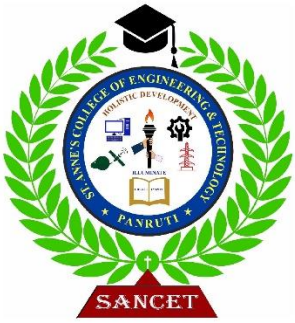
ST.ANNE'S

COLLEGE OF ENGINEERING AND TECHNOLOGY

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

Accredited by NAAC

ANGUCHETTYPALAYAM, PANRUTI – 607 106.



ET3491 - EMBEDDED SYSTEMS AND IOT DESIGN

OBSERVATION NOTE

(FOR III B.E ELECTRONICS AND COMMUNICATION ENGINEERING)

NAME : _____

REGISTER NO : _____

YEAR/SEMESTER: III Year / VI Semester

PERIOD : **FEB 2025 – MAY 2025**

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 **ET3491 - EMBEDDED SYSTEMS AND IOT DESIGN** course, along with the design of various components of the circuit and controller.
- The experiment's aim is also given at the beginning of each experiment. Once the student can design the circuit as per the circuit diagram, they are 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.

ET3491 - EMBEDDED SYSTEMS AND IOT DESIGN

SYLLABUS

COURSE OBJECTIVES:

- Learn the architecture and features of 8051.
- Study the design process of an embedded system.
- Understand the real–time processing in an embedded system.
- Learn the architecture and design flow of IoT.
- Build an IoT based system.

LIST OF EXPERIMENTS

Experiments using 8051

1. Programming Arithmetic and Logical Operations in 8051.
2. Generation of Square waveform using 8051.
3. Programming using on – Chip ports in 8051.
4. Programming using Serial Ports in 8051.
5. Design of a Digital Clock using Timers/Counters in 8051.

Experiments using ARM

1. Interfacing ADC and DAC
2. Blinking of LEDs and LCD
3. Interfacing keyboard and Stepper Motor.

Miniprojects for IoT

1. Garbage Segregator and Bin Level Indicator
2. Colour based Product Sorting
3. Image Processing based Fire Detection
4. Vehicle Number Plate Detection
5. Smart Lock System

OUTCOMES:

CO1: Explain the architecture and features of 8051.
CO2: Develop a model of an embedded system.
CO3: List the concepts of real time operating systems.
CO4: Learn the architecture and protocols of IoT.
CO5: Design an IoT based system for any application.

LIST OF EXPERIMENTS

S.No.	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

LIST OF EXPERIMENTS

S.No.	DATE	NAME OF THE EXPERIMENT	PAGE NO	DATE OF SUBMISSION	MARK (10)	SIGNATURE

INTRODUCTION
TO
KEIL µ VISION 5

INTRODUCTION TO KEIL µ VISION 5 SOFTWARE

The µVision5 IDE is a Windows-based software development platform that combines a robust editor, project manager, and make facility. µVision5 integrates all tools, including the C compiler, macro assembler, linker/locator, and HEX file generator. µVision4 helps expedite the development process of your embedded applications by providing the following:

- ✓ Full-featured source code editor
- ✓ Device database for configuring the development tool set
- ✓ Project manager for creating and maintaining your projects
- ✓ Integrated make facility for assembling, compiling, and linking your embedded applications
- ✓ Dialogs for all development tool settings
- ✓ True integrated source-level Debugger with high-speed CPU and peripheral simulator
- ✓ Advanced GDI interface for software debugging in the target hardware and for connection to Keil ULINK
- ✓ Flash programming utility for downloading the application program into Flash ROM
- ✓ Links to development tools manuals, device datasheets & users' guides

The Keil µVision5 IDE offers numerous features and advantages that help you quickly and successfully develop embedded applications. It is easy to use and guaranteed to help you achieve your design goals.

The installation steps for Keil software are given below:

1. Double click on Keil µvision4.exe file.
2. Then click on **Next**.
3. Tick the check box towards to license agreements and click **Next**.
4. Select Destination folder and click **Next**.
5. Fill the necessary text boxes and click **Next**.
6. Finally click on **Finish**.

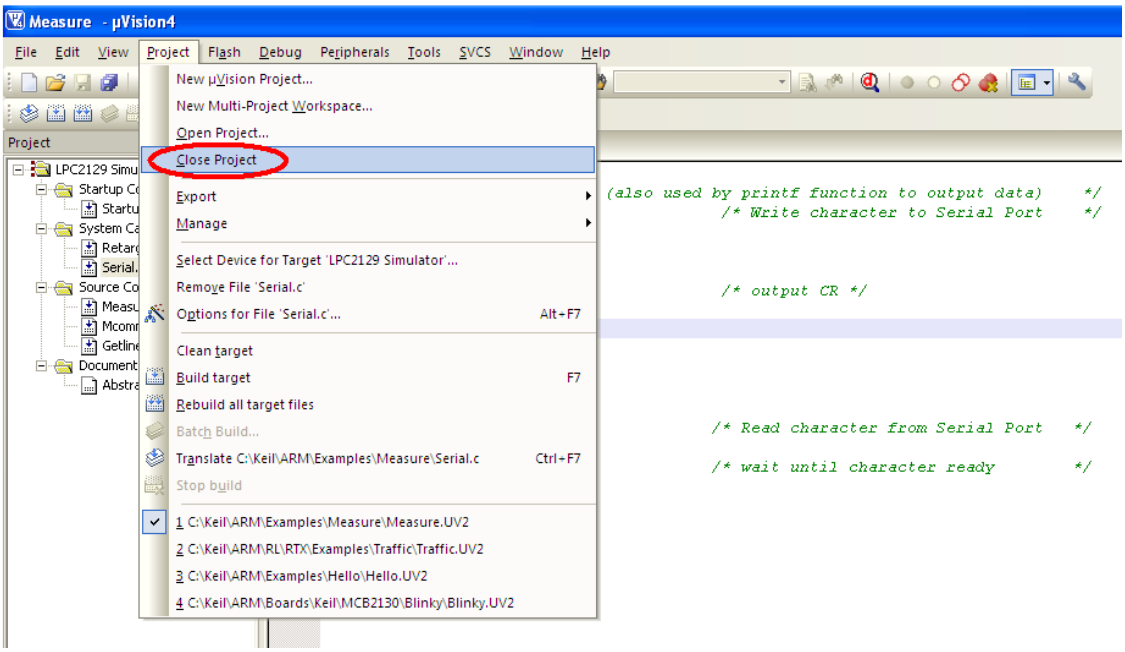
Software Flow

First open the icon keil µvision4 and the follow the steps are given below. The menu bar provides you with menus for editor operations, project maintenance, development tool option settings, program debugging, external tool control, window selection and manipulation, and on-line help. The toolbar buttons allow you to rapidly execute µVision4 commands. A Status Bar provides editor and debugger information. The various toolbars and the status bar can be enabled or disabled from the View Menu commands.

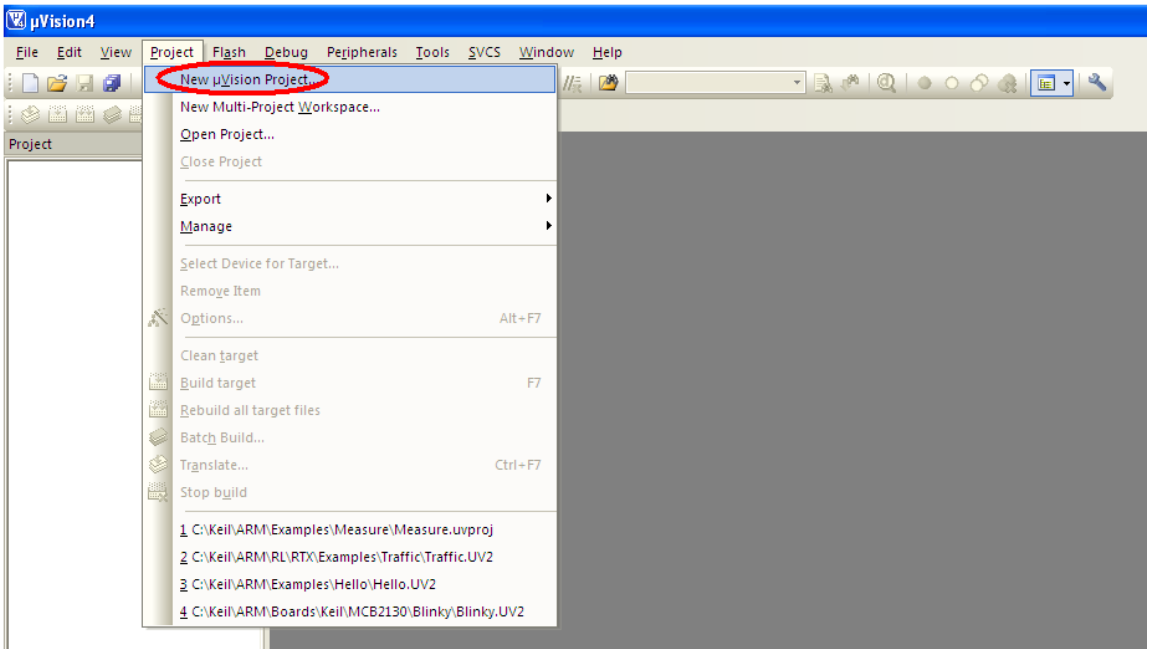
Creating a New Project

The mentioned procedures will explain the steps required to set up a simple application and to generate a HEX output.

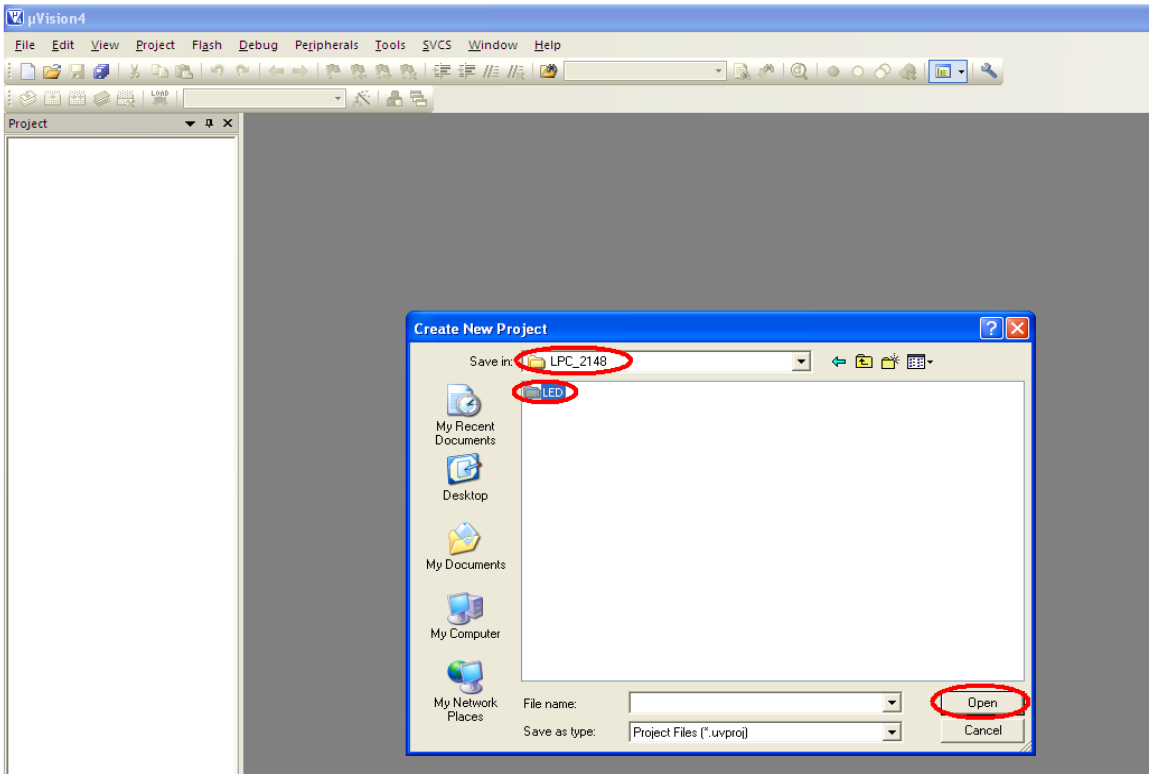
STEP 1: Go to "**Project**" and close the current project "**Close Project**".



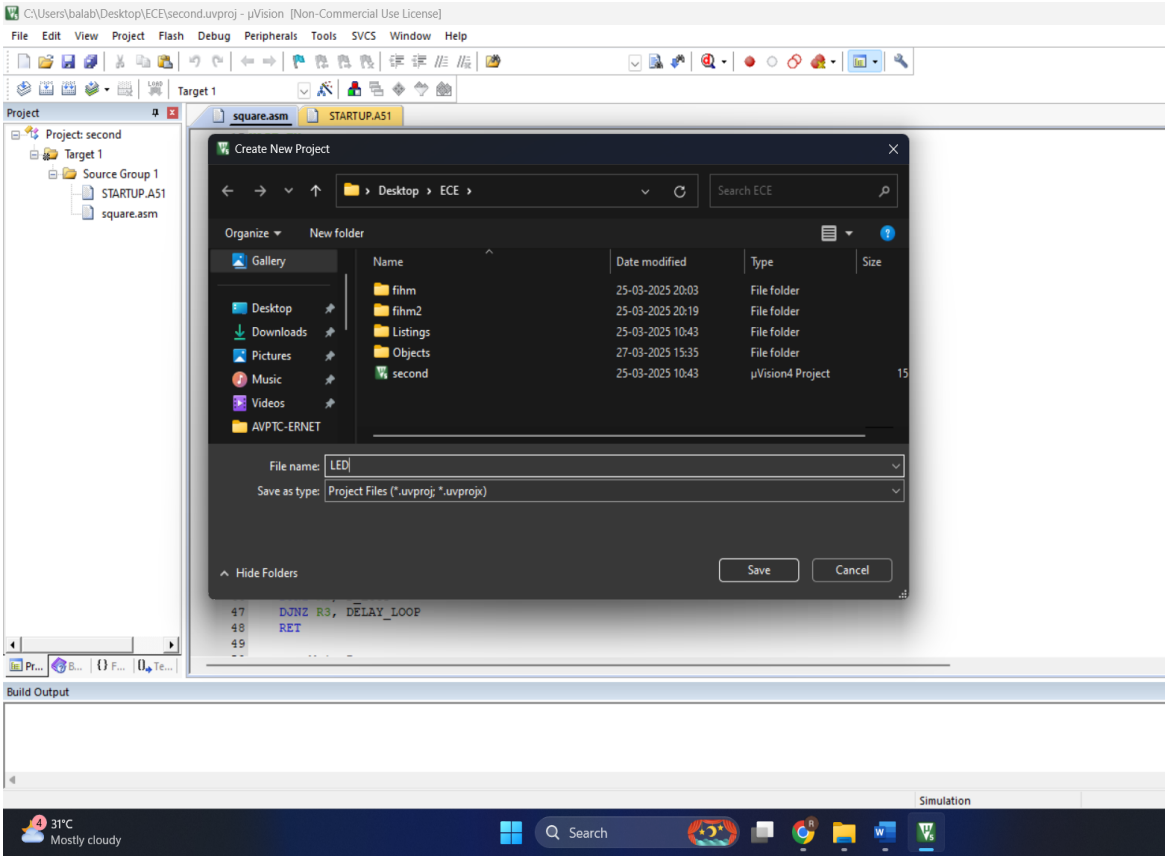
STEP 2: Go to the "**Project**" and click on "**New µvision Project**"



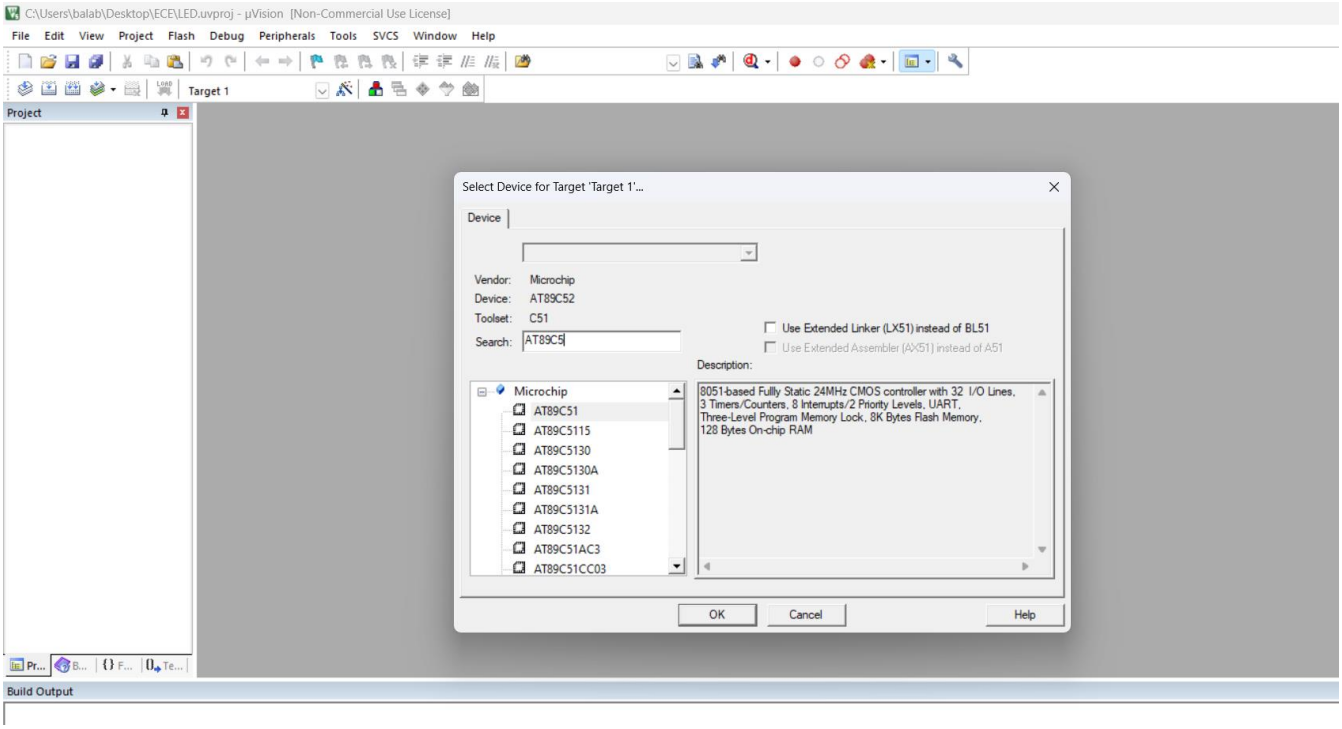
STEP 3: A small window will pop up with the name "**Create New Project**" and can be created and select destination path.



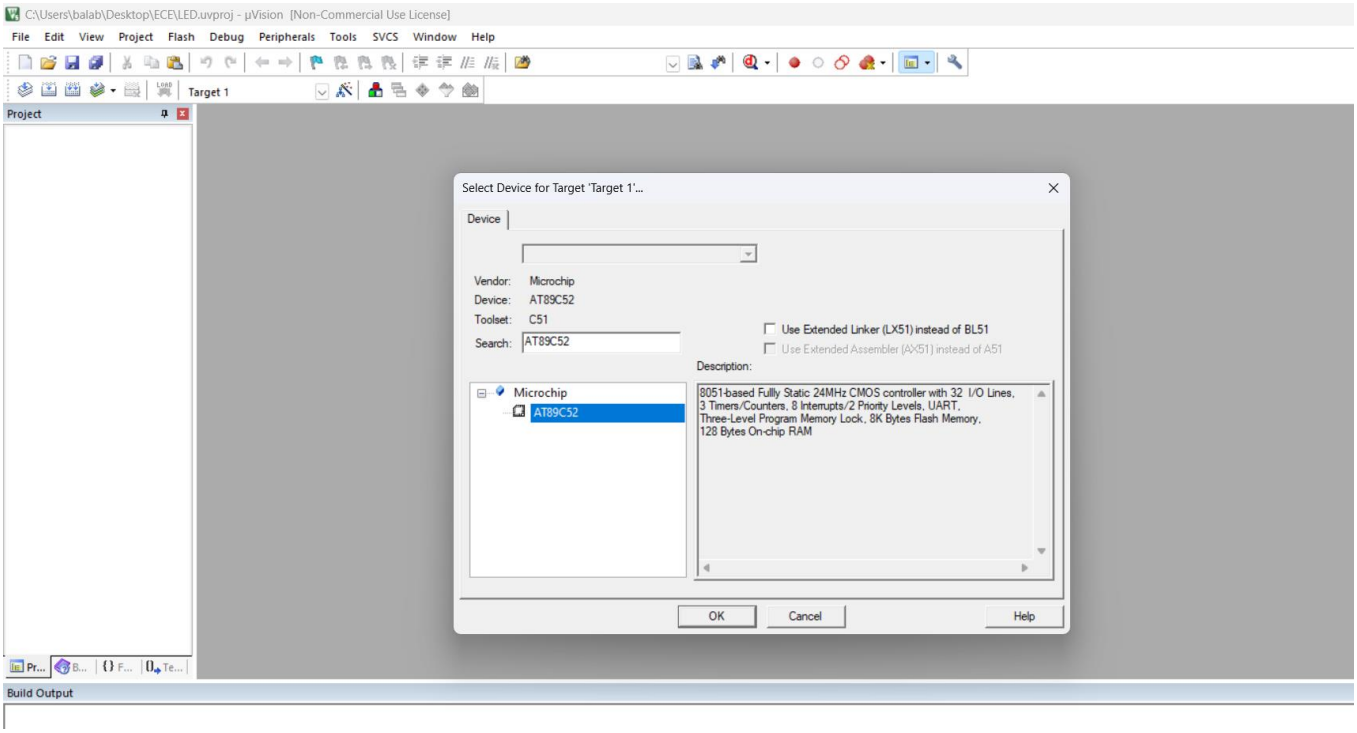
STEP 4: Create a folder and give a proper name that can be related to the Project.



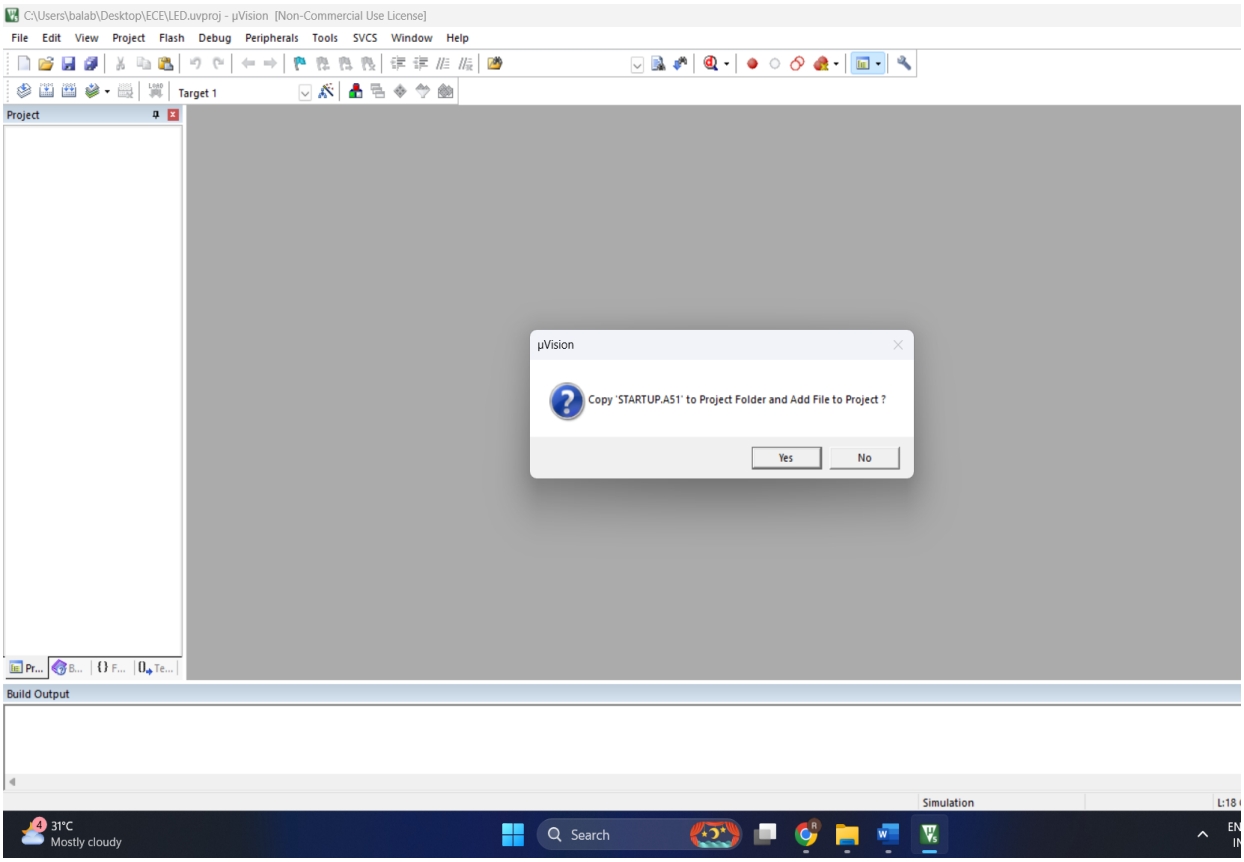
STEP 5: A small window will pop up with the name "**Select Device for Target 'Target 1'**", and select the data base Microchip.



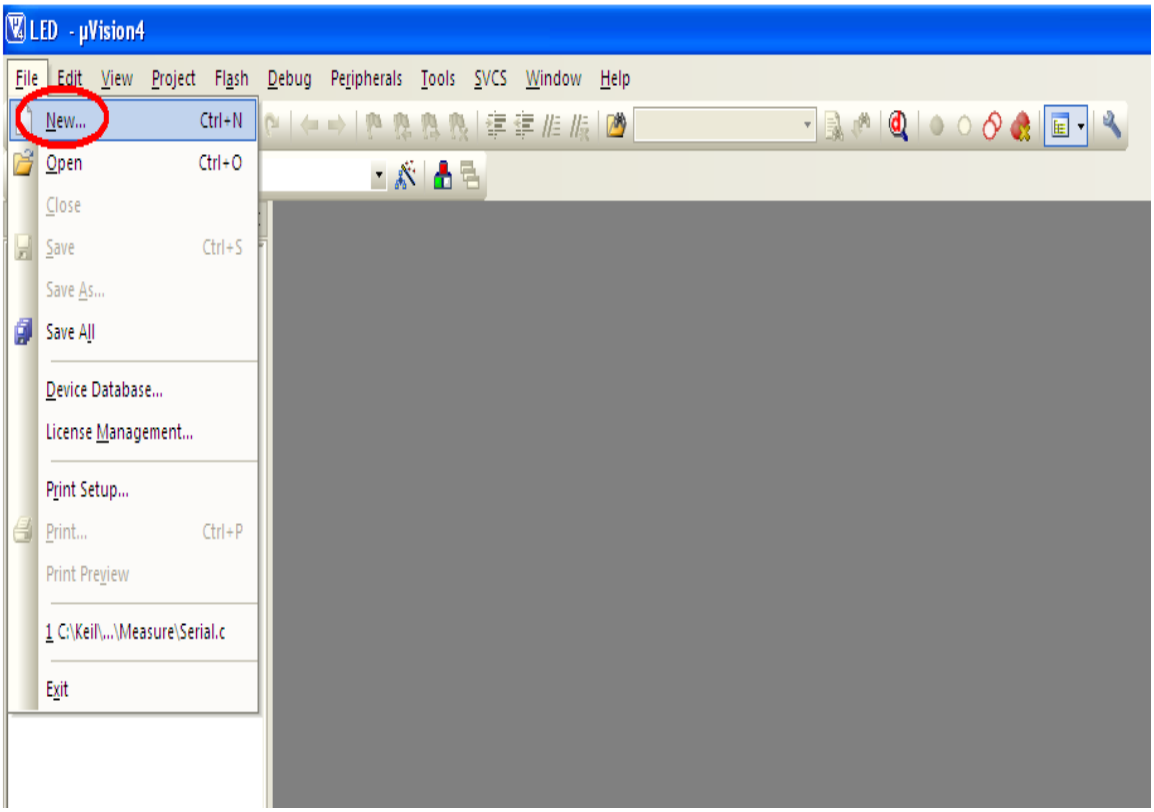
STEP 6: Within the Microchip, select **AT895C2.**



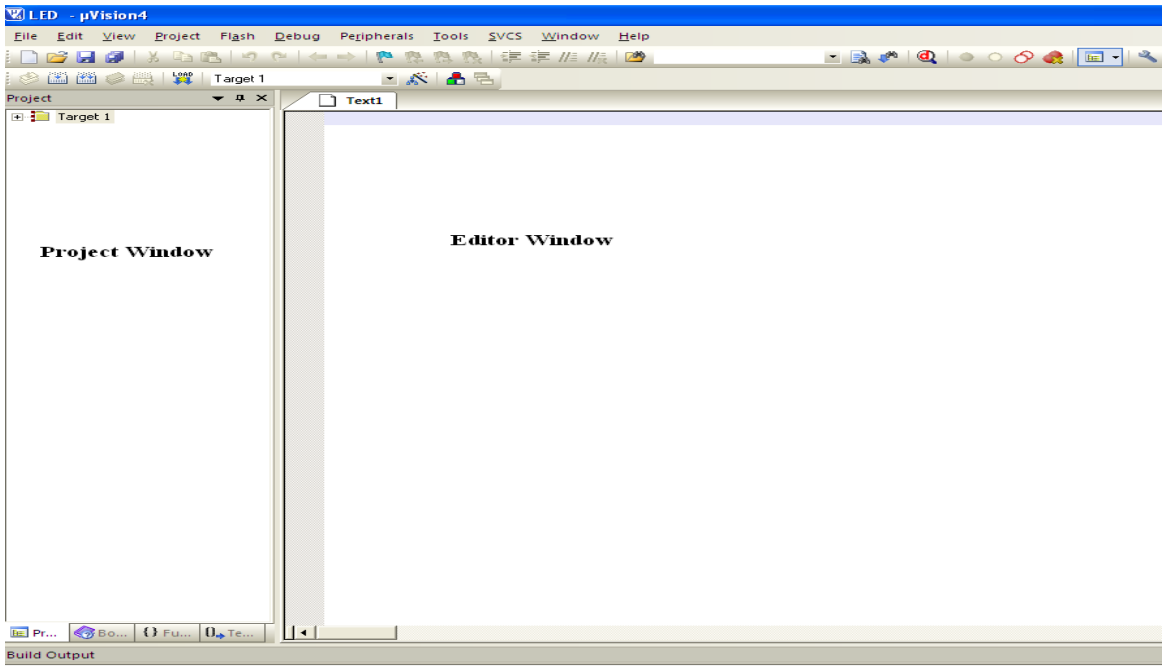
STEP 7: Add Startup file to the project by clicking "**Yes**".



STEP 8: Next go to "**File**" and click "**New**".

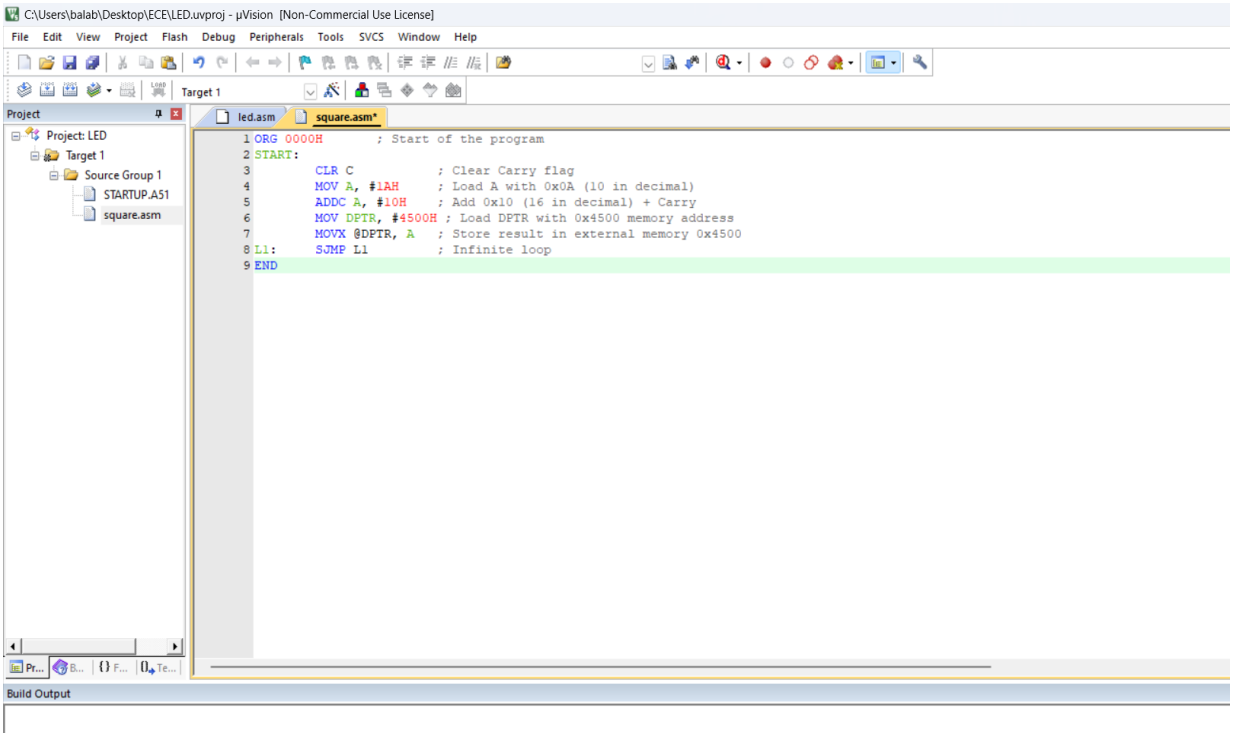


STEP 9: There are the three main windows are available in the keil IDE. First one is Project Workspace, the second one is Editor Window and third one is Output Window.

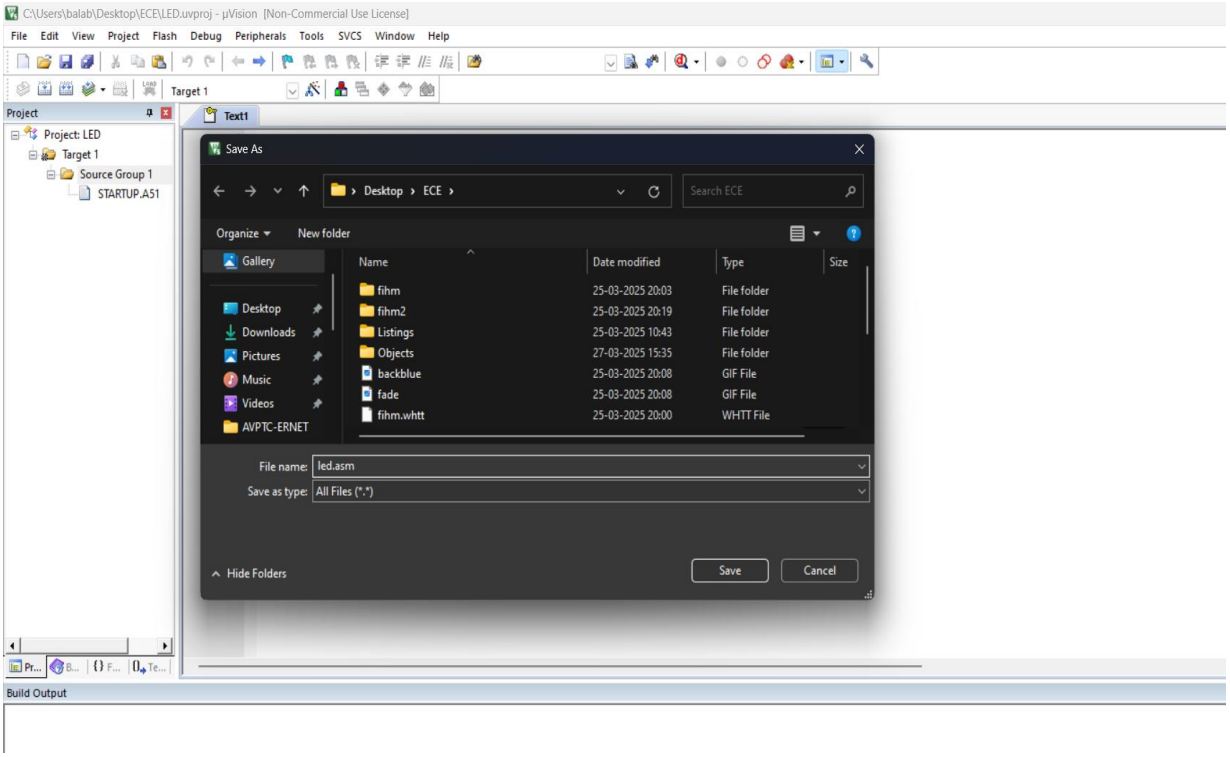


Output Window

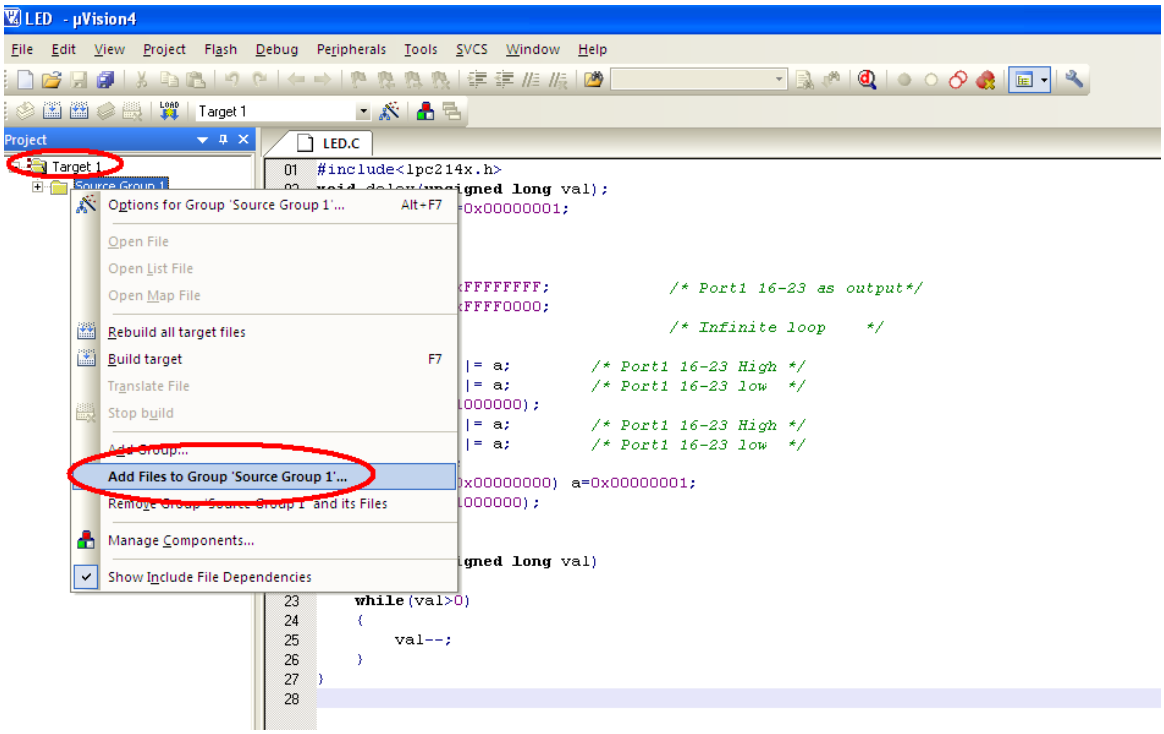
STEP 10: Can start to write *.asm code on the editor window.



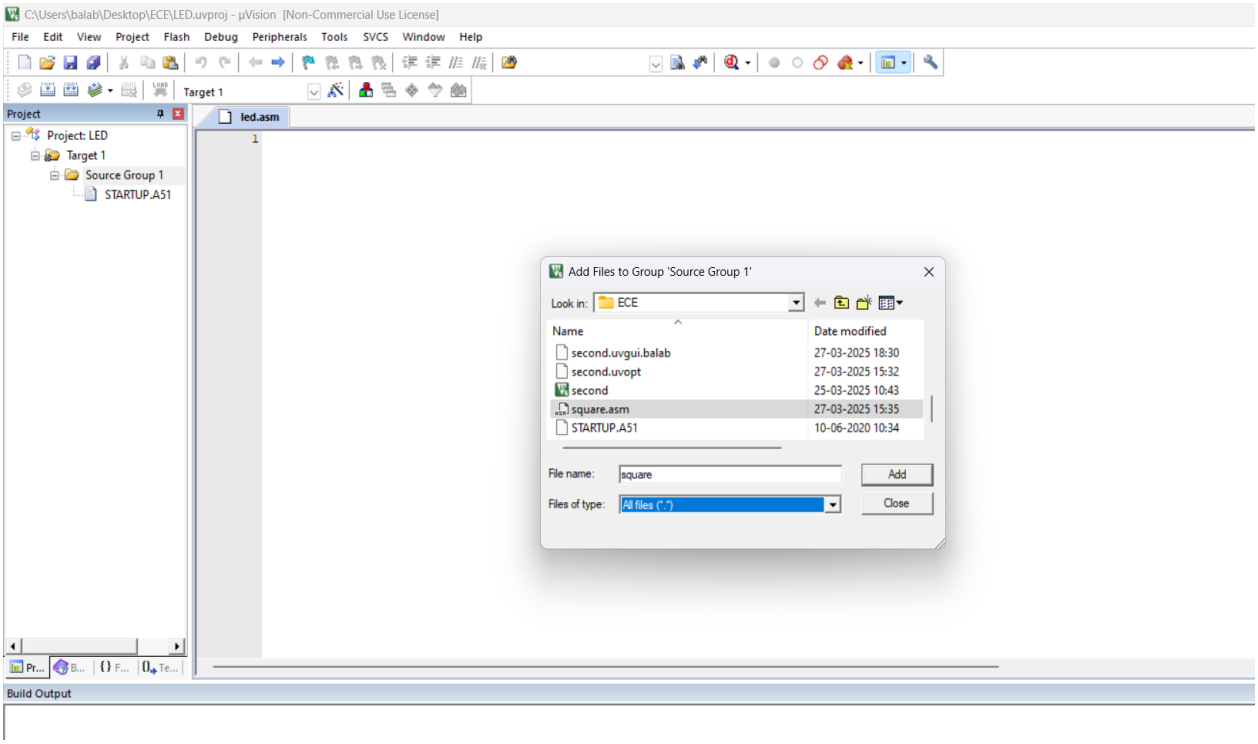
STEP 11: Can save the file, if the program is in "**C**" save as save as "**filename.ASM**".



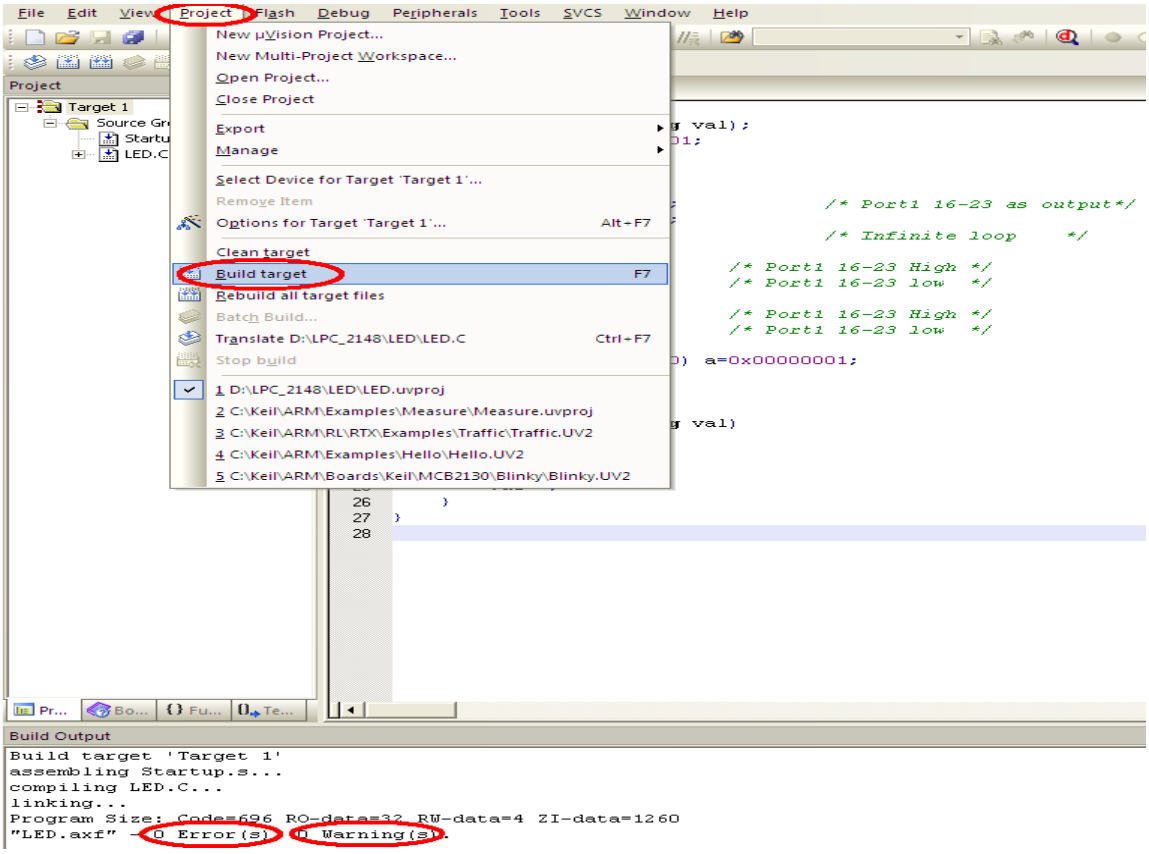
STEP 12: Add this source file to Group1, Go to the "**Project Workspace**" drag the +mark "**Target 1**" in that right click on "**Source Group1**" and click on "**Add Files to Group "Source Group1"**"".



STEP 13: A small window will pop up with the name "**Add Files to Group Source Group1",** by default, the Files of type will be in **All Files (*.asm).** If the program is asm select **ASM Source file (*.s,*.src,*.a*).**

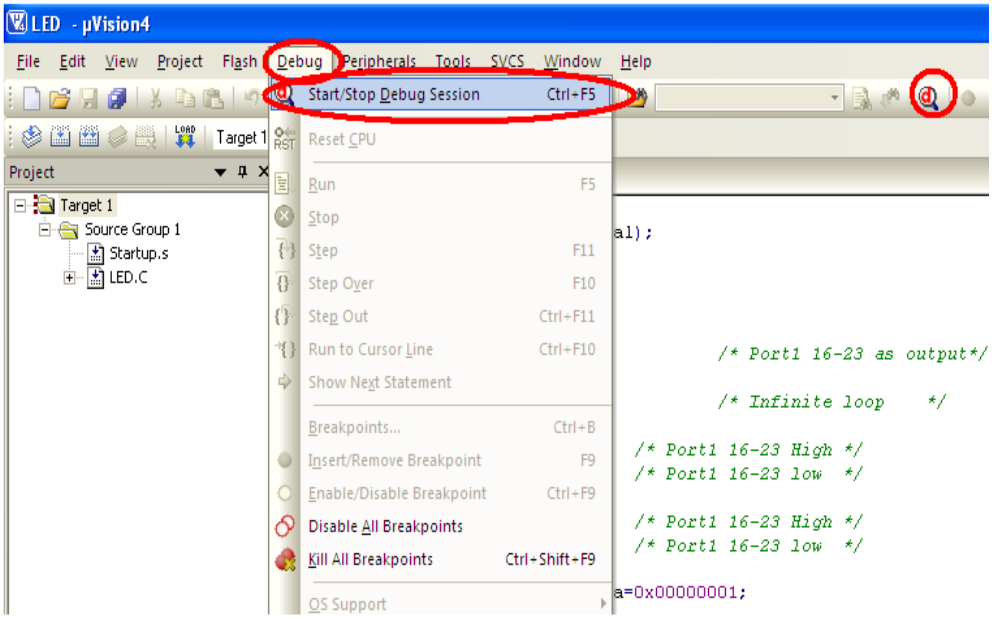


STEP 14: Then go to "**Project**" click on "**Build Target**" or **F7**. Output window will display related error and warning messages.

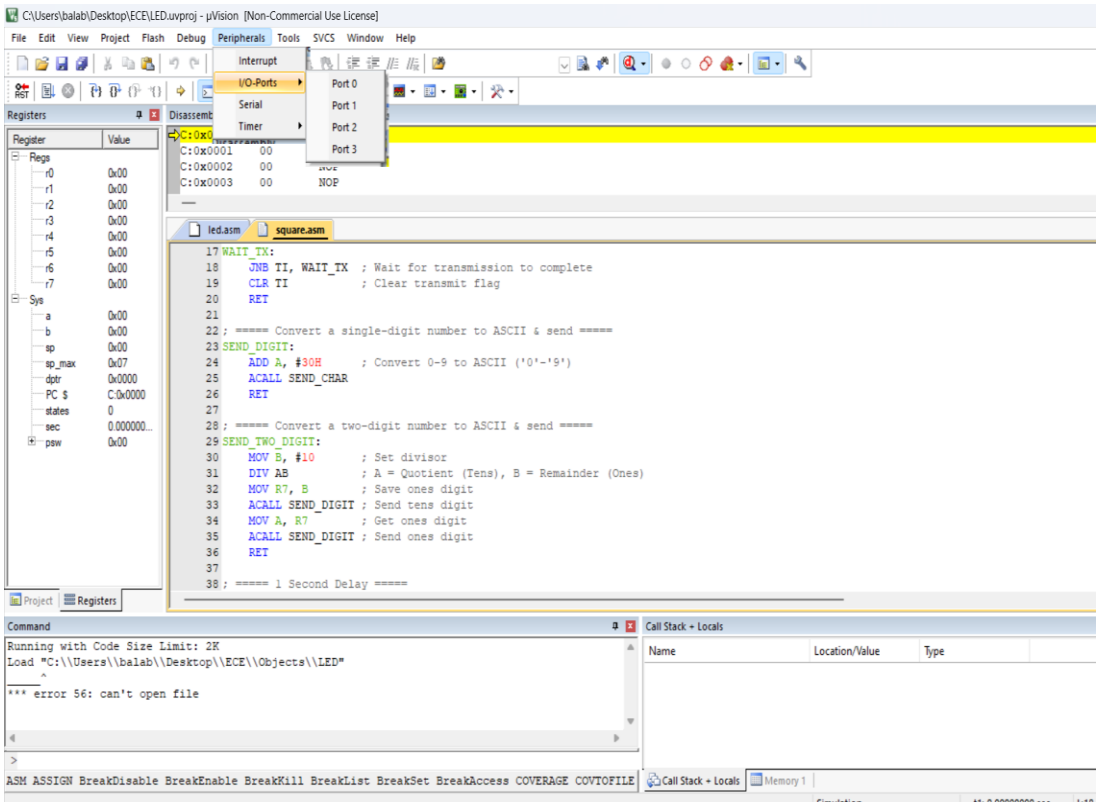


Simulation
Part:

STEP 15: Go to "**Project**" menu, click on "**Rebuild all target Files**" and start **Debug**. From **View** menu can get **Memory Window** and from **Peripherals** can get I/O ports, Serial etc. For access internal memory type **i:0x_memory** location example: **i:0x30** and for external and program memory **x:0x_memory location**, **c:0x_memory location** respectively. From **Register** window we can edit and access the values also.



STEP 16: If Output has to be seen on Port select Peripherals → IO Ports → Select Port



EXPERIMENTS USING 8051

EXP NO:	8051 Assembly Language program using Keil simulator
DATE	

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 µvision5 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 µVision: 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/.

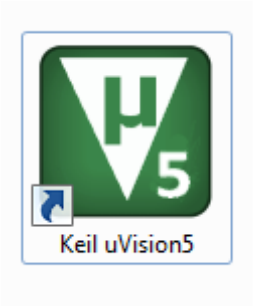
INSTALLATION OF KEIL SOFTWARE

Set up Keil IDE for Programming

Keil µVision 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.

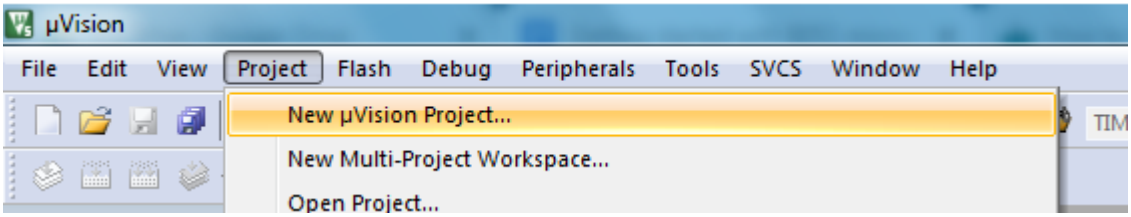
In this article, we will use the AT89C51ED2 microcontroller, which has:

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



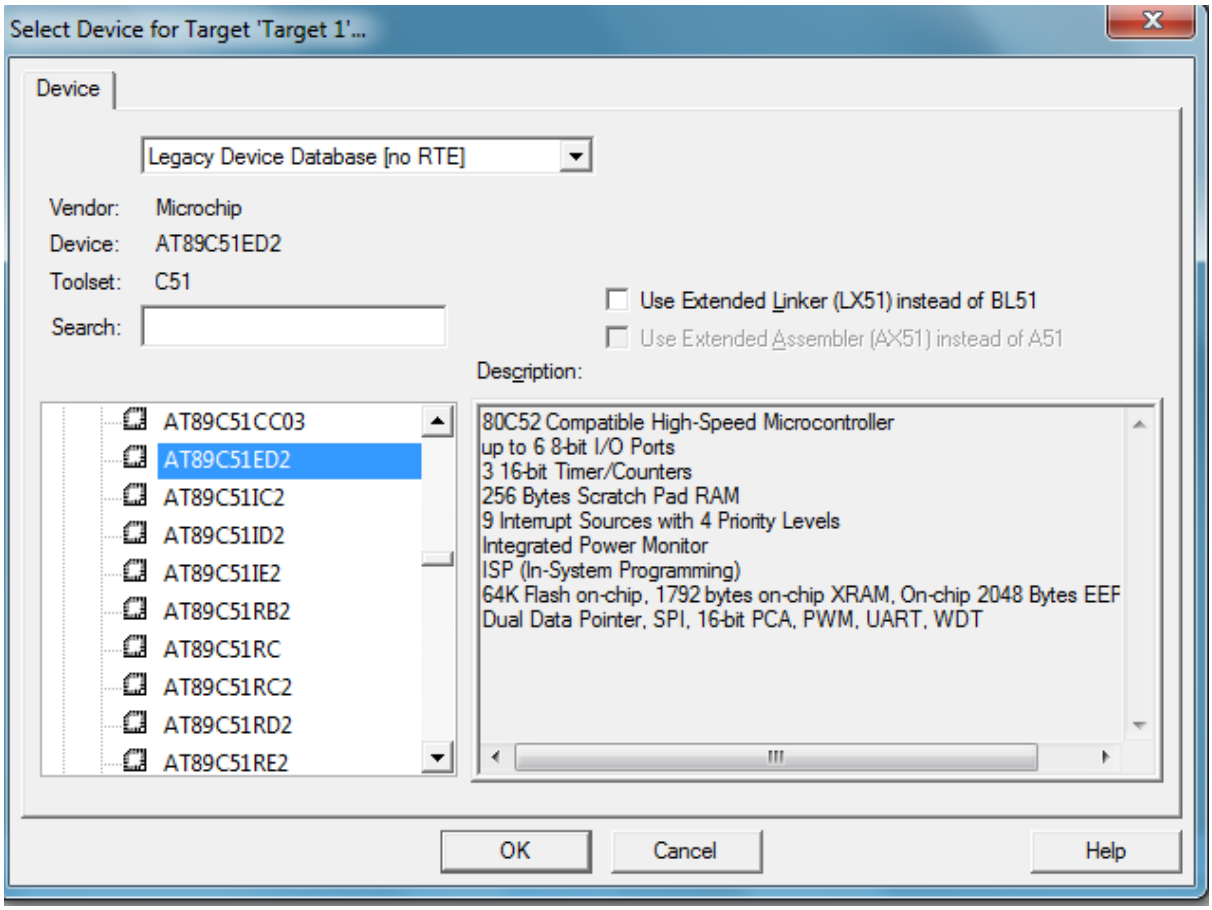
The Keil µVision icon.

To start writing a new program, you need to create a new project. Navigate to **project —> New µVision project**. Then save the new project in a folder.

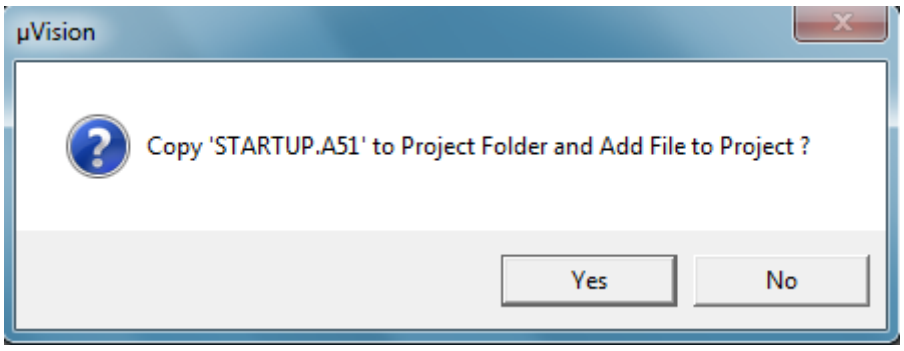


After saving the file, a new window will pop up asking you to select your microcontroller.

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

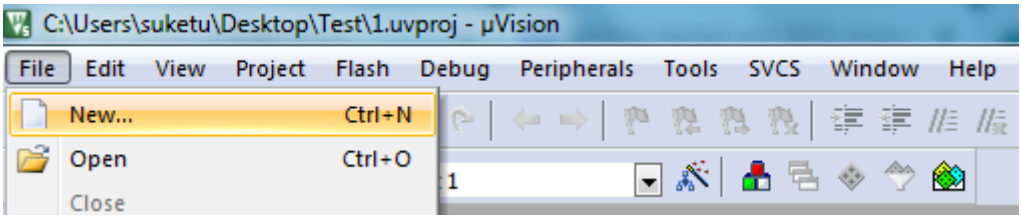


Select '**Yes**' in the next pop-up, as we do not need this file in our project.

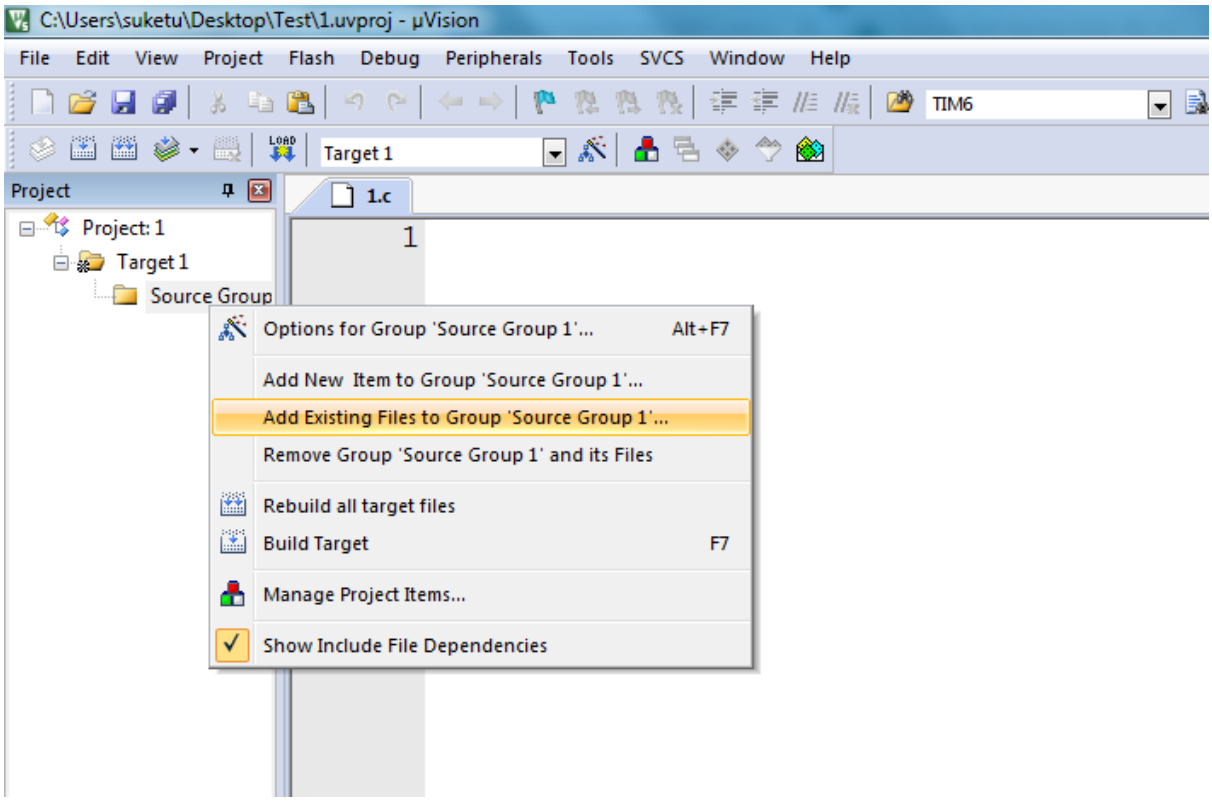


Our project workspace is now ready!

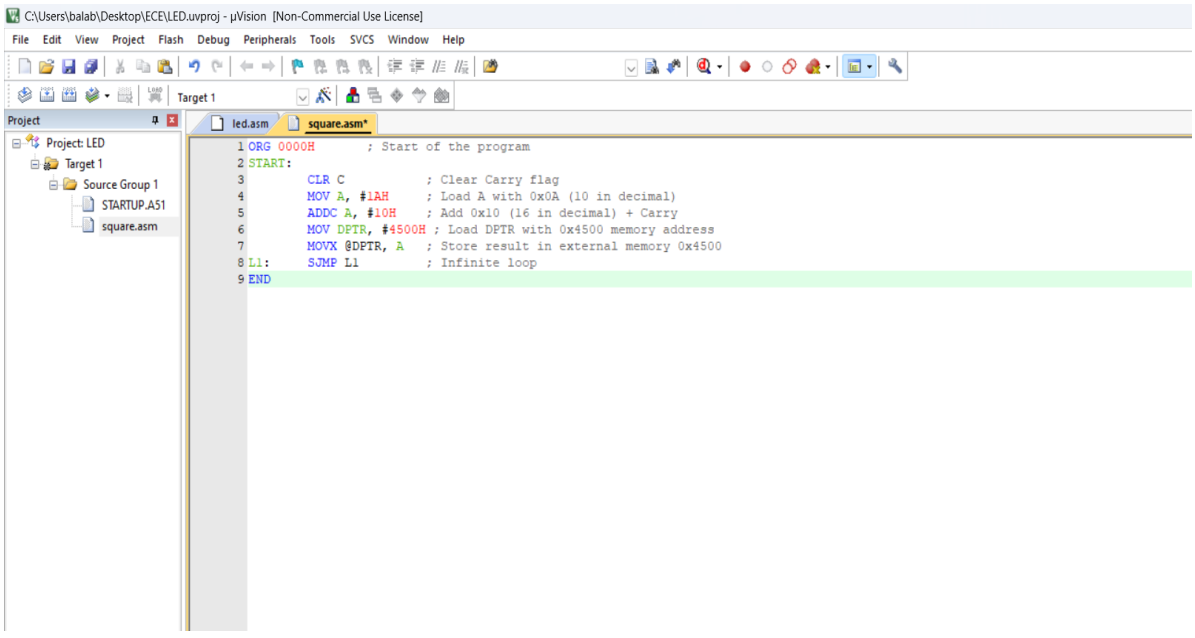
From here, we need to create a file where we can write our C code. Navigate to **File —> New**. Once the file is created, save it with .c extension in the same project folder.



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 μVision Project and Create a New Project, Select the 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 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:

EXP NO:	**Programming Arithmetic and Logical Operations in 8051**
DATE	

AIM:

To write and execute the Arithmetic and Logical program using the Keil simulator.

SOFTWARE TOOLS REQUIRED:

S.No	Software Requirements	Quantity
1	Keil µvision5 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 µVision 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 simulation by clicking Run or step run.

PROGRAM:

8-BIT ADDITION

```
ORG 0000H       ; Start of the program
START:
    CLR C       ; Clear Carry flag
    MOV A, #1AH     ; Load A with 0x0A (10 in decimal)
    ADDC A, #10H    ; Add 0x10 (16 in decimal) + Carry
    MOV DPTR, #4500H ; Load DPTR with 0x4500 memory address
    MOVX @DPTR, A   ; Store result in external memory 0x4500
L1:  SJMP L1        ; Infinite loop
END
```

PROGRAM

8-BIT SUBTRACTION

```
ORG 0000H       ; Start of the program
START:          CLR C          ; Clear Carry flag (used in SUBB)
      MOV A,  #0AH    ; Load A with 0x0A (decimal 10)
      SUBB A, #05H    ; Subtract 0x05 (decimal 5) from A (A = A - 5 - CY)
      MOV DPTR, #4500H ; Load DPTR with address 0x4500H
      MOVX @DPTR, A   ; Store result of subtraction at external memory (XDATA) 0x4500
L1:   SJMP L1         ; Infinite loop to hold execution
END
```

PROGRAM

8-BIT MULTIPLICATION

```
ORG 0000H       ; Start of the program
START:
    MOV A, #05H      ; Load A with 0x05 (decimal 5)
    MOV B, #03H      ; Load B with 0x03 (decimal 3)
    MUL AB           ; Multiply A × B (Result stored in A & B)

    MOV DPTR, #4500H  ; Load DPTR with address 0x4500H
    MOVX @DPTR, A     ; Store the lower byte of the result (A) at 0x4500H

    INC DPTR          ; Increment DPTR to 0x4501H
    MOV A, B          ; Move higher byte of result (B) to A
    MOVX @DPTR, A     ; Store the higher byte at 0x4501H

L1:  SJMP L1          ; Infinite loop to hold execution
END
```

PROGRAM

8-BIT DIVISION

```
ORG 0000H       ; Start of the program
START:
    MOV A, #H       ; Load A with 0x15 (decimal 21)
    MOV B, #03H0E       ; Load B with 0x03 (decimal 3)
    DIV AB          ; Divide A by B (A ÷ B)

    MOV DPTR, #4500H  ; Load DPTR with address 0x4500H
    MOVX @DPTR, A     ; Store the quotient in external memory (XDATA 0x4500)

    INC DPTR          ; Increment DPTR to 0x4501H
    MOV A, B          ; Move the remainder to A
    MOVX @DPTR, A     ; Store the remainder in external memory (XDATA 0x4501)

L1:   SJMP L1          ; Infinite loop to hold execution
END
```

PROGRAM

LOGICAL OPERATION

```
ORG 0000H        ; Start of program
START:
   MOV A, #0F0H   ; Load A with 1111 0000 (F0H)
   MOV R1, #0AAH  ; Load R1 with 1010 1010 (AAH)

   ; Perform AND operation
   ANL A, R1      ; A = A AND R1 (1010 0000)
   MOV DPTR, #4500H
   MOVX @DPTR, A   ; Store AND result in memory

   ; Perform OR operation
   ORL A, R1      ; A = A OR R1 (1010 1010)
   INC DPTR
   MOVX @DPTR, A   ; Store OR result in memory

   ; Perform XOR operation
   XRL A, R1      ; A = A XOR R1 (0000 0000)
   INC DPTR
   MOVX @DPTR, A   ; Store XOR result in memory

   ; Perform NOT (Complement) operation
   MOV A, #55H     ; Load A with 0101 0101 (55H)
   CPL A          ; Complement A (1010 1010)
   INC DPTR
   MOVX @DPTR, A   ; Store NOT result in memory


L1: SJMP L1        ; Infinite loop
END
```

RESULT:

EXP NO:	Generation of Square waveform using 8051
DATE	

AIM:

To write and execute the Generation of Square waveform program using the Keil simulator.

SOFTWARE TOOLS REQUIRED:

S.No	Software Requirements	Quantity
1	Keil µvision5 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 µVision 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 simulation by clicking Run or step run.

PROGRAM:

SQUARE WAVE GENERATION

```
ORG 0000H       ; Start of the program
START:
   MOV P1, #00H  ; Clear Port 1 (All pins LOW)

LOOP:
   SETB P1.0     ; Set P1.0 HIGH (1)
   CALL DELAY    ; Delay to keep HIGH
   CLR P1.0      ; Clear P1.0 LOW (0)
   CALL DELAY    ; Delay to keep LOW
   SJMP LOOP     ; Repeat forever

DELAY:
      MOV R7, #255
D1: MOV R6, #255
D2: DJNZ R6, D2
      DJNZ R7, D1
      RET
END
```

OUTPUT:

RESULT:

EXP NO:	Programming using on–Chip ports in 8051
DATE	

AIM:

To write and execute the Generation of Square waveform program using the Keil simulator.

SOFTWARE TOOLS REQUIRED:

S.No	Software Requirements	Quantity
1	Keil µvision5 IDE	1

THEORY:

Programming the **8051-microcontroller** using **on-chip ports** involves configuring and controlling its **four I/O ports (P0, P1, P2, P3)** for input and output operations. The **8051** has **32 general-purpose I/O pins**, divided into four **8-bit** ports.

8051 I/O Ports Overview

1. **Port 0 (P0.0 - P0.7)**
   - **Dual-purpose:** Acts as both I/O and lower address/data bus (AD0–AD7) in external memory mode.
   - **Requires external pull-up resistors for I/O operations** (open-drain configuration).
2. **Port 1 (P1.0 - P1.7)**
   - **Pure I/O port** (does not have any alternate function).
   - **Internal pull-ups available**.
3. **Port 2 (P2.0 - P2.7)**
   - **Dual-purpose:** Acts as both I/O and higher address bus (A8–A15) in external memory mode.
   - **Internal pull-ups available**.
4. **Port 3 (P3.0 - P3.7)**
   - **Multi-functional I/O port**, supporting functions like serial communication, interrupts, and timers.

Port 3 Alternate Functions

Pin	Function
P3.0	RXD (Serial Input)
P3.1	TXD (Serial Output)
P3.2	INT0 (External Interrupt 0)
P3.3	INT1 (External Interrupt 1)
P3.4	T0 (Timer 0 External Input)
P3.5	T1 (Timer 1 External Input)
P3.6	WR (External Memory Write)
P3.7	RD (External Memory Read)

Port	Pins	Primary Function	Alternate Functions
Port 0	P0.0 - P0.7	General I/O (Open-Drain)	Lower Address/Data Bus (AD0 - AD7) in external memory mode
Port 1	P1.0 - P1.7	General I/O (With internal pull-ups)	No alternate function
Port 2	P2.0 - P2.7	General I/O (With internal pull-ups)	Higher Address Bus (A8 - A15) in external memory mode
Port 3	P3.0 - P3.7	General I/O (With internal pull-ups)	Special Functions

PROCEDURE

1. Create a new project, go to "Project" and close the current project "Close Project".
2. Next, Go to the Project New μVision 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 in the editor window and save it with .asm extension.
6. Add this source file to the Group and click on "Build Target" or F7.
7. Go to debugging mode to see the simulation result by clicking Run or step run.

OUTPUT:

PROGRAM:

PORT AS OUTPUT (LED BLINKING)

```
ORG 0000H
START:
   MOV P1, #00H   ; Clear Port 1 (All Pins LOW)

LOOP:
   SETB P1.0      ; Turn ON LED (P1.0 = 1)
   CALL DELAY     ; Delay
   CLR P1.0       ; Turn OFF LED (P1.0 = 0)
   CALL DELAY     ; Delay
   SJMP LOOP      ; Repeat forever

DELAY:
   MOV R7, #255
D1: MOV R6, #255
D2: DJNZ R6, D2
   DJNZ R7, D1
   RET

END
```

OUTPUT:

PROGRAM:

PORT AS INPUT (SWITCH PRESS DETECTION)

```
ORG 0000H
START:
   MOV P2, #0FFH  ; Configure P2 as Input (Pull-up Mode)
   MOV P1, #00H   ; Clear Port 1

LOOP:
   JB P2.0, LED_OFF  ; If P2.0 = 1 (Not Pressed), Jump
   SETB P1.0         ; If P2.0 = 0 (Pressed), Turn ON LED
   SJMP LOOP

LED_OFF:
   CLR P1.0          ; Turn OFF LED
   SJMP LOOP

END
```

OUTPUT:

PROGRAM:

READING & WRITING DATA ON PORTS

```
ORG 0000H
START:
   MOV P0, #0FFH  ; Configure P0 as Input
   MOV P1, #00H   ; Configure P1 as Output

LOOP:
   MOV A, P0      ; Read Data from Port 0
   MOV P1, A      ; Send Data to Port 1
   SJMP LOOP      ; Repeat

END
```

OUTPUT:

PROGRAM:

INTERRUPT

```
ORG 0000H         ; Reset vector location
SJMP START        ; Jump to main program

ORG 0003H         ; Interrupt vector for External Interrupt 0 (INT0)
AJMP ISR          ; Jump to Interrupt Service Routine (ISR)

START:
   MOV P1, #00H   ; Initialize Port 1 (LED OFF)
   MOV IE, #81H   ; Enable External Interrupt 0 (EX0) (IE = 1000 0001b)
   MOV TCON, #01H ; Set INT0 to falling edge triggered mode (TCON = 0000 0001b)
   SETB P3.2      ; Configure P3.2 as input (External Interrupt 0)

MAIN_LOOP:
   SJMP MAIN_LOOP ; Stay in an infinite loop, waiting for interrupts

ISR:
   CPL P1.0       ; Toggle LED (Complement P1.0)
   RETI           ; Return from interrupt

END
```

RESULT:

EXP NO:	Programming using Serial Ports in 8051.
DATE	

AIM:

To write and execute the Serial Port program using the Keil simulator.

SOFTWARE TOOLS REQUIRED:

S.No	Software Requirements	Quantity
1	Keil µvision5 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 µVision 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 simulation by clicking Run or step run.

PROGRAM:

```
ORG 0000H
   ; Setup Timer1 for Serial Communication (9600 Baud Rate)
   MOV TMOD, #20H      ; Timer1 Mode 2 (Auto-Reload), Timer0 Mode 1
   MOV TH1, #0FDH      ; Load TH1 for 9600 baud rate (11.0592 MHz Crystal)
   MOV SCON, #50H      ; Serial Mode 1, 8-bit data
   SETB TR1            ; Start Timer 1

SEND_LOOP:
   MOV SBUF, #'A'      ; Load 'A' into SBUF
WAIT_TX:
   JNB TI, WAIT_TX     ; Wait for transmission to complete
   CLR TI              ; Clear transmit flag
   CALL TIMER0_DELAY_10S

   SJMP SEND_LOOP      ; Repeat transmission

; ******************** 10-Second Timer0 Delay Subroutine ********************
TIMER0_DELAY_10S:
   MOV R7, #20         ; Repeat Timer 0 Overflow 20 times (for ~10s delay)
DELAY_LOOP:
   MOV TMOD, #21H      ; Timer0 Mode 1 (16-bit), Timer1 unchanged
   MOV TH0, #3CH       ; Load high byte for 50ms delay
   MOV TL0, #0B0H      ; Load low byte (Values for 11.0592 MHz)
   SETB TR0            ; Start Timer0

WAIT_TIMER:
   JNB TF0, WAIT_TIMER ; Wait for Timer0 Overflow
   CLR TF0             ; Clear Timer0 Overflow Flag
   CLR TR0             ; Stop Timer0

   DJNZ R7, DELAY_LOOP ; Repeat until 10 seconds complete
   RET                 ; Return from subroutine

END
```

OUTPUT:

RESULT:

EXP NO:	Design of a Digital Clock using Timers/Counters in 8051.
DATE	

AIM:

To write and execute the Design of a Digital Clock using Timers & Counters program using the Keil simulator.

SOFTWARE TOOLS REQUIRED:

S.No	Software Requirements	Quantity
1	Keil µvision5 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 µVision 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 simulation by clicking Run or step run.

OUTPUT:

PROGRAM:

COUNTERS IN 8051

```
ORG 0000H
   MOV TMOD, #05H   ; Timer0 in Mode 1 (16-bit counter mode)
   SETB TCON.4      ; Set T0 (Enable external counting)
   MOV P1, #00H     ; Clear P1 to display count

COUNT_LOOP:
   MOV A, TL0       ; Read lower byte of count
   MOV P1, A        ; Display on Port 1
   SJMP COUNT_LOOP  ; Repeat

END
```

OUTPUT:

PROGRAM

DESIGN OF A DIGITAL CLOCK USING TIMERS

```
ORG 0000H
   SJMP MAIN

ORG 0030H

; ===== UART Initialization =====
UART_INIT:
   MOV TMOD, #20H   ; Timer1 Mode 2 (8-bit auto-reload)
   MOV TH1, #0FDH   ; Baud rate 9600
   MOV SCON, #50H   ; Serial Mode 1, 8-bit UART
   SETB TR1         ; Start Timer1
   RET

; ===== Send a character to UART =====
SEND_CHAR:
   MOV SBUF, A      ; Move data to UART buffer
WAIT_TX:
   JNB TI, WAIT_TX  ; Wait for transmission to complete
   CLR TI           ; Clear transmit flag
   RET

; ===== Convert a single-digit number to ASCII & send =====
SEND_DIGIT:
   ADD A, #30H      ; Convert 0-9 to ASCII ('0'-'9')
   ACALL SEND_CHAR
   RET

; ===== Convert a two-digit number to ASCII & send =====
SEND_TWO_DIGIT:
   MOV B, #10       ; Set divisor
   DIV AB           ; A = Quotient (Tens), B = Remainder (Ones)
   MOV R7, B        ; Save ones digit
   ACALL SEND_DIGIT ; Send tens digit
   MOV A, R7        ; Get ones digit
   ACALL SEND_DIGIT ; Send ones digit
   RET

; ===== 1 Second Delay =====
DELAY_1S:
```

```
    MOV R3, #20
DELAY_LOOP:
    MOV R2, #250
    MOV R1, #250
D_LOOP:
    DJNZ R1, D_LOOP
    DJNZ R2, D_LOOP
    DJNZ R3, DELAY_LOOP
    RET

; ===== Main Program =====
MAIN:
    ACALL UART_INIT

    MOV R6, #0  ; Hours (0-23)
    MOV R5, #0  ; Minutes (0-59)
    MOV R4, #0  ; Seconds (0-59)

LOOP:
    ; Send Hours
    MOV A, R6
    ACALL SEND_TWO_DIGIT
    MOV A, #':'
    ACALL SEND_CHAR

    ; Send Minutes
    MOV A, R5
    ACALL SEND_TWO_DIGIT
    MOV A, #':'
    ACALL SEND_CHAR

    ; Send Seconds
    MOV A, R4
    ACALL SEND_TWO_DIGIT

    ; New Line for UART Output
    MOV A, #13          ; Carriage return (CR)
    ACALL SEND_CHAR
    MOV A, #10          ; Line feed (LF)
    ACALL SEND_CHAR

    ; 1 Second Delay
    ACALL DELAY_1S
```

```
    ; Increment Time Logic
    INC R4
    CJNE R4, #60, LOOP
    MOV R4, #0
    INC R5
    CJNE R5, #60, LOOP
    MOV R5, #0
    INC R6
    CJNE R6, #24, LOOP
    MOV R6, #0

    SJMP LOOP

END
```

RESULT:

EXPERIMENTS USING ARM

EXP NO:	INTRODUCTION TO RASPBERRY PI PICO W
DATE	

Introduction to Raspberry Pi Pico W:

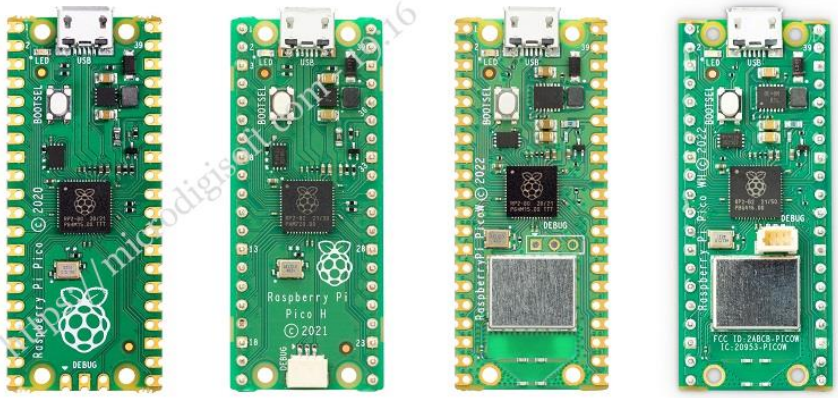
The Raspberry Pi Pico W is a compact and affordable microcontroller board developed by the Raspberry Pi Foundation. Building upon the success of the Raspberry Pi Pico, the Pico W variant brings wireless connectivity to the table, making it an even more versatile platform for embedded projects. In this 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 21 mm x 51 mm '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 **ARM 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 many 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:

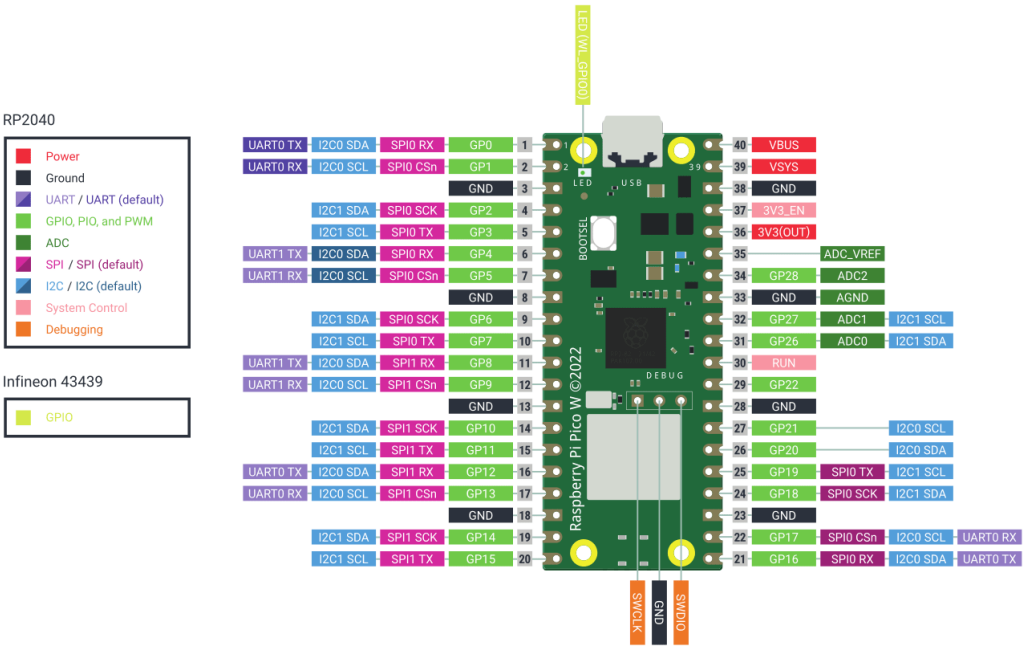
The Raspberry Pi Pico W can be programmed using MicroPython, a beginner-friendly programming language that allows for rapid prototyping and development. MicroPython provides a simplified syntax and high-level abstractions, making it easy for newcomers to get started. Additionally, the Pico W is compatible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



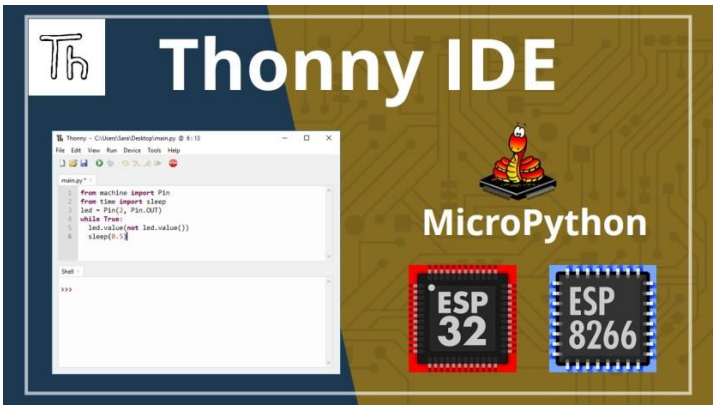
The Raspberry Pi Pico W brings wireless connectivity to the popular Raspberry Pi Pico microcontroller board. With its powerful RP2040 microcontroller, built-in Wi-Fi chip, extensive GPIO capabilities, and compatibility with MicroPython and C/C++ programming, the Pico W offers a versatile and affordable 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:

EXP NO:	INTRODUCTION TO PYTHON PROGRAMMING
DATE	

Getting Started with Thonny MicroPython (Python) IDE:

If you want to program your RP2040, 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.



We've experimented with several IDEs to program the RP2040, ESP32 and ESP8266 boards using MicroPython, and Thonny seemed a good choice. Although there are some bugs, it is constantly being updated and improved.

It allows you to program your RP2040, ESP32 and ESP8266 boards with MicroPython, and it is compatible with Windows, Mac OS X, and Linux. It even comes installed by default on the Raspberry Pi OS. Additionally, it's easy to install, so you shouldn't have problems with the installation process.

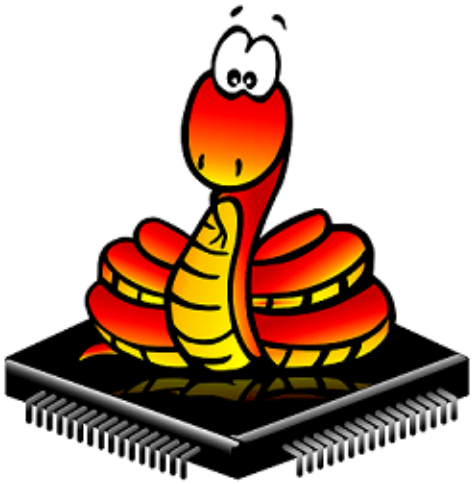
Alternatively, you may want to check the following compilation of MicroPython-compatible IDEs:

- MicroPython IDEs

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

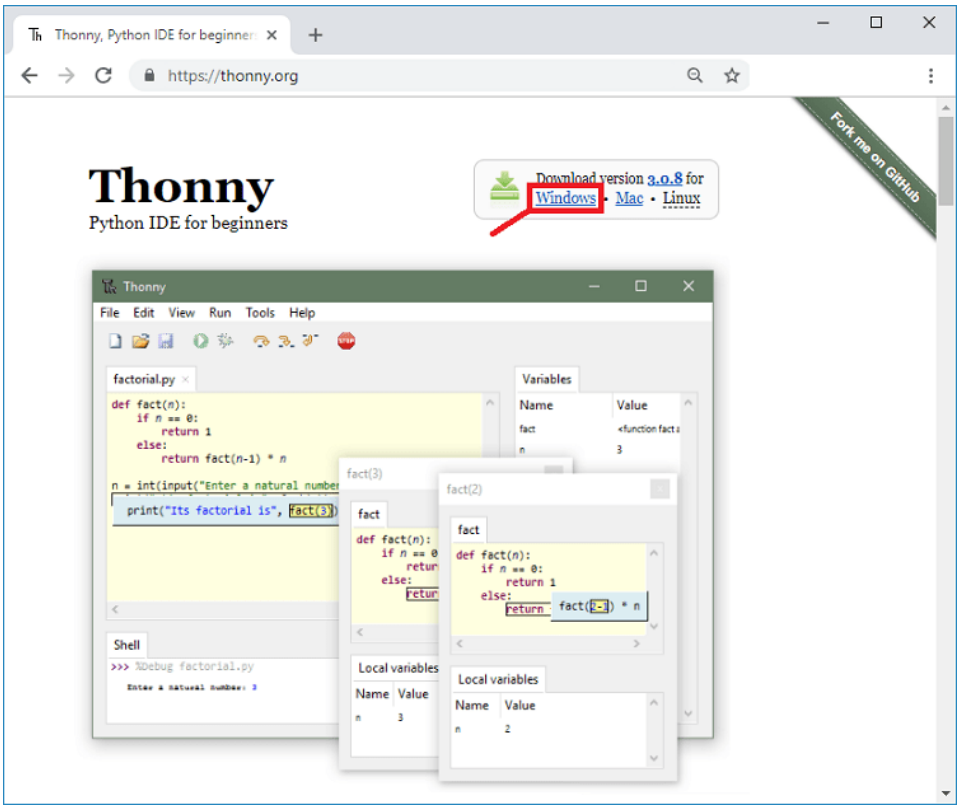
A) Installing Thonny IDE – Windows PC

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

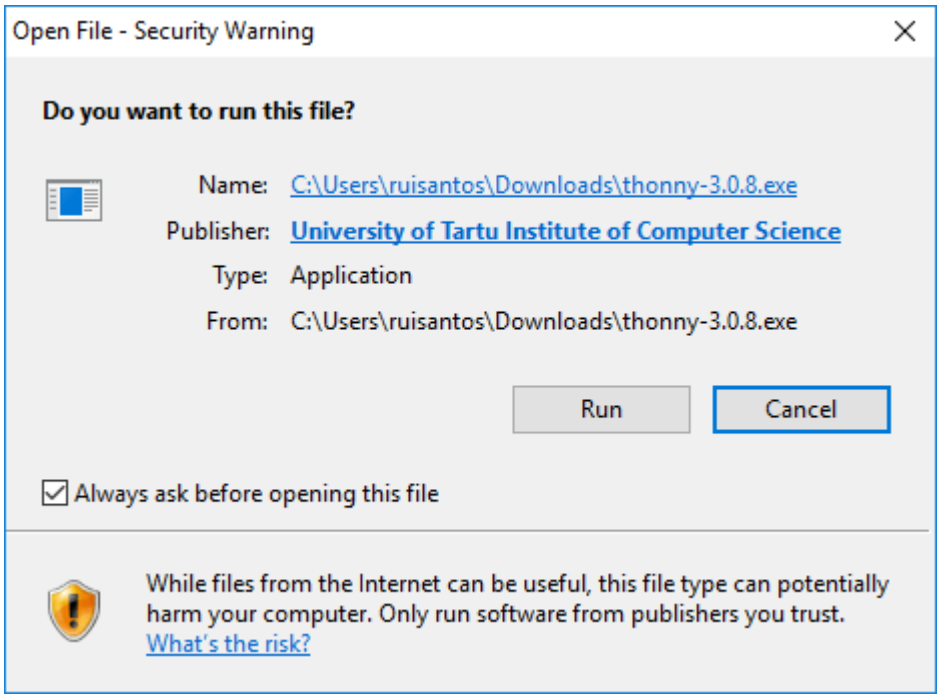
To install Thonny on your Windows PC, follow the next instructions:

1. Go to https://thonny.org

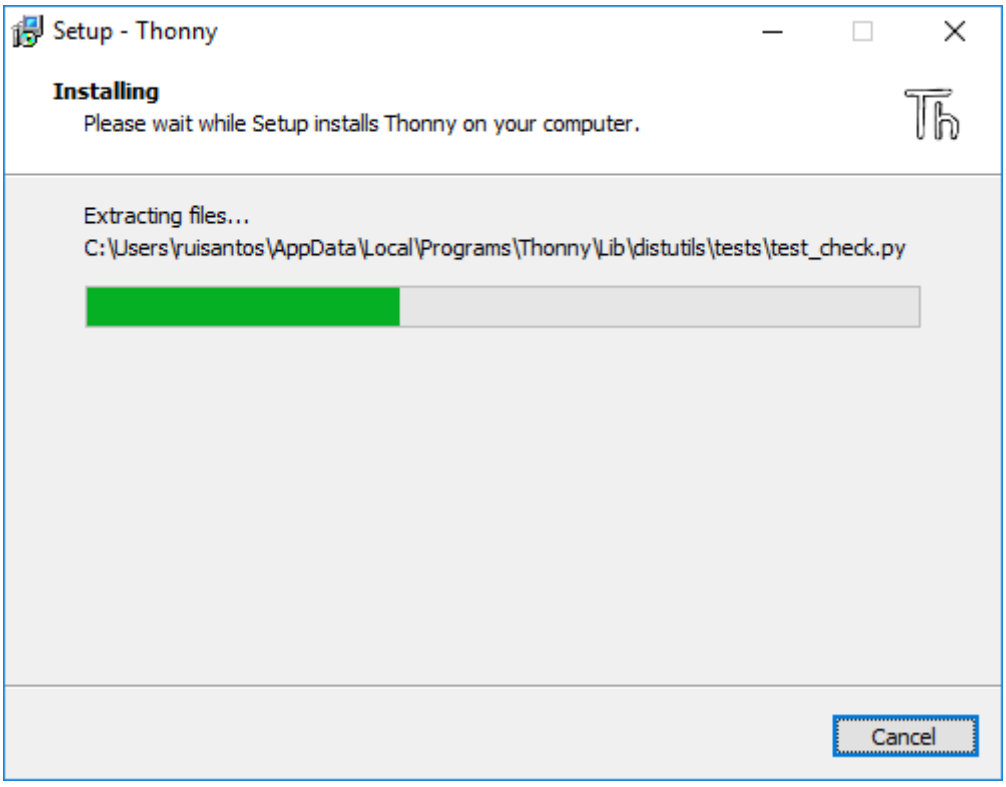
2. Download the version for Windows and wait a few seconds while it downloads.



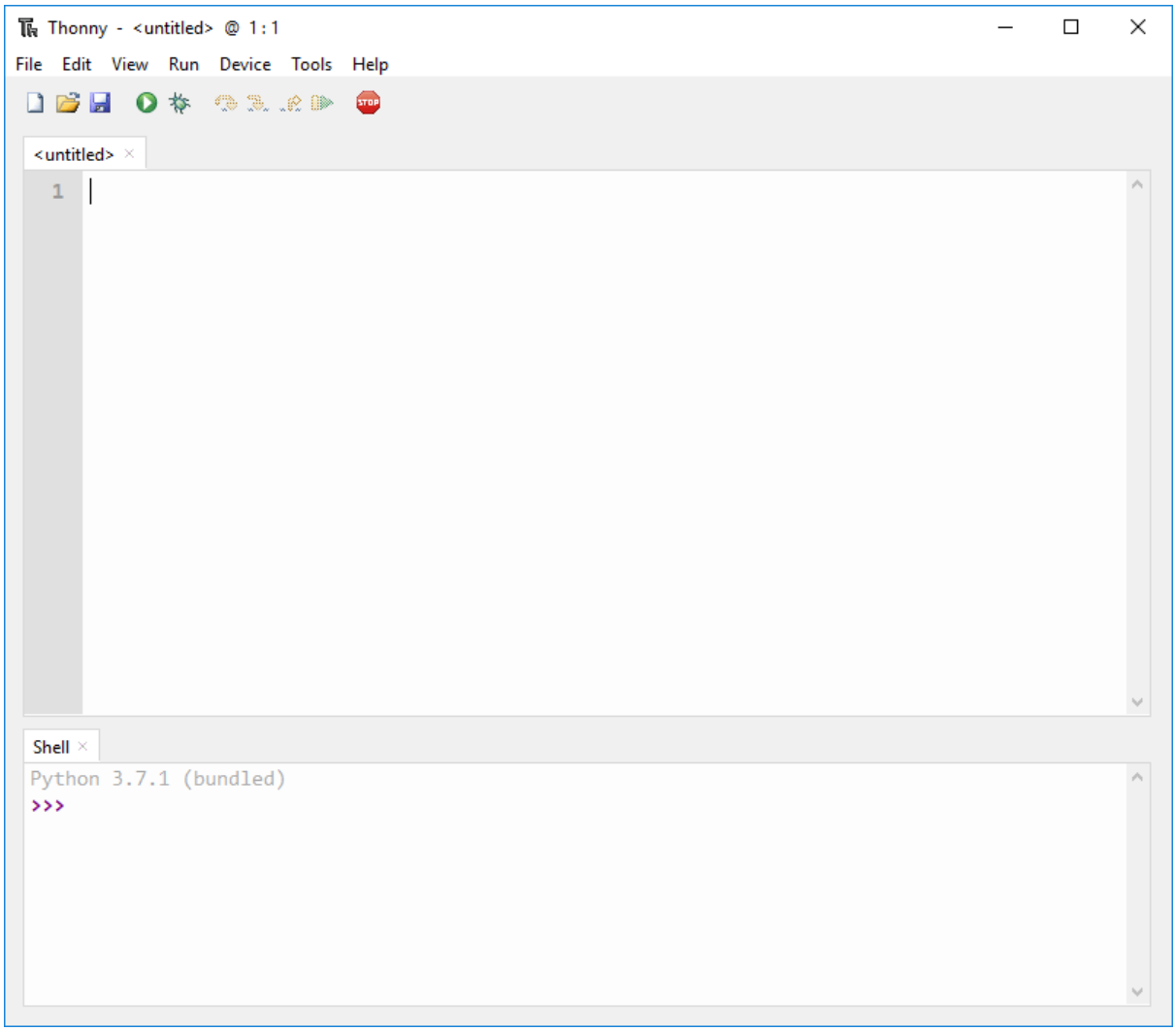
3. Run the *.exe* file.



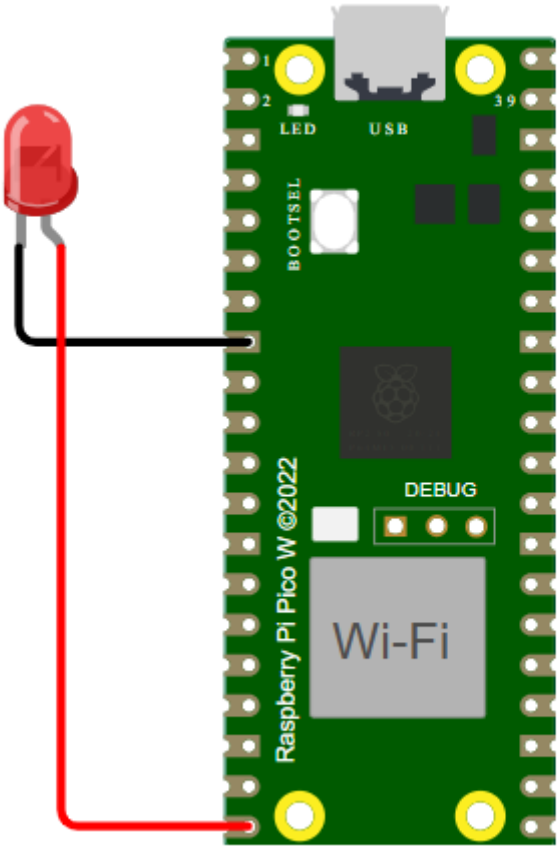
4. Follow the installation wizard to complete the installation process. You just need to click "Next".



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


Thonny - <untitled> @ 1 : 1
File Edit View Run Device Tools Help
<untitled>
1
Shell
Python 3.7.1 (bundled)
>>>

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board
GP16	LED

PROGRAM:

LED:

```
from machine import Pin
import time
LED = Pin(15, Pin.OUT)
while True:
    LED.value(1)
    time.sleep(1)
    LED.value(0)
    time.sleep(1)
```

RESULT:

EXP NO:	Blinking of LEDs and LCD
DATE	

AIM:

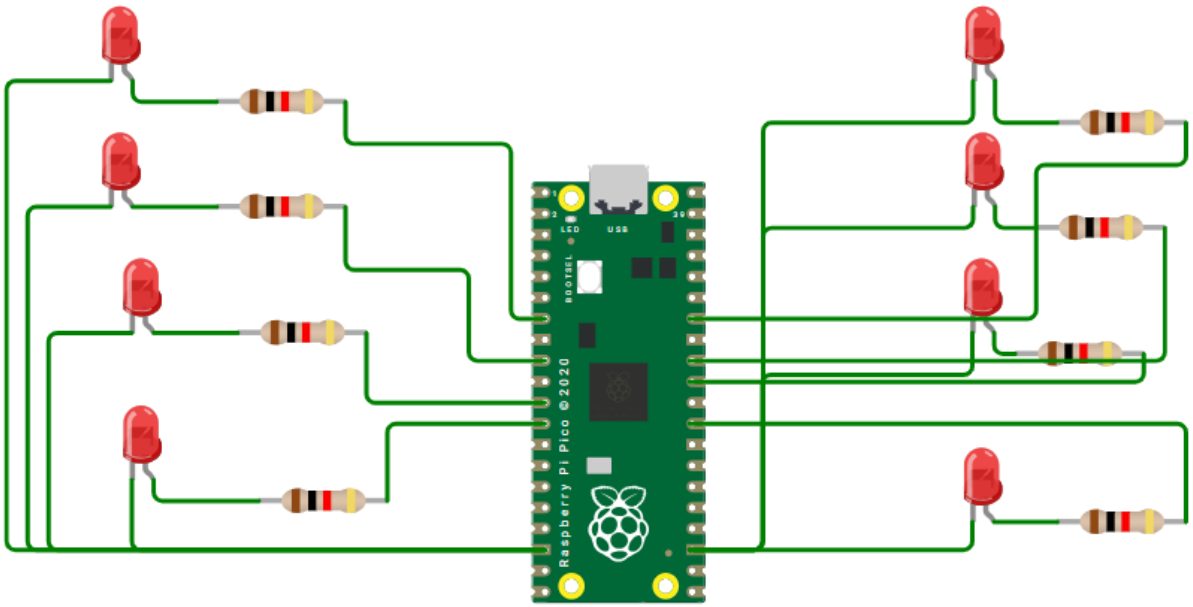
To interface the LED and LCD with the Raspberry Pi Pico W.

HARDWARE & SOFTWARE TOOLS REQUIRED:

S.No	Hardware & Software Requirements	Quantity
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	1
3	Jumper Wires	few
4	Micro USB Cable	1
5	I2C 16X2 LCD	1

PROCEDURE:

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board
GP5	LED1
GP6	LED2
GP8	LED3
GP9	LED4
GP28	LED5
GP27	LED6
GP26	LED7
GP22	LED8

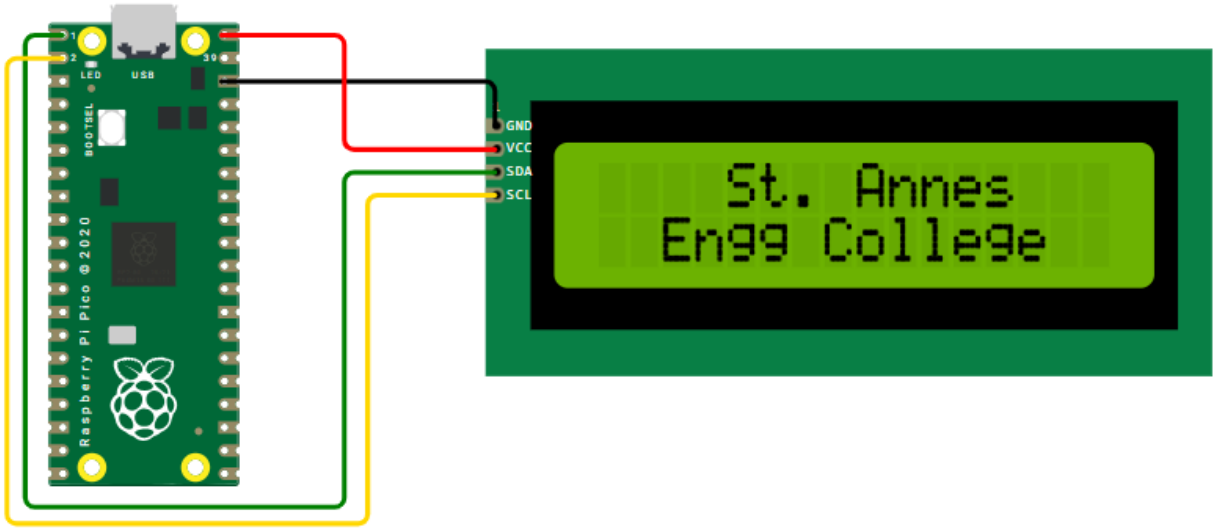
https://wokwi.com/projects/426591556468307969

PROGRAM:

LED:

```
from machine import Pin
import time

# Define LED pins
led_pins = [5, 6, 8, 9, 28, 27, 26, 22]

# Initialize LEDs as output
leds = [Pin(pin, Pin.OUT) for pin in led_pins]

while True:
    for led in leds:
        led.value(1)  # Turn on LED
        time.sleep(0.5)  # Delay
        led.value(0)  # Turn off LED
    time.sleep(0.5)  # Pause before the next cycle
```

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board	LCD Module
GP0	-	SDA
GP1	-	SCL
-	VCC (Or) 5V	VCC / 5V
-	GND	GND

https://wokwi.com/projects/426591739640935425

PROGRAM:

LCD:

```
from machine import I2C, Pin
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)  # 2 rows, 16 columns LCD

while True:
    lcd.clear()
    lcd.move_to(4, 0)
    lcd.putstr("St. Annes")

    lcd.move_to(2, 1)
    lcd.putstr("Engg College")

    sleep(5)
    lcd.clear()
    sleep(1)
```

RESULT:

EXP NO:	Interfacing ADC and DAC
DATE	

AIM:

To interface the ADC and DAC with the Raspberry Pi Pico W.

HARDWARE & SOFTWARE TOOLS REQUIRED:

S.No	Hardware & Software Requirements	Quantity
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	1
3	Jumper Wires	few
4	Micro USB Cable	1
5	Joystick Module	1

THEORY:

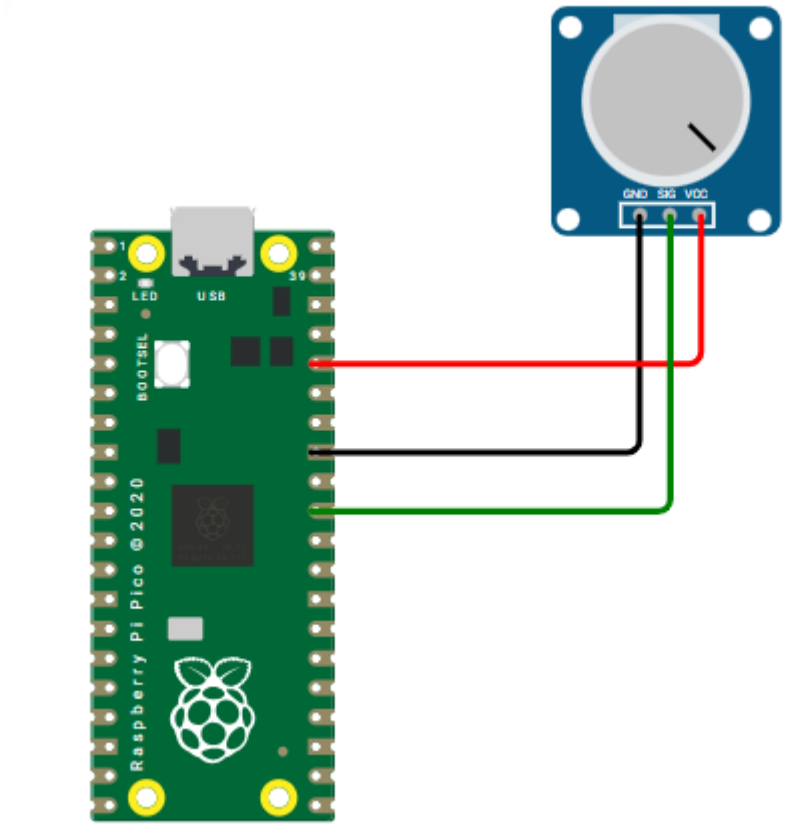
ADC (Analog-to-Digital Converter)

- The Raspberry Pi Pico has three 12-bit ADC channels: ADC0, ADC1, and ADC2.
- The voltage range is 0V to 3.3V.
- It converts analog voltage to a digital value (0 - 4095 for 12-bit resolution).

DAC (Digital-to-Analog Converter)

- The MCP4725 DAC module can be used with the I2C protocol.
- It receives a digital value and outputs an equivalent analog voltage.
- The output voltage is given by:
    - $Vout = DigitalValue \times Vref$
        - 4095
- where V_ref is typically 3.3V.

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board	Joystick Module
GP26	-	Vrx
-	VCC (Or) 5V	VCC / 5V
-	GND	GND

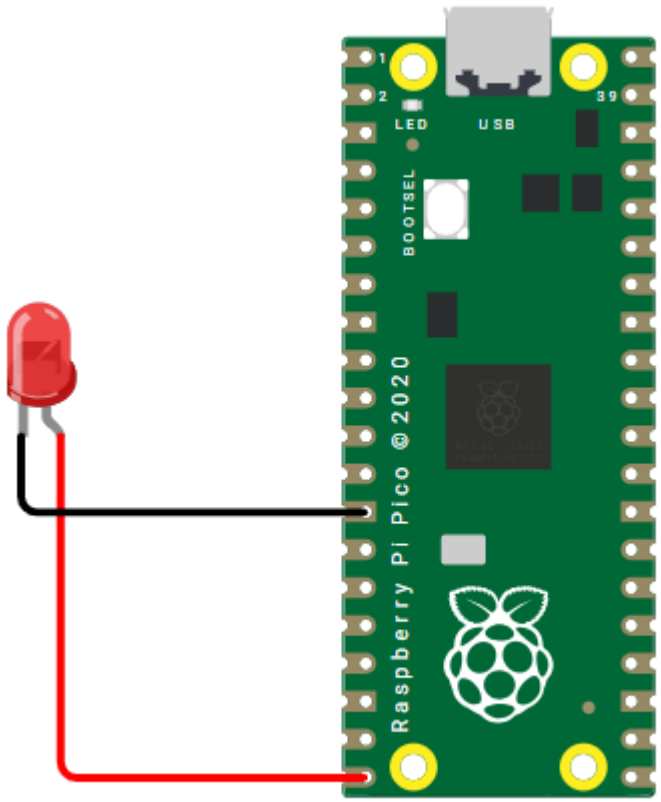
https://wokwi.com/projects/426594618555040769

PROGRAM:

ADC:

```
from machine import ADC
import time
pot= ADC(26)
VREF = 3.3

while True:
    # Convert to 12-bit (0-4095)
    x_val = pot.read_u16() >> 4

    # Convert ADC value to voltage
    x_voltage = (x_val / 4095) * VREF

    # Print ADC value and voltage
    print(f"X: {x_val} ({x_voltage:.2f}V)")

    time.sleep(0.1)
```

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board
GP15	LED1

https://wokwi.com/projects/426595147236673537

PROGRAM:

DAC:

```
from machine import Pin, PWM
import time

# Set up PWM on GP15
pwm = PWM(Pin(15))
pwm.freq(1000)  # Set frequency to 1 kHz

while True:
    for duty in range(0, 65536, 5000):  # Increase duty cycle
        pwm.duty_u16(duty)  # Set duty cycle (0-65535)
        voltage = (duty / 65535) * 3.3  # Convert to voltage
        print(f"PWM Output Voltage: {voltage:.2f}V")
        time.sleep(0.1)

    for duty in range(65535, -1, -5000):  # Decrease duty cycle
        pwm.duty_u16(duty)
        voltage = (duty / 65535) * 3.3
        print(f"PWM Output Voltage: {voltage:.2f}V")
        time.sleep(0.1)
```

PROCEDURE:

RESULT:

EXP NO:	Interfacing keyboard and Stepper Motor
DATE	

AIM:

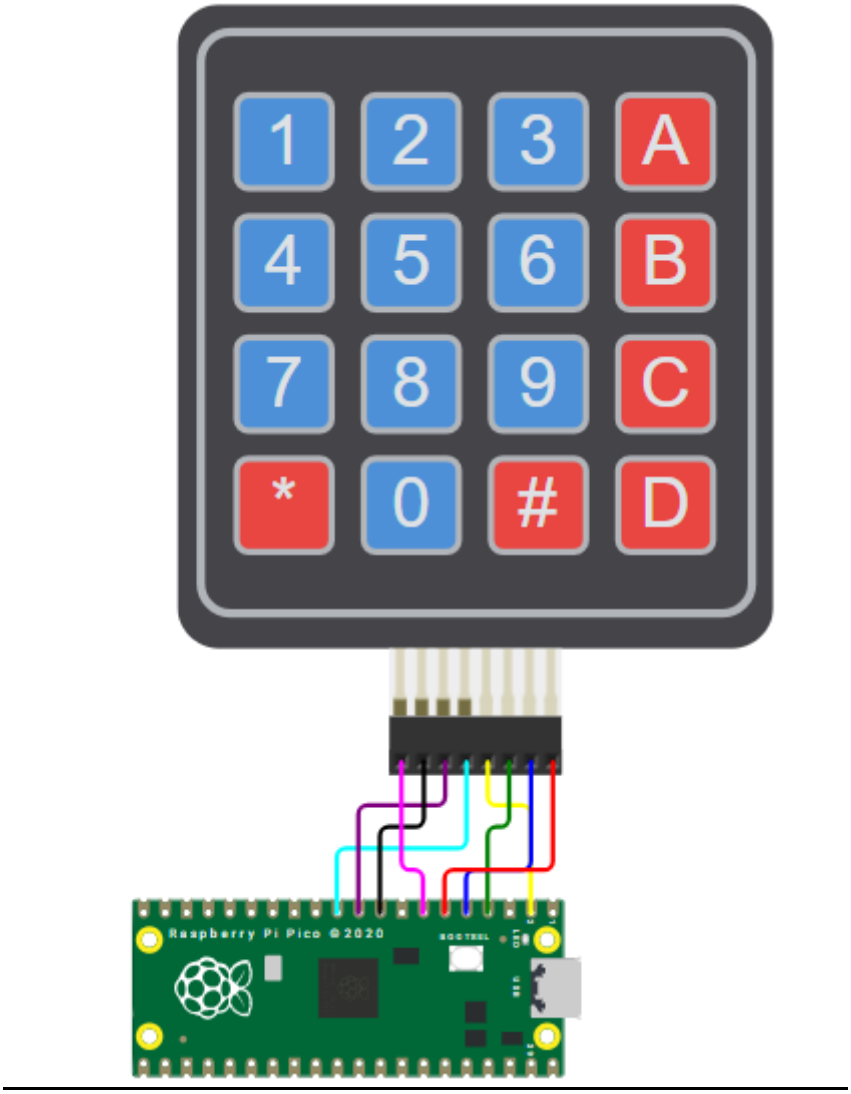
To interface the keyboard and Stepper Motor with the Raspberry Pi Pico W.

HARDWARE & SOFTWARE TOOLS REQUIRED:

S.No	Hardware & Software Requirements	Quantity
1	Thonny IDE	1
2	Raspberry Pi Pico Development Board	1
3	Jumper Wires	few
4	Micro USB Cable	1
5	Keyboard Module	1
6	Stepper Module	1

PROCEDURE:

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board	Keypad Module
GP1	-	C1
GP2	-	C2
GP3	-	C3
GP4	-	C4
GP5	-	R1
GP6	-	R2
GP7	-	R3
GP8	-	R4

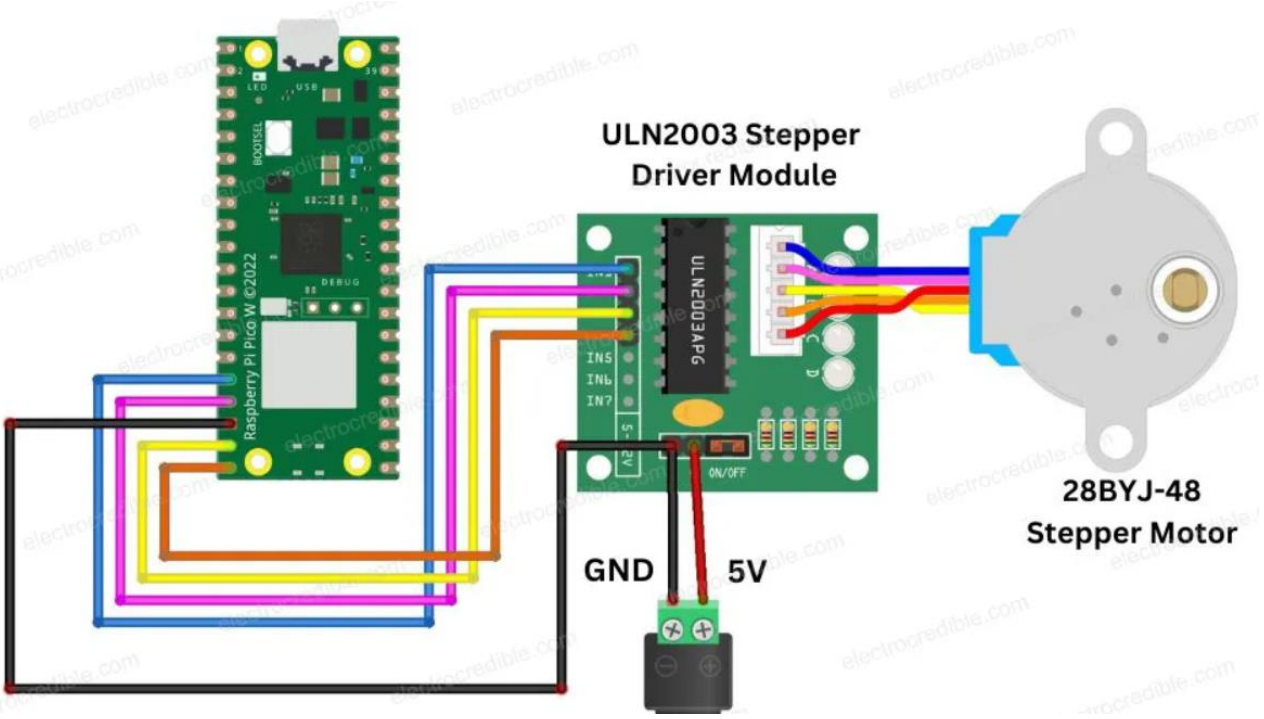
https://wokwi.com/projects/426596421916272641

PROGRAM:

KEYPAD:

```
from machine import Pin
import time  # Standard Python time library
# Define GPIO pins for columns (inputs with pull-up)
col_pins = [Pin(1, Pin.IN, Pin.PULL_UP),
            Pin(2, Pin.IN, Pin.PULL_UP),
            Pin(3, Pin.IN, Pin.PULL_UP),
            Pin(4, Pin.IN, Pin.PULL_UP)]

# Define GPIO pins for rows (outputs)
row_pins = [Pin(5, Pin.OUT),
            Pin(6, Pin.OUT),
            Pin(7, Pin.OUT),
            Pin(8, Pin.OUT)]

# Keypad Button Layout
key_map = [["1", "2", "3", "A"],
           ["4", "5", "6", "B"],
           ["7", "8", "9", "C"],
           ["*", "0", "#", "D"]]

def read_keypad():
    for r in range(4):
        row_pins[r].value(0)  # Activate row
        for c in range(4):
            if col_pins[c].value() == 0:  # Check if key is pressed
                time.sleep(0.02)  # Debounce
                while col_pins[c].value() == 0:  # Wait for key release
                    pass
                row_pins[r].value(1)  # Reset row
                return key_map[r][c]
        row_pins[r].value(1)  # Reset row
    return None

# Main loop
print("--- Keypad Ready ---")
while True:
    key = read_keypad()
    if key:
        print("Pressed:", key)
        time.sleep(0.3)  # Debounce delay
```

CIRCUIT DIAGRAM:



Raspberry Pi Pin	PICO Development Board
GP12	IN1
GP13	IN2
GP14	IN3
GP15	IN4

PROGRAM:

STEPPER MOTOR:

```
from machine import Pin
import time  # Use standard time module

# Define stepper motor pins
pins = [
    Pin(12, Pin.OUT),  # IN1
    Pin(13, Pin.OUT),  # IN2
    Pin(14, Pin.OUT),  # IN3
    Pin(15, Pin.OUT),  # IN4
]

# Full-step sequence for the stepper motor
full_step_sequence = [
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
]

while True:
    for step in full_step_sequence:
        for i in range(4):  # Loop through 4 motor pins
            pins[i].value(step[i])
        time.sleep(0.01)  # Step delay (adjust for speed)
```

RESULT:

Mini Projects for Internet of Things

EXP NO:	CLOUD PLATFORM TO LOG THE DATA
DATE	

AIM:

To set up a cloud platform to log the data

HARDWARE & SOFTWARE TOOLS REQUIRED:

S.No.	Software Requirements	Quantity
1	Blynk Platform	1

CLOUD PLATFORM-BLYNK:

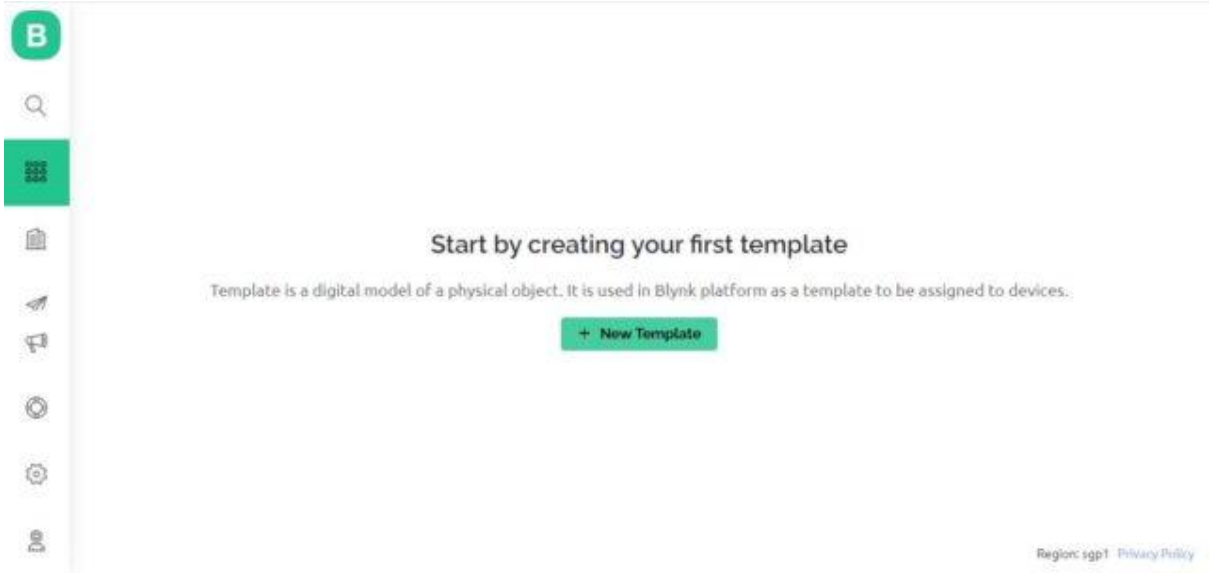
Blynk is a smart platform that allows users to create their Internet of Things applications without the need for coding or electronics knowledge. It is based on the idea of physical programming & provides a platform to create and control devices where users can connect physical devices to the Internet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

Setting up Blynk 2.0 Application

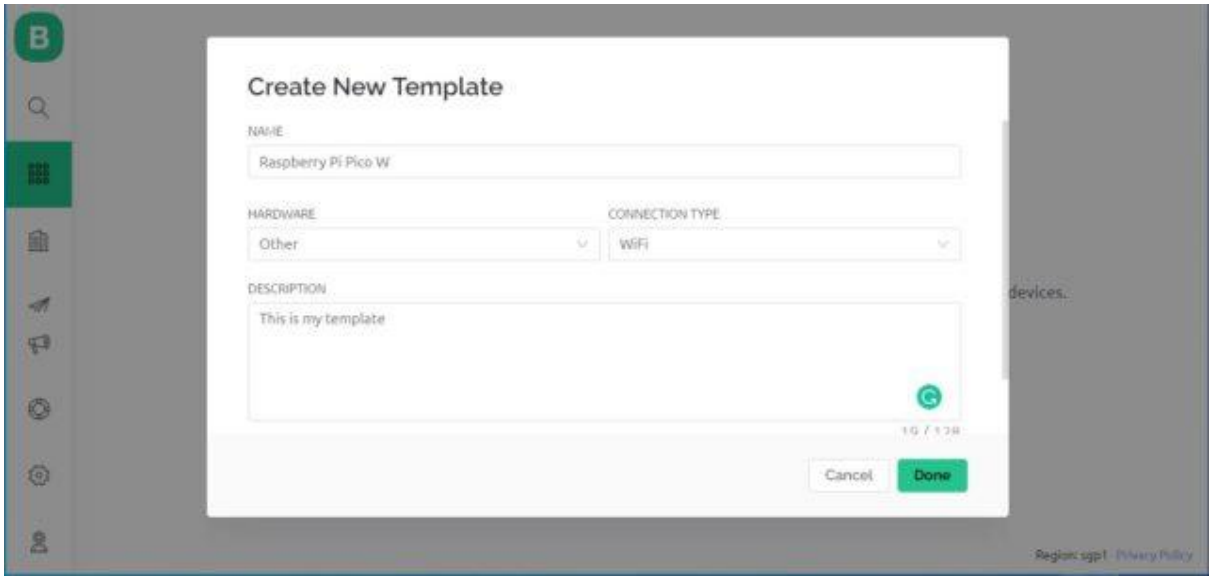
To control the LED using Blynk and Raspberry Pi Pico W, you need to create a Blynk project and set up a dashboard in the mobile or web application. Here's how you can set up the dashboard:

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.

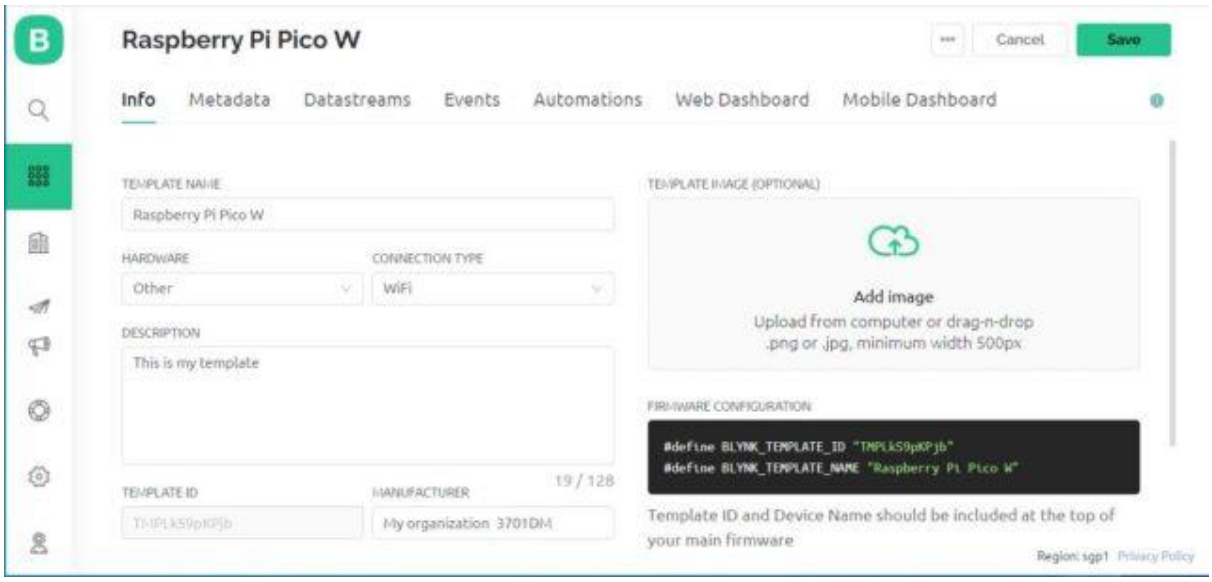
Step 2: Click on **+New Template**.



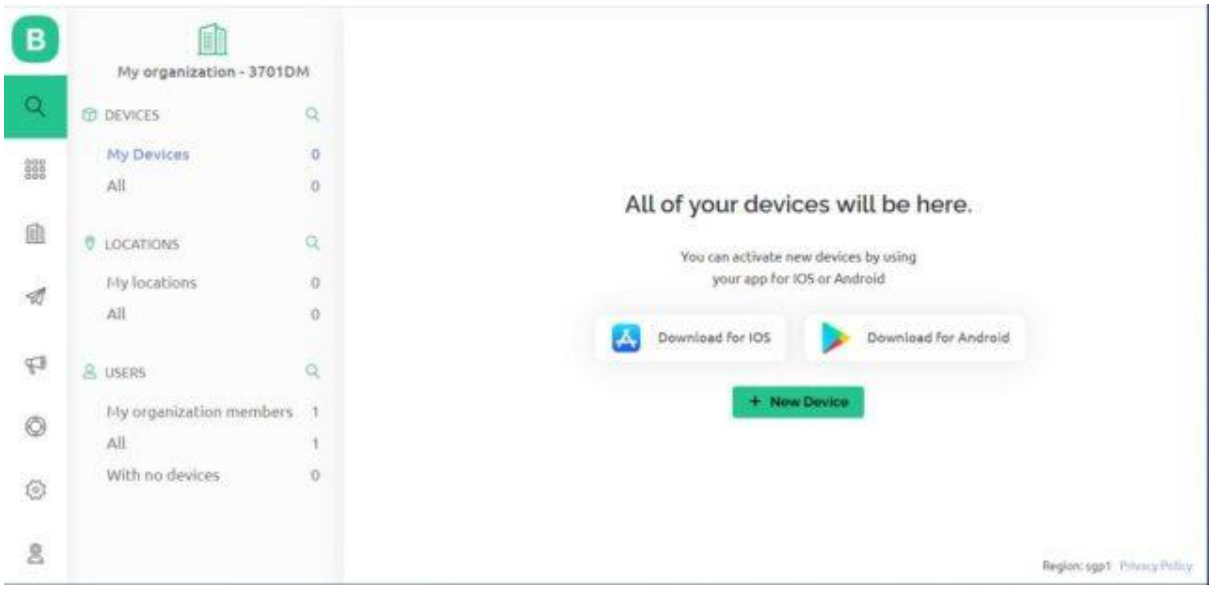
Step 3: Give any name to the Template such as Raspberry Pi Pico W. Select 'Hardware Type' as Other and 'Connection Type' as WiFi.



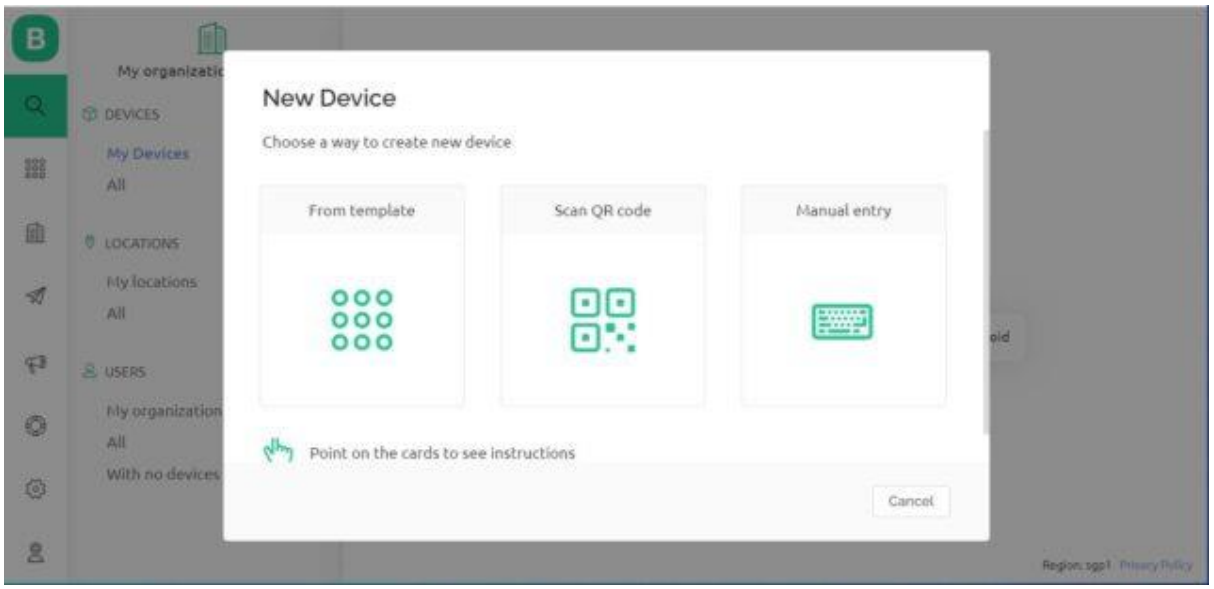
So a template will be created now.



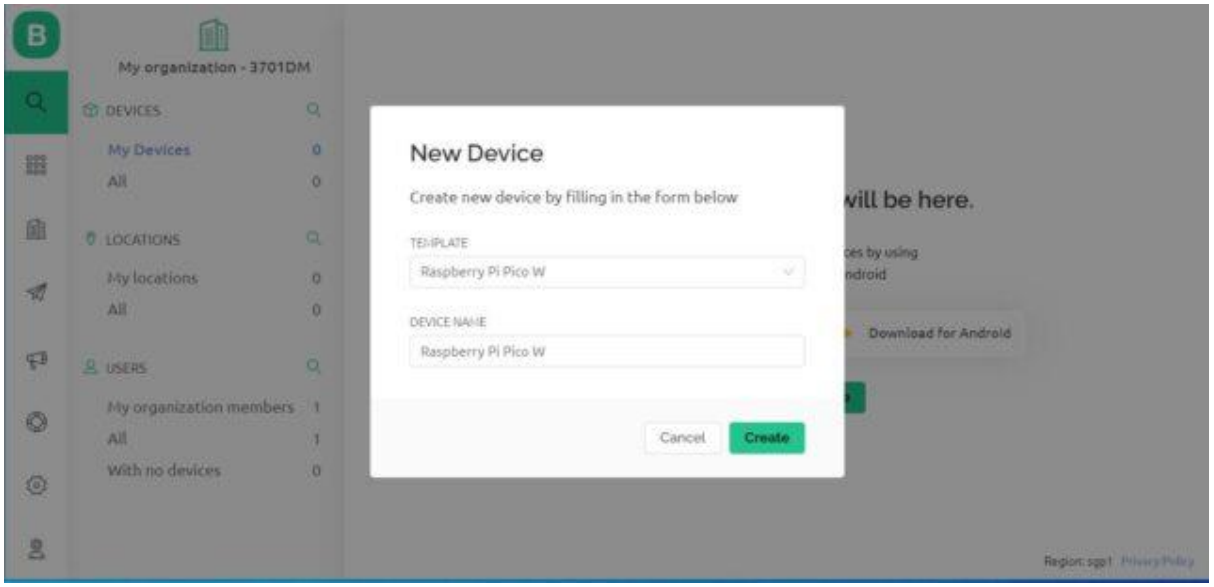
Step 4: Now we need to add a 'New Device' now.



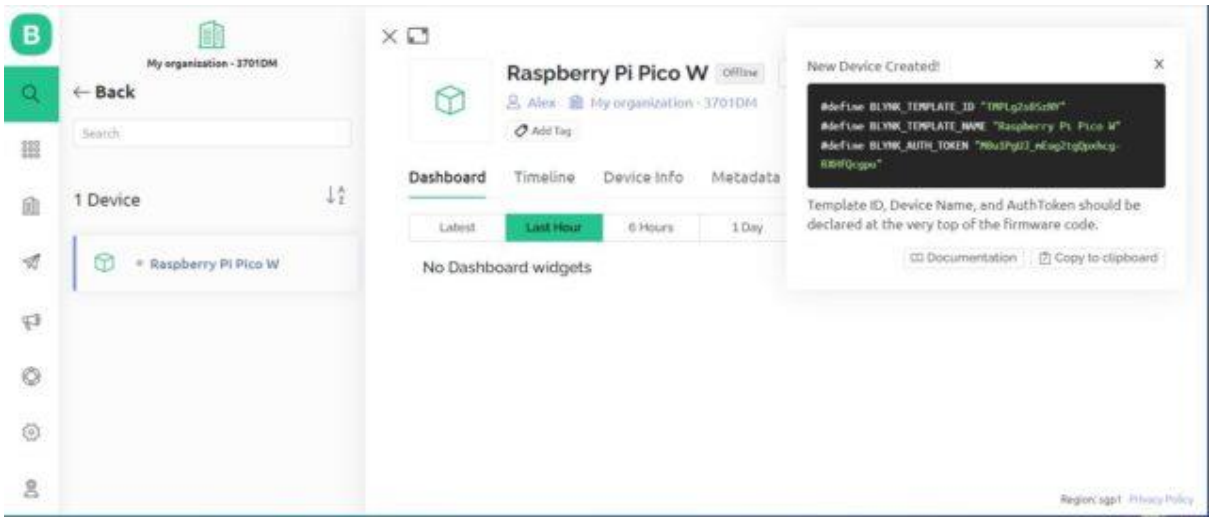
Select a New Device from 'Template'.



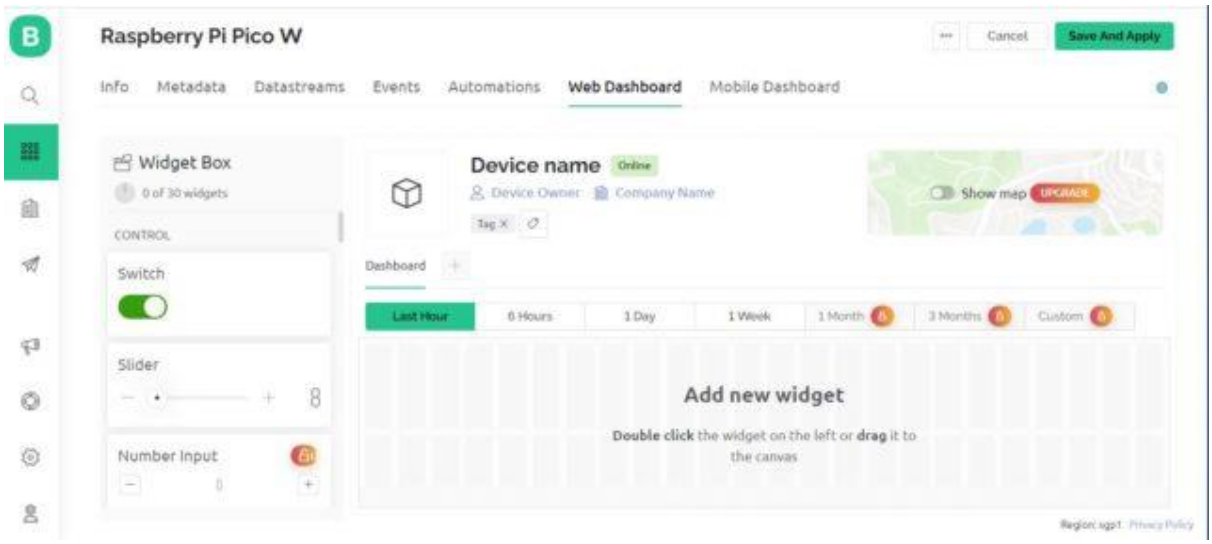
Select the device from a template that you created earlier and also give any name to the device. Click on Create.



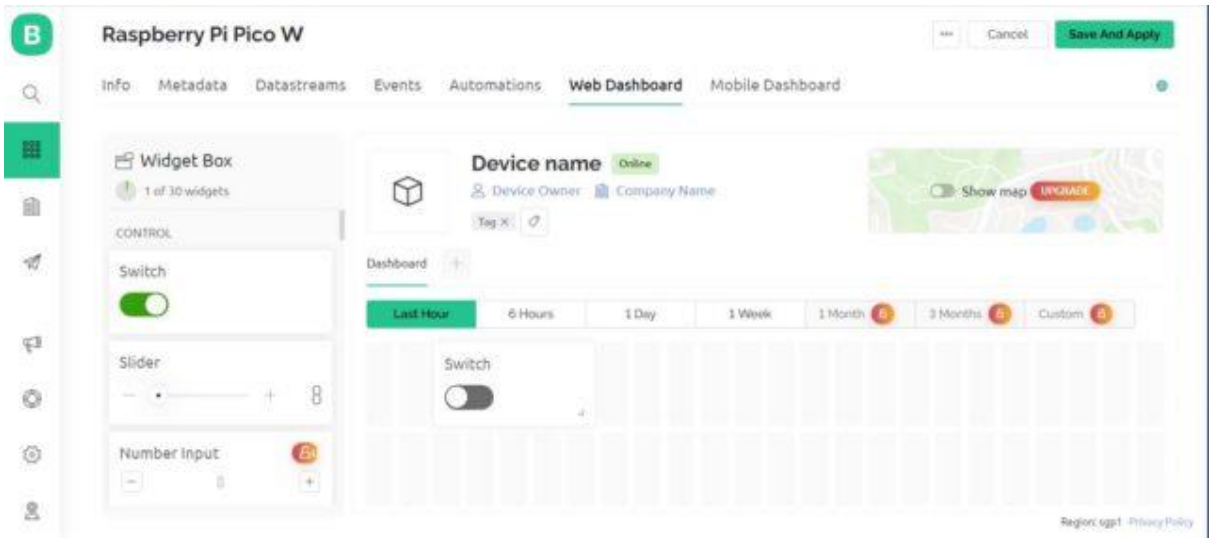
A new device will be created. You will find the Blynk Authentication Token Here. Copy it as it is necessary for the code.



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



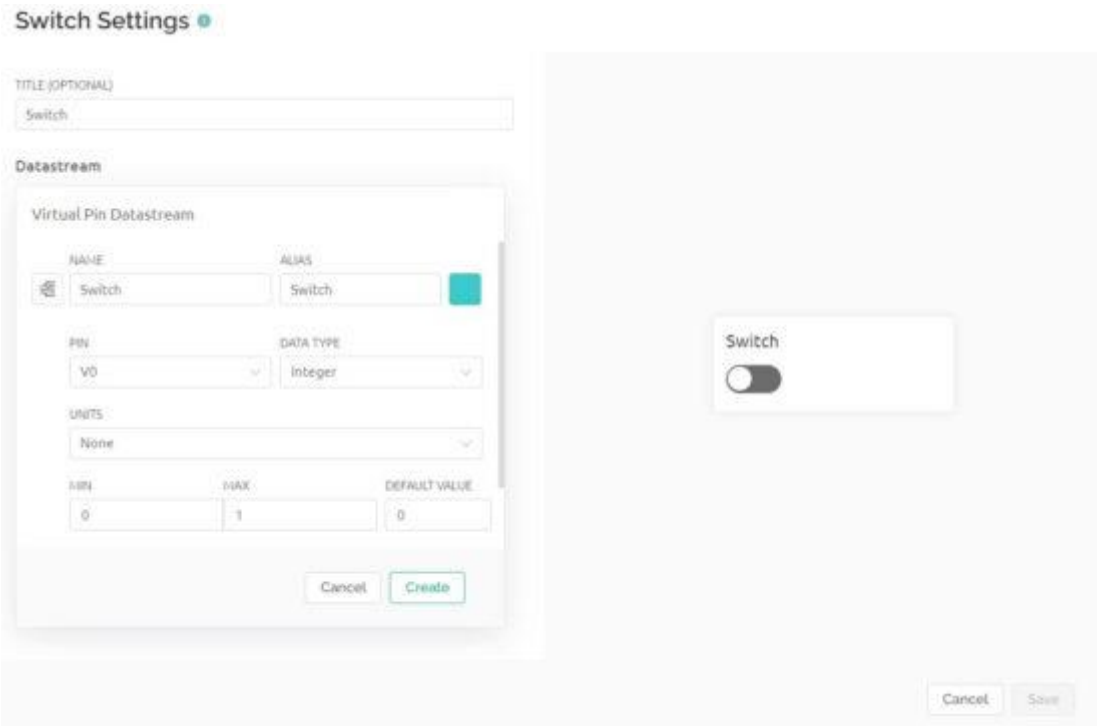
From the widget box drag a switch and place it on the dashboard screen.



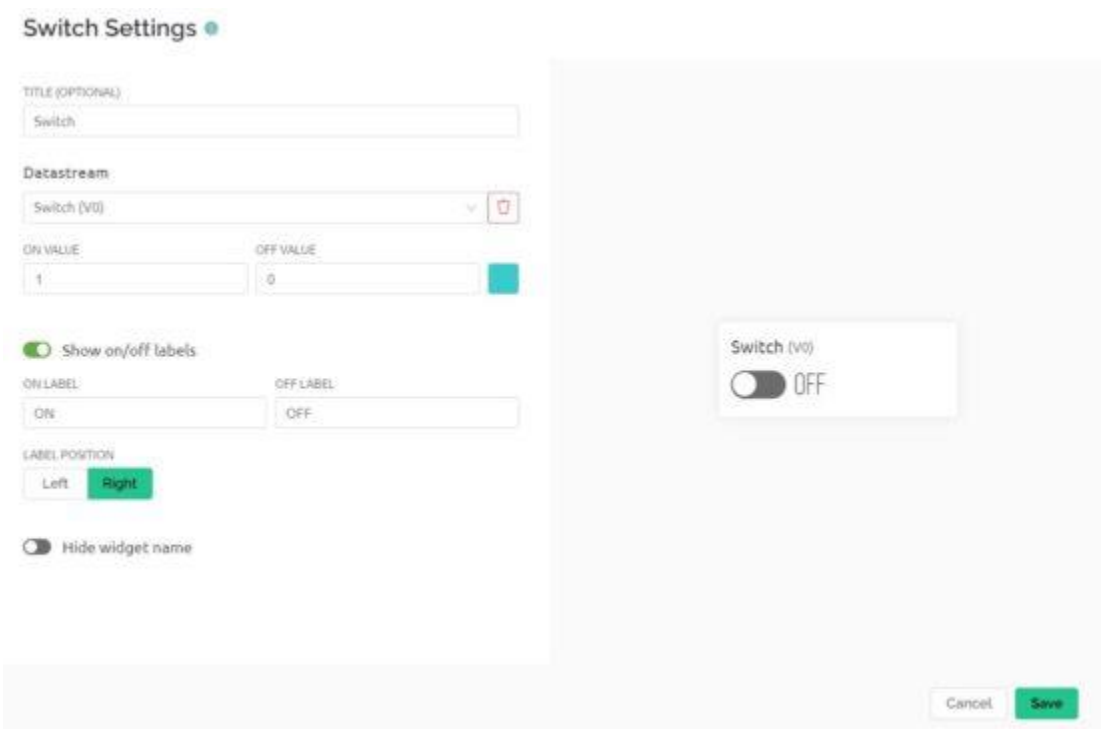
Step 6:

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.



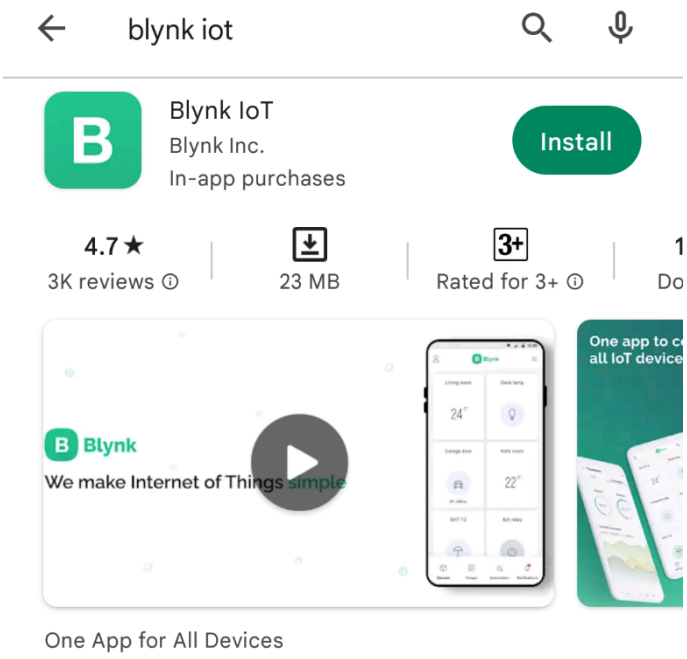
Configure the final steps again.



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:

GARBAGE SEGREGATOR AND BIN LEVEL INDICATOR

ABSTRACT:

The **Garbage Segregator and Bin Level Indicator** is an IoT-based smart waste management system designed to enhance efficiency and hygiene in waste disposal. The system integrates an **IR sensor** to detect human presence near the bin, triggering a **servo motor** to automatically open the lid, providing a touchless experience and reducing physical contact with waste bins.

An **ultrasonic sensor** placed inside the bin continuously monitors the waste level, ensuring real-time tracking of bin capacity. The collected data is transmitted to the **Blynk IoT platform**, allowing remote monitoring and sending alerts when the bin is full. This feature ensures timely waste collection, preventing overflow and improving sanitation in public and private spaces.

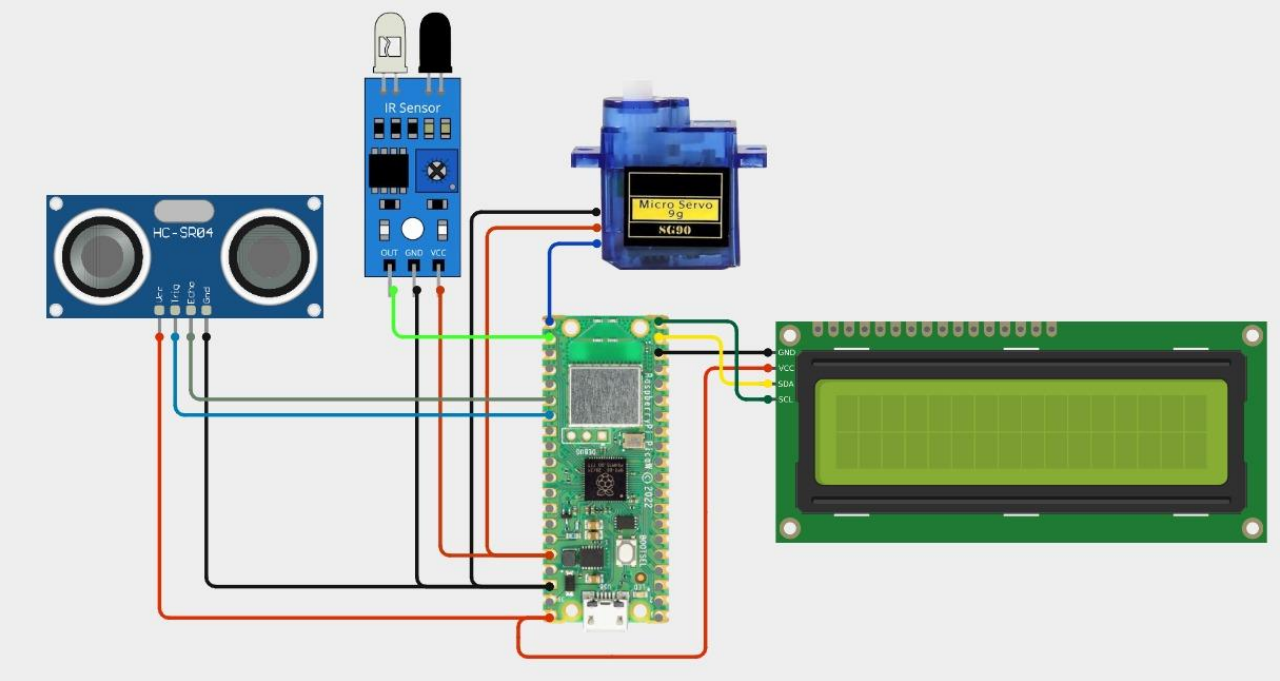
The system operates autonomously, reducing manual intervention and promoting **smart waste segregation**. By integrating IoT technology, it enables efficient waste disposal strategies for **smart cities, commercial spaces, and households**. The touchless mechanism further contributes to **public health and safety**, especially in high-traffic areas.

This project offers a **cost-effective and scalable** solution for modern waste management, making cities cleaner, reducing environmental impact, and supporting **sustainable waste management practices** through automation and real-time data monitoring.

HARDWARE REQUIRED:

- ✅ Raspberry Pi Pico W – Microcontroller for processing
- ✅ IR Sensors – Detect presence of objects in the bin
- ✅ Ultrasonic Sensor (HC-SR04) – Measure bin level
- ✅ Servo Motors – Control the sorting mechanism
- ✅ Blynk IoT Platform – For remote bin level monitoring
- ✅ LEDs & Buzzer – Indicate bin status
- ✅ Power Supply – Battery or adapter

CIRCUIT DIAGRAM:



PROGRAM:

```
from machine import Pin, PWM
import network
import urequests
import utime

# Wi-Fi & Blynk Credentials
SSID = "XXXX"
PASSWORD = "XXXX"
BLYNK_AUTH = "XXXX"

# Pin Configuration
IR_SENSOR = Pin(16, Pin.IN)
SERVO = PWM(Pin(15))
SERVO.freq(50)
TRIG = Pin(2, Pin.OUT)
ECHO = Pin(3, Pin.IN)

last_fill_percentage = -1  # Stores the last sent bin level

# Connect to Wi-Fi
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)

    for _ in range(10):
        if wlan.isconnected():
            print("✅ Connected to Wi-Fi:", wlan.ifconfig())
            return True
        print("⏳ Connecting to Wi-Fi...")
        utime.sleep(1)

    print("❌ Wi-Fi Connection Failed")
    return False

# Function to control servo
def move_servo(angle):
    duty = int((angle / 180) * 5000 + 1000)
    SERVO.duty_u16(duty)
    utime.sleep(1)
    SERVO.duty_u16(0)

# Function to measure distance using Ultrasonic Sensor
```

```
def measure_distance():
    TRIG.low()
    utime.sleep_us(2)
    TRIG.high()
    utime.sleep_us(10)
    TRIG.low()

    timeout = utime.ticks_ms() + 500  # 500ms timeout

    while ECHO.value() == 0:
        if utime.ticks_ms() > timeout:
            print("⚠️ Ultrasonic sensor timeout - No echo received")
            return 100

    signal_off = utime.ticks_us()

    while ECHO.value() == 1:
        if utime.ticks_ms() > timeout:
            print("⚠️ Ultrasonic sensor timeout - Stuck in echo high")
            return 100

    signal_on = utime.ticks_us()
    time_passed = signal_on - signal_off
    distance_cm = (time_passed * 0.0343) / 2

    if distance_cm < 0 or distance_cm > 100:
        print("⚠️ Invalid distance reading:", distance_cm)
        return 100

    return distance_cm

# Function to send data to Blynk via HTTP API
def send_to_blynk(level):
    global last_fill_percentage  # Use global variable to track previous level

    if level != last_fill_percentage:  # Send only if value changed
        url = f"https://blynk.cloud/external/api/update?token={BLYNK_AUTH}&V1={level}"
        try:
            print("📡 Sending data to Blynk:", level)
            response = urequests.get(url, timeout=5)  # Set timeout
            response.close()
            print("✅ Blynk updated:", level)
            last_fill_percentage = level  # Update last sent value
```

```
        except Exception as e:
            print("⚠️ Blynk Update Error:", e)
    else:
        print("ℹ️ Bin level unchanged, skipping Blynk update.")

# Main Function
def main():
    if not connect_wifi():
        return

    while True:
        print("🟢 Checking IR sensor...")
        if IR_SENSOR.value() == 0:
            print("🗑️ Garbage detected! Opening lid...")
            move_servo(90)
            utime.sleep(3)
            move_servo(0)
            print("Lid closed")

        print("🟢 Measuring bin level...")
        bin_level = measure_distance()
        print("📏 Bin Level:", bin_level, "cm")

        MAX_BIN_HEIGHT = 30
        fill_percentage = max(0, min(100, (100 - ((bin_level / MAX_BIN_HEIGHT) * 100))))
        print("📊 Bin Fill:", fill_percentage, "%")

        print("🟢 Sending data to Blynk (if changed)...")
        send_to_blynk(int(fill_percentage))

        if fill_percentage >= 80:
            print("⚠️ Bin is Full! ⚠️")

        print("🔄 Waiting 5 seconds before next cycle...")
        utime.sleep(5)

# Run the program
main()
```

APPLICATIONS:

- Smart Waste Management Systems
- Industrial Waste Segregation
- Automated Waste Disposal in Smart Cities

SMART LOCK SYSTEM

ABSTRACT

The **Smart Lock System** is an IoT-enabled security solution designed for remote and automated access control using a **Raspberry Pi Pico W** and the **Blynk IoT platform**. The system employs a **12V solenoid lock**, controlled via a **relay module**, to provide a secure and reliable locking mechanism. Integrated with a **Wi-Fi module (ESP8266)**, the system enables users to lock and unlock doors remotely via a smartphone application, eliminating the need for physical keys.

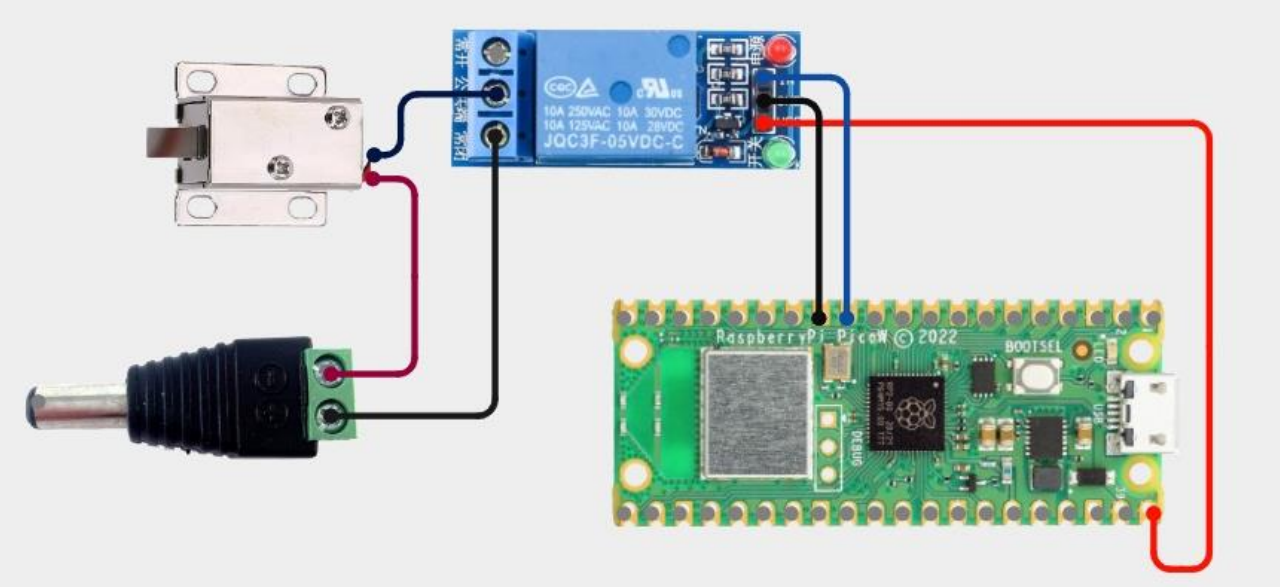
When an authorized user sends a command through the **Blynk app**, the Raspberry Pi Pico W processes the request and triggers the relay to control the solenoid lock. The system can also be enhanced with **multi-factor authentication methods**, such as an RFID module, keypad, or fingerprint sensor, for additional security. Real-time lock status updates can be monitored through the IoT dashboard, ensuring transparency and security.

This project finds applications in **homes, offices, bank lockers, and smart security systems**, providing enhanced protection against unauthorized access. By integrating IoT technology, it ensures seamless operation, remote monitoring, and improved security, making it a **cost-effective and scalable** solution for modern smart locking systems.

HARDWARE REQUIRED:

- ✅ Raspberry Pi Pico W – Main controller
- ✅ Solenoid Lock – Electronic locking mechanism
- ✅ Relay Module – Control high-power lock circuit
- ✅ Blynk IoT Platform – Remote unlocking via smartphone
- ✅ Wi-Fi Module (with Pico W) – Remote control

CIRCUIT DIAGRAM:



PROGRAM:

```
import network
import time
from machine import Pin
import BlynkLib

# WiFi and Blynk Credentials
SSID = "XXXX"
PASSWORD = "XXXX"
BLYNK_AUTH = "Token"


# Initialize WiFi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)

# Attempt to connect to WiFi
print("Connecting to WiFi...")
wlan.connect(SSID, PASSWORD)

timeout = 10  # Timeout for WiFi connection (10 seconds)
while not wlan.isconnected() and timeout > 0:
    print("Trying to connect...")
    time.sleep(1)
    timeout -= 1

if wlan.isconnected():
    print("✅ Connected to WiFi:", wlan.ifconfig())
else:
    print("❌ Failed to connect. Check credentials & signal strength.")
    exit()  # Stop the script if WiFi is not connected

# Initialize Blynk
blynk = BlynkLib.Blynk(BLYNK_AUTH, server="blynk.cloud", port=80, insecure=True)
# Relay connected to GPIO 15
relay = Pin(15, Pin.OUT)

# Blynk Virtual Pin Handling
@blynk.on("V0")
def lock_control(value):
    if int(value[0]) == 1:
        relay.value(1)  # Lock Open
```

```
        print(" 🔓 Door Unlocked")
    else:
        relay.value(0)  # Lock Closed
        print(" 🔒 Door Locked")

# Main Loop
while True:
    blynk.run()
```

APPLICATIONS:

- Home and Office Security
- ATM & Bank Locker Systems
- Hotel Room Lock Systems

COLOUR-BASED PRODUCT SORTING

ABSTRACT

The **Colour-Based Product Sorting System** is an automated solution designed to classify objects based on their color, enhancing efficiency in industrial sorting and packaging processes. The system employs a **TCS3200/TCS230 color sensor**, which detects the color of an object as it moves through the system. The detected color is then displayed on an **LCD/OLED screen**, providing real-time feedback to users.

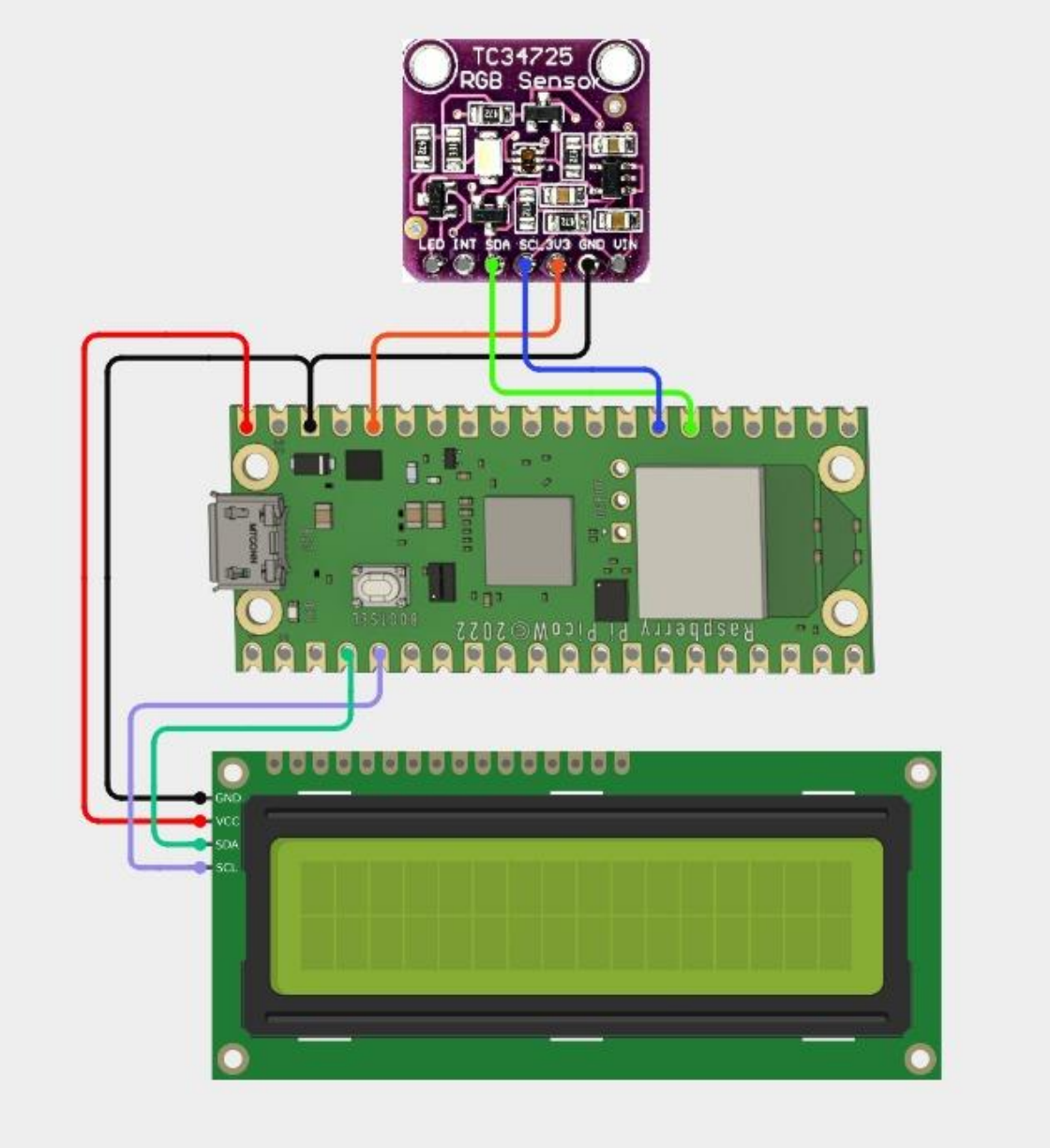
A **Raspberry Pi Pico W** acts as the main controller, processing the color data and triggering a **servo motor** to push the object into its respective sorting bin. The system ensures accurate and high-speed classification, reducing manual effort and human error in sorting operations. The use of **servo motors** allows precise movement, making the system adaptable for various industries, including **food processing, pharmaceuticals, recycling, and manufacturing**.

By integrating **IoT and automation**, this project optimizes resource management, increases productivity, and improves quality control in industrial applications. The system can be further enhanced with a **conveyor belt mechanism**, enabling continuous sorting for large-scale operations. With its ability to provide real-time sorting data and automation, the project offers a **cost-effective, scalable, and efficient** solution for modern production lines and material handling systems.

HARDWARE REQUIRED:

- ✅ Raspberry Pi Pico W – Main controller
- ✅ TCS3200/TCS230 Colour Sensor – Detect object color
- ✅ Servo Motors – Push objects into sorting bins
- ✅ LCD Display (16x2 or OLED) – Display detected color
- ✅ Power Supply – 12V DC Adapter or Battery

CIRCUIT DIAGRAM:



TCS3200 Pin	Raspberry Pi Pico W Pin
VCC	3.3V
GND	GND
S0	GP2
S1	GP3
S2	GP4
S3	GP5

PROGRAM:

```
from machine import Pin, PWM
from utime import sleep

# TCS3200 Sensor Pins
S0 = Pin(2, Pin.OUT)
S1 = Pin(3, Pin.OUT)
S2 = Pin(4, Pin.OUT)
S3 = Pin(5, Pin.OUT)
OUT = Pin(6, Pin.IN)

# IR Sensor Pin (GP10)
IR_SENSOR = Pin(10, Pin.IN, Pin.PULL_UP)  # Pull-up enabled

# Servo Motor on GPIO 9
servo = PWM(Pin(9))
servo.freq(50)  # Set PWM frequency to 50Hz (standard for servos)

# Set frequency scaling to 100%
S0.value(1)
S1.value(0)

def read_color():
    """Reads Red, Green, and Blue color frequency from TCS3200"""
    colors = ["Red", "Green", "Blue"]
    values = []

    for i, (s2, s3) in enumerate([(0, 0), (1, 1), (0, 1)]):  # Red, Green, Blue
        S2.value(s2)
        S3.value(s3)
        sleep(0.1)

        pulse_time = machine.time_pulse_us(OUT, 1, 100000)  # 100ms timeout

        if pulse_time < 0:
            print(f"⚠️ Error reading {colors[i]} - No valid pulse detected!")
            values.append(9999)  # Assign a large value to avoid misinterpretation
        else:
            values.append(pulse_time)

    # Normalize values to avoid negative values
    min_val = min(values)
```

```
    if min_val < 0:
        values = [v - min_val for v in values]

    return values[0], values[1], values[2]

def move_servo(angle):
    """Moves servo to the given angle (0-180)"""
    duty = int((angle / 180) * 5000 + 2500)  # Convert angle to PWM duty cycle
    servo.duty_u16(duty)
    sleep(1)

while True:
    if IR_SENSOR.value() == 0:  # Object detected (IR sensor goes LOW)
        print("\n🔍 Object Detected! Scanning Color...")
        r, g, b = read_color()
        print("Raw Values - Red:", r, "Green:", g, "Blue:", b)

        # Color detection logic
        if r < g and r < b:
            detected_color = "RED"
            move_servo(0)   # Move servo to RED position
        elif g < r and g < b:
            detected_color = "GREEN"
            move_servo(90)  # Move servo to GREEN position
        elif b < r and b < g:
            detected_color = "BLUE"
            move_servo(180) # Move servo to BLUE position
        else:
            detected_color = "UNKNOWN"

        print(f"🎨 Detected Color: {detected_color}")
        sleep(2)  # Wait before scanning again
    else:
        print("⏳ Waiting for object...")

    sleep(0.5)
```

APPLICATIONS:

- Automated Factory Sorting Systems
- Agricultural Sorting (Fruits, Vegetables)
- Pharmaceutical and Chemical Industry Sorting

FIRE DETECTION USING IMAGE PROCESSING

ABSTRACT

The **Image Processing-Based Fire Detection System** is an IoT-enabled fire monitoring solution that utilizes **OpenCV, Raspberry Pi Pico W, and PySerial** to detect fire in real-time and send alerts. The system employs a **camera module** connected to a PC or Raspberry Pi, which continuously captures video frames. The captured images are processed using **OpenCV-based flame detection algorithms**, analyzing color intensity and movement patterns to identify fire.

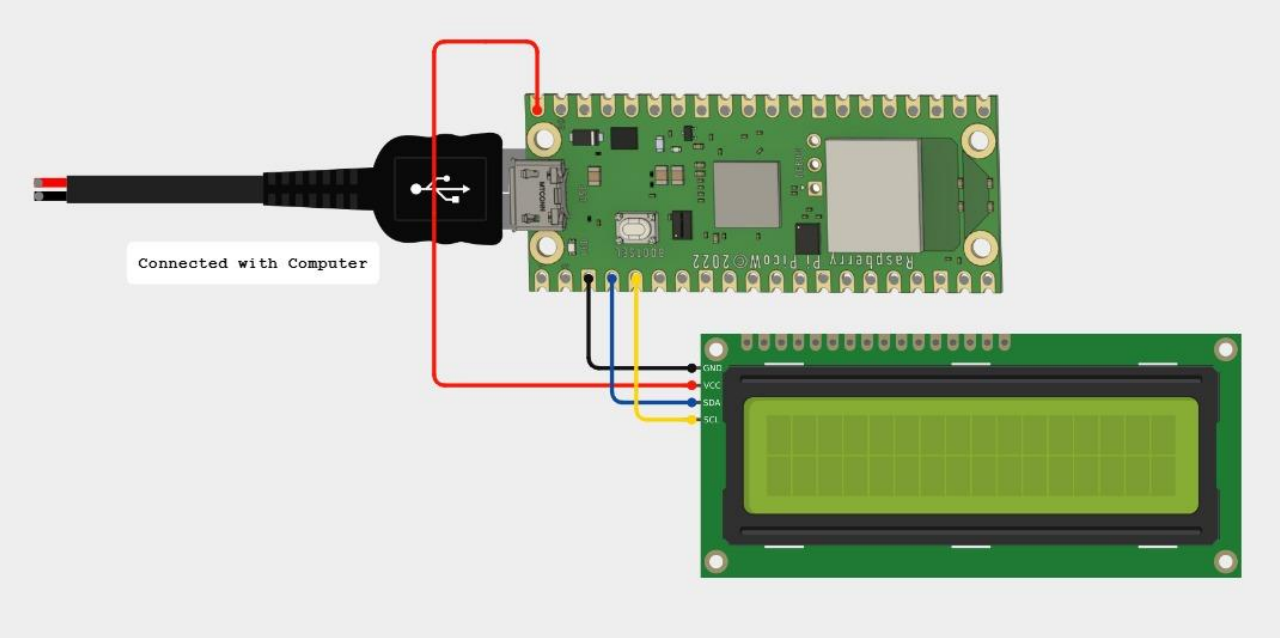
The **Raspberry Pi Pico W** acts as an IoT-enabled microcontroller, interfacing with external components such as **buzzers, LEDs, and relays** to trigger safety mechanisms. Communication between the **PC (running OpenCV) and the Pico W** is established via **PySerial**, allowing the Pico W to receive fire detection signals and activate appropriate response actions like **turning on an alarm, sending data to the Blynk app, or triggering a fire suppression system**.

Once a fire is detected, an **alert is sent to the Blynk IoT platform**, enabling remote monitoring and immediate action. This system is ideal for **smart homes, industries, and surveillance systems**, providing a **cost-effective, scalable, and automated** fire detection solution that minimizes response time and enhances safety.

HARDWARE REQUIRED:

- ✅ Raspberry Pi 4 or Raspberry Pi Pico W – Main processor
- ✅ Pi Camera / USB Camera – Capture images
- ✅ OpenCV Library – Image processing
- ✅ ML Model (Optional) – Train on fire detection
- ✅ Buzzer & LED – Alert system
- ✅ Wi-Fi Module (ESP8266 if using Pico W) – Send alerts
- ✅ Cloud Integration (Blynk, Firebase, or MQTT) – Remote notifications

CIRCUIT DIAGRAM:



PROGRAM:

```
import cv2
import numpy as np

def detect_fire(frame):
    # Convert frame to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define fire color range in HSV
    lower_bound = np.array([0, 120, 200], dtype=np.uint8)
    upper_bound = np.array([35, 255, 255], dtype=np.uint8)

    # Threshold the image to extract fire-like regions
    mask = cv2.inRange(hsv, lower_bound, upper_bound)

    # Apply morphological operations to remove noise
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # Find contours of the detected fire
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw bounding boxes around fire regions
    for contour in contours:
        if cv2.contourArea(contour) > 500:  # Filter small areas
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
            cv2.putText(frame, "Fire Detected!", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)

    return frame

# Initialize webcam
cap = cv2.VideoCapture(0)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    processed_frame = detect_fire(frame)
```

```
    # Display output
    cv2.imshow("Fire Detection", processed_frame)

    # Exit condition
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

APPLICATIONS:

- Fire Safety in Homes & Industries
- Surveillance Systems
- Forest Fire Monitoring

VEHICLE NUMBER PLATE DETECTION

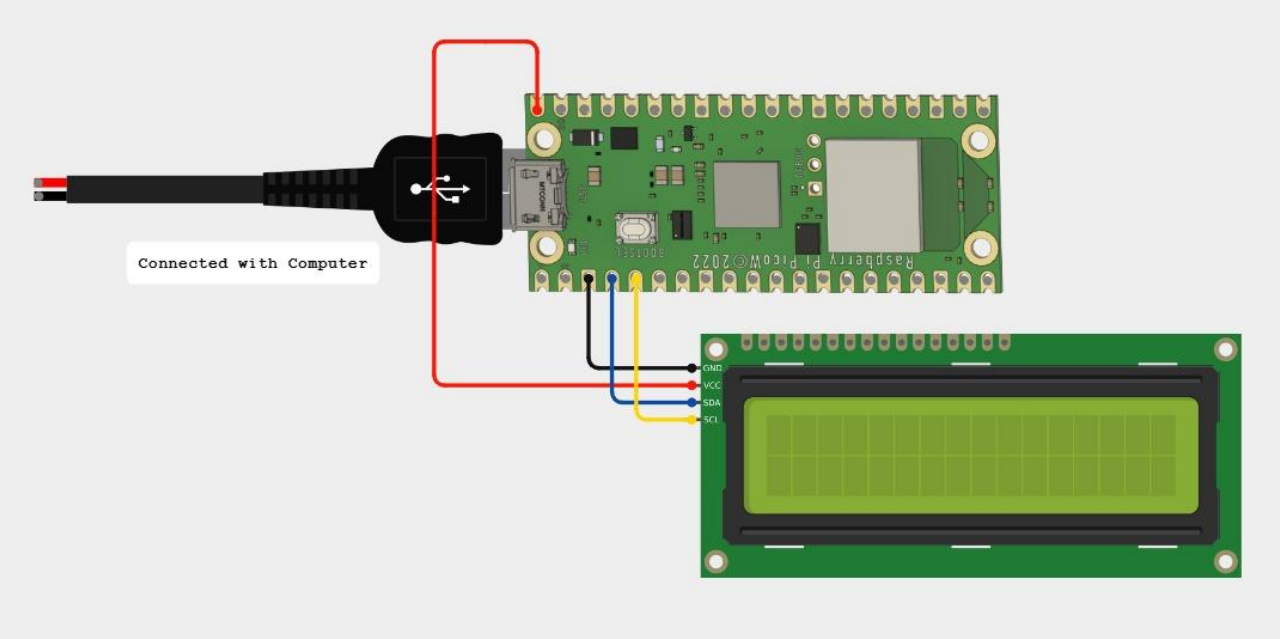
ABSTRACT:

The rapid urbanization and increase in vehicle usage have created the need for automated systems that can efficiently track and manage vehicle movements. **Vehicle Number Plate Detection (VNPD)** is a crucial technology that helps in automating traffic management, parking systems, toll collection, and security access control. This project focuses on developing a **low-cost, efficient, and real-time number plate detection system** using **Raspberry Pi Pico W, ESP32-CAM, OpenCV, and Optical Character Recognition (OCR)**. The system aims to **automatically capture images of vehicle license plates**, process them to extract alphanumeric characters, and display the detected number on an **I2C LCD display** connected to the **Raspberry Pi Pico W**.
The system is designed to **reduce manual intervention** in tracking vehicles and improve accuracy in monitoring entry and exit logs. By utilizing **OpenCV's image processing techniques and Tesseract OCR**, the system can accurately identify vehicle numbers under various lighting and environmental conditions. The extracted number is then sent to the **Raspberry Pi Pico W via Serial Communication (PySerial)**, where it is displayed on an LCD screen for real-time monitoring. This system can be implemented in various **smart city applications**, including **automated toll booths, restricted zone monitoring, and parking lot management**.

Apparatus Required:

- ✅ **Raspberry Pi 4 / Pico W** – Main processor
- ✅ **Pi Camera / USB Camera** – Capture vehicle images
- ✅ **OpenCV & OCR (Tesseract)** – Image processing and text recognition
- ✅ **Wi-Fi Module (ESP8266 with Pico W)** – Cloud data storage
- ✅ **LCD Display** – Show detected number
- ✅ **Blynk/Firebase Integration** – Send detected plates to the cloud

CIRCUIT DIAGRAM:



PROGRAM:

```
import cv2
import pytesseract
import numpy as np

# Set the path for Tesseract OCR
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

# Load Haar Cascade for number plate detection
plate_cascade = cv2.CascadeClassifier("haarcascade_russian_plate_number.xml")

# Initialize webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect number plates
    plates = plate_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4, minSize=(50,
50))

    for (x, y, w, h) in plates:
        # Draw a rectangle around the detected plate
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        # Extract the plate region
        plate_roi = gray[y:y+h, x:x+w]

        # Image Preprocessing
        plate_roi = cv2.GaussianBlur(plate_roi, (5, 5), 0)  # Reduce noise
        plate_roi = cv2.threshold(plate_roi, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)[1]  # Binarization
        plate_roi = cv2.resize(plate_roi, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)  #
Upscaling

        # Apply OCR to extract text
```

```
        text = pytesseract.image_to_string(plate_roi, config="--psm 7 --oem 3 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")
        text = text.strip().replace("\n", "").replace(" ", "")

        print("Detected Plate Number:", text)

        # Display the extracted text on the frame
        cv2.putText(frame, text, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    # Show the result
    cv2.imshow("Vehicle Number Plate Detection", frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()
```

Applications:

- Smart Parking & Toll Collection
- Traffic Law Enforcement
- Security and Surveillance Systems