB via ATMega16U2

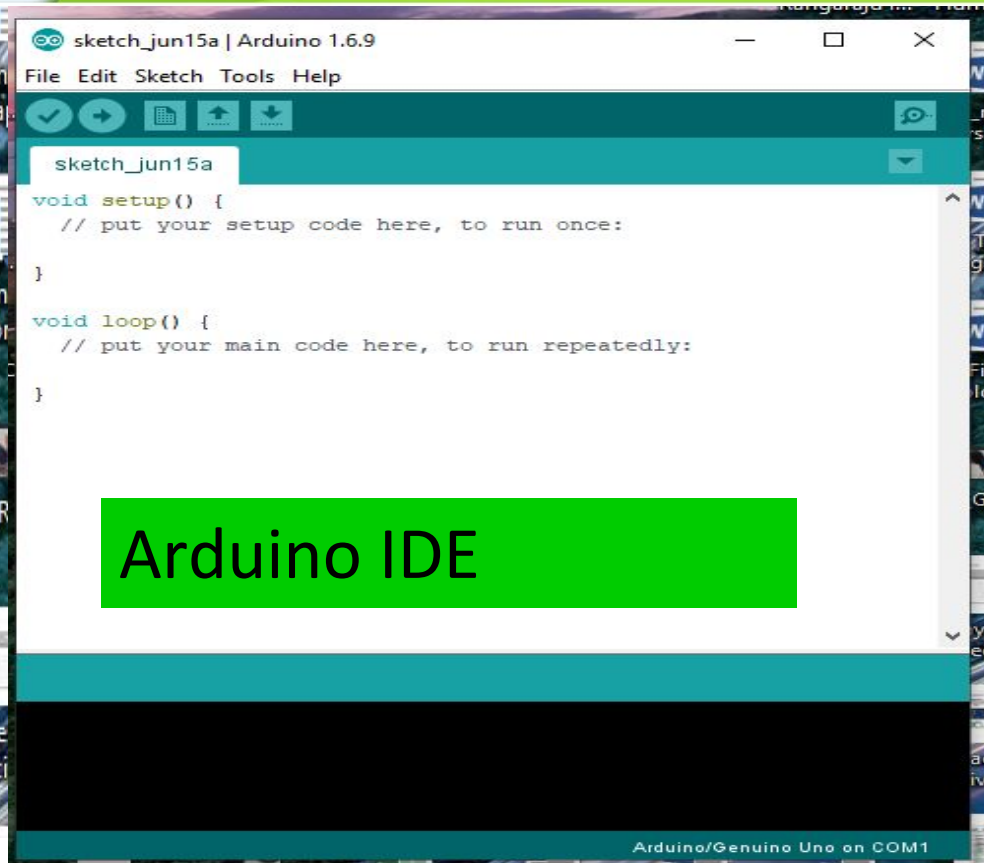


Set UP

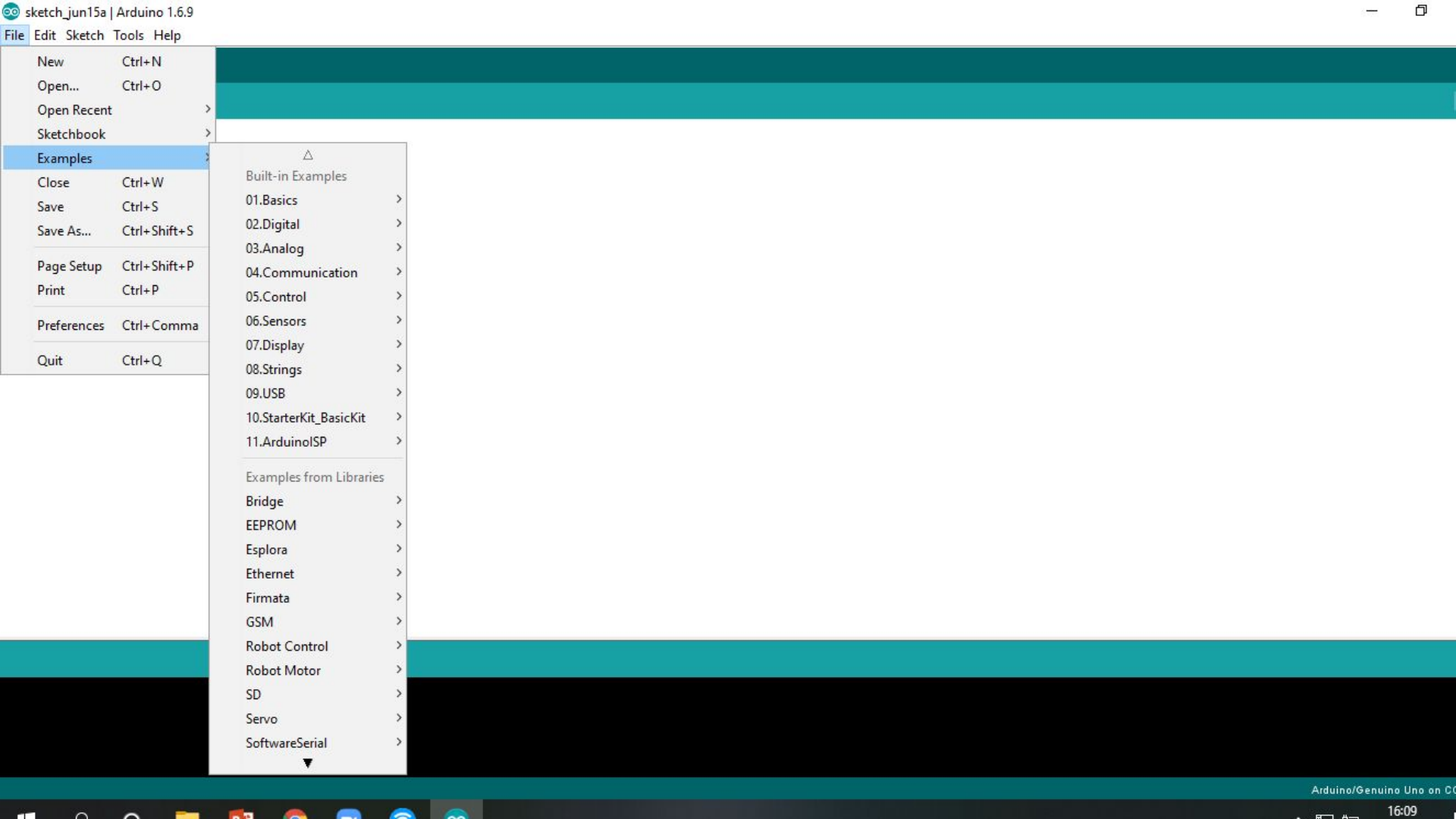
- Power the board by connecting it to a PC via USB cable
- Launch the Arduino IDE
- Set the board type and the port for the board
- TOOLS -> BOARD -> select your board
- TOOLS -> PORT -> select your port

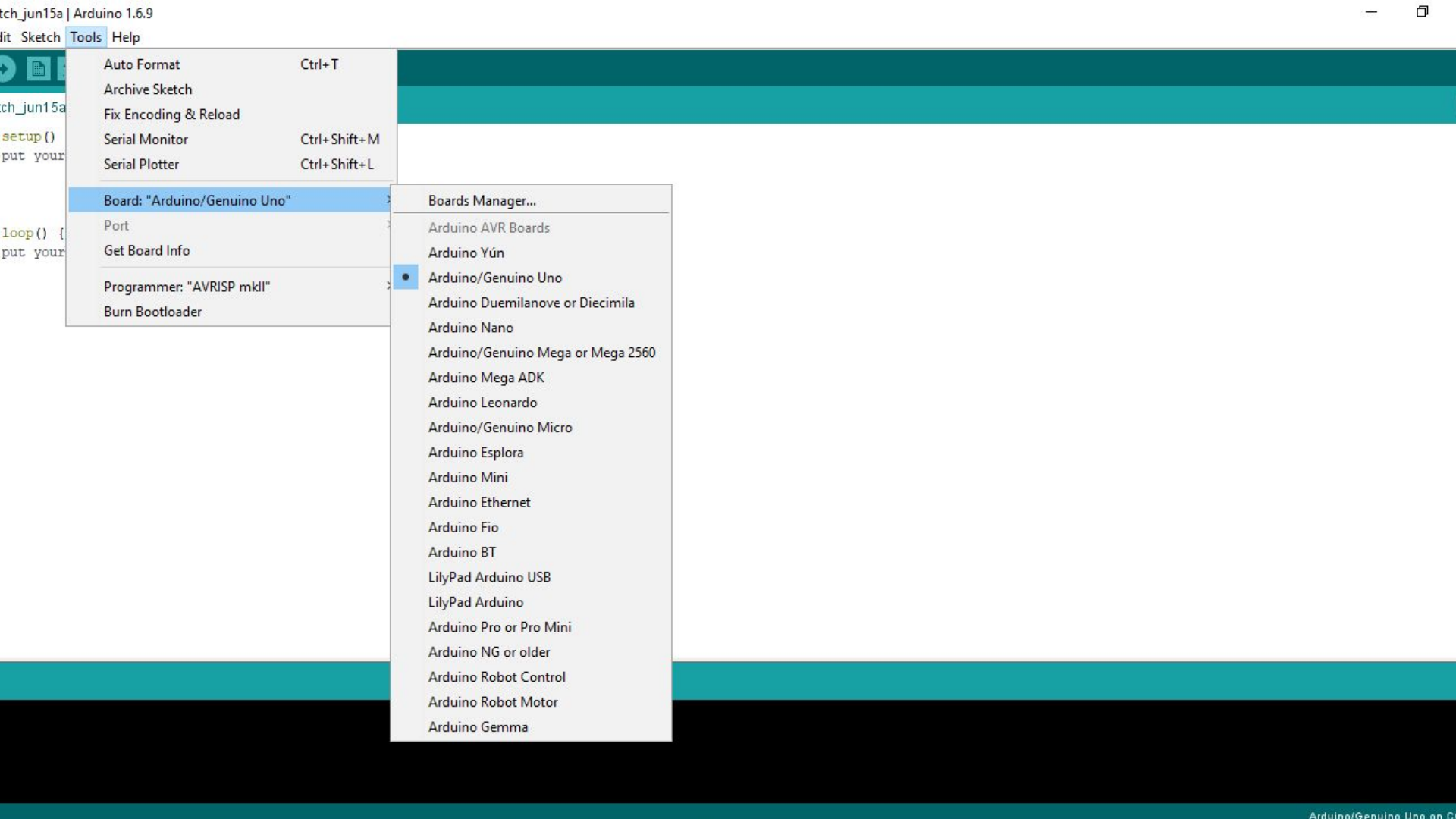
Launching Arduino





Arduino IDE

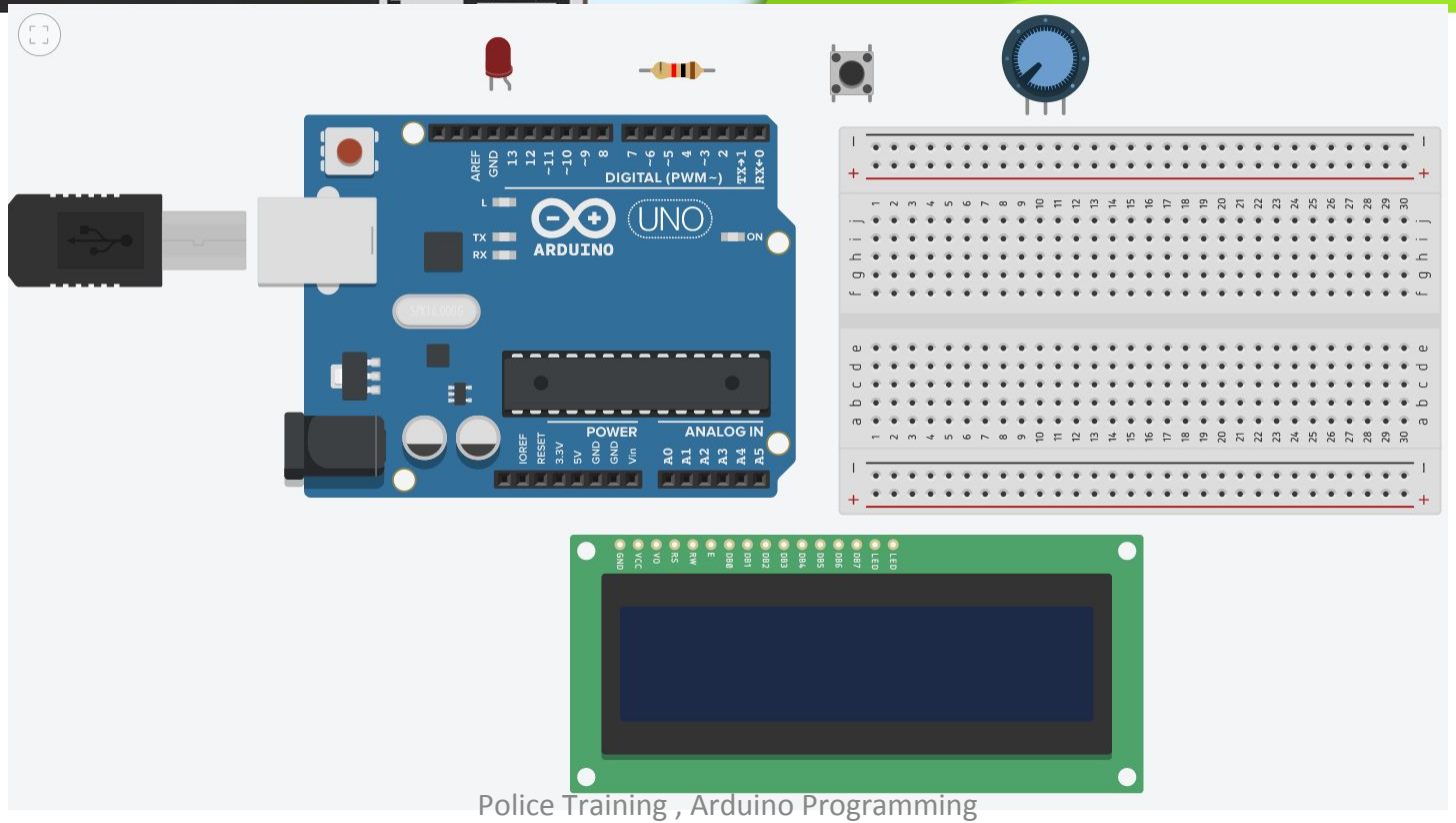

sketch_jun15a | Arduino 1.6.9
File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
Built-in Examples
01.Basics
02.Digital
03.Analog
04.Communication
05.Control
06.Sensors
07.Display
08.Strings
09.USB
10.StarterKit_BasicKit
11.ArduinoISP
Examples from Libraries
Bridge
EEPROM
Esplora
Ethernet
Firmata
GSM
Robot Control
Robot Motor
SD
Servo
SoftwareSerial
Arduino/Genuino Uno on C
16:09


tch_jun15a | Arduino 1.6.9
lit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Yún
Arduino/Genuino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino/Genuino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo
Arduino/Genuino Micro
Arduino Esplora
Arduino Mini
Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control
Arduino Robot Motor
Arduino Gemma

Thinkercad

- **Tinkercad** is a free, online 3D modeling program that runs in a web browser, known for its simplicity and ease of use.
- Circuits : Contains Basic plug and play components , arduino and some sensors.
- We can rig up circuit and simulate the model. Its behavior is exactly same as normal hardware model.
- We can develop prototype projects and later realize with hardware.
- No need to download any software it directly work in web browser

Components



Thinkercad

- Arduino Simulator

- Sample program
  - Blink LED
  - Blink Bulb using Relay

Arduino Functions

- Setup, Loop
- Digital IO functions
- Analog IO functions
- Serial Monitor functions
- Variables
- Operators

Arduino Functions

Setup() & loop() are the basic functions of ardunio. Setup function is used for defining port, pins and initiating devices and loop is to perform controlling actions.

```
void setup()
{
 Serial.begin(9600);
 pinMode(2, OUTPUT);
pinMode(A0,INPUT);
}
```

```
void loop()
{
   int x;
 Serial.println("arduino");
 x= analogRead(A0)
digitalWrite(2,x);
}
```

Digital IO functions

digitalRead(pin) , digitalWrite(pin,value) and pinMode(pin, status)

- digitalRead(pin) is used to read the boolean value of the pin used. The status can be HIGH or LOW.
- digitalWrite(pin, value) is used to send the status boolean value (HIGH or LOW) to the connected pin
- pinMode(pin, status) is used in setup function to define the selected pin status as INPUT or OUPTPUT

Analog IO functions

analogRead(pin) , analogWrite(pin,value)

- **analogRead(pin)** is used to read the value of the pin used. The values will be ranging from 0 to 1023 due to analog to digital convertor. Later it is mapped to the desired
- **analogWrite(pin, value)** is used to send the value to the devices connected to associated pin. Fading of LED is can be done by analogWrite() function.

Serial monitor functions

Serial.begin() , Serial.println(varible, format), Serial.available(), Serial.end()

- **Serial.begin(baud rate)** is used to activate serial monitor to be ready for reading or writing operations. Baud rate is the speed at which monitor operates. Normally it is used as 9600.
- **Serial.println(varible, format)** is used to print the variable value in the given format. The format can be HEX: hexadecimal, DEC: decimal.
- **Serial.available()** is used to check availability of the value for the variable in the serial monitor. It returns value greater than 0 if the variable is present.
- **Serial.end()** is used to close serial session

Variables & Control structures

Constants

HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false
Floating Point Constants
Integer Constants

Data Types

array
bool
byte
char
double
float
int

Bitwise Operators

& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Control structures

if
while
for

Programming with Arduino

```
int a =13;
float b=2.75;
char c='p';
  int d[3] ={3,2,8};
char
e[8]="arduino";
byte x=5;
byte y=7;
byte z;

void setup()
{

Serial.begin(9600);
}
```

```
void loop()
{

Serial.println("Practice
of serial commands");
Serial.println("a");
  Serial.println(a);
  Serial.println("b");
  Serial.println(b);
  Serial.println("c");
  Serial.println(c);
 Serial.println("int
array");
```

```
for(a=0;a<3;a++)
     Serial.println(d[a]);
  Serial.println("String");
   Serial.println(e);
  Serial.println("byte");
  Serial.println(x);
  Serial.println("byte");
  Serial.println(y);
  Serial.println("And
opeartion of x and Y");
  Serial.println(x & y);
 }
```

```
Practice of serial
commands
 a
13
 b
 2.75
c
p
int array
 3
2
 8
 String
 arduino
byte
5
Byte
 7
 And opeartion of x and Y
 5
```

x= 0000 0101

y= 0000 0111

x &y = 0000 0101 □5

X |y = 0000 0111 □7

Programming with Arduino

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

void loop()
{
  int x;
  while(Serial.available()>0)
  {
    x=Serial.read();
      Serial.println( "value of x is");
   Serial.write(x);
    Serial.println('\0');
  }
}
```

Output on Serial Monitor

```
value of x is
5
value of x is
 7
value of x is
 a
value of x is
 Z
```

Programming with Arduino

LED control through program

This experiment is aimed to practice the pinMode, digitalWrite functions of arduino.

Syntax: digitalWrite(pin, value)

pin: It is the pin number that is declared as OUTPUT in pinMode function.

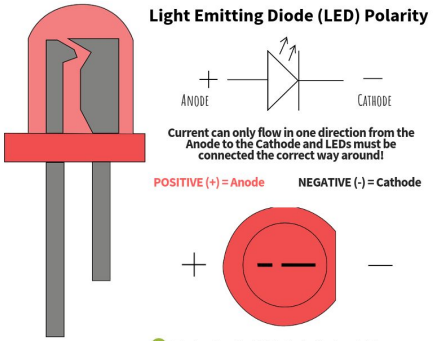
value: It is the boolean value LOW or HIGH

return : This function can not return any thing

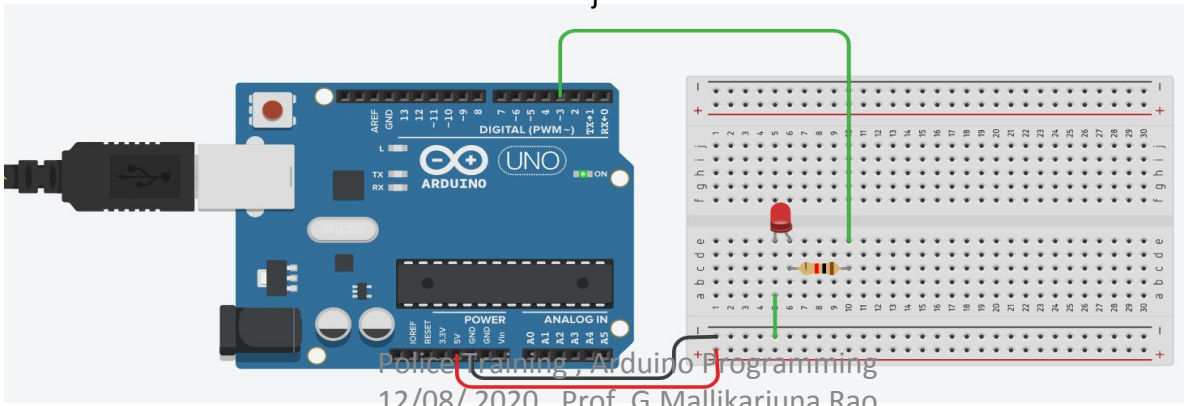
LED: Ligh Emiting Diode , when it is made conducting (anode to power supply and cathode to ground) it emits the light. The light colour is depend on its construction(Red or green light is produced by using Gallium-Phosphorus (GaP) as semiconductor).

Resistance: It LED is directly conned to the power supply it will draw huge current that will damage the diode. Hence to control the current a resistance is added serially.

LED & Interface diagram




```
void setup()
{
  pinMode(3, OUTPUT);
}
void loop()
{
  digitalWrite(3, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(3, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```



Programming with Arduino

LED & Switch Interface

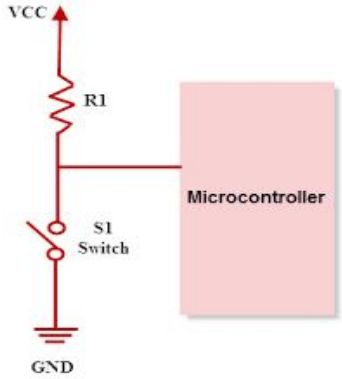
This experiment is aimed to expose digitalRead, digitalWrite and pinMode functions.

Syntax: digitalRead(pin)

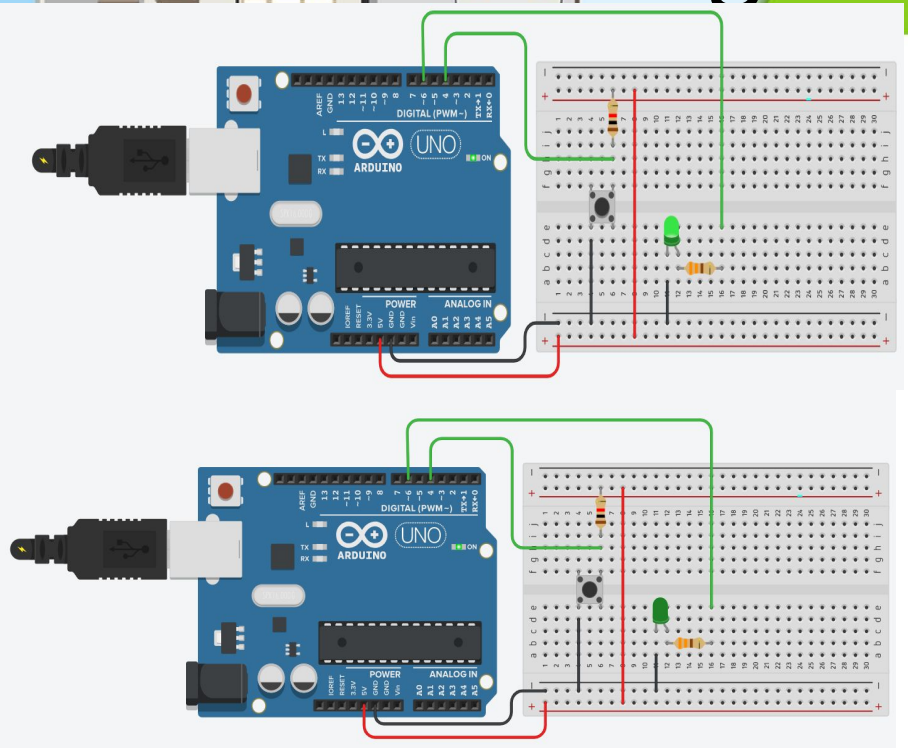
pin: It is the digial pin number which is declared as INPUT_PULLUP in pinMode function

Return: It return HIGH or LOW

Pullup resistor: It ensure released switch state is HIGH and pressed switch state is LOW. It overcome de-bouncing problem. It is connected in series with power supply and switch. The switch resistor junction is connected to arduino pin which is read as INPUT_PULLUP. The other end of the switch is connected to ground.



Interfacing Diagram




```
void setup()
{
  pinMode(6, OUTPUT);
  pinMode(4, INPUT_PULLUP);
}

void loop()
{
  bool x;
  x=digitalRead(4);
  digitalWrite(6,x);
}
```

Programming with Arduino

Fading of LED

This experiment is aimed to practice the pinMode, analogWrite functions of arduino.

Syntax: analogWrite(pin, brightness)

pin: It is the pin number that is declared as OUTPUT in pinMode function. This digital pin must be PWM (pulse width modulation pin). Here it can be any one of the digital pin 3,5,6 ,9,10,11.

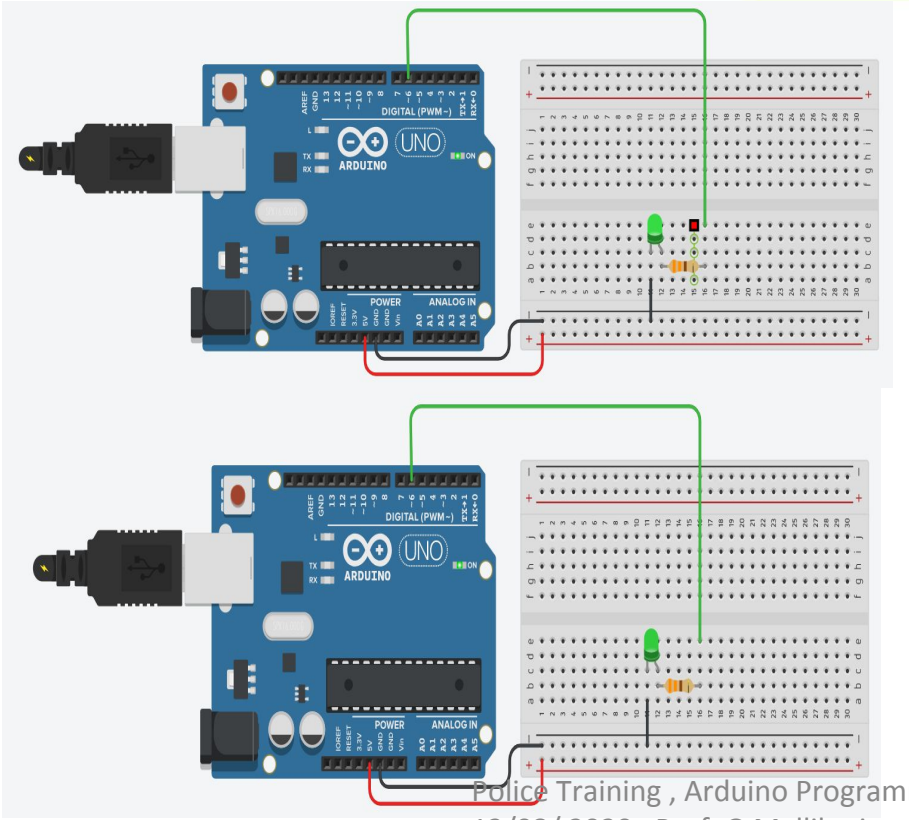
brightness: It is the value ranging from 0 to 255

return : This function will not return any thing

Delay value is going to control the duration on HIGH or LOW

PWM signals are **used** for a wide variety of control applications. Their main use is for controlling DC motors but it can also be **used** to control valves, pumps, hydraulics, and other mechanical parts.

Interfacing diagram




```
void setup()
{
  pinMode(6, OUTPUT);
}

void loop()
{
  int i;
  for(i=5; i<=255;i+=5)
  {
    analogWrite(6,i);
    delay(30);
    if (i == 255)
     i=5;
   }
}
```

Programming with Arduino

LED brightness control through Potentiameter

This experiment is aimed to practice the pinMode, analogWrite and analogRead functions of arduino.

Syntax: analogRead(pin)

pin: It is the pin number that is declared as INPUT in pinMode function.

Since analog pin internally connected to internal 10-bit analog to digital convertor the input value ranging from 0 to 1023.

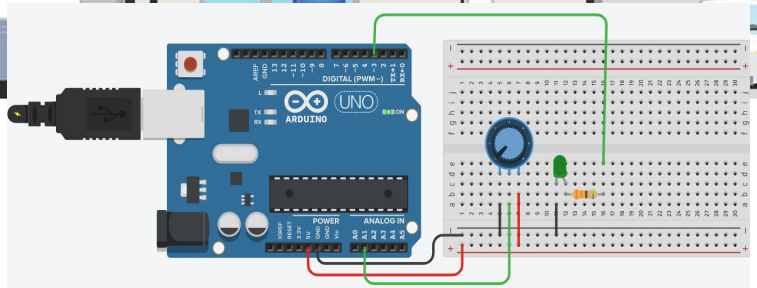
return : This function will return a value from 0 to 1023.

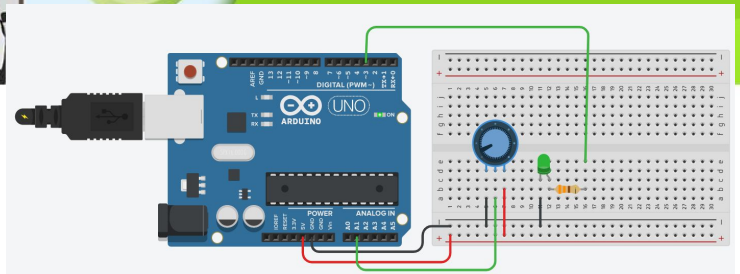
This value has to be mapped to the brightness value o to 255 using mapping function

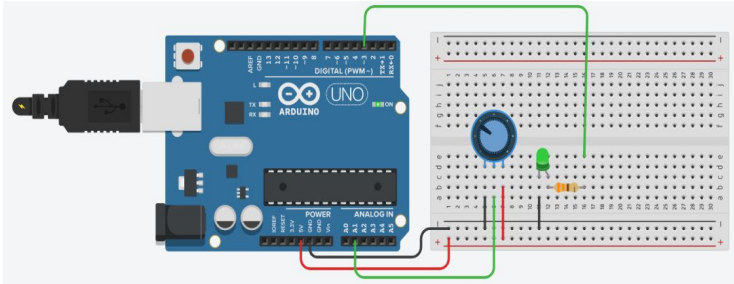
Syntax: x= **map(varialble, 0, 1023, 0,255);**

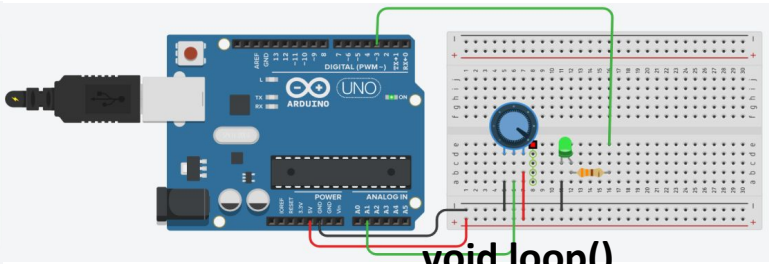
return: It return any value between 0 to 255 based on the variable value at that time.

Interface Diagram





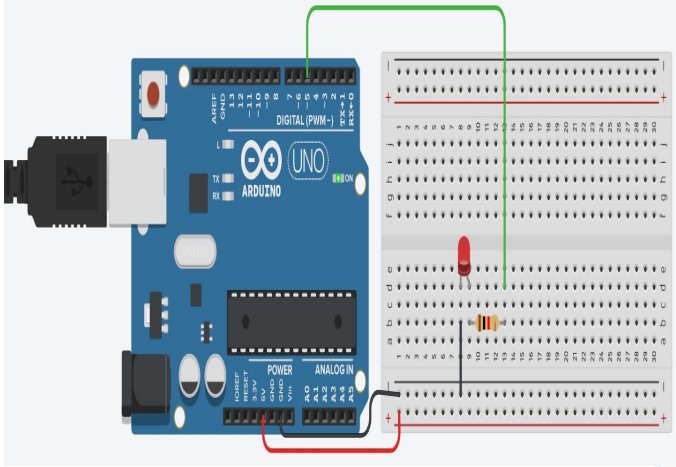




```
void setup()
{
  pinMode(A1,INPUT);
  pinMode(3, OUTPUT);
}
```

```
void loop()
{
  int x;
  x=analogRead(A1);
  x=
map(x,0,1023,0,255);
  analogWrite(3,x);
  delay(30); }
```

Virtual Switch



Text

```
void setup()
{
 Serial.begin(9600);
  pinMode(5,OUTPUT);
}

void loop()
{
  int x;
  while(Serial.available()>0)
  {
    x=Serial.read();
    x=x-48;
    if(x==1)
      digitalWrite(5,HIGH);
    if(x==0)
      digitalWrite(5,LOW);
  }
}
```

1 (Arduino Uno R3)

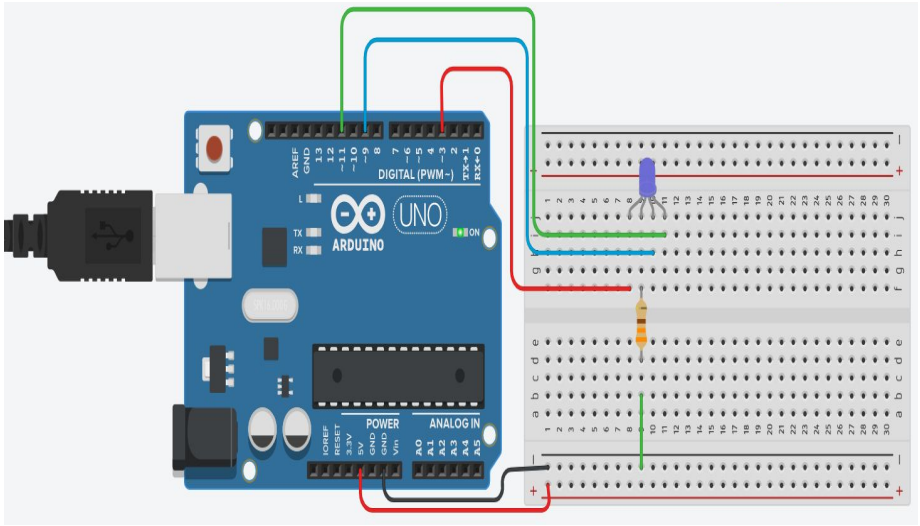
How it works

1. Add breakpoints by c
on the line numbers.
2. Start simulation.
3. Hover over the variab
while paused to see the
value.

Serial Monitor

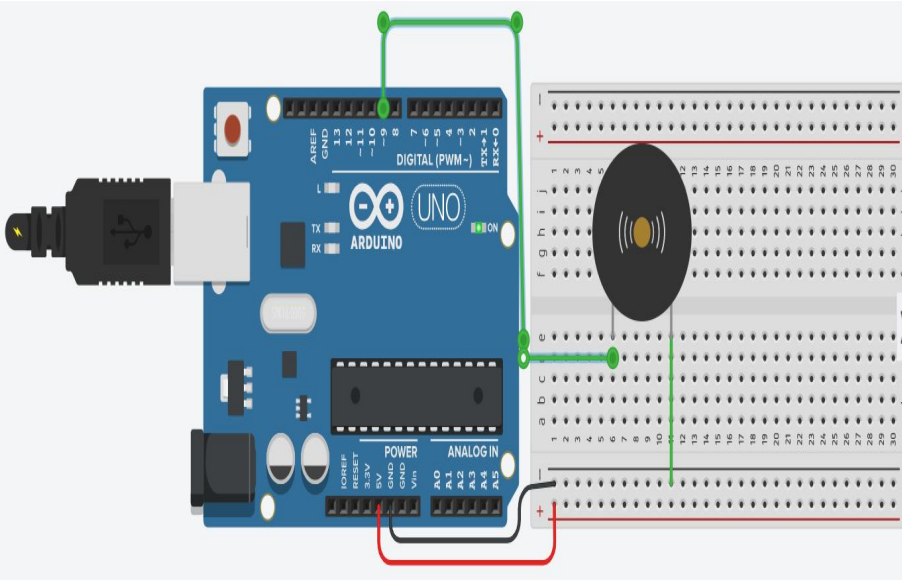
Send Clear

RGB FADING



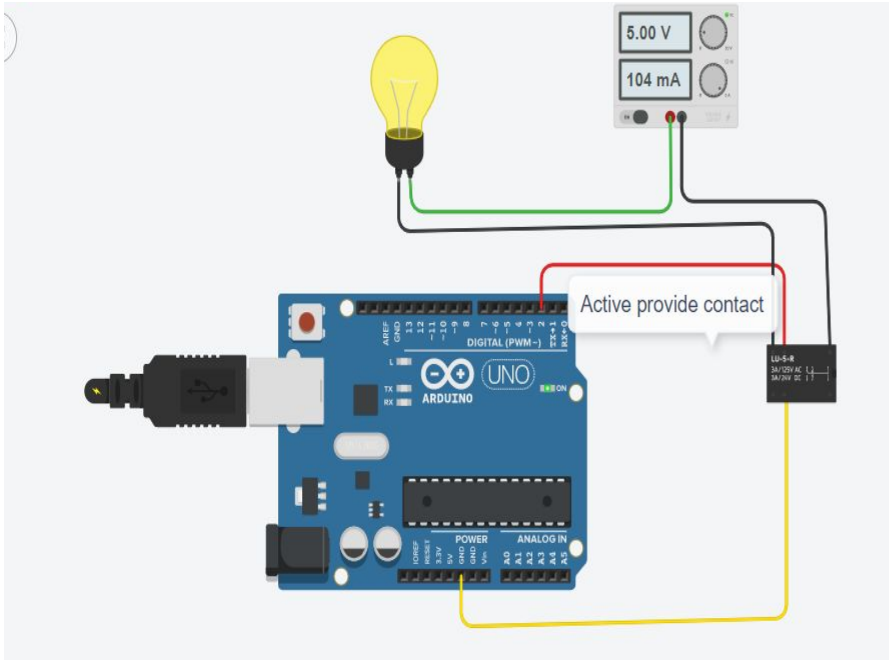

```
int i;
int j;
void setup()
{
   pinMode(3, OUTPUT);
   pinMode(9, OUTPUT);
   pinMode(11, OUTPUT);
 Serial.begin(9600);
}
void loop()
{
  while(Serial.available()>0)
 {
   j = Serial.read();
   j = j-48;
     switch(j)
   {
    case 0: fade(3);
            break;
   case 1: fade(9);
            break;
   case 2: fade(11);
           break;
    }
  }
}
void fade(int k)
{
 for(i=0;i<255;i++)
         {
          analogWrite(k,i);
          delay(10);
         }
      for(i=255;i>0;i--)
         {
         analogWrite(k,i);
          delay(10);
          }
}
```

Buzzer



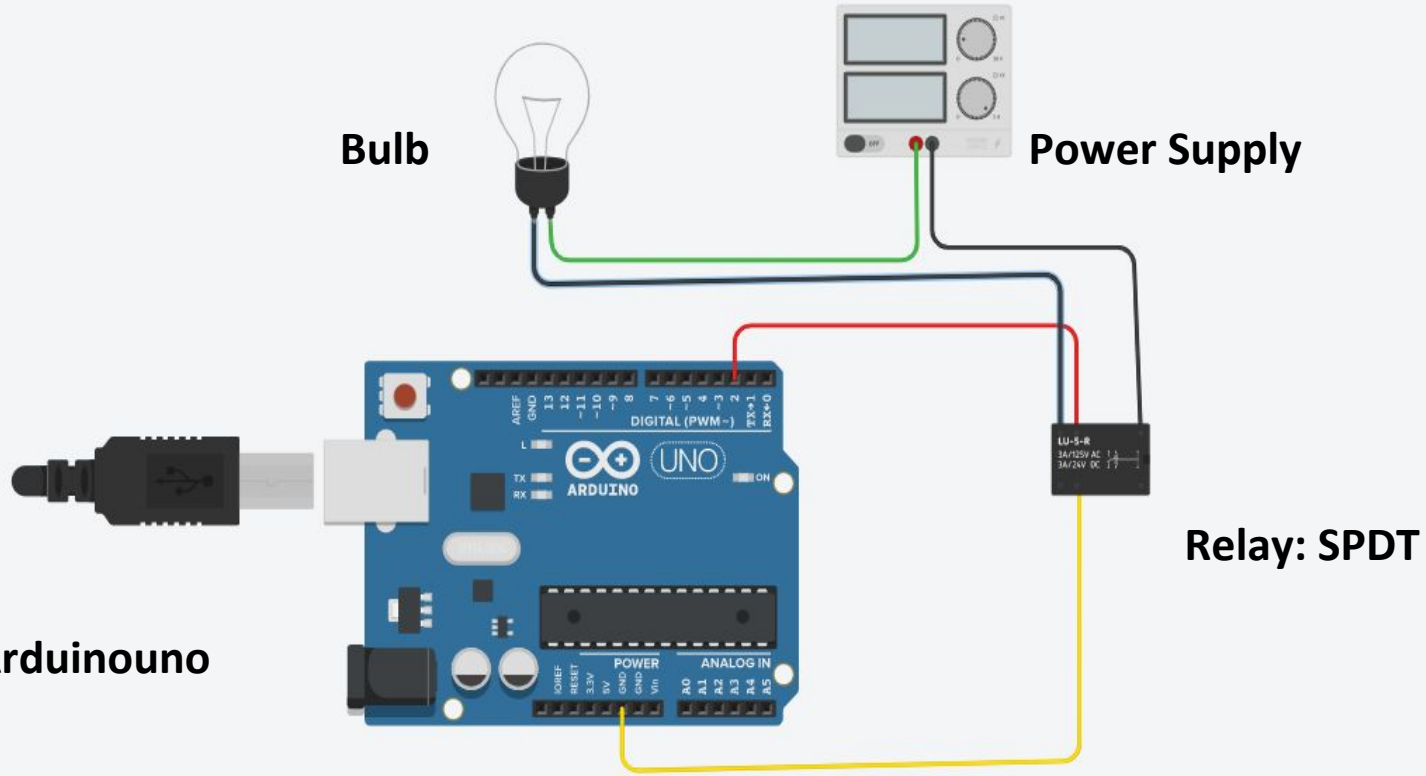

```
void setup()
{
  pinMode(9, OUTPUT);
}

void loop()
{
  int i;
  for(i=5;i<=255;i++)
  {
    analogWrite(9,i);
     delay(30);
    if (i==255)
       i=5;
  }
}
```

Relay Bulb Interfacing




```
int led=2;
void setup()
{
  pinMode(2,OUTPUT);

}
void loop()
{
  digitalWrite(2,HIGH);
    delay(1000);
  digitalWrite(2,LOW);
  delay(1000);
}
```

Bulb Blinking Using Relay



Any Quires…?
Thank You

Check Your knowledge

Newton Classroom | Meet - CSE - III, Micro Contr | meet.google.com/dmd-vspr-svu

Apps | esp32competetion... | GPU Accelerated C... | Numba_041_releas... | JNTUH | GitHub - GRIETIOTL... | https://griet.newto... | Newton Classroom | III B.TECH CSE -I SE... | Reading list

D — Dharavath Divya Dharavath

Y — Yalavarthi Sreeja Yalavarthi

H — Harish Rohan Kambhampaty Kam...

S — Saginala Raju Saginala

R — Rumana Tarannum Tarannum

K

A

M B — 36 others

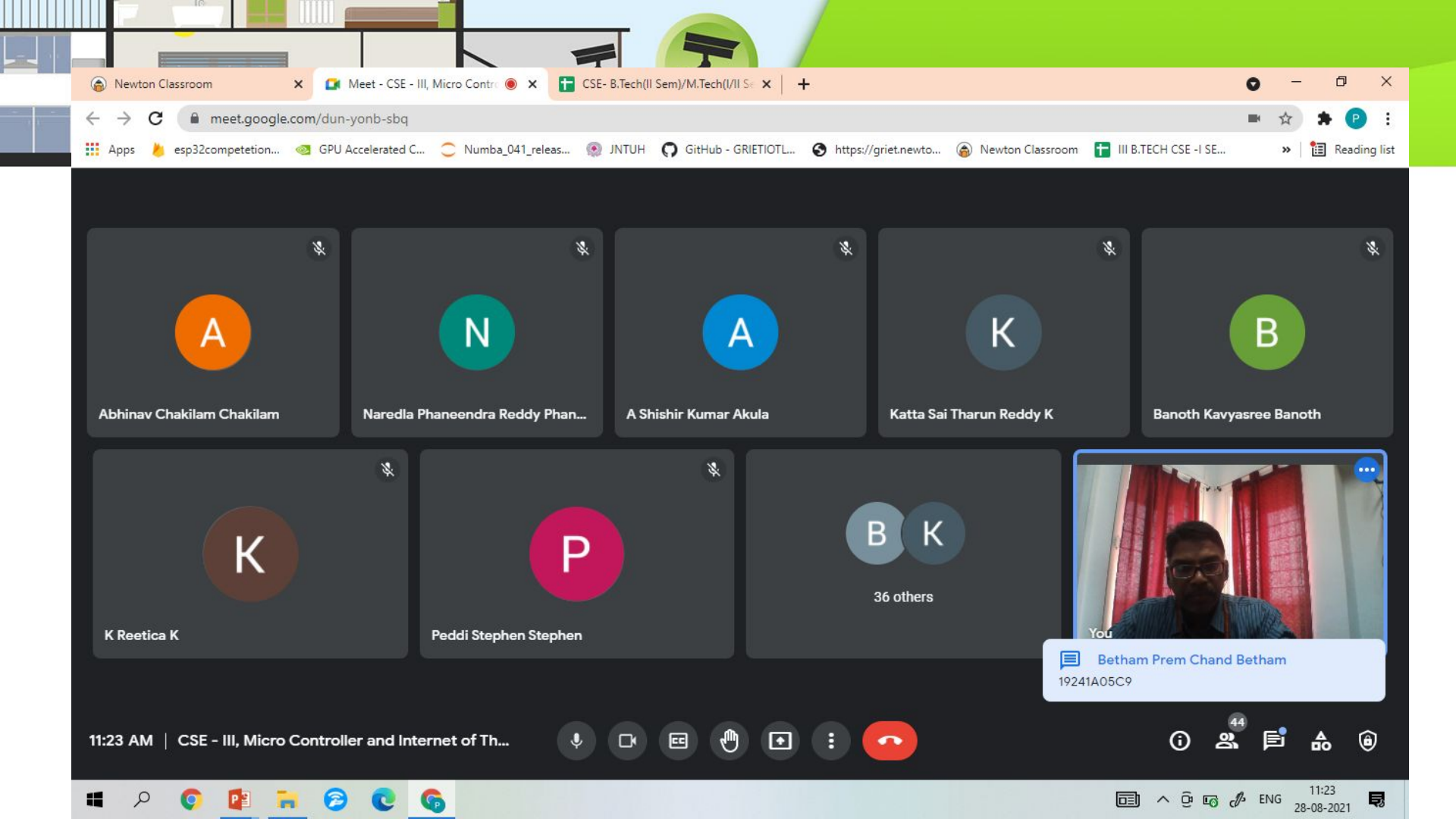
To see more people, change your layout to show more tiles

Change layout

10:12 AM | CSE - III, Micro Controller and Internet of Th...

44

ENG 10:12 28-08-2021


Newton Classroom
Meet - CSE - III, Micro Contr
CSE- B.Tech(II Sem)/M.Tech(I/II S
meet.google.com/dun-yonb-sbq
Apps
esp32competetion...
GPU Accelerated C...
Numba_041_releas...
JNTUH
GitHub - GRIETIOTL...
https://griet.newto...
Newton Classroom
III B.TECH CSE -I SE...
Reading list
A
Abhinav Chakilam Chakilam
N
Naredla Phaneendra Reddy Phan...
A
A Shishir Kumar Akula
K
Katta Sai Tharun Reddy K
B
Banoth Kavyasree Banoth
K
K Reetica K
P
Peddi Stephen Stephen
B K
36 others
You
Betham Prem Chand Betham
19241A05C9
11:23 AM | CSE - III, Micro Controller and Internet of Th...
44
ENG
11:23
28-08-2021

Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
http://arduino.cc/en/Reference/

Structure & Flow

Basic Program Structure

```
void setup() {
  // Runs once when sketch starts
}
void loop() {
  // Runs repeatedly
}
```

Control Structures

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;     // Exit a loop immediately
continue;  // Go to next iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
return x;  // x must match return type
return;    // For void return type
```

Function Definitions

```
<ret. type> <name>(<params>) { ... }
e.g. int double(int x) {return x*2;}
```

Operators

General Operators

=	assignment		
+	add	-	subtract
*	multiply	/	divide
%	modulo		
==	equal to	!=	not equal to
<	less than	>	greater than
<=	less than or equal to		
>=	greater than or equal to		
&&	and	||	or
!	not		

Compound Operators

++	increment
--	decrement
+=	compound addition
-=	compound subtraction
*=	compound multiplication
/=	compound division
&=	compound bitwise and
|=	compound bitwise or

Bitwise Operators

&	bitwise and	|	bitwise or
^	bitwise xor	~	bitwise not
<<	shift left	>>	shift right

Pointer Access

&	reference: get a pointer
*	dereference: follow a pointer

Built-in Functions

Pin Input/Output

```
Digital I/O - pins 0-13 A0-A5
  pinMode(pin,
    [INPUT, OUTPUT, INPUT_PULLUP])
  int digitalread(pin)
  digitalWrite(pin, [HIGH, LOW])

Analog In - pins A0-A5
  int analogRead(pin)
  analogReference(
    [DEFAULT, INTERNAL, EXTERNAL])

PWM Out - pins 3 5 6 9 10 11
  analogWrite(pin, value)
```

Advanced I/O

```
tone(pin, freq_Hz)
tone(pin, freq_Hz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
  [MSBFIRST, LSBFIRST], value)
unsigned long pulseIn(pin,
  [HIGH, LOW])
```

Time

```
unsigned long millis()
  // Overflows at 50 days
unsigned long micros()
  // Overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)
```

Math

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
```

Random Numbers

```
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)
```

Bits and Bytes

```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Type Conversions

```
char(val)     byte(val)
int(val)      word(val)
long(val)     float(val)
```

External Interrupts

```
attachInterrupt(interrupt, func,
  [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```

Libraries

```
Serial - comm. with PC or via RX/TX
begin(long speed)  // Up to 115200
end()
int available()  // #bytes available
int read()   // -1 if none available
int peek()   // Read w/o removing
flush()
print(data)     println(data)
write(byte)     write(char * string)
write(byte * data, size)
SerialEvent()  // Called if data rdy

SoftwareSerial.h - comm. on any pin
SoftwareSerial(rxPin, txPin)
begin(long speed)  // Up to 115200
listen()        // Only 1 can listen
isListening()  // at a time.
read, peek, print, println, write
  // Equivalent to Serial library

EEPROM.h - access non-volatile memory
byte read(addr)
write(addr, byte)
EEPROM[index]  // Access as array

Servo.h - control servo motors
attach(pin, [min_uS, max_uS])
write(angle)  // 0 to 180
writeMicroseconds(uS)
   // 1000-2000; 1500 is midpoint
int read()    // 0 to 180
bool attached()
detach()

Wire.h - I²C communication
begin()      // Join a master
begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte)              // Step 2
send(char * string)
send(byte * data, size)
endTransmission()       // Step 3
int available() // #bytes available
byte receive()  // Get next byte
onReceive(handler)
onRequest(handler)
```

Variables, Arrays, and Data

Data Types

boolean	true | false
char	-128 - 127, 'a' '$' etc.
unsigned char	0 - 255
byte	0 - 255
int	-32768 - 32767
unsigned int	0 - 65535
word	0 - 65535
long	-2147483648 - 2147483647
unsigned long	0 - 4294967295
float	-3.4028e+38 - 3.4028e+38
double	currently same as float
void	i.e., no return value

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // Includes \0 null termination
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

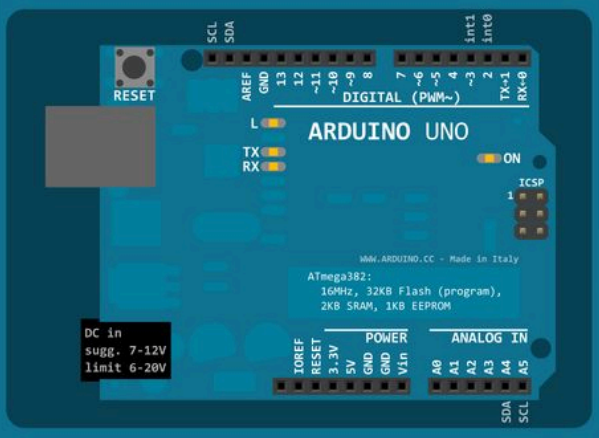
123	decimal
0b01111011	binary
0173	octal - base 8
0x7B	hexadecimal - base 16
123U	force unsigned
123L	force long
123UL	force unsigned long
123.0	force floating point
1.23e6	1.23*10^6 = 1230000

Qualifiers

static	persists between calls
volatile	in RAM (nice for ISR)
const	read-only
PROGMEM	in flash

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};
int myInts[6];   // Array of 6 ints
myInts[0] = 42;  // Assigning first
                 // index of myInts
myInts[6] = 12;  // ERROR! Indexes
                 // are 0 though 5
```



Adapted from:
- Original: Gavin Smith
- SVG version: Frederic Dufourg
- Arduino board drawing: Fritzing.org

Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
http://arduino.cc/en/Reference/

Structure & Flow

Basic Program Structure

```
void setup() {
  // Runs once when sketch starts
}
void loop() {
  // Runs repeatedly
}
```

Control Structures

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;     // Exit a loop immediately
continue;  // Go to next iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
return x;  // x must match return type
return;    // For void return type
```

Function Definitions

```
<ret. type> <name>(<params>) { ... }
e.g. int double(int x) {return x*2;}
```

Operators

General Operators

=	assignment		
+	add	-	subtract
*	multiply	/	divide
%	modulo		
==	equal to	!=	not equal to
<	less than	>	greater than
<=	less than or equal to		
>=	greater than or equal to		
&&	and	||	or
!	not		

Compound Operators

++	increment
--	decrement
+=	compound addition
-=	compound subtraction
*=	compound multiplication
/=	compound division
&=	compound bitwise and
|=	compound bitwise or

Bitwise Operators

&	bitwise and	|	bitwise or
^	bitwise xor	~	bitwise not
<<	shift left	>>	shift right

Pointer Access

&	reference: get a pointer
*	dereference: follow a pointer

Variables, Arrays, and Data

Data Types

boolean	true | false
char	-128 - 127, 'a' '$' etc.
unsigned char	0 - 255
byte	0 - 255
int	-32768 - 32767
unsigned int	0 - 65535
word	0 - 65535
long	-2147483648 - 2147483647
unsigned long	0 - 4294967295
float	-3.4028e+38 - 3.4028e+38
double	currently same as float
void	i.e., no return value

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // Includes \0 null termination
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

123	decimal
0b01111011	binary
0173	octal - base 8
0x7B	hexadecimal - base 16
123U	force unsigned
123L	force long
123UL	force unsigned long
123.0	force floating point
1.23e6	1.23*10^6 = 1230000

Qualifiers

static	persists between calls
volatile	in RAM (nice for ISR)
const	read-only
PROGMEM	in flash

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};
int myInts[6];   // Array of 6 ints
myInts[0] = 42;  // Assigning first
                 // index of myInts
myInts[6] = 12;  // ERROR! Indexes
                 // are 0 though 5
```

Built-in Functions

Pin Input/Output

```
Digital I/O - pins 0-13 A0-A5
  pinMode(pin,
    [INPUT, OUTPUT, INPUT_PULLUP])
  int digitalread(pin)
  digitalWrite(pin, [HIGH, LOW])

Analog In - pins A0-A5
  int analogRead(pin)
  analogReference(
    [DEFAULT, INTERNAL, EXTERNAL])

PWM Out - pins 3 5 6 9 10 11
  analogWrite(pin, value)
```

Advanced I/O

```
tone(pin, freq_Hz)
tone(pin, freq_Hz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
  [MSBFIRST, LSBFIRST], value)
unsigned long pulseIn(pin,
  [HIGH, LOW])
```

Time

```
unsigned long millis()
  // Overflows at 50 days
unsigned long micros()
  // Overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)
```

Math

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
```

Random Numbers

```
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)
```

Bits and Bytes

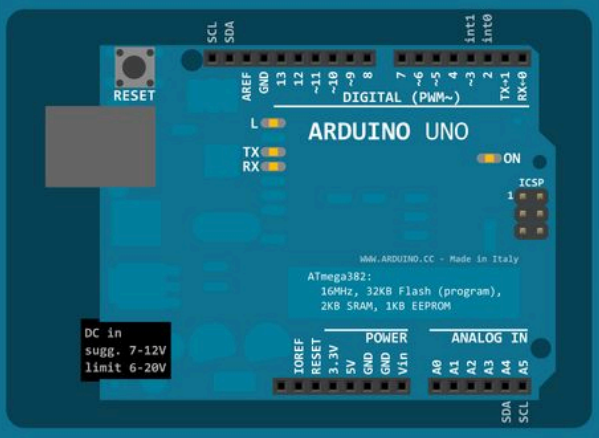
```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Type Conversions

```
char(val)     byte(val)
int(val)      word(val)
long(val)     float(val)
```

External Interrupts

```
attachInterrupt(interrupt, func,
  [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```



Libraries

```
Serial - comm. with PC or via RX/TX
begin(long speed)  // Up to 115200
end()
int available()  // #bytes available
int read()   // -1 if none available
int peek()   // Read w/o removing
flush()
print(data)     println(data)
write(byte)     write(char * string)
write(byte * data, size)
SerialEvent()  // Called if data rdy

SoftwareSerial.h - comm. on any pin
SoftwareSerial(rxPin, txPin)
begin(long speed)  // Up to 115200
listen()        // Only 1 can listen
isListening()  // at a time.
read, peek, print, println, write
  // Equivalent to Serial library

EEPROM.h - access non-volatile memory
byte read(addr)
write(addr, byte)
EEPROM[index]  // Access as array

Servo.h - control servo motors
attach(pin, [min_uS, max_uS])
write(angle)  // 0 to 180
writeMicroseconds(uS)
   // 1000-2000; 1500 is midpoint
int read()    // 0 to 180
bool attached()
detach()

Wire.h - I²C communication
begin()      // Join a master
begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte)              // Step 2
send(char * string)
send(byte * data, size)
endTransmission()       // Step 3
int available() // #bytes available
byte receive()  // Get next byte
onReceive(handler)
onRequest(handler)
```