Roll Number : ____________ Seat Number : ____________

# Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University Of Mumbai

Hashu Advani memorial Complex Collector's Colony, R C Marg, Chembur Mumbai 400074, phone Number 022-61532532

## CERTIFICATE

DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

Certified that Mr./Miss.________________________________________

Department **of Electronics and Computer Science SE - Semester IV** has successfully completed necessary experiments in **(NECMML42) Embedded Systems and RTOS lab** course under my supervision in VES Institute of Technology during the academic year 2024-2025.

**Dr. Asawari Dudhwadkar**
Subject Teacher

**Head of the Department**
(Electronics and Computer Science)

VESIT V.E.S. Since 1962 VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

**NECMML42: Embedded Systems and RTOS Lab**

**Department of Electronics and Computer Science**

**Subject:** Embedded Systems and RTOS **Division:** D6 **Semester :** IV
**Faculty In charge:** Dr. Asawari Dudhwadkar **Academic Year**: 2024-2025

**Course Objectives:**

1. To study concepts involved in Embedded Hardware and Software for System realisation.
2. To learn the concepts of modern microcontroller cores like the ARM-Cortex.
3. To learn Real-time programming to design time-constrained embedded systems.

**Course Outcomes:**

1. Identify and describe various characteristic features and applications of Embedded systems.
2. Analyse and select hardware for Embedded system implementation.
3. Evaluate various communication protocols for Embedded system implementation.
4. Compare GPOS and RTOS and investigate the concepts of RTOS.
5. Evaluate and use various tools for testing and debugging embedded systems.
6. Design a system for different requirements based on the life cycle for the embedded system
7. Keeping oneself aware of ethics and environmental issues.

VESIT V.E.S. Since 1962

VIVEKANAND EDUCATION SOCIETY

INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

**NECMML42: Embedded Systems and RTOS Lab**

# Index

| Sr. No | Name of Experiment | DOP | DOS | CO Mappings | Sign |
|---|---|---|---|---|---|
| 1 | To print data using UART function for an embedded system using MS-51 Board ( Nuvoton 51)(using Arduino IDE). | | | CO1,CO2, CO3 | |
| 2 | To run onboard buzzer for alarming purposes using MS-51 Board ( Nuvoton 51). | | | CO1,CO2, CO3 | |
| 3 | To implement a on board LED blinking using Nuvoton 51 board. | | | CO1,CO2, CO3 | |
| 4 | To interface an ADC with an embedded system using MS-51 Board ( Nuvoton 51)(using Arduino IDE). | | | CO1,CO2, CO3 | |
| 5 | To blink LED on STM32 NUCLEO board. | | | CO1,CO2, CO3 | |
| 6 | To change the speed of the blinking of LEDs on STM32 with and without FreeRTOS. | | | CO1,CO2, CO3,CO4 | |
| 7 | To implement binary semaphore for task synchronization using FreeRTOS. | | | CO1,CO2, CO3,CO4 | |
| 8 | To create tasks and scheduling them as per priority using STM32 and FreeRTOS. | | | CO1,CO2, CO3,CO4 | |
| 9 | Assignment - 1 and CA 1 and 2 | | | CO1,CO2, CO3,CO4 | |
| 10 | Additional Lab - Interface Ultrasonic, IR, Temp & Humidity sensor using Arduino Board | | | CO2, CO7 | |

**Dr. Asawari Dudwadkar**

Subject Teacher

**Head of the Department**

(Electronics and Computer Science)

# Experiment 1

**Aim:** To print Data using UART function for an Embedded System

**Apparatus:** MS-51PC0AE Development Board, KeiluVision5, Nuvoton ISP programming tool, Arduino IDE,

**Bin File Location:**
*“8051_D11_EXPERIMENTS\SampleCode\RegBased\UART0_Printf\Keil\Output\UART0_Printf.bin”*

## Specifications of MS51PC0AE:

The MS-51PC0AE is an 8-bit microcontroller. Some key features:

- **CPU:** 8-bit core, compatible with the 8051 instruction set.
- **Clock Speed:** Operates at up to 25 MHz.
- **Memory:**
  - 4 KB Flash ROM for program storage.
  - 128 Bytes SRAM.
- **UART:** Includes an on-chip UART peripheral for serial communication.
  **I/O Ports:** 32 general-purpose I/O pins.
- **Timers:** Two 16-bit timers.
- **Interrupts:** Supports external and internal interrupts.
- **Power Supply:** Operates at 2.7V to 5.5V.

## Procedure:

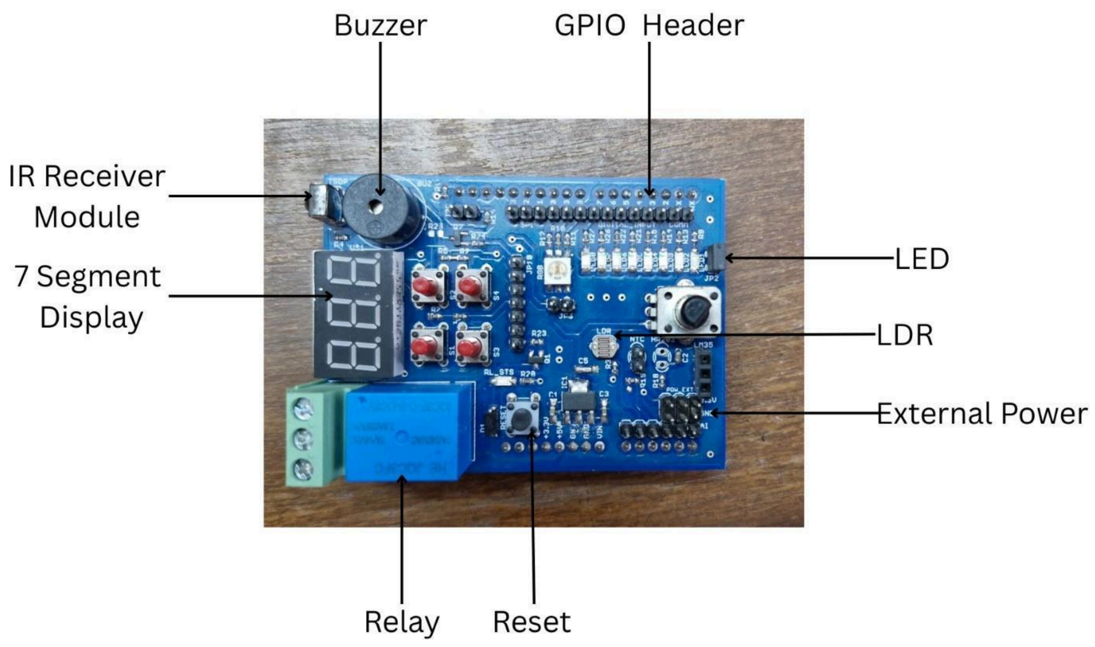
1. Connect the MS51PC0AE development board to the power supply and establish a UART connection to your PC.
2. Open Keil uVision IDE, create a new project, and select the 8051 microcontroller (MS-51PC0AE).
3. Write the C code to initialize the UART and transmit data.
4. Compile the code and check for errors.
5. Load the compiled code onto the MS51PC0AE using a suitable programmer.
   a. Open NuvoISP
   b. In Connection Interface, choose “UART” and the COM port accordingly.

c. Press "Reset Button" on the board
d. Load the bin file by clicking on "APROM"
e. The relative path will be similar to: "*8051_D11_EXPERIMENTS\SampleCode\RegBased\UART0_Printf\Keil\Output\UART0_Printf.bin*"
f. Select APROM under the Programming section at the Bottom
g. Click "Start"

6. Open a serial terminal on the PC (Arduino IDE), set the baud rate, and connect to the correct COM port.

7. Run the program, and observe the transmitted data on the terminal.

**Figure MS51PC0AE Development Board:**



It is the **MS51PC0AE Development Board**, which is based on Nuvoton's **MS51 series microcontroller** — a modern 8051-compatible MCU with several integrated peripherals.

**Key Features of the MS51PC0AE Development Board:**

1. 8-bit 8051-compatible core

2. Flash: 4 KB
3. SRAM: 128 Bytes
4. Clock Speed: Up to 24 MHz (HIRC)
5. Voltage Range: 2.7V to 5.5V
6. Peripherals: UART, SPI, I2C, Timers, ADC, PWM

**Code:**

```
#include "ms51_32k.h"
void main (void)
{
    MODIFY_HIRC(HIRC_24);
    Enable_UART0_VCOM_printf_24M_115200();
    while(1)
    {
      printf("\n Hello !");
      Timer0_Delay(24000000,300,1000);
    }
}
```

**Output:**

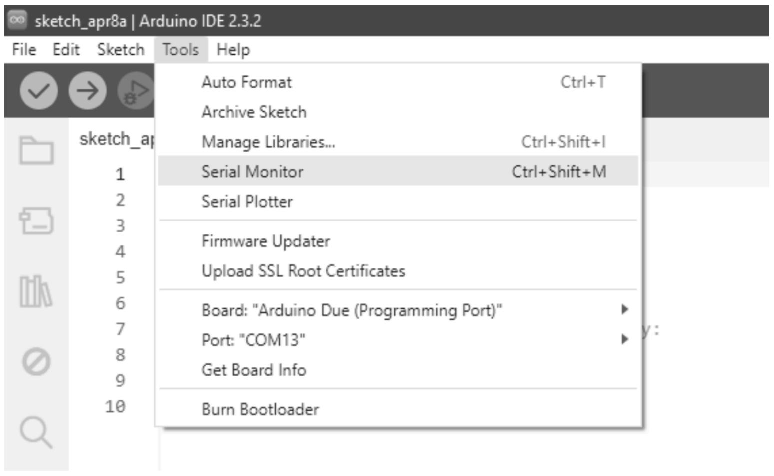
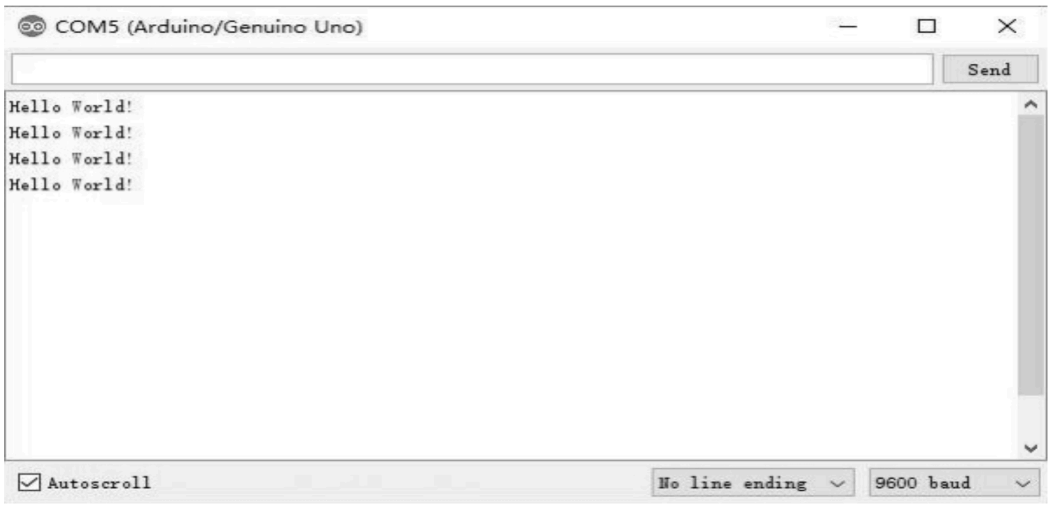

sketch_apr8a | Arduino IDE 2.3.2
File Edit Sketch Tools Help
sketch_ap
Auto Format Ctrl+T
Archive Sketch
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter
Firmware Updater
Upload SSL Root Certificates
Board: "Arduino Due (Programming Port)"
Port: "COM13"
Get Board Info
Burn Bootloader

**Fig 1: Select Serial monitor to check output**


COM5 (Arduino/Genuino Uno)

```
Hello World!
Hello World!
Hello World!
Hello World!
```

**Fig 2: Output of the experiment**

## Takeaways:

- **Understanding of UART:** This experiment deepened the understanding of asynchronous serial communication and how UART works within embedded systems.
- **Microcontroller Programming:** Gained hands-on experience in setting up UART communication in an 8051 microcontroller using C and embedded libraries.
- **Tool Familiarity:** Learned how to use industry-standard tools such as Keil uVision, NuvoISP, and Arduino IDE for embedded development and debugging.
- **Clock Configuration Importance:** Realized the significance of configuring the internal oscillator (HIRC) to a precise frequency (24 MHz) to maintain UART baud rate accuracy.
- **Practical Debugging:** Understood the importance of checking COM port settings, baud rate matching, and using reset and flashing procedures correctly during programming.

## Conclusion:

The experiment successfully demonstrated UART communication using the MS51PCOAE microcontroller, confirming correct configuration and functionality by transmitting "Hello !" in a loop with visible delay.

VESIT VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)
V.E.S. Since 1962

**NECMML42: Embedded Systems and RTOS Lab**

# Experiment No. 2

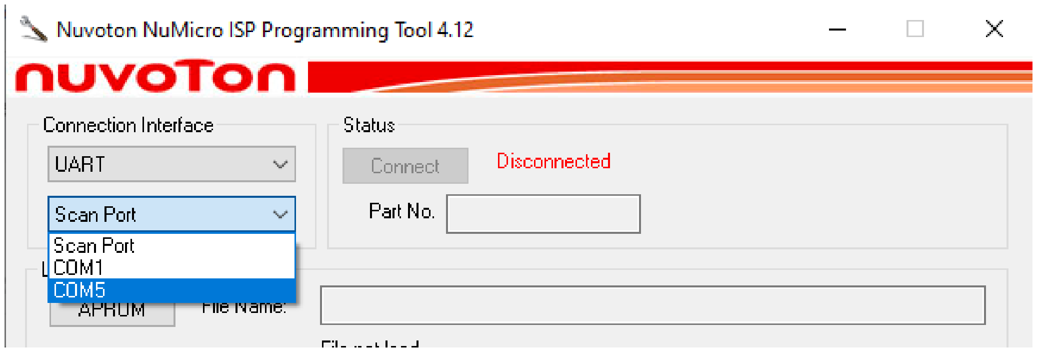
**Aim:** To run onboard Buzzer for alarming purpose using Nuvoton.

**Apparatus:** MS51PC0AE Development Board, Keil uVision5, Nuvoton ISP programming tool, I/O board with BuzzerAgents.

**Bin File Location: "***D:\8051_total_4exp_D11\SampleCode\RegBased\GPIO_Input_Output - buzzer \Keil\Output\GPIO_Input_Output.bin*"

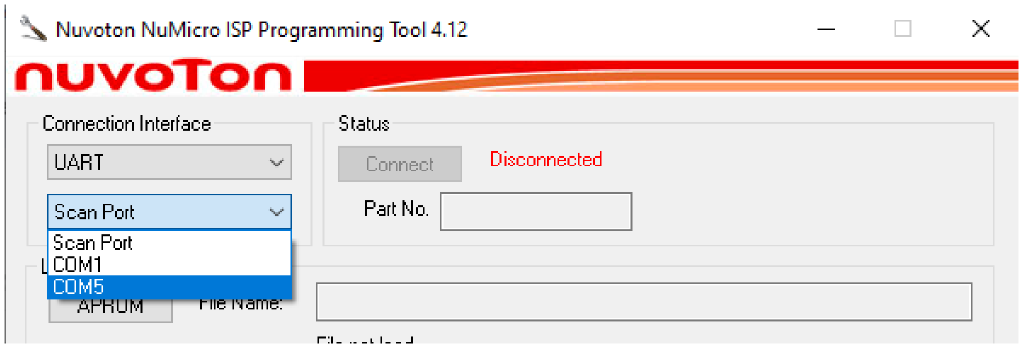
**Procedure:**

1. **Prepare and Connect**
   - Connect the Nuvoton 8051 board to your PC via the appropriate interface.
   - Launch the Nuvoton ISP software.
   - In the connection settings, select UART.



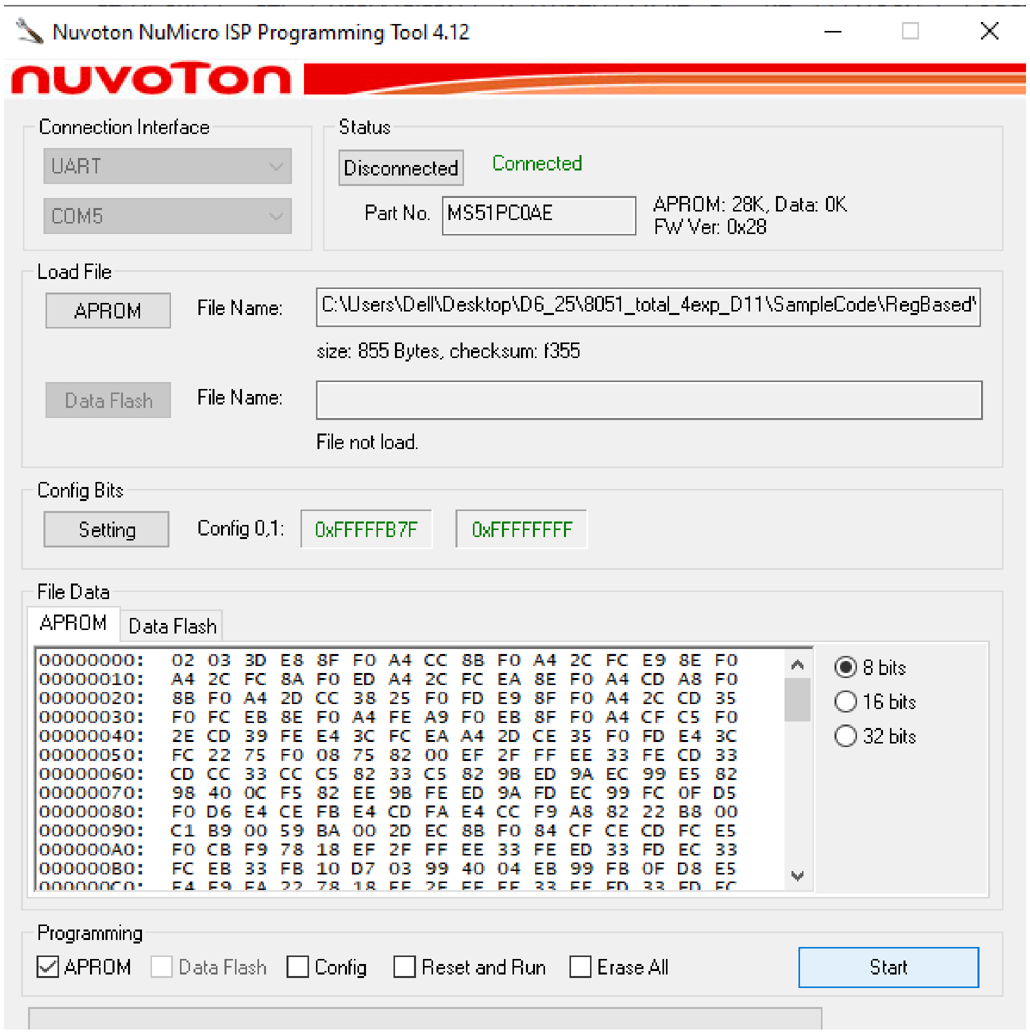
Connection Interface: UART
Scan Port
Disconnected
Part No.

   - Choose the correct COM port (e.g., COM3) from the dropdown list.



Connection Interface: UART
Scan Port
Disconnected
Part No.

- Click Connect and immediately press the board's Reset button. Once connected, the software will display a "Connected" status.


Nuvoton NuMicro ISP Programming Tool 4.12

Connection Interface: UART, COM5

Status: Disconnected | Connected

Part No. MS51PC0AE    APROM: 28K, Data: 0K  FW Ver: 0x28

Load File

APROM  File Name: C:\Users\Dell\Desktop\D6_25\8051_total_4exp_D11\SampleCode\RegBased\

size: 855 Bytes, checksum: f355

Data Flash  File Name:

File not load.

Config Bits

Setting  Config 0,1: 0xFFFFFB7F  0xFFFFFFFF

File Data

APROM | Data Flash

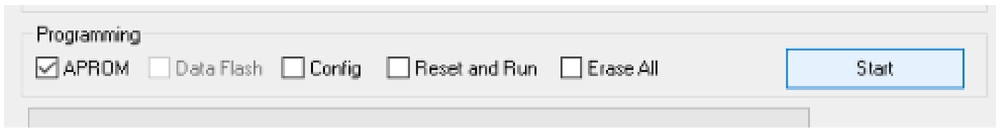
```
00000000:  02 03 3D E8 8F F0 A4 CC 8B F0 A4 2C FC E9 8E F0
00000010:  A4 2C FC 8A F0 ED A4 2C FC EA 8E F0 A4 CD A8 F0
00000020:  8B F0 A4 2D CC 38 25 F0 FD E9 8F F0 A4 2C CD 35
00000030:  F0 FC EB 8E F0 A4 FE A9 F0 EB 8F F0 A4 CF C5 F0
00000040:  2E CD 39 FE E4 3C FC EA A4 2D CE 35 F0 FD E4 3C
00000050:  FC 22 75 F0 08 75 82 00 EF 2F FF EE 33 FE CD 33
00000060:  CD CC 33 CC C5 82 33 C5 82 9B ED 9A EC 99 E5 82
00000070:  98 40 0C F5 82 EE 9B FE ED 9A FD EC 99 FC 0F D5
00000080:  F0 D6 E4 CE FB E4 CD FA E4 CC F9 A8 82 22 B8 00
00000090:  C1 B9 00 59 BA 00 2D EC 8B F0 84 CF CE CD FC E5
000000A0:  F0 CB F9 78 18 EF 2F FF EE 33 FE ED 33 FD EC 33
000000B0:  FC EB 33 FB 10 D7 03 99 40 04 EB 99 FB 0F D8 E5
000000C0:  F4 F9 FA 22 78 18 EE 2F FF EE 33 FE FD 33 FD FC
```

(● 8 bits) ○ 16 bits ○ 32 bits

Programming

[✓] APROM  [ ] Data Flash  [ ] Config  [ ] Reset and Run  [ ] Erase All  Start

2. **Load the Binary File**
   - Click on the file selection button and load the binary file located at: D:\8051_total_4exp_D11\SampleCode\RegBased\GPIO_Input_Output\Keil\GPIO_Input_Output.bin
   - In the programming section, check the APROM option to ensure the main application area is targeted.


Programming

[✓] APROM  [ ] Data Flash  [ ] Config  [ ] Reset and Run  [ ] Erase All  Start

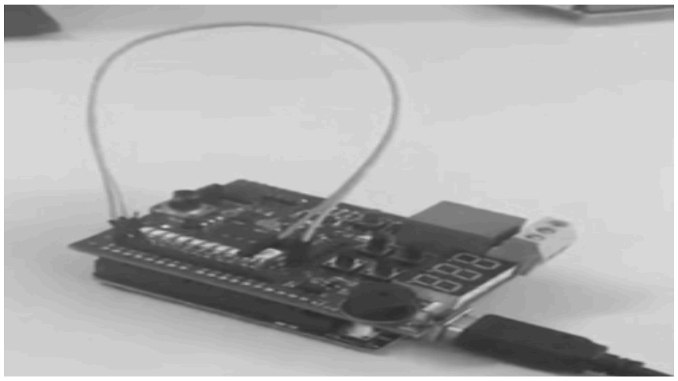
3. **Program the Board**
   - Click the Start/Program button to begin the upload process.
   - If prompted, press the reset button on the board again to initiate programming.
   - Once programming starts and completes, the board will automatically run the new code.
4. **Verify the Output**
   - After the programming process, the buzzer on the board will start ringing, confirming that the code has been successfully uploaded and is running.

Buzzer is on for the duration of timer delay given to 0x00 of the port the buzzer is connected to. We can configure this delay as per our requirement in milliseconds and also keep it on continuously in absence of delay.

**Code:-**

```
#include         "ms51_32k.h"
unsigned char temp _at_ 0x08;
unsigned char idata itemp _at_ 0x90;
unsigned char xdata xtemp _at_ 0x80;
void main (void)
{
  ALL_GPIO_PUSHPULL_MODE;
  while(1)
  {
    P3 = 0x00;
    Timer0_Delay(16000000,200,1000);;
    P3=0xff;
    Timer0_Delay(16000000,200,1000);;
  }
}
```

**Output:-**



## Takeaways:

1. **Buzzer can serve as an effective audio alert device** in embedded systems when controlled through microcontroller output pins.

2. **Timer delays are crucial** for managing how long the buzzer stays ON or OFF, enabling various types of alert patterns.

3. **Simple C code using delay functions** can be used to automate buzzer behavior, making it easy to integrate into larger systems.

4. **Onboard peripherals like UART and serial monitor** help in real-time debugging and status monitoring during development.

5. **Microcontroller-based control offers flexibility**, allowing the buzzer to be triggered conditionally based on different sensor inputs or events.

## Conclusion:

The experiment demonstrated using an MS-51 Board microcontroller to operate an onboard buzzer for alarm-based applications, simulating an alert system with programmed timer delays.

**NECMML42: Embedded Systems and RTOS Lab**

# Experiment 3

**Aim:** To implement on board LED blinking using Nuvoton 51 board.

**Apparatus:** MS51 Board, KeiluVision5, NuvotonISP programming tool, Arduino IDE.

**Bin File Location:** "*D:\8051_total_4exp_D11\SampleCode\RegBased\GPIO_Input_Output \Keil\GPIO_Input_Output.bin*"

**Specifications of 8051 board:**

The MS51PC0AE is an 8-bit microcontroller based on the popular 8051 architecture. Here are some key features:

- **Architecture:** 8-bit Harvard architecture
- **Core:** 8051-compatible
- **Clock Speed:** Typically 11.0592 MHz to 33 MHz (some variants go higher)
- **Instruction Set:** CISC (Complex Instruction Set Computing)
- **Memory:**
  - **Flash ROM / Program Memory:** 4 KB – 64 KB (varies by model) *(e.g., AT89C51 = 4 KB, P89V51RD2 = 64 KB)*
  - **RAM (Internal Data Memory):** 128 Bytes – 256 Bytes
  - **EEPROM:** Some variants have onboard EEPROM

## Power Supply

- **Operating Voltage:** Typically 5V DC
- **Power Consumption:** Low (varies by clock and mode)

**Timers:**

- **Timers:** 2 x 16-bit Timers/Counters (Timer 0 and Timer 1)
- **Modes:** 4 Timer modes (Mode 0 – Mode 3)

**Communication Interfaces**

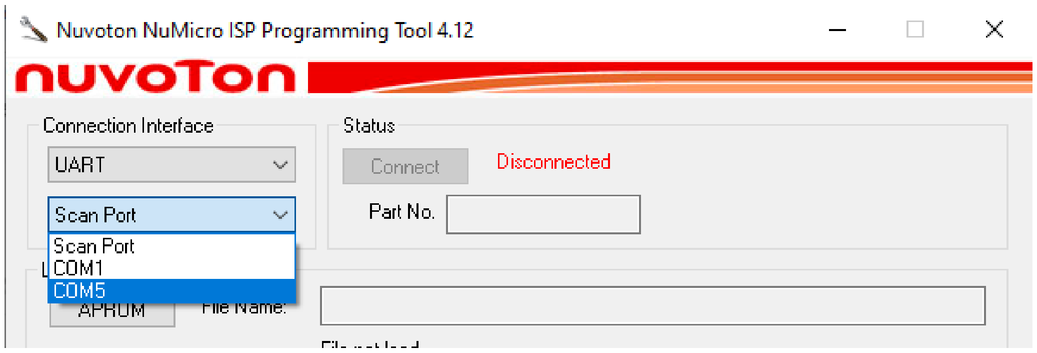
- **UART / Serial Port:** 1 x Full-Duplex Serial Port (RS232 through MAX232)
- **I2C/SPI:** Not built-in, but can be implemented in software

**GPIO Pins: 32 I/O pins, divided across 4 ports:**

- **Port 0:** Dual-purpose (Address/Data)
- **Port 1:** Standard I/O
- **Port 2:** I/O or high address byte
- **Port 3:** I/O and alternate functions (UART, interrupts, timers)

**Procedure:**

1. Open the Nuvoton ISP software.
2. Select UART as connection interface. Then from the dropdown of scan port, select the port on which 8051 board is connected.


Nuvoton NuMicro ISP Programming Tool 4.12

Connection Interface: UART

Scan Port: Scan Port, COM1, COM5

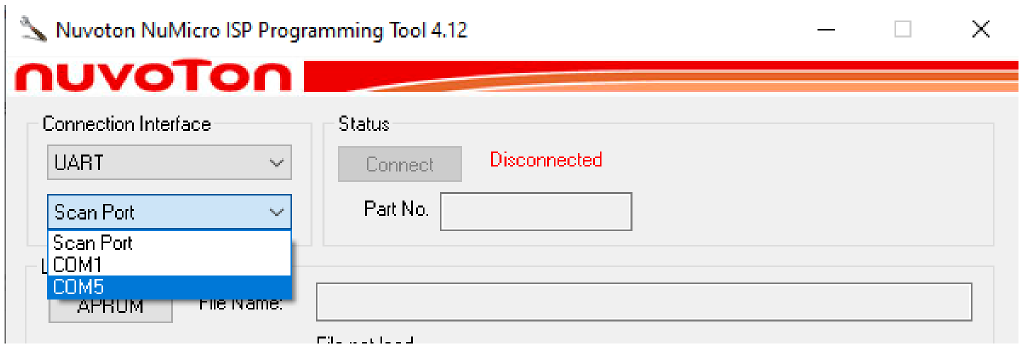
Status: Disconnected

Connect

Part No.

File Name:

3. Click on the “connect” button then click on the reset button of the nuvoton board. It will show connected status.


Nuvoton NuMicro ISP Programming Tool 4.12

Connection Interface: UART

Scan Port: Scan Port, COM1, COM5

Status: Disconnected

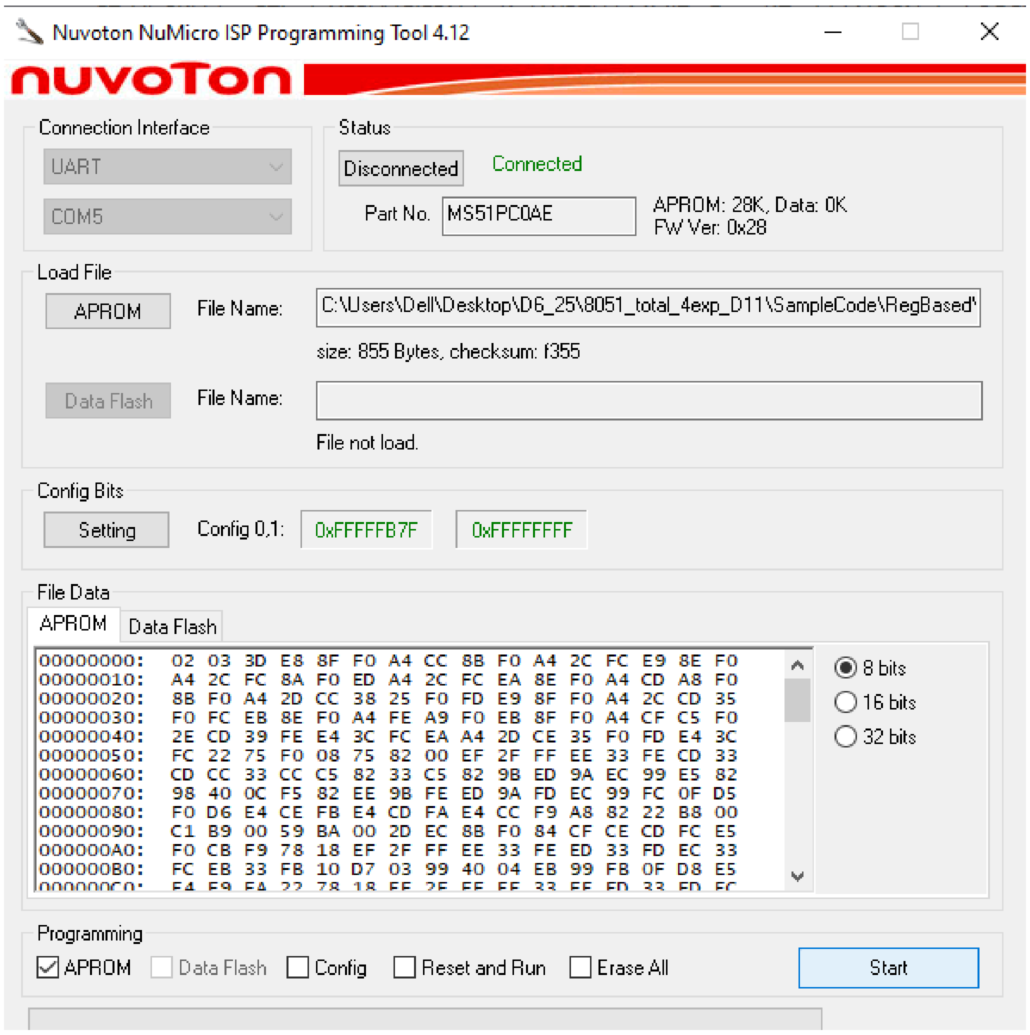
Connect

Part No.

File Name:

4. Select the bin file of the experiment saved at :
   *D:\8051_total_4exp_D11\SampleCode\RegBased\GPIO_Input_Output \Keil\GPIO_Input_Output.bin*

5. Select the APROM checkfield in the programming section.



Nuvoton NuMicro ISP Programming Tool 4.12

Connection Interface: UART, COM5

Status: Disconnected, Connected

Part No. MS51PC0AE — APROM: 28K, Data: 0K, FW Ver: 0x28

Load File

APROM — File Name: C:\Users\Dell\Desktop\D6_25\8051_total_4exp_D11\SampleCode\RegBased\

size: 855 Bytes, checksum: f355

Data Flash — File Name:

File not load.

Config Bits

Setting — Config 0,1: 0xFFFFFB7F 0xFFFFFFFF

File Data: APROM | Data Flash

```
00000000:  02 03 3D E8 8F F0 A4 CC 8B F0 A4 2C FC E9 8E F0
00000010:  A4 2C FC 8A F0 ED A4 2C FC EA 8E F0 A4 CD A8 F0
00000020:  8B F0 A4 2D CC 38 25 F0 FD E9 8F F0 A4 2C CD 35
00000030:  F0 FC EB 8E F0 A4 FE A9 F0 EB 8F F0 A4 CF C5 F0
00000040:  2E CD 39 FE E4 3C FC EA A4 2D CE 35 F0 FD E4 3C
00000050:  FC 22 75 F0 08 75 82 00 EF 2F FF EE 33 FE CD 33
00000060:  CD CC 33 CC C5 82 33 C5 82 9B ED 9A EC 99 E5 82
00000070:  98 40 0C F5 82 EE 9B FE ED 9A FD EC 99 FC 0F D5
00000080:  F0 D6 E4 CE FB E4 CD FA E4 CC F9 A8 82 22 B8 00
00000090:  C1 B9 00 59 BA 00 2D EC 8B F0 84 CF CE CD FC E5
000000A0:  F0 CB F9 78 18 EF 2F FF EE 33 FE ED 33 FD EC 33
000000B0:  FC EB 33 FB 10 D7 03 99 40 04 EB 99 FB 0F D8 E5
```

[● 8 bits] 16 bits 32 bits

Programming

[✓ APROM] Data Flash, Config, Reset and Run, Erase All — Start

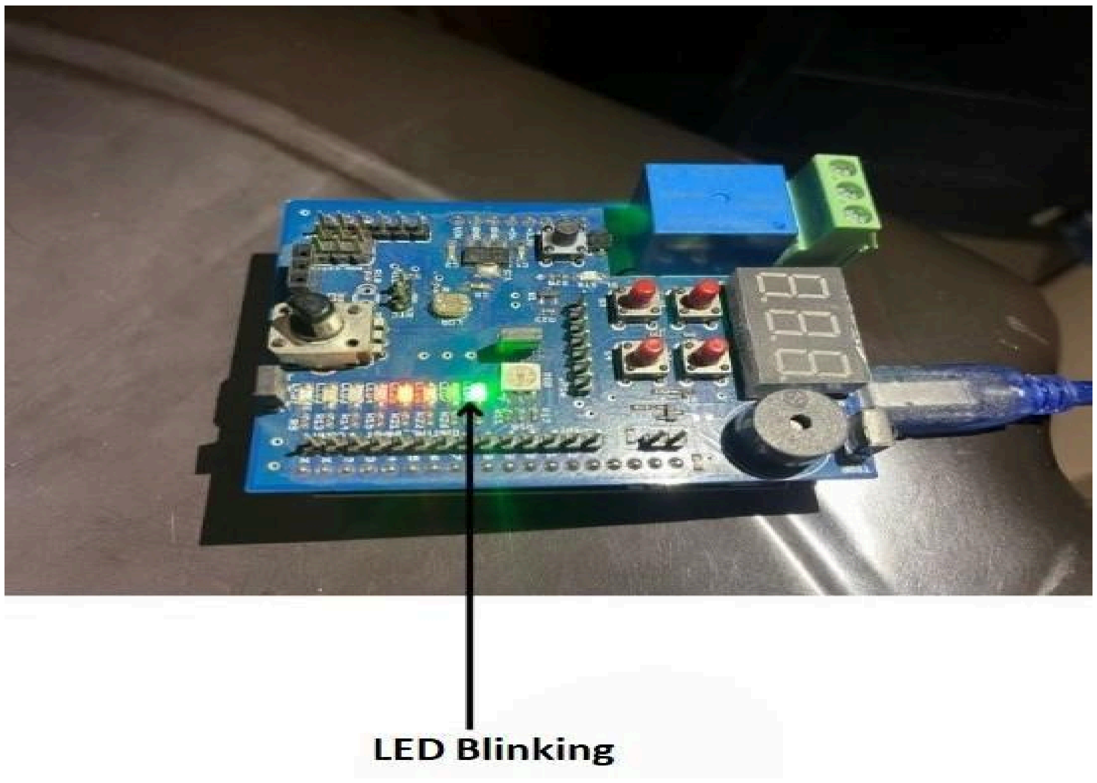
6. Click Start button

**Code:**

```
#include "ms51_32k.h"
unsigned int counter =
0;
void UART0_Init(void)
{
    P06_QUASI_MODE;
P07_QUASI_MODE;
    SCON = 0x50;
TMOD |= 0x20;
set_PCON_SMOD;
set_T3CON_BRCK;
    TH1 = 256 -
(24000000/16/9600);
set_TCON_TR1;
}
void
UART0_Send_Char(char c)
{
    SBUF = c;
    while (!TI);
    TI = 0;
}
void
UART0_Send_String(char*
str)
{
    while (*str)
    {

UART0_Send_Char(*str++)
;
    }
}
void
UART0_Send_Number(unsig
ned int num)
{
    char buffer[6];
    sprintf(buffer,
"%u", num);
UART0_Send_String(buffe
r);
}
void main(void)
{
```

```
MODIFY_HIRC(HIRC_24);
UART0_Init();
   while (1)
    {
UART0_Send_String("Coun
ter: ");
UART0_Send_Number(count
er);

UART0_Send_String("\r\n
");
        counter++;

Timer0_Delay1ms(1000);}
}
```

**Output:**



**Takeaways:**

**Understanding 16-bit Counter Implementation**

- Learned how to use `unsigned int` (16-bit) for counter operations in embedded C.
- Gained hands-on with incrementing logic and overflow behavior in counters.

**UART Communication Skills**

- Initialized and configured **UART0** for serial transmission at **9600 baud**.
- Used `SBUF`, `TI`, and appropriate flags for reliable character and string transmission.

**Real-Time Delay Using Timer0**

- Applied **Timer0_Delay1ms()** to create 1-second intervals between counter increments.
- Understood the use of software-based delays for timing control.

**Serial Monitoring and Debugging**

- Monitored real-time data via **serial terminal tools** like PuTTY, TeraTerm, or Arduino Serial Monitor.
- Practiced debugging UART communication through visible counter values.

**Embedded Programming with Keil & NuvotonISP**

- Compiled code in **Keil uVision** and flashed using **Nu-Link via NuvotonISP**.
- Understood the workflow of developing, compiling, and uploading firmware for 8051-based chips.

**Conclusion:**

The experiment demonstrated a 16-bit binary counter using the MS51 microcontroller, transmitting counter values over UART and verifying them on a serial monitor, showcasing real-time counting and serial communication capabilities.

## EXPERIMENT 4

**Aim:** To program onboard ADC and observe results using LDR and Potentiometer.

**Apparatus:** 8051 Board, KeiluVision4, Nuvoton ISP programming tool, Arduino IDE

**Theory:**

**ADC**: The MS51PC0AE is embedded with a 12-bit SAR ADC. The ADC (analog-to-digital converter) allows conversion of an analog input signal to a 12-bit binary representation of that signal. The MS51PC0AE is selected as an 8-channel input in single end mode. The internal band-gap voltage, 1.22 V can also be the internal ADC input. The analog input, multiplexed into one sample and hold circuit, charges a sample and holds the capacitor. The output of the sample and hold capacitor is the input to the converter. The converter then generates a digital result of this analog level via successive approximation and stores the result in the result registers. The ADC controller also supports continuous conversion and storage of result data into XRAM.

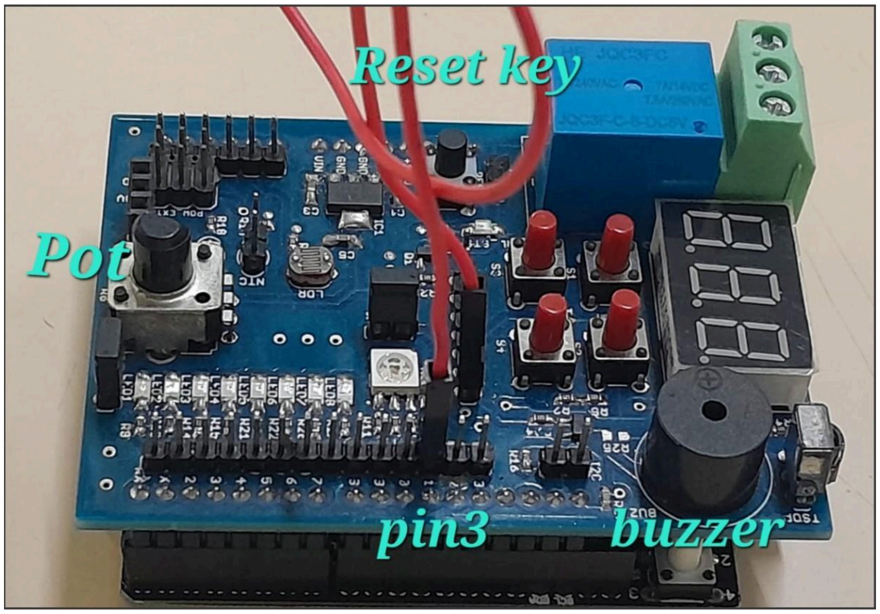
**LDR:** (Light Dependent Resistor) as the name states is a special type of resistor that works on the photoconductivity principle that means that resistance changes according to the intensity of light. Its resistance decreases with an increase in the intensity of light. It is often used as a light sensor, light meter, Automatic Street light, and in areas where we need to have light sensitivity. It is also called a Light Sensor.

**Potentiometer**: A potentiometer is a manually adjustable variable resistor with 3 terminals. Two of the terminals are connected to the opposite ends of a resistive element, and the third terminal connects to a sliding contact, called a wiper, moving over the resistive element. The potentiometer essentially functions as a variable resistance divider. The resistive element can be seen as two resistors in series (the total potentiometer resistance), where the wiper position determines the resistance ratio of the first resistor to the second resistor. If a reference voltage is applied across the end terminals, the position of the wiper determines the output voltage of the potentiometer.

**Procedure:**

1. Open the project file as per path:
2. D:\8051_total_4exp_D11\SampleCode\RegBased\ADC_Simple\Keil\ADC_Sim p le.bin.
3. Connect the board with CPU and Open Nuvoton ISP programming tool.
4. Select UART under the drop down menu and scan & select Com Port.
5. Click connect and press the reset button on 8051 board.
6. Click APROM File option then browse and select the new Bin file generated in your main project folder (D-drive).
7. Select the checkbox of APROM and Reset and Run.
8. Click Start to upload the code.
9. Open Arduino IDE, and select the same port selected in the nuvoton programming tool.
10. Open the serial monitor and set the baud rate to 115200.
11. Check for the data printed on the serial monitor.
12. Vary the POTENTIOMETER / light falling on LDR sensor and observe the change in Output

**Circuit Diagram:**



**CODE:**

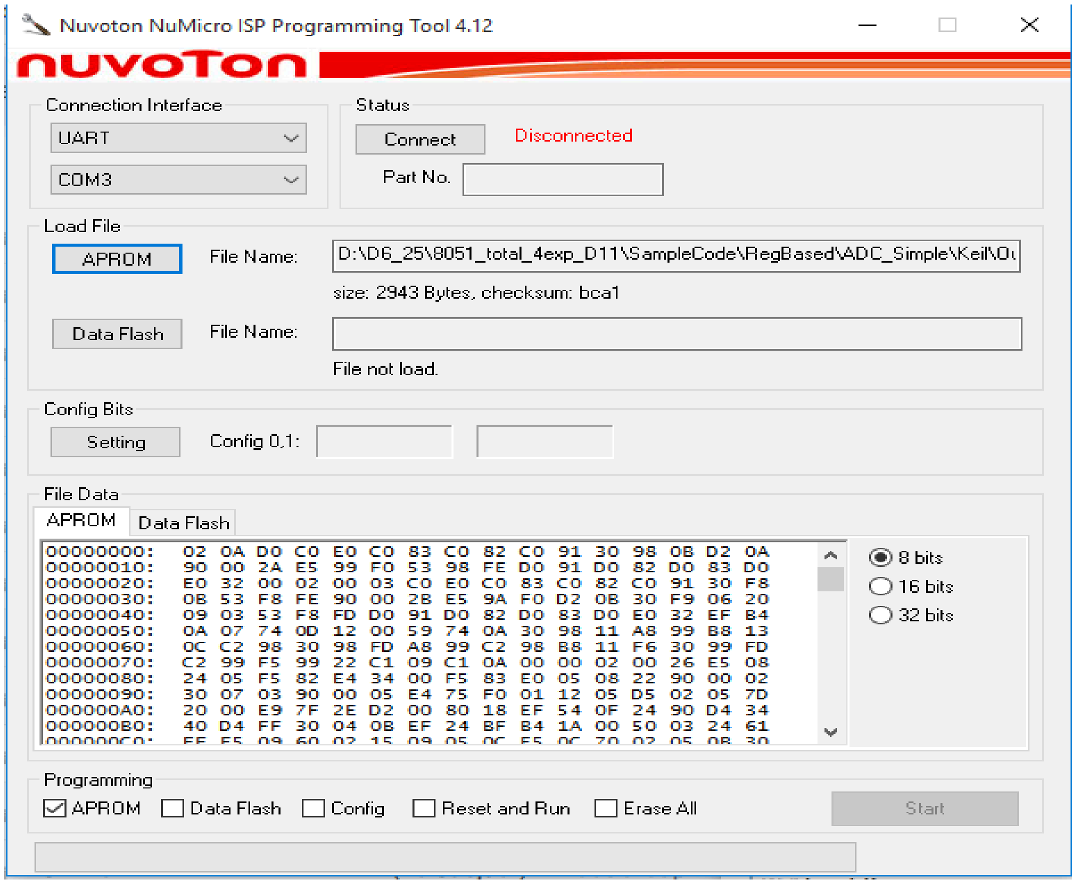
```
#include "ms51_32k.h"
 bit ADC_CONT_FINAL_FLAG;
void main (void)
{
    unsigned int temp;
    MODIFY_HIRC(HIRC_24);
    P06_PUSHPULL_MODE;
    UART_Open(24000000,UART0_Timer3,115200);
    ENABLE_UART0_PRINTF;
    ENABLE_ADC_CH9;
    Potentiometer while(1)
    {
      Timer2_Delay(24000000,128,300,1000);;
      clr_ADCCON0_ADCF;
      set_ADCCON0_ADCS;
      while(ADCF == 0);
      temp=(ADCRH<<4)+(ADCRL&0x0F);
      printf ("\n ADC Value = 0x%02X", temp);
      P35 ^= 1;
    }
}
```

**OUTPUT:**

Steps:

1. Select UART and COM3 in connection interface.
2. Load file – APROM (load the file path).
3. Select APROM in the programming section.
4. Select 8 bits and status – connect

# NECMML42: Embedded Systems and RTOS Lab


Nuvoton NuMicro ISP Programming Tool 4.12

Connection Interface: UART, COM3

Status: Connect — Disconnected

Part No.:

Load File

APROM — File Name: D:\D6_25\8051_total_4exp_D11\SampleCode\RegBased\ADC_Simple\Keil\Ou

size: 2943 Bytes, checksum: bca1

Data Flash — File Name:

File not load.

Config Bits

Setting — Config 0,1:

File Data

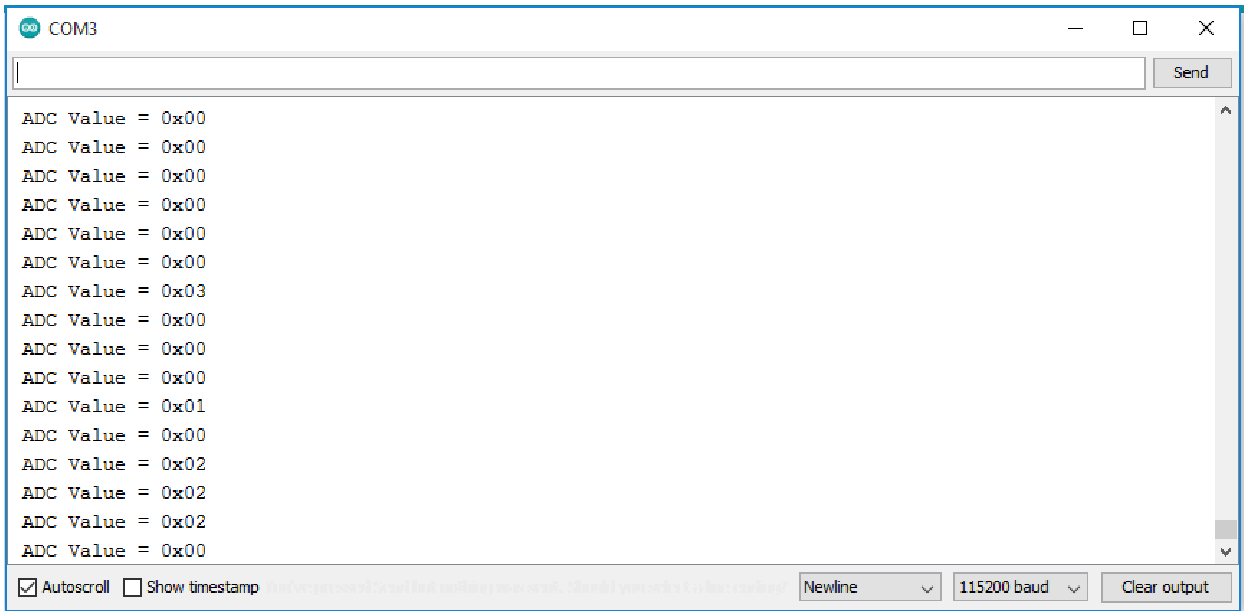
APROM | Data Flash

```
00000000:  02 0A D0 C0 E0 C0 83 C0 82 C0 91 30 98 0B D2 0A
00000010:  90 00 2A E5 99 F0 53 98 FE D0 91 D0 82 D0 83 D0
00000020:  E0 32 00 02 00 03 C0 E0 C0 83 C0 82 C0 91 30 F8
00000030:  0B 53 F8 FE 90 00 2B E5 9A F0 D2 0B 30 F9 06 20
00000040:  09 03 53 F8 FD D0 91 D0 82 D0 83 D0 E0 32 EF B4
00000050:  0A 07 74 0D 12 00 59 74 0A 30 98 11 A8 99 B8 13
00000060:  0C C2 98 30 98 FD A8 99 C2 98 B8 11 F6 30 99 FD
00000070:  C2 99 F5 99 22 C1 09 C1 0A 00 00 02 00 26 E5 08
00000080:  24 05 F5 82 E4 34 00 F5 83 E0 05 08 22 90 00 02
00000090:  30 07 03 90 00 05 E4 75 F0 01 12 05 D5 02 05 7D
000000A0:  20 00 E9 7F 2E D2 00 80 18 EF 54 0F 24 90 D4 34
000000B0:  40 D4 FF 30 04 0B EF 24 BF B4 1A 00 50 03 24 61
```

[✓ 8 bits] 16 bits 32 bits

Programming

[✓] APROM [ ] Data Flash [ ] Config [ ] Reset and Run [ ] Erase All — Start


COM3

Send

```
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x03
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x00
ADC Value = 0x01
ADC Value = 0x00
ADC Value = 0x02
ADC Value = 0x02
ADC Value = 0x02
ADC Value = 0x00
```

[✓] Autoscroll [ ] Show timestamp — Newline — 115200 baud — Clear output

**TAKEAWAY:**

1. Learned how to program and configure the onboard ADC of a microcontroller.
2. Understood the working principle of ADC in converting analog signals to digital values.
3. Gained practical knowledge of interfacing sensors like LDR and potentiometer with ADC.
4. Realized the importance of ADC in real-time applications for monitoring physical parameters like light and resistance.
5. Developed better clarity on how sensor data can be processed and used in embedded and IoT-based projects.

## CONCLUSION:

The experiment successfully demonstrated ADC functionality by reading analog inputs from an LDR and potentiometer, converting them to digital values, and showcasing sensor interfacing in embedded system

NECMML42: Embedded Systems and RTOS Lab

# Experiment 5

**Aim:** To blink two LEDs at same time on STM32 NUCLEO board.

**Apparatus:** STM32 NUCLEO board, Arduino IDE, LEDs, Breadboard, wires.

**Specifications of STM32F401RET6:**

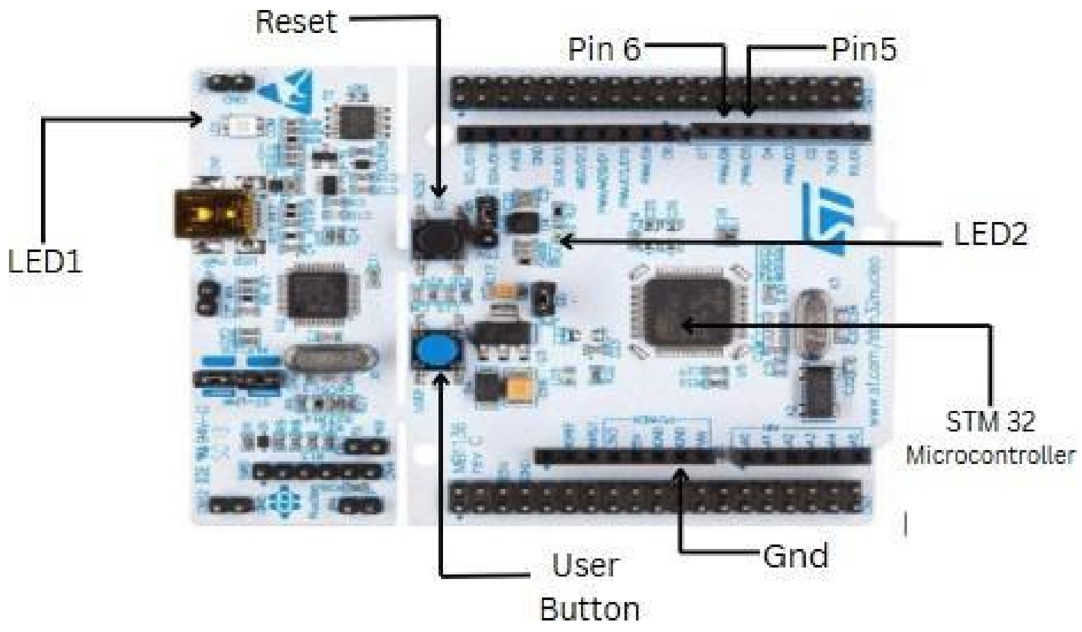
The MS51PC0AE is an 8-bit microcontroller based on the popular 8051 architecture. Here are some key features:

- **Operating Characteristics:**
    - VCC: 1.6 to 5.5 V
    - Temperature Range: -40℃ to +85℃
- **Memory:**
    - Up to 128-KB code flash memory
    - 4-KB data flash memory (100,000 program/erase (P/E) cycles)
    - 16-KB SRAM
    - Memory protection units
    - 128-bit unique ID
- **Clock:**
    - Main clock oscillator (MOSC) (1 to 20 MHz)
    - Sub-clock oscillator (SOSC) (32.768 kHz)
    - High-speed on-chip oscillator (HOCO) (24/32/48/64 MHz)
    - Middle-speed on-chip oscillator (MOCO) (8 MHz)
    - Low-speed on-chip oscillator (LOCO) (32.768 kHz)
    - Clock trim function for HOCO/MOCO/LOCO
    - IWDT-dedicated on-chip oscillator (15 kHz)
- **Timers:**
    - General PWM Timer 32-bit (GPT32) × 1
    - General PWM Timer 16-bit (GPT16) × 6
    - Low Power Asynchronous General Purpose Timer (AGT) × 2
    - Watchdog Timer (WDT)

- **Connectivity:**
  - Serial Communications Interface (SCI) × 4
  - Asynchronous interfaces
  - 8-bit clock synchronous interface
  - Simple IIC
  - Simple SPI
  - Smart card interface
  - Serial Peripheral Interface (SPI) × 1
  - I2C bus interface (IIC) × 1
- **Up to 56 pins for general I/O ports:**
  - 5 -V tolerance, open drain, input pull-up, switchable driving ability
- **Analog:**
  - 12-bit A/D Converter (ADC12)
  - Low-Power Analog Comparator (ACMPLP) × 2
  - Temperature Sensor (TSN)

**Procedure:**

1. Open Arduino IDE
2. Select Board Nucleo-64
3. Select Part Nucleo F401RE.
4. Connect STM32 Board with computer and check the port used under tools section.
5. Select Port number accordingly
6. Make necessary breadboard connections with board Pins and LEDs.
7. Upload the code and observe the output.



The **STM32F401RET6** is a 32-bit ARM Cortex-M4 based microcontroller from STMicroelectronics. It's commonly used in Nucleo boards and other development platforms for embedded system applications.

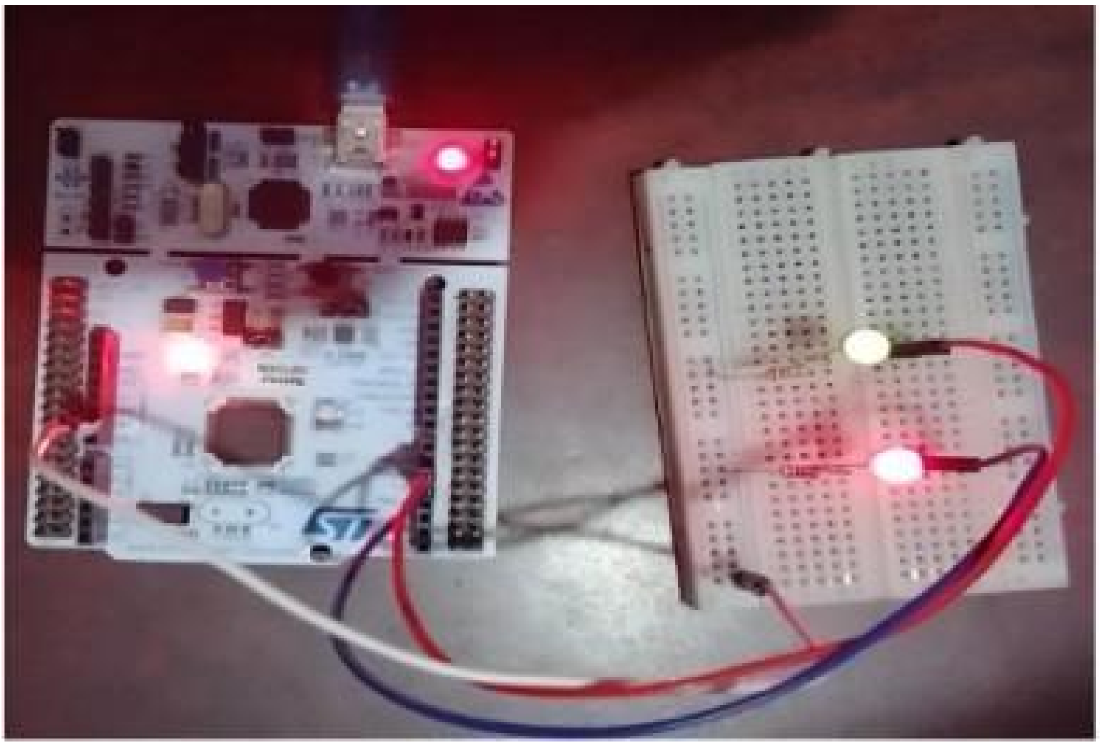
**Key Features of the STM32F401RET6 Board:**

1. ARM Cortex-M4 Core with FPU: 84 MHz clock speed with hardware floating-point support for efficient signal processing.
2. 512 KB Flash & 96 KB SRAM: Sufficient memory for mid-size embedded applications.
3. Multiple Communication Interfaces: Includes USART, SPI, I2C, USB OTG, and CAN for versatile connectivity.
4. Advanced Timer & ADC Support: Multiple timers and a 12-bit ADC with up to 16 channels, useful for real-time control and sensor interfacing.
5. Low Power Modes & On-Chip Peripherals: Supports various low-power modes, real-time clock, watchdogs, and DMA for efficient system performance.

**Code:**

```
#include <FreeRTOS.H>

#include <task.h>

// Define LED pins

#define LED1_PIN PA5

#define LED2_PIN PA6

#define TASK1_DELAY 500
#define TASK2_DELAY 2000
void task1(void *pvParameters)
{
(void)pvParameters; pinMode(LED1_PIN, OUTPUT);
while (1)
{digitalWrite(LED1_PIN, HIGH);vTaskDelay(pdMS_TO_TICKS(TASK1_DELAY));
digitalWrite(LED1_PIN, LOW);vTaskDelay(pdMS_TO_TICKS(TASK1_DELAY));}
}
void task2(void *pvParameters) {

(void)pvParameters; pinMode(LED2_PIN, OUTPUT);

while (1)

{digitalWrite(LED2_PIN, HIGH);vTaskDelay(pdMS_TO_TICKS(TASK2_DELAY));
digitalWrite(LED2_PIN, LOW);vTaskDelay(pdMS_TO_TICKS(TASK2_DELAY));}

}

void setup()

{xTaskCreate(task1, "Task1", 100, NULL, 1, NULL);xTaskCreate(task2, "Task2", 100, NULL,
2, NULL);vTaskStartScheduler();}

void loop() {}
```

**Output:**



**Takeaways:**

1. Parallel Task Execution with FreeRTOS: Learned how to create and run multiple tasks simultaneously using FreeRTOS (xTaskCreate), enabling independent LED blinking.
2. GPIO Access on STM32 Board: Gained hands-on experience with configuring and controlling GPIO pins using pinMode() and digitalWrite() functions.
3. Delay Management using vTaskDelay(): Understood how task delays control LED blink timing and how changes in delay values affect blinking frequency and duty cycle.
4. Task Scheduling and Prioritization: Implemented simple task scheduling by assigning priorities to tasks, illustrating FreeRTOS's multitasking capabilities.
5. STM32-Arduino IDE Integration: Practiced integrating STM32 with the Arduino IDE environment for programming and uploading code using board-specific configurations.

**Conclusion:**

The experiment demonstrated controlling LED blink frequency by varying GPIO set and reset delays, allowing for adjustable blink speed and on/off durations, and showcasing GPIO port access on the STM32 board.

# Experiment 6

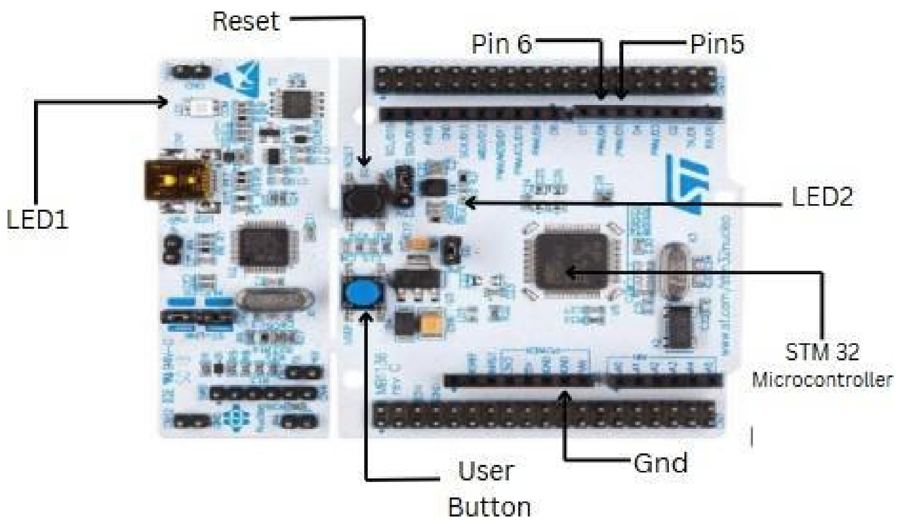
**Aim:** To change the brightness of LEDs on STM32 with and without FreeRTOS

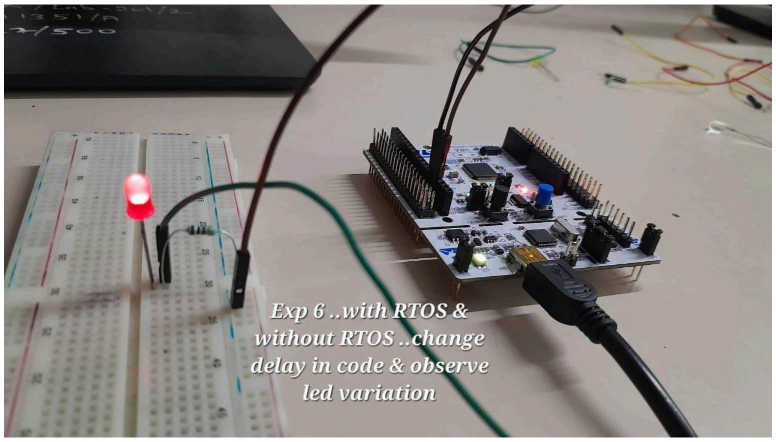
**Apparatus:** STM32 NUCLEO board, Arduino IDE, LED, Breadboard, wires.

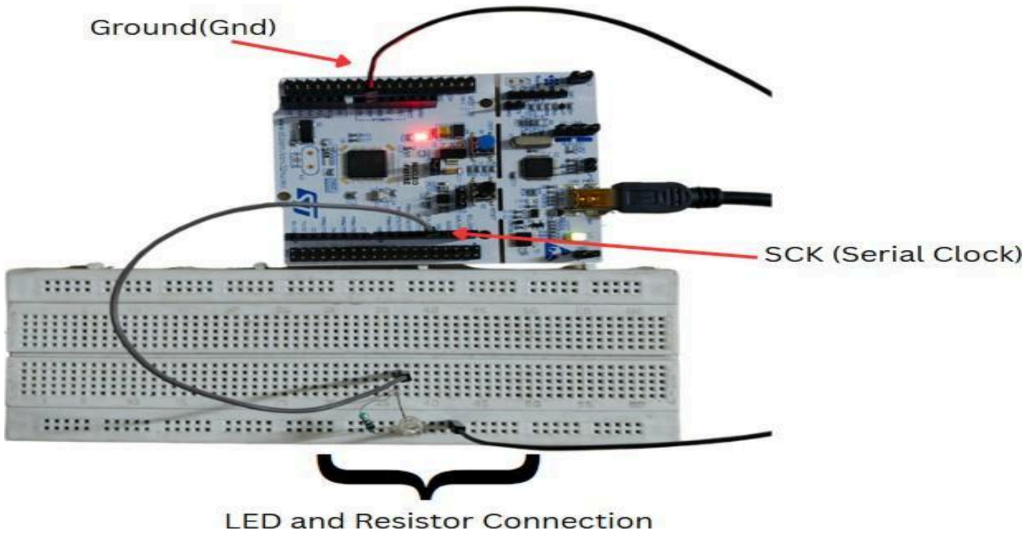
## STM32F401RET6 Specifications:

- Operates at a voltage range of **1.6 to 5.5 V**.
- Supports a temperature range of **-40°C to +85°C**.
- Includes **128 KB Flash Memory**, **4 KB Data Flash**, and **16 KB SRAM**.
- Features a **128-bit Unique ID** for secure identification.
- Supports **multiple on-chip clock sources** with **clock trimming functionality**.
- Offers **SCI**, **SPI**, **I2C**, and **Smart Card Interface** for communication.
- Provides **56 GPIO pins** with **5-V tolerance** and **open-drain capability**.
- Includes **one 32-bit** and **multiple 16-bit PWM timers**.
- Equipped with **asynchronous timers** and a **Watchdog Timer (WDT)**.
- Features a **12-bit ADC**, **2 low-power analog comparators**, and an **internal temperature sensor**.

## STM32 NUCLEO Connection Board:

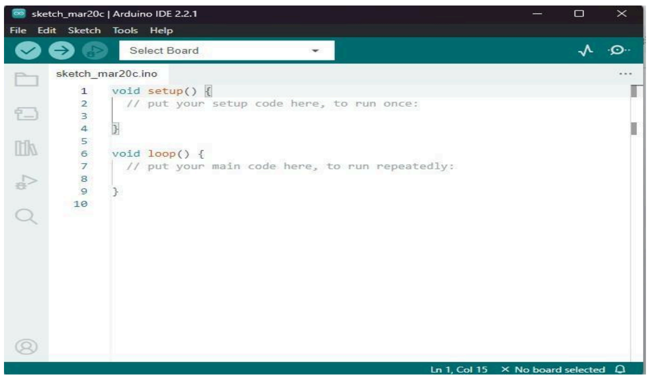



Exp 6 ..with RTOS & without RTOS ..change delay in code & observe led variation



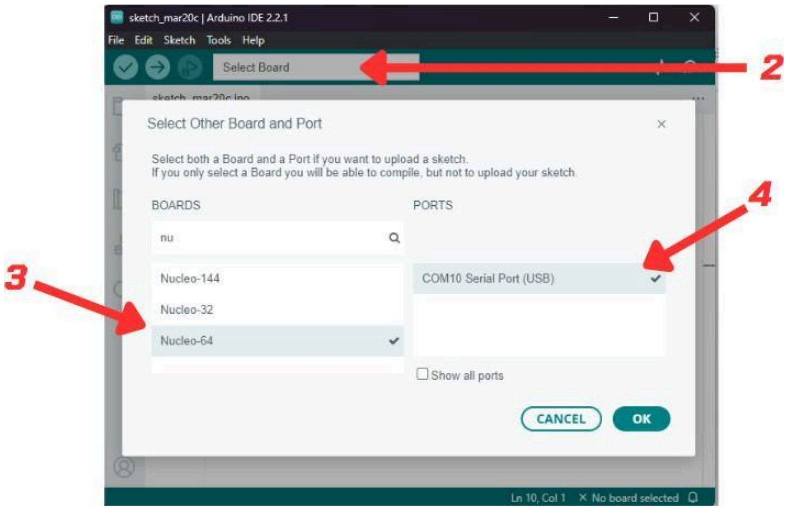
**Procedure:**

1. Open Arduino IDE


sketch_mar20c | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Select Board
sketch_mar20c.ino
Ln 1, Col 15 × No board selected

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

2. Select Board Nucleo-64
3. Connect STM32 Board with computer.
4. Select Port number accordingly


sketch_mar20c | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Select Board
2
3
4
CANCEL
OK
Ln 10, Col 1 × No board selected

Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS: nu — Nucleo-144, Nucleo-32, Nucleo-64 ✓

PORTS: COM10 Serial Port (USB) ✓

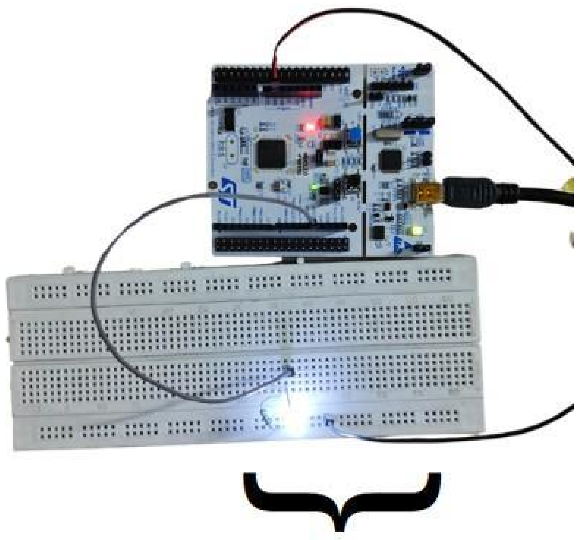
☐ Show all ports

5. Make necessary breadboard connections with board Pins and LEDs.
6. Upload the code and observe the output.

## A) Code using RTOS

```
#include <Arduino.h>
#include <FreeRTOS.h>
#include <task.h>
#define LED_PIN_1 PA5

#define INITIAL_DELAY_MS 500 // Initial delay in milliseconds
#define MAX_COUNT 10 // Maximum count for brightness increase
TaskHandle_t task1Handle;
TaskHandle_t task2Handle;
void task1(void *pvParameters)
{int brightness = 0;int count = 0;int delay_ms = INITIAL_DELAY_MS;
pinMode(LED_PIN_1, OUTPUT);while (1)
{analogWrite(LED_PIN_1, brightness); // Set brightness of LED for Task 1
delay(delay_ms); brightness += 255 / MAX_COUNT; if (brightness > 255) {brightness =
0;count++;}
if (count >= MAX_COUNT){count = 0;delay_ms = INITIAL_DELAY_MS;} } }
void setup()
{Serial.begin(9600);xTaskCreate(task1,     "Task     1",     1000,     NULL,     1,     task
Handle);vTaskStartScheduler();}
void loop() {}
```

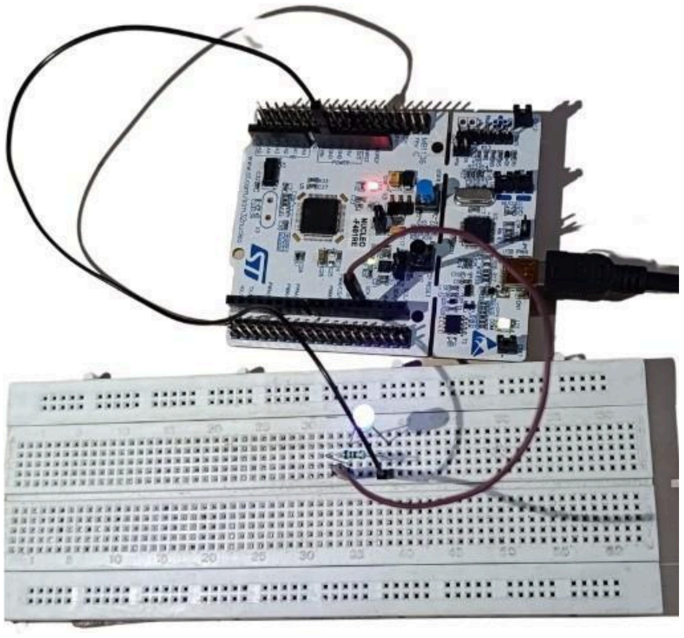
**Output:**



## B) Code without RTOS

```
#include <Arduino.h>
#define LED_PIN PA5
void setup() {pinMode(LED_PIN, OUTPUT);}
void loop()
{
static int brightness = 0; static bool increasing = true;
static int count = 10000; // Initial count value
analogWrite(LED_PIN, brightness);
if (increasing) {brightness += 20; if (brightness >= 255) {increasing = false;}}
else {brightness -= 20; if (brightness <= 0) {increasing = true;count--;} }
delayMicroseconds(count);
if (count <= 10000) {count = 100000;}
}
```

**Output:**



**Takeaways:**

1. **Works in Many Conditions**
   It can handle different power levels (1.6V to 5.5V) and temperatures (as low as -40°C), so it's great for all kinds of projects — even in tough environments.

2. **Enough Memory for Your Code**
   You get 128 KB space to store your program and 16 KB RAM to run it — that's plenty for most embedded applications.

3. **Can Talk to Many Devices**
   It supports communication methods like SPI, I2C, and UART, so you can easily connect it to sensors, displays, or other microcontrollers.

4. **Great for Timed Tasks**
   It has timers to create things like blinking LEDs, motor control, and alarms — and it even has a special timer (WDT) to reset itself if something goes wrong.

5. **Can Sense Things Too**
   With built-in tools like an analog-to-digital converter (ADC) and a temperature sensor, it can read real-world data like voltage and heat.

## Conclusion:

The LED was successfully interfaced with the Nucleo-64 board using both FreeRTOS and without an RTOS. With FreeRTOS, the LED blinking speed can be changed in real-time, while without it, the speed remains fixed due to continuous loop-based delays.

VESIT V.E.S. Since 1962 VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

# Experiment 7

**Aim:** To implement Binary Semaphore for Task Synchronization using FreeRTOS

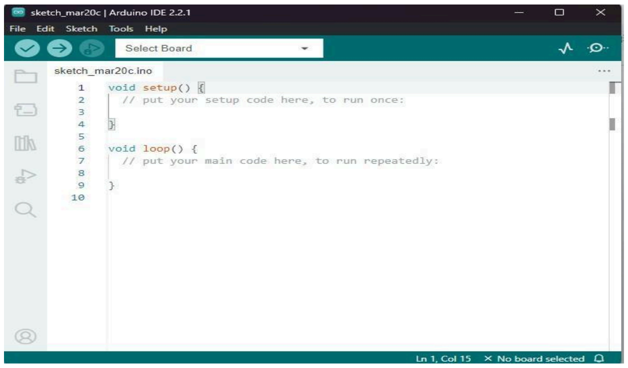
**Requirements:** STM32 NUCLEO board, Arduino IDE.

**Specifications of STM32F401RET6:**

- **Operating Voltage:** 1.6V – 5.5V
- **Temperature Range:** -40°C to +85°C
- **Memory:** 128 KB Flash, 4 KB Data Flash (100K P/E cycles), 16 KB SRAM, Memory Protection Units, 128-bit Unique ID
- **Clock System:** Multiple on-chip oscillators (1–64 MHz), sub/low-speed clocks, clock trim, 15 kHz IWDT oscillator
- **Connectivity:** 4× SCI, SPI, I2C, simple IIC/SPI, smart card interface, async/sync interfaces
- **Timers:** 1× 32-bit GPT, 6× 16-bit GPT, 2× Low-power AGT, Watchdog Timer
- **I/O Ports:** Up to 56 GPIOs, 5V-tolerant, open-drain, input pull-up, adjustable drive
- **Analog Features:** 12-bit ADC, 2× Comparators, Temperature Sensor.

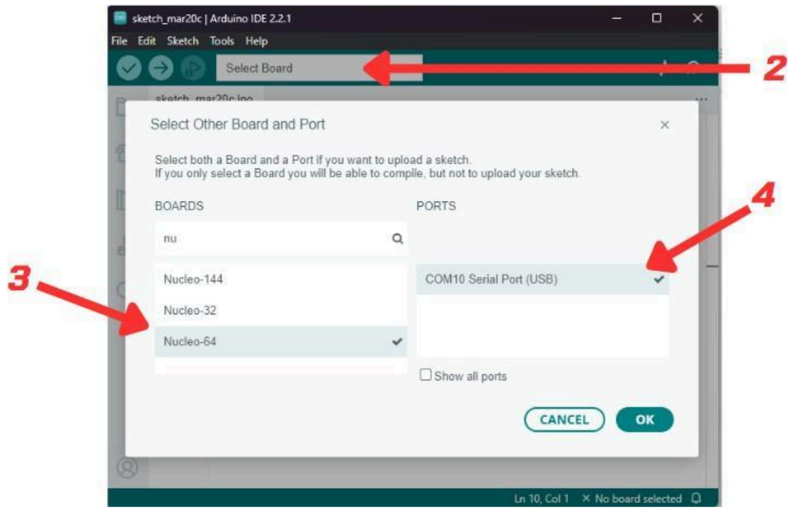
## Procedure:

1. Open Arduino IDE


sketch_mar20c | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Select Board
sketch_mar20c.ino
Ln 1, Col 15 ✕ No board selected

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

2. Select Board Nucleo-64
3. Connect STM32 Board with computer.
4. Select Port number accordingly


sketch_mar20c | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Select Board
2
Select Other Board and Port
BOARDS
PORTS
nu
Nucleo-144
Nucleo-32
Nucleo-64
COM10 Serial Port (USB)
Show all ports
CANCEL
OK
3
4
Ln 10, Col 1 × No board selected

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

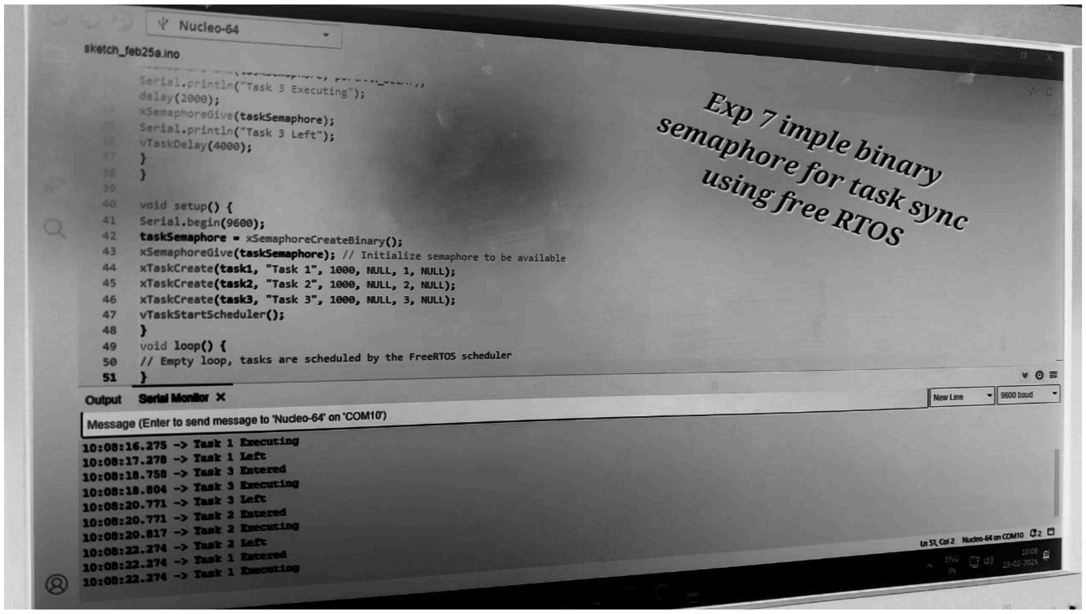
5. Make necessary breadboard connections with board Pins and LEDs.
6. Upload the code and observe the output.
7. Open the Serial monitor and observe the output.

**Code in main.c:**

```
#include <Arduino.h>
#include <FreeRTOS.h>
#include <task.h>
#include <semphr.h>
SemaphoreHandle_t taskSemaphore;
void task1(void *pvParameters) {
while (1) {
Serial.println("Task 1 Entered");
xSemaphoreTake(taskSemaphore, portMAX_DELAY);
Serial.println("Task 1 Executing");
delay(1000);
xSemaphoreGive(taskSemaphore);
Serial.println("Task 1 Left");
vTaskDelay(2000);
}
}
void task2(void *pvParameters) {
```

```
while (1) {
Serial.println("Task 2 Entered");
xSemaphoreTake(taskSemaphore, portMAX_DELAY);
Serial.println("Task 2 Executing");
delay(1500);
xSemaphoreGive(taskSemaphore);
Serial.println("Task 2 Left");
vTaskDelay(3000);
}
}
void task3(void *pvParameters) {
while (1) {
Serial.println("Task 3 Entered");
xSemaphoreTake(taskSemaphore, portMAX_DELAY);
Serial.println("Task 3 Executing");
delay(2000);
xSemaphoreGive(taskSemaphore);
Serial.println("Task 3 Left");
vTaskDelay(4000);
}
}

void setup() {
Serial.begin(9600);
taskSemaphore = xSemaphoreCreateBinary();
xSemaphoreGive(taskSemaphore); // Initialize semaphore to be available
xTaskCreate(task1, "Task 1", 1000, NULL, 1, NULL);
xTaskCreate(task2, "Task 2", 1000, NULL, 2, NULL);
xTaskCreate(task3, "Task 3", 1000, NULL, 3, NULL);
vTaskStartScheduler();
}
void loop() {
// Empty loop, tasks are scheduled by the FreeRTOS scheduler
}
```

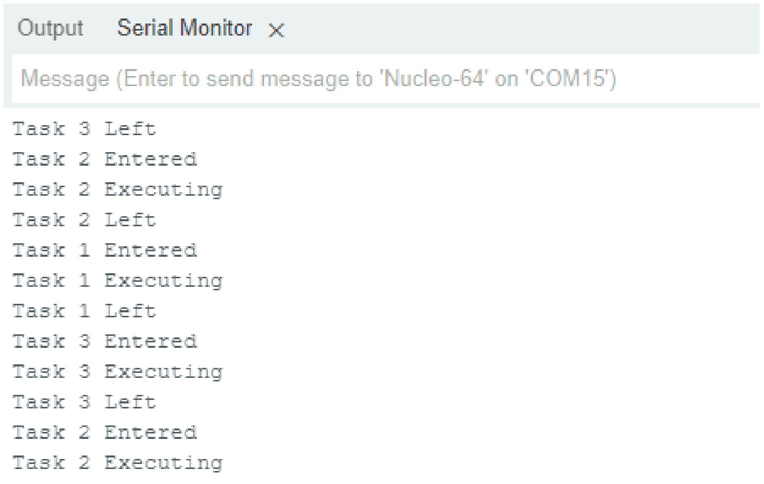
## Output:



```
    Serial.println("Task 3 Executing");
    delay(2000);
    xSemaphoreGive(taskSemaphore);
    Serial.println("Task 3 Left");
    vTaskDelay(4000);
    }
    }

    void setup() {
    Serial.begin(9600);
    taskSemaphore = xSemaphoreCreateBinary();
    xSemaphoreGive(taskSemaphore); // Initialize semaphore to be available
    xTaskCreate(task1, "Task 1", 1000, NULL, 1, NULL);
    xTaskCreate(task2, "Task 2", 1000, NULL, 2, NULL);
    xTaskCreate(task3, "Task 3", 1000, NULL, 3, NULL);
    vTaskStartScheduler();
    }
    void loop() {
    // Empty loop, tasks are scheduled by the FreeRTOS scheduler
    }
```

Message (Enter to send message to 'Nucleo-64' on 'COM10')

```
10:08:16.275 -> Task 1 Executing
10:08:17.278 -> Task 1 Left
10:08:18.758 -> Task 3 Entered
10:08:18.804 -> Task 3 Executing
10:08:20.771 -> Task 3 Left
10:08:20.771 -> Task 2 Entered
10:08:20.817 -> Task 2 Executing
10:08:22.274 -> Task 2 Left
10:08:22.274 -> Task 1 Entered
10:08:22.274 -> Task 1 Executing
```



Message (Enter to send message to 'Nucleo-64' on 'COM15')

```
Task 3 Left
Task 2 Entered
Task 2 Executing
Task 2 Left
Task 1 Entered
Task 1 Executing
Task 1 Left
Task 3 Entered
Task 3 Executing
Task 3 Left
Task 2 Entered
Task 2 Executing
```

## Takeaways:

1. **Binary Semaphore enables task synchronization**
   – It ensures that only one task accesses the shared resource at a time, avoiding conflicts.

2. **Task execution depends on semaphore availability**
   – A task will wait (block) until it successfully takes the semaphore, ensuring proper task order.
3. **Task priorities affect execution order**
   – Higher-priority tasks (like Task 3) execute first, but all tasks must wait for the semaphore to proceed.
4. **FreeRTOS scheduler manages task switching efficiently**
   – It automatically handles which task runs next based on priority and semaphore status.
5. **Serial Monitor helps observe real-time task behavior**
   – Output shows clear entry, execution, and exit logs for each task, validating correct semaphore use.

## Conclusion:

A binary semaphore is implemented to manage task execution, with output observed based on assigned priority levels. Initially, Task 3, 2, and 1 execute in order, after which the scheduler manages task switching based on the semaphore's availability and task status.

VIVEKANAND EDUCATION SOCIETY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

# Experiment 8

**Aim:** To create tasks and scheduling them as per priority using STM32 and FREERTOS.

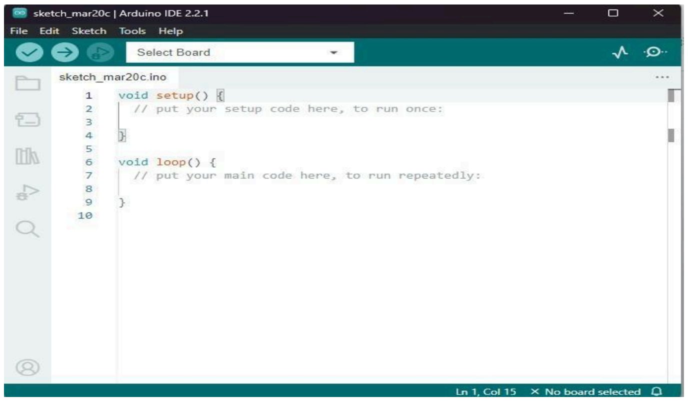
**Requirements:** STM32F401RET6 NUCLEO-64 board, Arduino IDE, FreeRTOS

**Specifications of STM32F401RET6:**

- **Operating Voltage:** 1.6V – 5.5V
- **Temperature Range:** -40°C to +85°C
- **Memory:** 128 KB Flash, 4 KB Data Flash (100K P/E cycles), 16 KB SRAM, Memory Protection Units, 128-bit Unique ID
- **Clock System:** Multiple on-chip oscillators (1–64 MHz), sub/low-speed clocks, clock trim, 15 kHz IWDT oscillator
- **Connectivity:** 4× SCI, SPI, I2C, simple IIC/SPI, smart card interface, async/sync interfaces
- **Timers:** 1× 32-bit GPT, 6× 16-bit GPT, 2× Low-power AGT, Watchdog Timer
- **I/O Ports:** Up to 56 GPIOs, 5V-tolerant, open-drain, input pull-up, adjustable drive
- **Analog Features:** 12-bit ADC, 2× Comparators, Temperature Sensor

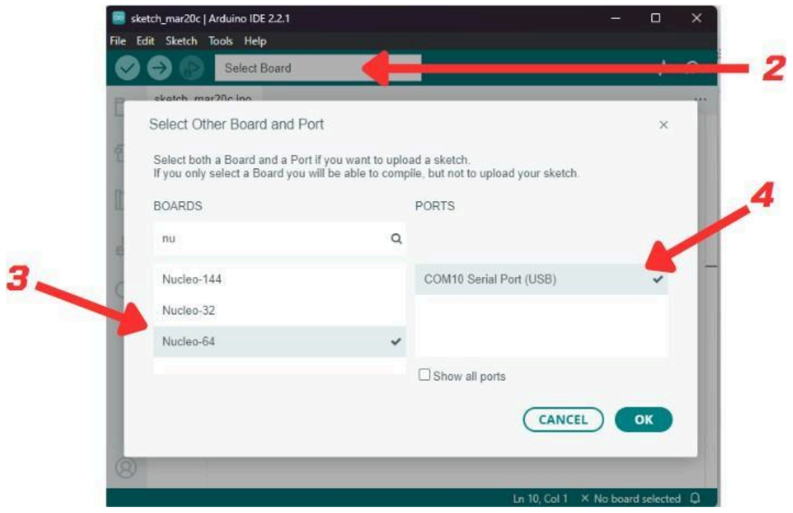
**Procedure:**

1. Open Arduino IDE


sketch_mar20c | Arduino IDE 2.2.1
File Edit Sketch Tools Help
Select Board
sketch_mar20c.ino
Ln 1, Col 15 × No board selected

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

2. Select Board Nucleo-64
3. Connect STM32 Board with computer.
4. Select Port number accordingly



sketch_mar20c | Arduino IDE 2.2.1

File Edit Sketch Tools Help

Select Board

Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS: nu — Nucleo-144, Nucleo-32, Nucleo-64 ✓

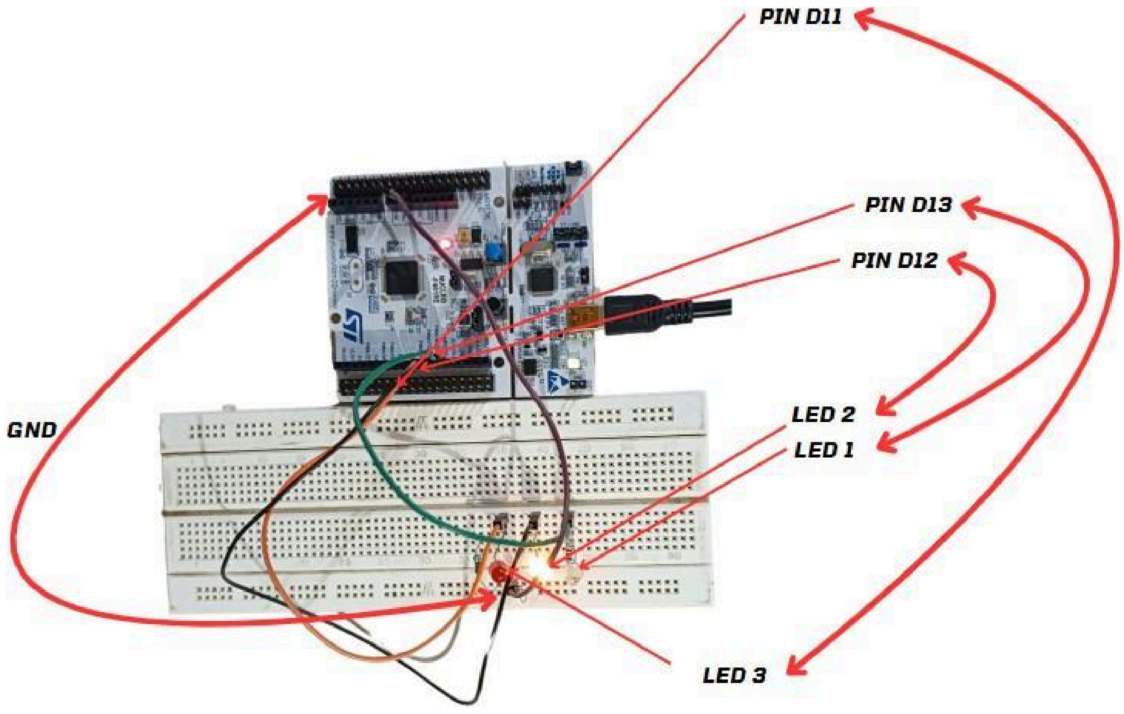
PORTS: COM10 Serial Port (USB) ✓

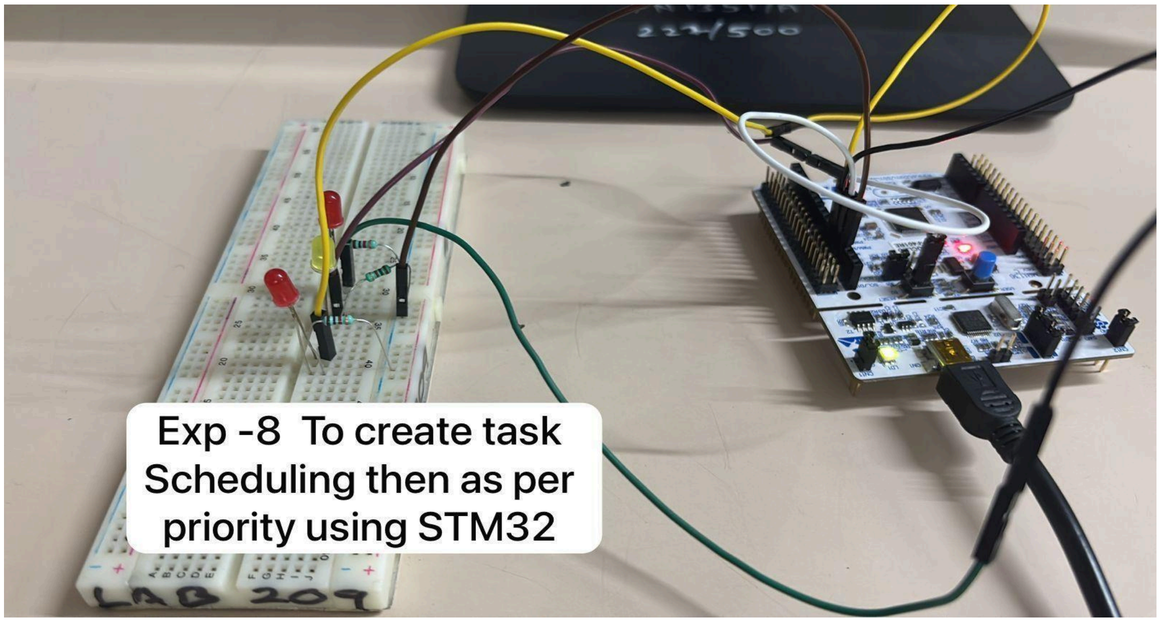
☐ Show all ports

CANCEL OK

Ln 10, Col 1 × No board selected

5. Make necessary breadboard connections with board Pins and LEDs.




Exp -8 To create task Scheduling then as per priority using STM32
LAB 209

6. Upload the code and observe the output.

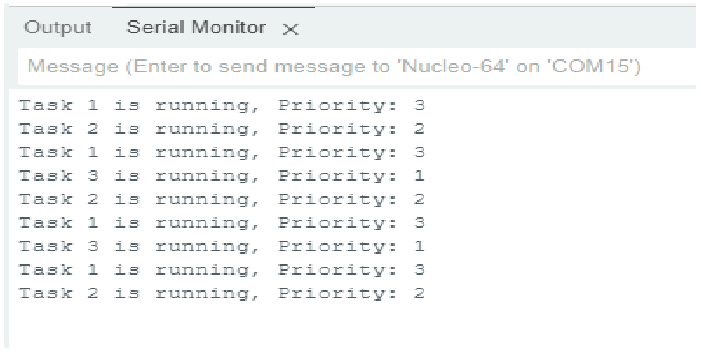
**Code in main.c file:**

```
#include <Arduino.h>
#include <FreeRTOS.h>
#include <task.h>
#define LED_PIN_1 PA5 // Define LED pin for Task 1
#define LED_PIN_2 PA6 // Define LED pin for Task 2
#define LED_PIN_3 PA7 // Define LED pin for Task 3
TaskHandle_t task Handle;
TaskHandle_t task2Handle;
TaskHandle_t task Handle;
void task1(void
*pvParameters)
{ pinMode(LED_PIN_1, OUTPUT);
pinMode(LED_PIN_2, OUTPUT);
pinMode(LED_PIN_3, OUTPUT);
while (1) {
digitalWrite(LED_PIN_1, HIGH);
digitalWrite(LED_PIN_2, LOW);
digitalWrite(LED_PIN_3, LOW);
```

```
Serial.print("Task 1 is running, Priority: ");
Serial.println(uxTaskPriorityGet(NULL));
vTaskDelay(1000);
digitalWrite(LED_PIN_1, LOW);
vTaskDelay(2000);
}}
void task2(void *pvParameters)
{
pinMode(LED_PIN_1, OUTPUT);
pinMode(LED_PIN_2, OUTPUT);
pinMode(LED_PIN_3, OUTPUT);
while (1) {
digitalWrite(LED_PIN_1, LOW);
digitalWrite(LED_PIN_2, HIGH);
digitalWrite(LED_PIN_3, LOW);
Serial.print("Task 2 is running, Priority: ");
Serial.println(uxTaskPriorityGet(NULL));

vTaskDelay(1000);
digitalWrite(LED_PIN_2, LOW);
vTaskDelay(3000);
}}
void task3(void *pvParameters)
{
pinMode(LED_PIN_1, OUTPUT);
pinMode(LED_PIN_2, OUTPUT);
pinMode(LED_PIN_3, OUTPUT);
while (1) {
digitalWrite(LED_PIN_1, LOW);
digitalWrite(LED_PIN_2, LOW);
digitalWrite(LED_PIN_3, HIGH);
Serial.print("Task 3 is running, Priority: ");
Serial.println(uxTaskPriorityGet(NULL));
vTaskDelay(1000);
digitalWrite(LED_PIN_3, LOW);
```

```
vTaskDelay(4000);
}}
void setup()
{ Serial.begin(9600);
xTaskCreate(task1, "Task 1", 500, NULL, 3, task Handle); // Highest priority
xTaskCreate(task2, "Task 2", 500, NULL, 2, &task Handle);
xTaskCreate(task3, "Task 3", 500, NULL, 1, &task Handle); // Lowest priority
vTaskStartScheduler();
}
void loop() {}
```

**Output:**



```
Task 1 is running, Priority: 3
Task 2 is running, Priority: 2
Task 1 is running, Priority: 3
Task 3 is running, Priority: 1
Task 2 is running, Priority: 2
Task 1 is running, Priority: 3
Task 3 is running, Priority: 1
Task 1 is running, Priority: 3
Task 2 is running, Priority: 2
```

**Takeaways:**

1. **FreeRTOS Enables Multitasking:** FreeRTOS allows running multiple tasks independently on the STM32 board.

2. **Priority-Based Execution:** Tasks with higher priority run first, helping manage time-critical operations.

3. **LEDs Represent Task States:** Three LEDs were used to visually show each task's activity.

4. **Task Delays Control Timing:** vTaskDelay() is used to control how long each task runs or waits.

5. **Arduino IDE Compatibility:** STM32 NUCLEO-64 works smoothly with Arduino IDE and FreeRTOS.

## Conclusion:

This project demonstrates task scheduling using FreeRTOS on STM32. All three tasks run independently with priorities, confirming successful multitasking. Multitasking is possible using STM32 board.

## 1) HUMIDITY & TEMPERATURE SENSOR

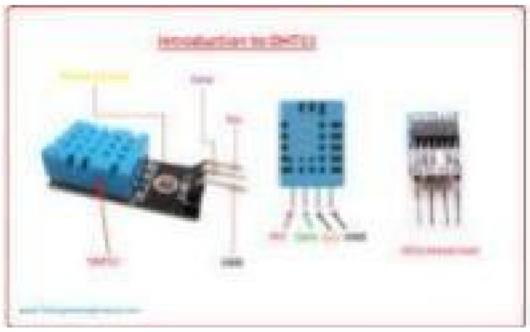


**Fig 1.0**

DHT11 Temperature and Humidity Sensor

A **Humidity & Temperature Sensor** is an electronic device used to measure the **moisture content (humidity)** and **temperature** of the surrounding environment. These sensors are widely used in weather monitoring, HVAC systems, greenhouses, industrial applications, and smart home devices. The DHT11 is a basic, low-cost digital temperature and humidity sensor. It's widely used in hobbyist projects and applications where high accuracy isn't critical.

1. **Sensor Characteristics:**

- Measurement Range:
    - Temperature: 0°C to 50°C
    - Humidity: 20% to 80% RH
- Accuracy:
    - Temperature: ±2°C
    - Humidity: ±5% RH
- Resolution:
    - Temperature: 1°C
    - Humidity: 1% RH
- Operating Voltage: 3.3V to 5V
- Digital Output: Provides a digital signal.

- Sampling Rate: 1 Hz (one reading per second)

2. **Working Principle:**

- Humidity Sensing: The DHT11 uses a capacitive humidity sensor. This sensor consists of two electrodes with a moisture-holding substrate between them. Changes in humidity alter the substrate's conductivity, which in turn changes the capacitance. The sensor's internal IC measures this capacitance change and converts it into a digital humidity value.
- Temperature Sensing: The DHT11 uses an NTC (Negative Temperature Coefficient) thermistor to measure temperature. A thermistor's resistance changes with temperature. The sensor's IC measures this resistance change and converts it into a digital temperature value.
- **Digital Signal Transmission:** The DHT11 transmits temperature and humidity data as a digital signal over a single wire. This signal consists of a 40-bit data packet.

3. **Interfacing with Arduino:**

- Wiring:
  - VCC: Connect to the Arduino's 5V or 3.3V pin.
  - GND: Connect to the Arduino's GND pin.
  - Data: Connect to a digital pin on the Arduino. A pull-up resistor (typically 10 kΩ) is often required on the data line.

- Communication: The DHT11 uses a custom single-wire communication protocol. Arduino libraries (like the "DHT sensor library" by Adafruit) simplify this communication.

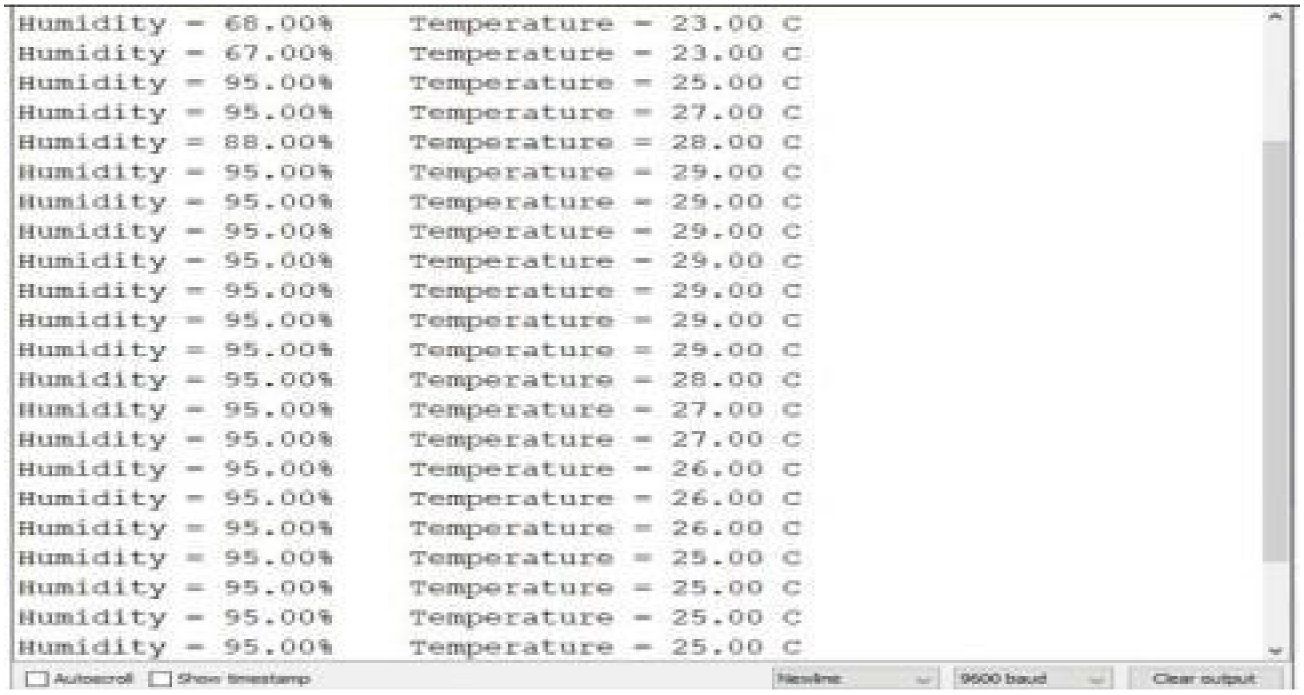
4. **Applications:**

- Basic weather monitoring.
- Home automation.
- Environmental monitoring.
- Educational projects.

## CODE FOR TEMP. & HUMIDITY SENSOR:-

```
#include <DHT.h>
#define DHTPIN 7
#define DHTTYPE DHT11
DHT dht(DHTPIN,
DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
    Serial.print("Humidity:
    ");
    Serial.print(humidity);
    Serial.print("% Temperature: ");
    Serial.print(temperature);
    Serial.println("°C");
  }

  delay(2000);
```

**OUTPUT:**



```
Humidity = 68.00%     Temperature = 23.00 C
Humidity = 67.00%     Temperature = 23.00 C
Humidity = 95.00%     Temperature = 25.00 C
Humidity = 95.00%     Temperature = 27.00 C
Humidity = 88.00%     Temperature = 28.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 29.00 C
Humidity = 95.00%     Temperature = 28.00 C
Humidity = 95.00%     Temperature = 27.00 C
Humidity = 95.00%     Temperature = 27.00 C
Humidity = 95.00%     Temperature = 26.00 C
Humidity = 95.00%     Temperature = 26.00 C
Humidity = 95.00%     Temperature = 26.00 C
Humidity = 95.00%     Temperature = 25.00 C
Humidity = 95.00%     Temperature = 25.00 C
Humidity = 95.00%     Temperature = 25.00 C
Humidity = 95.00%     Temperature = 25.00 C
```

Autoscroll  Show timestamp  Newline  9600 baud  Clear output

Fig 1.1

**CONCLUSION:-**

Through the practical, we successfully interfaced the humidity and temperature sensor with the microcontroller and observed real-time readings. This helped us understand how the sensor measures environmental conditions and how the data can be read, processed, and displayed. The experiment enhanced our knowledge of sensor calibration, digital data communication, and real-world applications of humidity and temperature monitoring. Overall, it provided hands-on experience in working with environmental sensors used in modern IoT and automation systems.

## 2) ULTRASONIC SENSOR

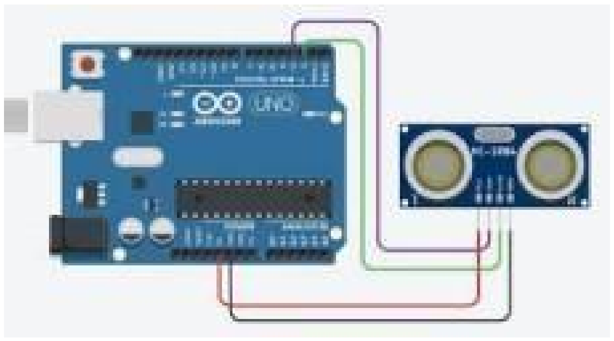


**Fig 2.0**

An Ultrasonic Sensor is an electronic device that uses ultrasound waves to measure distance to an object. It emits high-frequency sound waves (typically around 40 kHz) and measures the time it takes for the echo to return after hitting an object.

The HC-SR04 is a widely used, low-cost ultrasonic distance sensor. It's commonly found in robotics, automation, and other projects where non-contact distance measurement is required.

1. **Sensor Characteristics**: Measurement Range: Distance: 2cm to 400cm Angle: 15 Degrees

- **Accuracy**: Distance: 3mm
- **Resolution**: Distance: 0.3cm
- **Operating Voltage**: 5V DC Digital Output: Echo pulse width (proportional to distance). Trigger Input: 10µS pulse.

**2.** **Working Principle**: Trigger: To initiate a measurement, a 10µS pulse is sent to the sensor's trigger (TRIG) pin.

- **Transmission**: Upon receiving the trigger pulse, the sensor transmits an 8-cycle burst of 40kHz ultrasonic sound waves. Reception: When these sound waves encounter an object, they are reflected back towards the sensor. The receiver (ECHO) pin goes HIGH.
- **Echo Pulse**: The ECHO pin remains HIGH for a duration proportional to the time it takes for the sound wave to travel to the object and back.

- **Distance Calculation**: Distance = (Speed of sound × Time) / 2.

**3. Interfacing with Arduino**: **Wiring**: VCC: Connect to the Arduino's 5V pin.

- GND: Connect to the Arduino's GND pin. TRIG: Connect to a digital output pin on the Arduino
- ECHO: Connect to a digital input pin on the Arduino.
- Communication: The HC-SR04 uses trigger and echo pulses. Arduino's pulseIn()
- function simplifies the echo pulse duration measurement.

4. **Applications**:

- Robotics obstacle avoidance.
- Automated object detection.
- Security system intrusion detection.
- Liquid level measurement.
- Parking assistance systems.
- Distance-based control systems.

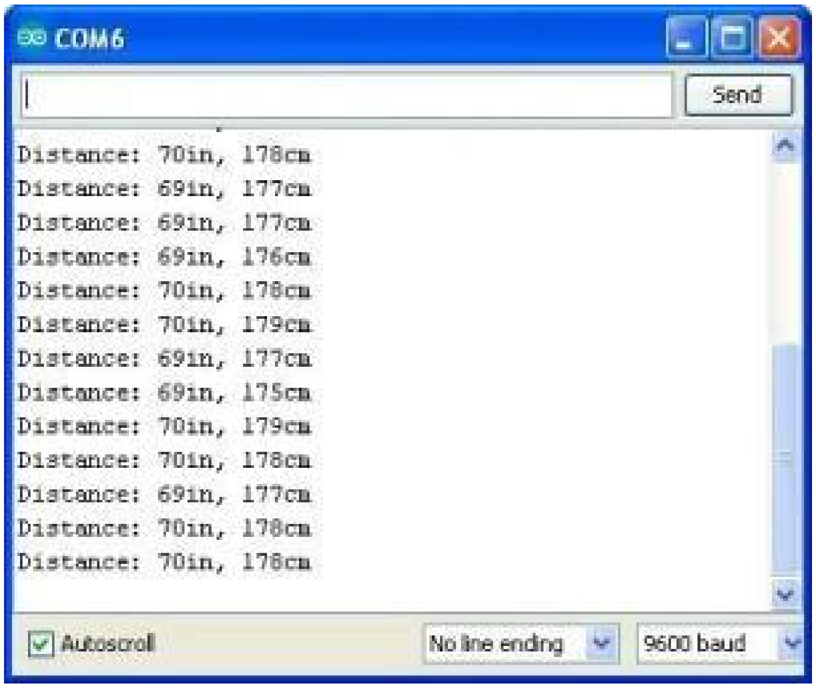
**CODE FOR ULTRASONIC SENSOR:-**

```
#define TRIG_PIN 9
#define ECHO_PIN 10
#define LED_PIN 13

void setup() {
 pinMode(TRIG_PIN,
 OUTPUT);
 pinMode(ECHO_PIN, INPUT);
 pinMode(LED_PIN, OUTPUT);
 Serial.begin(9600);
 }

void loop() {
```

```
long duration;
float distance;

// Send a pulse to the ultrasonic sensor to start measurement
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);

// Measure the duration of the pulse
duration = pulseIn(ECHO_PIN, HIGH);

// Calculate distance in centimeters
distance = duration * 0.0343 / 2;

// Print distance on the serial
monitor Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");
// Control LED based on distance
if (distance < 20) {
digitalWrite(LED_PIN, HIGH);
Serial.println("Object detected nearby!");
} else {
digitalWrite(LED_PIN,
LOW);
Serial.println("No object detected.");
}

delay(500); // Wait half a second before the next measurement
}
```

**Trigger to pin 6 of uno Echo to pin 7 of uno board Vcc & ground to power pin**

**OUTPUT**



```
Distance: 70in, 178cm
Distance: 69in, 177cm
Distance: 69in, 177cm
Distance: 69in, 176cm
Distance: 70in, 178cm
Distance: 70in, 179cm
Distance: 69in, 177cm
Distance: 69in, 175cm
Distance: 70in, 179cm
Distance: 70in, 178cm
Distance: 69in, 177cm
Distance: 70in, 178cm
Distance: 70in, 178cm
```

**Fig 2.1**

**CONCLUSION:-**

The practical using the ultrasonic sensor helped us understand how sound waves can be used for non contact distance measurement. By interfacing the sensor with a microcontroller and analyzing the time taken for the echo to return, we were able to accurately measure the distance to various objects. This experiment provided valuable insights into sensor integration, real-time data processing, and the applications of ultrasonic sensing in fields like robotics, automation, and level detection.

## 3) IR SENSOR

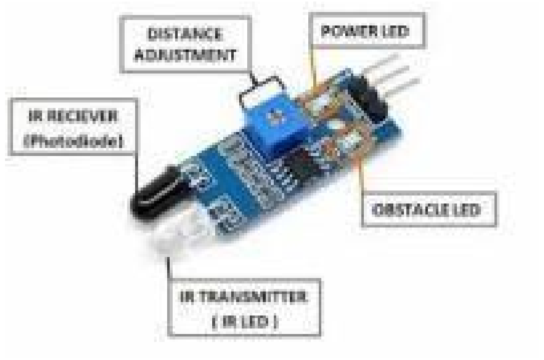


**Fig 3.0**

An IR (Infrared) Sensor is an electronic device that detects infrared radiation from objects in its surroundings. It is commonly used for object detection, proximity sensing, and line-following robots. IR Distance Sensor (e.g., Sharp GP2Y0A21YK0F) The IR distance sensor uses infrared light to measure distances. It's used in robotics, automation, and object detection.

**1. Sensor Characteristics:**

- **Measurement Range:** Distance: 10cm to 80cm
- **Angle:** Varies (depending on model)
- **Accuracy: Distance:** Varies(non-linear)
- **Resolution: Distance:** Varies (analog output)
- **Operating Voltage:** Typically 4.5V to 5.5V DC
- **Analog Output:** Voltage proportional to distance.
- **Response Time:** Varies (typically tens of milliseconds).

**2. Working Principle:**

- **Emission:** An infrared LED emitsinfrared light. Reflection: The emitted light reflects off objects**.**
- **Detection:** A position-sensitive detector (PSD) detectsthe reflected light.
- **Voltage Output:** The sensor outputs an analog voltage, which is inversely proportional to the distance.

- **Non-Linearity:** The relationship between voltage and distance is non-linear.

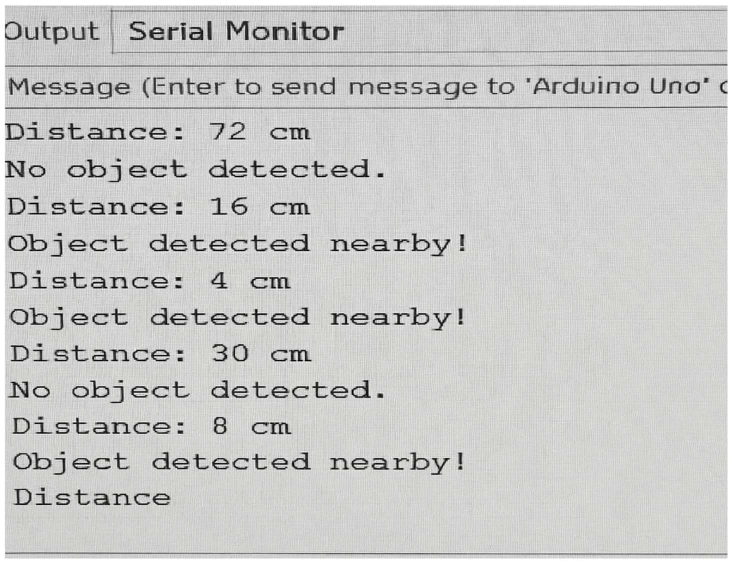
**3. Interfacing with Arduino:**

- **Wiring:** VCC: Connect to the Arduino's 5V pin.
- **GND:** Connect to the Arduino's GND pin.
- **Output**: Connect to an analog input pin on the Arduino.
- **Communication:** The sensor provides an analog voltage. Arduino's `analogRead()` function reads this voltage. Calibration is required for distance conversion. **4. Applications:**
- Robotics obstacle detection.
- Object proximity sensing.
- Automated navigation.
- Gesture recognition.
- Liquid level detection.

**CODE FOR IR SENSOR**

```
#define IR_SENSOR_PIN 2
#define LED_PIN 13

void setup() {
 pinMode(IR_SENSOR_PIN,
 INPUT); pinMode(LED_PIN,
 OUTPUT);
 Serial.begin(9600);
}

void loop() {
 int sensorValue = digitalRead(IR_SENSOR_PIN);

 if (sensorValue == HIGH) {
 // No object detected
 digitalWrite(LED_PIN, LOW); // Turn OFF LED
 Serial.println("No Object Detected");
} else {
// Object detected
digitalWrite(LED_PIN, HIGH); // Turn ON LED
Serial.println("Object Detected");
}

delay(500); // Delay to avoid flooding the Serial Monitor
}
```

**OUTPUT:-**



```
Distance: 72 cm
No object detected.
Distance: 16 cm
Object detected nearby!
Distance: 4 cm
Object detected nearby!
Distance: 30 cm
No object detected.
Distance: 8 cm
Object detected nearby!
Distance
```

**Fig 3.1**

**CONCLUSION:-**

In the IR sensor practical, we successfully demonstrated how infrared technology can be used for object detection and proximity sensing. By analyzing the reflected IR light from nearby objects, the sensor effectively identified their presence. This hands-on experience improved our understanding of how IR sensors work and their role in automation, robotics, and electronic projects. The practical reinforced key concepts like signal transmission, reflection, and digital output handling in real-world applications.