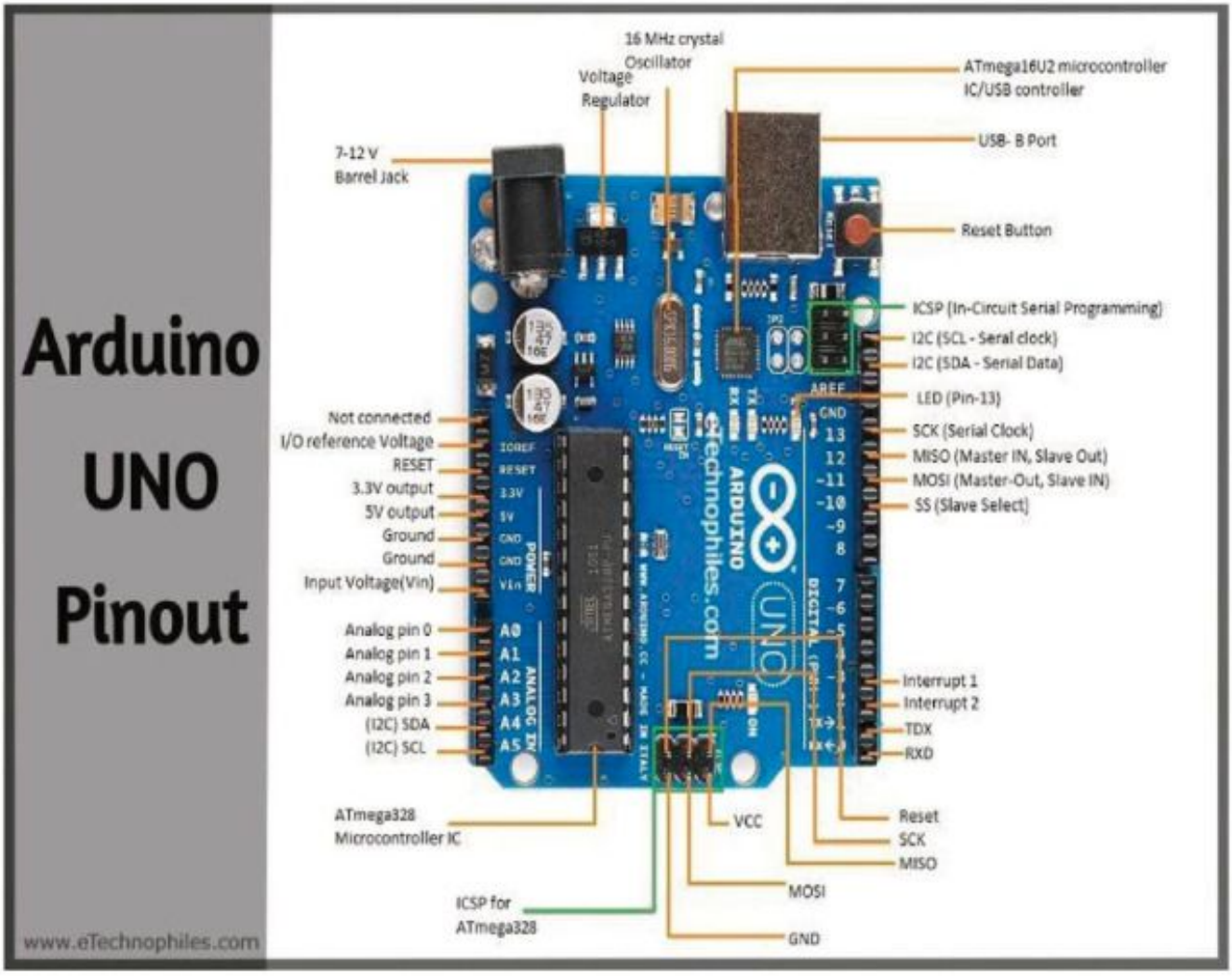
AVR Bare Metal Programming

Arduino :

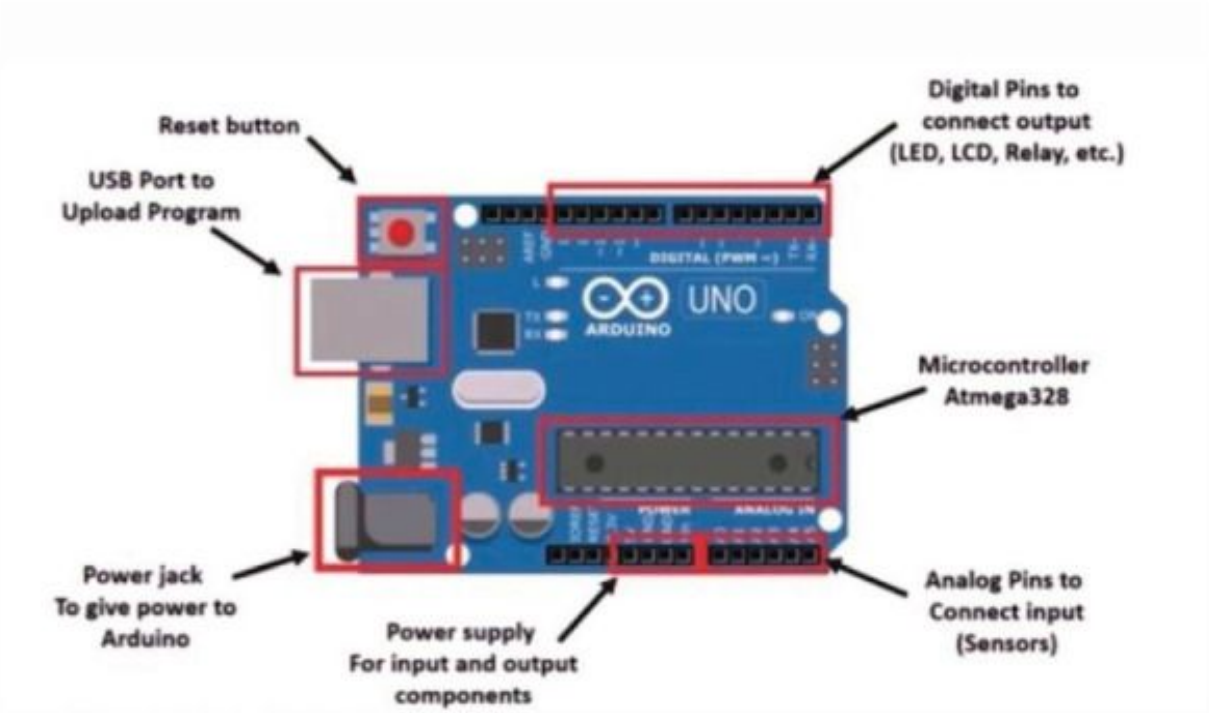
Arduino is an open-source electronics platform based on easy-to-use hardware and software.

Arduino boards are able to read inputs - light on a sensor, button, or message - and turn it into an output - activating a motor, turning on an LED, publishing something online.



There are two functions One is called setup(), the other is called loop(). The first is called once, when the program starts, the second is repeatedly called while your program is running.

Part of the Arduino Programming Language is the built-in libraries that allow you to easily integrate with the functionality provided by the Arduino board.



Your first Arduino program will surely involve making a led turn on the light, and then turn off. To do so, you will use the pinMode(), delay() and digitalWrite() functions, along with some constants like HIGH, LOW, OUTPUT.

```
#define LED_PIN 13

void setup() {
  // Configure pin 13 to be a digital output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // Turn on the LED
  digitalWrite(LED_PIN, HIGH);
  // Wait 1 second (1000 milliseconds)
  delay(1000);
  // Turn off the LED
  digitalWrite(LED_PIN, LOW);
  // Wait 1 second
```

```
  delay(1000);
}
```

Digital I/O

- **digitalRead()** reads the value from a digital pin. Accepts a pin number as a parameter, and returns the HIGH or LOW constant.
- **digitalWrite()** writes a HIGH or LOW value to a digital output pin. You pass the pin number and HIGH or LOW as parameters.
- **pinMode()** sets a pin to be an input, or an output. You pass the pin number and the INPUT or OUTPUT value as parameters.
- pulseIn() reads a digital pulse from LOW to HIGH and then to LOW again, or from HIGH to LOW and to HIGH again on a pin. The program will block until the pulse is detected. You specify the pin number and the kind of pulse you want to detect (LHL or HLH). You can specify an optional timeout to stop waiting for that pulse.
- pulseInLong() is same as pulseIn(), except it is implemented differently and it can't be used if interrupts are turned off. Interrupts are commonly turned off to get a more accurate result.
- shiftIn() reads a byte of data one bit at a time from a pin.
- shiftOut() writes a byte of data one bit at a time to a pin.
- tone() sends a square wave on a pin, used for buzzers/speakers to play tones. You can specify the pin, and the frequency. It works on both digital and analog pins.
- noTone() stops the tone() generated wave on a pin.

Analog I/O

- analogRead() reads the value from an analog pin.
- analogReference() configures the value used for the top input range in the analog input, by default 5V in 5V boards and 3.3V in 3.3V boards.
- analogWrite() writes an analog value to a pin

- analogReadResolution() lets you change the default analog bits resolution for analogRead(), by default 10 bits. Only works on specific devices (Arduino Due, Zero and MKR)
- analogWriteResolution() lets you change the default analog bits resolution for analogWrite(), by default 10 bits. Only works on specific devices (Arduino Due, Zero and MKR)

Time functions

- **delay()** pauses the program for a number of milliseconds specified as parameter
- delayMicroseconds() pauses the program for a number of microseconds specified as parameter

Arduino sets two constants we can use to

HIGH equates to a high level of voltage, which can differ depending on the hardware (>2V on 3.3V boards like Arduino Nano, >3V on 5V boards like Arduino Uno) LOW equates to a low level of voltage. Again, the exact value depends on the board used

Then we have 3 constants we can use in combination with the pinMode() function:

- INPUT sets the pin as an input pin
- OUTPUT sets the pin as an output pin
- INPUT_PULLUP sets the pin as an internal pull-up resistor

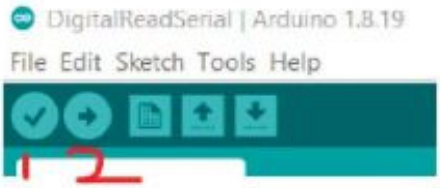
The other constant we have is LED_BUILTIN, which points to the number of the on-board pin, which usually equates to the number 13.

Process to check the memory is occupied by the code using Arduino library and the bare metal programming:

LED Blink program:

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);  // pinMode  will configure pin as input or output
}


// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

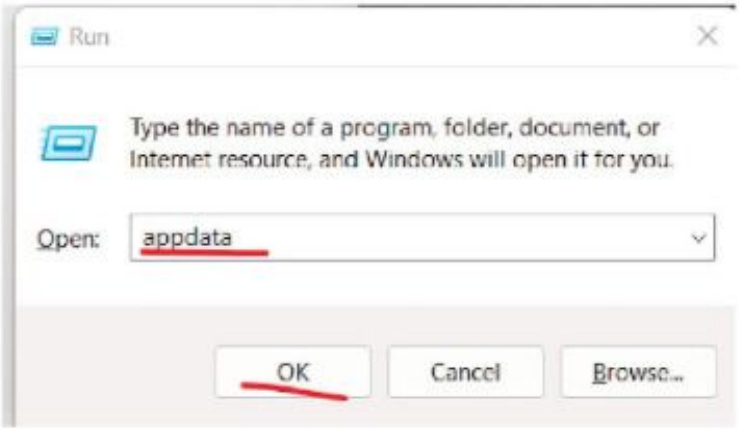
Write the program in the Arduino IDE verify it-> uploade it



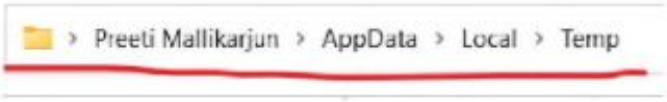
Go to Arduino Directory folder go in this path "Hardware ->tools ->avr ->bin", type **cmd** in address bar

then type avr-size.exe-

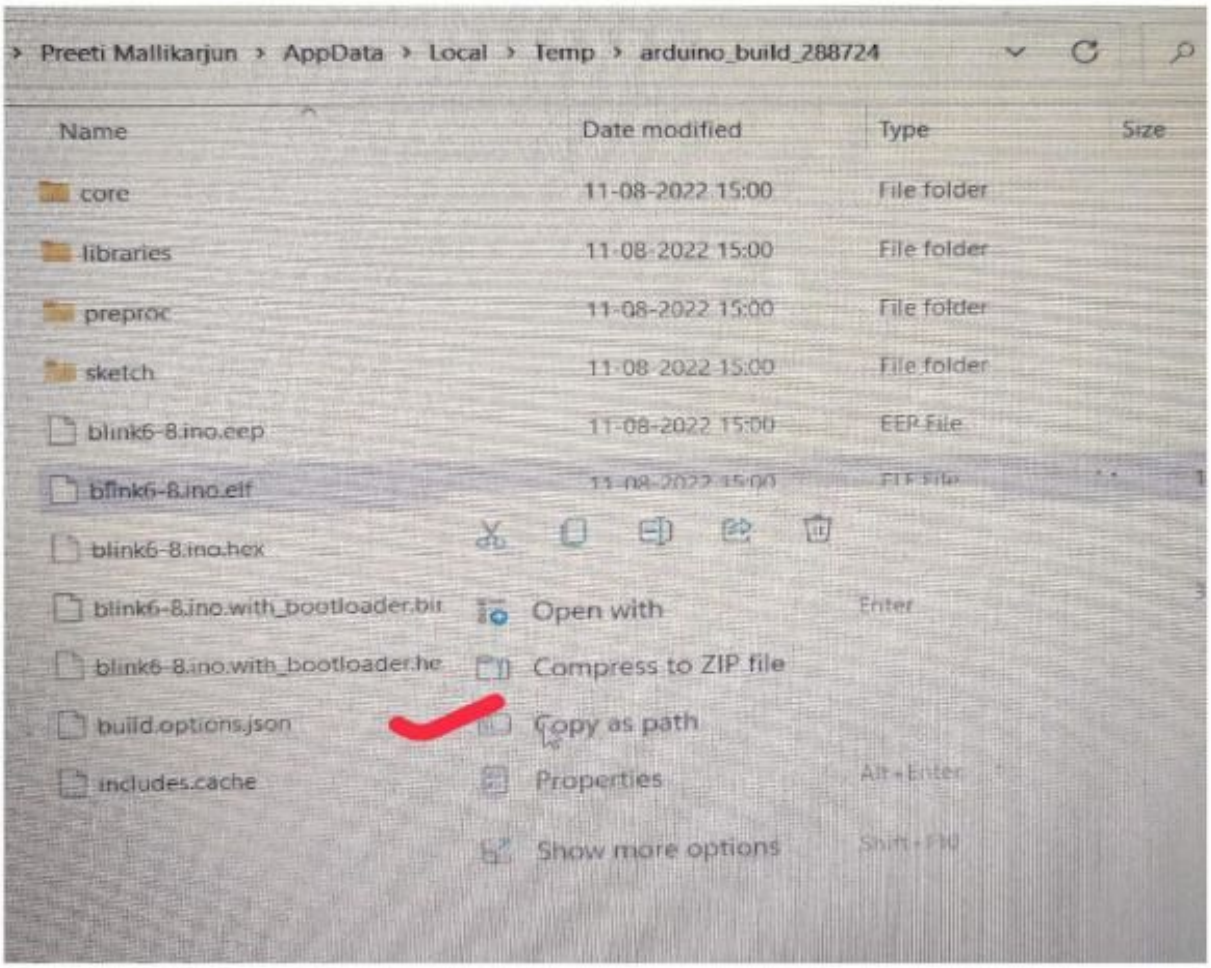
→ press windows r →run window pop ups



Appdata→Local->Temp



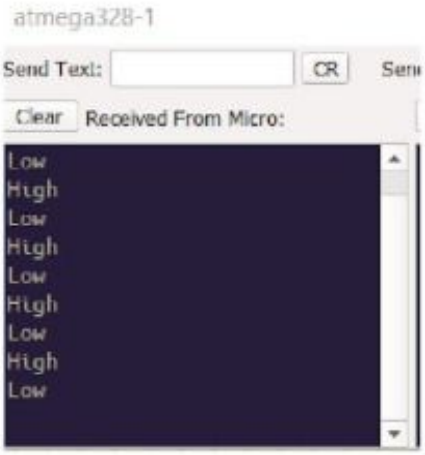
->Arduino_build_xxxxx(in this choose ur program hex file->right click->select as path

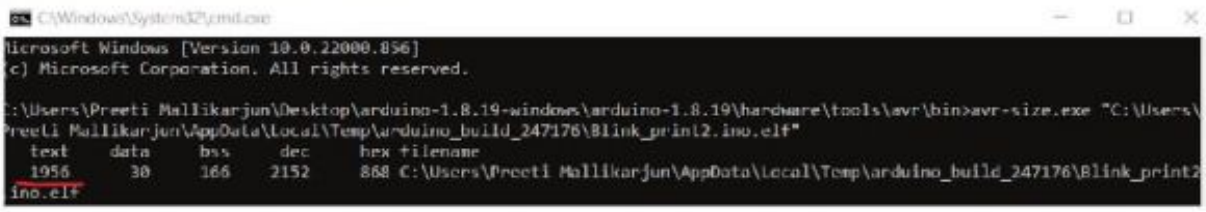


In cmd write avr-size.exe space copied path\name of elf file->enter

2. Another example for LED Blink program using Serial.println function

```
void setup() {
 // initialize digital pin LED_BUILTIN as an output.
 pinMode(LED_BUILTIN, OUTPUT);
Serial.begin(9600);
}
// the loop function runs over and over again forever
void loop() {
 digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
 Serial.println("ON");
 delay(1000);                       // wait for a second
 digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
 Serial.println("OFF");
 delay(1000);                       // wait for a second
}
```





Measuring clock cycles taken by the instructions using pinMode() functions

```
void setup()
{
  // put your setup code here, to run once:
  TCCR1B = bit(CS10); //configure the timer
  TCNT1 = 0; //set the counter to 0
pinMode(LED_BUILTIN,OUTPUT);
unsigned int cycles = TCNT1;
Serial.begin(9600);
Serial.println(cycles);
}
void loop()
{
  // put your main code here, to run repeatedly:
digitalWrite(LED_BUILTIN,HIGH);
delay(500);
digitalWrite(LED_BUILTIN,LOW);
delay(500);
}
```

To count clock cycles we need the timer and counter so we need to configure it

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

15.11.2 TCCR1B – Timer/Counter1 Control Register B

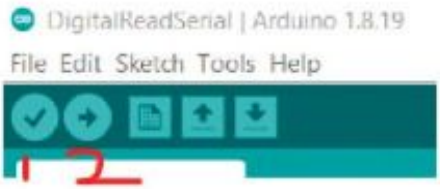
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-6. Clock Select Bit Description

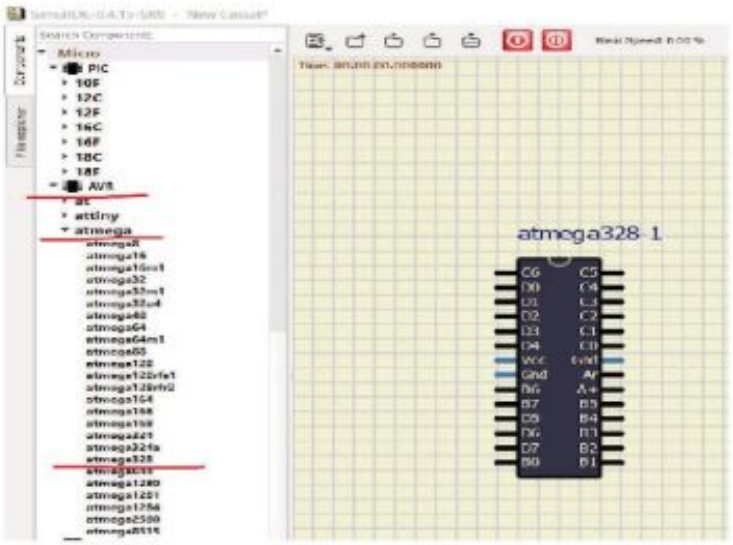
CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

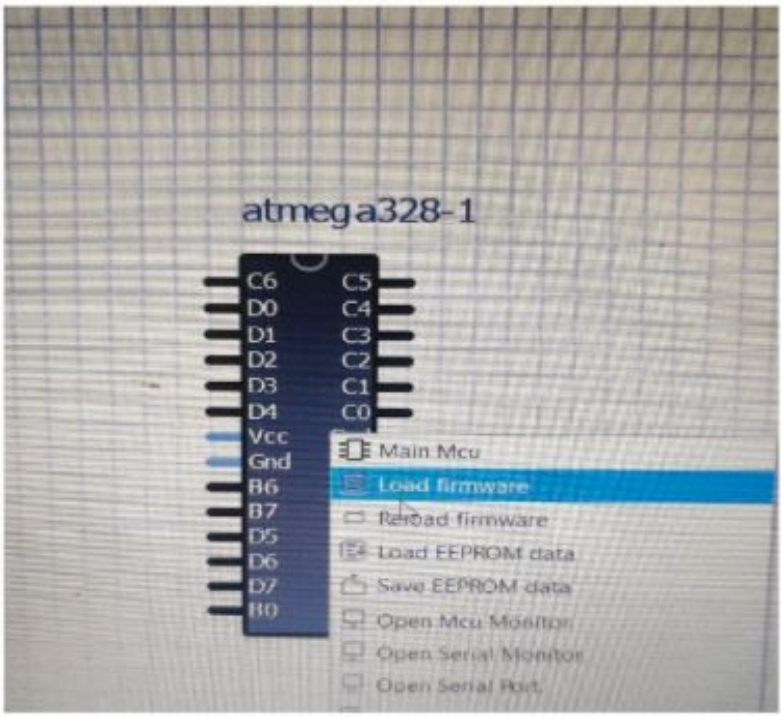
TCNT--→ Its is timer counter register of 16 bit which counts from 0000 to ffff

Write program in the Arduino IDE , Verify and upload



Now open the SimulIDE → drag and drop the atmega 328 microcontroller

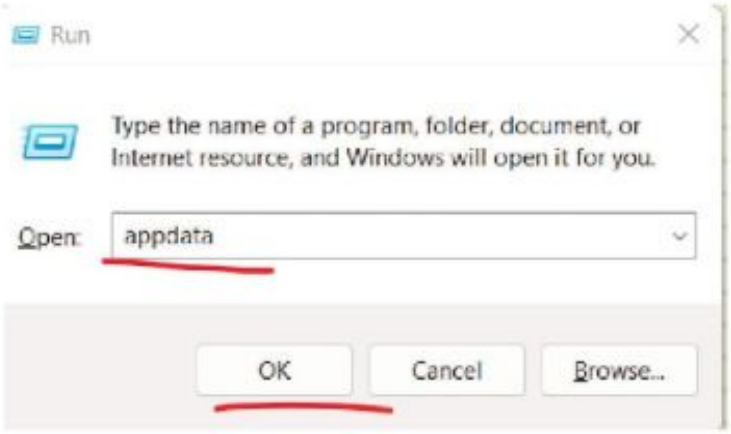




A window opens for Path

Jus hold window button pressed and click on r button from keyboard

Run Prompt opens →write appdata→OK



Follow this path Local→Temp→

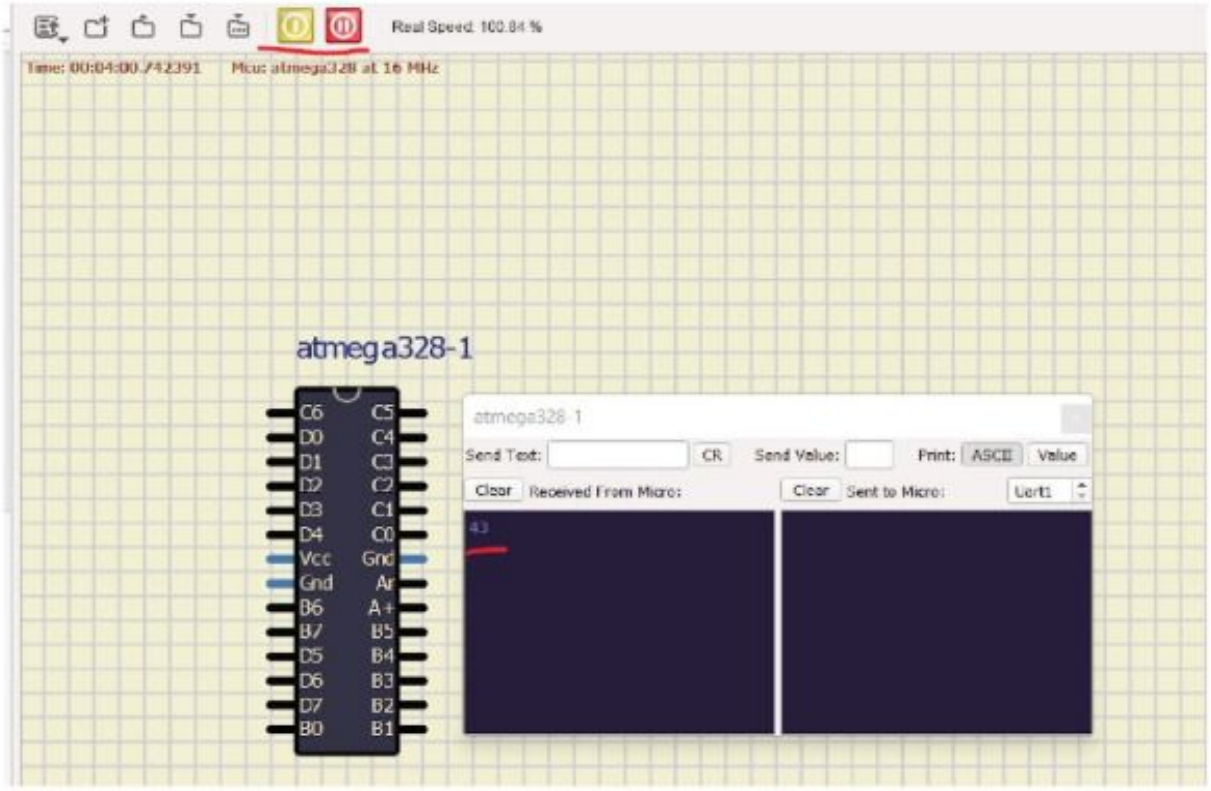
Select build file

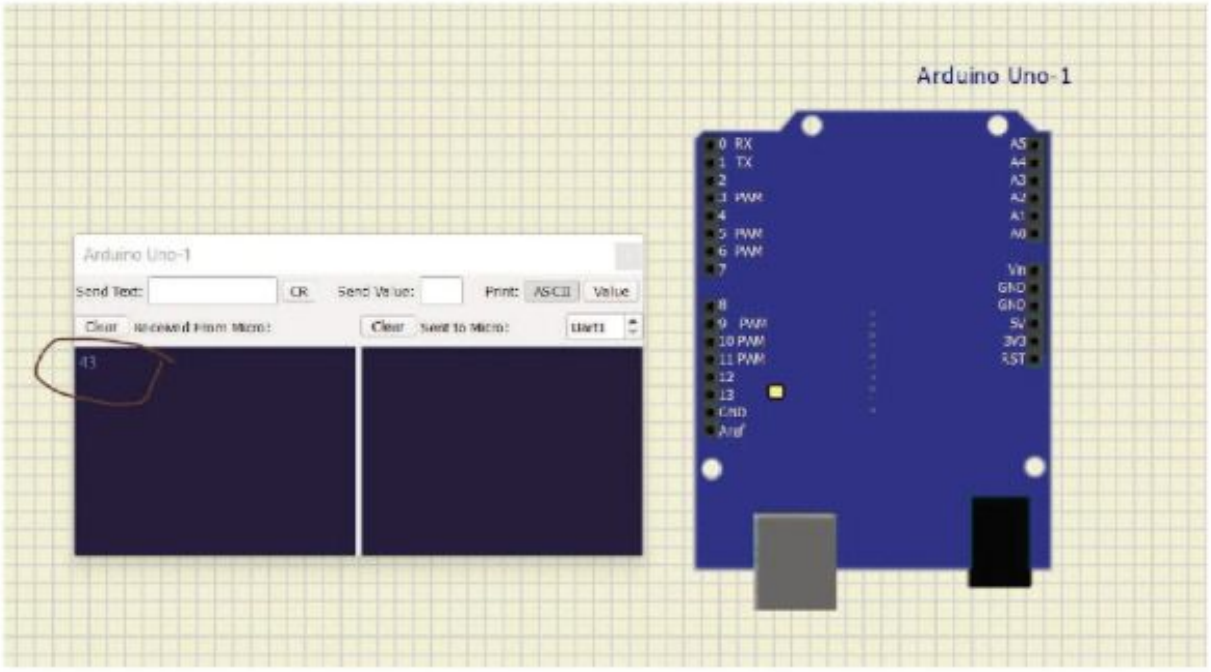
->Arduino_build_xxxxx(in this choose ur program elf file->right click->select as path

Go to SimullDE →paste that path→OK

Now right click on Microcontroller →Open Serial Monitor

Now press power button



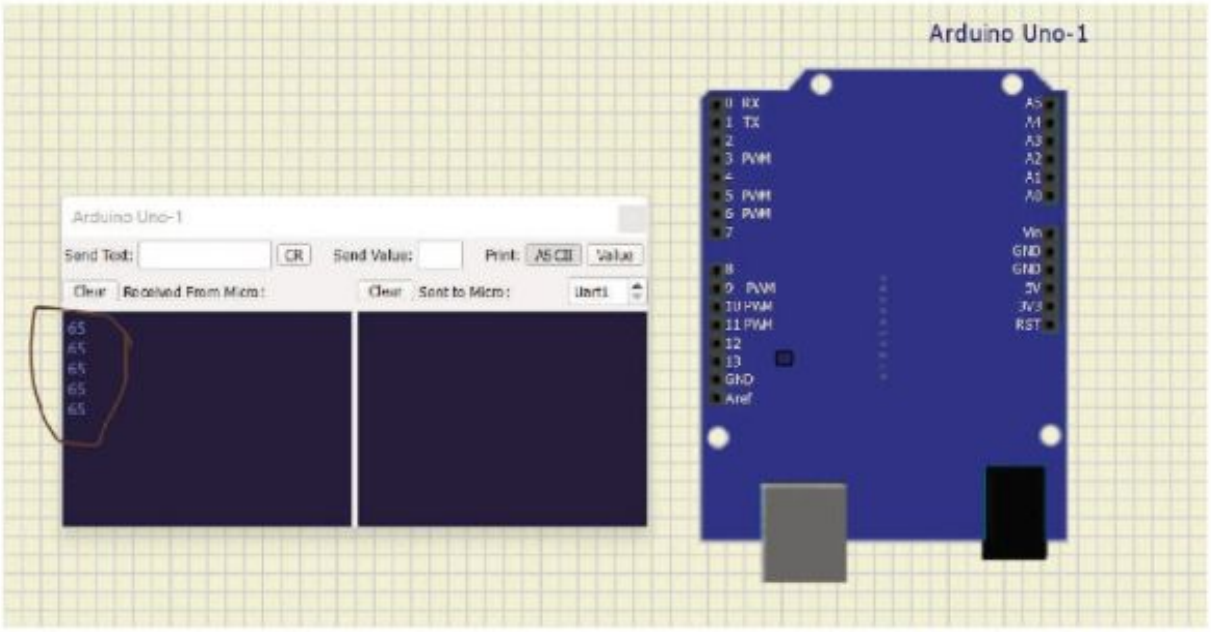


43 clock cycles taken to execute the all instruction in the code

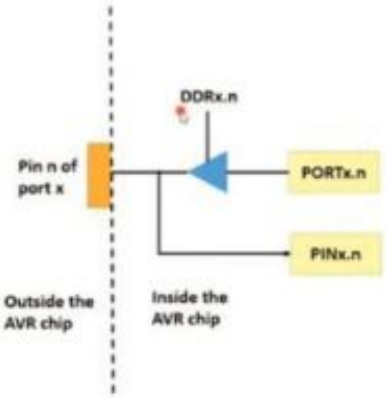
Measuring Clock Cycles taken by the instruction using digitalWrite() function

Same procedure follow for below program

```
void setup()
{
 // put your setup code here, to run once:
 Serial.begin(9600);
 TCCR1B = bit(CS10);
pinMode(PD6,OUTPUT);
}
void loop()
{
 // put your main code here, to run repeatedly:
   TCNT1 = 0;
digitalWrite(PD6,HIGH);
unsigned int cycles = TCNT1;
Serial.println(cycles);
delay(500);
digitalWrite(PD6,LOW);
delay(500);
}
```



Internal circuit diagram for writing a data to pin



DDRx , Data direction Register register will help to configure your port pins as input or output

PORTx register will write the data to the port pin which you have configured

PINx register will read the data from port pins.

If you write 0 to DDRX register , PORT acts as input PORT→ the condition of pin(physical pin) will be input

If DDRX=1 , PORT acts as output →the condition of pin will be output

DDR Register:

The **data direction register** (DDR) is most likely the first register that you configure since the DDR register determines if pins on a specific port are inputs or outputs. The DDR register is 8 bits long and each bit corresponds to a pin on that I/O port

GPIO in microcontroller:

A GPIO (general-purpose input/output) port handles both incoming and outgoing digital signals. As an input port, it can be used to communicate to the CPU the ON/OFF signals received from switches, or the digital readings received from sensor

Register	Description
DDRx	Used to configure the respective PORT as output/input
PORTx	Used to write the data to the Port pins
PINx	Used to Read the data from the port pins

Registers for GPIO

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIND – The Port D Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

If You want to access these registers we have to use the address of these registers given in data sheet

To access I/O devices, two implementations are there

- Memory Mapped I/O - I/O addresses are mapped into the same address space as program memory and same set of instructions are used to access the addresses mapped to I/O devices as well as program memory.
- PORT Mapped I/O - I/O addresses are mapped into a separate address space and different set of instructions are used to access that I/O addresses.
- AVR memory architecture uses Memory mapped I/O.

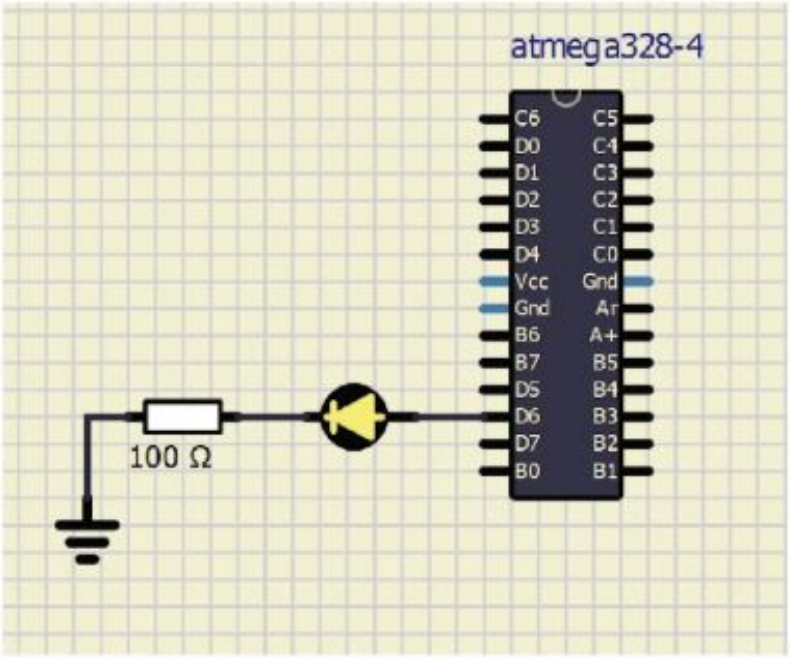
LED Blinking program using Bare Metal Programming (using registers)

```
unsigned char *portd = (unsigned char *)0x2B;
void setup()
{
 unsigned char *ddrd = (unsigned char *)0x2A;
 *ddrd |= 0b01000000;
}
void loop()
{
 *portd |= (0b01000000);  // 0xff
delay(500);
 *portd &= ~(0b01000000); // 0x00
delay(500);
}
```

Please follow the same procedure to load elf file

```
C:\Program Files (x86)\Arduino\hardw
\blink_bare.ino.elf
   text    data     bss     dec
    640       0       9     649
```

Now we can compare that using Arduino library LED blink program takes 924Kbytes and using Bare Metal programming the blink program takes 640 Kbytes of memory



Now let's examine how much memory and how many clocks this code is taking:

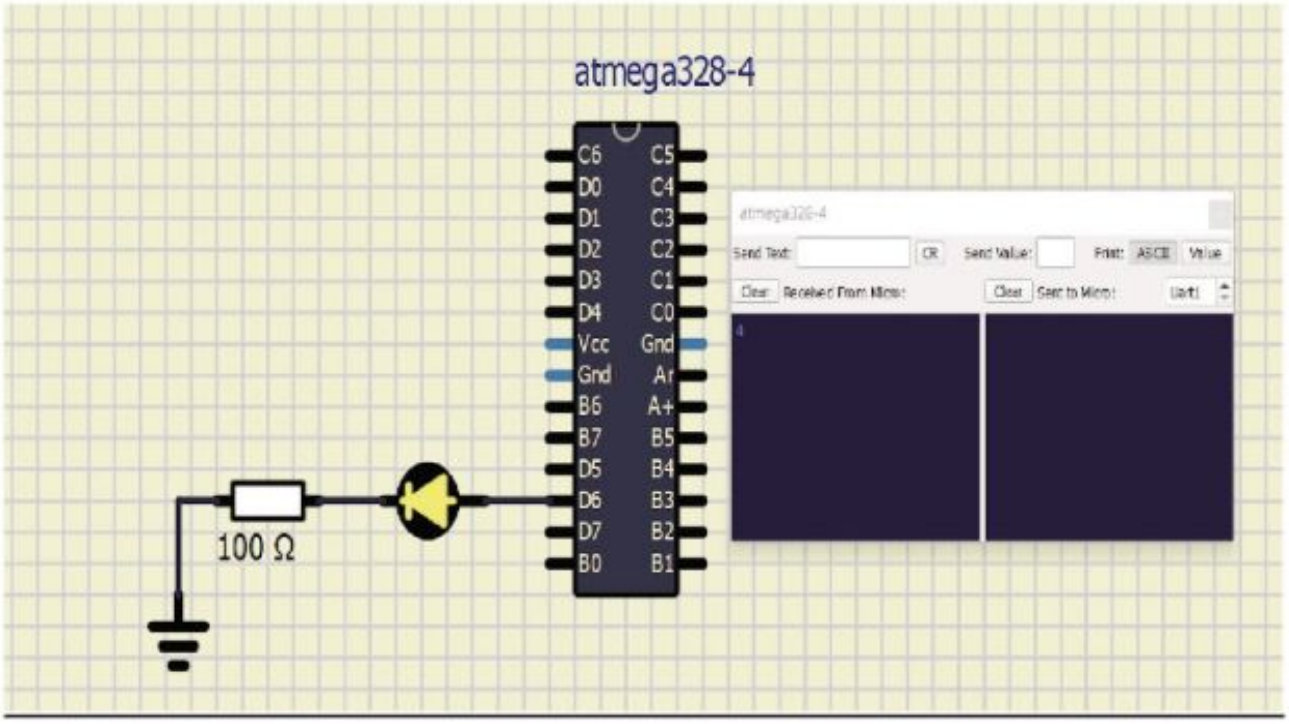
Using Bare Metal:

```
unsigned char *portd = (unsigned char *)0x2B;

void setup()
{

TCCR1B = bit (CS10);
Serial.begin(9600);

unsigned char *ddrd = (unsigned char *)0x2A;

TCNT1 = 0;
```

```
*ddrd |= 0b01000000; //setting the 6th bit
unsigned int cycles = TCNT1;
Serial.println(cycles);
}
void loop()
{
 *portd = (0b01000000); //setting the 6th bit
 delay(1000);
 *portd &= ~(0b01000000);//clearing the 6th bit
 delay(1000);
}
```

blink_bare

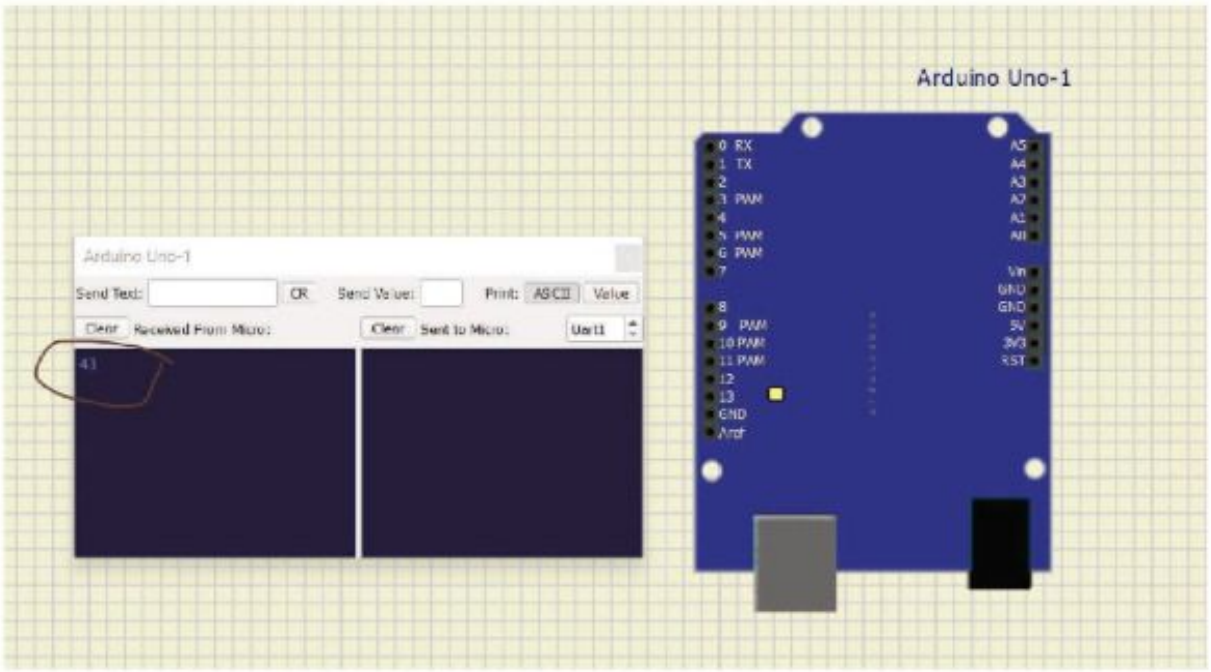
```
unsigned char *portd = (unsigned char *)0x2B;
void setup() {
  TCCR1B = bit(CS10);
  Serial.begin(9600);
  unsigned char *ddrd = (unsigned char *)0x2A;
  TCNT1 = 0;
  *ddrd |= 0b01000000;//Setting the 6th bit
  unsigned int cycles = TCNT1;
  Serial.println(cycles);
}

void loop() {
  *portd |= (0b01000000);//setting the 6th bit
  delay(1000);
  *portd &= ~(0b01000000);//clearing the 6th bit
  delay(1000);
}
```



So we can conclude that using Bare metal programming we can save the Memory as well as Clock cycle

Using Arduino Libraries

```
void setup() {
  TCCR1B = bit(CS10);//configuring the timer
  TCNT1 = 0;//set the counter to 0
  pinMode(LED_BUILTIN,OUTPUT);
  //record the timer counter
  unsigned int cycles = TCNT1;
  Serial.begin(9600);
  //print the result
  Serial.println(cycles);
}

void loop() {
  digitalWrite(LED_BUILTIN,HIGH);
  delay(500);
  digitalWrite(LED_BUILTIN,LOW);
  delay(500);
}
```



So we can Conclude that Using Bare metal programming, memory can be saved and clock cycle taken to execute the program also less

Using Arduino Library	**Using Bare metal Programming**
Memory = 924 K Bytes	**Memory = 640 K Bytes**
Clock Cycles= 43 cycles	**Clock Cycles = 4**

The above data I considered for LED Blinking program using Arduino library and Bare metal programming