

# ST. ANNE'S

## COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)

Accredited by NAAC

ANGUCHETTYPALAYAM, PANRUTI – 607 106.



### CS3691 EMBEDDED SYSTEMS AND IOT

#### OBSERVATION NOTE

(FOR III B.E COMPUTER SCIENCE AND ENGINEERING STUDENTS)

NAME : \_\_\_\_\_

REGISTER NO : \_\_\_\_\_

YEAR/SEMESTER : III Year / VI Semester

PERIOD : FEB 2024 – MAY 2024

AS PER ANNA UNIVERSITY (CHENNAI) SYLLABUS

2021 REGULATION

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PREPARED BY: Mr. S. BALABASKER, AP/ECE

## **ABOUT OBSERVATION NOTES & PREPARATION OF RECORD**

- ❖ This Observation contains the basic diagrams of the circuits enlisted in the syllabus of the **CS3691 EMBEDDED SYSTEMS AND IOT** course, along with the design of various components of the circuit and controller.
- ❖ The aim of the experiment is also given at the beginning of each experiment. Once the student can design the circuit as per the circuit diagram, he/she is supposed to go through the instructions carefully and do the experiments step by step.
- ❖ They should note down the readings (observations) and tabulate them as specified.
- ❖ It is also expected that the students prepare the theory relevant to the experiment referring to prescribed reference books/journals in advance, and carry out the experiment after understanding thoroughly the concept and procedure of the experiment.
- ❖ They should get their observations verified and signed by the staff within two days and prepare & submit the record of the experiment when they come to the laboratory in the subsequent week.
- ❖ The record should contain experiment No., Date, Aim, Apparatus required, Theory, Procedure, and result on one side (i.e., Right-hand side, where rulings are provided) and Circuit diagram, Design, Model Graphs, Tabulations, and Calculations on the other side (i.e., Left-hand side, where no rulings are provided)
- ❖ All the diagrams and table lines should be drawn in pencil
- ❖ The students are directed to discuss & clarify their doubts with the staff members as and when required. They are also directed to follow strictly the guidelines specified.

# CS3691 EMBEDDED SYSTEMS AND IOT

## SYLLABUS

### COURSE OBJECTIVES:

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

### LIST OF EXPERIMENTS

1. Write 8051 Assembly Language experiments using simulator.
2. Test data transfer between registers and memory.
3. Perform ALU operations.
4. Write Basic and arithmetic Programs Using Embedded C.
5. Introduction to Arduino platform and programming
6. Explore different communication methods with IoT devices (Zigbee, GSM, Bluetooth)
7. Introduction to Raspberry PI platform and python programming
8. Interfacing sensors with Raspberry PI
9. Communicate between Arduino and Raspberry PI using any wireless medium
10. Setup a cloud platform to log the data
11. Log Data using Raspberry PI and upload to the cloud platform
12. Design an IOT based system

### OUTCOMES:

- CO1: Explain the architecture of embedded processors.
- CO2: Write embedded C programs.
- CO3: Design simple embedded applications.
- CO4: Compare the communication models in IOT
- CO5: Design IoT applications using Arduino/Raspberry Pi /open platform.







**PROGRAMS**





<b>EXP NO:</b>	<b>8051 Assembly Language program using Keil simulator</b>
<b>DATE</b>	

**AIM:**

To write 8051 Assembly Language Program for an 8-bit addition using Keil simulator and execute it.

**SOFTWARE REQUIRED:**

S.No	Software Requirements	Quantity
1	Keil $\mu$ vision5 IDE	1

**INTRODUCTION TO 8051 SIMULATORS:**

A simulator is software that will execute the program and show the results exactly to the program running on the hardware, if the programmer finds any errors in the program while simulating the program in the simulator, he can change the program and re-simulate the code and get the expected result, before going to the hardware testing. The programmer can confidently dump the program in the hardware when he simulates his program in the simulator and gets the expected results.

8051 controller is a most popular 8-bit controller which is used in a large number of embedded applications and many programmers write programs according to their application. So testing their programs in the software simulators is a way. Simulators will help the programmer to understand the errors easily and the time taken for the testing is also decreased.

These simulators are very useful for students because they do need not to build the complete hardware for testing their program and validate their program very easily in an interactive way.

**List of 8051 Simulators:**

The list of simulators is given below with their features:

**1. MCU 8051:** MCU 8051 is an 8051 simulator that is very simple to use and has an interactive IDE (Integrated Development Environment). It is developed by Martin Osmera and most important of all is that it is completely free. There are many features for this IDE they are

- ✓ It supports both C and assembly language for compilation and simulation

- ✓ It has an in-built source code editor, graphical notepad, ASCII charts, Assembly symbol viewer, etc. It also supports several 8051 ICs like at89c51, A89S52, 8051, 8052, etc.
- ✓ It will support certain electronic simulations like LED, 7segment display, LCD etc. which will help in giving the output when you interface these things to the hardware directly.
- ✓ It has tools like hex decimal editors, base converters, special calculator, file converters, inbuilt hardware programmers, etc.
- ✓ It has syntax validation, pop base auto-completion etc.

You can download this tool from <https://sourceforge.net/projects/mcu8051ide/files/>.

**2. EDSIM 51:** This is a virtual 8051 interfaced with virtual peripherals like 7 segment display, motor, keypad, UART etc. This simulator is exclusively for students developed by James Rogers,. The features of this simulator are

- ✓ Have virtual peripherals like ADC, DAC with scope to display, comparator etc.
- ✓ Supports only assembly language
- ✓ IDE is completely written in JAVA and supports all the OS.
- ✓ Completely free and with user guide, examples, etc.

You can download this simulator from the <https://www.edsim51.com/index.html>.

**3. 8051 IDE:** This simulation software is exclusively for the Windows operating system (98/xp). The features of this simulator are

- ✓ Text editor, assembler, and software simulate in one single program.
- ✓ Has facilities like Breakpoint setter, execute to break point, predefined simulator watch window, etc.
- ✓ It is available in both free version and paid version.

You can download this tool from <https://www.acebus.com/win8051.htm>

**4. KEIL  $\mu$ Vision:** KEIL is the most popular software simulator. It has many features like interactive IDE and supports both C and assembly languages for compilation and simulation.

You can download and get more information from <https://www.keil.com/c51/>.

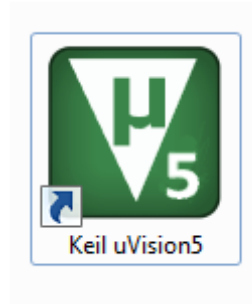
## INSTALLATION OF KEIL SOFTWARE

## Set up Keil IDE for Programming

Keil  $\mu$ Vision IDE is a popular way to program MCUs containing the 8051 architectures. It supports over 50 microcontrollers and has good debugging tools including logic analyzers and watch windows.

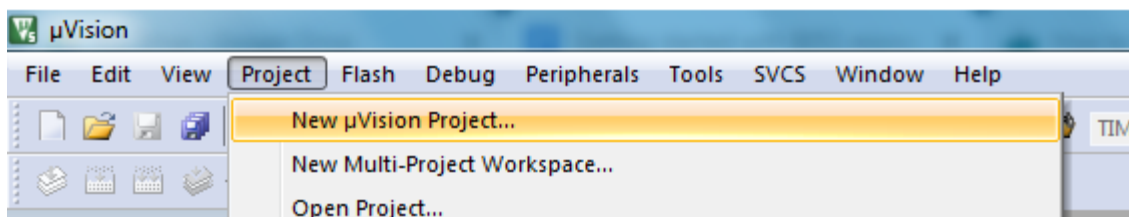
In this article, we will use the AT89C51ED2 microcontroller, which has:

- 64 KB FLASH ROM
- On-chip EEPROM
- 256 Bytes RAM
- In-System programming for uploading the program
- 3 Timer/counters
- SPI, UART, PWM



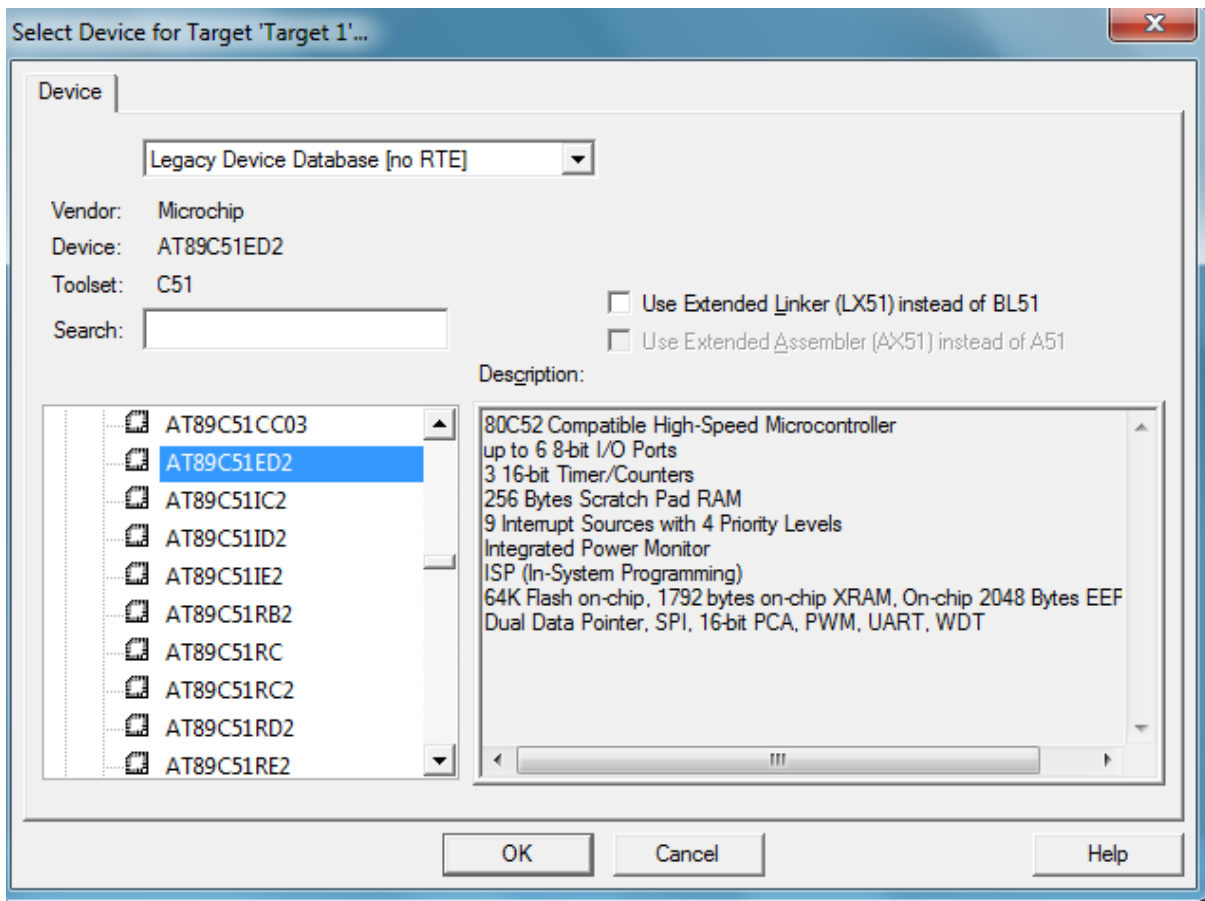
The Keil  $\mu$ Vision icon.

To start writing a new program, you need to create a new project. Navigate to **project**  $\rightarrow$  **New  $\mu$ Vision project**. Then save the new project in a folder.

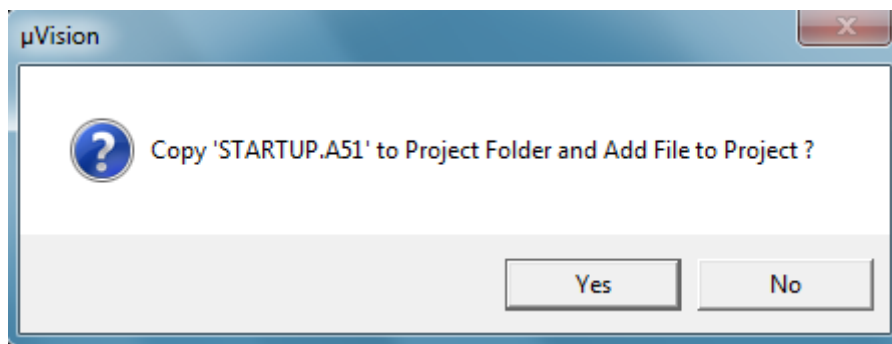


After saving the file, a new window will pop up asking you to select your microcontroller.

As discussed, we are using AT89C51/AT89C51ED2/AT89C52, so select this controller under the Microchip section (as Atmel is now a part of Microchip).

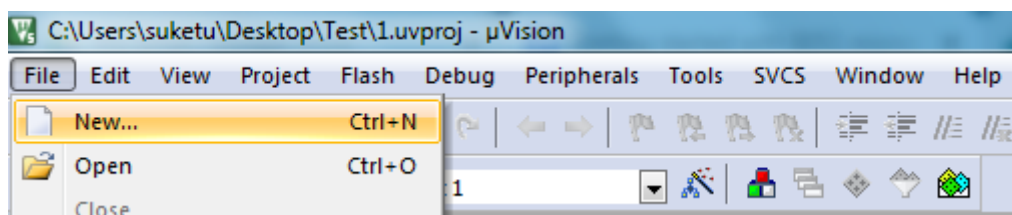


Select 'Yes' in the next pop-up, as we do not need this file in our project.

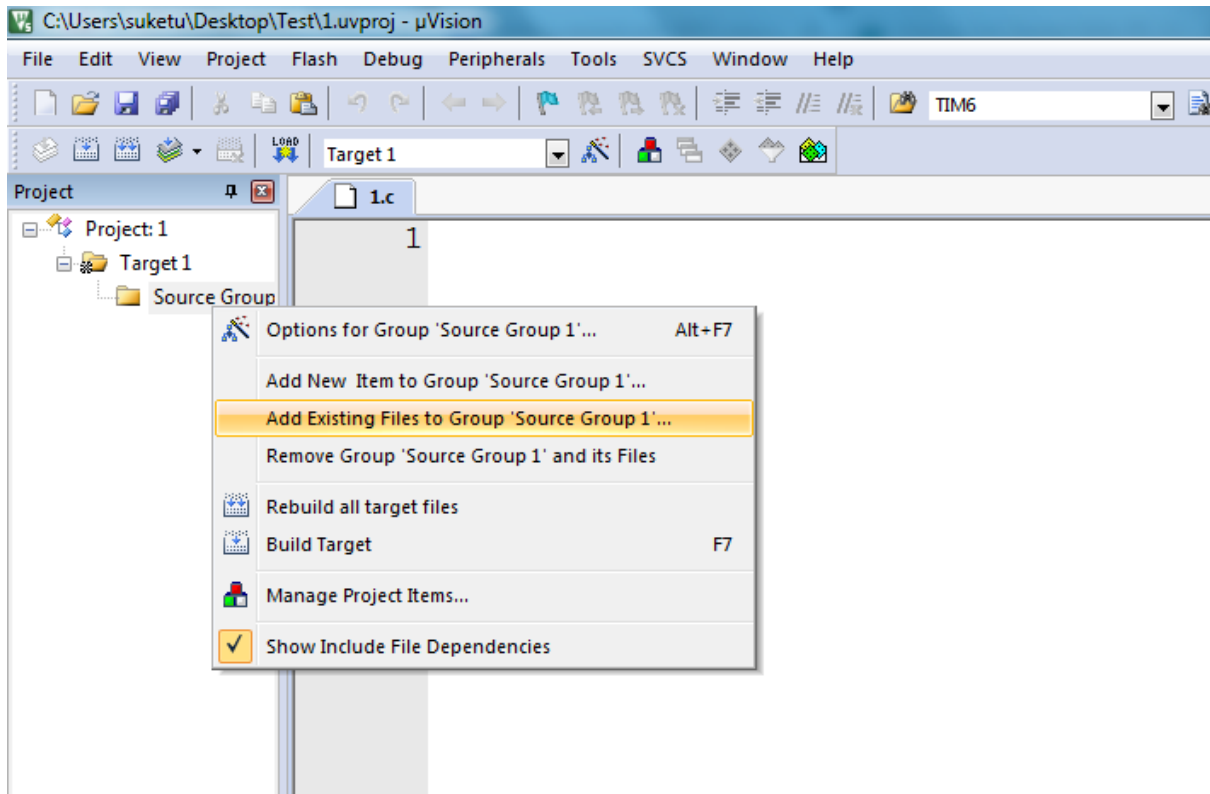


Our project workspace is now ready!

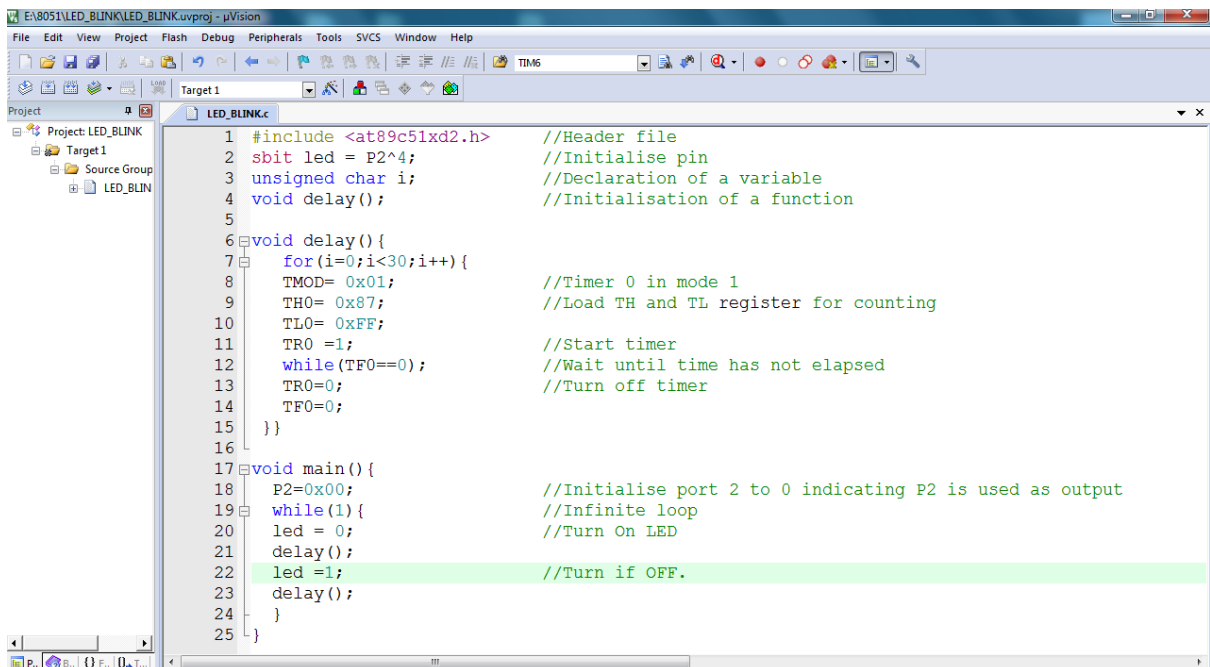
From here, we need to create a file where we can write our C code. Navigate to **File** → **New**. Once the file is created, save it with .c extension in the same project folder.



Next, we have to add that .c or .asm file to our project workspace. Select **Add Existing** Files and then select the created .c or .asm file to get it added.



The workspace and project file are ready.



## **PROCEDURE**

1. Create a new project, go to “Project” and close the current project “Close Project”.
2. Next Go to the Project New  $\mu$ Vision Project and Create New Project Select Device for Target.
3. Select the device AT89C51ED2 or AT89C51 or AT89C52
4. Add Startup file Next go to “File” and click “New”.
5. Write a program on the editor window and save it with .asm extension.
6. Add this source file to Group and click on “Build Target” or F7.
7. Go to debugging mode to see the result of simulation by clicking Run or step run.8.

**PROGRAM:**

```
ORG 0000H  
CLR C  
MOV A, #20H  
ADD A, #21H  
MOV R0, A  
END
```

**OUTPUT:**

**RESULT:**



<b>EXP NO:</b>	<b>Test data transfer between registers and memory</b>
<b>DATE</b>	

**AIM:**

To write and execute an Assembly language program to transfer data between registers and memory.

**SOFTWARE REQUIRED:**

<b>S.No</b>	<b>Software Requirements</b>	<b>Quantity</b>
1	Keil $\mu$ vision5 IDE	1

**PROCEDURE**

1. Create a new project, go to "Project" and close the current project "Close Project".
2. Next Go to the Project New  $\mu$ Vision Project and Create a New Project Select Device for the Target.
3. Select the device AT89C51ED2 or AT89C51 or AT89C52
4. Add Startup file Next go to "File" and click "New".
5. Write a program on the editor window and save it with .asm extension.
6. Add this source file to Group and click on "Build Target" or F7.
7. Go to debugging mode to see the result of the simulation by clicking Run or Step run.

**PROGRAM:**

**TYPE-I:**

```
ORG 0000H  
CLR C  
MOV R0, #55H  
MOV R1, #6FH  
MOV A, R0  
MOV 30H, A  
MOV A, R1  
MOV 31H, A  
END
```

**OUTPUT:**

**PROGRAM:**

**TYPE-II:**

```
    ORG 0000H
    CLR C
    MOV R0, #30H
    MOV R1, #40H
    MOV R7, #06H
BACK: MOV A, @R0
    MOV @R1, A
    INC R0
    INC R1
    DJNZ R7, BACK
    END
```

**OUTPUT:**

**RESULT:**

<b>EXP NO:</b>	<b>ALU operations</b>
<b>DATE</b>	

**AIM:**

To write and execute the ALU program using the Keil simulator.

**SOFTWARE TOOLS REQUIRED:**

S.No	Software Requirements	Quantity
1	Keil $\mu$ vision5 IDE	1

**PROCEDURE**

1. Create a new project, go to "Project" and close the current project "Close Project".
2. Next Go to the Project New  $\mu$ Vision Project and Create New Project Select Device for Target.
3. Select the device AT89C51ED2 or AT89C51 or AT89C52
4. Add Startup file Next go to "File" and click "New".
5. Write a program on the editor window and save it with .asm extension.
6. Add this source file to Group and click on "Build Target" or F7.
7. Go to debugging mode to see the result of simulation by clicking Run or step run.

**PROGRAM:**

```
ORG 0000H
CLR C
//ADDITION
MOV A, #20H
ADD A, #21H
MOV 41H, A
```

```
//SUBTRACTION
MOV A, #20H
SUBB A, #18H
MOV 42H, A
```

```
//MULTIPLICATION
MOV A, #03H
MOV B, #04H
MUL AB
MOV 43H, A
```

```
//DIVISION
MOV A, #95H
MOV B, #10H
DIV AB
MOV 44H, A
MOV 45H, B
```

```
//AND
MOV A, #25H
MOV B, #12H
ANL A, B
MOV 46H, A
```

```
//OR
MOV A, #25H
MOV B, #15H
ORL A, B
MOV 47H, A
```

```
//XOR
MOV A, #45H
MOV B, #67H
```

```
XRL A, B  
MOV 48H, A
```

```
//NOT  
MOV A, #45H  
CPL A  
MOV 49H, A  
END
```

**OUTPUT:**

**RESULT:**



<b>EXP NO:</b>	<b>WRITE BASIC PROGRAMS USING EMBEDDED C</b>
<b>DATE</b>	

**AIM:**

To write a basic embedded C program to control a port 0 pin 0 connected to an 8051 microcontroller using a Keil simulator.

**SOFTWARE REQUIRED:**

S.No	Software Requirements	Quantity
1	Keil $\mu$ vision5 IDE	1

**PROCEDURE**

1. Create a new project, go to “Project” and close the current project “Close Project”.
2. Next Go to the Project New  $\mu$ Vision Project and Create a New Project Select Device for the Target.
3. Select the device AT89C51ED2 or AT89C51 or AT89C52
4. Add Startup file Next go to “File” and click “New”.
5. Write a program on the editor window and save it with .asm extension.
6. Add this source file to Group and click on “Build Target” or F7.
7. Go to debugging mode to see the result of the simulation by clicking Run or Step run.



**PROGRAM:**

```
#include<REG51.h>
int i,j;
sbit LED = P2^0;
void main()
{
  while(1)
  {
    LED = 0;
    for(j=0;j<10000;j++);
    LED = 1;
    for(j=0;j<10000;j++);
  }
}
```

**OUTPUT:**

**RESULT:**

<b>EXP NO:</b>	<b>ARITHMETIC PROGRAMS USING EMBEDDED C</b>
<b>DATE</b>	

**AIM:**

To write an embedded C program for addition, subtraction, multiplication, and division using the Keil simulator.

**SOFTWARE REQUIRED:**

S.No	Software Requirements	Quantity
1	Keil $\mu$ vision5 IDE	1

**PROCEDURE**

1. Create a new project, go to "Project" and close the current project "Close Project".
2. Next Go to the Project New  $\mu$ vision Project and Create New Project Select Device for Target.
3. Select the device AT89C51ED2 or AT89C51 or AT89C52
4. Add Startup file Next go to "File" and click "New".
5. Write a program on the editor window and save it with the .asm extension.
6. Add this source file to Group and click on "Build Target" or F7.
7. Go to debugging mode to see the result of the simulation by clicking Run or Step run.



**PROGRAM:**

```
#include<REG51.H>
unsigned char a, b;
void main()
{
    a=0x10;
    b=0x04;
    P0=a-b;
    P1=a+b;
    P2=a*b;
    P3=a/b;
while(1);
}
```

**OUTPUT:**

Port P0= \_\_\_\_\_

Port P1= \_\_\_\_\_

Port P2= \_\_\_\_\_

Port P3= \_\_\_\_\_

**RESULT:**



<b>EXP NO:</b>	<b>INTRODUCTION TO THE ARDUINO PLATFORM</b>
<b>DATE</b>	

**AIM:**

To study the basics of Arduino Uno board and Arduino IDE 2.0 software.

**Hardware & SOFTWARE TOOLS REQUIRED:**

S.No.	Hardware & Software Requirements	Quantity
1	Arduino IDE 2.0	1
2	Arduino Uno Board	1

**INTRODUCTION TO ARDUINO:**

Arduino is a project, open-source hardware, and software platform used to design and build electronic devices. It designs and manufactures microcontroller kits and single-board interfaces for building electronics projects. The Arduino boards were initially created to help students with the non-technical background. The designs of Arduino boards use a variety of controllers and microprocessors. Arduino is an easy-to-use open platform for creating electronic projects. Arduino boards play a vital role in creating different projects. It makes electronics accessible to non-engineers, hobbyists, etc.

The various components present on the Arduino boards are a Microcontroller, Digital Input/output pins, USB Interface and Connector, Analog Pins, reset buttons, Power buttons, LEDs, Crystal oscillators, and Voltage regulators. Some components may differ depending on the type of board. The most standard and popular board used over time is Arduino UNO. The ATmega328 Microcontroller present on the UNO board makes it rather powerful than other boards. There are various types of Arduino boards used for different purposes and projects. The Arduino Boards are organized using the Arduino (IDE), which can run on various platforms. Here, IDE stands for Integrated Development Environment. Let's discuss some common and best Arduino boards.

## TYPES OF ARDUINO BOARDS

### 1) Arduino UNO

Arduino UNO is based on an ATmega328P microcontroller. It is easy to use compared to other boards, such as the Arduino Mega board, etc. The Arduino UNO includes 6 analog pin inputs, 14 digital pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. It is the most used and of standard form from the list of all available Arduino Boards.



### 2) Arduino Nano

The Arduino Nano is a small Arduino board based on ATmega328P or ATmega628 Microcontroller. The connectivity is the same as the Arduino UNO board. The Nano board is defined as a sustainable, small, consistent, and flexible microcontroller board. It is small in size compared to the UNO board. The devices required to start our projects using the Arduino Nano board are Arduino IDE and mini-USB. The Arduino Nano includes an I/O pin set of 14 digital pins and 8 analog pins. It also includes 6 Power pins and 2 Reset pins.



### 3) Arduino Mega

The Arduino Mega is based on the ATmega2560 Microcontroller. The ATmega2560 is an 8-bit microcontroller. We need a simple USB cable to connect to the computer and the AC to DC adapter or battery to get started with it. It has the advantage of working with more memory space. The Arduino Mega includes 54 I/O digital pins and 16 Analog Input/Output (I/O), ICSP header, a reset

button, 4 UART (Universal Asynchronous Receiver/Transmitter) ports, USB connection, and a power jack.



#### 4) Arduino Micro

The Arduino Micro is based on the ATmega32U4 Microcontroller. It consists of 20 sets of pins. The 7 pins from the set are PWM (Pulse Width Modulation) pins, while 12 pins are analog input pins. The other components on board are a reset button, a 16MHz crystal oscillator, an ICSP header, and a micro-USB connection. The USB is built in the Arduino Micro board.



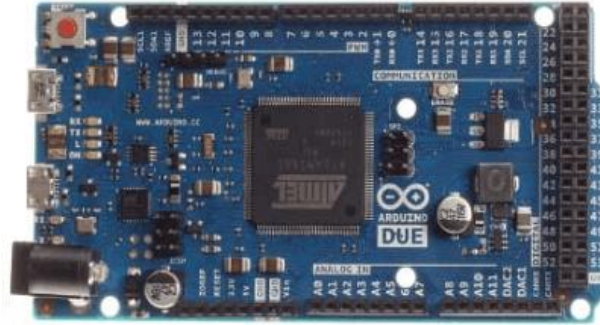
#### 5) Arduino Leonardo

The basic specification of the Arduino Leonardo is the same as the Arduino Micro. It is also based on the ATmega32U4 Microcontroller. The components present on the board are 20 analog and digital pins, a reset button, a 16MHz crystal oscillator, an ICSP header, and a micro USB connection.



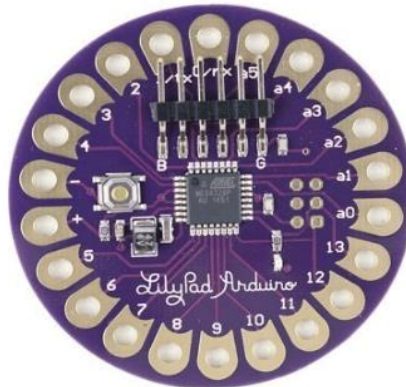
#### 6) Arduino Due

The Arduino Due is based on the 32-bit ARM core. It is the first Arduino board that has been developed based on the ARM Microcontroller. It consists of 54 Digital Input/Output pins and 12 Analog pins. The Microcontroller present on the board is the Atmel SAM3X8E ARM Cortex-M3 CPU. It has two ports, namely, a native USB port and a Programming port. The micro side of the USB cable should be attached to the programming port.



### 7) Arduino Lilypad

The Arduino Lilypad was initially created for wearable projects and e-textiles. It is based on the ATmega168 Microcontroller. The functionality of Lilypad is the same as other Arduino Boards. It is a round, lightweight board with a minimal number of components to keep the size of the board small. The Arduino Lilypad board was designed by Sparkfun and Leah. It was developed by Leah Buechley. It has 9 digital I/O pins.



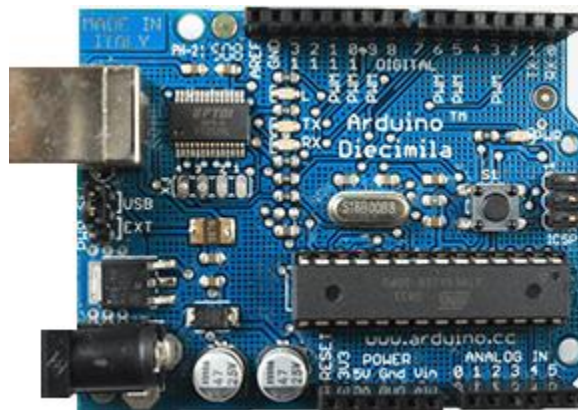
### 8) Arduino Bluetooth

The Arduino Bluetooth board is based on ATmega168 Microcontroller. It is also named as **Arduino BT board**. The components present on the board are 16 digital pins, 6 analog pins, reset button, 16MHz crystal oscillator, ICSP header, and screw terminals. The screw terminals are used for power. The Arduino Bluetooth Microcontroller board can be programmed over the Bluetooth as a wireless connection.



### 9) Arduino Diecimila

The Arduino Diecimila is also based on the ATmeg628 Microcontroller. The board consists of 6 analog pin inputs, 14 digital Input/Output pins, a USB connector, a power jack, an ICSP (In-Circuit Serial Programming) header, and a reset button. We can connect the board to the computer using the USB and can power on the board with the help of an AC to DC adapter. The Diecimila was initially developed to mark the 10000 delivered boards of Arduino. Here, Diecimila means 10,000 in Italian.



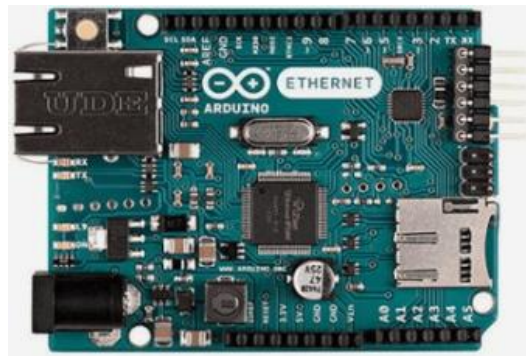
### 10) Arduino Robot

The Arduino Robot is called as the tiny computer. It is widely used in robotics. The board comprises of the speaker, five-button, color screen, two motors, an SD card reader, a digital compass, two potentiometers, and five floor sensors. The Robot Library can be used to control the actuators and the sensors.



### 11) Arduino Ethernet

The Arduino Ethernet is based on the ATmega328 Microcontroller. The board consists of 6 analog pins, 14 digital I/O pins, crystal oscillator, reset button, ICSP header, a power jack, and an RJ45 connection. With the help of the Ethernet shield, we can connect our Arduino board to the internet.



### 12) Arduino Zero

The Arduino Zero is generally called as the 32-bit extension of the Arduino UNO. It is based on ATmel's SAM21 MCU. The board consists of 6 analog pin inputs, 14 digital Input/Output pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header, UART port pins, a power header, and AREF button. The Embedded debugger of Atmel is also supported by the Arduino Zero. The function of Debugger is to provide a full debug interface, which does not require additional hardware.



### 13) Arduino Esplora

The Arduino Esplora boards allow easy interfacing of sensors and actuators. The outputs and inputs connected on the Esplora board make it unique from other types of Arduino boards. The board includes outputs, inputs, a small microcontroller, a microphone, a sensor, a joystick, an accelerometer, a temperature sensor, four buttons, and a slider.



### 14) Arduino Pro Micro

The structure of Arduino Pro Micro is similar to the Arduino Mini board, except the Microcontroller ATmega32U4. The board consists of 12 digital Input/output pins, 5 PWM (Pulse Width Modulation) pins, Tx and Rx serial connections, and 10-bit ADC (Analog to Digital Converter).



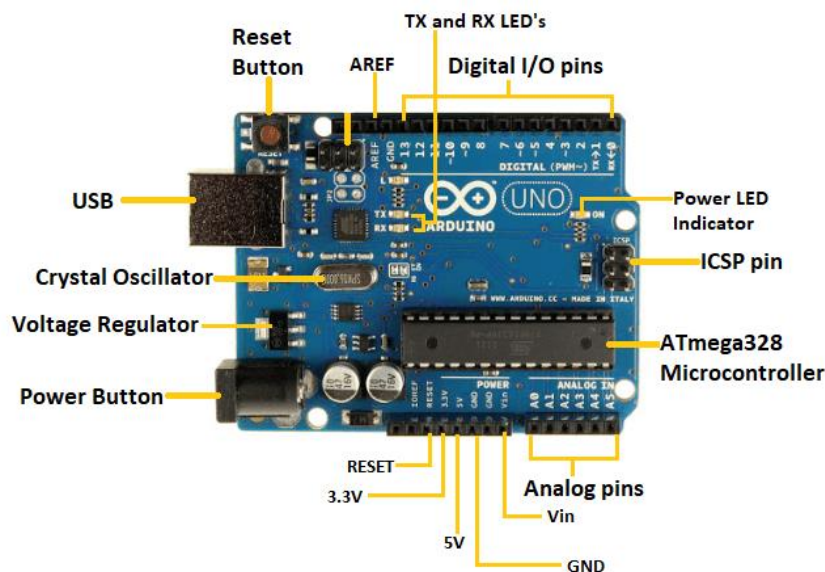
## INTRODUCTION TO ARDUINO UNO:

The Arduino UNO is a standard board of Arduino. Here UNO means 'one' in Italian. It was named UNO to label the first release of Arduino Software. It was also the first USB board released by Arduino. It is considered a powerful board used in various projects. Arduino. cc developed the Arduino UNO board. Arduino UNO is based on an ATmega328P microcontroller. It is easy to use compared to other boards, such as the Arduino Mega board, etc. The board consists of digital and analog Input/Output pins (I/O), shields, and other circuits. The Arduino UNO includes 6 analog pin inputs, 14 digital pins, a USB connector, a power jack, and an ICSP (In-Circuit Serial Programming) header. It is programmed based on IDE, which stands for Integrated Development Environment. It can run on both online and offline platforms. The IDE is common to all available boards of Arduino.

The Arduino board is shown below:



The components of Arduino UNO board are shown below:



Let's discuss each component in detail.



- **ATmega328 Microcontroller**- It is a single-chip Microcontroller of the ATmel family. The processor code inside it is of 8-bit. It combines **Memory (SRAM, EEPROM, and Flash), Analog to Digital Converter, SPI serial ports, I/O lines, registers, timers, external and internal interrupts, and oscillator.**
- **ICSP pin** - The In-Circuit Serial Programming pin allows the user to program using the firmware of the Arduino board.
- **Power LED Indicator**- The ON status of the LED shows the power is activated. When the power is OFF, the LED will not light up.
- **Digital I/O pins**- The digital pins have the value HIGH or LOW. The pins numbered from D0 to D13 are digital pins.
- **TX and RX LED's**- The successful flow of data is represented by the lighting of these LED's.
- **AREF**- The Analog Reference (AREF) pin is used to feed a reference voltage to the Arduino UNO board from the external power supply.
- **Reset button**- It is used to add a Reset button to the connection.
- **USB**- It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.
- **Crystal Oscillator**- The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.
- **Voltage Regulator**- The voltage regulator converts the input voltage to 5V.
- **GND**- Ground pins. The ground pin acts as a pin with zero voltage.
- **Vin**- It is the input voltage.
- **Analog Pins**- The pins numbered from A0 to A5 are analog pins. The function of Analog pins is to read the analog sensor used in the connection. It can also act as GPIO (General Purpose Input Output) pin.

## TECHNICAL SPECIFICATIONS OF ARDUINO UNO

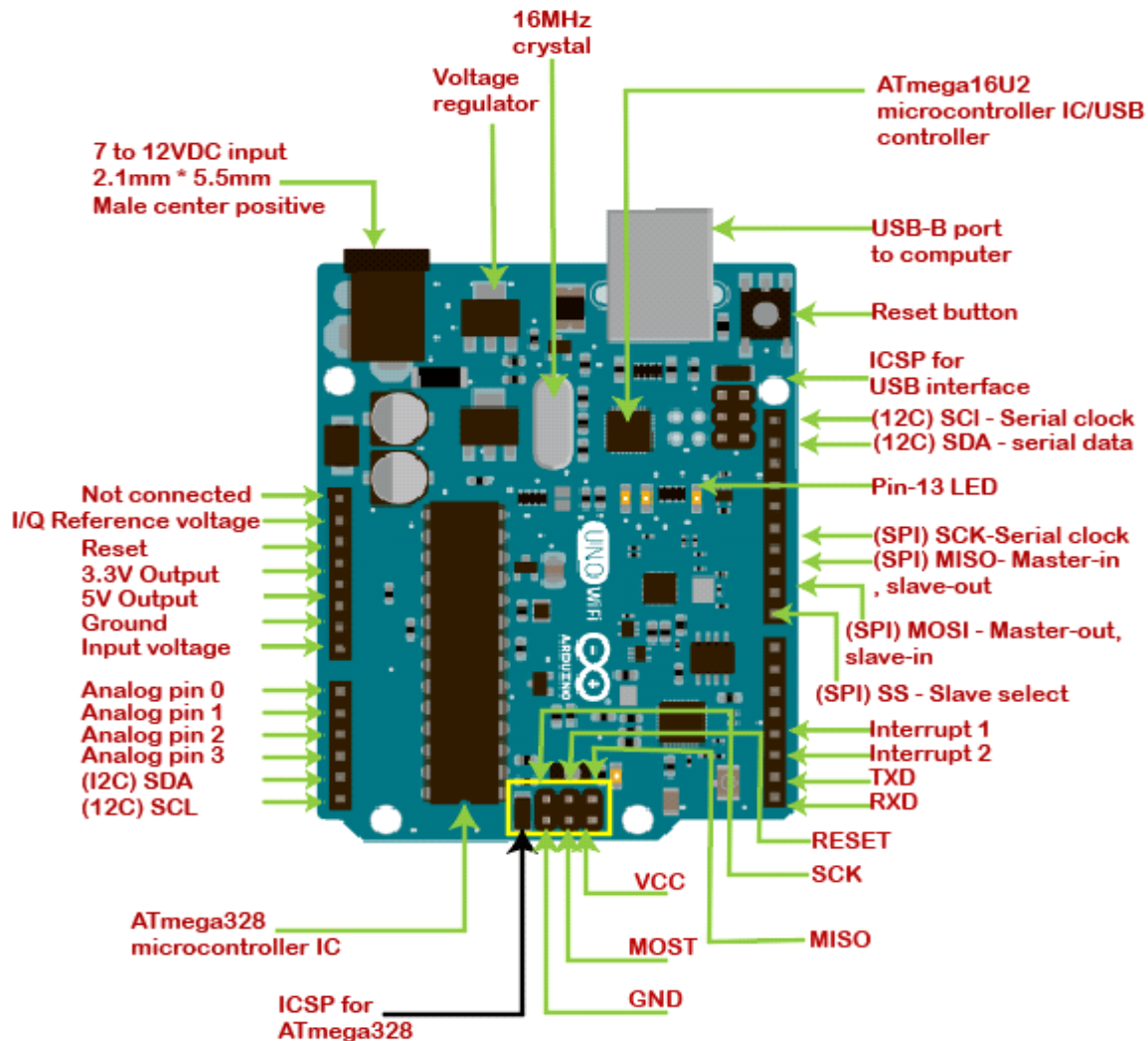
The technical specifications of the Arduino UNO are listed below:

- There are 20 Input/Output pins present on the Arduino UNO board. These 20 pins include 6 PWM pins, 6 analog pins, and 8 digital I/O pins.
- The PWM pins are Pulse Width Modulation capable.
- The crystal oscillator present in Arduino UNO comes with a frequency of 16MHz.
- It also has an Arduino-integrated WIFI module. Such Arduino UNO board is based on the Integrated WIFI ESP8266 Module and ATmega328P microcontroller.
- The input voltage of the UNO board varies from 7V to 20V.
- Arduino UNO automatically draws power from the external power supply. It can also draw power from the USB.

## ARDUINO UNO PINOUT

The Arduino UNO is a standard board of Arduino, which is based on an **ATmega328P** microcontroller. It is easier to use than other types of Arduino Boards.

The Arduino UNO Board, with the specification of pins, is shown below:



Let's discuss each pin in detail.

**ATmega328 Microcontroller**- It is a single chip Microcontroller of the ATmel family. The processor core inside it is of 8-bit. It is a low-cost, low powered, and a simple microcontroller. The Arduino UNO and Nano models are based on the ATmega328 Microcontroller.

**Voltage Regulator:** The voltage regulator converts the input voltage to 5V. The primary function of voltage regulator is to regulate the voltage level in the Arduino board. For any changes in the input voltage of the regulator, the output voltage is constant and steady.

**GND** - Ground pins. The ground pins are used to ground the circuit.

**TXD and RXD:** TXD and RXD pins are used for serial communication. The TXD is used for transmitting the data, and RXD is used for receiving the data. It also represents the successful flow of data.

**USB Interface:** The USB Interface is used to plug-in the USB cable. It allows the board to connect to the computer. It is essential for the programming of the Arduino UNO board.

**RESET:** It is used to add a Reset button to the connection.

**SCK:** It stands for **Serial Clock**. These are the clock pulses, which are used to synchronize the transmission of data.

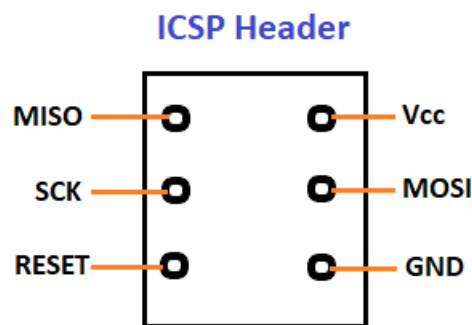
**MISO:** It stands for **Master Input/ Slave Output**. The save line in the MISO pin is used to send the data to the master.

**VCC:** It is the modulated DC supply voltage, which is used to regulate the IC's used in the connection. It is also called as the primary voltage for IC's present on the Arduino board. The Vcc voltage value can be negative or positive with respect to the GND pin.

**Crystal Oscillator-** The Crystal oscillator has a frequency of 16MHz, which makes the Arduino UNO a powerful board.

**ICSP:** It stands for **In-Circuit Serial Programming**. The users can program the Arduino board's firmware using the ICSP pins. The program or firmware with the advanced functionalities is received by microcontroller with the help of the ICSP header. The ICSP header consists of 6 pins.

The structure of the ICSP header is shown below:



**SDA:** It stands for **Serial Data**. It is a line used by the slave and master to send and receive data. It is called as a **data line**, while SCL is called as a clock line.

**SCL:** It stands for **Serial Clock**. It is defined as the line that carries the clock data. It is used to synchronize the transfer of data between the two devices. The Serial Clock is generated by the device and it is called as master.

**SPI:** It stands for **Serial Peripheral Interface**. It is popularly used by the microcontrollers to communicate with one or more peripheral devices quickly. It uses conductors for data receiving, data sending, synchronization, and device selection (for communication).

**MOSI:** It stands for Master Output/ Slave Input. The MOSI and SCK are driven by the Master.

**SS:** It stands for **Slave Select**. It is the Slave Select line, which is used by the master. It acts as the enable line.

**I<sup>2</sup>C:** It is the two-wire serial communication protocol. It stands for Inter Integrated Circuits. The I<sup>2</sup>C is a serial communication protocol that uses SCL (Serial Clock) and SDA (Serial Data) to receive and send data between two devices.

**3.3V and 5V** are the operating voltages of the board.

## **INTRODUCTION TO ARDUINO IDE 2.0:**

The Arduino IDE 2.0 is an open-source project, currently in its beta-phase. It is a big step from its sturdy predecessor, Arduino IDE 1.8.10, and comes with revamped UI, improved board & library manager, autocomplete feature and much more.

In this tutorial, we will go through step by step, how to download and install the software.

### **Download the editor**

Downloading the Arduino IDE 2.0 is done through the Arduino Software page. Here you will also find information on the other editors available to use.

### **Requirements**

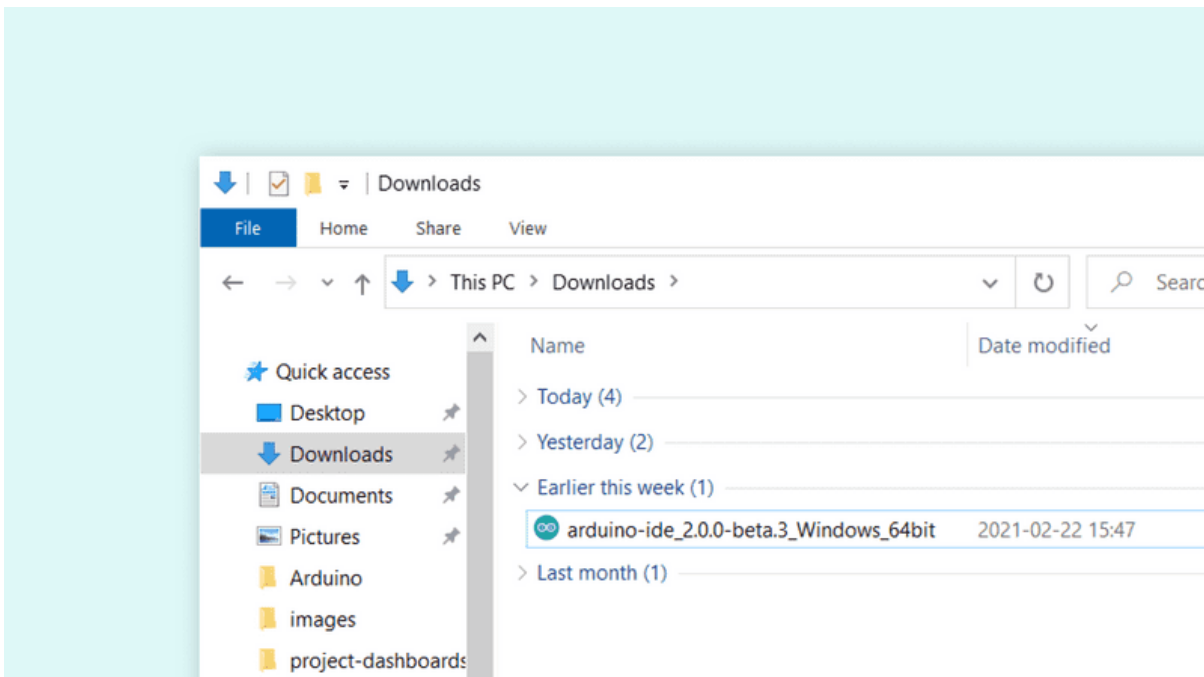
- **Windows** - Win 10 and newer, 64 bits
- **Linux** - 64 bits
- **Mac OS X** - Version 10.14: "Mojave" or newer, 64 bits

### **Installation**

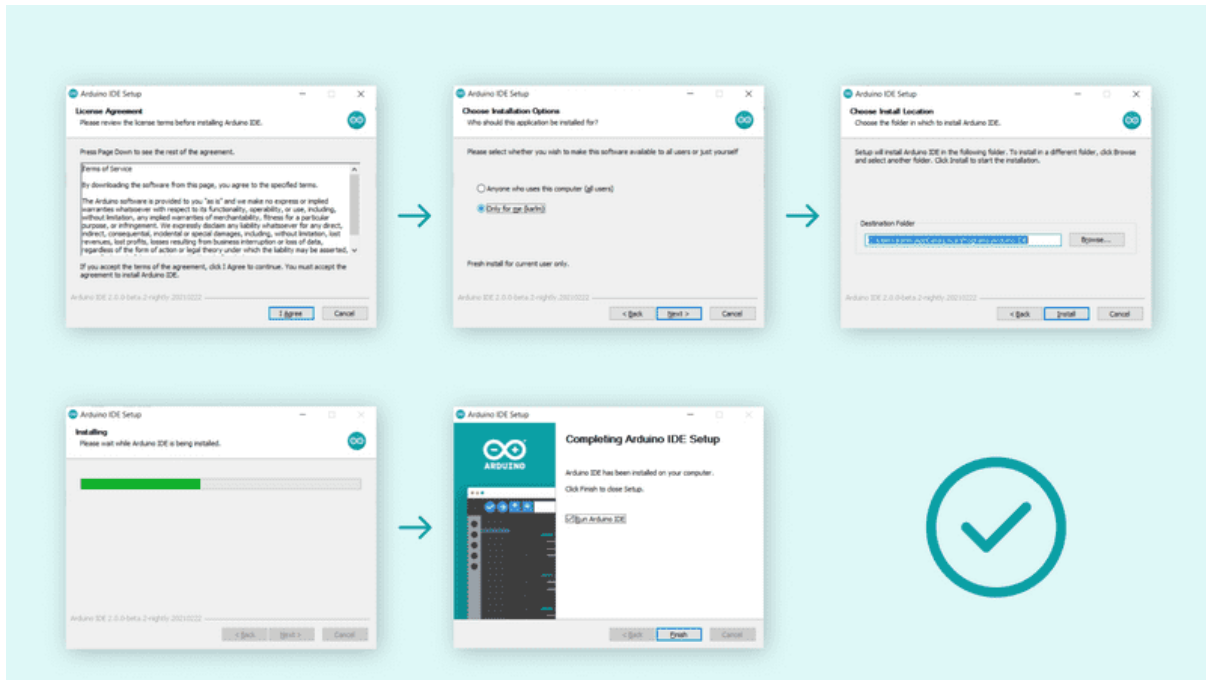
#### **Windows**

**Download URL:** <https://www.arduino.cc/en/software>

To install the Arduino IDE 2.0 on a Windows computer, simply run the file downloaded from the software page.



Follow the instructions in the installation guide. The installation may take several minutes.



You can now use the Arduino IDE 2.0 on your windows computer!

## How to use the board manager with the Arduino IDE 2.0

The board manager is a great tool for installing the necessary cores to use your Arduino boards. In this quick tutorial, we will take a look at how to install one, and choosing the right core for your board!

### Requirements

- Arduino IDE 2.0 installed.

### Why use the board manager?

The board manager is a tool that is used to install different cores on your local computer. So what is a **core**, and why is it necessary that I install one?

Simply explained, a core is written and designed for specific microcontrollers. As Arduino have several different types of boards, they also have different type of microcontrollers.

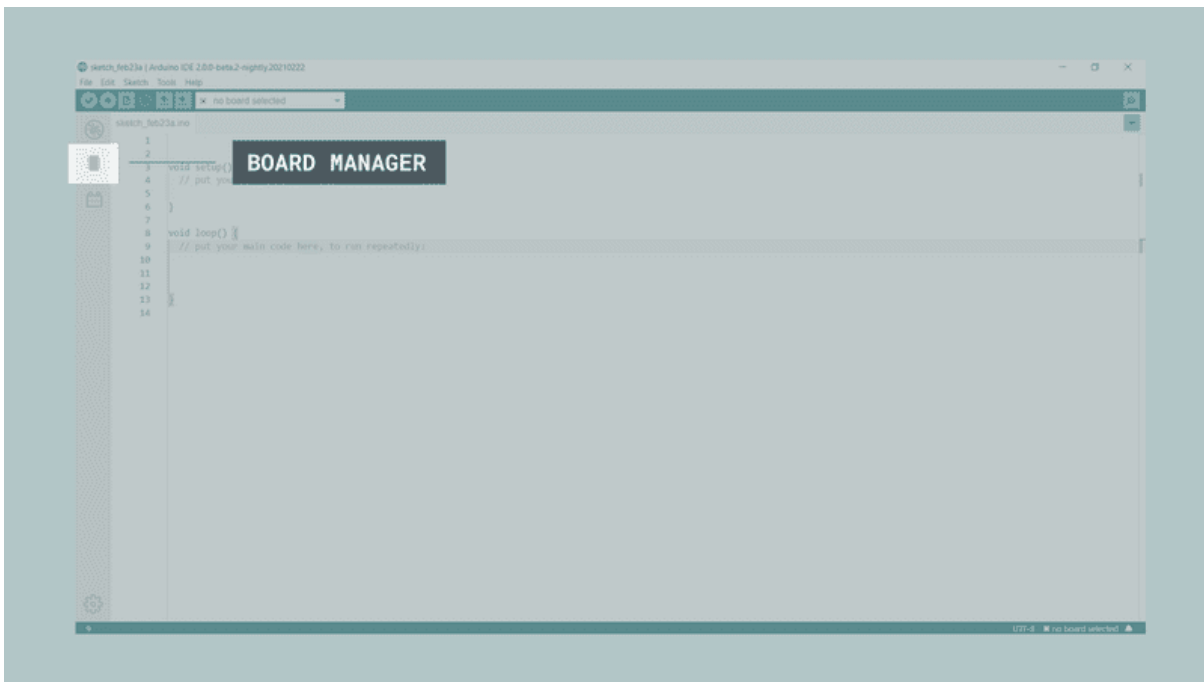
For example, an Arduino UNO has an **ATmega328P**, which uses the **AVR core**, while an Arduino Nano 33 IoT has a **SAMD21** microcontroller, where we need to use the **SAMD core**.

In conclusion, to use a specific board, we need to install a specific core.

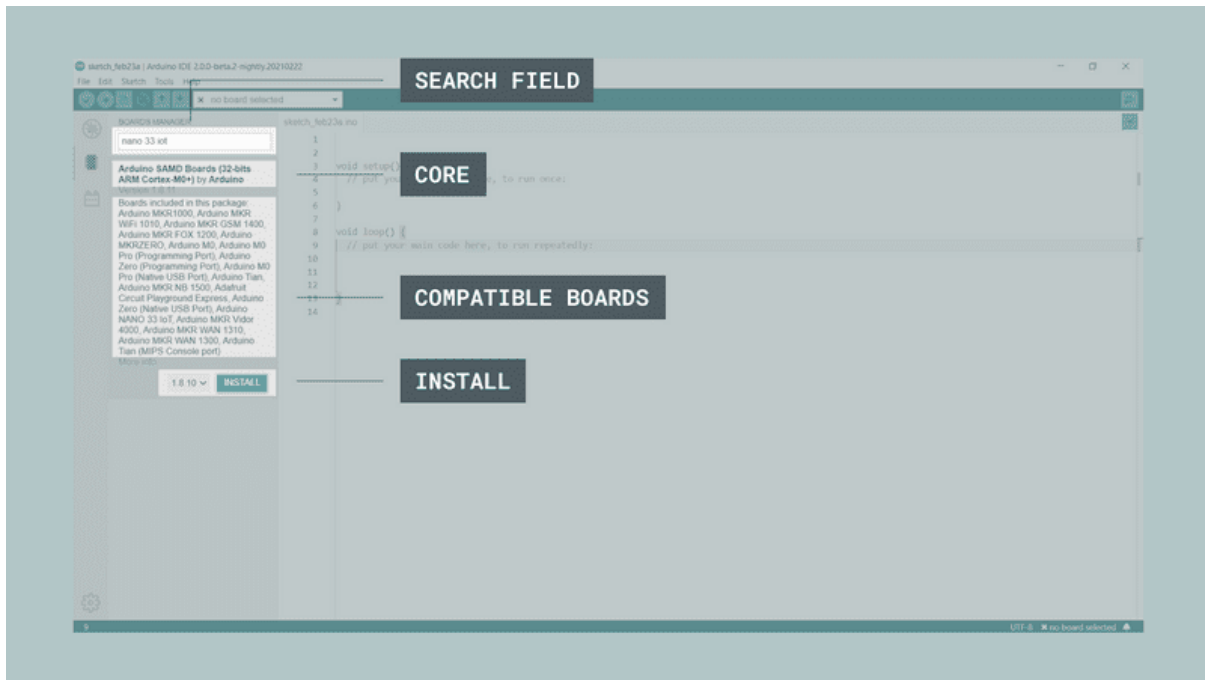
### Installing a core

Installing a core is quick and easy, but let's take a look at what we need to do.

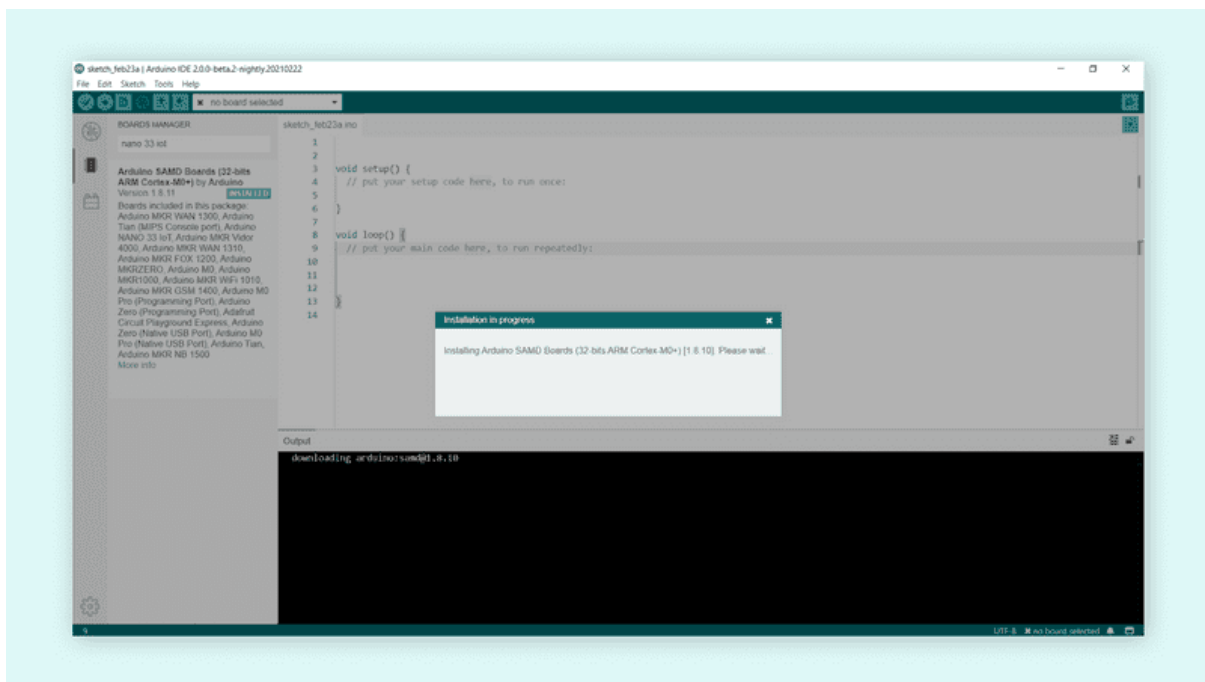
1. Open the Arduino IDE 2.0.
2. With the editor open, let's take a look at the left column. Here, we can see a couple of icons. Let's click the on the "**computer chip**" icon.



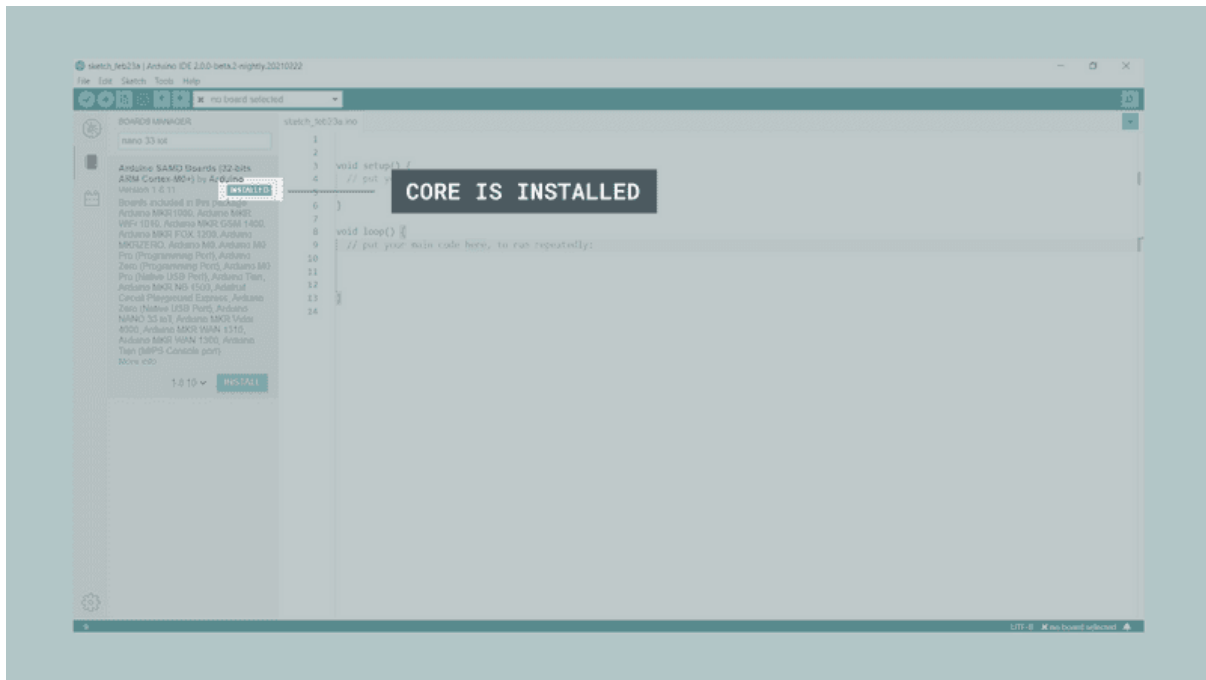
1. A list will now appear of all available cores. Now let's say we are using an **Nano 33 IoT** board, and we want to install the core. Simply enter the name in the search field, and the right core (SAMD) will appear, where the Nano 33 IoT features in the description. Click on the "**INSTALL**" button.



4. This will begin an installation process, which in some cases may take several minutes.



5. When it is finished, we can take a look at the core in the boards manager column, where it should say "INSTALLED".



You have now successfully downloaded and installed a core on your machine, and you can start using your Arduino board!



## How to upload a sketch with the Arduino IDE 2.0

In the Arduino environment, we write **sketches** that can be uploaded to Arduino boards. In this tutorial, we will go through how to select a board connected to your computer, and how to upload a sketch to that board, using the Arduino IDE 2.0.

### Requirements

- Arduino IDE 2.0 installed.

### Verify VS Upload

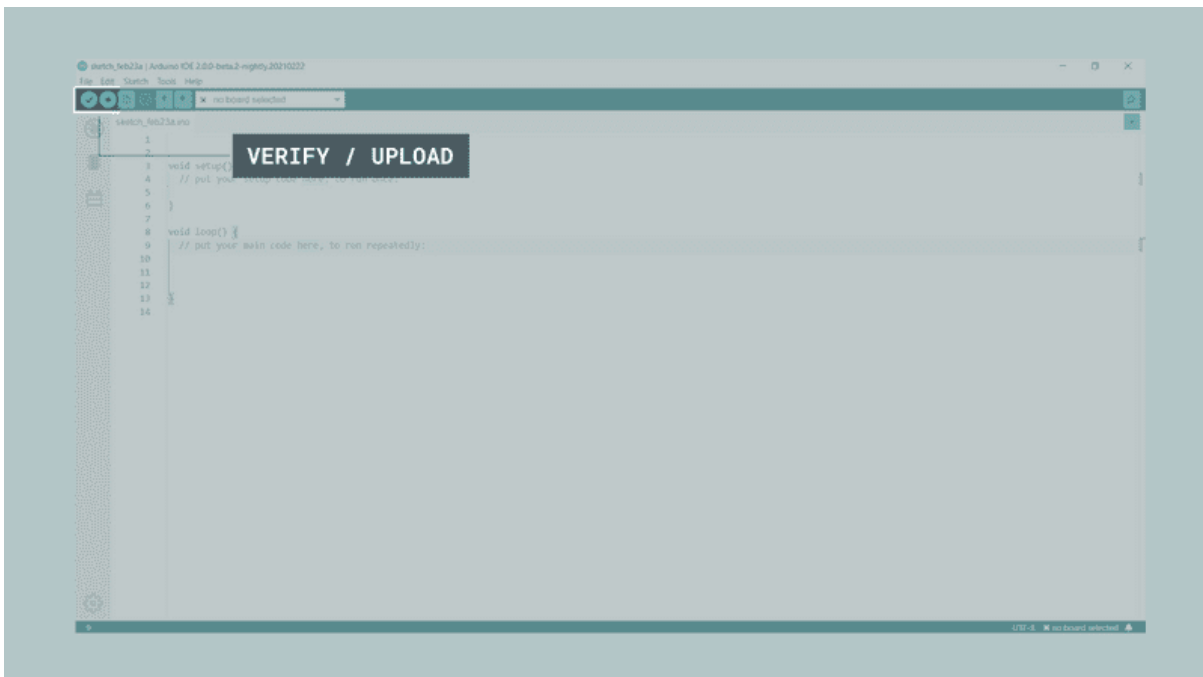
There are two main tools when uploading a sketch to a board: **verify** and **upload**. The verify tool simply goes through your sketch, checks for errors and compiles it. The upload tool does the same, but when it finishes compiling the code, it also uploads it to the board.

A good practice is to use the verifying tool before attempting to upload anything. This is a quick way of spotting any errors in your code, so you can fix them before actually uploading the code.

### Uploading a sketch

Installing a core is quick and easy, but let's take a look at what we need to do.

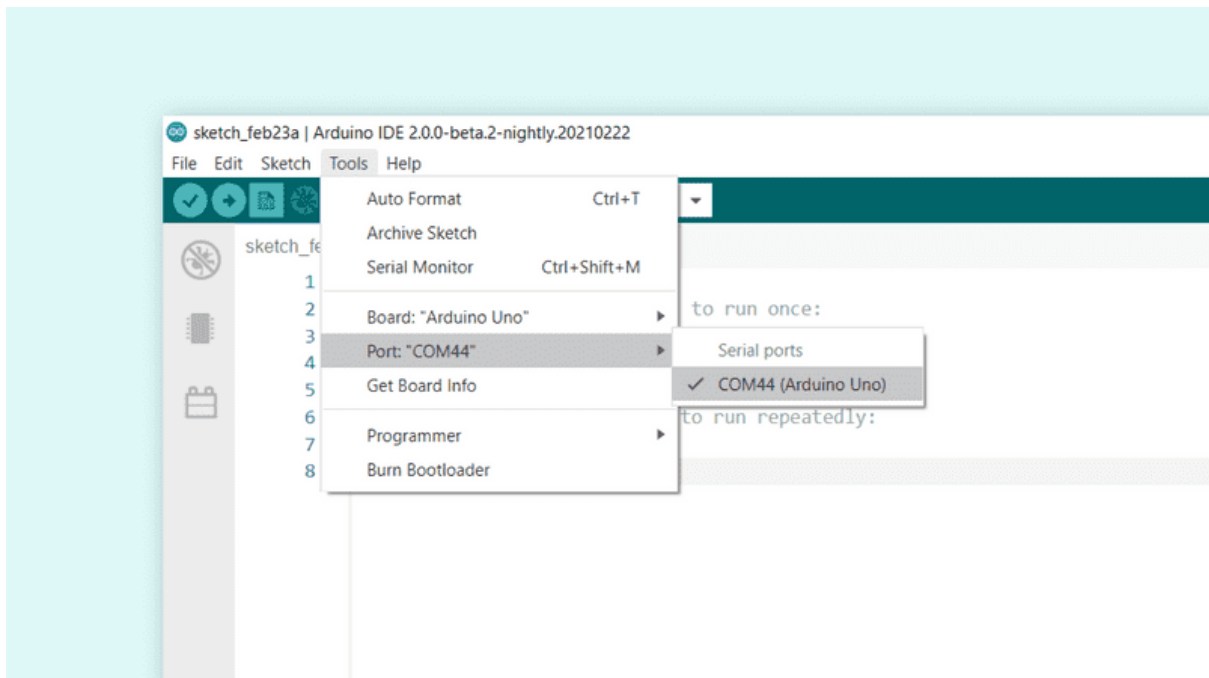
1. Open the Arduino IDE 2.0.
2. With the editor open, let's take a look at the navigation bar at the top. At the very left, there is a **checkmark** and an **arrow pointing right**. The checkmark is used to **verify**, and the arrow is used to **upload**.



3. Click on the verify tool (checkmark). Since we are verifying an empty sketch, we can be sure it is going to compile. After a few seconds, we can see the result of the action in the console (black box in the bottom).



1. Now we know that our code is compiled, and that it is working. Now, before we can upload the code to our board, we will first need to select the board that we are using. We can do this by navigating to **Tools > Port > {Board}**. The board(s) that are connected to your computer should appear here, and we need to select it by clicking it. In this case, our board is displayed as **COM44 (Arduino UNO)**.



5. With the board selected, we are good to go! Click on the **upload** button, and it will start uploading the sketch to the board.
6. When it is finished, it will notify you in the console log. Of course, sometimes there are some complications when uploading, and these errors will be listed here as well.



you have now uploaded a sketch to your Arduino board!

### **How to install and use a library with the Arduino IDE 2.0**

A large part of the Arduino programming experience is the **use of libraries**. Thousands of libraries can be found online, and the best-documented ones can be found and installed directly through the editor. In this tutorial, we will go through how to install a library using the library manager in the Arduino IDE 2.0. We will also show how to access examples from a library that you have installed.

#### **Requirements**

- Arduino IDE 2.0 installed.

#### **Why use libraries?**

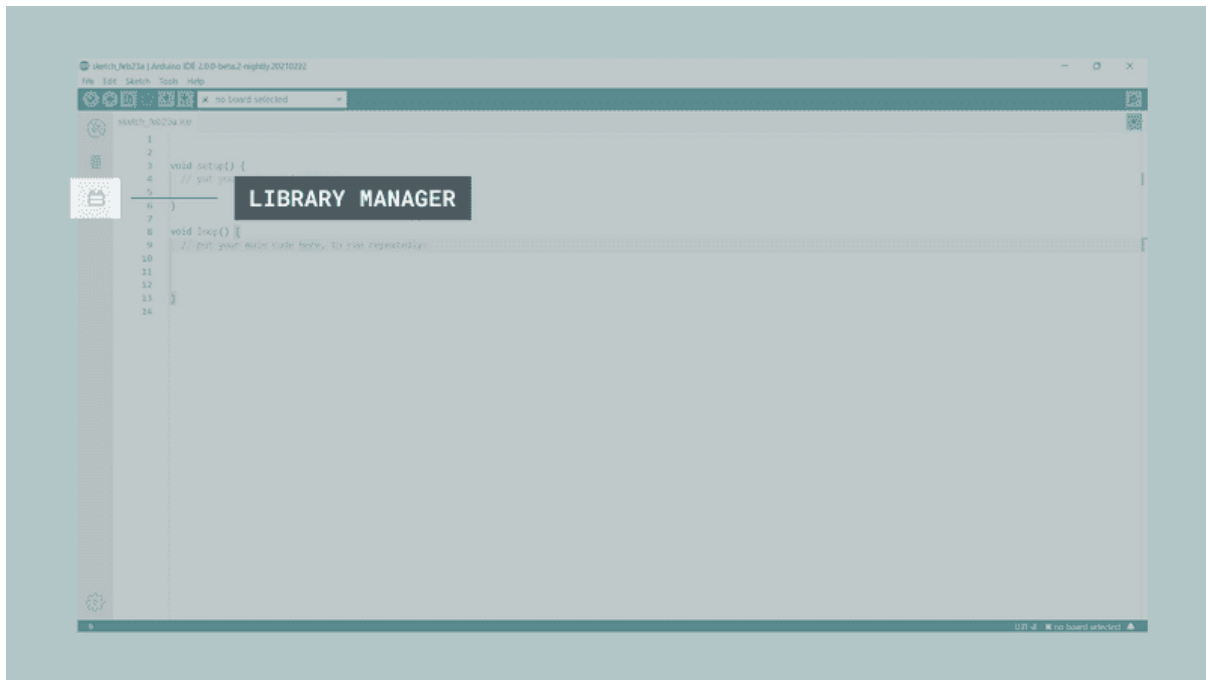
Libraries are incredibly useful when creating a project of any type. They make our development experience much smoother, and there almost an infinite amount out there. They are used to interface with many different sensors, RTCs, Wi-Fi modules, RGB matrices and of course with other components on your board.

Arduino has many official libraries, but the real heroes are the Arduino community, who develop, maintain and improve their libraries on a regular basis.

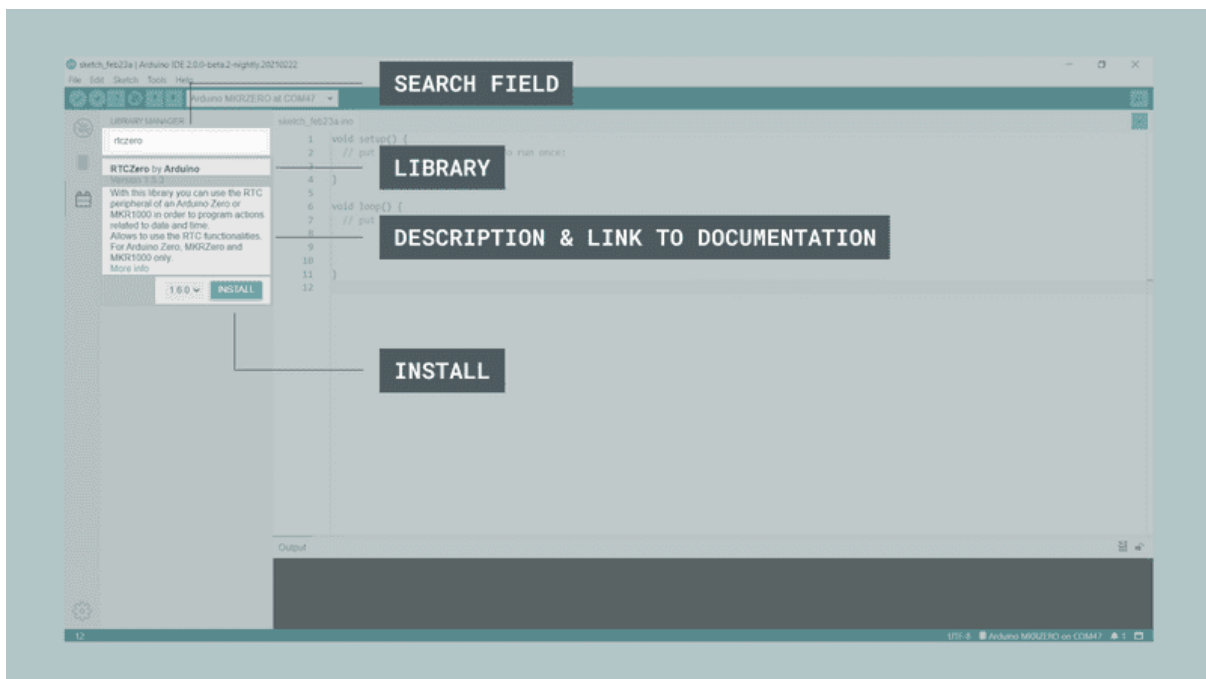
#### **Installing a library**

Installing a library is quick and easy, but let's take a look at what we need to do.

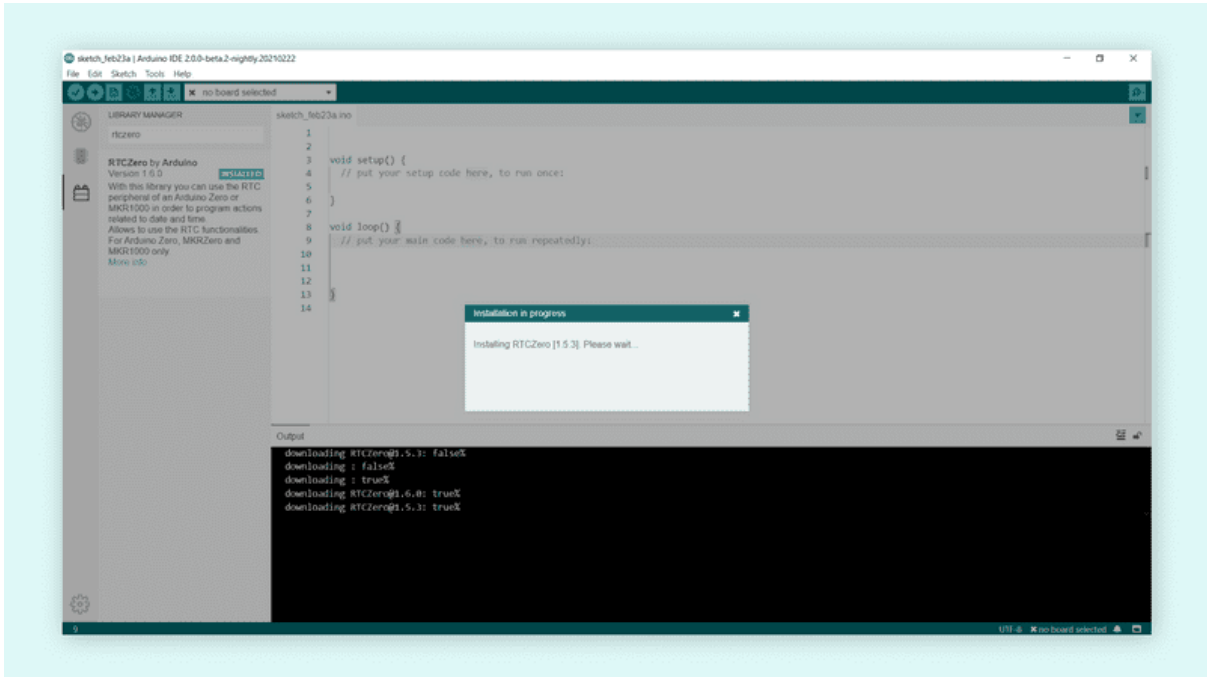
1. Open the Arduino IDE 2.0.
2. With the editor open, let's take a look at the left column. Here, we can see a couple of icons. Let's click the on the "**library**" icon.



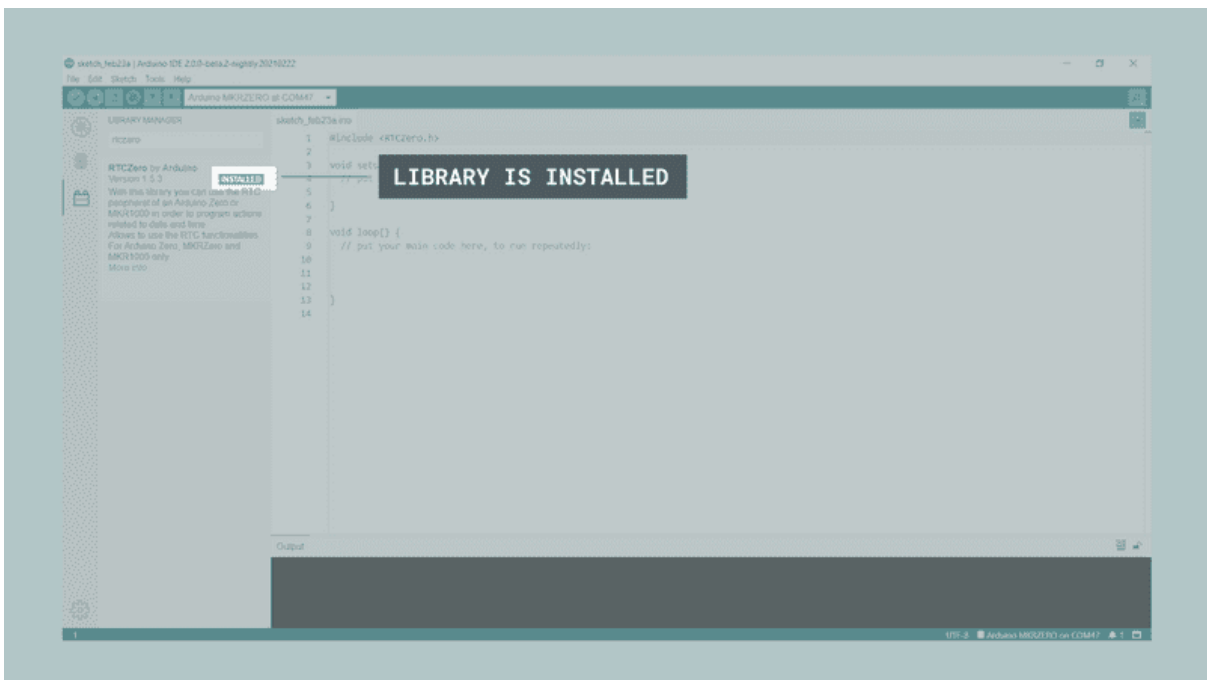
3. A list will now appear of all available libraries, where we can also search for the library we want to use. In this example, we are going to install the **RTCZero** library. Click on the **"INSTALL"** button to install the library.



4. This process should not take too long, but allow up to a minute to install it.



- When it is finished, we can take a look at the library in the library manager column, where it should say "INSTALLED".



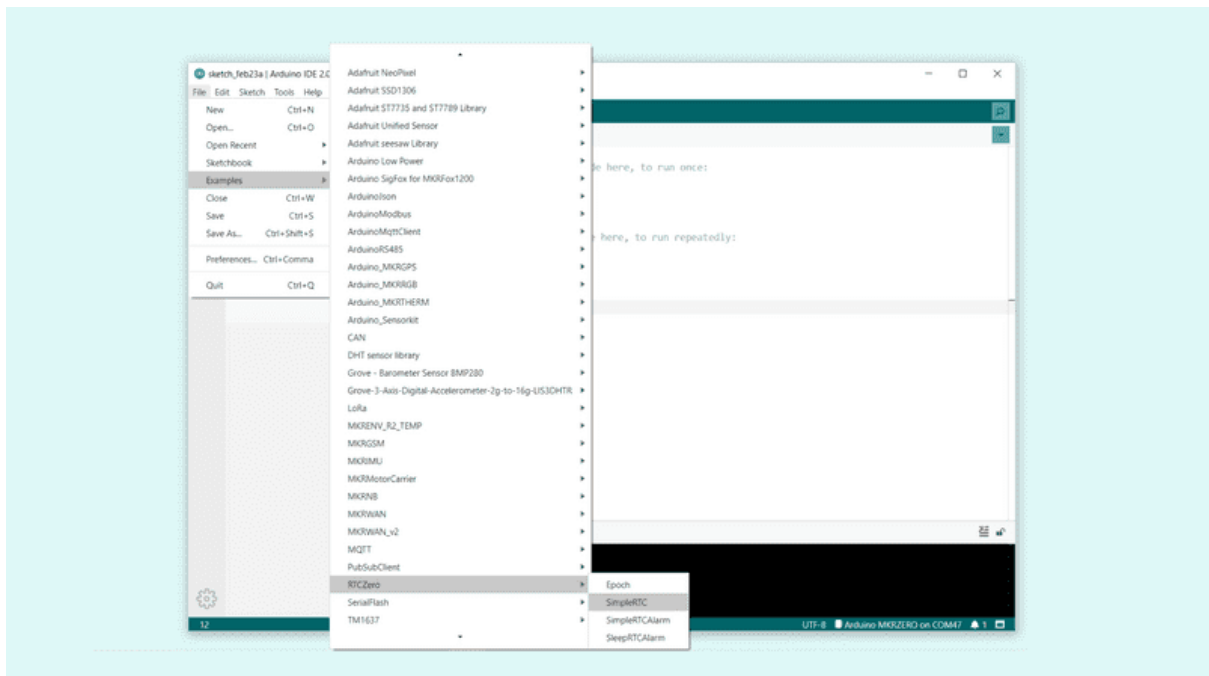
You have now successfully downloaded and installed a library on your machine.

## Including a library

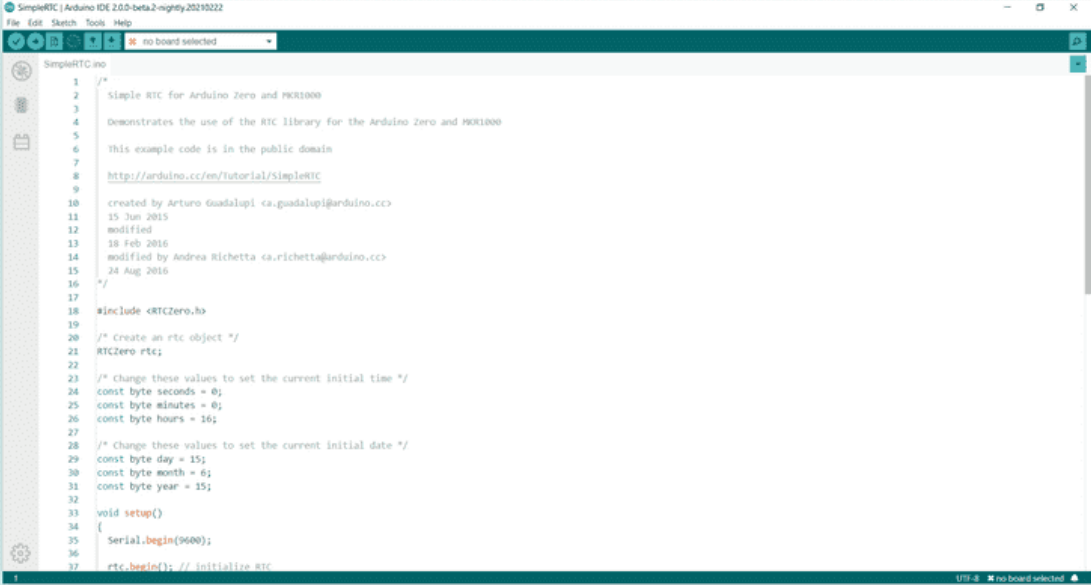
To use a library, you first need to include the library at the top of the sketch.



Almost all libraries come with already made examples that you can use. These are accessible through **File > Examples > {Library} > {Example}**. In this example, we are choosing the **RTCZero > SimpleRTC**.



The chosen example will now open up in a new window, and you can start using it however you want to.



```
SimpleRTC.ino
1  /*
2  Simple RTC for Arduino Zero and MKR1000
3
4  Demonstrates the use of the RTC library for the Arduino Zero and MKR1000
5
6  This example code is in the public domain
7
8  http://arduino.cc/en/tutorial/SimpleRTC
9
10 created by Arturo Guadalupi <a.guadalupi@arduino.cc>
11 15 Jun 2015
12 modified
13 18 Feb 2016
14 modified by Andrea Ricchetta <a.ricchetta@arduino.cc>
15 24 Aug 2016
16 */
17
18 #include <RTCZero.h>
19
20 /* Create an rtc object */
21 RTCZero rtc;
22
23 /* Change these values to set the current initial time */
24 const byte seconds = 0;
25 const byte minutes = 0;
26 const byte hours = 16;
27
28 /* Change these values to set the current initial date */
29 const byte day = 15;
30 const byte month = 6;
31 const byte year = 15;
32
33 void setup()
34 {
35   Serial.begin(9600);
36
37   rtc.begin(); // initialize RTC
```

**RESULT:**



<b>EXP NO:</b>	<b>INTRODUCTION TO ARDUINO PROGRAMMING</b>
<b>DATE</b>	

**AIM:**

To write and execute different Arduino programming for analog, digital signals and serial communication.

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Arduino IDE 2.0	1
2	Arduino UNO Development Board	1
3	Jumper Wires	few
4	Arduino USB Cable	1
5	Joystick Module	1

**PROCEDURE:**

**CONNECTION:**

Arduino UNO Pin	Arduino Development Board
2	LED

**PROGRAM:****DIGITAL WRITE:**

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

**CONNECTION:**

<b>Arduino UNO Pin</b>	<b>Arduino Development Board</b>
<b>2</b>	<b>LED</b>
<b>5</b>	<b>S1 (SW 1)</b>

**DIGITAL READ:**

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(5, INPUT_PULLUP);
}
void loop() {
  int sw=digitalRead(5);
  if(sw==1)
  {
    for(int i=0; i<5; i++)
    {
      digitalWrite(2, HIGH);
      delay(1000);
      digitalWrite(2, LOW);
      delay(1000);
    }
  }
  else
  {
    digitalWrite(2, LOW);
  }
}
```

**CONNECTION:**

<b>Arduino UNO Pin</b>	<b>Arduino Development Board</b>	<b>Joystick Module</b>
<b>2</b>	<b>LED</b>	
<b>VCC or 5V</b>		<b>+5V</b>
<b>GND</b>		<b>GND</b>
<b>A0</b>		<b>VRx or VRy</b>

**ANALOG READ:**

```
void setup() {  
  pinMode(2, OUTPUT);  
  Serial.begin(9600);  
}  
void loop() {  
  int joystick=analogRead(A0);  
  Serial.println(joystick);  
  
  if(joystick>800)  
    digitalWrite(2, HIGH);  
  else  
    digitalWrite(2, LOW);  
  
  delay(500);  
}
```

**CONNECTION:**

Arduino UNO Pin	Arduino Development Board
3	LED

*PWM Pins: 3, 5, 6, 9, 10, 11*



**ANALOG WRITE:**

```
void setup() {  
  pinMode(3, OUTPUT);  
}  
void loop() {  
  for(int i=0; i<256;i++) {  
    analogWrite(3,i);  
    delay(20);  
  }  
  for(int i=255; i>=0;i--) {  
    analogWrite(3,i);  
    delay(20);  
  }  
}
```

**CONNECTION:**

Arduino UNO Pin	Arduino Development Board
4	LED

**SERIAL COMMUNICATION:**

```
void setup() {  
    Serial.begin(9600);  
    pinMode(4, OUTPUT);  
}  
  
void loop() {  
if(Serial.available()>0)  
    {  
        char data=Serial.read();  
        Serial.println(data);  
        if(data=='1'){  
            digitalWrite(4,HIGH);  
        }  
        else if(data=='2'){  
            digitalWrite(4,LOW);  
        }  
    }  
}
```

**RESULT:**

<b>EXP NO:</b>	<b>Different communication methods with IoT devices (Zigbee, GSM, Bluetooth)</b>
<b>DATE</b>	

**AIM:**

To Explore different communication methods with IoT devices (Zigbee, GSM, Bluetooth).

**DIFFERENT COMMUNICATION METHODS:**

IoT devices require reliable and efficient communication methods to transmit data and interact with other devices or systems. Here are three commonly used communication methods for IoT devices:

**Zigbee:**

Zigbee is a low-power wireless communication protocol designed for short-range communication between devices. It operates on the 2.4 GHz frequency band and supports mesh networking, allowing devices to communicate with each other through intermediate nodes. Zigbee is commonly used in home automation, industrial control, and smart energy applications.

**GSM (Global System for Mobile Communications):**

GSM is a widely used cellular network technology that enables IoT devices to connect to the internet using SIM cards. It operates on various frequency bands and provides wide coverage, making it suitable for applications that require long-range communication. GSM is commonly used in applications such as asset tracking, remote monitoring, and smart cities.

**Bluetooth:**

Bluetooth is a short-range wireless communication technology that operates on the 2.4 GHz frequency band. It is commonly used for connecting IoT devices to smartphones, tablets, and other nearby devices. Bluetooth Low Energy (BLE) is a power-efficient version of Bluetooth that is ideal for battery-powered IoT devices. Bluetooth is widely used in applications such as wearable devices, healthcare monitoring, and home automation.

Each communication method has its advantages and limitations, and the choice depends on the specific requirements of the IoT application. Factors to consider include range, power consumption, data rate, security, and interoperability with other devices or systems.

**RESULT:**

<b>EXP NO:</b>	<b>BLUETOOTH COMMUNICATION</b>
<b>DATE</b>	

**AIM:**

To write a program to control an LED using a Bluetooth module.

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Arduino IDE 2.0	1
2	Arduino UNO Development Board	1
3	Jumper Wires	few
4	Arduino USB Cable	1
5	HC-05 Bluetooth Module	1

**PROCEDURE**

**CONNECTIONS:**

<b>Arduino UNO Pin</b>	<b>Bluetooth Module</b>	<b>Arduino Development Board</b>
<b>VCC</b>	<b>5V</b>	<b>-</b>
<b>GND</b>	<b>GND</b>	<b>-</b>
<b>2</b>	<b>Tx</b>	<b>-</b>
<b>3</b>	<b>Rx</b>	<b>-</b>
<b>4</b>	<b>-</b>	<b>LED</b>



**PROGRAM:**

```
#include<SoftwareSerial.h>
SoftwareSerial mySerial(2,3); //rx,tx
void setup() {
mySerial.begin(9600);
Serial.begin(9600);
pinMode(4, OUTPUT);
}

void loop() {
if(mySerial.available(>0)
{
char data=mySerial.read();
Serial.println(data);
if(data=='1'){
digitalWrite(4,HIGH);
Serial.println("LED ON");
}
else if(data=='2'){
digitalWrite(4,LOW);
Serial.println("LED OFF");
}
}
}
```

**RESULT:**

<b>EXP NO:</b>	<b>ZIGBEE COMMUNICATION</b>
<b>DATE</b>	

**AIM:**

To write a program to control an LED using a Zigbee module.

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Arduino IDE 2.0	1
2	Arduino UNO Development Board	2
3	Jumper Wires	few
4	Arduino USB Cable	2
5	Zigbee Module	2

**PROCEDURE**

**CONNECTIONS:****TRANSMITTER:**

<b>Arduino UNO Pin</b>	<b>Zigbee Module</b>
<b>VCC</b>	<b>5V</b>
<b>GND</b>	<b>G</b>
<b>2</b>	<b>Tx</b>
<b>3</b>	<b>Rx</b>

**PROGRAM:****TRANSMITTER SIDE:**

```
#include<SoftwareSerial.h>
SoftwareSerial mySerial(2,3); //rx,tx
void setup() {
    mySerial.begin(9600);
    Serial.begin(9600);
}

void loop() {
    mySerial.write('A');
    Serial.println('A');
    delay(100);
    mySerial.write('B');
    Serial.println('B');
    delay(100);
}
```

**CONNECTIONS:****RECEIVER:**

<b>Arduino UNO Pin</b>	<b>Zigbee Module</b>	<b>Arduino Development Board</b>
-	<b>5V</b>	<b>5V</b>
-	<b>G</b>	<b>GND</b>
<b>2</b>	<b>Tx</b>	-
<b>3</b>	<b>Rx</b>	-
<b>4</b>	-	<b>LED1</b>

**RECEIVER SIDE:**

```
#include<SoftwareSerial.h>
SoftwareSerial mySerial(2,3); //rx,tx
void setup() {
    mySerial.begin(9600);
    Serial.begin(9600);
    pinMode(4, OUTPUT);
}

void loop() {
if(mySerial.available(>0)
{
    char data=mySerial.read();
    Serial.println(data);
    if(data=='A')
        digitalWrite(4,HIGH);
    else if(data=='B')
        digitalWrite(4,LOW);
}
}
```

**RESULT:**



<b>EXP NO:</b>	<b>INTRODUCTION TO THE RASPBERRY PI PLATFORM</b>
<b>DATE</b>	

### **Introduction to Raspberry Pi Pico W:**

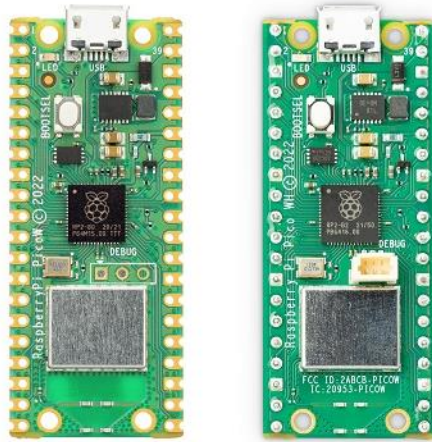
The Raspberry Pi Pico W is a compact and affordable microcontroller board developed by the Raspberry Pi Foundation. Building upon the success of the Raspberry Pi Pico, the Pico W variant brings wireless connectivity to the table, making it an even more versatile platform for embedded projects. In this article, we will provide a comprehensive overview of the Raspberry Pi Pico W, highlighting its key features and capabilities.

#### **Features:**

- RP2040 microcontroller with 2MB of flash memory
- On-board single-band 2.4GHz wireless interfaces (802.11n)
- Micro USB B port for power and data (and for reprogramming the flash)
- 40 pins 21mmx51mm ‘DIP’ style 1mm thick PCB with 0.1" through-hole pins also with edge castellations
- Exposes 26 multi-function 3.3V general purpose I/O (GPIO)
- 23 GPIO are digital-only, with three also being ADC-capable
- Can be surface mounted as a module
- 3-pin ARM serial wire debug (SWD) port
- Simple yet highly flexible power supply architecture
- Various options for easily powering the unit from micro-USB, external supplies, or batteries
- High quality, low cost, high availability
- Comprehensive SDK, software examples, and documentation
- Dual-core Cortex M0+ at up to 133MHz
- On-chip PLL allows variable core frequency
- 264kByte multi-bank high-performance SRAM

### **Raspberry Pi Pico W:**

The Raspberry Pi Pico W is based on the RP2040 microcontroller, which was designed by Raspberry Pi in-house. It combines a powerful ARM Cortex-M0+ processor with built-in Wi-Fi connectivity, opening up a range of possibilities for IoT projects, remote monitoring, and wireless communication. The Pico W retains the same form factor as the original Pico, making it compatible with existing Pico accessories and add-ons.



### **RP2040 Microcontroller:**

At the core of the Raspberry Pi Pico W is the RP2040 microcontroller. It features a dual-core ARM Cortex-M0+ processor running at 133MHz, providing ample processing power for a wide range of applications. The microcontroller also includes 264KB of SRAM, which is essential for storing and manipulating data during runtime. Additionally, the RP2040 incorporates 2MB of onboard flash memory for program storage, ensuring sufficient space for your code and firmware.

### **Wireless Connectivity:**

The standout feature of the Raspberry Pi Pico W is its built-in wireless connectivity. It includes an onboard Cypress CYW43455 Wi-Fi chip, which supports dual-band (2.4GHz and 5GHz) Wi-Fi 802.11b/g/n/ac. This allows the Pico W to seamlessly connect to wireless networks, communicate with other devices, and access online services. The wireless capability opens up new avenues for IoT projects, remote monitoring and control, and real-time data exchange.

### **GPIO and Peripherals:**

Similar to the Raspberry Pi Pico, the Pico W offers a generous number of GPIO pins, providing flexibility for interfacing with external components and peripherals. It features 26 GPIO pins, of which 3 are analog inputs, and supports various protocols such as UART, SPI, I2C, and PWM. The Pico W also includes onboard LED indicators and a micro-USB port for power and data connectivity.

### **MicroPython and C/C++ Programming:**

The Raspberry Pi Pico W can be programmed using MicroPython, a beginner-friendly programming language that allows for rapid prototyping and development. MicroPython provides a simplified syntax and high-level abstractions, making it easy for newcomers to get started. Additionally, the

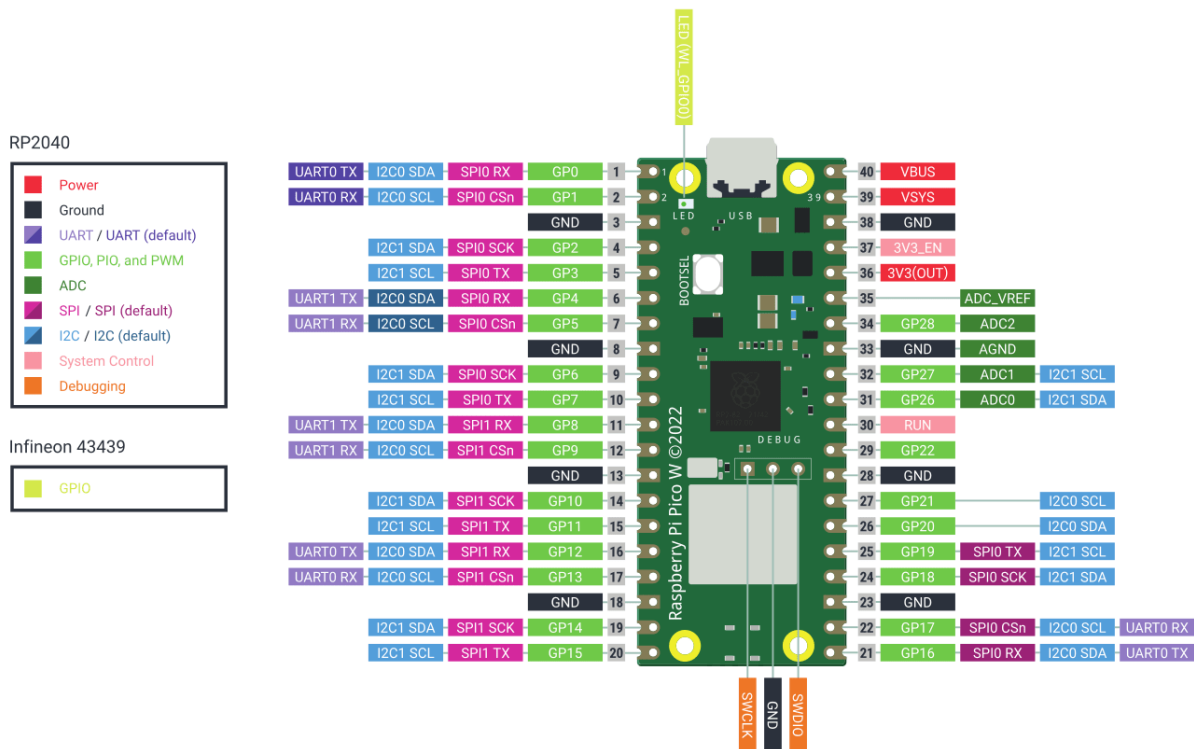
Pico W is compatible with C/C++ programming, allowing experienced developers to leverage the rich ecosystem of libraries and frameworks available.

**Programmable Input/Output (PIO) State Machines:**

One of the unique features of the RP2040 microcontroller is the inclusion of Programmable Input/Output (PIO) state machines. These state machines provide additional processing power and flexibility for handling real-time data and timing-critical applications. The PIO state machines can be programmed to interface with custom protocols, generate precise waveforms, and offload tasks from the main processor, enhancing the overall performance of the system.

**Open-Source and Community Support**

As with all Raspberry Pi products, the Pico W benefits from the vibrant and supportive Raspberry Pi community. Raspberry Pi provides extensive documentation, including datasheets, pinout diagrams, and programming guides, to assist developers in understanding the board’s capabilities. The community offers forums, online tutorials, and project repositories, allowing users to seek help, share knowledge, and collaborate on innovative projects.



The Raspberry Pi Pico W brings wireless connectivity to the popular Raspberry Pi Pico microcontroller board. With its powerful RP2040 microcontroller, built-in Wi-Fi chip, extensive GPIO capabilities, and compatibility with MicroPython and C/C++ programming, the Pico W offers a versatile and affordable platform for a wide range of embedded projects. Whether you are a beginner or an experienced developer, the Raspberry Pi Pico W provides a user-friendly and flexible platform to bring your ideas to life and explore the exciting world of wireless IoT applications.

**RESULT:**

<b>EXP NO:</b>	<b>INTRODUCTION TO PYTHON PROGRAMMING</b>
<b>DATE</b>	

### Getting Started with Thonny MicroPython (Python) IDE:

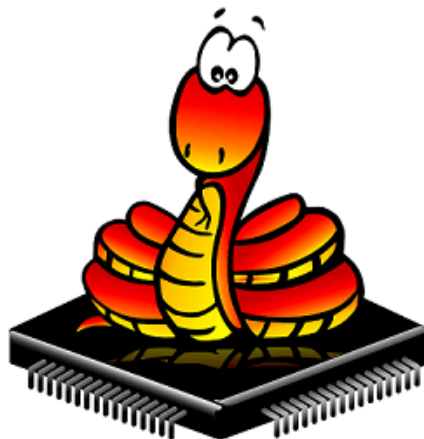
If you want to program your ESP32 and ESP8266 with MicroPython firmware, it's very handy to use an IDE. you'll have your first LED blinking using MicroPython and Thonny IDE.



### What is MicroPython?

MicroPython is a Python 3 programming language re-implementation targeted for microcontrollers and embedded systems. MicroPython is very similar to regular Python. Apart from a few exceptions, the language features of Python are also available in MicroPython. The most significant difference between Python and MicroPython is that MicroPython was designed to work under constrained conditions.

Because of that, MicroPython does not come with the entire pack of standard libraries. It only includes a small subset of the Python standard libraries, but it includes modules to easily control and interact with the GPIOs, use Wi-Fi, and other communication protocols.



## Thonny IDE:

Thonny is an open-source IDE which is used to write and upload MicroPython programs to different development boards such as Raspberry Pi Pico, ESP32, and ESP8266. It is extremely interactive and easy to learn IDE as much as it is known as the beginner-friendly IDE for new programmers. With the help of Thonny, it becomes very easy to code in MicroPython as it has a built-in debugger that helps to find any error in the program by debugging the script line by line.

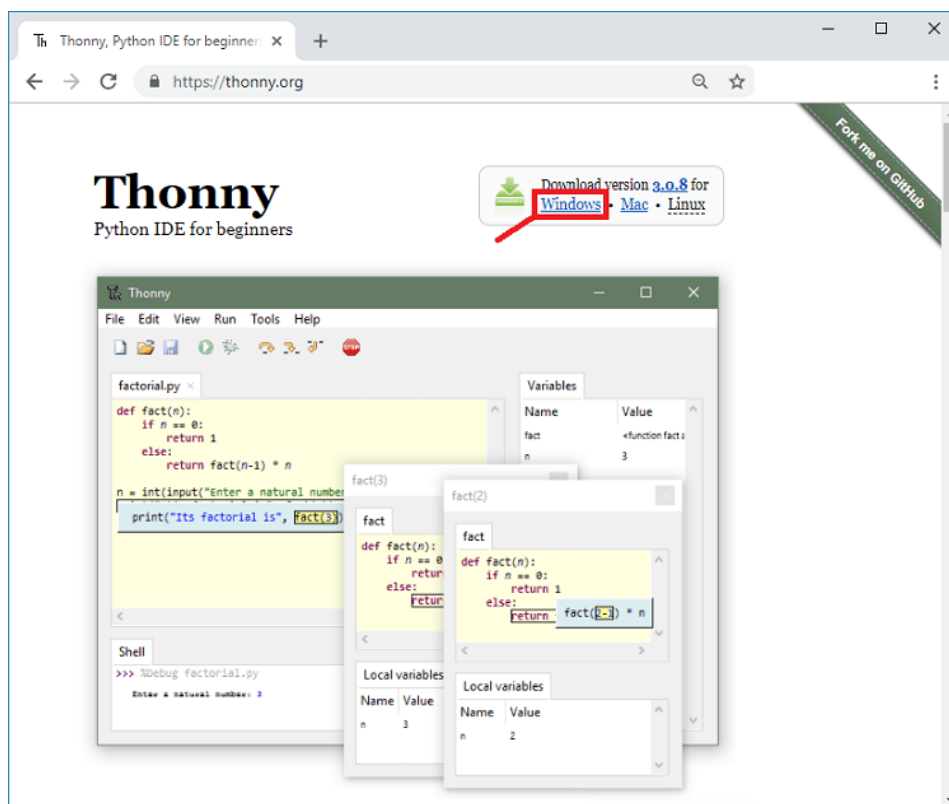
You can realize the popularity of Thonny IDE from this that it comes pre-installed in Raspbian OS which is an operating system for a Raspberry Pi. It is available to install on Windows, Linux, and Mac OS.

### A) Installing Thonny IDE – Windows PC

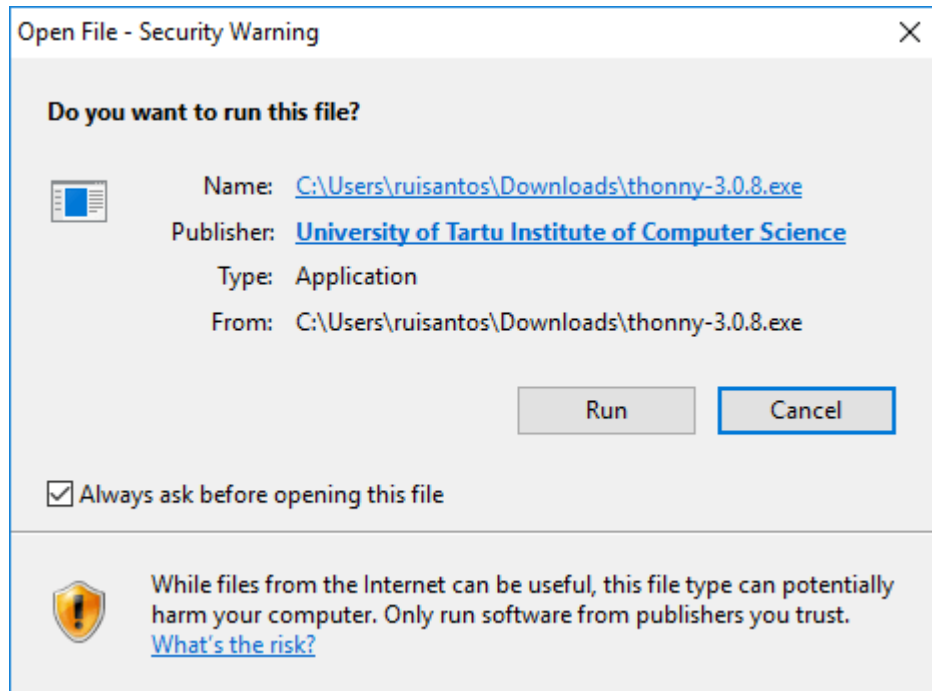
Thonny IDE comes installed by default on Raspbian OS that is used with the Raspberry Pi board.

To install Thonny on your Windows PC, follow the next instructions:

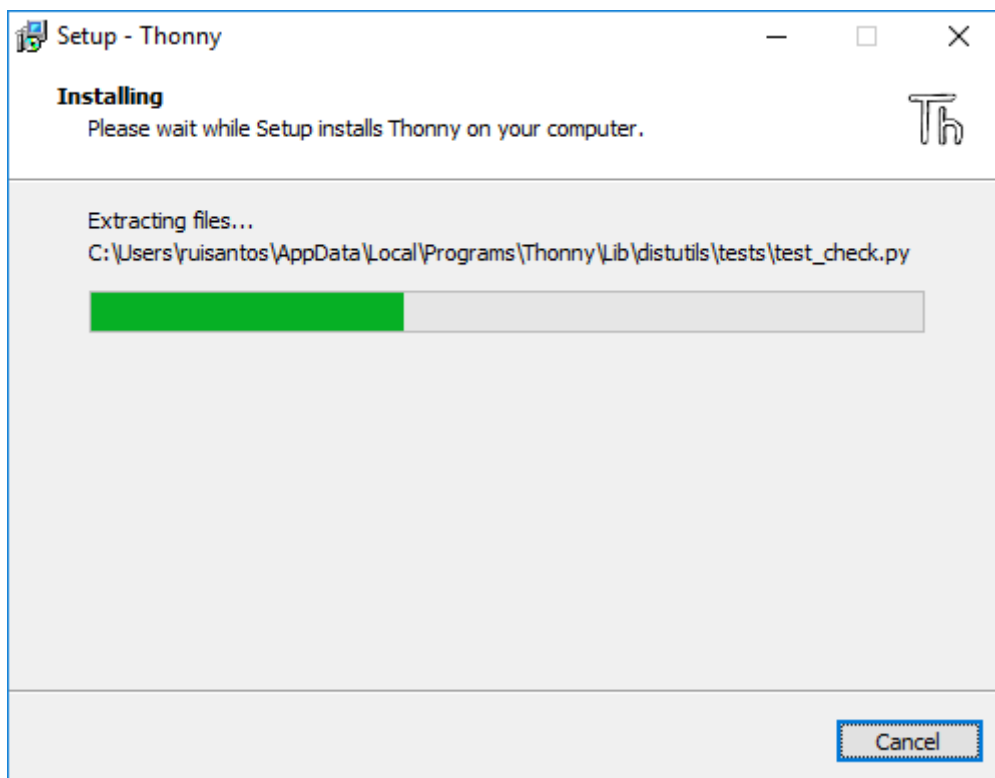
1. Go to <https://thonny.org>
2. Download the version for Windows and wait a few seconds while it downloads.



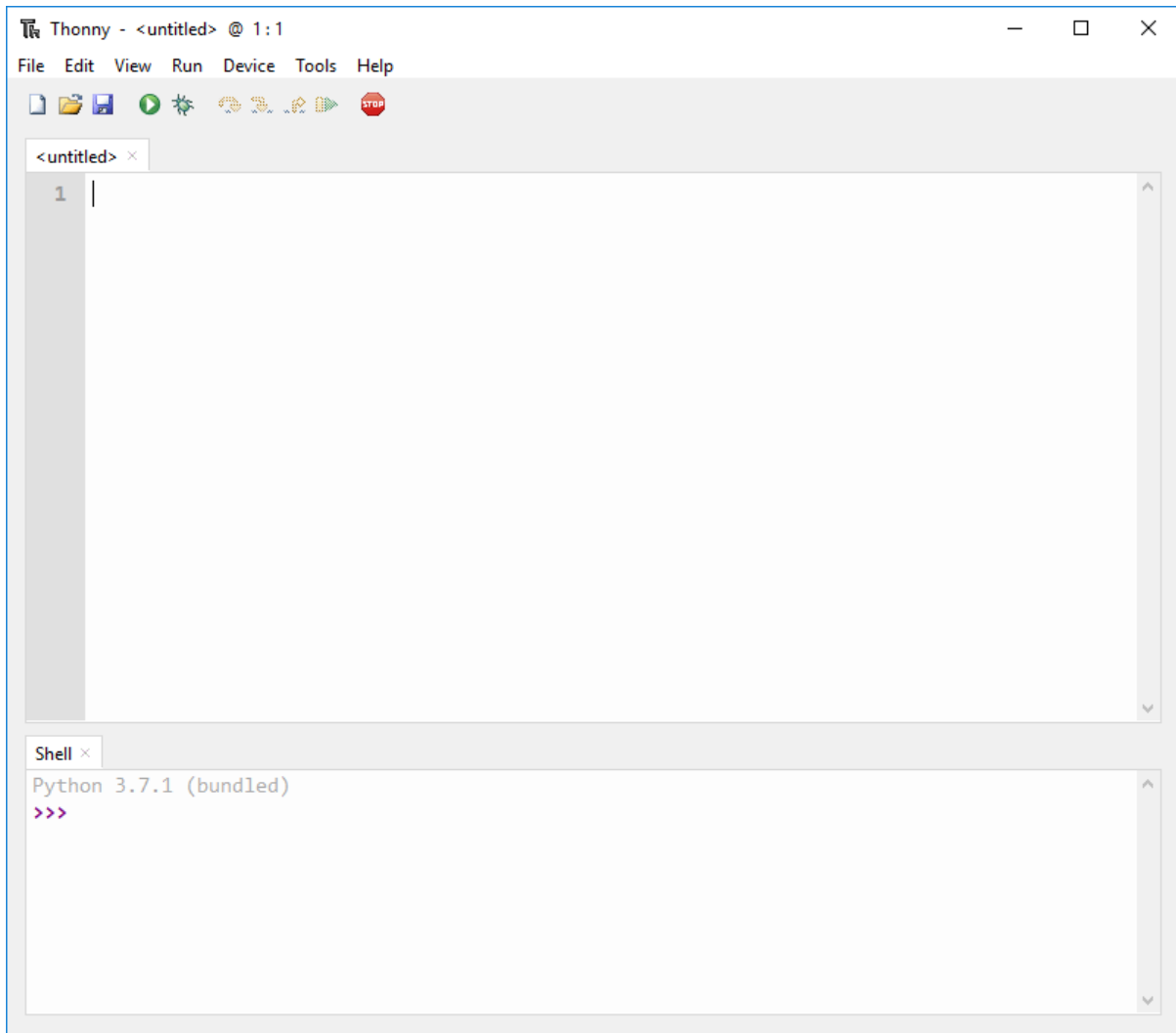
3. Run the .exe file.



4. Follow the installation wizard to complete the installation process. You just need to click “Next”.



5. After completing the installation, open Thonny IDE. A window as shown in the following figure should open.







**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>
<b>GP16</b>	<b>LED</b>

**PROGRAM****LED:**

```
from machine import Pin
import time
LED = Pin(16, Pin.OUT)
while True:
    LED.value(1)
    time.sleep(1)
    LED.value(0)
    time.sleep(1)
```

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board (RGB)</b>
<b>GP16</b>	<b>R</b>
<b>GP17</b>	<b>G</b>
<b>GP18</b>	<b>B</b>
<b>GND</b>	<b>COM</b>

**RGB:**

```
from machine import Pin
from time import sleep_ms,sleep
r=Pin(16,Pin.OUT)
y=Pin(17,Pin.OUT)
g=Pin(18,Pin.OUT)
```

```
while True:
```

```
    r.value(1)
    sleep_ms(1000)
    r.value(0)
    sleep_ms(1000)
    y.value(1)
    sleep(1)
    y.value(0)
    sleep(1)
    g.value(1)
    sleep(1)
    g.value(0)
    sleep(1)
```

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>
<b>GP16</b>	<b>LED</b>
<b>GP15</b>	<b>SW1</b>

**SWITCH CONTROLLED LED:**

```
from machine import Pin
from time import sleep
led=Pin(16,Pin.OUT)
sw=Pin(15,Pin.IN)
while True:
    bt=sw.value()
    if bt== True:
        print("LED ON")
        led.value(1)
        sleep(2)
        led.value (0)
        sleep(2)
        led.value (1)
        sleep(2)
        led.value(0)
        sleep(2)
    else:
        print("LED OFF")
        sleep(0.5)
```

**RESULT:**



<b>EXP NO:</b>	<b>INTERFACING SENSORS WITH RASPBERRY PI</b>
<b>DATE</b>	

**AIM:**

To interface the IR sensor and Ultrasonic sensor with Raspberry Pico.

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	1
3	Jumper Wires	few
4	Micro USB Cable	1
5	IR Sensor	1
6	Ultrasonic sensor	1

**PROCEDURE**

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>	<b>IR Sensor Module</b>
<b>GP16</b>	<b>BUZZER</b>	<b>-</b>
<b>GP15</b>	<b>-</b>	<b>OUT</b>
<b>-</b>	<b>5V</b>	<b>VCC</b>
<b>-</b>	<b>GND</b>	<b>GND</b>

**PROGRAM:****IR Sensor:**

```
from machine import Pin
from time import sleep
buzzer=Pin(16,Pin.OUT)
ir=Pin(15,Pin.IN)
while True:
    ir_value=ir.value()
    if ir_value== True:
        print("Buzzer OFF")
        buzzer.value(0)
    else:
        print("Buzzer ON")
        buzzer.value (1)
    sleep(0.5)
```

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>	<b>Ultrasonic Sensor Module</b>
<b>GP16</b>	<b>BUZZER</b>	<b>-</b>
<b>GP15</b>	<b>-</b>	<b>ECHO</b>
<b>GP14</b>	<b>-</b>	<b>TRIG</b>
<b>-</b>	<b>5V</b>	<b>VCC</b>
<b>-</b>	<b>GND</b>	<b>GND</b>

**ULTRASONIC SENSOR:**

```
from machine import Pin, PWM
import utime
trigger = Pin(14, Pin.OUT)
echo = Pin(15, Pin.IN)
buzzer = Pin(16, Pin.OUT)

def measure_distance():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()

    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance

while True:
    dist = measure_distance()
    print(f"Distance : {dist} cm")
    if dist <= 10:
        buzzer.value(1)
        utime.sleep(0.01)
    else:
        buzzer.value(0)
        utime.sleep(0.01)
    utime.sleep(0.5)
```

**RESULT:**

<b>EXP NO:</b>	<b>COMMUNICATE BETWEEN ARDUINO AND RASPBERRY PI</b>
<b>DATE</b>	

**AIM:**

To write and execute the program to Communicate between Arduino and Raspberry PI using any wireless medium (Bluetooth)

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	1
3	Arduino Uno Development Board	1
4	Jumper Wires	few
5	Micro USB Cable	1
6	Bluetooth Module	2

**PROCEDURE**

**CONNECTIONS:**

<b>Arduino UNO Pin</b>	<b>Arduino Development Board</b>	<b>Bluetooth Module</b>
<b>2</b>	<b>-</b>	<b>Tx</b>
<b>3</b>	<b>-</b>	<b>Rx</b>
<b>-</b>	<b>GND</b>	<b>GND</b>
<b>-</b>	<b>5V</b>	<b>5V</b>



**PROGRAM:****MASTER****ARDUINO:**

```
#include<SoftwareSerial.h>
SoftwareSerial mySerial(2,3); //rx,tx
void setup() {
    mySerial.begin(9600);
}

void loop() {
    mySerial.write('A');
    delay(1000);
    mySerial.write('B');
    delay(1000);
}
```

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>	<b>Bluetooth Module</b>
<b>GP16</b>	<b>LED</b>	<b>-</b>
<b>VCC</b>	<b>-</b>	<b>+5V</b>
<b>GND</b>	<b>-</b>	<b>GND</b>
<b>GP1</b>	<b>-</b>	<b>Tx</b>
<b>GP0</b>	<b>-</b>	<b>Rx</b>

**SLAVE****RASPBERRY PI PICO**

```
from machine import Pin, UART
```

```
uart = UART(0, 9600)
```

```
led = Pin(16, Pin.OUT)
```

```
while True:
```

```
    if uart.any() > 0:
```

```
        data = uart.read()
```

```
        print(data)
```

```
        if "A" in data:
```

```
            led.value(1)
```

```
            print('LED on \n')
```

```
            uart.write('LED on \n')
```

```
        elif "B" in data:
```

```
            led.value(0)
```

```
            print('LED off \n')
```

```
            uart.write('LED off \n')
```

**RESULT:**

<b>EXP NO:</b>	<b>CLOUD PLATFORM TO LOG THE DATA</b>
<b>DATE</b>	

**AIM:**

To set up a cloud platform to log the data from IoT devices.

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

S.No.	Software Requirements	Quantity
1	Blynk Platform	1

**CLOUD PLATFORM-BLYNK:**

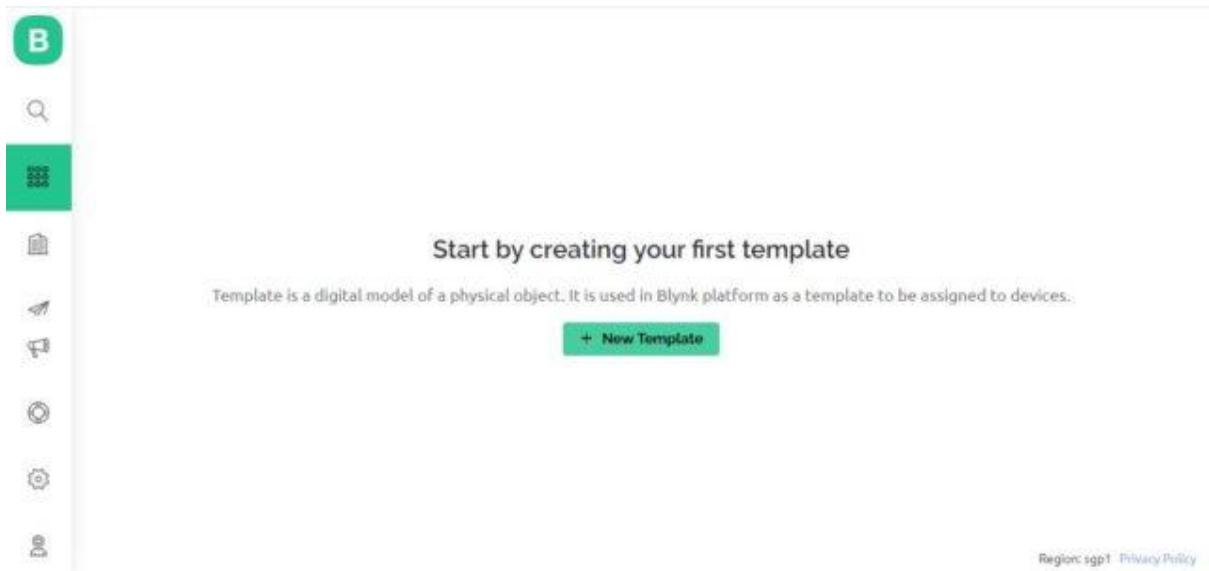
Blynk is a smart platform that allows users to create their Internet of Things applications without the need for coding or electronics knowledge. It is based on the idea of physical programming & provides a platform to create and control devices where users can connect physical devices to the Internet and control them using a mobile app.

**Setting up Blynk 2.0 Application**

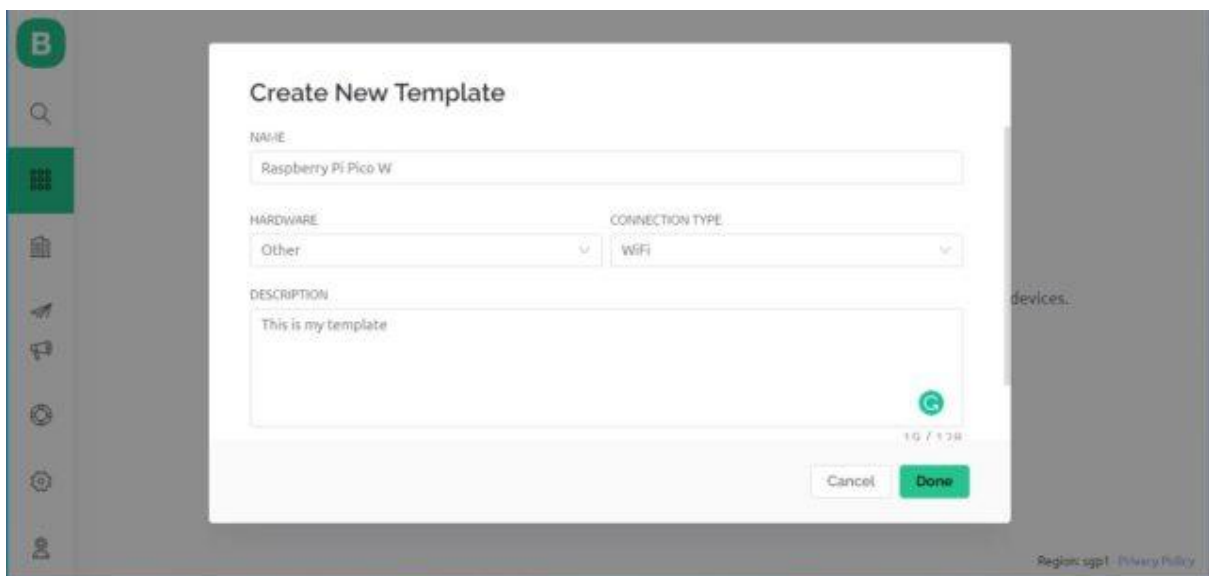
To control the LED using Blynk and Raspberry Pi Pico W, you need to create a Blynk project and set up a dashboard in the mobile or web application. Here's how you can set up the dashboard:

**Step 1:** Visit **blynk.cloud** and create a Blynk account on the Blynk website. Or you can simply sign in using the registered Email ID.

**Step 2:** Click on **+New Template**.



**Step 3:** Give any name to the Template such as Raspberry Pi Pico W. Select 'Hardware Type' as Other and 'Connection Type' as WiFi.



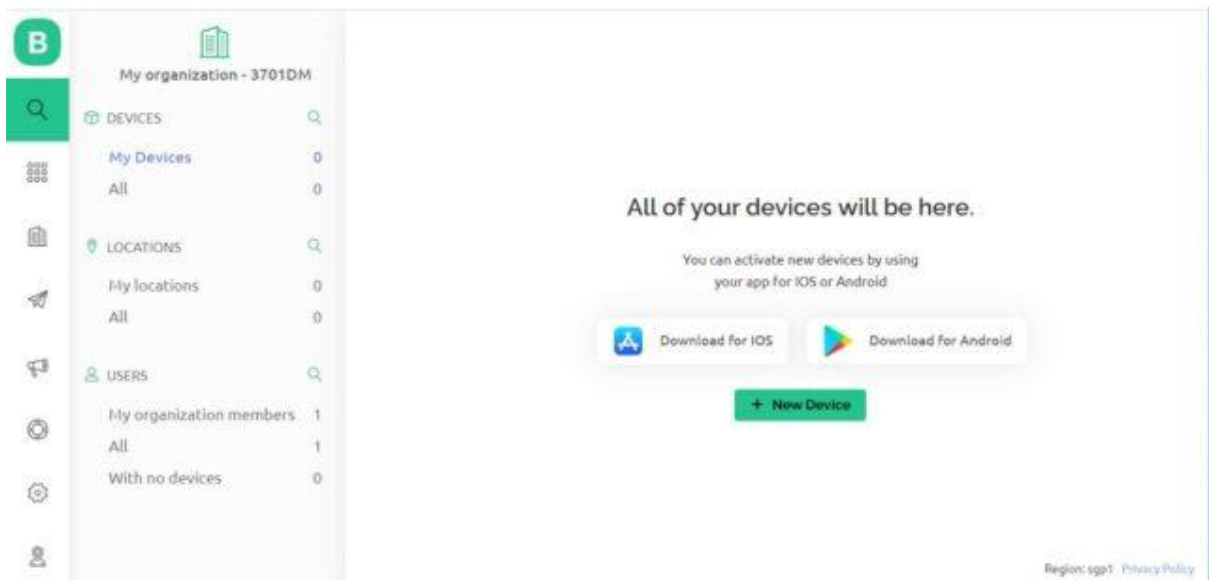
So a template will be created now.

The screenshot shows the Blynk web interface for creating a new device template. The page title is "Raspberry Pi Pico W". The interface includes a sidebar with navigation icons and a main content area with the following sections:

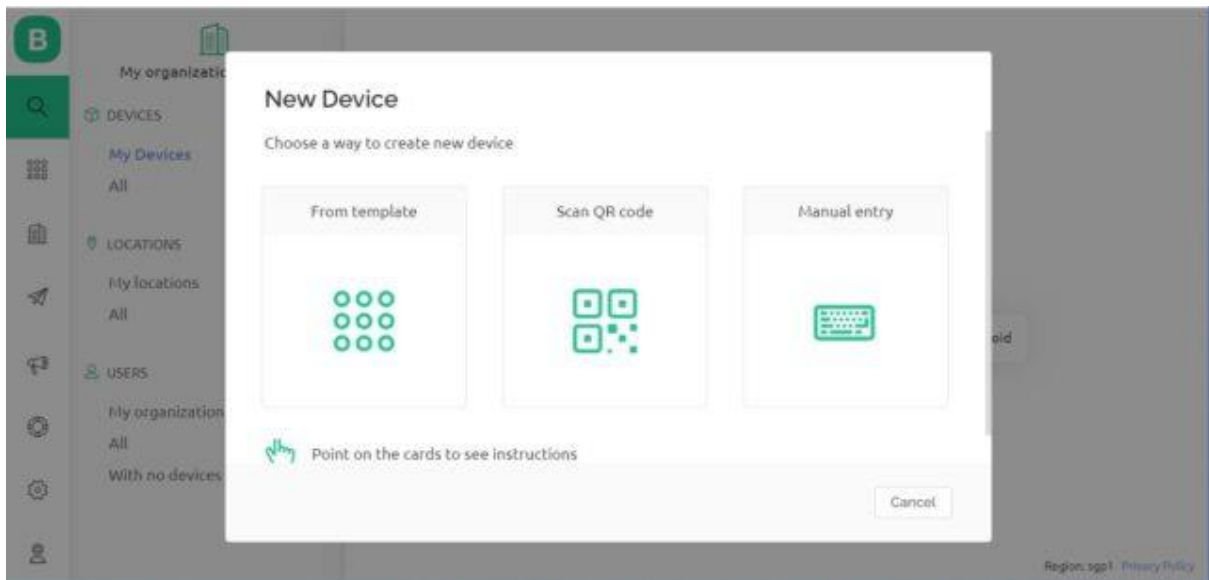
- Info** (selected), Metadata, Datastreams, Events, Automations, Web Dashboard, Mobile Dashboard
- TEMPLATE NAME:** Raspberry Pi Pico W
- HARDWARE:** Other (dropdown)
- CONNECTION TYPE:** WIFI (dropdown)
- DESCRIPTION:** This is my template
- TEMPLATE IMAGE (OPTIONAL):** Add image. Upload from computer or drag-n-drop .png or .jpg, minimum width 500px.
- FIRMWARE CONFIGURATION:**

```
#define BLYNK_TEMPLATE_ID "TMPLK59p12Jb"
#define BLYNK_TEMPLATE_NAME "Raspberry Pi Pico W"
```
- TEMPLATE ID:** TMPLK59p12Jb
- MANUFACTURER:** My organization 3701DM
- 19 / 128
- Template ID and Device Name should be included at the top of your main firmware
- Region: sgp1 Privacy Policy

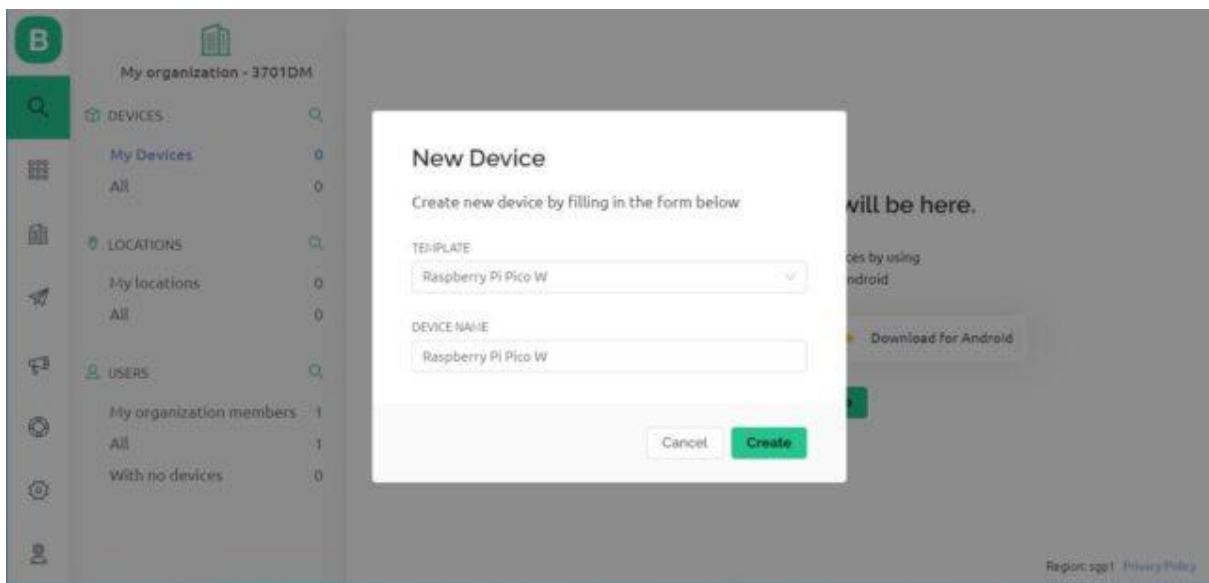
**Step 4:** Now we need to add a 'New Device' now.



Select a New Device from 'Template'.

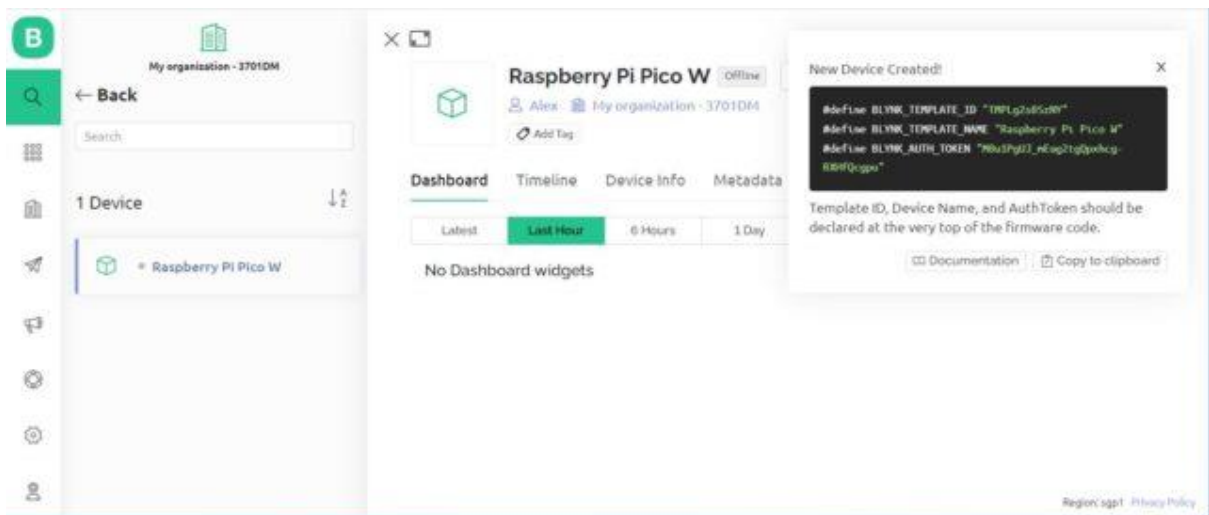


Select the device from a template that you created earlier and also give any name to the device. Click on Create.

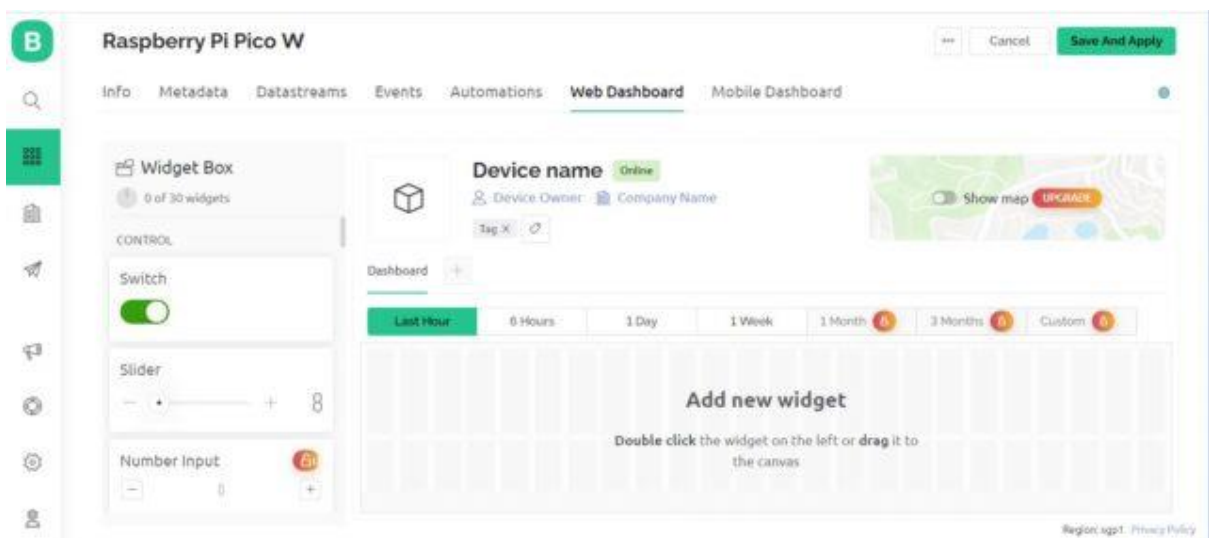


A new device will be created. You will find the Blynk Authentication Token Here. Copy it as it is necessary for the code.

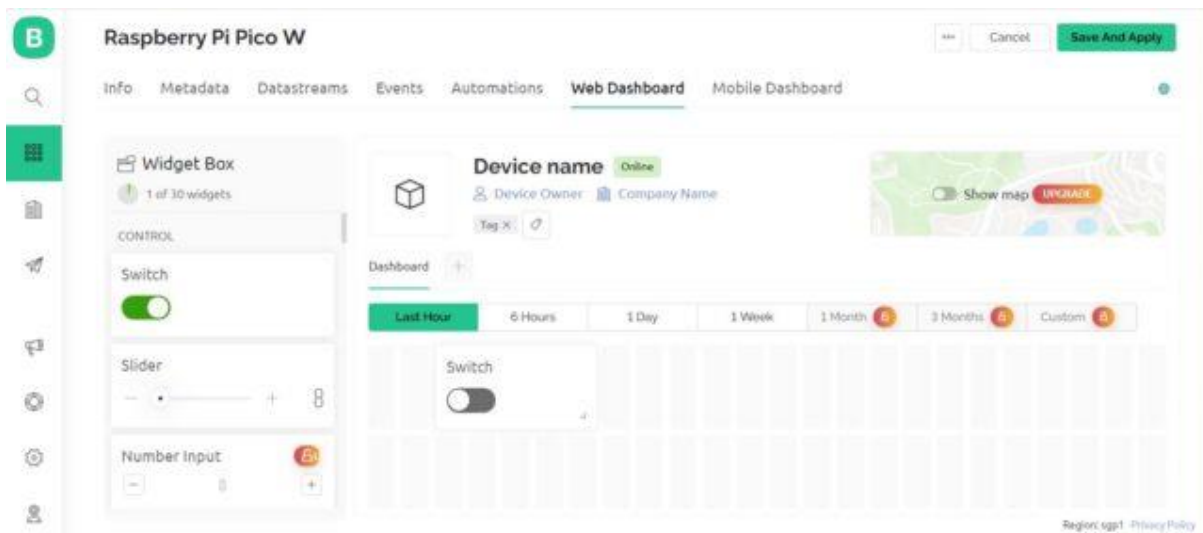




**Step 5:** Now go to the dashboard and select 'Web Dashboard'.

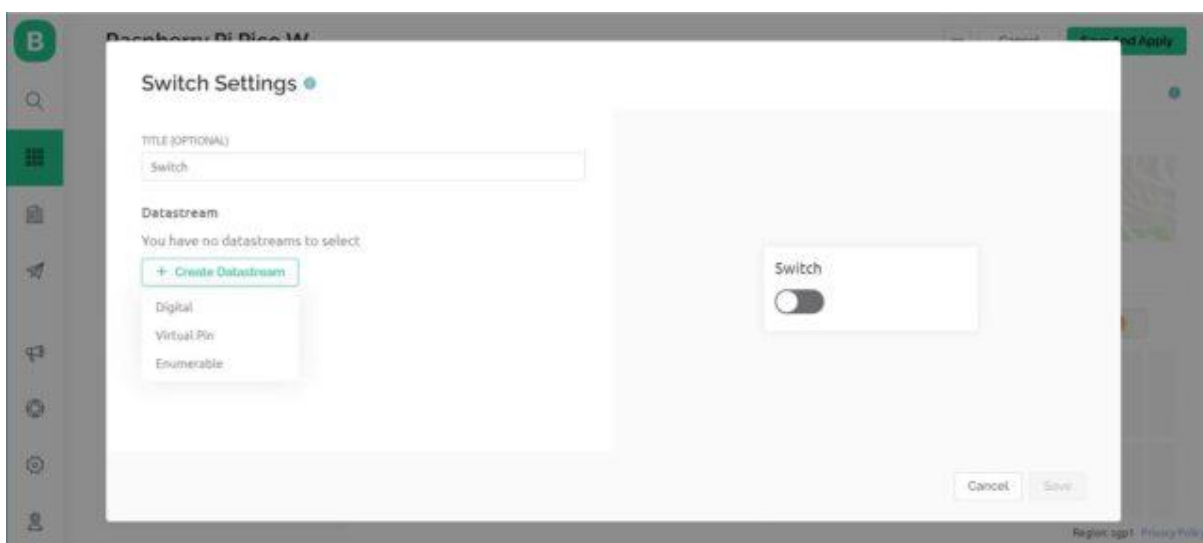


From the widget box drag a switch and place it on the dashboard screen.

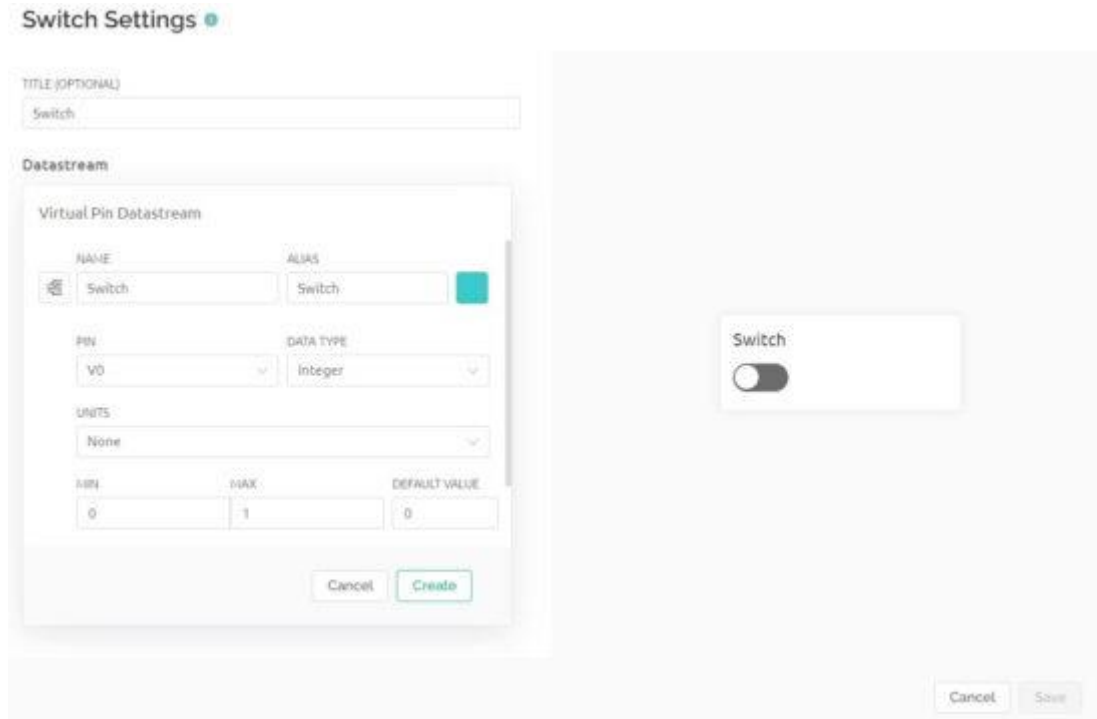


### Step 6:

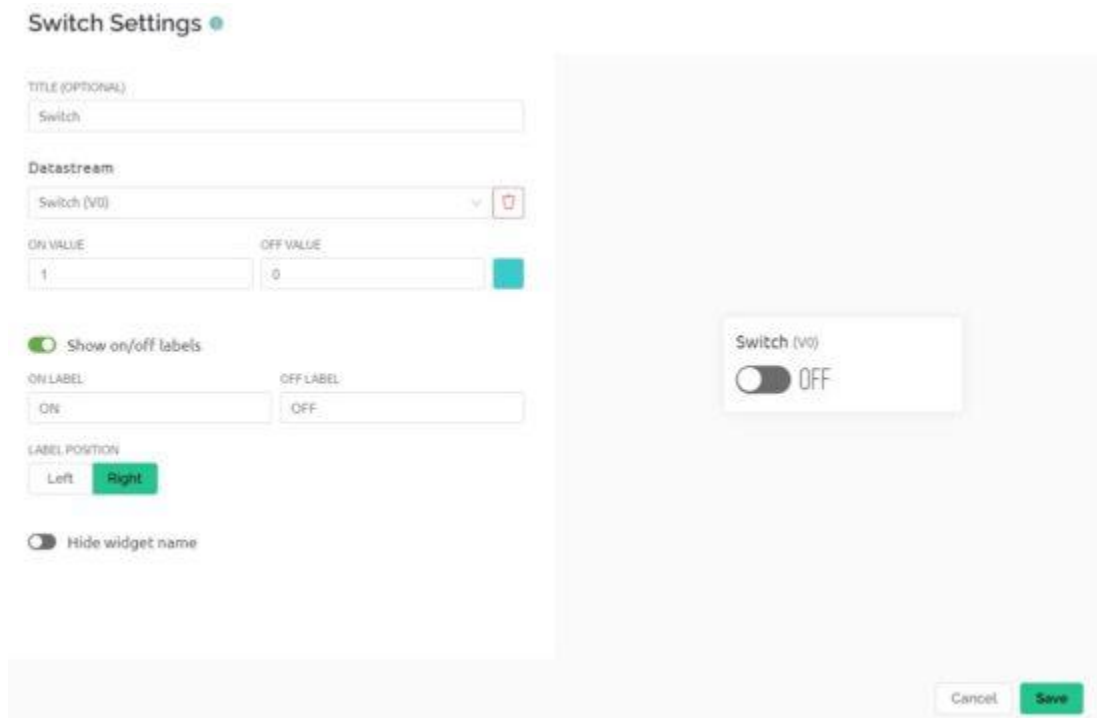
On the switch board click on Settings and here you need to set up the Switch. Give any title to it and Create Datastream as Virtual Pin.



Configure the switch settings as per the image below and click on create.



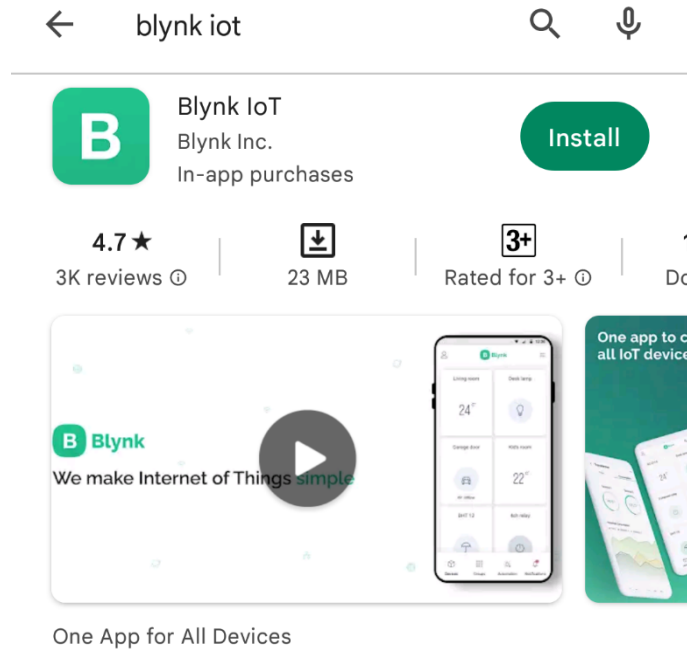
Configure the final steps again.



With this Blynk dashboard set up, you can now proceed to program the Raspberry Pi Pico W board to control the LED.

**Step 7:**

To control the LED with a mobile App or Mobile Dashboard, you also need to setup the Mobile Phone Dashboard. The process is similarly explained above.



Install the Blynk app on your smartphone The Blynk app is available for iOS and Android. Download and install the app on your smartphone. then need to set up both the Mobile App and the Mobile Dashboard in order to control the LED with a mobile device. The process is explained above.

1. Open Google Play Store App on an android phone
2. Open Blynk.App
3. Log In to your account (using the same email and password)
4. Switch to Developer Mode
5. Find the “**Raspberry Pi Pico Pico W**” template we created on the web and tap on it
6. Tap on the “**Raspberry Pi Pico Pico W**” template (this template automatically comes because we created it on our dashboard).
7. tap on plus icon on the left-right side of the window
8. Add one button Switch
9. Now We Successfully Created an android template
10. it will work similarly to a web dashboard template



**RESULT:**

<b>EXP NO:</b>	<b>Log Data using Raspberry PI and upload it to the cloud platform</b>
<b>DATE</b>	

**AIM:**

To write and execute the program Log Data using Raspberry PI and upload it to the cloud platform

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	few
3	Jumper Wires	1
4	Micro USB Cable	1

**PROCEDURE**

**CONNECTIONS:**

<b>Raspberry Pi Pico Pin</b>	<b>Raspberry Pi Pico Development Board</b>	<b>LCD Module</b>
-	<b>5V</b>	<b>VCC</b>
-	<b>GND</b>	<b>GND</b>
<b>GP0</b>	-	<b>SDA</b>
<b>GP1</b>	-	<b>SCL</b>



**PROGRAM:**

```

from machine import Pin, I2C, ADC
from utime import sleep_ms
from pico_i2c_lcd import I2cLcd
import time
import network
import BlynkLib

adc = machine.ADC(4)
i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)
I2C_ADDR=i2c.scan()[0]
lcd=I2cLcd(i2c,I2C_ADDR,2,16)

wlan = network.WLAN()
wlan.active(True)
wlan.connect("Wifi_Username","Wifi_Password")

BLYNK_AUTH = 'Your_Token'

# connect the network
wait = 10
while wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('network connection failed')
else:
    print('connected')
    ip=wlan.ifconfig()[0]
    print('IP: ', ip)

"Connection to Blynk"
# Initialize Blynk
blynk = BlynkLib.Blynk(BLYNK_AUTH)

lcd.clear()

```

```
while True:
    ADC_voltage = adc.read_u16() * (3.3 / (65536))
    temperature_celcius = 27 - (ADC_voltage - 0.706)/0.001721
    temp_fahrenheit=32+(1.8*temperature_celcius)
    print("Temperature in C: {}".format(temperature_celcius))
    print("Temperature in F: {}".format(temp_fahrenheit))

    lcd.move_to(0,0)
    lcd.putstr("Temp:")
    lcd.putstr(str(round(temperature_celcius,2)))
    lcd.putstr("C ")
    lcd.move_to(0,1)
    lcd.putstr("Temp:")
    lcd.putstr(str(round(temp_fahrenheit,2)))
    lcd.putstr("F")
    time.sleep(5)

    blynk.virtual_write(3, temperature_celcius)
    blynk.virtual_write(4, temp_fahrenheit)
    blynk.log_event(temperature_celcius)

    blynk.run()

    time.sleep(5)
```

**RESULT:**

<b>EXP NO:</b>	<b>Design an IOT-based system</b>
<b>DATE</b>	

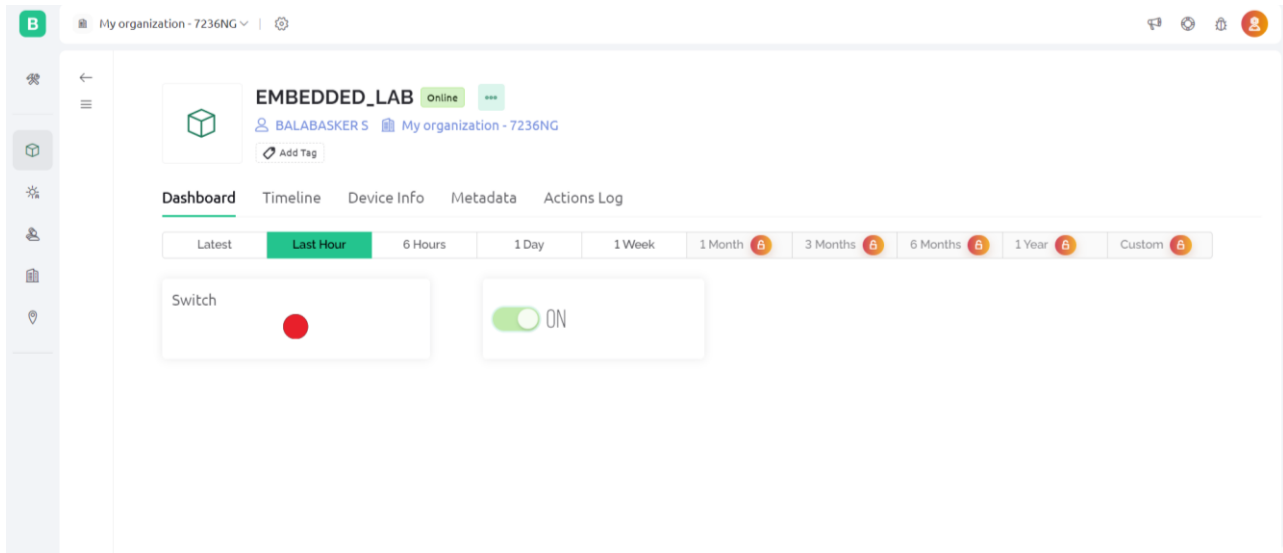
**AIM:**

To design a Smart Home Automation IOT-based system

**HARDWARE & SOFTWARE TOOLS REQUIRED:**

<b>S.No</b>	<b>Hardware &amp; Software Requirements</b>	<b>Quantity</b>
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	few
3	Jumper Wires	1
4	Micro USB Cable	1
5	LED or Relay	1

**PROCEDURE**



### CONNECTIONS:

Raspberry Pi Pico Pin	Raspberry Pi Pico Development Board
GP16	LED 1

**PROGRAM:**

```

import time
import network
import BlynkLib
from machine import Pin
led=Pin(16, Pin.OUT)

wlan = network.WLAN()
wlan.active(True)
wlan.connect("Wifi_Username","Wifi_Password")
BLYNK_AUTH = 'Your_Token'

# connect the network
wait = 10
while wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('network connection failed')
else:
    print('connected')
    ip=wlan.ifconfig()[0]
    print('IP: ', ip)

"Connection to Blynk"
# Initialize Blynk
blynk = BlynkLib.Blynk(BLYNK_AUTH)

# Register virtual pin handler
@blynk.on("V0") #virtual pin V0
def v0_write_handler(value): #read the value
    if int(value[0]) == 1:
        led.value(1) #turn the led on
    else:

```

```
    led.value(0) #turn the led off  
while True:  
    blynk.run()
```

**RESULT:**

## EXTRA PROGRAMS

### LCD DISPLAY:

```
from machine import Pin, I2C
from time import sleep
from pico_i2c_lcd import I2cLcd
i2c=I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)
I2C_ADDR=i2c.scan()[0]
lcd=I2cLcd(i2c,I2C_ADDR,2,16)
```

while True:

```
    lcd.move_to(3,0)
    lcd.putstr("Ediylabs")
    sleep(5)
    lcd.clear()
```

<b>Raspberry Pi Pico Development Board</b>	<b>16X2 LCD Display</b>
<b>5V</b>	<b>VCC</b>
<b>GND</b>	<b>GND</b>
<b>GP0</b>	<b>SDA</b>
<b>GP1</b>	<b>SCL</b>

## **DHT 11 Sensor:**

```
from machine import Pin, I2C
import utime as time
from dht import DHT11, InvalidChecksum
while True:
    time.sleep(1)
    pin = Pin(16, Pin.OUT, Pin.PULL_DOWN)
    sensor = DHT11(pin)
    t = (sensor.temperature)
    h = (sensor.humidity)
    print("Temperature: {}".format(sensor.temperature))
    print("Humidity: {}".format(sensor.humidity))
    time.sleep(2)
```

## **Connection**

<b>Raspberry Pi Pico Development Board</b>	<b>DHT11</b>
<b>5V</b>	<b>VCC</b>
<b>GND</b>	<b>GND</b>
<b>GP16</b>	<b>DATA</b>



## **SERVO MOTOR:**

```
from time import sleep
```

```
from machine import Pin, PWM
```

```
pwm = PWM(Pin(1))
```

```
pwm.freq(50)
```

```
while True:
```

```
    for position in range(1000,9000,50):
```

```
        pwm.duty_u16(position)
```

```
        sleep(0.01)
```

```
    for position in range(9000,1000,-50):
```

```
        pwm.duty_u16(position)
```

```
        sleep(0.01)
```

## **Connection**

<b>Raspberry Pi Pico Development Board</b>	<b>SERVOMOTOR</b>
<b>GND</b>	<b>BROWM</b>
<b>5V</b>	<b>RED</b>
<b>GP1</b>	<b>ORANGE</b>

## **STEPPER MOTOR:**

```
from machine import Pin
```

```
from time import sleep
```

```
IN1 = Pin(12,Pin.OUT)
```

```
IN2 = Pin(13,Pin.OUT)
```

```
IN3 = Pin(14,Pin.OUT)
```

```
IN4 = Pin(15,Pin.OUT)
```

```
pins = [IN1, IN2, IN3, IN4]
```

```
sequence = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]
```

```
while True:
```

```
    for step in sequence:
```

```
        for i in range(len(pins)):
```

```
            pins[i].value(step[i])
```

```
            sleep(0.001)
```

## **Connection**

<b>Raspberry Pi Pico Development Board</b>	<b>STEPPER MOTOR</b>
<b>5V</b>	<b>+ (5V)</b>
<b>GND</b>	<b>- (GND)</b>
<b>GP12</b>	<b>IN1</b>
<b>GP13</b>	<b>IN2</b>
<b>GP14</b>	<b>IN3</b>
<b>GP15</b>	<b>IN4</b>