SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

Approved by AICTE, Affiliated to Anna University, Chennai, Accredited by NBA,
'A' Grade Accreditation by NAAC & ISO 21001:2018 Certified Institution

SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF ELECTRONICS AND INSTRUMENTATION ENGINEERING

VI SEMESTER

EI3665 - EMBEDDED SYSTEMS

(Theory cum laboratory)

LAB MANUAL

Regulation - 2023

Academic Year 2025 - 2026 (EVEN SEMESTER)

Prepared by

Ms. K.S.JAIBHAVANI, Assistant Professor - EIE

LABORATORY PRACTICE

SAFETY RULES

1. SAFETY is of paramount importance in the Laboratories.
2. Electricity NEVER EXECUSES careless persons. So, exercise enough care and attention in handling electrical & electronic equipment and follow safety practices in the laboratory. (Electricity is a good servant but a bad master).
3. Avoid direct contact with any voltage source and power line voltages. (Otherwise, any such contact may subject you to electrical shock).
4. Wear rubber-soled shoes. (To insulate you from earth so that even if you accidentally contact a live point, current will not flow through your body to earth and hence you will be protected from electrical shock).
5. Wear laboratory-coat and avoid loose clothing. (Loose clothing may get caught on an equipment/instrument and this may lead to an accident particularly if the equipment happens to be a rotating machine).
6. Girl students should have their hair tucked under their coat or have it in a knot.
7. Do not wear any metallic rings, bangles, bracelets, wristwatches and neck chains. (When you move your hand/body, such conducting items may create a short circuit or may touch a live point and thereby subject you to electrical shock).
8. Be certain that your hands are dry and that you are not standing on wet floor. (Wet parts of the body reduce the contact resistance thereby increasing the severity of the shock).
9. Ensure that the power is OFF before you start connecting up the circuit. (Otherwise you will be touching the live parts in the circuit).
10. Get your circuit diagram approved by the staff member and connect up the circuit strictly as per the approved circuit diagram.
11. Check power chords for any sign of damage and be certain that the chords use safety plugs and do not defeat the safety feature of these plugs by using ungrounded plugs.
12. When using connection leads, check for any insulation damage in the leads and avoid such defective leads.
13. Switch on the power to your circuit and equipment, only after getting them checked up and approved by the staff member.
14. Do not make any change in the connection without the approval of the staff member.
15. In case you notice any abnormal condition in your circuit (like insulation heating up, resistor heating up etc.), switch off the power to your circuit immediately and inform the staff member.
16. Keep hot soldering iron in the holder when not in use.
17. After completing the experiment show your readings to the staff member and switch off the power to your circuit after getting approval from the staff member.

EI3665 - EMBEDDED SYSTEMS

Theory cum laboratory

LABORATORY SYLLABUS

LIST OF PRACTICAL EXPERIMENTS:

1. **Implementation of specific tasks using Embedded C/Python programming.**
2. **Interfacing input devices with 8051/PIC16F877A/LPC4088.**
3. **Interfacing output devices with 8051/PIC16F877A/LPC4088.**
4. **Implementation of recurring tasks using the timers and interrupts of 8051/PIC microcontroller/ LPC4088.**
5. **Interfacing ADC & DAC with 8051 microcontrollers.**
6. **PWM generation using PIC16F877A/LPC4088.**
7. **Interfacing RTC with microcontroller.**
8. **Establishing serial data transmission through UART.**
9. **Establishing serial data communication using I2C and SPI protocols.**
10. **Wireless data communication using IoT (Zigbee/ GSM/ Bluetooth).**
11. **Design and implementation of ON/OFF control strategy.**
12. **Implementation of basic experiments using Raspberry PI / Arduino.**

LIST OF EXPERIMENTS

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**.

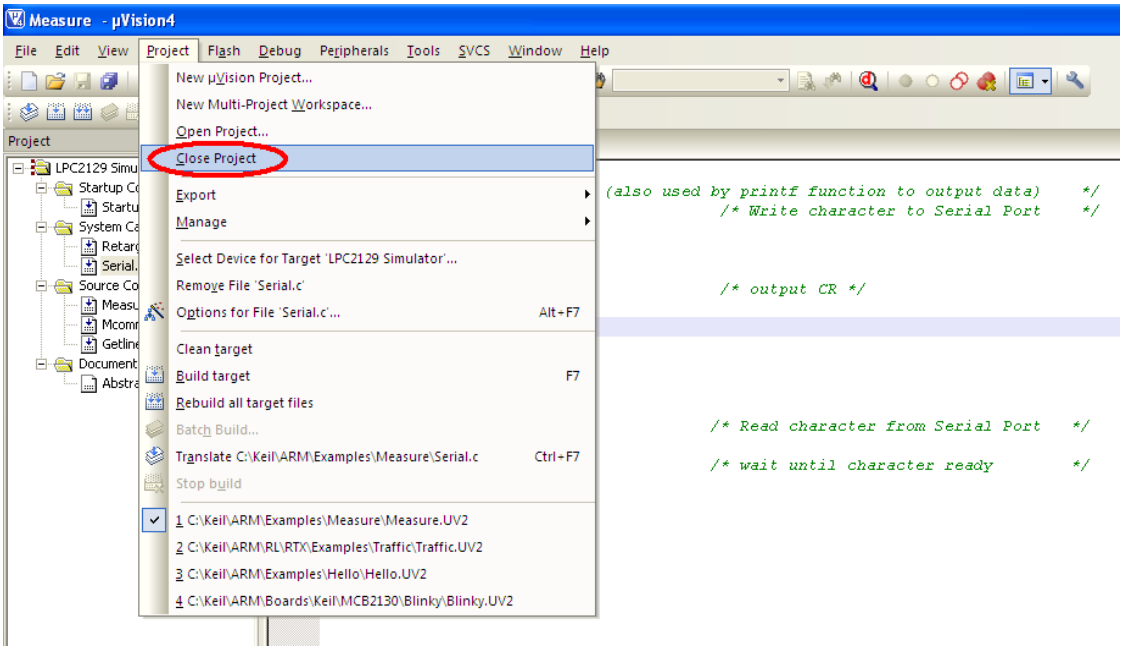
Software Flow

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.

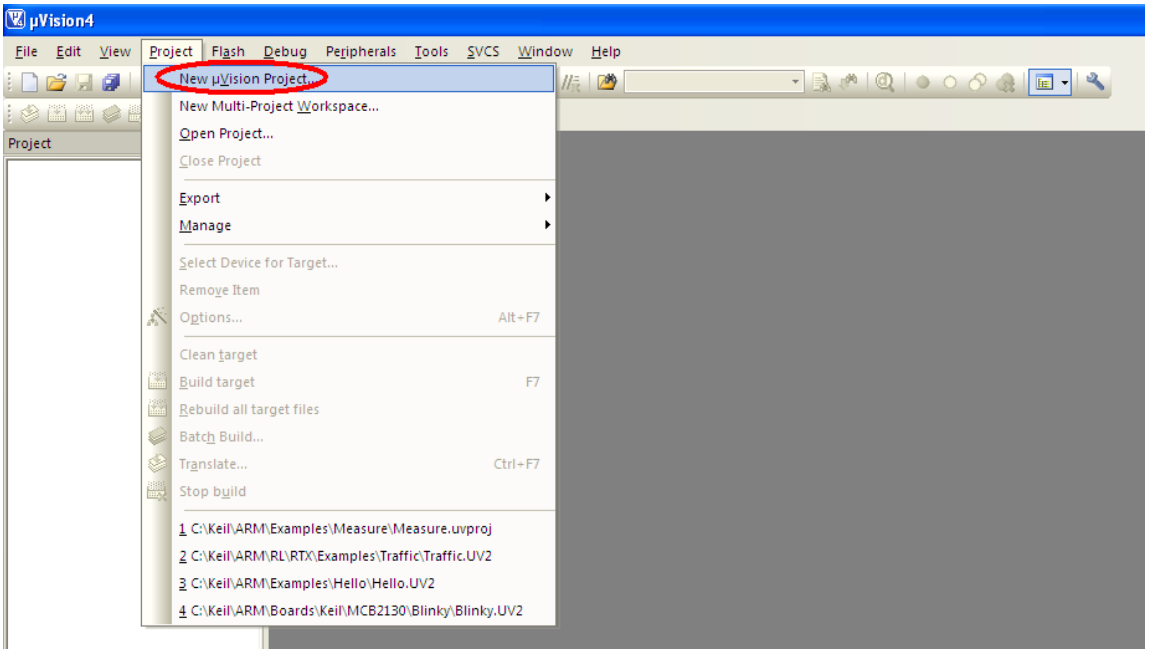
Creating a New Project

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

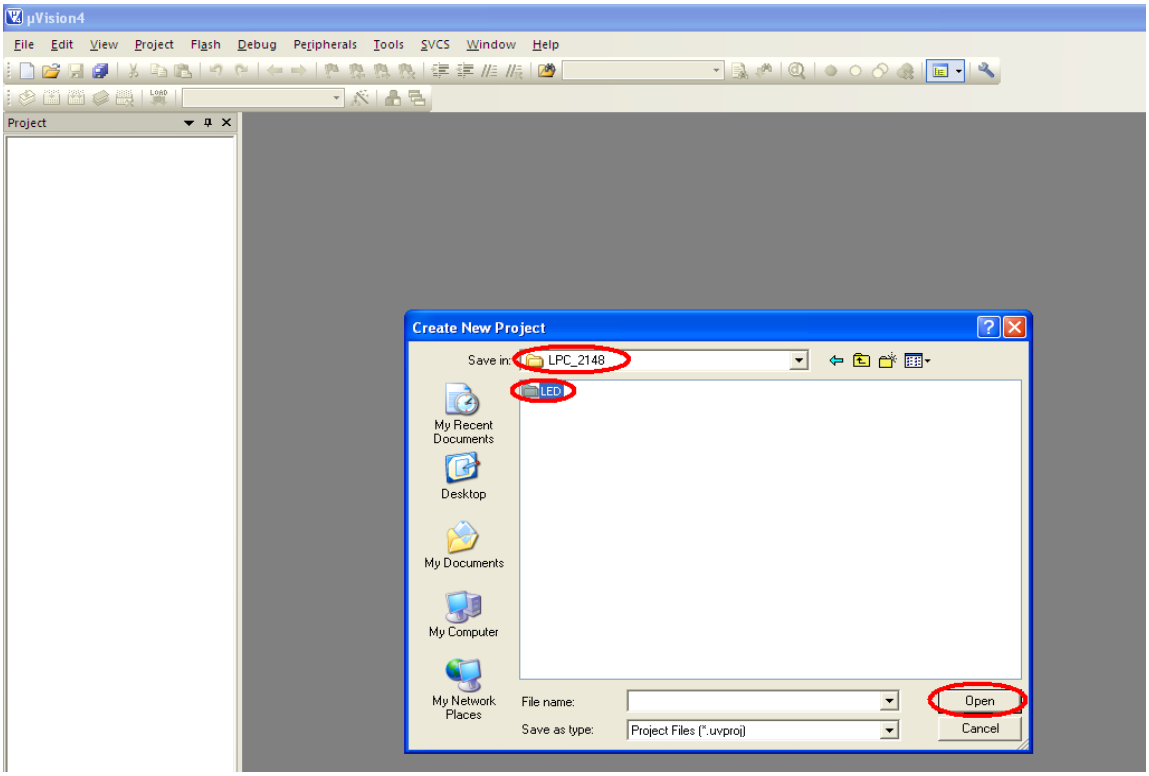
STEP 1: Go to "**Project**" and close the current project "**Close Project**".



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

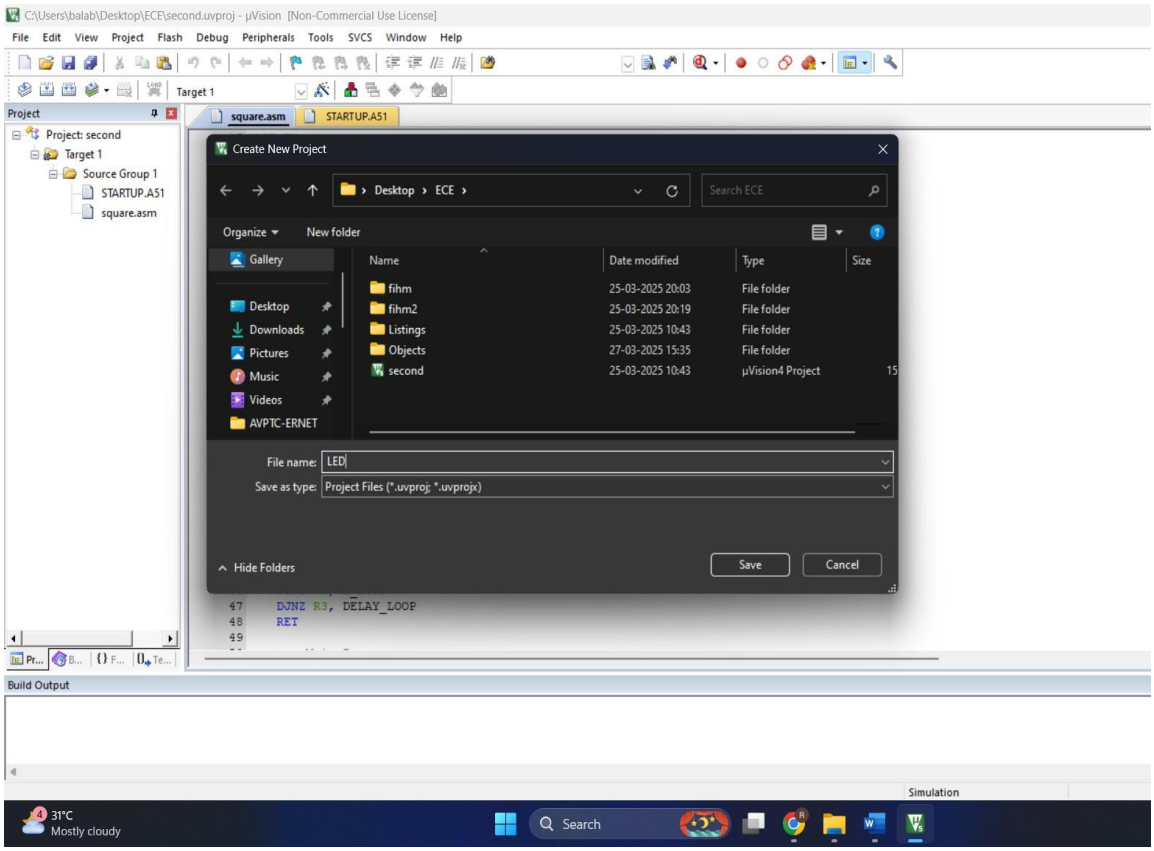


STEP 3: A small window will pop up with the name "**Create New Project**" and can be created and select destination path.

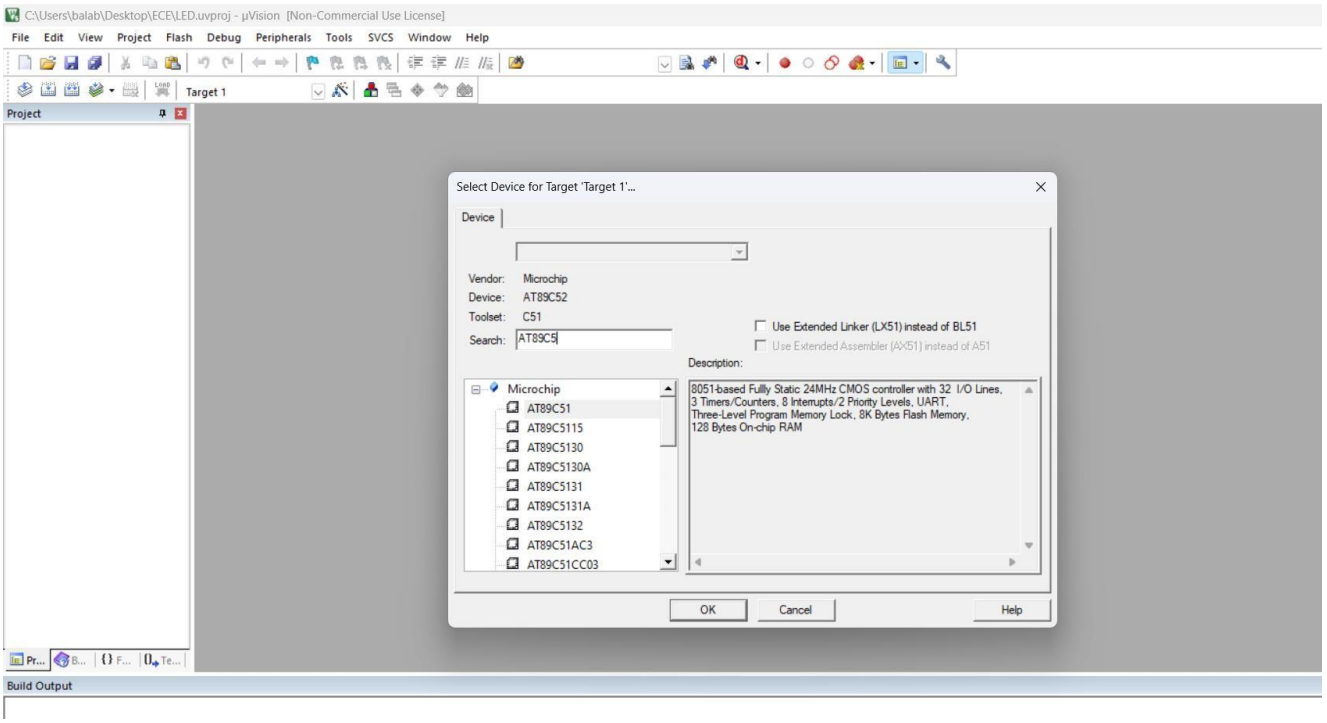


STEP 4: Create a folder and give a proper name that can be related to the Project.

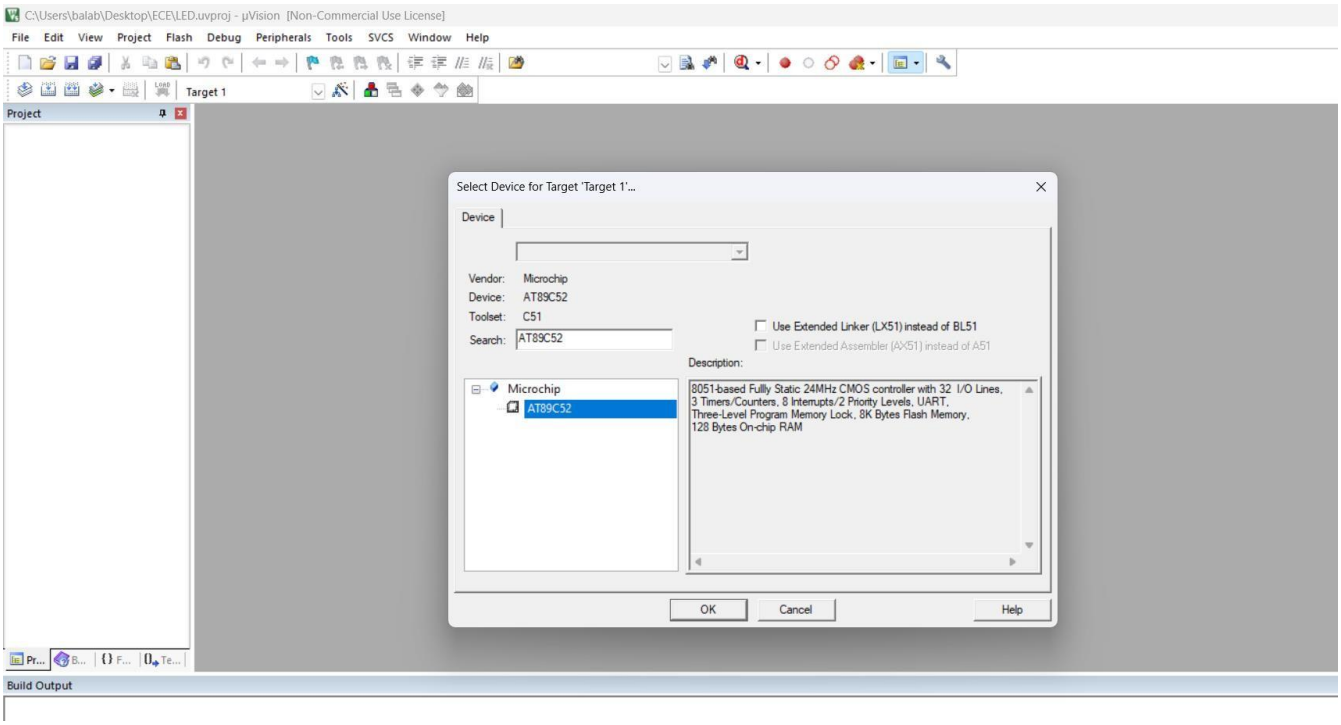
f



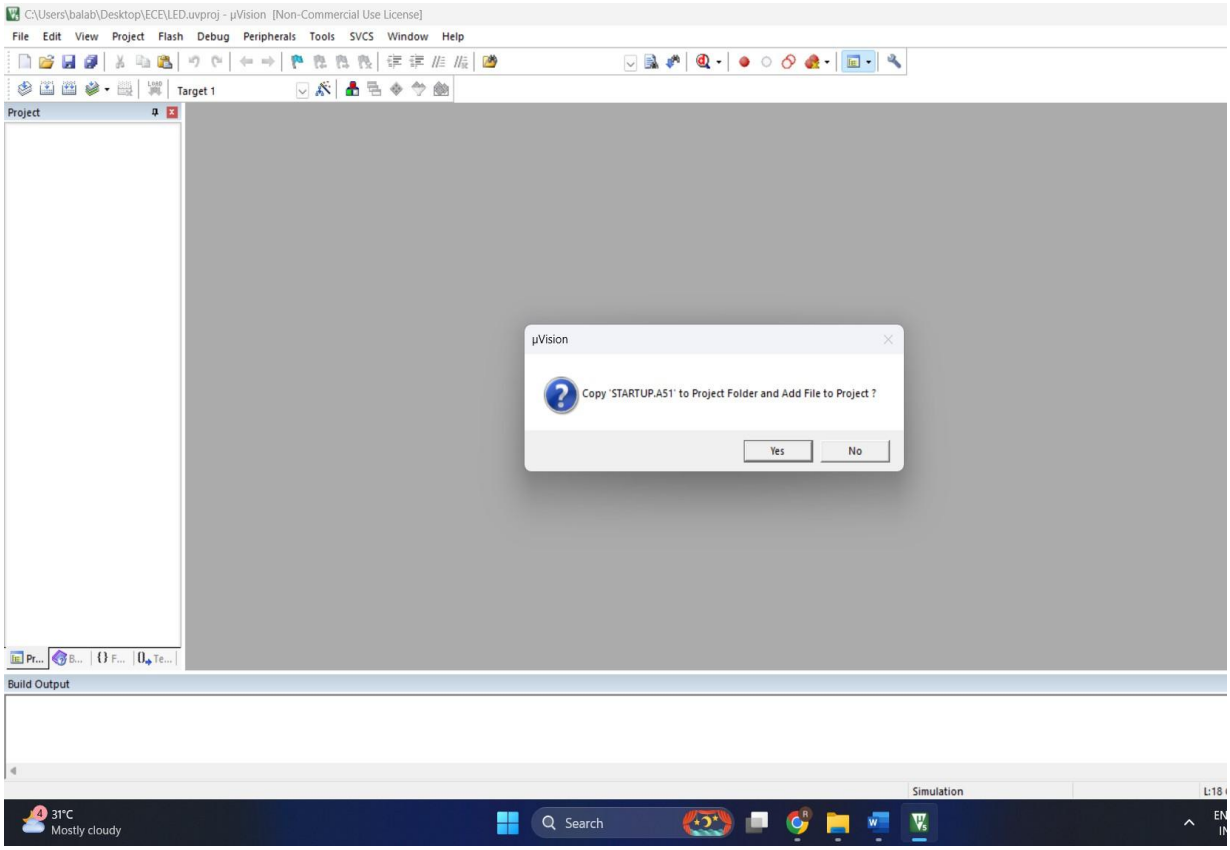
STEP 5: A small window will pop up with the name "**Select Device for Target 'Target 1'**", and select the data base Microchip.



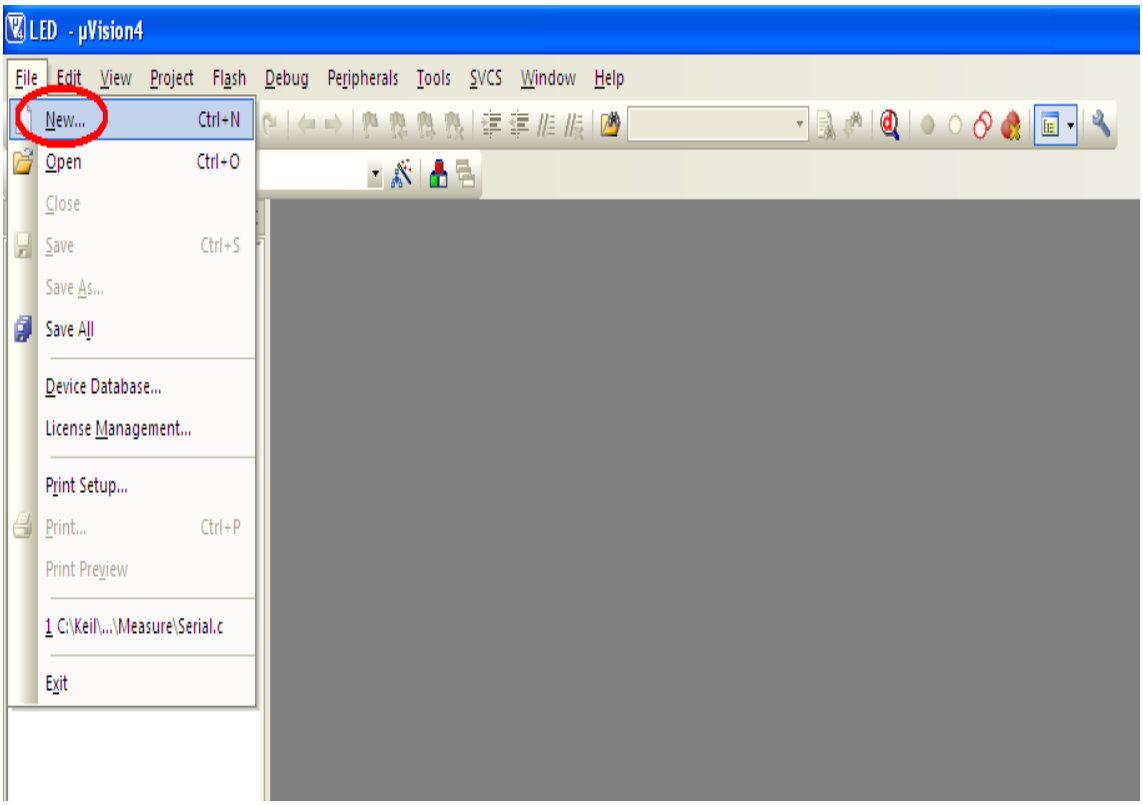
STEP 6: Within the Microchip, select **AT895C2.**



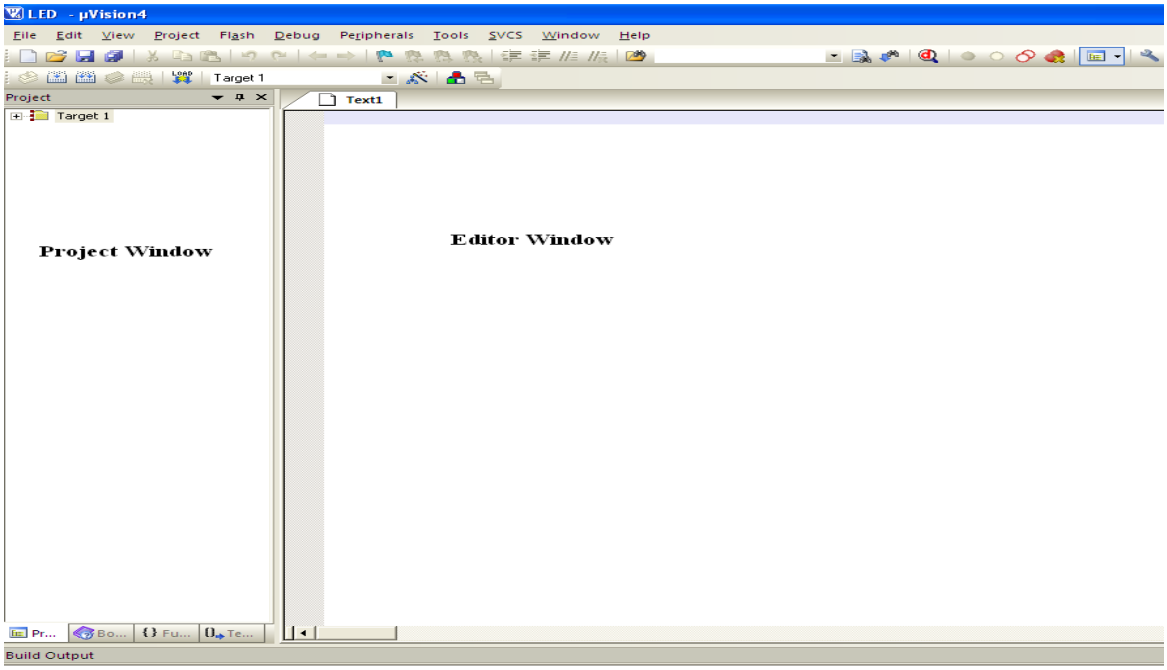
STEP 7: Add Startup file to the project by clicking "**Yes**".



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

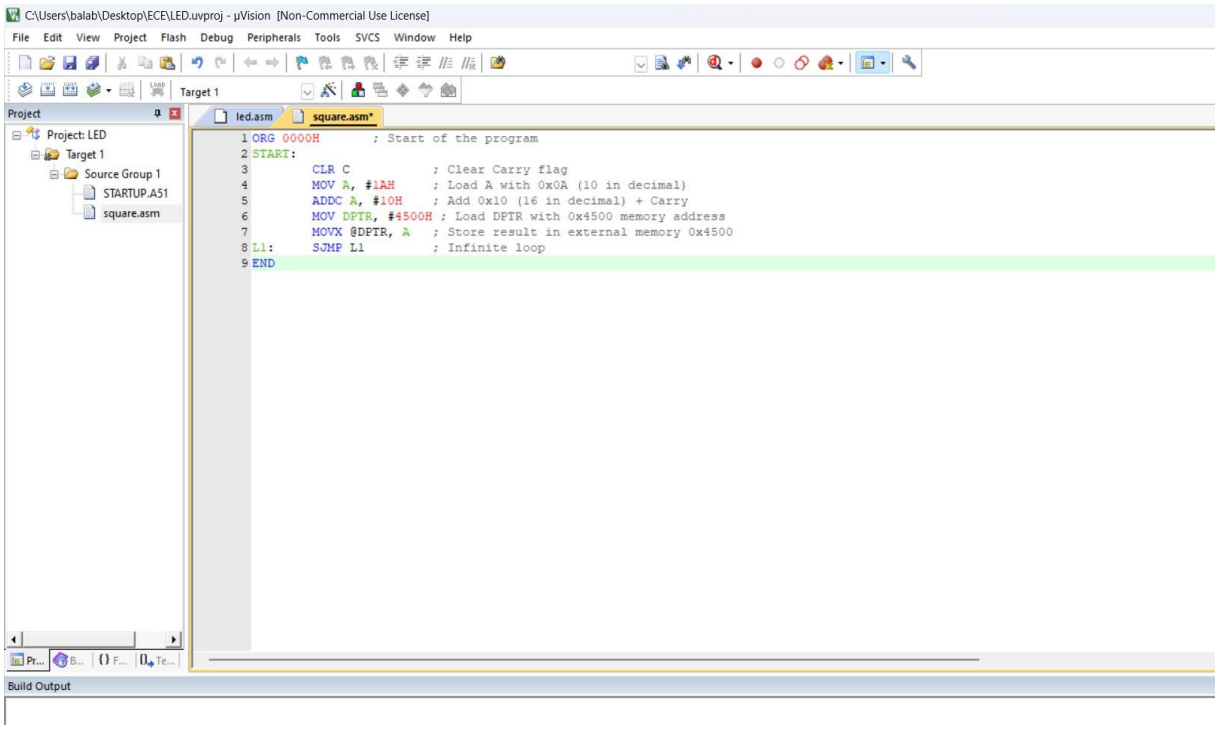


STEP 9: There are the three main windows are available in the keil IDE. First one is Project Workspace, the second one is Editor Window and third one is Output Window.

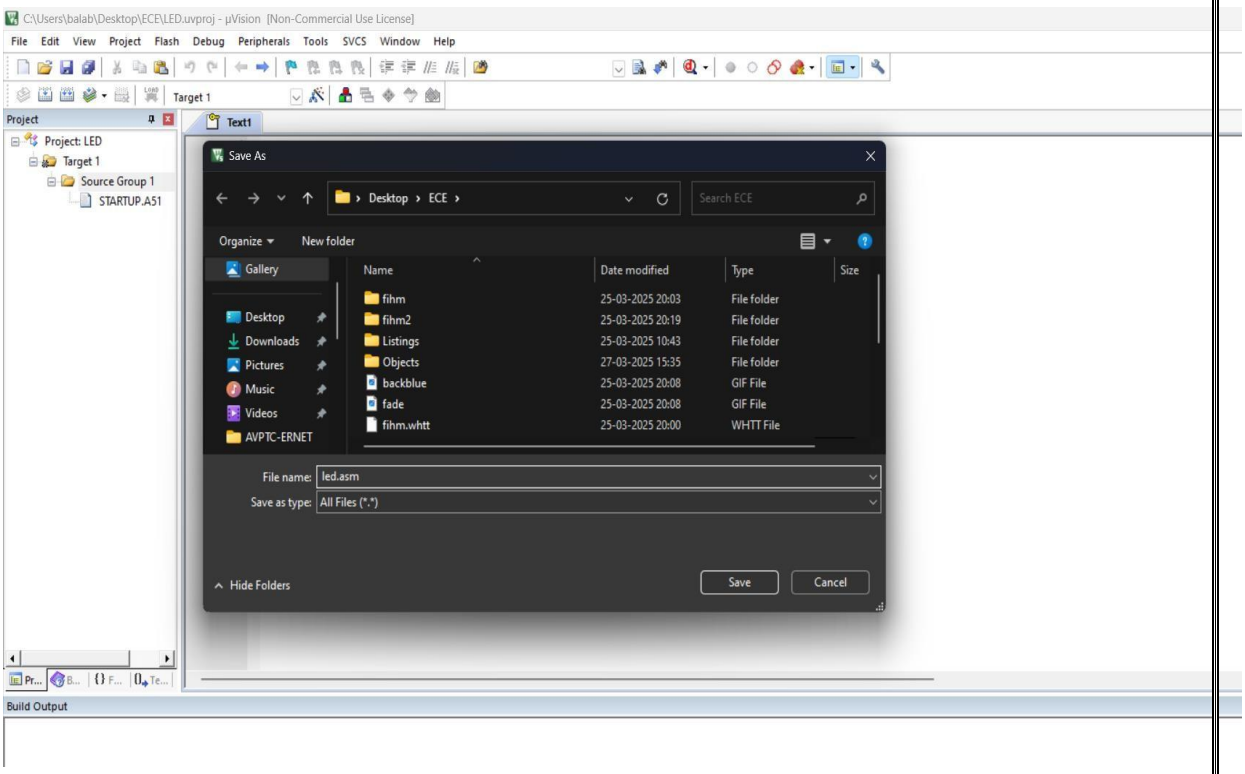


Output Window

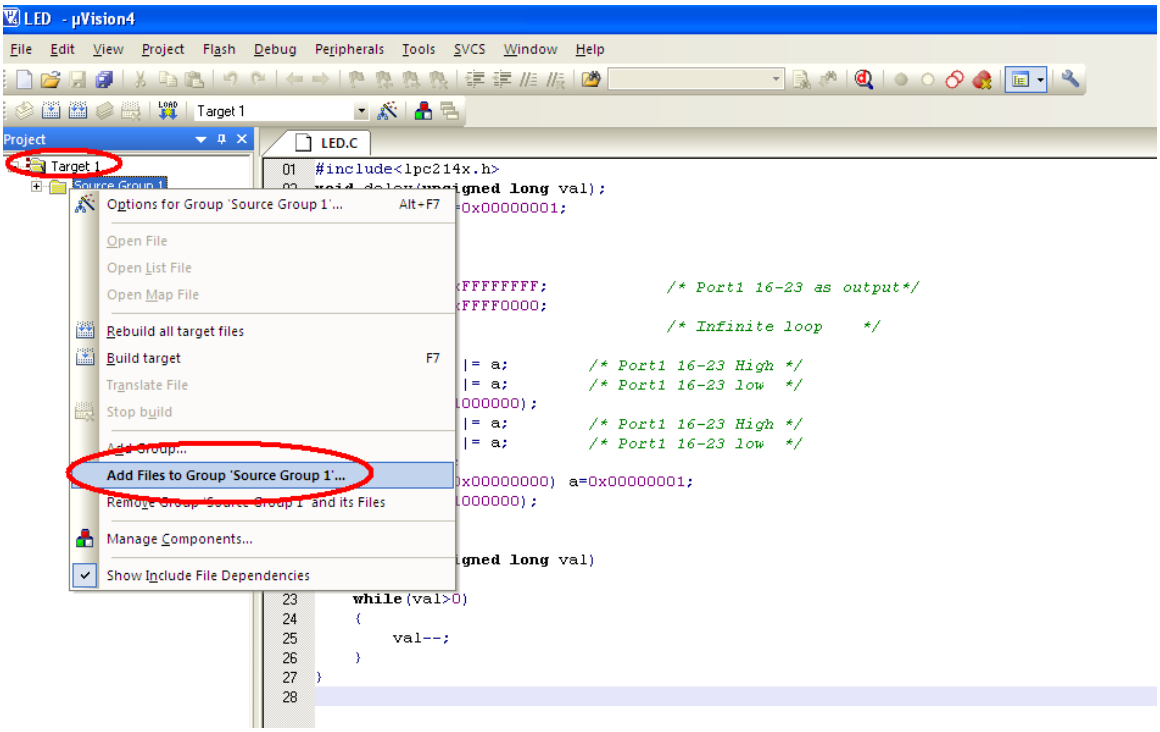
STEP 10: Can start to write *.asm code on the editor window.



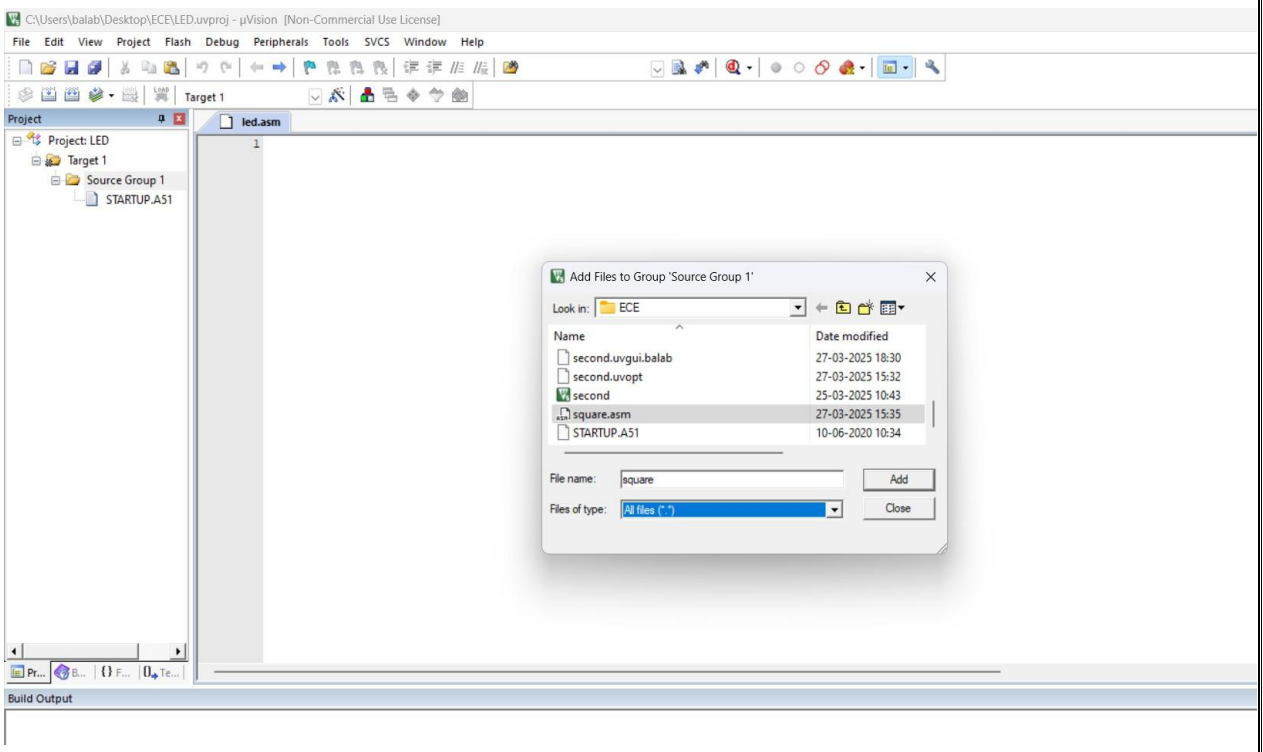
STEP 11: Can save the file, if the program is in "**C**" save as save as "**filename.ASM**".



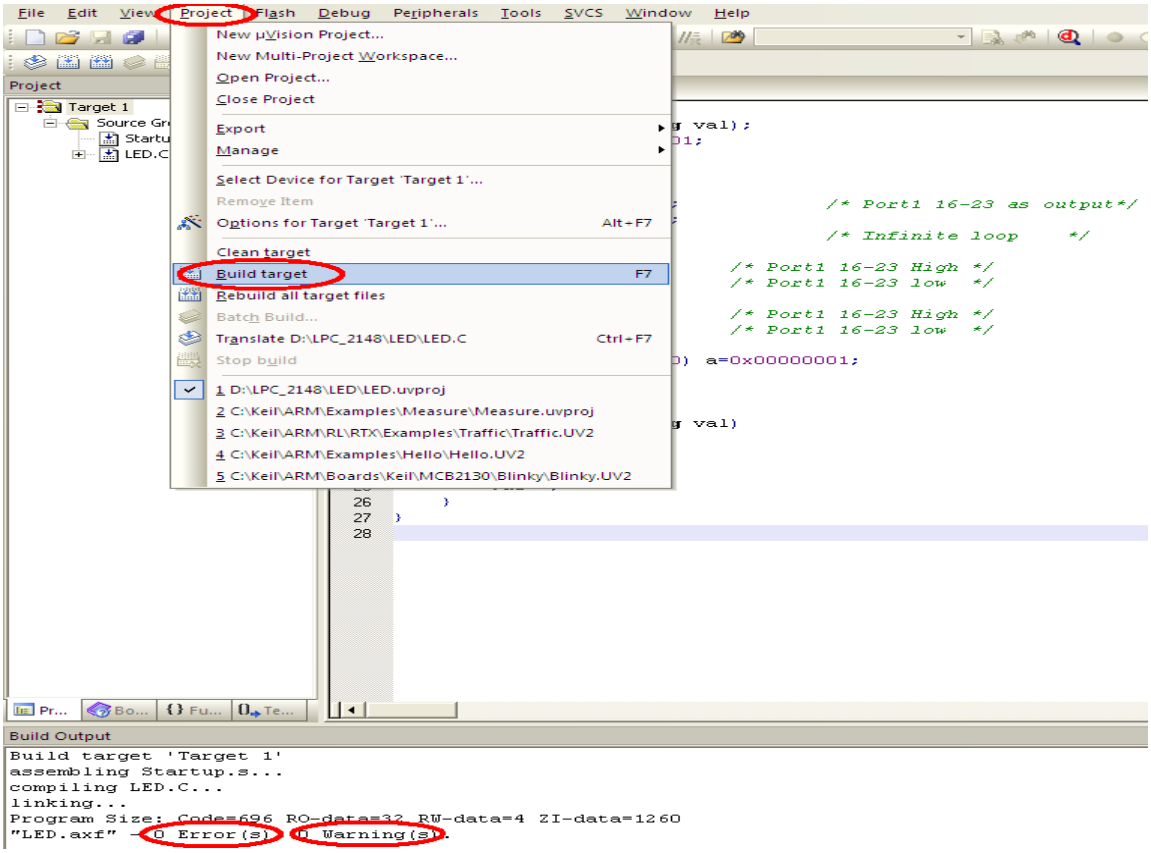
STEP 12: Add this source file to Group1, Go to the "**Project Workspace**" drag the +mark "**Target 1**" in that right click on "**Source Group1**" and click on "**Add Files to Group "Source Group1"**"".



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*).**

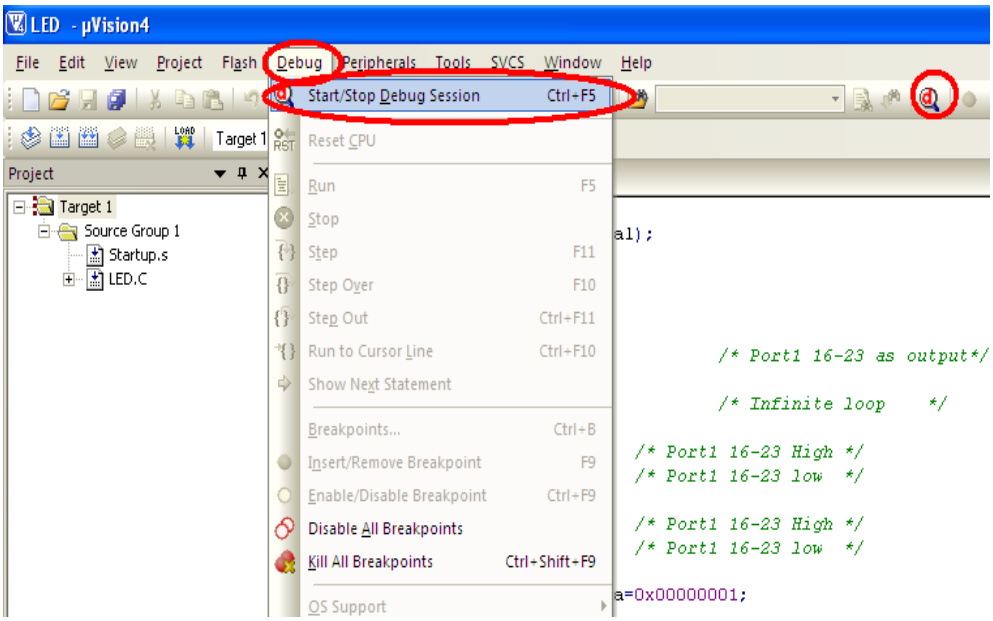


STEP 14: Then go to "**Project**" click on "**Build Target**" or **F7**. Output window will display related error and warning messages.

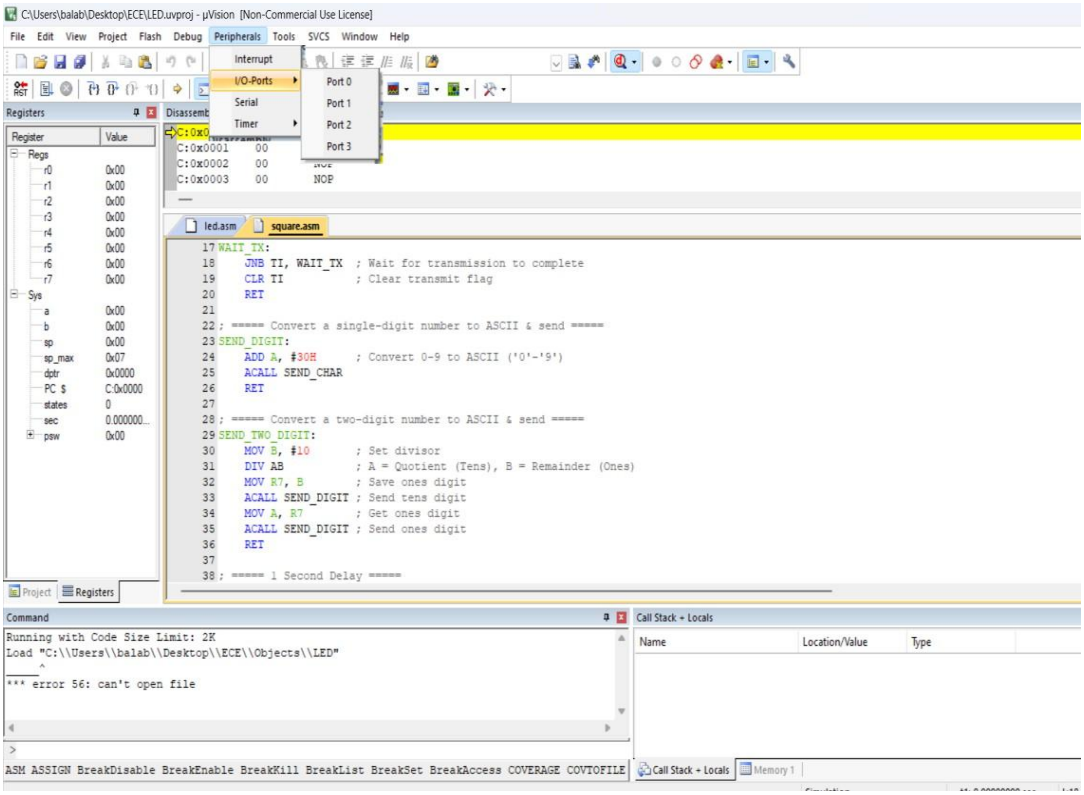


Simulation
Part:

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



EXP NO: 1	PROGRAMMING ARITHMETIC AND LOGICAL OPERATIONS IN 8051
DATE	

AIM:

To write 8051 Assembly Language Program to demonstrate arithmetic and logic operations 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

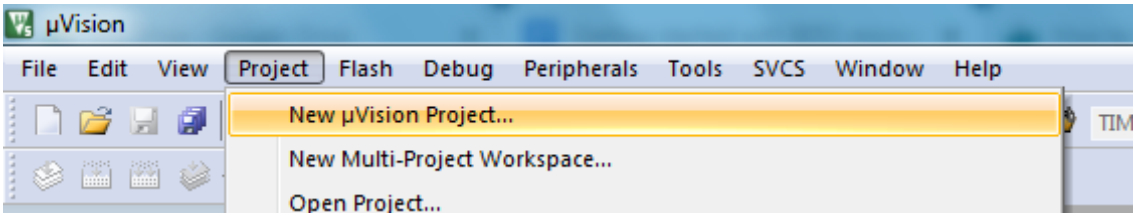
Set up Keil IDE for Programming

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

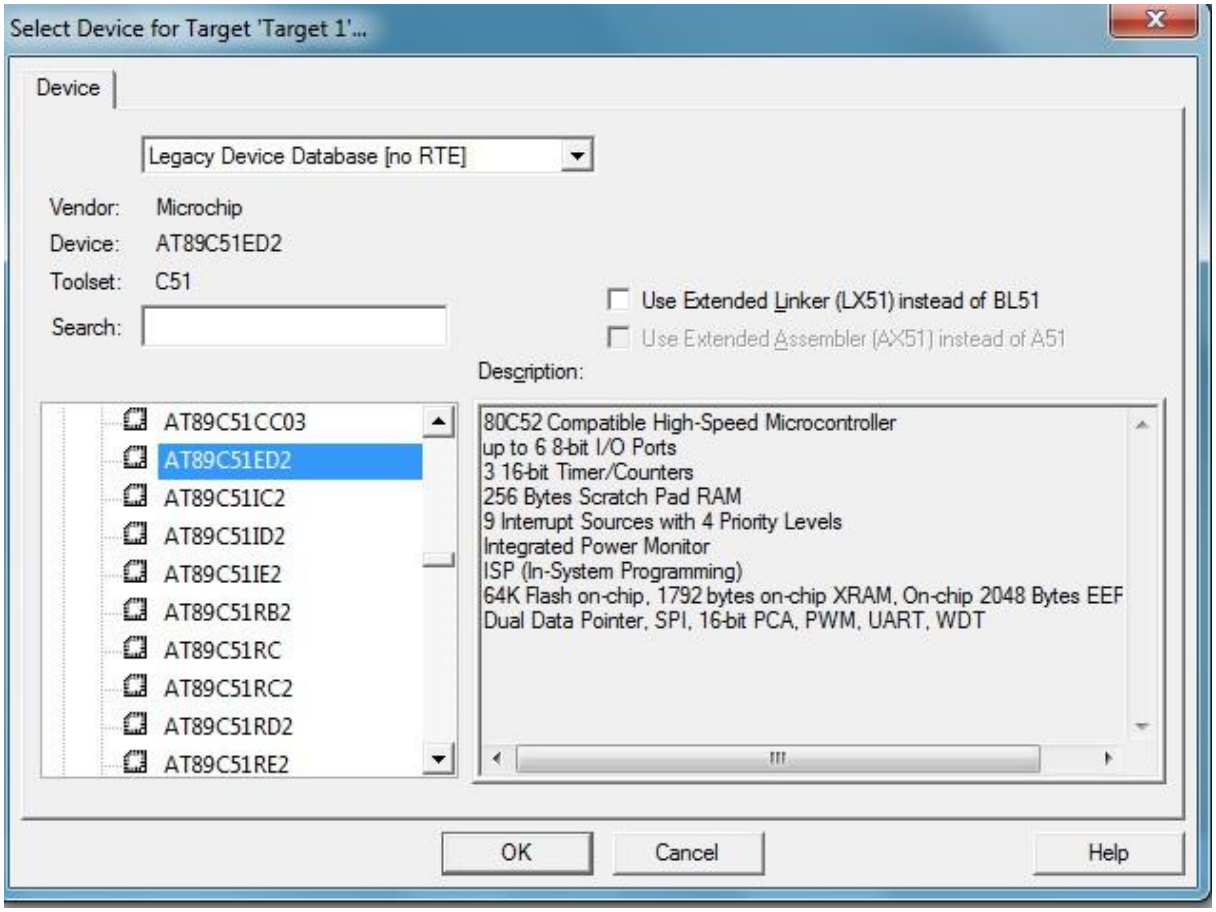
The Keil µVision icon.

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.

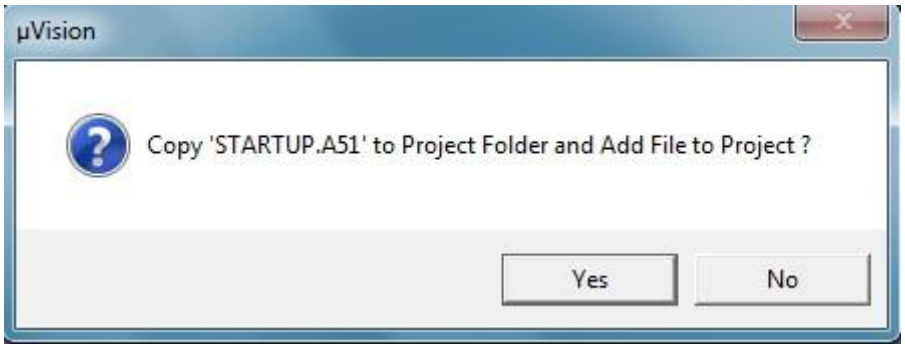


After saving the file, a new window will pop up asking you to select your microcontroller.

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

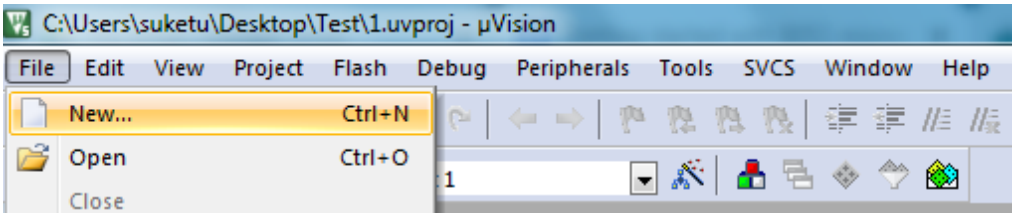


Select '**Yes**' in the next pop-up, as we do not need this file in our project.

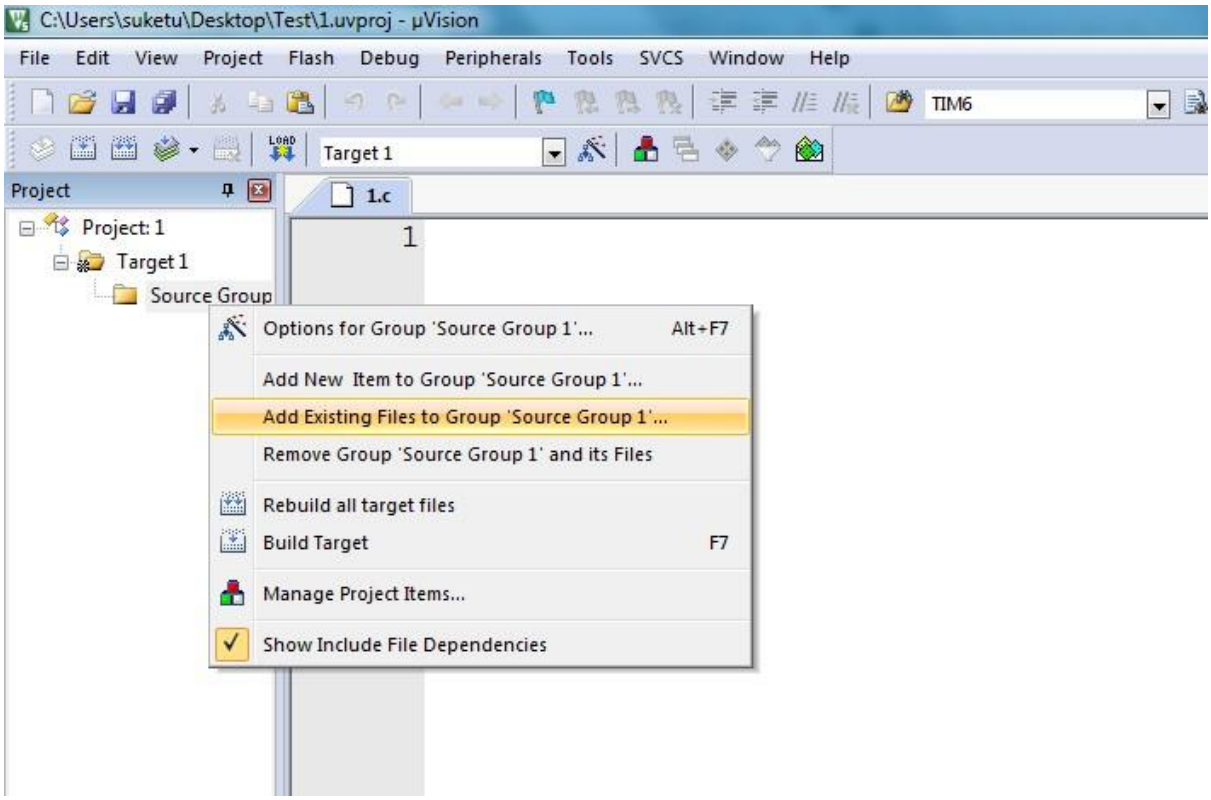


Our project workspace is now ready!

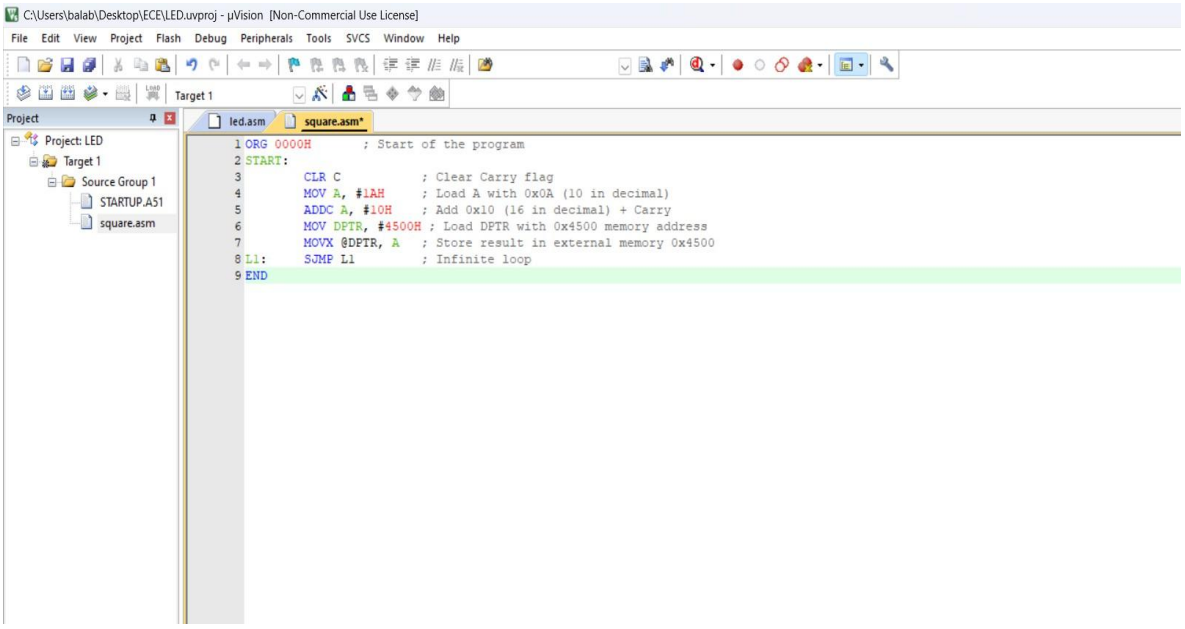
From here, we need to create a file where we can write our C code. Navigate to **File —> New**. Once the file is created, save it with .c extension in the same project folder.



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:

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
```

OUTPUT : Thus the program for 8 bit addition was executed.

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
```

OUTPUT : Thus the program for 8 bit Subtraction was executed.

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
```

OUTPUT : Thus the program for 8 bit multiplication was executed.

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
```

OUTPUT :

Thus the program for 8 bit division was executed.

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
```

OUTPUT : Thus the program for the operations was executed.

Embedded C Program to perform basic arithmetic operations like addition, subtraction, multiplication and division

```
# include<reg51.h>
void main(void)
{
unsigned char x,y,z, a,b,c, d,e,f, p,q,r; //define variables
 //addition
x=0x12; //first 8-bit number
y=0x34; //second 8-bit number
P0=0x00; //declare port 0 as output port
z=x+y; // perform addition
P0=z; //display result on port 0
 //subtraction
a=0x12; //first 8-bit number
b=0x34; //second 8-bit number
P1=0x00; //declare port 1 as output port
c=b-a; // perform subtraction
P1=c; //display result on port 1
 //multiplication
d=0x12; //first 8-bit number
e=0x34; //second 8-bit number
P2=0x00; //declare port 2 as output port
f=e*d; // perform multiplication
P2=f; //display result on port 2
 //division
p=0x12; //first 8-bit number
q=0x34; //second 8-bit number
P3=0x00; //declare port 3 as output port
r=q/p; // perform division
P3=r; //display result on port 3
 while(1);
}
```

Embedded C Program to perform basic Logic operations like OR, AND, EXOR, NOT

```
 # include<reg51.h>
void main(void)
{
unsigned char x,y,z, a,b,c, d,e,f, p,q,r; //define variables
 // AND
x=0x12; //first 8-bit number
y=0x34; //second 8-bit number
P0=0x00; //declare port 0 as output port
z=x&y; // perform AND
P0=z; //display result on port 0
 //OR
a=0x12; //first 8-bit number
b=0x45;//second 8-bit number
P1=0x00; //declare port 1 as output port
```

```
=a|b;// perform OR
P1=c; //display result on port 1
 //EXOR
d=0x12; //first 8-bit number
e=0x34; //second 8-bit number
P2=0x00; //declare port 2 as output port
f=e^d; // perform EXOR
P2=f; //display result on port 2
 //NOT
p=0x12; // 8-bit number
P3=0x00; //declare port 3 as output port
r=~p; // perform Complement
P3=r; //display result on port 3
 while(1);
} c
```

RESULT:

Thus 8051 Assembly Language Program to demonstrate arithmetic and logic operations using Keil simulator is written and executed.

VIVA QUESTIONS:

1. What is the difference between Embedded C and Python in embedded applications?
2. Why is a cross-compiler required for embedded C programming?
3. Explain the role of GPIO configuration in task implementation.
4. What are real-time constraints, and how are they handled in embedded programs?
5. How do delays implemented using software differ from hardware timer delays?

EXP NO: 2	**Interfacing input devices with 8051/PIC16F877A/LPC4088.Keypad Interfacing With 8051 Microcontroller**
DATE	

AIM:

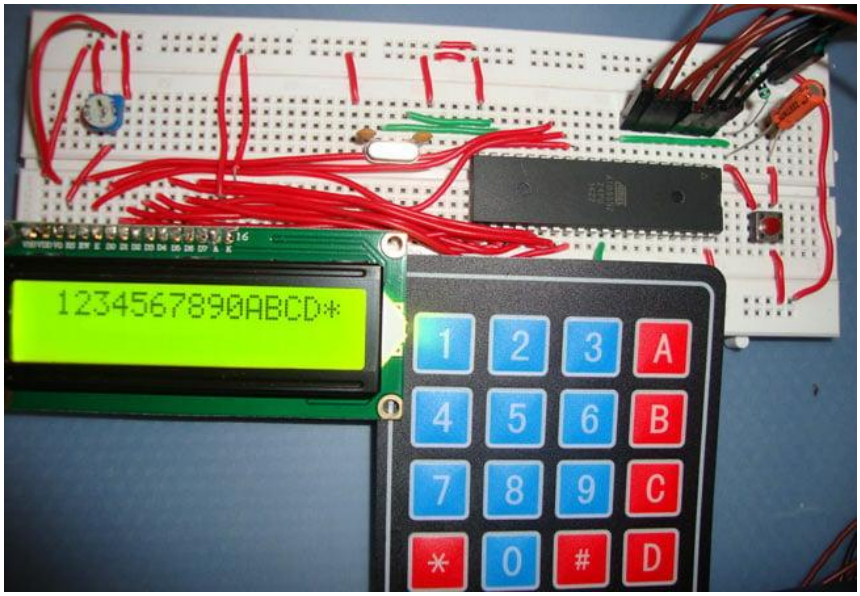
To write and execute program to implement Keypad interfacing with 8051 Microcontroller

SOFTWARE /HARDWARE REQUIRED:

S.No	Requirements	Quantity
1	Microcontroller 8051 (AT89S52/AT89C51)	1
2	Keypad Type 4x4 Matrix (16 keys)	1
3	Display Interface 16x2 LCD (recommended)	1
4	GPIO Pins Required 8 pins (1 complete port)	1
5	Embedded C / Assembly	

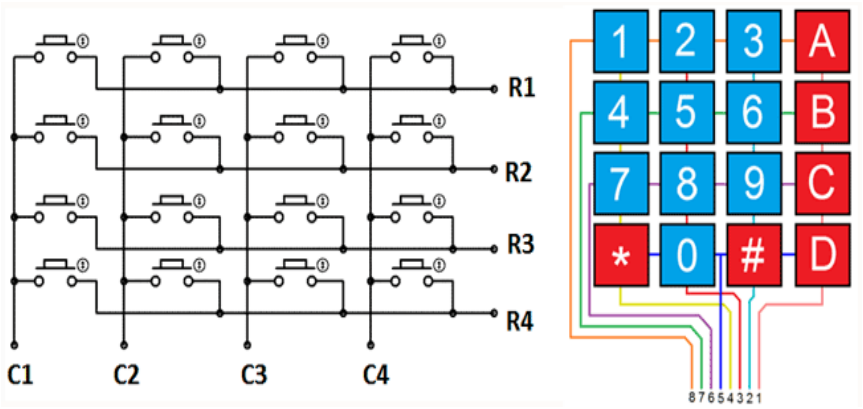
Keypad interfacing with 8051 microcontroller represents one of the fundamental skills for embedded systems developers. Matrix keypads are an input device found in calculators, security systems, ATMs, and various industrial control panels. Keypads are widely used input devices in various electronics and embedded projects. They are used to take inputs in the form of numbers and letters, and feed the same into the system for further processing. In this tutorial, we are going to **interface a 4x4 matrix keypad with an 8051 microcontroller**. This collection of tutorials demonstrates interfacing 4×4 matrix keypads with Arduino, various microcontrollers and development boards, including Keypad Interfacing with Arduino Uno, and Interfacing I2C LCD and 4x4 Keypad with Raspberry Pi Zero W. The guides cover hardware wiring, row-column scanning techniques, and software implementation for detecting key presses. Several examples also integrate 16×2 LCDs, including I2C variants, to display input or computation results, as seen in the Arduino calculator project. Together, these resources provide practical insights into embedded system design, input handling, and efficient microcontroller programming for real-world applications.

decision-making across various sectors including healthcare, agriculture, and manufacturing.



What is a 4x4 Matrix Keypad?

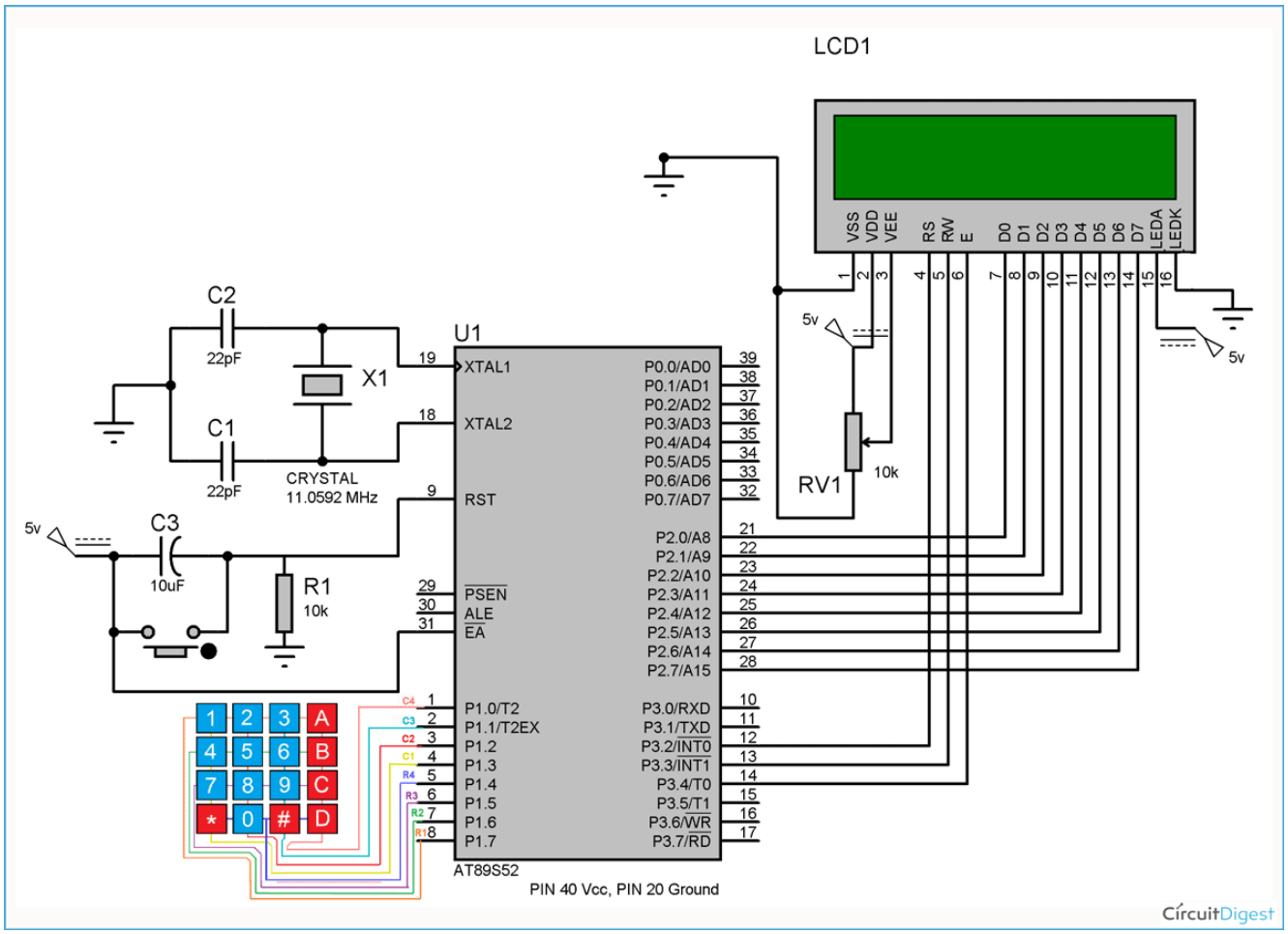
Before we 4x4 matrix keypad interface with the microcontroller, first we need to understand how it works. Matrix keypad consists of a set of Push buttons, which are interconnected. Like in our case, we are using a 4X4 matrix keypad, in which there are 4 push buttons in each of four rows. And the terminals of the push buttons are connected according to the diagram. In the first row, one terminal of all 4 push buttons is connected, and another terminal of the 4 push buttons represents each of the 4 columns. The same goes for each row. So we are getting 8 terminals to connect with a microcontroller. Unlike individual switches that require one GPIO pin each, the matrix configuration allows interfacing of a keyboard with an 8051 microcontroller using only 8 pins total—significantly optimising port usage for embedded applications.



Internal Structure of 4X4 Matrix Keypad

If you want to properly interface to the 4x4 matrix keypad using an 8051 microcontroller pinout, it is important to know the terminal pinout of the keypad. Each keypad has 8 terminals - four for rows (R1-R4) and four for columns (C1-C4). In this matrix, the push buttons are wired such that:

- In each row, one terminal of all four buttons is wired to a common row line
- The other terminal of each button is wired to its column line
- When you press a button, an electrical connection occurs between the corresponding row and column
- By using a matrix arrangement, you can interface the keypad with an 8051 microcontroller by following a systematic row-column scanning process.



Circuit Diagram: Keypad Interfacing with 8051 Microcontroller (AT89S52)

For proper keypad interfacing with the 8051 microcontroller matrix configuration, follow this systematic connection pattern:

As shown in the above circuit diagram, to interface the Keypad, we need to connect 8 terminals of the keypad to any port (8 pins) of the microcontroller. Like we have connected keypad terminals to Port 1 of the 8051. Whenever any button is pressed, we need to get the location of the button, which means the corresponding ROW and COLUMN no. Once we get the location of the button, we can print the character accordingly.

Keypad Terminal	8051 Port Pin	Function
Row 1 (R1)	P1.4	Output
Row 2 (R2)	P1.5	Output
Row 3 (R3)	P1.6	Output
Row 4 (R4)	P1.7	Output
Column 1 (C1)	P1.0	Input
Column 2 (C2)	P1.1	Input
Column 3 (C3)	P1.2	Input
Column 4 (C4)	P1.3	Input

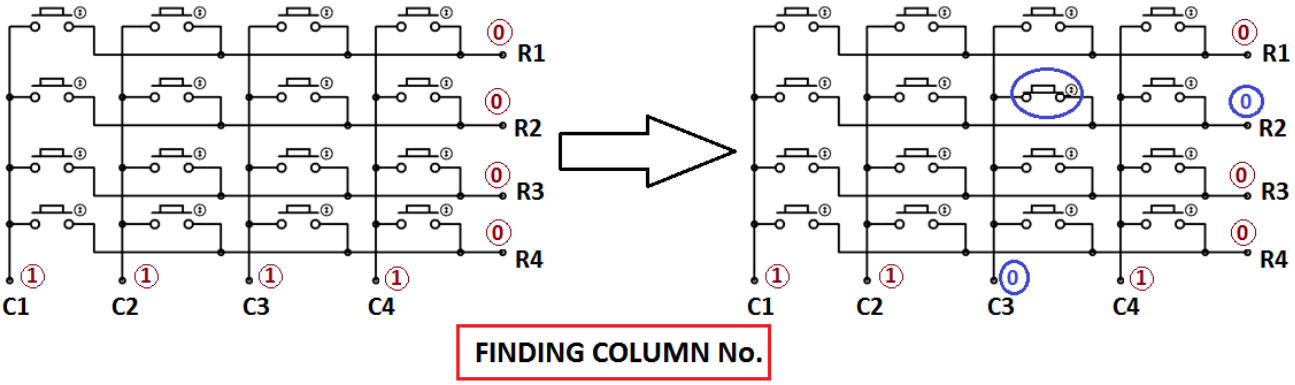
How Does 4x4 Matrix Keypad Scanning Work?

The key detection algorithm for a 4x4 matrix keypad with 8051 microcontroller programming uses a systematic row-column scanning technique. When implementing 8051 keypad interfacing, the microcontroller must identify which specific button was pressed by determining both its row and column position.

Now the question is how to get the location of the pressed button? I am going to explain this in steps below, and also want you to look at the code:

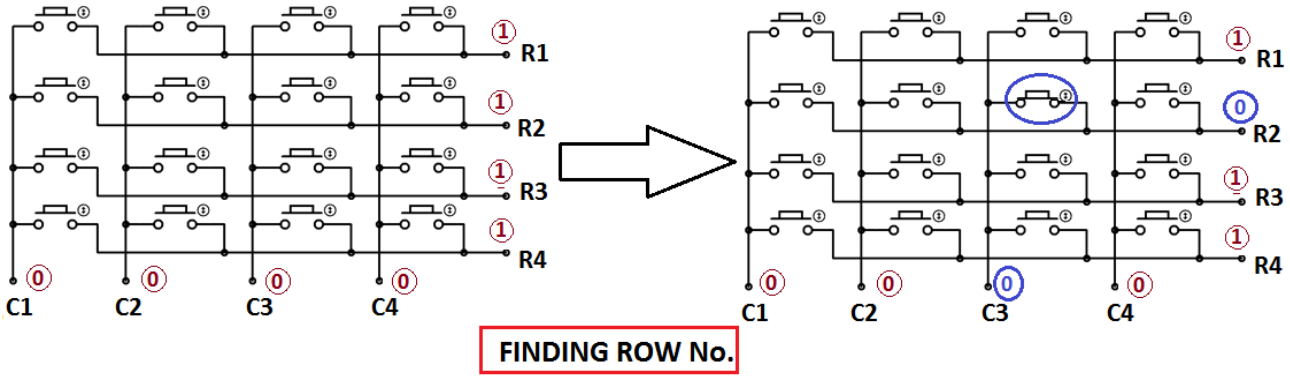
1. First, we have made all the Rows to Logic level 0 and all the columns to Logic level 1.

2. Whenever we press a button, the column and row corresponding to that button get shortened and the corresponding column logic level 0. Because that column becomes connected (shorted) to the row, which is at Logic level 0. So we get the column no. See main() function.



3. Now we need to find the Row no, so we have created four functions corresponding to each column. Like if any button of column one is pressed, we call the function row_finder1() to find the row no.

4. In row_finder1() function, we reversed the logic levels, which means now all the Rows are 1 and columns are 0. Now the Row of the pressed button should be 0 because it has become connected (shorted) to the column whose button is pressed, and all the columns are at 0 logic. So we have scanned all rows for 0.



5. So whenever we find the Row at logic 0, it means that is the row of the pressed button. So now we have column no. (got in step 2) and row no., and we can print the number of that button using the lcd_data function.

Program code:

```
#include<reg51.h>
#define display_port P2      //Data pins connected to port 2 on microcontroller
sbit rs = P3^2;  //RS pin connected to pin 2 of port 3
sbit rw = P3^3;  // RW pin connected to pin 3 of port 3
sbit e =  P3^4;  //E pin connected to pin 4 of port 3
sbit C4 = P1^0;     // Connecting keypad to Port 1
sbit C3 = P1^1;
sbit C2 = P1^2;
sbit C1 = P1^3;
sbit R4 = P1^4;
sbit R3 = P1^5;
sbit R2 = P1^6;
sbit R1 = P1^7;
void msdelay(unsigned int time)  // Function for creating delay in milliseconds.
{
   unsigned i,j ;
   for(i=0;i<time;i++)
   for(j=0;j<1275;j++);
}
void lcd_cmd(unsigned char command)  //Function to send command instruction to
LCD
{
   display_port = command;
   rs= 0;
   rw=0;
   e=1;
   msdelay(1);
   e=0;
}
```

```
void lcd_data(unsigned char disp_data)  //Function to send display data to LCD
{
   display_port = disp_data;
   rs= 1;
   rw=0;
   e=1;
   msdelay(1);
   e=0;
}
 void lcd_init()    //Function to prepare the LCD  and get it ready
{
   lcd_cmd(0x38);  // for using 2 lines and 5X7 matrix of LCD
   msdelay(10);
   lcd_cmd(0x0F);  // turn display ON, cursor blinking
   msdelay(10);
   lcd_cmd(0x01);  //clear screen
   msdelay(10);
   lcd_cmd(0x81);  // bring cursor to position 1 of line 1
   msdelay(10);
}
void row_finder1() //Function for finding the row for column 1
{
R1=R2=R3=R4=1;
C1=C2=C3=C4=0;
if(R1==0)
lcd_data('1');
if(R2==0)
lcd_data('4');
if(R3==0)
lcd_data('7');
```

```
if(R4==0)
lcd_data('*');
}
void row_finder2() //Function for finding the row for column 2
{
R1=R2=R3=R4=1;
C1=C2=C3=C4=0;
if(R1==0)
lcd_data('2');
if(R2==0)
lcd_data('5');
if(R3==0)
lcd_data('8');
if(R4==0)
lcd_data('0');
}
void row_finder3() //Function for finding the row for column 3
{
R1=R2=R3=R4=1;
C1=C2=C3=C4=0;
if(R1==0)
lcd_data('3');
if(R2==0)
lcd_data('6');
if(R3==0)
lcd_data('9');
if(R4==0)
lcd_data('#');
}
void row_finder4() //Function for finding the row for column 4
```

```
{
R1=R2=R3=R4=1;
C1=C2=C3=C4=0;
if(R1==0)
lcd_data('A');
if(R2==0)
lcd_data('B');
if(R3==0)
lcd_data('C');
if(R4==0)
lcd_data('D');
}
void main()
{
   lcd_init();
   while(1)
   {
      msdelay(30);
      C1=C2=C3=C4=1;
        R1=R2=R3=R4=0;
        if(C1==0)
        row_finder1();
        else if(C2==0)
         row_finder2();
         else if(C3==0)
      row_finder3();
      else if(C4==0)
      row_finder4();
   }
}
```

RESULT: **P**rogram to implement Keypad interfacing with 8051 Microcontroller is written and verified.

VIVA QUESTIONS:

1. What types of input devices are commonly interfaced with microcontrollers?
2. How does a microcontroller detect a switch press?
3. What is switch debouncing, and why is it required?
4. Explain polling and interrupt methods for input detection.
5. How are input pins configured in 8051 / PIC microcontrollers?

EXP NO: 3	**Interfacing output devices with 8051/PIC16F877A/LPC4088. LCD Interfacing with 8051 Microcontroller**
DATE	

Aim :

To interface LCD with 8051 Microcontroller

Hardware / Software Requirement:

S.No	Requirements	Quantity
1	Microcontroller 8051 (AT89S52/AT89C51)	1
3	Display Interface 16x2 LCD (recommended)	1
4	GPIO Pins Required 8 pins (1 complete port)	1
5	Embedded C / Assembly	

LCD Interfacing with 8051 Microcontroller

The interfacing of a 16x2 LCD display with 8051 microcontroller is very popular and one of the most important in embedded systems programming. This tutorial will explain the complete LCD interfacing of 8051 C program, circuit connections, and coding strategies in a program with the use of the AT89S52 microcontroller.

The display units are the most important output devices in embedded projects and electronics products. The 16X2 LCD module is designed to display characters per line, which means in this case sixteen characters across two lines. Each character occupies a 5x7 matrix space on display. Interfacing LCD with 8051 microcontroller is a little complicated at the beginning, but when one properly understands the 16-pin configuration and control signals, it becomes direct and easy to design for practical applications.

16x2 LCD Display Module Pinout

Before interfacing LCD with 8051 diagram, there are some important things to note about the 16-pin configuration. In order to perform 16x2 LCD display interfacing with the 8051 microcontroller, we need to connect power pins, control pins, data pins, backlight pins etc.to complete the communication properly.

We can divide it into five categories: Power Pins, contrast pin, Control Pins, Data pins and Backlight pins.

Category	Pin NO.	Pin Name	Function
Power Pins	1	VSS	Ground Pin, connected to Ground
	2	VDD or Vcc	Voltage Pin +5V
Contrast Pin	3	V0 or VEE	Contrast Setting, connected to Vcc thorough a variable resistor.
Control Pins	4	RS	Register Select Pin, RS=0 Command mode, RS=1 Data mode
	5	RW	Read/ Write pin, RW=0 Write mode, RW=1 Read mode
	6	E	Enable, a high to low pulse need to enable the LCD
Data Pins	7-14	D0-D7	Data Pins, Stores the Data to be displayed on LCD or the command instructions
Backlight Pins	15	LED+ or A	To power the Backlight +5V
	16	LED- or K	Backlight Ground

Control Pins Detailed Explanation (RS, RW, E)

The control pins are a key component in interfacing LCD with 8051 microcontrollers. Without learning how they work, the communication to the LCD will not be successful:

All the pins are understandable by their name and functions, except the control pins, so they are explained below:

RS: RS is the register select pin. We need to set it to 1 if we are sending some data to be displayed on the LCD. And we will set it to 0 if we are sending some command instruction like clear the screen (hex code 01).

RW: This is a read/write pin; we will set it to 0 if we are going to write some data on the LCD. And set it to 1, if we are reading from the LCD module. Generally, this is set to 0, because we do not need to read data from the LCD. Only one instruction, "Get LCD status", needs to be read some times. In most LCD interfacing with 8051 C program implementations, this pin is kept at 0 since reading from LCD is rarely required.

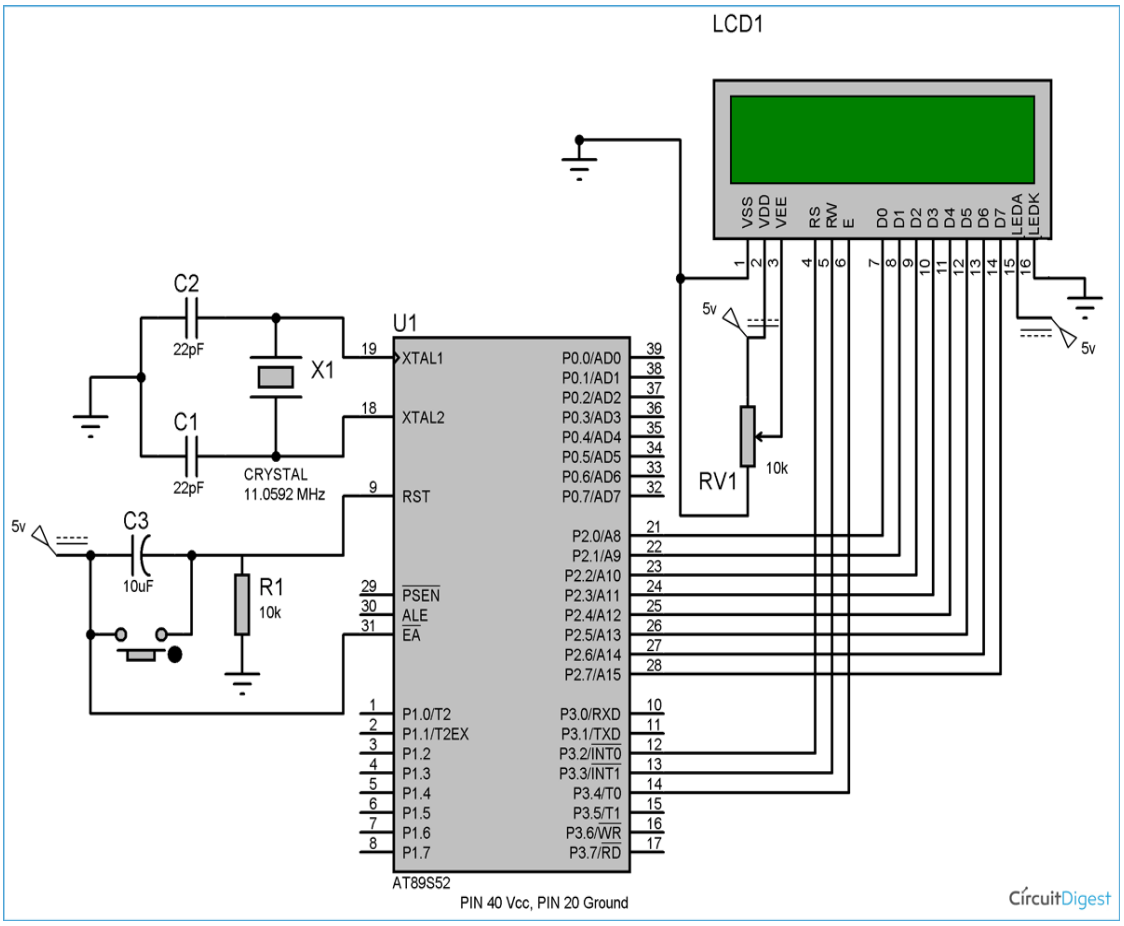
E: This pin is used to enable the module when a high-to-low pulse is given to it. A pulse of 450 ns should be given. That transition from HIGH to LOW makes the module ENABLE.

LCD Interfacing with 8051 Diagram and Hardware Setup

The circuit diagram for interfacing the LCD with 8051 shows all of the hardware connections needed to successfully connect the AT89S52 microcontroller and the 16x2 LCD module. Having your wiring right is important to ensure proper functionality.
There are some preset command instructions on the LCD. We have used them in our program below to prepare the LCD (in the lcd_init() function). Some important command instructions are given below:

Hex Code	Command to LCD Instruction Register
0F	LCD ON, cursor ON
01	Clear display screen
02	Return home
04	Decrement cursor (shift cursor to left)
06	Increment cursor (shift cursor to right)
05	Shift display right
07	Shift display left
0E	Display ON, cursor blinking
80	Force cursor to beginning of first line
C0	Force cursor to beginning of second line
38	2 lines and 5×7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
C1	Jump to second line, position 1
OC	Display ON, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2

LCD Interfacing with 8051 Diagram and Explanation



The circuit diagram for **LCD interfacing with 8051 microcontroller** is shown in the above figure. If you have a basic understanding of 8051, then you must know about EA(PIN 31), XTAL1 & XTAL2, RST pin(PIN 9), Vcc and Ground Pin of the 8051 microcontroller. I have used these Pins in the above circuit. If you don't have any idea about that, then I recommend that you read this Article, LED Interfacing with 8051 Microcontroller, before going through LCD interfacing.

Hardware Connections and Wiring Guide

So besides these above pins, we have connected the data pins (D0-D7) of the LCD to the Port 2 (P2_0–P2_7) microcontroller. And control pins RS, RW and E to the pins 12,13,14 (pins 2,3,4 of port 3) of the microcontroller, respectively.

Power Supply and Contrast Adjustment Setup

PIN 2(VDD) and PIN 15(Backlight supply) of the LCD are connected to a voltage (5V), and PIN 1 (VSS) and PIN 16(Backlight ground) are connected to ground.

Pin 3(V0) is connected to the voltage (Vcc) through a variable resistor of 10k to adjust the contrast of the LCD. The middle leg of the variable resistor is connected to PIN 3, and the other two legs are connected to the voltage supply and Ground.

PROGRAM

```
// Program for LCD Interfacing with 8051 Microcontroller (AT89S52)
#include<reg51.h>
#define display_port P2      //Data pins connected to port 2 on microcontroller
sbit rs = P3^2;  //RS pin connected to pin 2 of port 3
sbit rw = P3^3;  // RW pin connected to pin 3 of port 3
sbit e =  P3^4;  //E pin connected to pin 4 of port 3
void msdelay(unsigned int time)  // Function for creating delay in milliseconds.
{
   unsigned i,j ;
   for(i=0;i<time;i++)
   for(j=0;j<1275;j++);
}
void lcd_cmd(unsigned char command)  //Function to send command instruction to LCD
{
   display_port = command;
   rs= 0;
   rw=0;
   e=1;
   msdelay(1);
   e=0;
}
void lcd_data(unsigned char disp_data)  //Function to send display data to LCD
{
   display_port = disp_data;
   rs= 1;
```

```
    rw=0;

    e=1;

    msdelay(1);

    e=0;

}

 void lcd_init()    //Function to prepare the LCD  and get it ready

{

    lcd_cmd(0x38);  // for using 2 lines and 5X7 matrix of LCD

    msdelay(10);

    lcd_cmd(0x0F);  // turn display ON, cursor blinking

    msdelay(10);

    lcd_cmd(0x01);  //clear screen

    msdelay(10);

    lcd_cmd(0x81);  // bring cursor to position 1 of line 1
    msdelay(10);
}

void main()

{

    unsigned char a[15]="CIRCUIT DIGEST";    //string of 14 characters with a null
terminator.
    int l=0;
    lcd_init();
    while(a[l] != '\0') // searching the null terminator in the sentence
    {
        lcd_data(a[l]);
        l++;

        msdelay(50);

    }
}
```

RESULT: **P**rogram to implement LCD (Output) interfacing with 8051 Microcontroller is written and verified.

VIVA QUESTIONS:

1. What is the difference between current sourcing and current sinking?
2. Why is a driver circuit required for interfacing motors or relays?
3. Explain how an LED is interfaced with a microcontroller.
4. What precautions should be taken while interfacing high-power output devices?
5. How does port direction configuration affect output operation?

EXP NO: 4	**Implementation of recurring tasks using the timers and interrupts of 8051/PIC microcontroller/ LPC4088 - TIMER PROGRAMING**
DATE	

AIM:

To toggle all bits of port-1 continuously with some delay using timer0 in mode -1 to generate the delay

Hardware / Software Requirement:

S.No	Requirements	Quantity
1	Microcontroller 8051 (AT89S52/AT89C51)	1
3	Display Interface 16x2 LCD (recommended)	1
4	GPIO Pins Required 8 pins (1 complete port)	1
5	Embedded C / Assembly	

PROCEDURE:

1. Open Keil u Vision 5 on the computer, click on Project, and select New u Vision Project.
2. Enter the file name as "timer" and save it.
3. In the window search AT89C51 and select and press OK.
4. In the project window select TARGET and check for Source Group 1 and right-click and select "Add new item for Source Group 1".
5. Generate it and click the 'C' file and name it as "timer" and add it.
6. Now type the program for the timer and save it. Click project window > Source group>timer.c and right click > Build Project OR Use Shortcut key F7. The output shows 0 error
7. To run the program> Click Debug > Start/Stop Session>OK.
8. To check output>Click Peripherals>I/O Ports>Port 1. Click Again to Debug and RUN. Check whether the bits are continuously Toggling.

CONVERTING THE C FILE INTO A HEX FILE:

1. To generate HEX File>Right Click Target>Click target target 1> Window appears.
2. Enter the frequency as 11.0592 MHz > click Output in that window and create Hex file "CHECK THE BOX">OK

3. Right Click Timer>Build project> Create Hex file

PROTEUS 8 PROFESSIONAL:

1. New Project> Rename as the timer.extension>Next>Next>Next>Finish
2. Select "Pick Devices icon" > Search for AT89C51> OK> Place it.
3. Select "Pick Devices icon" > Search for LED >Place 8 LEDs and joint each one from P1-P8.
4. In terminal mode>Click Ground > and connect all the LEDs to the ground.
5. Double-click the microcontroller>Enter Frequency 11.0952.
6. Program file>timer.hex>check it>Open >OK>RUN (Clicking the play button in the bottom of the window).

PROGRAM:

```
#include<reg51.h>
void DELAY
(void); void
main(void)
{
        while(1) {
                P1=0x55; // P1=0101 0101
                DELAY ();
                P1=0xAA; //  P1=1010 1010
                DELAY ();
                }
        }
void DELAY (void)
{
        TMOD=0x01; // timer 0 in mode 1- 16 bit timer
        TH0=0x4B;
        TL0=0xFE;
        TR0=1; // timer0 run bit=1 timer start
        while(TF0==0); // timer flag bit = 0
        TR0=0; // timer0 run bit is reset
        TF0=0; // again timer flag bit is made
        0
}
```

CALCULATION:

i) Timer time:

fc=11.0592 MHz

fT=fc/12=11.0592/12=0.9216MHz (In 8051 clock will be internally divided by 12)

Tc=1/fT = 1/0.9216=1.085 μ sec

ii) number of clock pulses = 50m sec/1.085 μ sec= 46.0829 *10^(3)= 46082

iii) TH 0 & TL 0 : MODE 1 (16 BIT)

$2^{16} = 65536-46082=(19454)_{10} = (4BFE)_{16}$

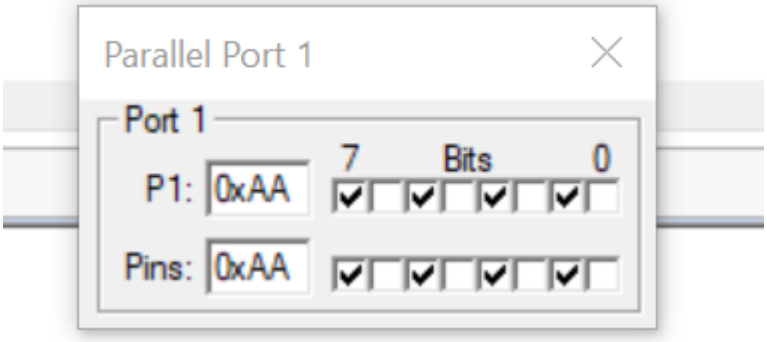
TMOD REGISTER INITIALIZATION

TIMER 1				**TIMER 0**			
GATE	**C/T'**	**M1**	**M0**	**GATE**	**C/T'**	**M1**	**M0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**

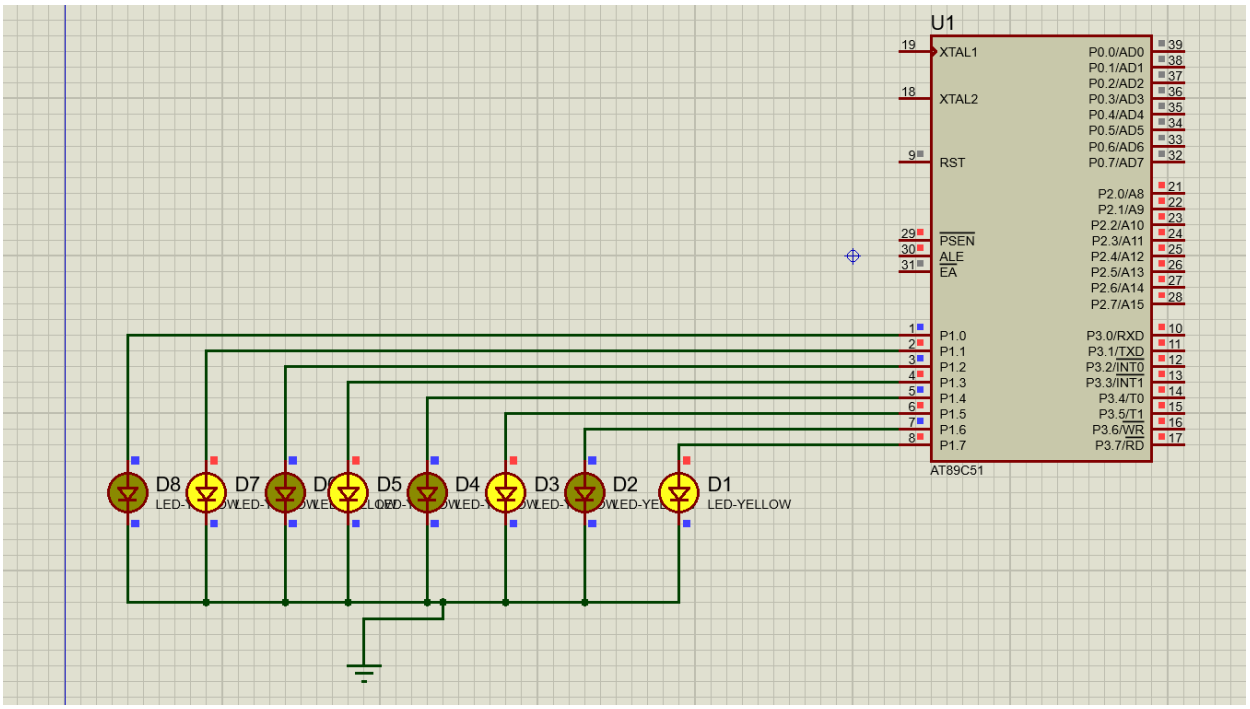
TMOD=0X01

OUTPUTS:

CONTINUOUS TOGGLING OF BITS AT PORT-1



SIMULATION IN PROTEUS:



RESULT:

Thus toggling of all bits of port-1 continuously with some delay using timer0 in mode -1 is generated.

VIVA QUESTIONS:

1. What is the function of a timer in a microcontroller?
2. Differentiate between timer mode and counter mode.
3. What is an interrupt, and why is it used?
4. Explain interrupt priority with an example.
5. How does timer overflow generate an interrupt?

EXP NO: 5. A	**Interfacing DAC with 8051 microcontroller.**
DATE	

AIM:

To Interface DAC with 8051 microcontrollers.

Apparatus/Software Required

S.No	Requirements	Quantity
1	Microcontroller 8051 (AT89S52/AT89C51)	1
3	MC1408 (DAC0808)	1
4	CRO / DSO	1
5	Embedded C / Assembly	

Theory

The Digital to Analog converter (DAC) is a device, that is widely used for converting digital pulses to analog signals. There are two methods of converting digital signals to analog signals. These two methods are binary weighted method and R/2R ladder method. In this article we will use the MC1408 (DAC0808) Digital to Analog Converter. This chip uses R/2R ladder method. This method can achieve a much higher degree of precision. DACs are judged by its resolution. The resolution is a function of the number of binary inputs. The most common input counts are 8, 10, 12 etc. Number of data inputs decides the resolution of DAC. So if there are n digital input pin, there are 2^n analog levels. So 8 input DAC has 256 discrete voltage levels.

The MC1408 DAC (or DAC0808)

In this chip the digital inputs are converted to current. The output current is known as I_{out} by connecting a resistor to the output to convert into voltage. The total current provided by the I_{out} pin is basically a function of the binary numbers at the input pins D_0 - D_7 (D_0 is the LSB and D_7 is the MSB) of DAC0808 and the reference current I_{ref}. The following formula is showing the function of I_{out}

$$I_{Out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

The I_{ref} is the input current. This must be provided into the pin 14. Generally 2.0mA is used as I_{ref} We connect the I_{out} pin to the resistor to convert the current to voltage. But in real life it may cause inaccuracy since the input resistance of the load will also affect the output voltage. So practically I_{ref} current input is isolated by connecting it to an Op-Amp with R_f = 5K as feedback resistor. The feedback resistor value can be changed as per requirement.

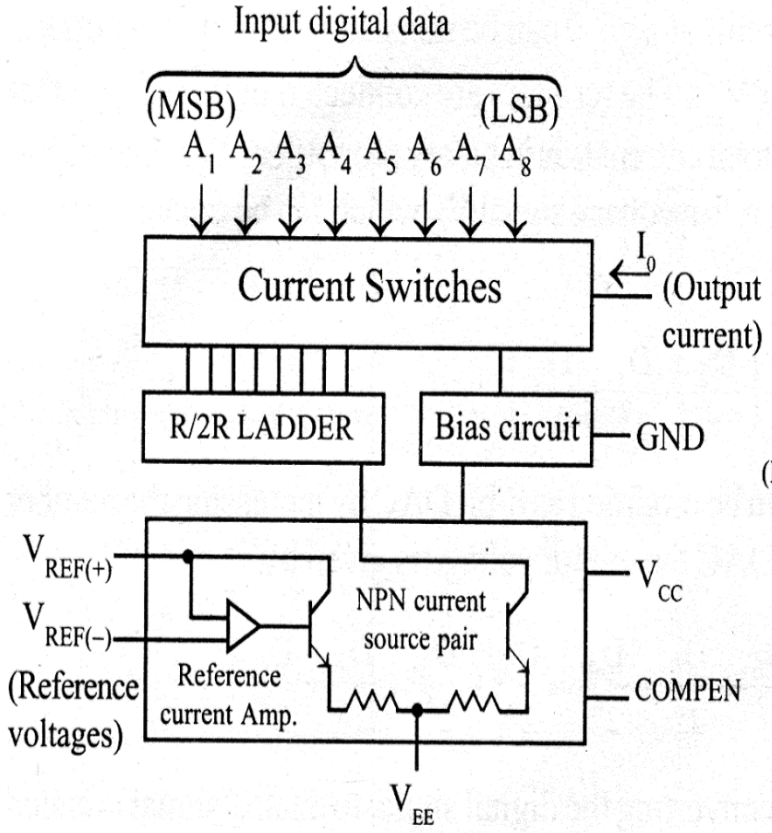


Fig a : Block diagram of DAC0808

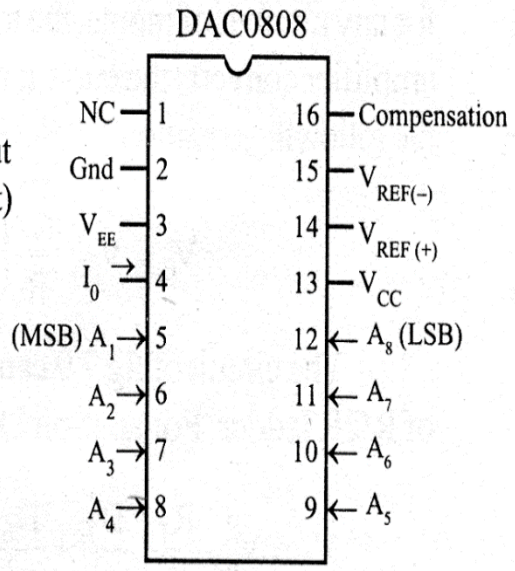


Fig b : Pin configuration

Note : The direction of output current is towards the IC

Generating Sinewave using DAC and 8051 Microcontroller

For generating sinewave, at first we need a look-up table to represent the magnitude of the sine value of angles between 0Â° to 360Â°. The sine function varies from -1 to +1.

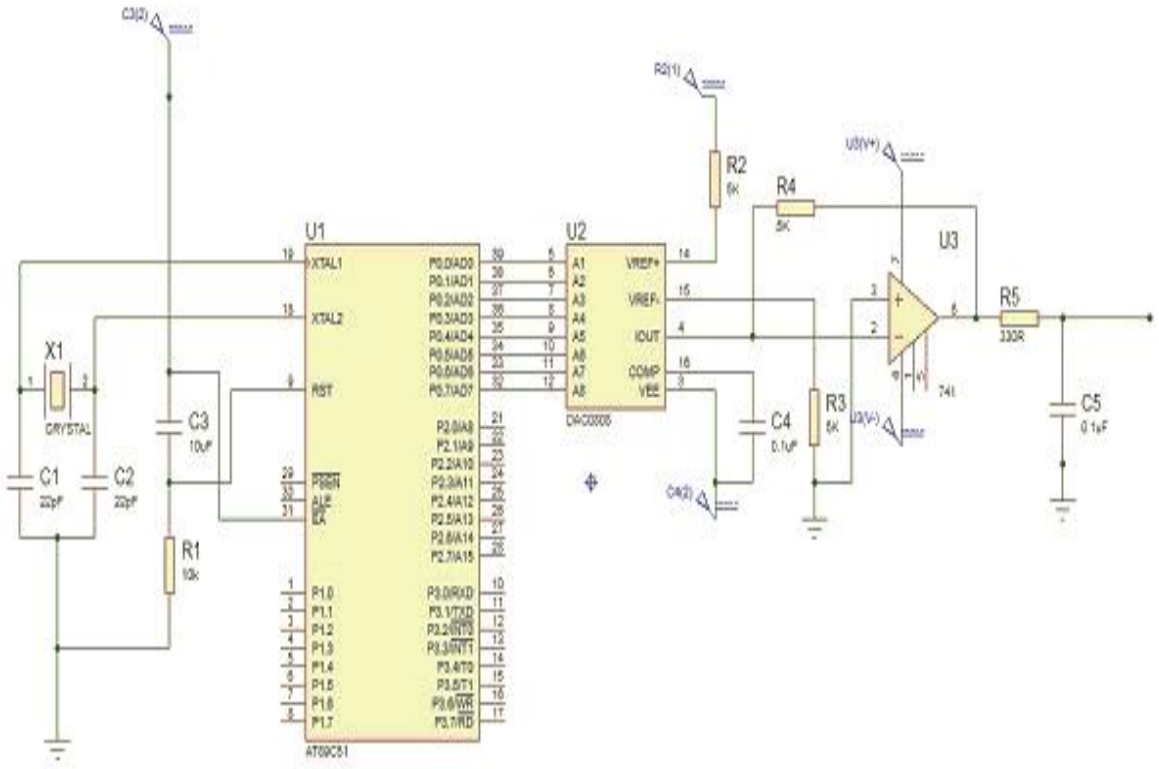
In the table only integer values are applicable for DAC input. In this example we will consider 30Â° increments and calculate the values from degree to DAC input. We are assuming full-scale voltage of 10V for DAC output. We can follow this formula to get the voltage ranges.

$$V_{out}=5+5\sin(\omega t)$$

Let us see the lookup table according to the angle and other parameters for DAC.

Angle(in ?)	sin?	V_{out} (Voltage Magnitude)	Values sent to DAC
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128

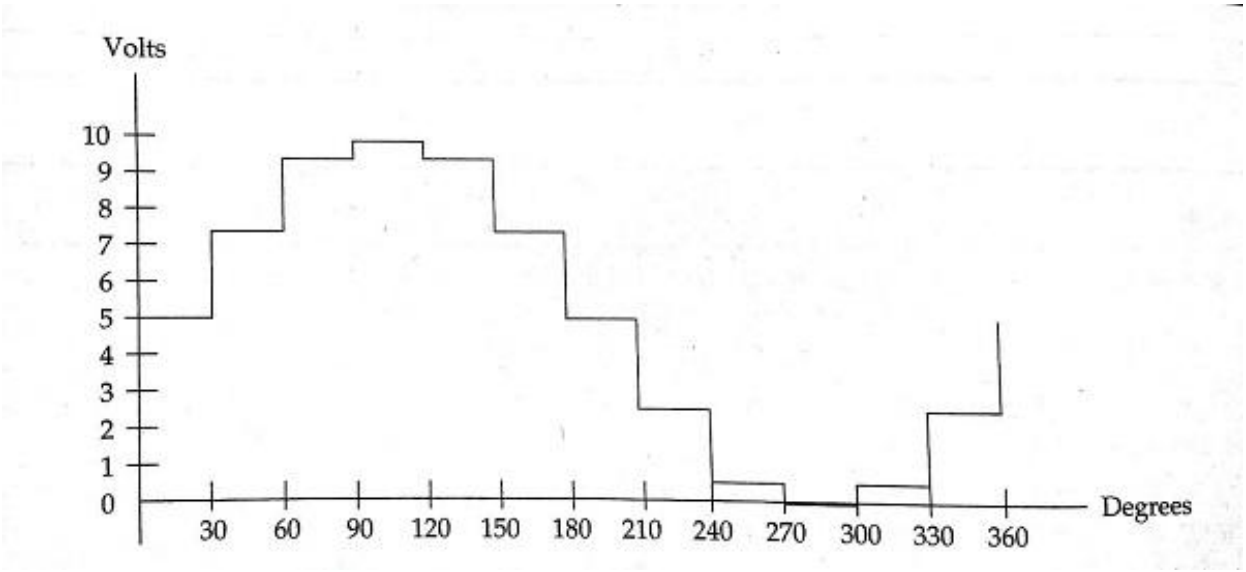
Circuit Diagram



PROGRAM

```
#include<reg51.h>
sfr DAC = 0x80; //Port P0 address
void main(){
  int sin_value[12] = {128,192,238,255,238,192,128,64,17,0,17,64};
  int i;
  while(1){
    //infinite loop for LED blinking
    for(i = 0; i<12; i++){
      DAC = sin_value[i];
    }
  }
}
```

Output



RESULT: Thus the Interface of DAC with 8051 microcontrollers is programmed and executed.

VIVA QUESTIONS:

1. What is the need for ADC in microcontroller systems?
2. Explain the working principle of an ADC.
3. What are resolution and conversion time in ADC?
4. Why is DAC required in embedded systems?
5. How is analog voltage reconstructed from digital data using DAC?

EXP NO: 5. B	**Interfacing ADC with 8051 microcontroller.**
DATE	

AIM:

To Interface ADC with 8051 microcontrollers.

Apparatus/Software Required

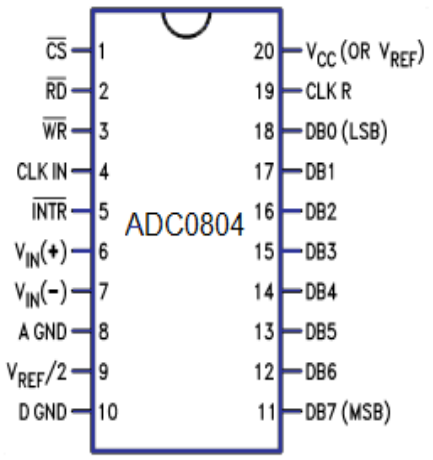
S.No	Requirements	Quantity
1	Microcontroller 8051 (AT89S52/AT89C51)	1
3	ADC0804	1
5	Embedded C / Assembly	

Theory

ADC (Analog to digital converter) forms a very essential part in many embedded projects and this article is about interfacing an ADC to 8051 embedded controller. ADC 0804 is the ADC used here and before going through the interfacing procedure, we must neatly understand how the ADC 0804 works.

ADC 0804.

ADC0804 is an 8 bit successive approximation analogue to digital converter from National semiconductors. The features of ADC0804 are differential analogue voltage inputs, 0-5V input voltage range, no zero adjustment, built in clock generator, reference voltage can be externally adjusted to convert smaller analogue voltage span to 8 bit resolution etc. The pin out diagram of ADC0804 is shown in the figure below.



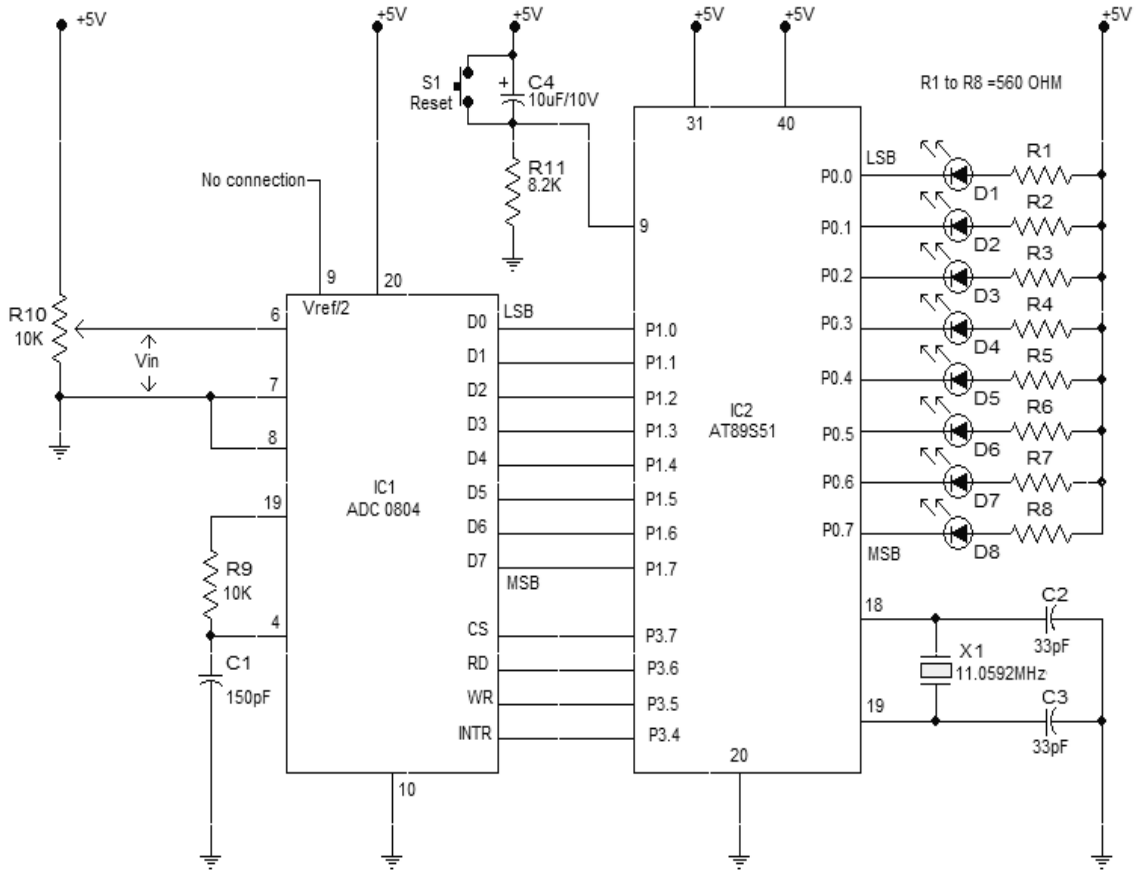
The voltage at Vref/2 (pin9) of ADC0804 can be externally adjusted to convert smaller input voltage spans to full 8 bit resolution. Vref/2 (pin9) left open means input voltage span is 0-5V and step size is 5/255=19.6V. Have a look at the table below for different Vref/2 voltages and corresponding analogue input voltage spans.

Vref/2 (pin9) (volts)	Input voltage span (volts)	Step size (mV)
Left open	0 – 5	5/255 = 19.6
2	0 – 4	4/255 = 15.69
1.5	0 – 3	3/255 = 11.76
1.28	0 – 2.56	2.56/255 = 10.04
1.0	0 – 2	2/255 = 7.84
0.5	0 – 1	1/255 = 3.92

Steps for converting the analogue input and reading the output from ADC0804.

- Make CS=0 and send a low to high pulse to WR pin to start the conversion.
- Now keep checking the INTR pin. INTR will be 1 if conversion is not finished and INTR will be 0 if conversion is finished.
- If conversion is not finished (INTR=1) , poll until it is finished.
- If conversion is finished (INTR=0), go to the next step.
- Make CS=0 and send a high to low pulse to RD pin to read the data from the ADC.

Circuit diagram



Interfacing ADC to 8051

The figure above shows the schematic for interfacing ADC0804 to 8051. The circuit initiates the ADC to convert a given analogue input , then accepts the corresponding digital data and displays it on the LED array connected at P0. For example, if the analogue input voltage Vin is 5V then all LEDs will glow indicating 11111111 in binary which is the equivalent of 255 in decimal. AT89s51 is the microcontroller used here. Data out pins (D0 to D7) of the ADC0804 are connected to the port pins P1.0 to P1.7 respectively. LEDs D1 to D8 are connected to the port pins P0.0 to P0.7 respectively. Resistors R1 to R8 are current limiting resistors. In simple words P1 of the microcontroller is the input port and P0 is the output port. Control signals for the ADC (INTR, WR, RD and CS) are available at port pins P3.4 to P3.7 respectively. Resistor R9 and capacitor C1 are associated with the internal clock circuitry of the ADC. Preset resistor R10 forms a voltage divider which can be used to apply a particular input analogue voltage to the ADC. Push button S1, resistor R11 and capacitor C4 forms a debouncing reset mechanism. Crystal X1 and capacitors C2,C3 are associated with the clock circuitry of the microcontroller.

Program

```
ORG 00H

MOV P1,#11111111B // initiates P1 as the input port

MAIN: CLR P3.7 // makes CS=0

      SETB P3.6 // makes RD high

      CLR P3.5 // makes WR low

      SETB P3.5 // low to high pulse to WR for starting conversion

WAIT: JB P3.4,WAIT // polls until INTR=0

      CLR P3.7 // ensures CS=0

      CLR P3.6 // high to low pulse to RD for reading the data from
ADC

      MOV A,P1 // moves the digital data to accumulator

      CPL A // complements the digital data (*see the notes)

      MOV P0,A // outputs the data to P0 for the LEDs

      SJMP MAIN // jumps back to the MAIN program

      END
```

Notes

- The entire circuit can be powered from 5V DC.
- ADC 0804 has active low outputs and the instruction CPL A complements it t0 have a straight forward display. For example, if input is 5V then the output will be 11111111 and if CPL A was not used it would have been 00000000 which is rather awkward to see.

Result

Successfully interfaced ADC 0804 with 8051Microcontroller.

VIVA QUESTIONS:

1. What is the need for DAC in microcontroller systems?
2. Explain the working principle of an DAC.
3. What are resolution and conversion time in DAC?
4. Why is DAC required in embedded systems?
5. How is analog voltage reconstructed from digital data using DAC?

EXP NO: 6	PWM generation using PIC16F877A/LPC4088.
DATE	

AIM:

- To Generate PWM Signals using PIC16F877A

Apparatus/Software Required

S.No	Requirements	Quantity
1	Microcontroller **PIC16F877A**	1
3	DSO / CRO	1
5	Embedded C / Assembly	

Theory

PWM signal generation is a vital tool in every embedded engineers arsenal, they come in very handy for lot of applications like controlling the position of servo motor, switching few power electronic ICs in converters/invertors and even for a simple LED brightness control. In PIC microcontrollers PWM signals can be generated using the *Compare, Capture and PWM* (CCP) modules by setting the required Registers.

The **PIC16F877A** can generate PWM signals only on pins RC1 and RC2, if we use the CCP modules. But we might encounter situations, where we need more pins to have PWM functionality. For instance in my case, I want to control 6 RC servo motors for my robotic arm project for which the CCP module is hopeless. In these scenarios we can **program the GPIO pins to produce PWM signals using timer modules**. This way we can generate as many PWM signals with any required pin. There are also other hardware hacks like using a multiplexer IC, but why invest on hardware when the same can be achieved though programming. So in this tutorial we will learn **how to convert a PIC GPIO pin into a PWM pin** and to test it we will simulate it on proteus with digital oscilloscope and also **control the position of Servo motor using the PWM signal** and vary its duty cycle by varying a potentiometer.

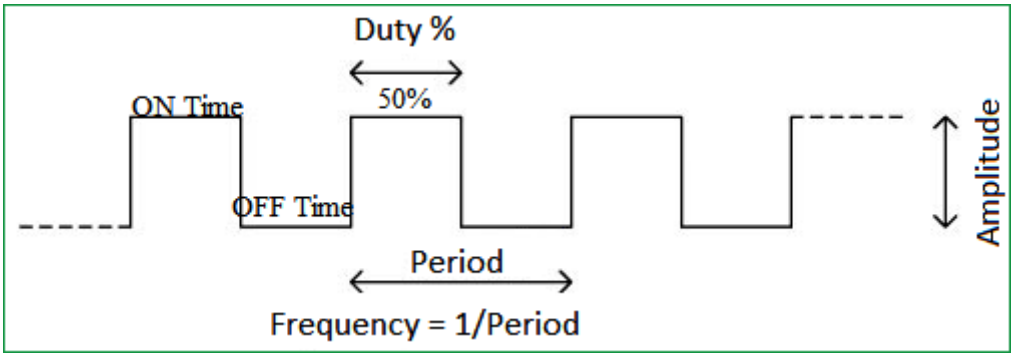
What is a PWM Signal?

Before we get into the details, let us brush up a bit on what PWM Signals are. **Pulse Width Modulation (PWM)** is a digital signal which is most commonly used in control circuitry. This signal is set high (5v) and low (0v) in a predefined time and speed. The time during which the signal stays high is called the *"on time"* and the time during which the signal stays low is called the *"off time"*. There are two important parameters for a PWM as discussed below:

Duty cycle of the PWM

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle.

Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)



Frequency of a PWM

The frequency of a PWM signal determines how fast a PWM completes one period. One Period is complete ON and OFF of a PWM signal as shown in the above figure. In our tutorial we will set a frequency of 5KHz.

Calculating Duty Cycle for PWM

To generate PWM signal on a GPIO pin we have to simply turn it on and off for a pre-defined time. But it is not as simple as it sounds. This on time and off time should be accurate for every cycle so we simply cannot use delay functions, hence we employ a timer module and use the timer interrupts**.** Also we have to consider the duty cycle and the frequency of the PWM signal that we generate. The following variable names are used in program to define the parameters.

This is the **standard formulae where frequency is simply the reciprocal of time**. The value of frequency has to be decided and set by the user based on his/her application requirement.

$$T\_TOTAL = (1/PWM\_Frequency)$$

When the user changes the Duty cycle value, our program should automatically adjust the T_ON time and T_OFF time according to that. So the above formulae can be used to **calculate T_ON based on the value of Duty Cycle and T_TOTAL**.

$$T\_ON = (Duty\ Cycle*T\_TOTAL)/100$$

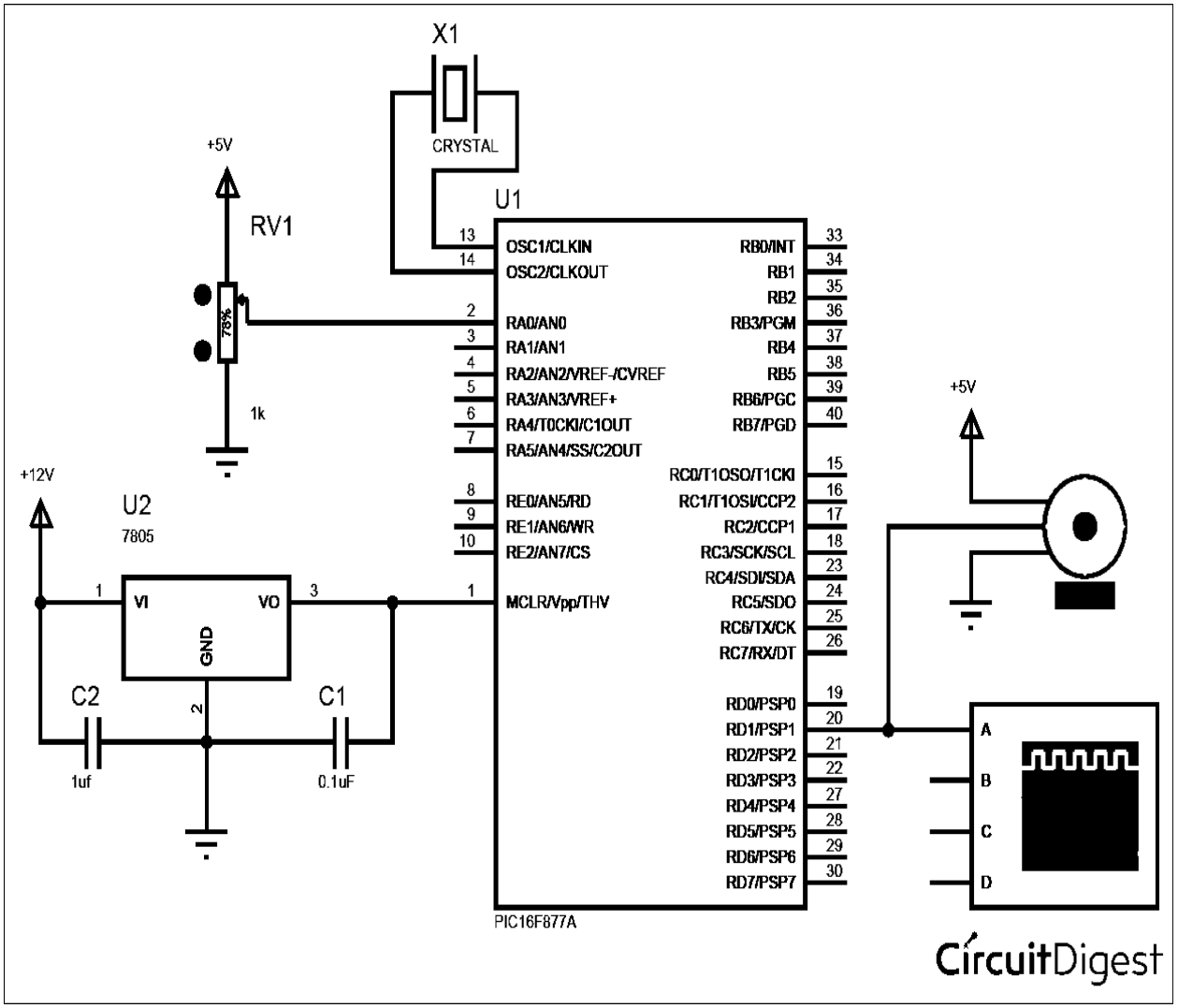
Since the Total time of the PWM signal for one full cycle will be the sum of on time and off time. We can **calculate the off time T_OFF** as shown above.

$$T\_OFF = T\_TOTAL - T\_ON$$

Variable Name	Refers to
PWM_Frequency	Frequency of the PWM Signal
T_TOTAL	Total time taken for one complete cycle of PWM
T_ON	On time of the PWM signal
T_OFF	Off time of the PWM signal
Duty_cycle	Duty cycle of the PWM signal

With these formulae in mind we can begin programming the PIC microcontroller. The program involves the PIC Timer Module and PIC ADC Module to create a PWM signal based with a varying Duty cycle according to the ADC value form the POT. If you are new to using these modules then it is strongly recommended to read the appropriate tutorial by clicking on the hyperlinks.

Circuit Diagram



PROGRAM

```
/*
 * File:   PIC_GPIO_PWM.c
 * Author: Aswinth
 *
 * Created on 17 October, 2018, 11:59 AM
 */
// CONFIG
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial
Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data
EEPROM code protection off)
#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write
protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code
protection off)
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
#include <xc.h>
#define _XTAL_FREQ 20000000
#define PWM_Frequency 0.05 // in KHz (50Hz)
//TIMER0    8-bit  with 64-bit Prescalar
//$$RegValue = 256-((Delay * Fosc)/(Prescalar*4))  delay in sec and Fosc in hz  ->Substitute
value of Delay for calculating RegValue
int POT_val; //variable to store value from ADC
int count; //timer variable
int T_TOTAL = (1/PWM_Frequency)/10; //calculate Total Time from frequency (in milli
sec)) //2msec
int T_ON=0; //value of on time
int Duty_cycle; //Duty cycle value
```

```
void ADC_Initialize() //Prepare the ADC module
{
  ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected
  ADCON1 = 0b11000000; // Internal reference voltage is selected
}
unsigned int ADC_Read(unsigned char channel) //Read from ADC
{
  ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits
  ADCON0 |= channel<<3; //Setting the required Bits
  __delay_ms(2); //Acquisition time to charge hold capacitor
  GO_nDONE = 1; //Initializes A/D Conversion
  while(GO_nDONE); //Wait for A/D Conversion to complete
  return ((ADRESH<<8)+ADRESL); //Returns Result
}
void interrupt timer_isr()
{
   if(TMR0IF==1) // Timer flag has been triggered due to timer overflow -> set to overflow
for every 0.1ms
   {
      TMR0 = 248;     //Load the timer Value
      TMR0IF=0;       // Clear timer interrupt flag
      count++; //Count increments for every 0.1ms -> count/10 will give value of count in ms
   }
   if (count <= (T_ON) )
      RD1=1;
   else
      RD1=0;
   if (count >= (T_TOTAL*10) )
      count=0;
}
void main()
{
/*****Port Configuration for Timer ******/
   OPTION_REG = 0b00000101;  // Timer0 with external freq and 64 as prescalar // Also
Enables PULL UPs
```

```
    TMR0=248;       // Load the time value for 0.0001s; delayValue can be between 0-256 only
    TMR0IE=1;       //Enable timer interrupt bit in PIE1 register
    GIE=1;          //Enable Global Interrupt
    PEIE=1;         //Enable the Peripheral Interrupt
    /***********_____***********/
    /*****Port Configuration for I/O ******/
    TRISD = 0x00; //Instruct the MCU that all pins on PORT D are output
    PORTD=0x00; //Initialize all pins to 0
    /***********_____***********/
    ADC_Initialize();
    while(1)
    {
      POT_val = (ADC_Read(0)); //Read the value of POT using ADC
      Duty_cycle = (POT_val * 0.0976); //Map 0 to 1024 to 0 to 100
      T_ON = ((Duty_cycle * T_TOTAL)*10 / 100); //Calculate On Time using formulae unit
in milli seconds
      __delay_ms(100);
    }
}
```

RESULT: Successfully generated PWM signal using PIC16F877A Microcontroller.

VIVA QUESTIONS:

1. What is PWM, and where is it used?
2. How does duty cycle affect PWM output?
3. Explain the role of timers in PWM generation.
4. What is the advantage of PWM over analog control?
5. How is PWM used for speed control of a DC motor?

EXP NO: 7	INTERFACING RTC WITH MICROCONTROLLER.
DATE	

AIM:

- To interface RTC with microcontroller.

Apparatus / Software Required

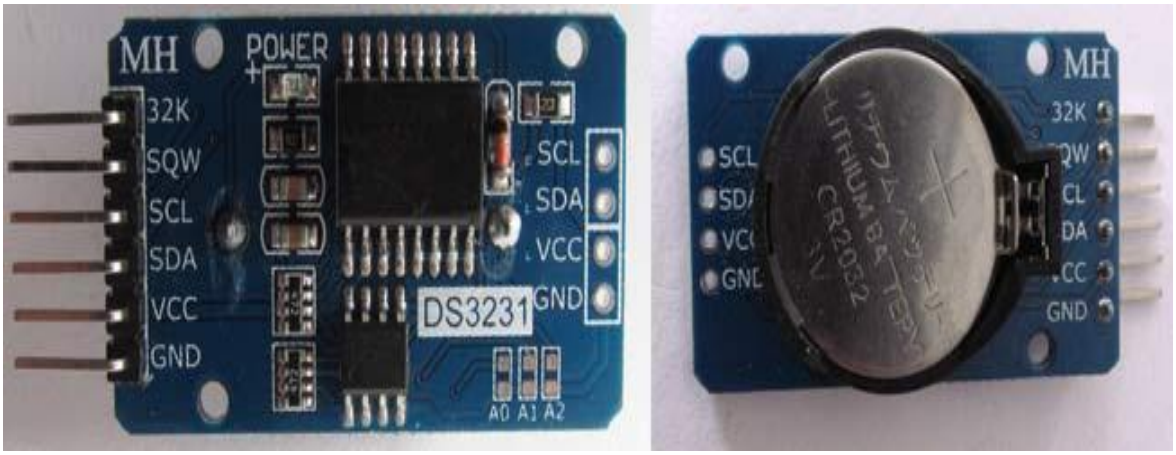
S.No	Requirements	Quantity
1	Microcontroller **PIC16F877A**	1
2	**RTC Module DS1307/DS3231**	1
3		
4	Embedded C / Assembly	

INTRODUCTION

Almost all embedded devices are designed to interact with the real world. They act as a bridge to communicate between the digital world and the real world. To make this process easier and efficient, the digital world would sometimes need to keep track of the time of and date of the real world. This way the digital world will know what time/day it is in the real world and can even distinguish between day or night time. It can also act as a time source to perform certain tasks at a specified time or date. So here we are interfacing a RTC module with PIC Microcontroller and display the time and date on the 16x2 LCD. This project can also be used as Digital Clock**.**

RTC Module:

The most common way for a microcontroller to keep track of the real worlds time or date is by using an **RTC IC**. The term RTC stands for Real Time Clock; this IC keeps track of the real world time and date and would share this information with the microcontroller whenever requested. The RTC IC that we are using here is the most popular and accurate **DS3231**. This IC drifts only by few seconds each year and hence is highly reliable. For the sake of this tutorial we are using the **DS3231 RTC module** which can be easily purchased online or from the local hardware shop. The module comes with a 3V coin cell which powers the RTC module always and hence once the time and date is set it will be updated as long as the coin cell is alive.



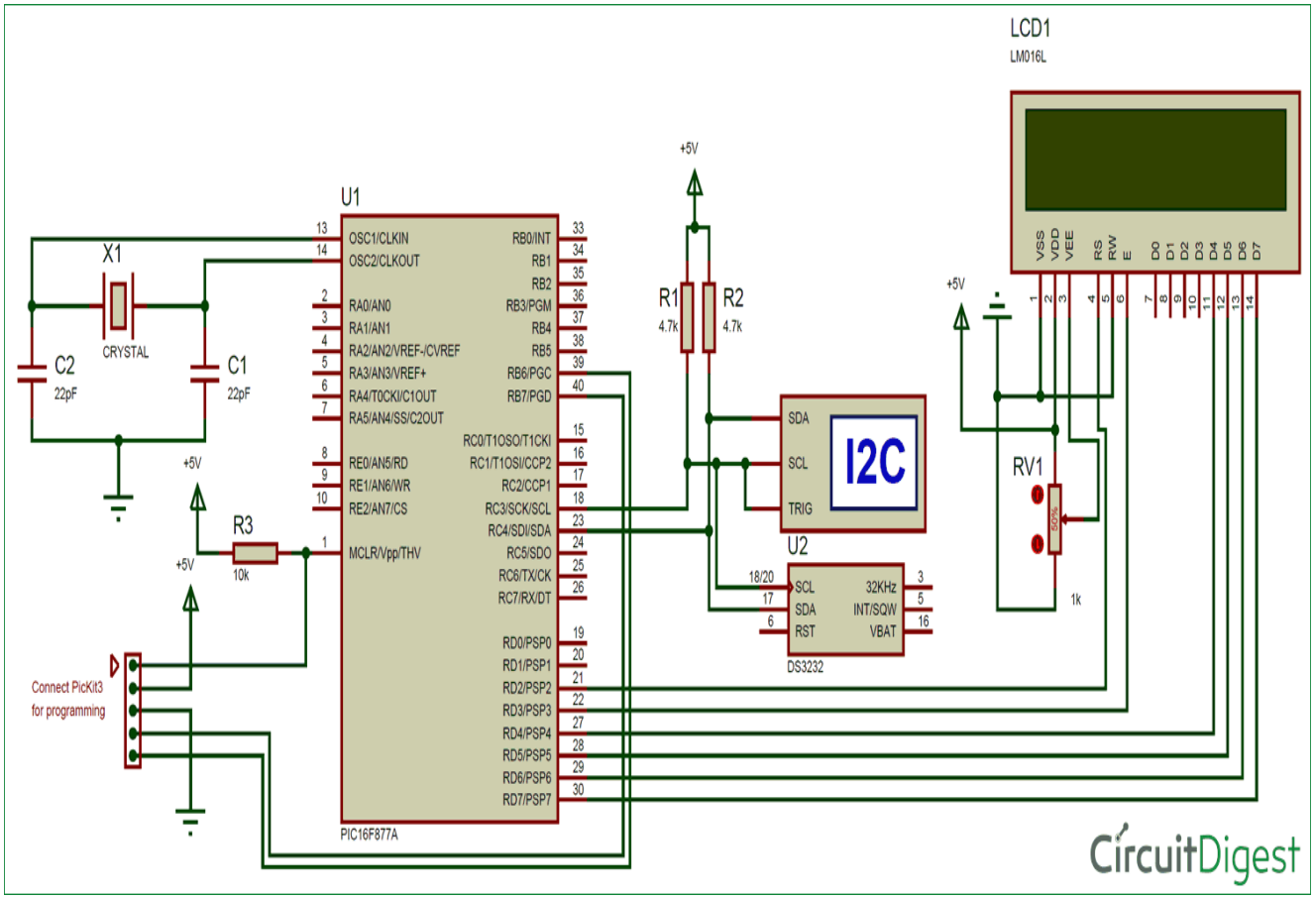
The DS3231 module communicates with the help of I2C protocol, so if you are not aware of what it is and how it is used with PIC read the I2C with PIC tutorial before proceeding. Also in this tutorial we are going to create a header file which can be used to communicate with our RTC module and also test the same on hardware by displaying the time and date on a LCD display so it is also important to learn how to interface LCD display with PIC microcontroller. The header file created in this tutorial for DS3231 can later be used/modified to suit your applications.

Connecting the DS3231 RTC with PIC Microcontroller:

Circuit diagram for **PIC Microcontroller based Digital Clock** is given below. As told earlier the DS3231 works with the help of I2C communication so it will have a Serial Clock (SCL) and a Serial Data (SDA) pin which has to be connected to the I2C pins on our PIC which is the pin 18(SCL) and pin 23 (SDA). A pull up resistor of value 4.7k is used to keep the bus at high state when idle.

An LCD display is also connected to the pins on Port D to display the current date and time. The complete circuit diagram was designed on proteus and is shown below. We are going to use the same to simulate or program later in this tutorial.

CIRCUIT DIAGRAM



The program includes three header files altogether. They are the *lcd.h* file for working with LCD display, the *PIC16F877a_I2C.h* file for working with I2C communication with PIC and finally the *PIC16F877a_DS3231.h* file to work with RTC modules. All the three header files are required for this program and are available in the ZIP file above. Further below I will explain the main program which uses all these header file to read the time and date from the RTC module and display it on the LCD screen. After that I will explain what actually is happening inside the RTC header file. As always begin the program by setting up the configuration bits and setting the clock frequency as 20MHz since that is what we have used in our hardware.

PROGRAM

```
 * File:   RTC_with_PIC_main.c
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code protection off)
#define _XTAL_FREQ 20000000 //We are running on 20MHz crystal
//Define the LCD pins
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
/*Set the current value of date and time below*/
int sec = 00;
int min = 55;
int hour = 10;
int date = 06;
int month = 05;
int year = 18;
/*Time and Date Set*/
#include <xc.h>
#include "lcd.h" //Header for using LCD module
#include "PIC16F877a_I2C.h" // Header for using I2C protocal
#include "PIC16F877a_DS3231.h" //Header for using DS3231 RTC module
int main()
{
   TRISD = 0x00; //Make Port D pins as outptu for LCD interfacing
   Lcd_Start(); // Initialize LCD module
   I2C_Initialize(100); //Initialize I2C Master with 100KHz clock
   Set_Time_Date(); //set time and date on the RTC module
    //Give an intro message on the LCD
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Print_String("  RTC with PIC");
    Lcd_Set_Cursor(2,1);
    Lcd_Print_String(" -Circuit Digest");
```

```
    __delay_ms(1500); //display for 1.5sec
while(1)
  {
   Update_Current_Date_Time(); //Read the current date and time from RTC module
   //Split the into char to display on lcd
    char sec_0 = sec%10;
    char sec_1 = (sec/10);
    char min_0 = min%10;
    char min_1 = min/10;
    char hour_0 = hour%10;
    char hour_1 = hour/10;
    char date_0 = date%10;
    char date_1 = date/10;
    char month_0 = month%10;
    char month_1 = month/10;
    char year_0 = year%10;
    char year_1 = year/10;
     //Display the Time on the LCD screen
      Lcd_Clear();
      Lcd_Set_Cursor(1,1);
      Lcd_Print_String("TIME: ");
      Lcd_Print_Char(hour_1+'0');
      Lcd_Print_Char(hour_0+'0');
      Lcd_Print_Char(':');
      Lcd_Print_Char(min_1+'0');
      Lcd_Print_Char(min_0+'0');
      Lcd_Print_Char(':');
      Lcd_Print_Char(sec_1+'0');
      Lcd_Print_Char(sec_0+'0');
      //Display the Date on the LCD screen
      Lcd_Set_Cursor(2,1);
      Lcd_Print_String("DATE: ");
      Lcd_Print_Char(date_1+'0');
      Lcd_Print_Char(date_0+'0');
      Lcd_Print_Char(':');
      Lcd_Print_Char(month_1+'0');
      Lcd_Print_Char(month_0+'0');
      Lcd_Print_Char(':');
      Lcd_Print_Char(year_1+'0');
      Lcd_Print_Char(year_0+'0');
      __delay_ms(500); //refresh for every 0.5 sec
  }
   return 0;
}
```

RESULT: Successfully interfaced RTC with microcontroller.

VIVA QUESTIONS:

1. What is the function of a Real-Time Clock (RTC)?
2. Why is battery backup required in RTC?
3. Which communication protocol is commonly used with RTC?
4. How does RTC maintain accurate time?
5. What is the role of crystal oscillator in RTC operation?

EXP NO: 8	ESTABLISHING SERIAL DATA TRANSMISSION THROUGH UART.
DATE	

AIM:

To establish serial data transmission through UART.

Apparatus/Software Required

S.No	Requirements	Quantity
1	ARDUINO BOARDS	2
2	Embedded C / Assembly	

INTRODUCTION

Universal Asynchronous Receiver-Transmitter (UART), a serial communication protocol that can be used to send data between an Arduino board and other devices. This is the protocol used when you send data from an Arduino to your computer, using the classic Serial. Print () method.

UART is one of the most used device-to-device (serial) communication protocols. It's the protocol used by Arduino boards to communicate with the computer. It allows an asynchronous serial communication in which the data format and transmission speed are configurable. It's among the earliest serial protocols and even though it has in many places been replaced by SPI and I2C it's still widely used for lower-speed and lower-throughput applications because it is very simple, low-cost and easy to implement.

Communication via UART is enabled by the Serial class, which has a number of methods available, including reading & writing data.

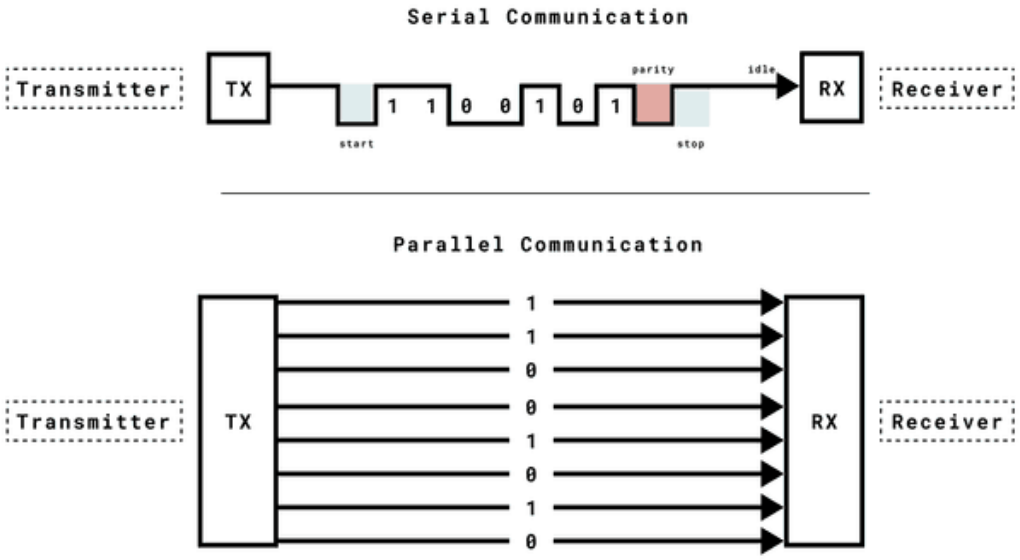
Arduino UART Pins

The default TX/RX pins on an Arduino board are the D0(RX) and D1(TX) pins. Some boards have additional serial ports, see table below:

Form Factor	RX	TX	RX1	TX1	RX2	TX2	RX3	TX3
MKR	D0	D1						
UNO	D0	D1						
Nano	D0	D1						
Mega	D0	D1	D19	D18	D17	D16	D15	D14

How UART Works

UART operates by transmitting data as a series of bits, including a start bit, data bits, an optional parity bit, and stop bit(s). Unlike parallel communication, where multiple bits are transmitted simultaneously, UART sends data serially, one bit at a time. As the name reveals the protocol operates asynchronous which means that it doesn't rely on a shared clock signal. Instead, it uses predefined baud rates to determine the timing of data bits.



Serial / Parallel Communication

As seen in the image above when using parallel communication an 8-bit message would require eight cables while serial communication only requires one cable for sending messages and one for receiving.

Transmit / Receive Messages

This example allows you to send messages (strings) back and forth between devices. Upload the following sketch to both devices:

PROGRAM

```
String sendMessage;
String receivedMessage;

void setup() {
  Serial.begin(9600);    // Initialize the Serial monitor for debugging
  Serial1.begin(9600);   // Initialize Serial1 for sending data
}

void loop() {
  while (Serial1.available() > 0) {
    char receivedChar = Serial1.read();
    if (receivedChar == '\n') {
      Serial.println(receivedMessage);  // Print the received message in the Serial monitor
      receivedMessage = "";  // Reset the received message
    } else {
      receivedMessage += receivedChar;  // Append characters to the received message
    }
  }

  if (Serial.available() > 0) {
    char inputChar = Serial.read();
    if (inputChar == '\n') {
      Serial1.println(sendMessage);  // Send the message through Serial1 with a newline character
      sendMessage = "";  // Reset the message
    } else {
      sendMessage += inputChar;  // Append characters to the message
    }
  }
}
```

RESULT: Successfully established serial data communication through UART.

VIVA QUESTIONS:

1. What is UART, and how does it work?
2. Differentiate between synchronous and asynchronous communication.
3. What is baud rate, and why must it be same for transmitter and receiver?
4. Explain start bit, stop bit, and parity bit.
5. What causes framing error in UART communication?

XP NO: 9	ESTABLISHING SERIAL DATA COMMUNICATION USING I2C AND SPI PROTOCOLS.
DATE	

AIM:

To establish serial data transmission through I2C AND SPI PROTOCOLS.

Apparatus/Software Required

S.No	Requirements	Quantity
1	ARDUINO BOARDS	2
2	Embedded C / Assembly	

INTRODUCTION

I2C means Inter-integrated circuit communication protocol. It is a 2-wire serial communication protocol for short range data transfer applications. It is a very popular communication protocol used in embedded projects to interface I2C based sensors, digit displays, and communication modules. Devices that want to communicate with each connects through an I2C bus. I2C bus supports multiple slave devices and multiple master devices. Many Sensors use this serial communication protocol to transfer their data to microcontrollers or through this protocol different slave circuits are able to communication with master circuits. It is only applicable for short distance data transmission.

I2C Pins

The distinguishing feature of I2C over SPI is that it uses only two wires to carry out communication. One wire is SCL (serial clock line) which synchronizes the transmission of data between devices and the other wire is SDA (serial data) which carries the actual data to be transmitted. These lines are open-drain lines which means these need to be connected to pull up resistors if devices are active low. Each slave device which is connected with the bus will have a unique 8-bit address. The communication between specific devices using two wires is accomplished by the fact that each device has its own unique device ID or address and using this address; master can choose any specific device to communicate.

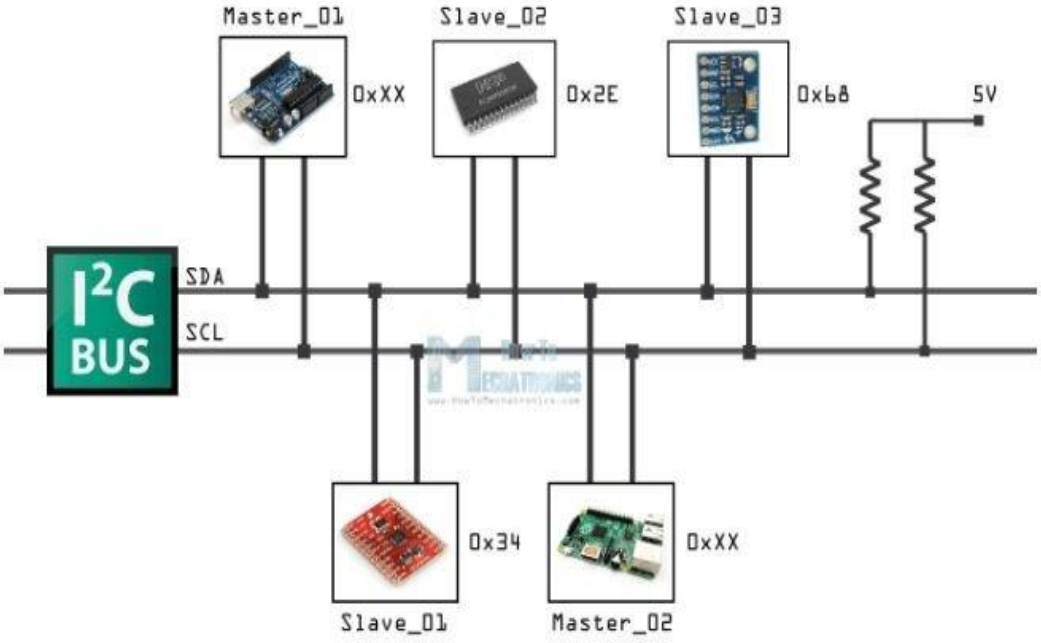
Slave Devices

Each slave devices has unique address that is utilized to recognize the device on the bus. In other words, slave address helps the master device to send information to a specific slave device on the bus.

Master devices

Master devices can send and get information. Slave react to whatever an master sends. When sending information on the bus, just a single device can send information at a time.

In short, we only need two wires for transferring data or communicate with different numbers of devices. As we all know that Arduino has limited pins, I2C allows to connect with multiple devices at a same time. only shortcoming is that you cannot use this protocol for long distance data transferring.



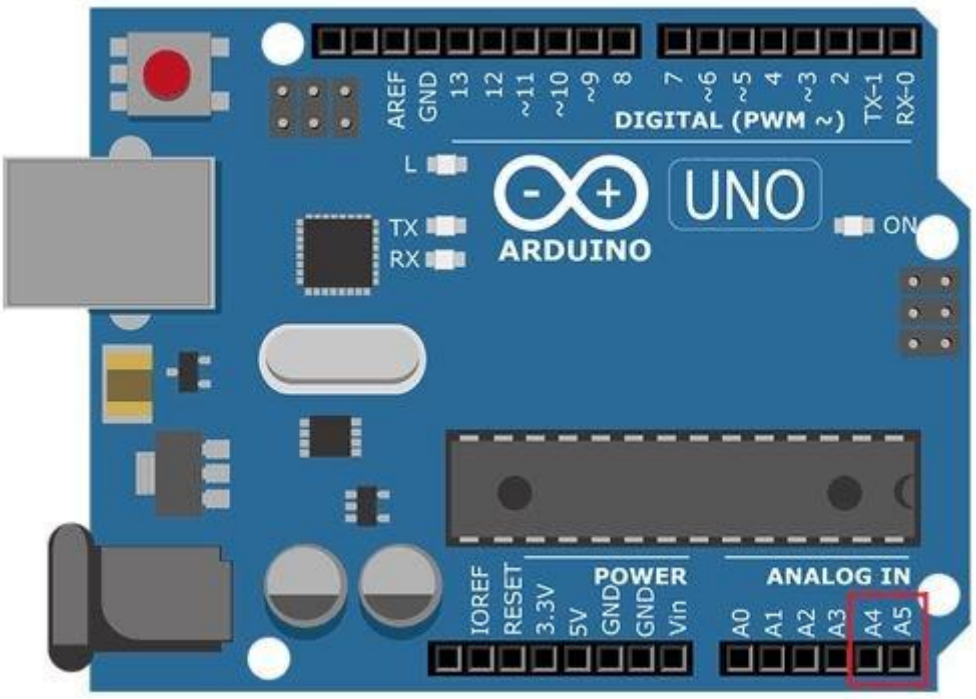
Data transfer bit by bit serially along a wire (the SDA line). Like SPI, I2C is concurrent, the output of bits is synchronized to the testing of bits by a clock signal shared between the master and the slave.

Arduino I2C communication Pins

For I2C communication, different boards of Arduino have different pins dedicated as SDA and SCL pins. Below list shows these pin numbers in different boards.

1. In Arduino UNO, Pin A4 = SDA and Pin A5 = SCL
2. For Arduino Mega2560, Pin 20 = SDA and Pin 21 = SCL
3. In Arduino Leonardo, Pin 2 = SDA and Pin 3 = SCL
4. For Arduino Due, Pin 20 = SDA and Pin 21 = SCL, SDA1, SCL1

Below figure shows SDA and SCL pins in Arduino UNO which will be used



Complete Master Arduino Code

```
#include <Wire.h>
int LED=13;
int x = 0;

void setup()
{
 Wire.begin();
 Serial.begin(9600);
 pinMode(LED,OUTPUT);
}

void loop()
{
 Wire.beginTransmission(9);
 Wire.write(x);
 Wire.endTransmission();

 x++;
 if (x > 6)
 {
  x = 0;
 }
 delay(200);
}
```

Complete Slave Device Code

```
#include <Wire.h>
int LED = 13;
int x = 0;

void setup()
{
  pinMode (LED, OUTPUT);
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}

void receiveEvent(int bytes)
{
  x = Wire.read();
}

void loop()
{
  if (x <= 3)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
}
```

RESULT: Successfully established serial data communication through I2C.

VIVA QUESTIONS:

1. What are the main differences between I²C and SPI protocols?
2. Explain master and slave concept in I²C communication.
3. What is the function of SCL and SDA lines?
4. Why is SPI faster than I²C?
5. What is chip select (CS), and why is it required in SPI?

EXP NO: 10	WAP(Wireless Application Protocol) FOR LED BLINK
DATE	

AIM:

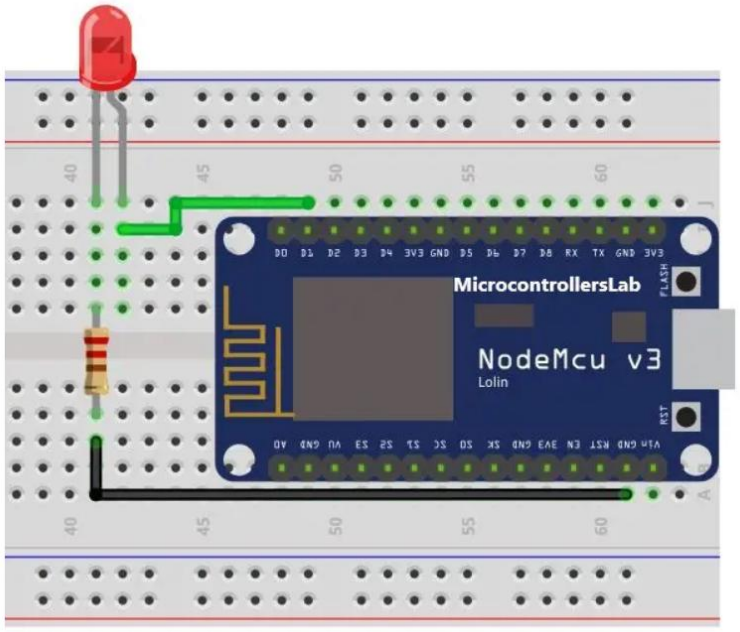
To control (blink) an LED wirelessly using Node MCU by implementing a simple web server (WAP-based interface) that allows the user to turn the LED ON or OFF from a browser on the same Wi-Fi network.

Materials Required

- Node MCU (ESP8266)
- LED (any color)
- 220Ω resistor
- Breadboard and jumper wires
- Micro-USB cable
- Computer or smartphone with Wi-Fi capability
- Arduino IDE

Circuit Diagram

- **LED Anode (+):** Connect to NodeMCU D4 (GPIO2) via 220Ω resistor.
- **LED Cathode (–):** Connect to NodeMCU GND.



Procedure

1. **Hardware Setup:**
   - Assemble the circuit as described above.
   - Connect NodeMCU to your computer via USB.
2. **Software Setup:**
   - Open Arduino IDE.
   - Install the ESP8266 board package (if not already installed).
   - Select NodeMCU 1.0 (ESP-12E Module) and the correct COM port.
3. **Program NodeMCU as a WAP Web Server:**
   - The NodeMCU will create a simple web page accessible via Wi-Fi. Users can turn the LED ON or OFF by clicking buttons on the web page.

WAP-Based Web Server Code

```
#include <ESP8266WiFi.h>
// Replace with your network credentials
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
WiFiServer server(80);
#define LED_PIN D4



void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("Connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop() {
  WiFiClient client = server.available();
  if (!client) return;

  while (!client.available()) {
    delay(1);
  }

  String request = client.readStringUntil('\r');
  client.flush();

  // Parse request and control LED
  if (request.indexOf("/LED=ON") != -1) {
    digitalWrite(LED_PIN, HIGH);
  }
  if (request.indexOf("/LED=OFF") != -1) {
    digitalWrite(LED_PIN, LOW);
  }
  // HTML response
  String html = "<!DOCTYPE html><html><head><title>LED Control</title></head><body>";
  html += "<h1>ESP8266 LED Control (WAP)</h1>";
  html += "<p><a href=\"/LED=ON\"><button>ON</button></a></p>";
  html += "<p><a href=\"/LED=OFF\"><button>OFF</button></a></p>";
  html += "</body></html>";

  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
```

```
  client.println(html);
  delay(1);
}
```

Testing and Observations

1. Upload the code to NodeMCU.
2. Open Serial Monitor to find the IP address assigned to NodeMCU (e.g., 192.168.1.100).
3. On any device connected to the same Wi-Fi, open a browser and enter the IP address.
4. The web page will display ON and OFF buttons. Click to control the LED wirelessly.

RESULT: The LED can be turned ON or OFF from any device on the same Wi-Fi network using a WAP-based web interface hosted by Node MCU.

VIVA QUESTIONS:

1. What is IoT, and how does it differ from traditional embedded systems?
2. Compare Zigbee, GSM, and Bluetooth technologies.
3. What are the advantages of wireless communication?
4. How is data transmitted wirelessly between devices?
5. What are common applications of IoT-based embedded systems?

EXP NO: 11 DATE	**Design and implementation of ON/OFF control strategy.**

AIM:

To Design and implement an Arduino program to control the state of an LED using a push button.

Apparatus/Software Required

S.No	Requirements	Quantity
1	ARDUINO BOARDS	2
2	Embedded C / Assembly	

PROGRAM:

```
// constants won't change. They're used here to set pin numbers: const int buttonPin = 9;
// the number of the pushbutton pin const int relayPin = 10; // the number of the LED pin
// variables will change:
int button State = 0;
// variable for reading the pushbutton status
void setup()
{
// initialize the LED pin as an output: pin Mode (relay Pin, OUTPUT);
// initialize the pushbutton pin as an input: pin Mode(buttonPin, INPUT);
}
void loop()
{
// read the state of the pushbutton value: button State = digital Read(buttonPin);
// check if the pushbutton is pressed. If it is, the button State is HIGH: if (button State == HIGH)
{
// turn LED on: digital Write (relay Pin, HIGH);
}
else
{
// turn LED off: digital Write (relay Pin, LOW);
}
}
```

RESULT: Thus an Arduino program to control the state of an LED using a push button was implemented.

VIVA QUESTIONS:

1. What is ON/OFF control?
2. How does ON/OFF control differ from PID control?
3. What is hysteresis, and why is it used?
4. Mention one real-time application of ON/OFF control.
5. What are the limitations of ON/OFF control systems?

EXP NO: 12	IMPLEMENTATION OF BASIC EXPERIMENTS USING RASPBERRY PI / ARDUINO.
DATE	

AIM:

To Design and Implement an basic Experiments Using Raspberry Pi / Arduino.

APPARATUS REQUIRED:

Arduino microcontroller kit

PROGRAM:

```
const int ledPin = 10;
const int ldrPin = A3;
void setup()
{
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
pinMode(ldrPin, INPUT);
}
void loop()
{
 int ldrStatus = analogRead(ldrPin);
 if (ldrStatus <= 350)
{
digitalWrite(ledPin, HIGH);
Serial.print("Its DARK, Turn on the LED : ");
 Serial.println(ldrStatus);
 }
Else
 {
digitalWrite(ledPin, LOW);
Serial.print("Its BRIGHT, Turn off the LED : ");
 Serial.println(ldrStatus);
 }
}
```

RESULT: Thus an Arduino program to control the state of an LED using LDR was implemented.

VIVA QUESTIONS:

1. What is the difference between Arduino and Raspberry Pi?
2. Is Raspberry Pi a microcontroller or a microprocessor? Why?
3. What programming languages are used in Arduino and Raspberry Pi?
4. How are GPIO pins accessed in Raspberry Pi?
5. What are typical applications of Arduino and Raspberry Pi?

EXP NO: 13	DESIGN OF IOT SYSTEM – GAS LEAKAGE DETECTION SYSTEM
DATE	

AIM:

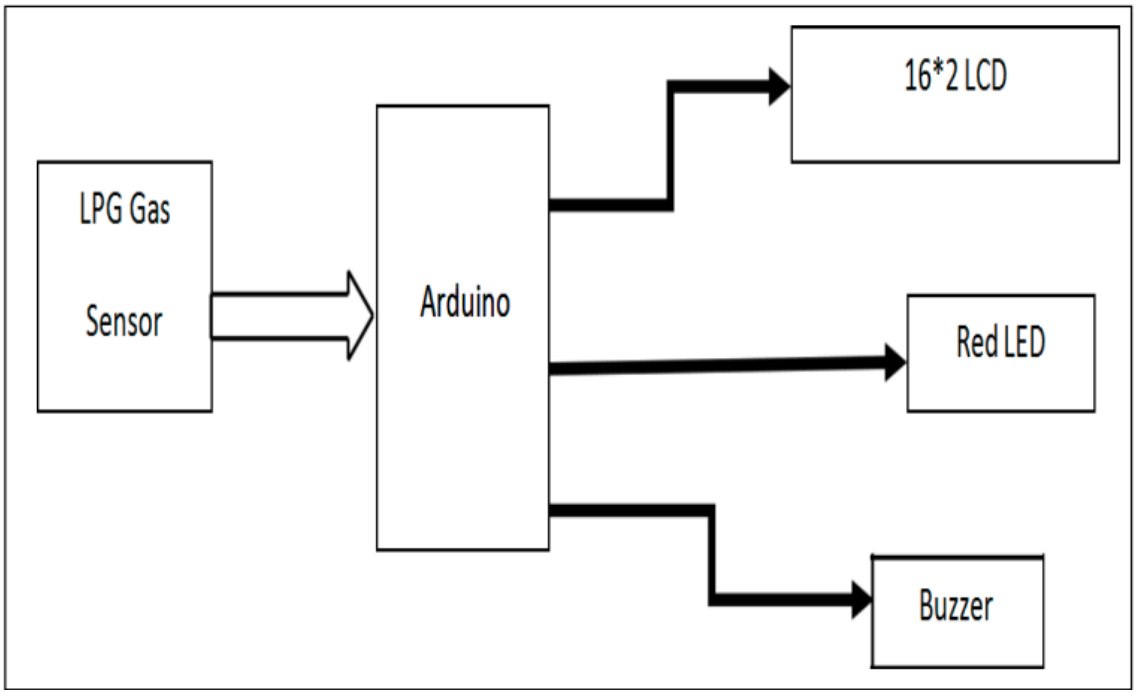
To design and implement an IoT-enabled gas leakage detection system and generates alarm.

THEORY :

Gas leakage is a serious problem and nowadays it is observed in many places like residences, industries, and vehicles like Compressed Natural Gas (CNG), buses, cars, etc. It is noticed that due to gas leakage, dangerous accidents occur. The Liquefied petroleum gas (LPG), or propane, is a flammable mixture of hydrocarbon gases used as fuel in many applications like homes, hostels, industries, automobiles, and vehicles because of its desirable properties which include high calorific value, less smoke, less soot, and meager harm to the environment. Liquid petroleum gas (LPG) is highly inflammable and can burn even at some distance from the source of leakage.

A gas leakage detector becomes vital and helps to protect people from the dangers of gas leakage There are different gas detection techniques used . This system is a low-cost advanced sensor-based gas leakage detector, alert and control system .

BLOCK DIAGRAM:



Materials Required

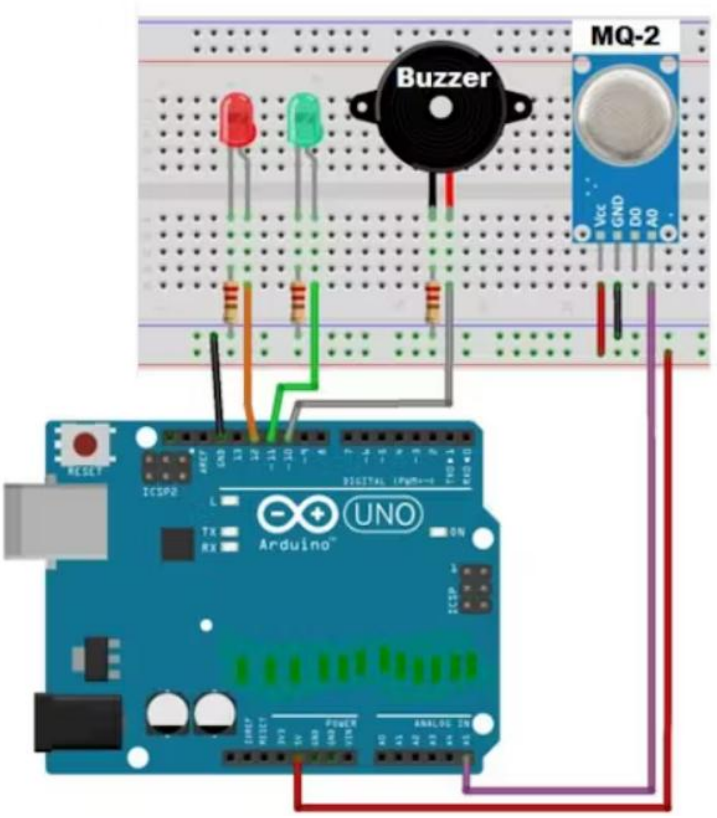
ARDUINO

MQ-6 GAS SENSOR

16*2 LCD

BUZZER

CIRCUIT DIAGRAM



Program :

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(5,6,8,9,10,11);
int redled = 2;
int greenled = 3;
int buzzer = 4;
int sensor = A0;
int sensorThresh = 400;
```

```
void setup()
{
pinMode(redled, OUTPUT);
pinMode(greenled,OUTPUT);
pinMode(buzzer,OUTPUT);
pinMode(sensor,INPUT);
Serial.begin(9600);
lcd.begin(16,2);
}
void loop()
{
int analogValue = analogRead(sensor);
Serial.print(analogValue);
if(analogValue>sensorThresh)
{
digitalWrite(redled,HIGH);
digitalWrite(greenled,LOW);
tone(buzzer,1000,10000);
lcd.clear();
lcd.setCursor(0,1);
lcd.print("ALERT");
delay(1000);
lcd.clear();
lcd.setCursor(0,1);
lcd.print("EVACUATE");
delay(1000);
}
else
{
digitalWrite(greenled,HIGH);
digitalWrite(redled,LOW);
noTone(buzzer);
```

```
lcd.clear();
lcd.setCursor(0,0);



lcd.print("SAFE");
delay(1000);
lcd.clear();
lcd.setCursor(0,1);
lcd.print("ALL CLEAR");
delay(1000);
}
}
```

RESULT: Thus, an IoT-enabled gas leakage detection system that utilizes LPG gas detection to raise alarm is designed and implemented.

VIVA QUESTIONS:

1. What is the principle of operation of a gas leakage detection system?
2. Which gas sensor is used in this experiment, and how does it detect gas concentration?
3. Why is IoT used in gas leakage detection instead of a standalone embedded system?
4. What actions are taken by the system when gas concentration exceeds the threshold level?
5. How is sensor data transmitted to the cloud or user in the IoT gas leakage detection system?