Practical Manual 2017-2018

On

Embedded Systems

(Course Code-USIT4P5)

For

S.Y.B.Sc. I.T.

(Semester IV)

Prepared By
Mrs.Archana Bhide
R.J.College,Ghatkopar.

Index

Introduction to embedded system Practical

Embedded System can be defined as a system that has embedded software and computer hardware which makes it a system dedicated for application(s) or a specific part of an application.

Constituents of embedded system hardware –

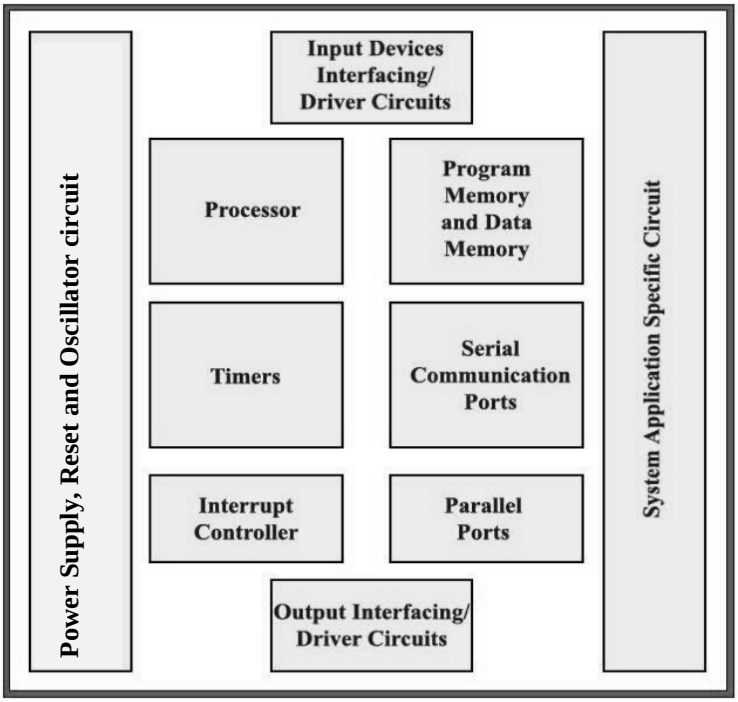


Fig 1 Components of embedded system hardware

Processor : Processor has program flow control unit (CU) and execution unit (EU). The CU includes fetch unit for fetching instruction. EU include has circuits that implement the instructions related to data transfer operations and data conversion from one to another. EU also includes ALU and circuits that execute instruction for a program which controls different tasks.

System Timers: A timer circuit is configured as a system clock, which generates system interrupts periodically which performs required operation, or it can be configured as Real Time Clock (RTC) that generates system interrupts for schedulers, real time programs. RTC can also be used to obtain software controlled delays. It also functions as driver for software timers.

Interrupt handler / controller: Interrupt handling mechanism handles multiple interrupts generated from various processes simultaneously. There can be number of interrupt sources or group of sources. The service to these sources can have priorities according to the system priorities. Certain sources of interrupts are non maskable and cannot be disabled.

There is a separate program and data memory for the controller. Internal and external RAM for registers, temporary data and stack, Internal and external ROM for program memory, internal flash memory, memory cards, memory buffers of ports, cache memories.
Communication ports: several communication ports serial as well as parallel to communicate with various peripheral devices like keyboards touch screens, sensors, transducers etc. (input) or printers, LED / LCD display, network etc. (output).

Power supply, Reset and Oscillator Circuit: Many embedded systems have their own power supply; however fee systems use PC power by connecting to computer. Power supply has range of voltages in one of following range 5 ±0.25V, 3.3 ±0.3V, 2 ±0.2V and 1.5 ±0.2V
Reset circuit helps the processor to begin processing of instructions from a starting address. This address is set by default in processor Program Counter (PC) on a power up. A program to be executed after reset is - a system program which executes from beginning, or a boot program, or a system initialization program. Reset circuit activates for a fixed period and then deactivates.

Software components of embedded system -

An embedded system processor executes software that is specific to a given application system. Software components include editor, assembler, linker, and loader.
The steps involved in executing an embedded assembly program are as followed –

1. An assembler translates the assembly software into machine codes.
2. The code is linked with other required codes. As there are several codes required to linked together for the final binary file. Linker produces binary file. This binary file can directly be run on a computer and is called as an executable file.
3. Loader program performs task of reallocating the codes after finding the physical memory address available at a given instant. The loader is part of operating system and places the codes into its memory after reading the 'exe' file.
4. These codes are to be then located as ROM image.

5. Lastly, this ROM image file is either burned into PROM or flash memory using a device programmer tool, or it is masked for the PROM from the ROM image file.

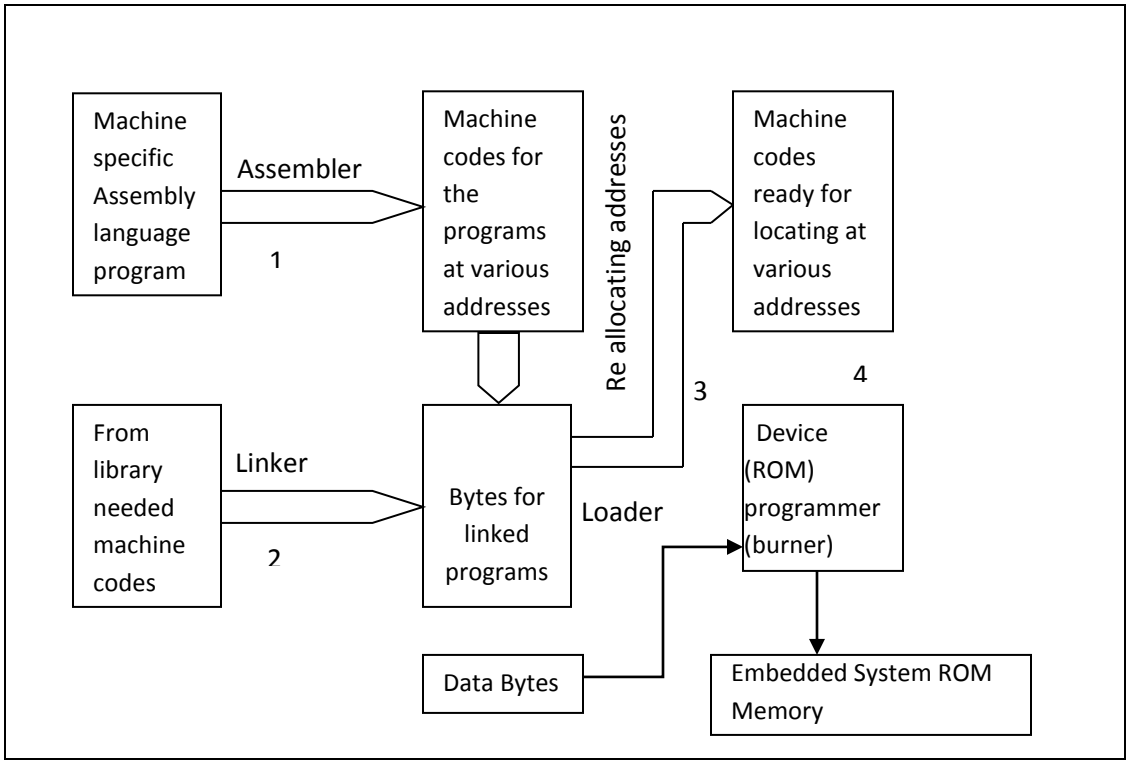


Fig 2 Process of converting assembly language program into the machine codes and obtaining the ROM image

Software is developed in higher level languages to save efforts instead of using assembly language. Following are different program layers in embedded C

Processor commands
Main Function
Interrupt service Routines
Tasks 1 ….. N
Kernel and scheduler
Standard Library functions

Fig 3 Program Layers in embedded C

C Has a feature that adds assembly instructions when using certain processor specific features and coding for a specific section. Fig 3 shows the layers in the process.

A C program is converted to a ROM image in the way shown in Fig 4. A compiler generates object code. It assembles the code according to processor instruction set and other specification. C compiler for embedded system then optimizes the code before uses code optimizer to optimize the code before linking. After compilation linker links the object code with other needed codes. This puts together all the functions used including the codes required for device and driver management.

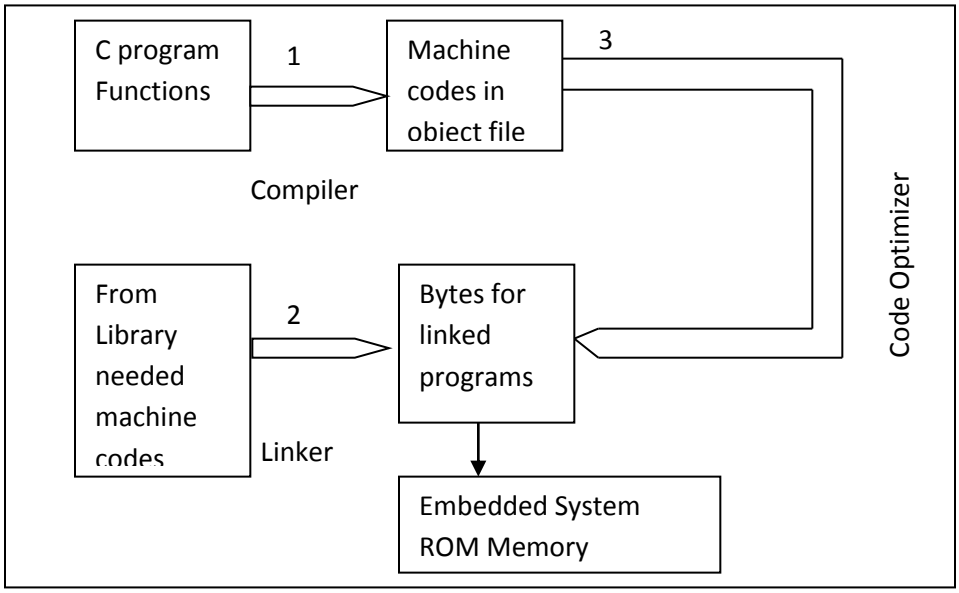


Fig 4 process of converting C program into ROM image

Introduction to 8051 Microcontroller

8051 architecture

8051 architecture consists of 8 bit CPU with processing capability, oscillator driver unit, 4k on chip program memory, 128 bytes internal data memory, 128 bytes of SFR, 32 general purpose I/O lines organized into 4 8-bit bidirectional ports, two 16 bit timer units and a full duplex programmable UART (Universal Asynchronous Receiver Transmitter) for serial data transmission.

Fig below shows the internal architecture of 8051 microcontroller

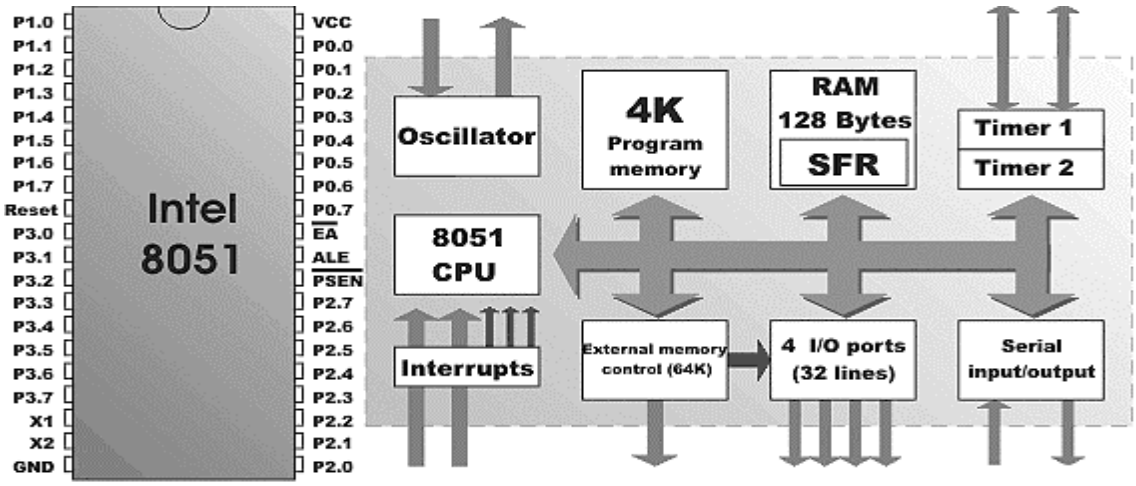


Fig 5 (A) Pin configuration and (B) Internal architecture of 8051 microcontroller

8051 memory organization

8051 is build around Harvard architecture. Program and data memory is separated logically. Separate address spaces are assigned for program and data memory. 8051's address bus is 16 bit wide and it can address up to 64KB memory.

Program or code memory is lowest 4K bytes of program memory as on chip memory (built in chip memory). Internal and external program memory is switched between using changing logic levels of pin External Access ($\overline{EA}$). 1 on this pin means instructions are to be executed from the program memory up to 4K and logic 0 on this pin means that chip is in external program execution mode. The control signal for external program memory execution is $\overline{PSEN}$ (program Strobe Enable). This is not activated for internal program memory.

Basic 8051 architecture supports 128 bytes of internal data memory and 128 bytes of special function register memory. SFR is not available for the user for general data memory operation. The address range of internal user data memory is 00 H to 7F H. SFR reside at memory area 80 H to FF H. control signals used for external data memory address are $\overline{RD}$ and $\overline{WR}$. 16 bit register holding the address to be accessed is Data Pointer (DPTR). DPTR is made up of two 8 bit registers DPH and DPL. This register is accessible to the user and its contents can be modified. In external data memory operations port 0 emits the contents of DPL and port2 emits the contents of DPH.

Internal data memory addresses are always one byte long. It can accommodate up to 256 bytes of internal data memory (Ranging from 0 to 255). 8051 uses Direct addressing of data memory greater than 7F H

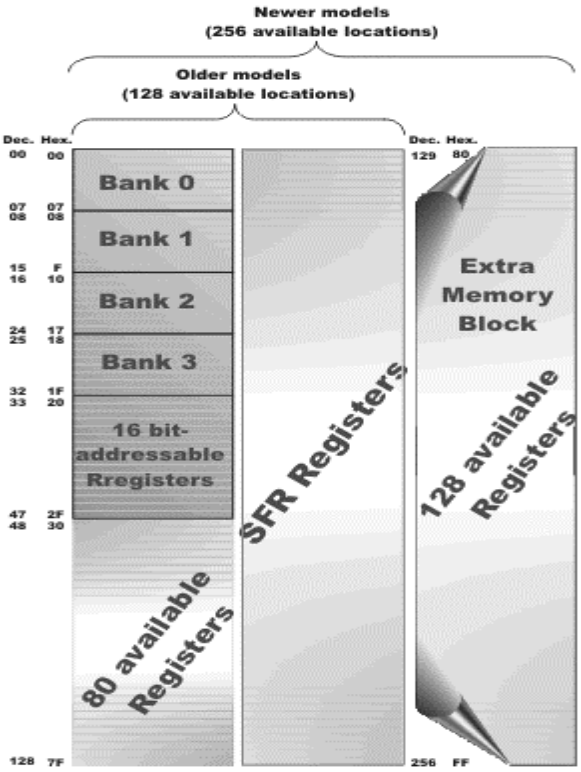


Fig 6 Memory organization of 8051

8051 registers:

8051 registers can be broadly classified as CPU registers and scratchpad registers.

CPU registers:

Accumulator

B register

Program Status word

Data Pointer
Program Counter
Stack Pointer
Scratchpad Registers (R0 – R7)
8051 Ports:
8051 Timer units:

TIMER 0 AND TIMER 1 OPERATION

Timer 0 and Timer 1

The "Timer" or "Counter" function is selected by control bits C/T in the Special Function Register TMOD. These two Timer/Counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both Timers/Counters.
Mode 3 is different. The four operating modes are described in the following text.

TMOD : Timer/Counter Mode Control Register (Not Bit Addressable)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

Note 1 :

M1	M0		OPERATING MODE
0	0	0	13–bit Timer
0	1	1	16–bit Timer/Counter
1	0	2	8–bit Auto–Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8–bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8–bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

Figure 7 Timer/Counter 0/1 Mode Control (TMOD) Register

Mode 0

Putting either Timer into Mode 0 makes it an 8-bit Counter with a divide-by-32 pre-scalar. In this mode, the Timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements). TR1 is a control bit in the Special Function Register TCON (Figure 8).

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers. Mode 0 operation is the same for the Timer 0 as for Timer 1.
Substitute TR0, TF0, and INT0 for the corresponding Timer 1 signals in Figure 2. There are two different GATE bits, one for Timer1 (TMOD.7) and one for Timer 0 (TMOD.3).

TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

Figure 8. Timer/Counter 0/1 Control (TCON) Register

Mode 1
Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

Mode 2
Mode 2 configures the Timer register as an 8-bit Counter (TL1) with automatic reload.

Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is
preset by software. The reload leaves TH1 unchanged. Mode 2 operation is the same for Timer/Counter 0.

Mode 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0. Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 7. TL0 uses the Timer 0 control bits: C/T, GATE, TR0, and TF0, as well as the INT0 pin. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer on the counter. With Timer 0 in Mode 3, an 80C51 can look like it has three Timer/Counters. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt

Introduction to TK Base Trainer Kit

Trainer kits used for performing Embedded Systems practical with 8051 microcontroller are TKBase kits. These kits are made up of –

- CPU specific daughter board.
- Base board section
- User Interface

Base Board Section includes

- LEDs
- Four digit 7 segment display
- Hex Keypad
- RS233C serial port

LEDs

There are 8 general purpose LEDs and 3 LEDs to indicate power. The LEDs can be connected to any port pins of 8051RD2 processor. As there are 8 LEDs, range of data values from 0000 0000 – 1111 1111 can be displayed.

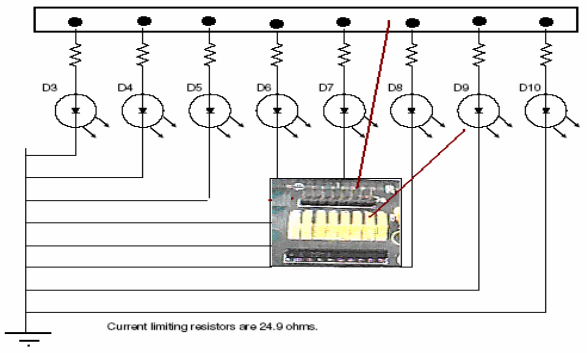


Fig 9 LED interface

7 Segment Display

Four character 7 segment display is controlled by 8051RD2. Each digit shares 8 common control signals (a …g) to light individual LED segments. Each individual character has a separate anode control input pin (1 ... 4). Pin number for each 8051RD2 pin connected to LED display appears in parentheses. To light an individual signal, drive the individual segment control signal high and anode control signal for the individual character low. To enable a specific display the 8051 port pins P1.0 thru P1.3 are utilized. The LED control signals are time multiplexed to display data on four display characters shown below.

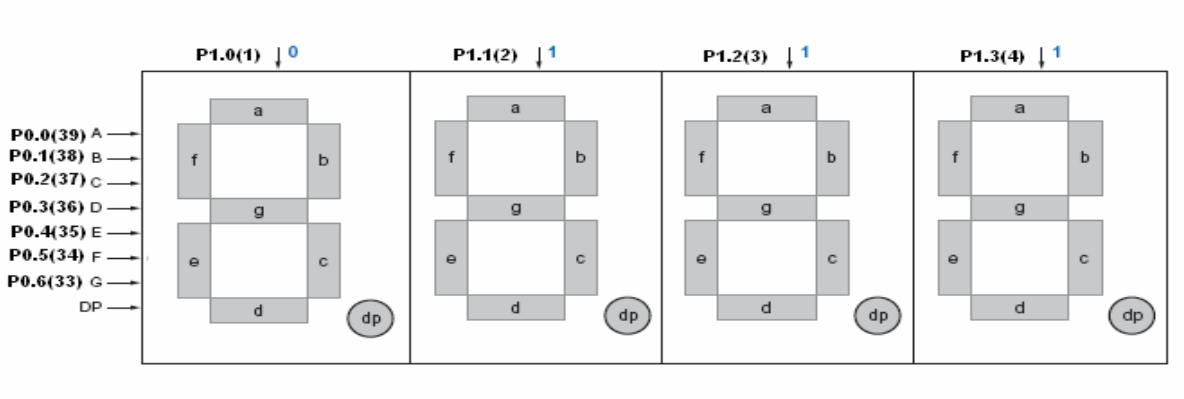


Fig 10 Four 7 segment display characters.

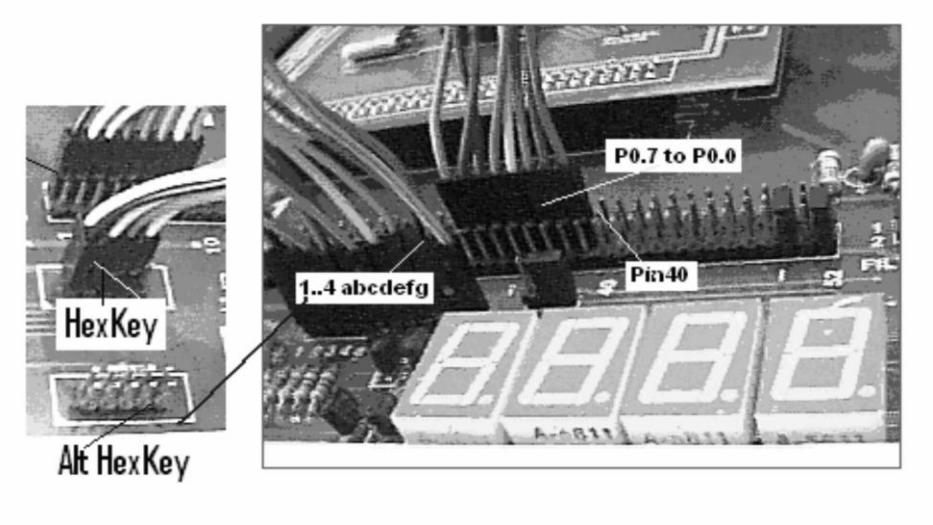


Fig 11 using 7 segment to display output

Hex Keypad

8051RD2 signal pins from pin 1 through 40 are made available to the users from IO header CN5 from pin marked 1 through 40. The first 8 pins of CN5 are mapped to first 8 pins of 8051RD2. 16 hex keypad is interfaced to 8051RD2 as matrix of 4 scan signals and 4 return signals. Pressing a key switch push button generates a logic Low on the associated RD2 pin.

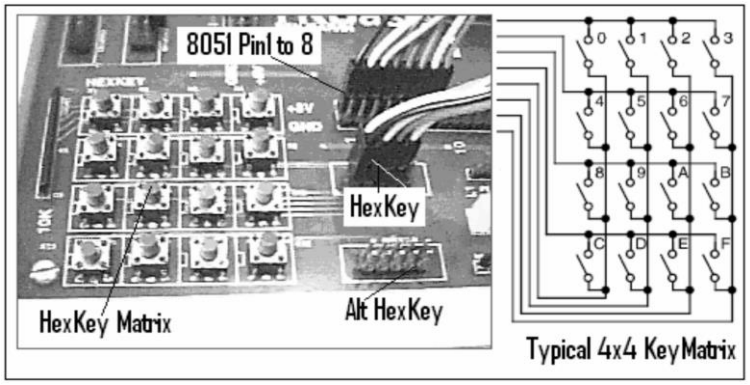


Fig 12 Hex Keypad

Thus, 4 X 4 keyboard matrix, giving a total of 16 keys with each key representing a single hexadecimal value as shown in Fig 12. The 8051RD2 microcontroller program scans keyboard matrix to detect which key was pressed. After detection, corresponding hex code is displayed on the 4 character 7 segment display. P1.0 to P1.3 are assigned as output (key scan) and P1.4 to P1.7 are assigned as inputs (key returns). User program detects which key has been pressed while a lookup table defines the value of each key.

RS232C Serial Port

RS232 signals are used to implement the interface (RD and TD signals). RS232 transmit and receive signals appear on the male DB9 connector labeled SER-D9. The user provides RS232 UART code. A standard serial cable can be used to connect the 8051RD2 with the PC. A standard MAX232 RS232 is used to drive RX and TX signals. Fig 13 shows the connection between the Serial port and DB9connector, including the MAX232 RS-232 voltage converter. A jumper is placed between RX TX pin for an easy loopback test.

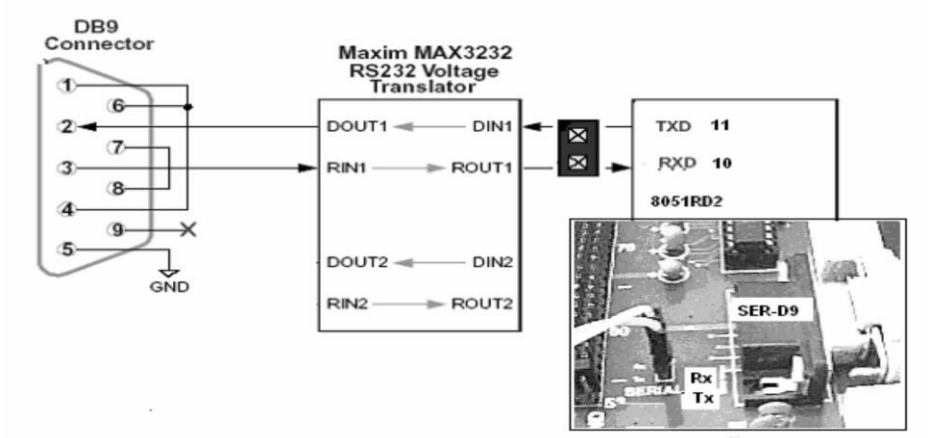


Fig 13 Connection using the DB9 connector.

User interface section with following features

- Hex Keypad
- 8 LED indicator
- Four multiplexed 7 segment displays
- LCD 16 characters X 2 lines
- Relay
- Serial port with MAX 232 IC
- USB connector
- Pull up 5.5 V and 3.3 V
- 12C NVRAM
- 12C RTC PCF8583
- 12C ADC/ DAC PCF8591

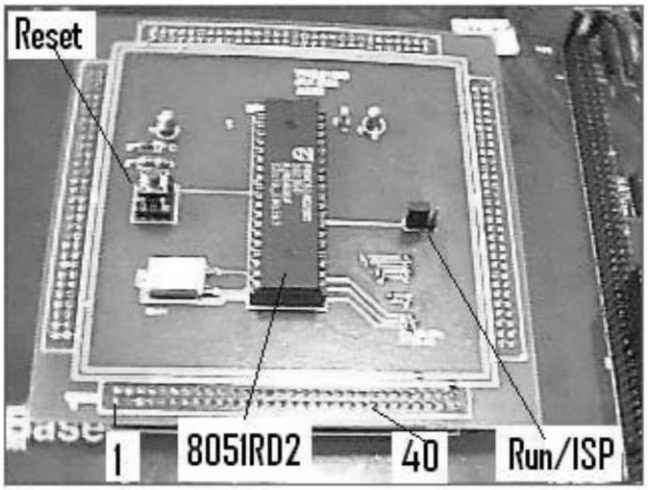


Fig 14 Daughter board of universal TKB51RD

Introduction to Development Environment

Software Used

Execution of Embedded C program is done in two steps. First step is performed at the host machine. Here, a C program is compiled and built to generate a hex file, which is assembly code generated by a tool. Several compilers are available for compiling and linking program. Following are few examples –

- Crossware Tools for Embedded Development
- µVision 5

1. Crossware Tool

Steps to be followed –

Generation of hex file in crossware

1. Boot the Crossware Embedded Development studio.
2. IDE of the software looks as given in fig below –

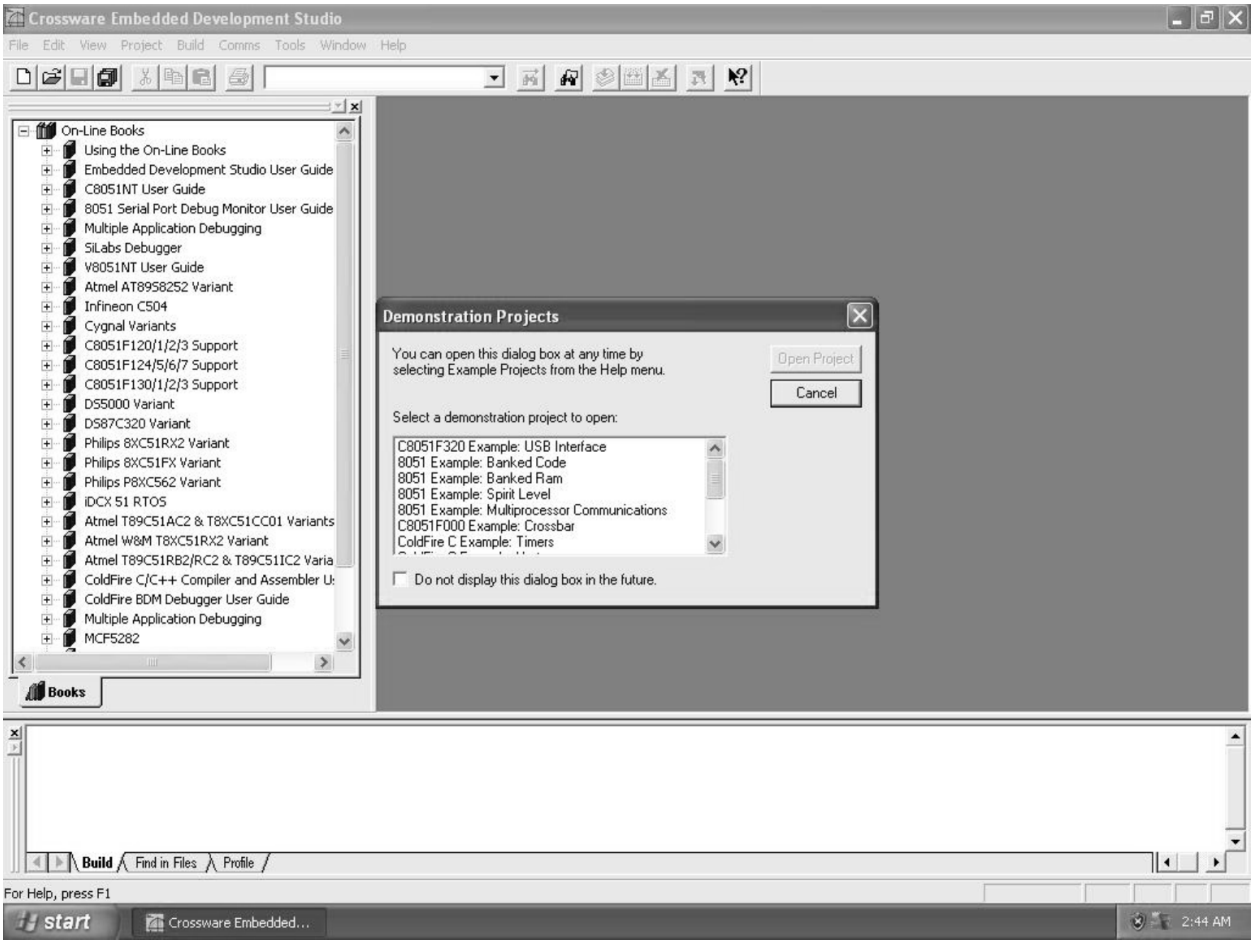


Fig 15 Crossware C compiler IDE

3. Cancel the demonstration project window and start with a new program. New opens the following dialog box.

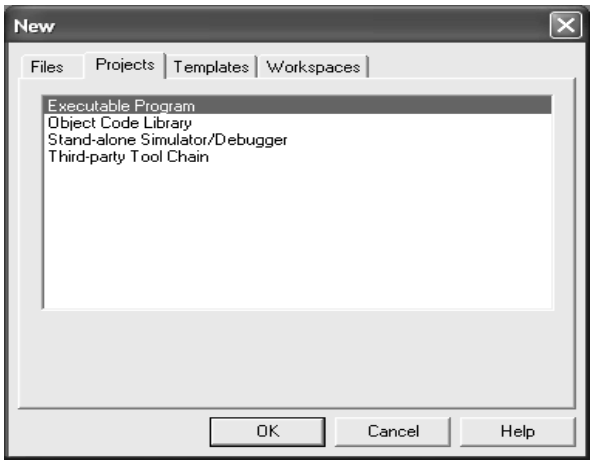


Fig 16 new dialog box.

4. Select an executable program. This will allow the user to select the target. Here, we need to select the target family as 8051, group NXP and family member as NXP89C51RD2, configured for normal operation as shown in fig below –

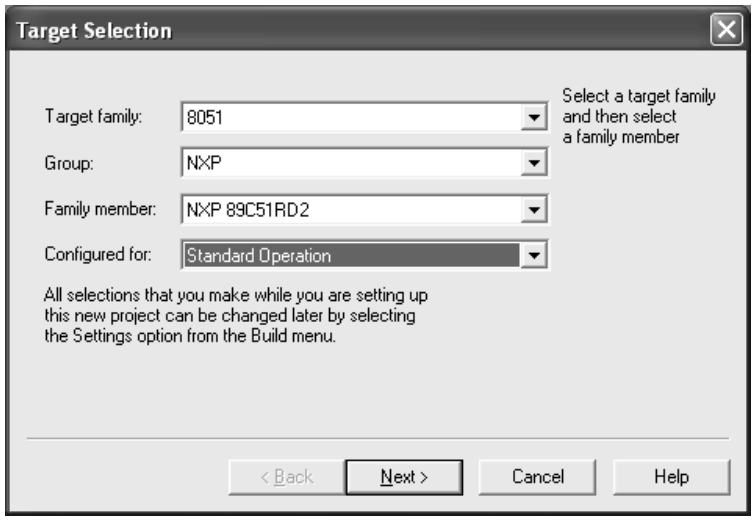


Fig 17 Target Selection

5. Chose the directory to be used to store the file and give a suitable name to the project without any extension.

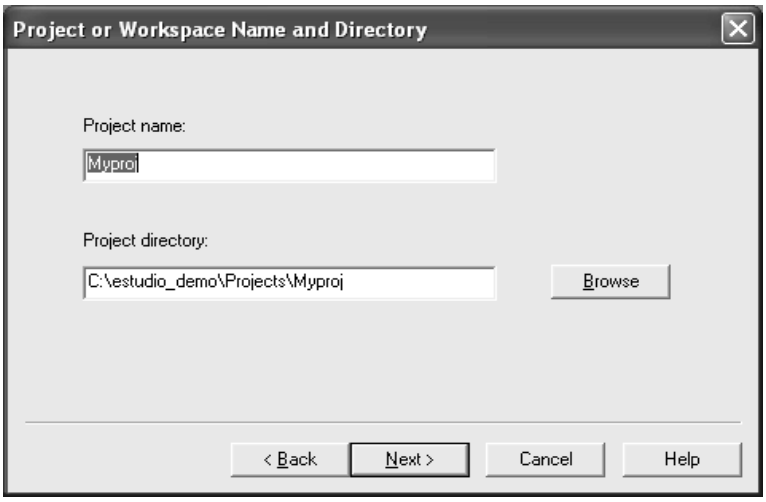


Fig 18 Save the Project

6. Compiler automatically selects min and max address of code n data RAM, however we need to select memory model of compiler configuration tab to 'Large'.

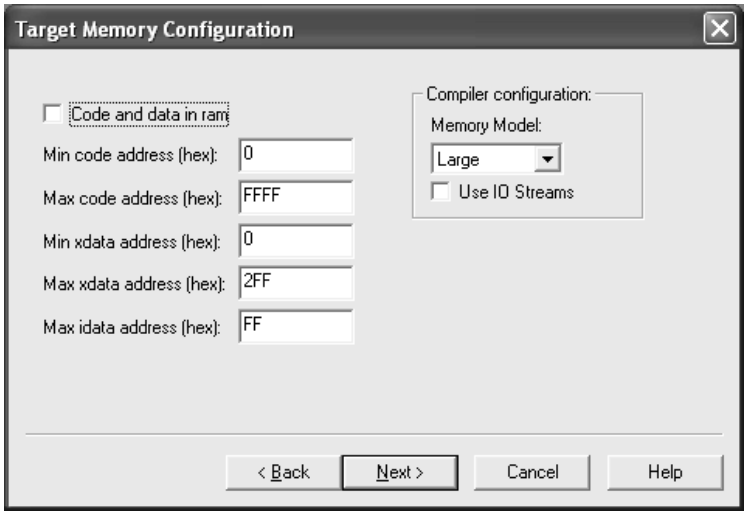


Fig 19 Compiler memory configuration

7. There are two options we can chose in the program file format. Intel hex format which will be the source file for the Flash Magic software which will in turn communicate with 8051 microcontroller and hardware assembly connected into it. Other option is to use IEEE 695 file which can be used for step by step execution and stimulation of the result. To directly use the file to run the hardware assembly chose intel hex format. To simulate the output, use IEEE 695 format.

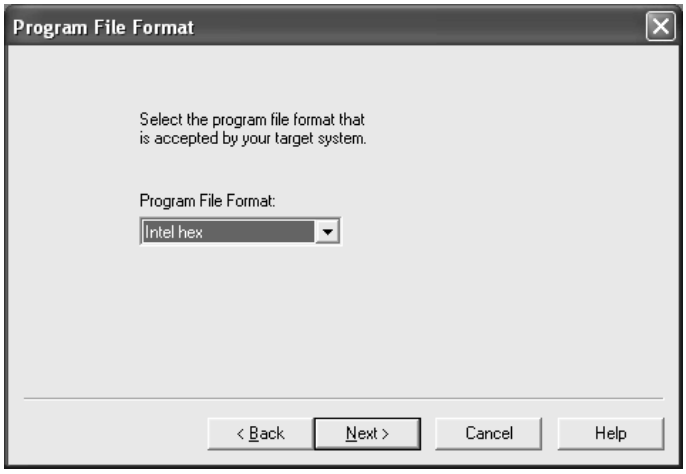


Fig 20 Program file format

8. In the next dialog box, cancel creation of startup file. And select C as the programming language.

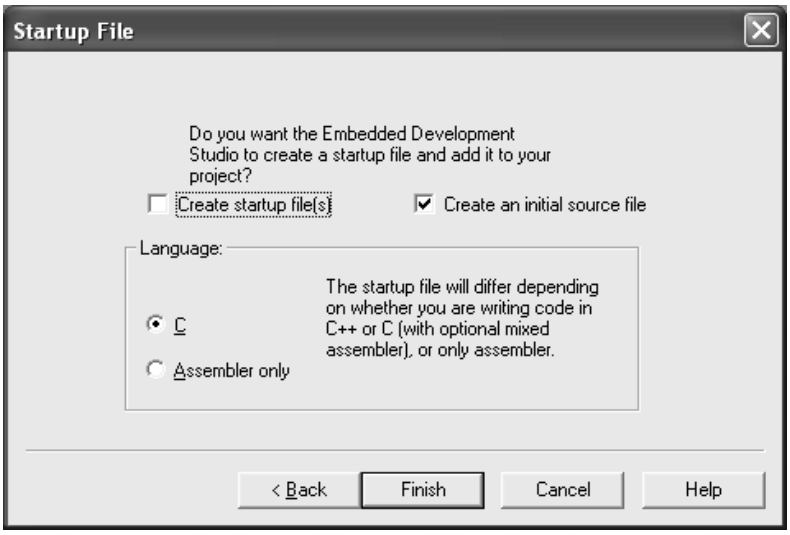


Fig 21 Final step in creation of project

9. After creation of project IDE will look as follows –

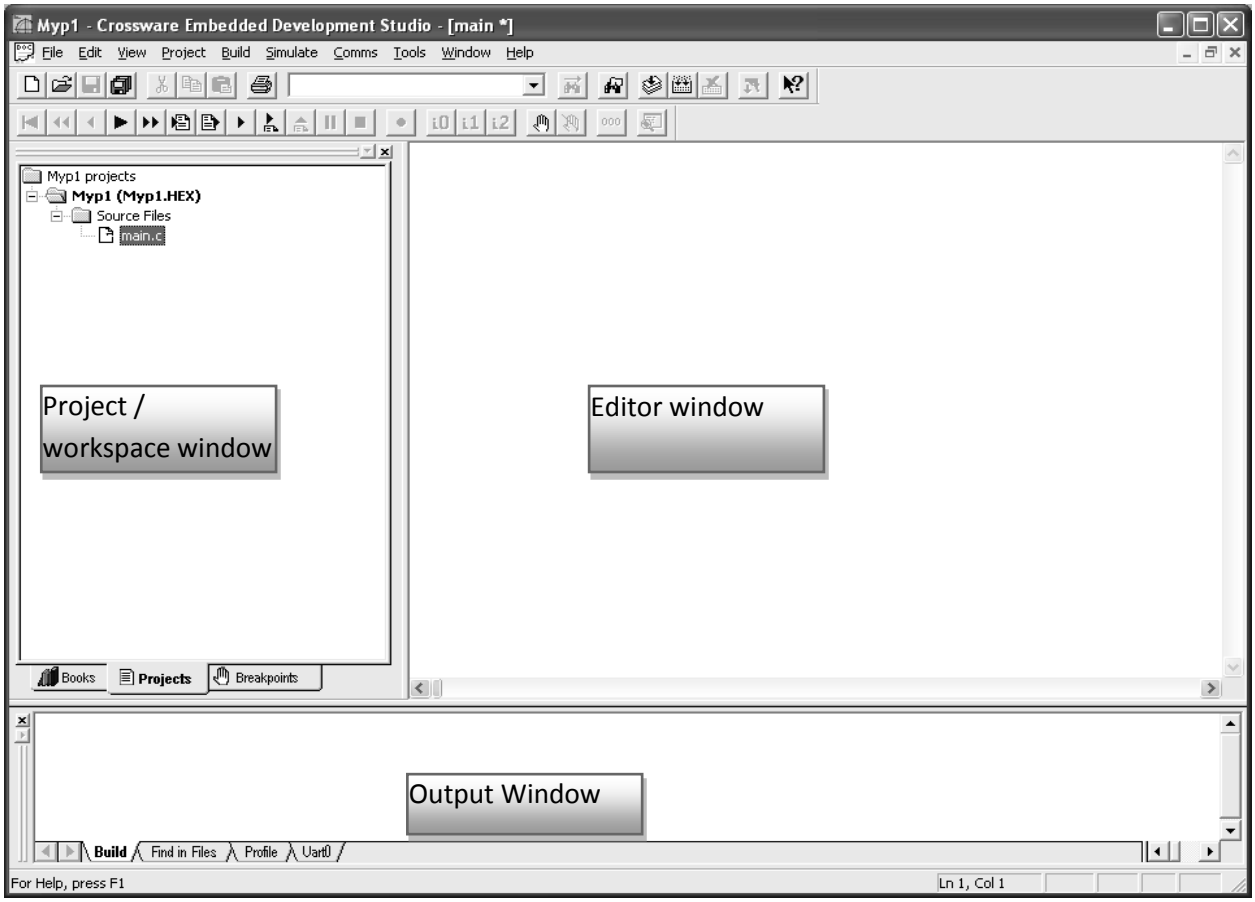


Fig 22 Crossware IDE

10. Write the embedded c program in the main.c, which is open in the editor window. Save the file. Compile to create object file.
11. To create the hex file, build the complied main.c file. This will create the required hex file. We can use the hex file to burn the flash memory of the microcontroller.
12. To create a new crossware project, first close the existing project then open a new workspace. We can now create a new project or open existing one. Note that a project cannot be compiled or saved without having a workspace.

Simulation of the program

1. Compile and build the file main.c. Chose the Simulate menu and
   a. Go to 'Step Info' to see the result at every step, 'Go' to directly see the output directly.
   b. 'step to cursor' option can be used to see the results at a desired location.
   c. To see the register contents at every execution step, go to window – registers.

d. To see results in memory, select window- memory. Memory includes external data (Xdata), Internal memory (Idata), Special Function Register memory (SFR) and code memory (code) as shown in fig below.
e. We can also see the results on PORTs, PSW, Interrupts from view – appropriate option.

2. After finishing simulation terminate and return to the main screen.

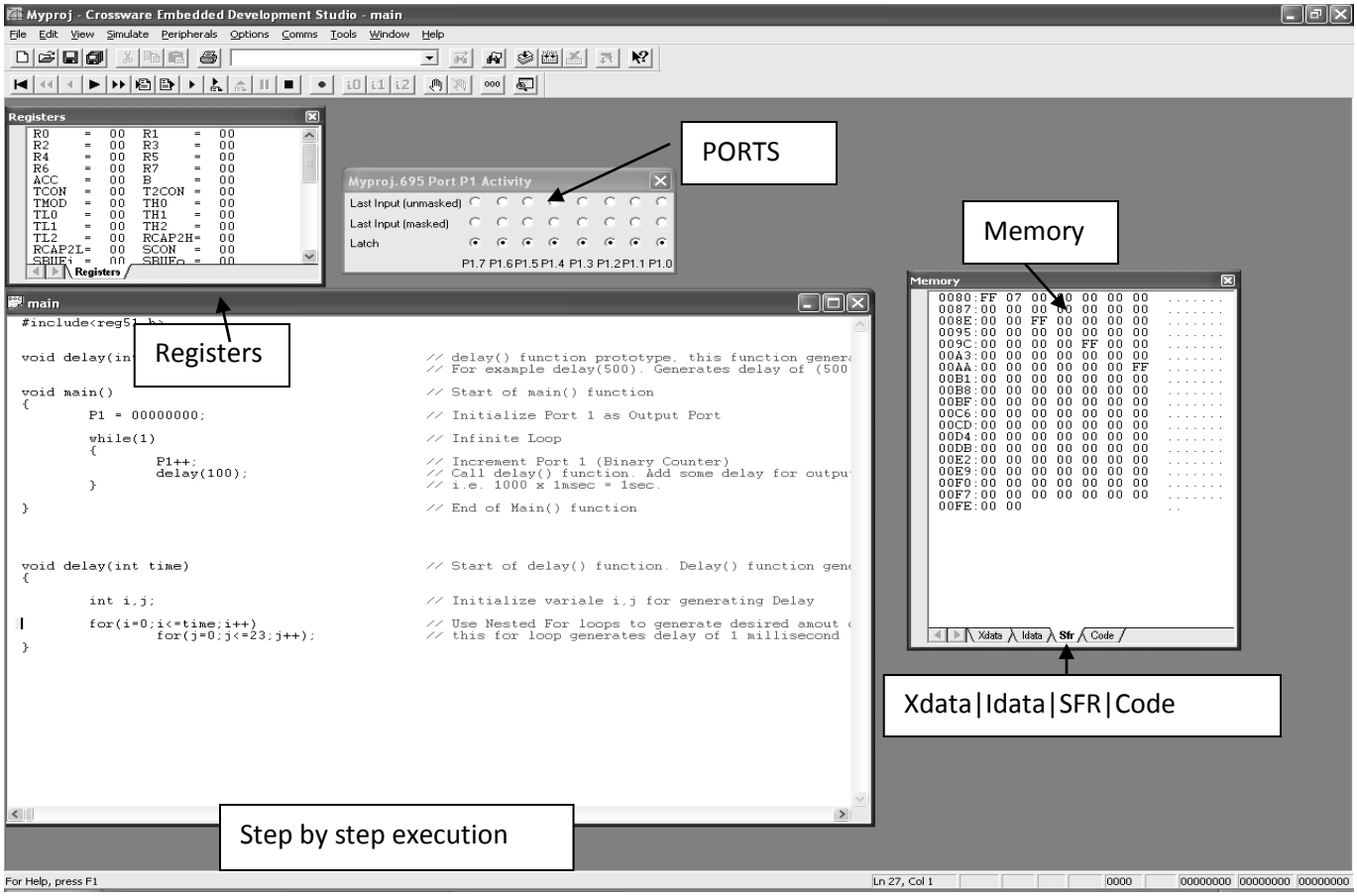


Fig 23 Stimulation output of the program with contents of ports, memory and registers

Use of µvision Keil

Example of a popular IDE development tool is µvision. µvision is a licensed IDE tool from Keil software (www.keil.com) company. The demo tool can be installed from the web site. To start with the IDE, double click the executable file or choose the Keil µvision X (X is version number) from the start menu – all programs on the host computer with Microsoft © Windows operating system. The IDE view is shown in Fig 24

IDE consists of project window consisting of Register view, menu bars, text editor window, output window etc.

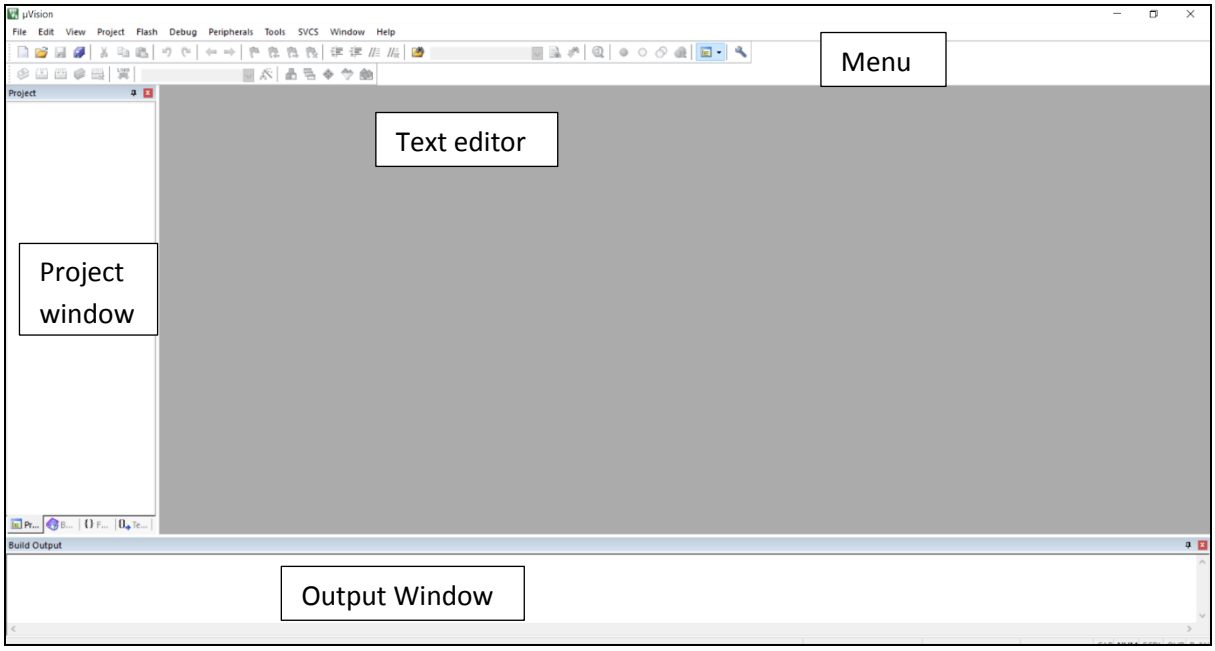


Fig 24 Keil µvision 5 Integrated Development Environment

Working with µvision

Basic steps in using µvision 5 are as follows –

Step 1

Create a new project. Name and Save project.
Select the device for target. First select the vendor(Atmel). Then choose the target processors (89C51). This is shown in Fig 25

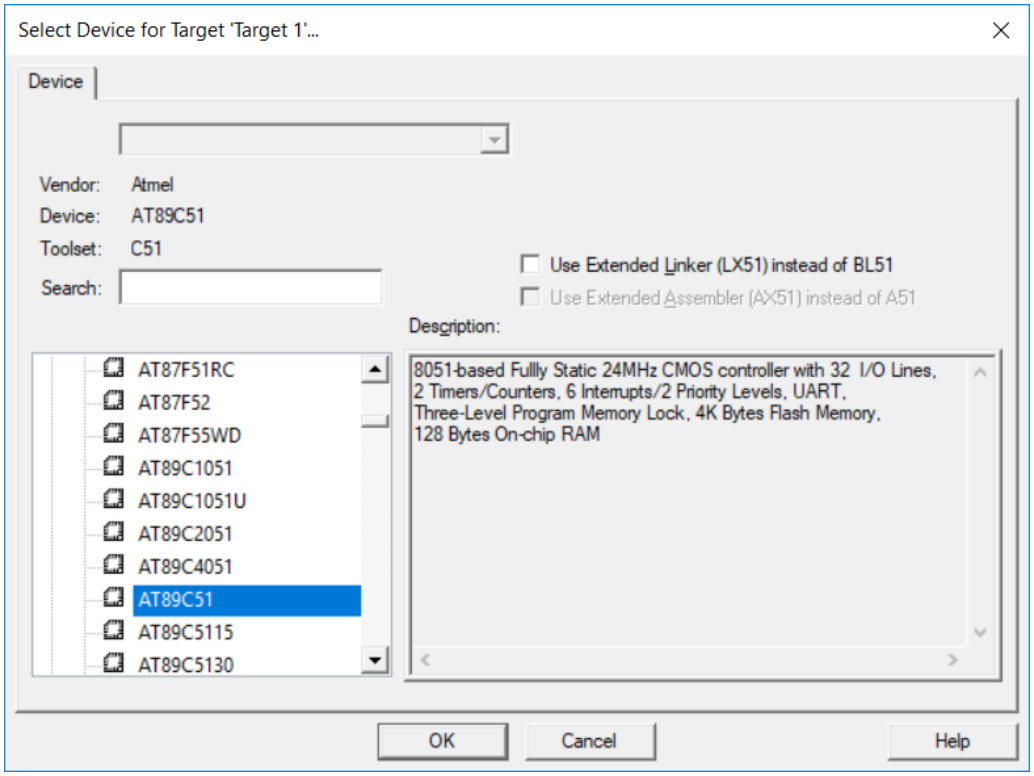


Fig 25 Target CPU Selection for Keil µvision 5 IDE

Step 2
Once target processor is selected, IDE will automatically add required start up assembly files to the project. The dialog box shown in Fig 26 is prompted.

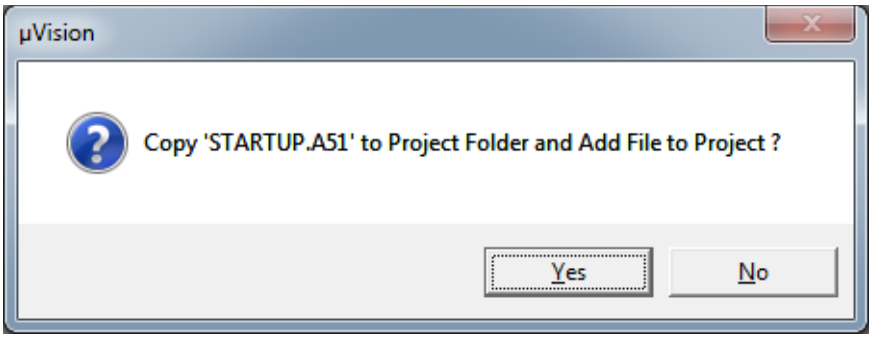


Figure 26 Startup File addition

A target group with the name Target1 is automatically created under Files section of the project window. Target1 contains the source group and start up file. This is shown in Fig 27

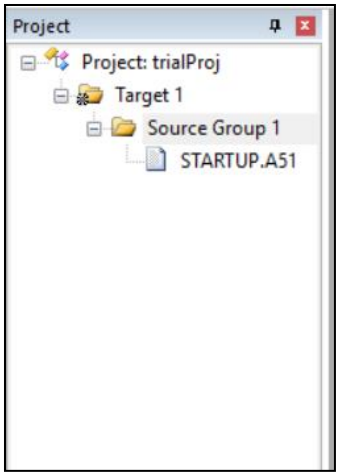


Fig 27 Project window with Target created and startup added

Step 3

Add a new file to the project.
One method is to click on the File – New option. Now a text file will be created. It needs to be saved and then added to the project.
Another method will be to right click on the source group and directly add a new c file.
Now the file can be edited.

A header file "reg51.h" is needed in the C program. This includes all target specific declarations for 8051 family.

Fig 28 shows a C file added to the project to show the "hello world" message.

The programming implements a modular approach. Only one of the files in the source group can have the main () function. ***More than one file containing main () function will result into an error.***

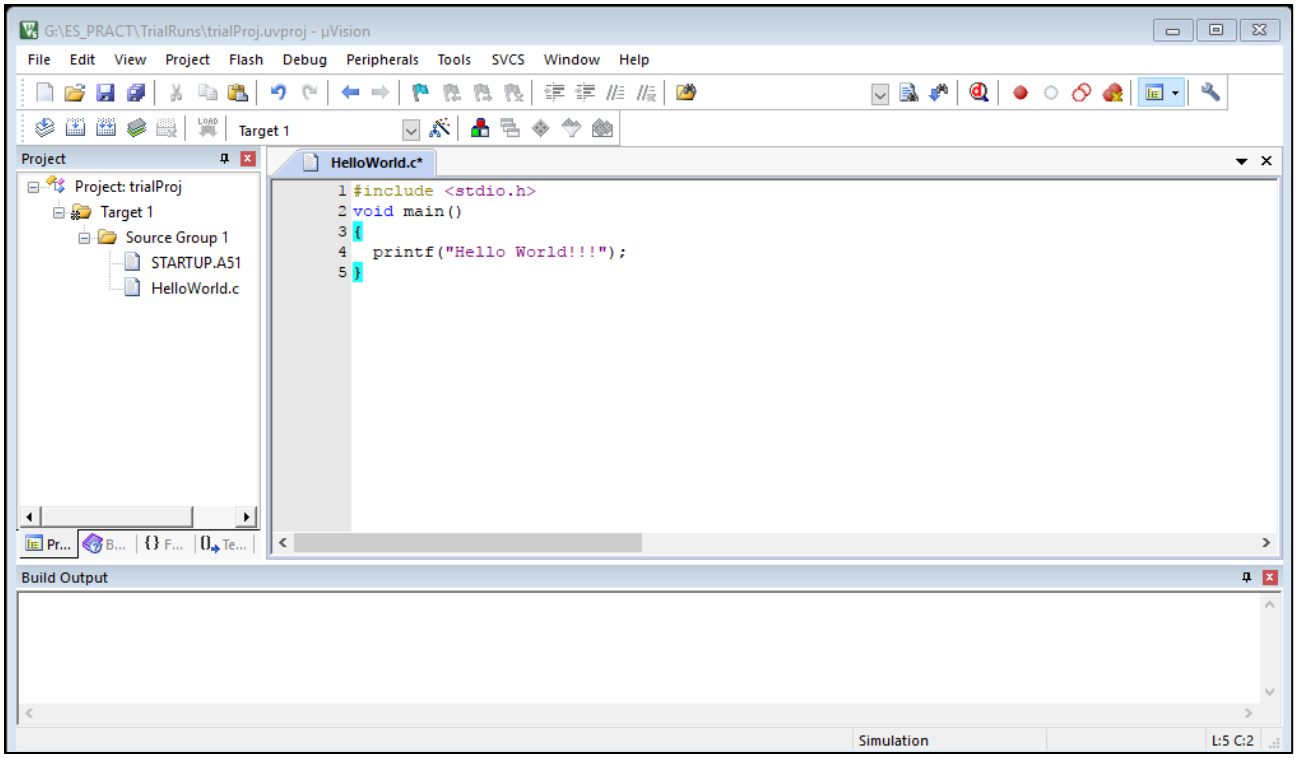


Fig 28 HellowWorld.c added to the project.

Step 4
Target must now be configured.
For target configuration, go to the target properties by using the popup menu as shown in Fig 29 from the options.
Device configuration is already done at the time of device selection.
Now select the target tab and configure the following –

Choose on chip or external ROM. If external code memory is used specify the starting and ending address.

Select the clock frequency for the designed system. Typical values would be 6 MHz, 11.0592 MHz, 12 MHz etc. (Select 11.0592MHz for AT89C51)
Output tab holds setting for output file generation from the source code. the source file can either be converted into executable hex file, or a library file. The output and Target settings are shown in Fig 29(a), (b) and (c).

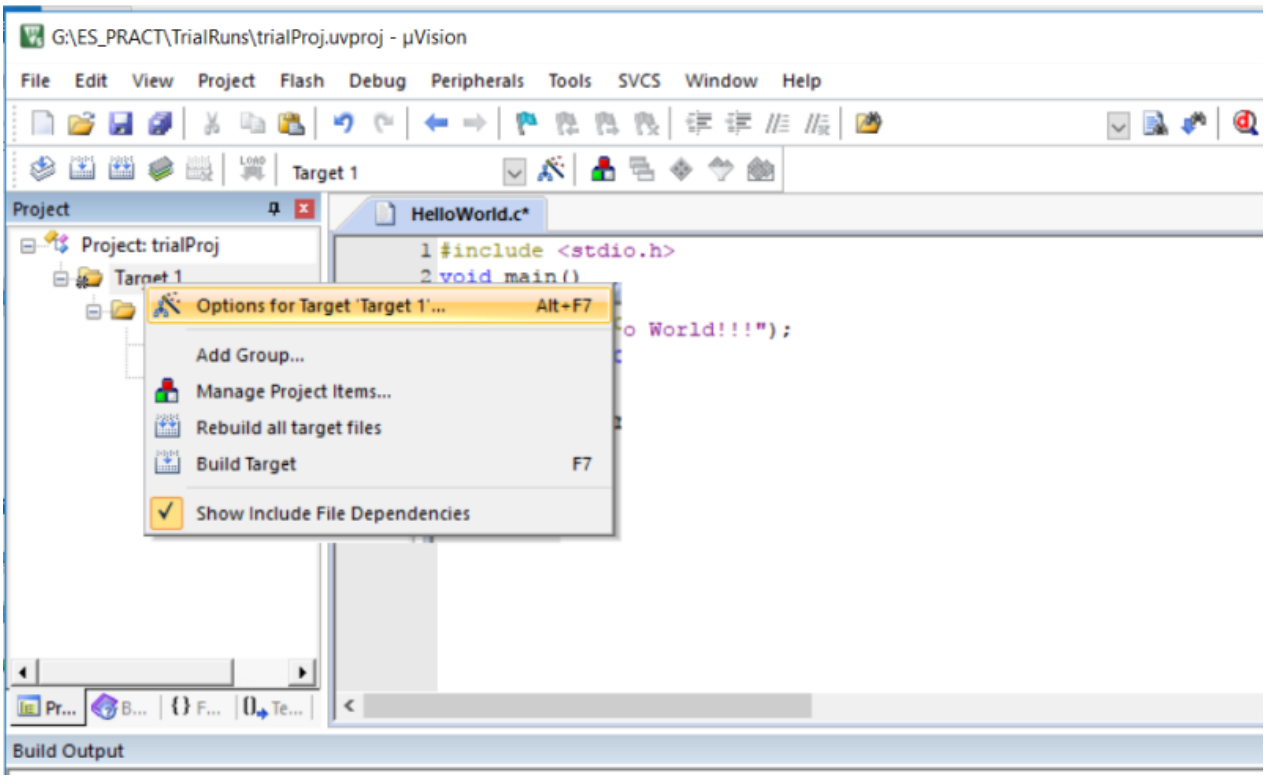


Fig 29(a) choosing options for target

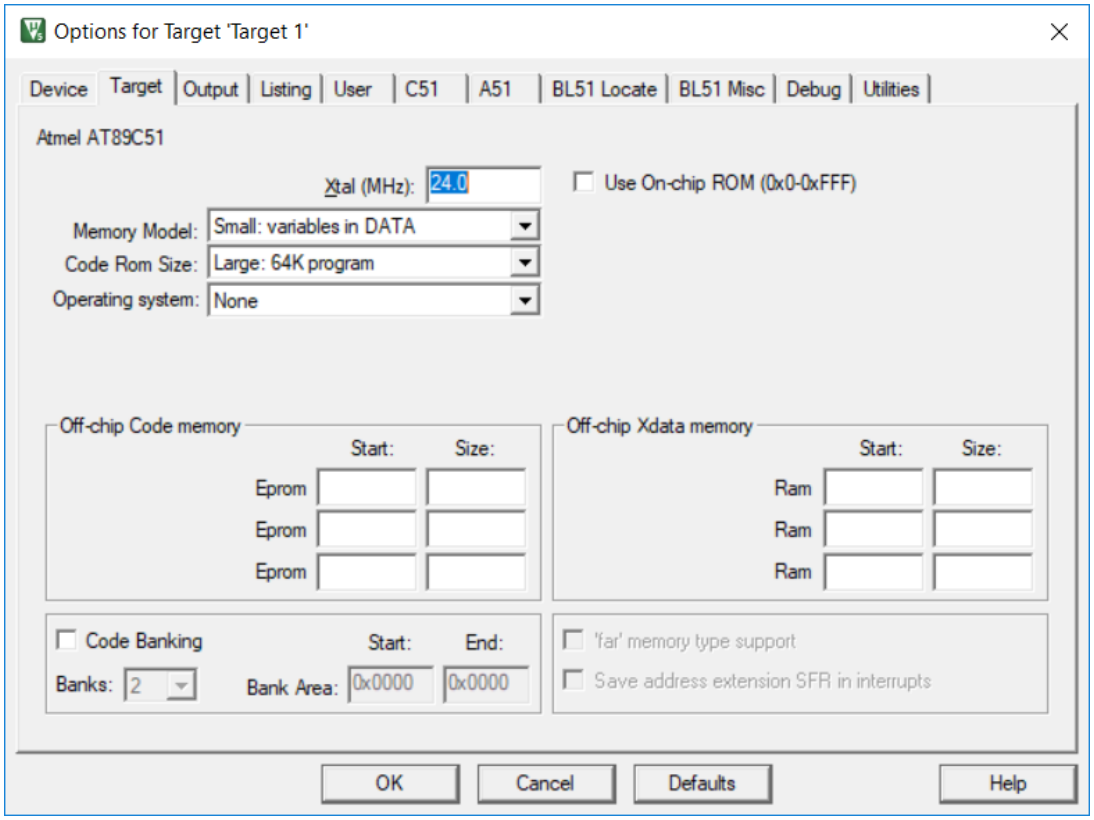


Fig 29(b) Target options

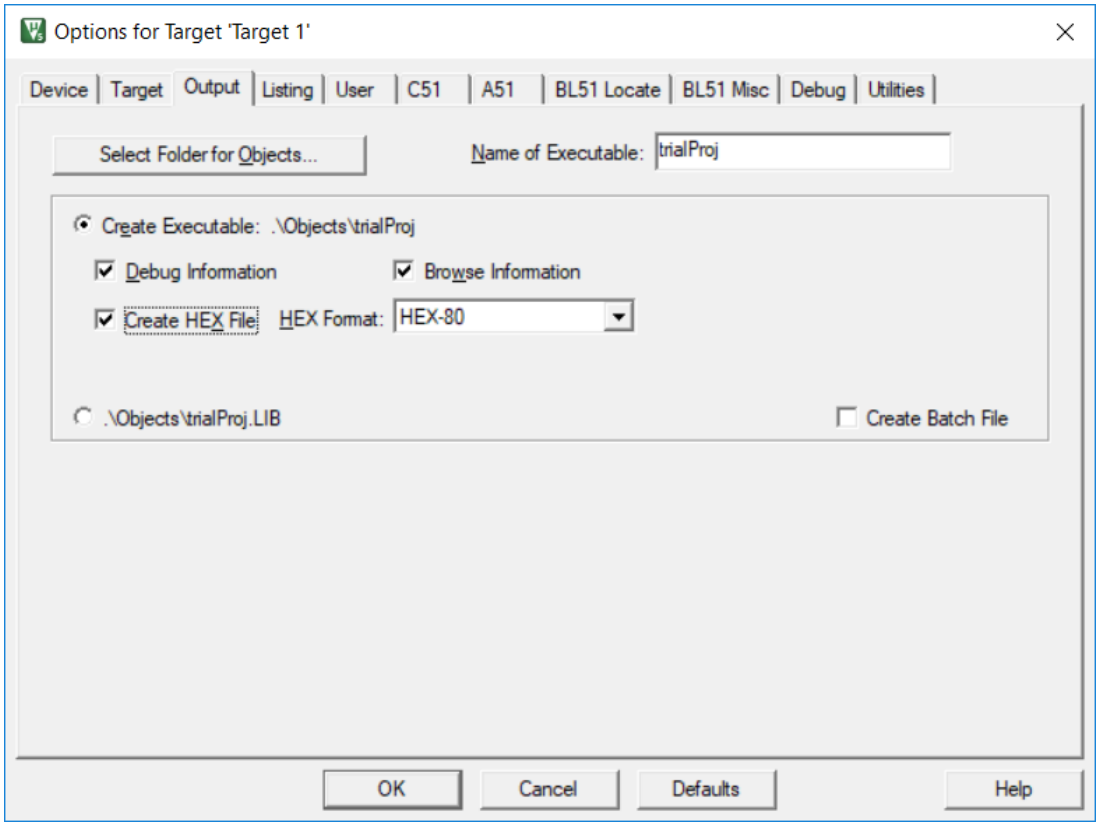


Fig 29(c) Output configuration

Debug tab consists of setting for firmware debugging.
It supports both, simulation type firmware debugging and debugging for target hardware.
If firmware option is selected, code may not be copied into target machine.
Hardware debugging is possible through the common port to which target device is connected. This is shown in Fig 30

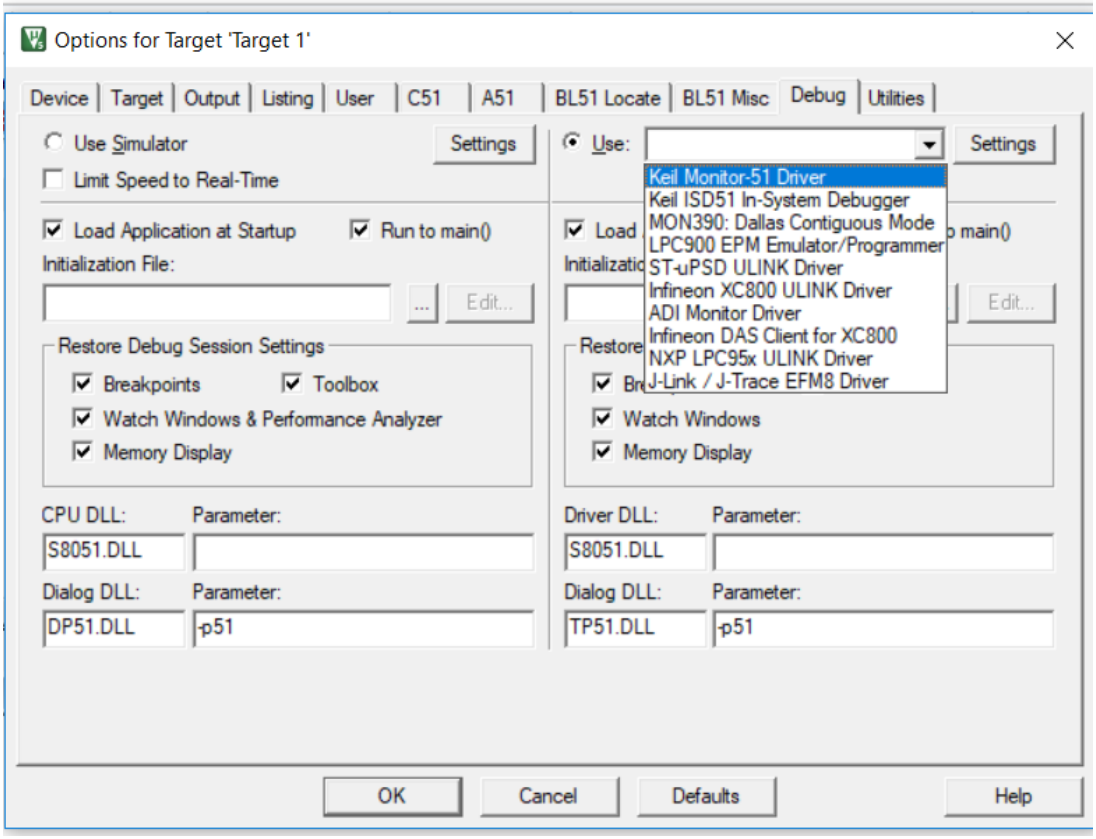


Fig 30 Firmware debugging options

Step 4

Compiling and building the target program can be done by using the build option. This will convert the code written in embedded C to machine language.
The output window after building the target of the hello world program is shown in Fig 31.

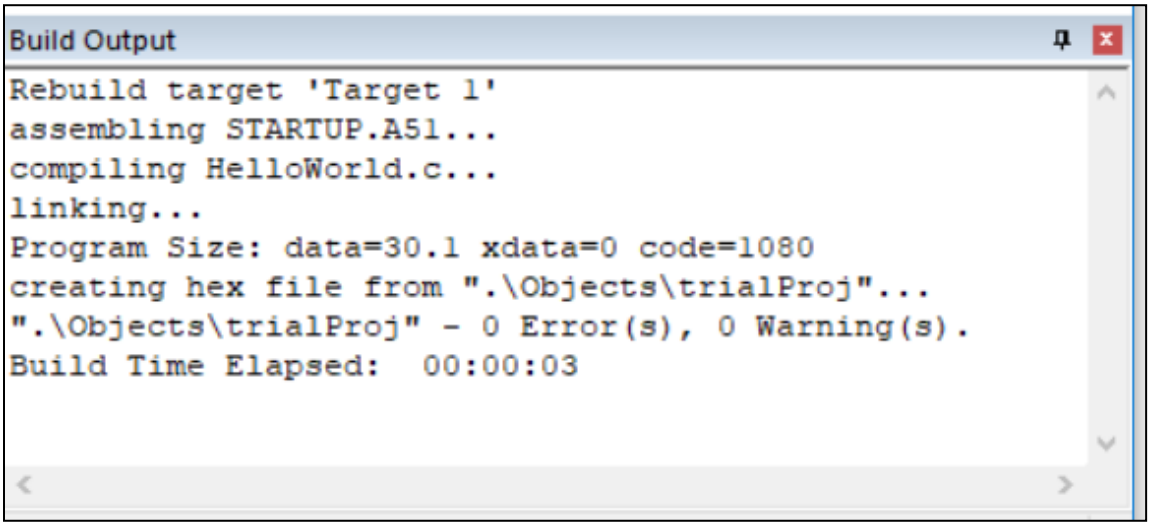


Fig 31 Build output for the HelloWorld program.

As seen in the output window, HelloWorld.c is compiled and linked using the startup.asm file for default initializations.
Output shows the size of data memory (data = 30), code memory (code = 1080)

The target is now ready. The IDE can be further used to simulate the output in the disassembly and register window.

Using Flash Magic

Software required to use the hex file in the embedded systems is flash magic. Run the flash magic software. Following interface appears

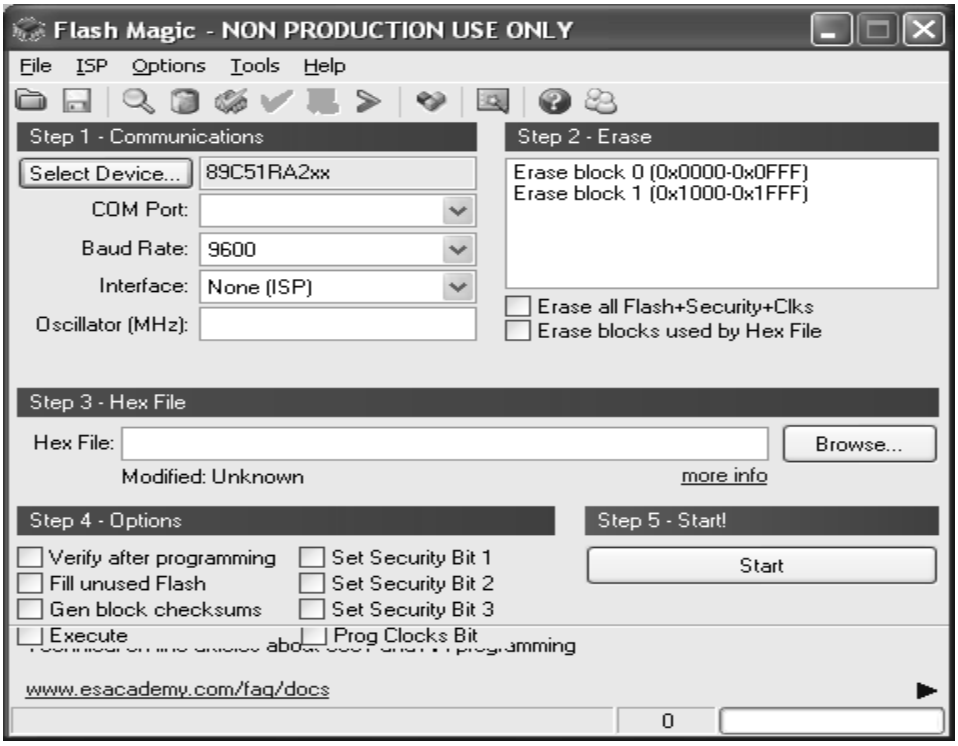


Fig 32 flash magic interface

Step 1
Select the communication device as 89V51RD2. This is the NXP microcontroller used for executing the embedded system programs.

Step 2
Select the appropriate COM port. If a serial cable is used to connect the TKbase kit to the PC at the default serial port, this would be a COM1 port. It can be checked from control panel – hardware and sound – Device Manager - ports

Step 3
Select COM1 and baud rate as 9600.

If a USB to serial converter is used to connect the PC with the TKbase trainer, the device drivers would be installed for a relevant COM port.
This can be checked in the Device manager. Select the relevant COM port. Baud rate for this communication will however, remain 9600.

Step 4
Select the default None (ISP) interface.

Step 5
From options – Advance options – hardware config tab, uncheck the marks for use DTR for control RST as well as Assert DTR and RTS while COM port open option. This is shown below

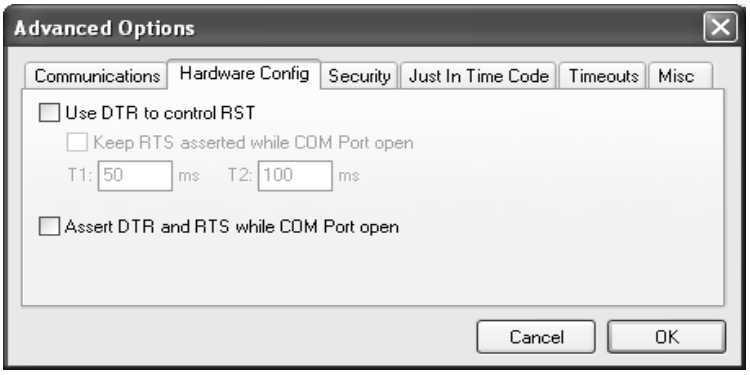


Fig 33 setting options for flash magic communication software

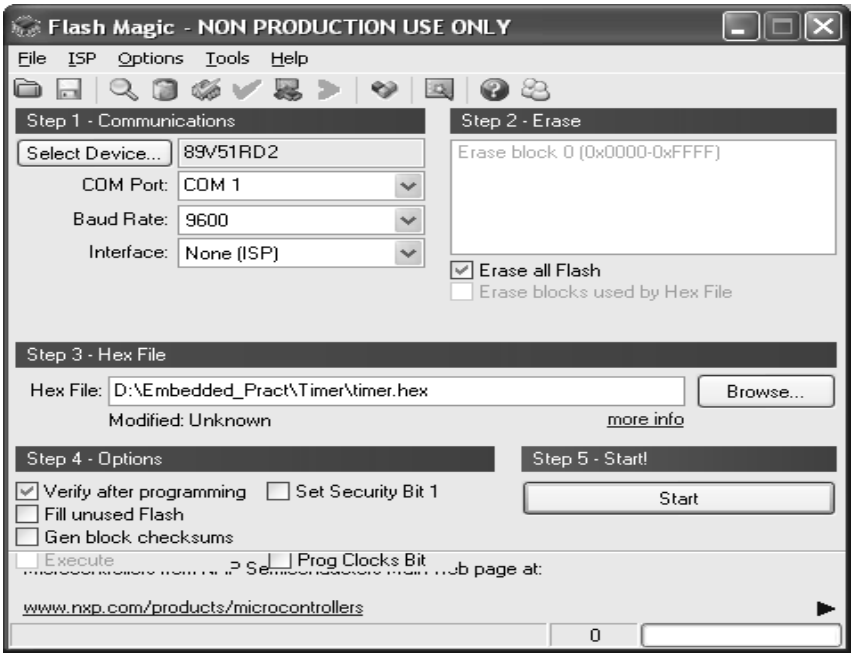


Fig 34 after all correct settings

Step 6

Choose to erase all flash memory contents to make place or to load the fresh program into 8051 memory.

Step 7

Browse and select the compiled and build program in the form of hex file.
Choose the verify after programming option in step4. Do not check any other option.

Step 8

Connect the TKbase trainer kit with the PC using relevant port. Use the serial data cable, or USB to serial converter or bridge for this connection. Do necessary hardware connections.

Step 9

Give power to the kit. Press the reset button on the kit. Then click on the start button of the flash magic interface as shown above. After the software prompts to reset device in ISP mode, press reset again.

Step 10

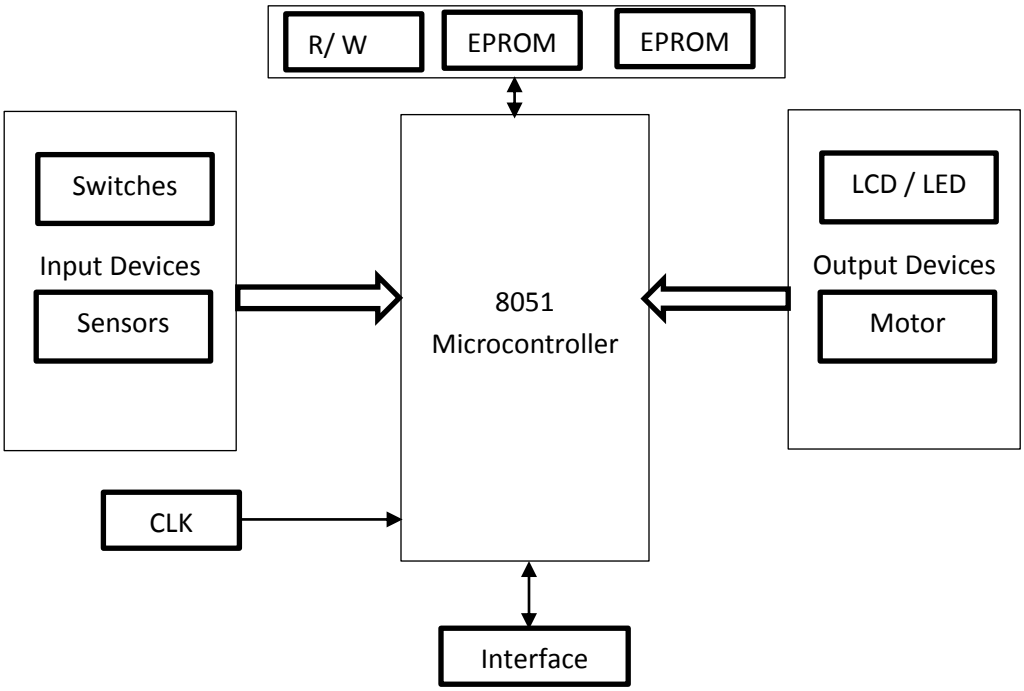
Now the hex file is downloaded. Press reset again to execute the downloaded program.

Practical 1

Design and develop a reprogrammable embedded computer using 8051 microcontrollers and to show the following aspects.

a. Programming
b. Execution
c. Debugging

The components of a reprogrammable embedded computer using 8051 microcontroller are shown below –



Block diagram of 8051 microcontroller embedded system

A reprogrammable can be considered as an embedded system which can be reprogrammed number of times easily, while providing flexibility. In other words, we can say that for a reprogrammable embedded system, Microcontroller unit can be reprogrammed without actually completely pulling out the MCU.

The reprogrammable microcontroller based embedded system also provides debug facility for users.

The Reprogrammable embedded system consists of:

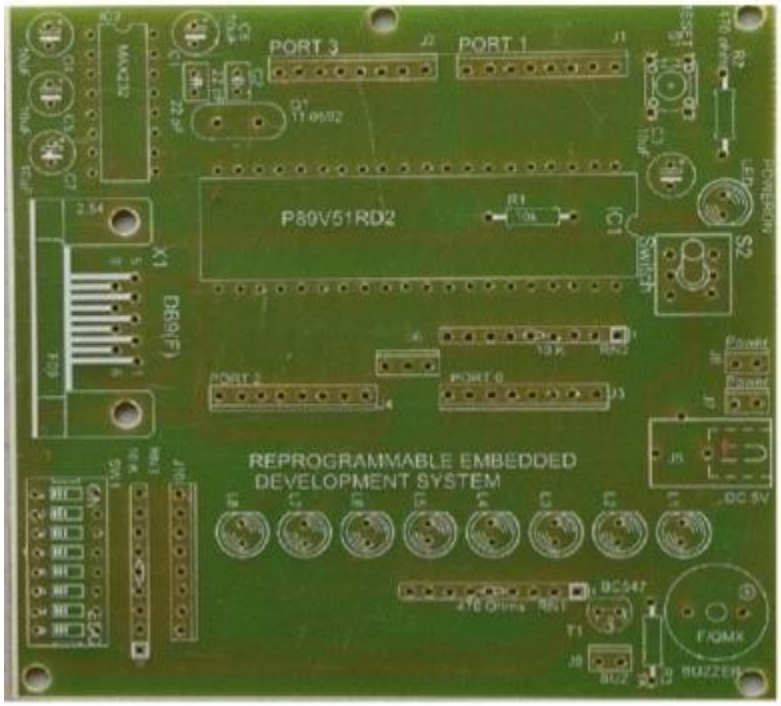
- Sockets for placing microcontroller- 40 pin

- DC socket for external power supply (DC 5V)
- 1 LED for power on indication and 1 push button for reset
- 11.0592 MHz Quartz Crystal Oscillator
- 8 LEDs for output pin state indication at port P0
- 1 DIP switch (8 switch) for input pin activation
- Connector and driver for serial communication RS232
- Multiple-pin connectors for direct access to I/O ports
- Connector for SPI programming
- 1 Piezo buzzer for audio/frequency output
- Additional power supply connectors

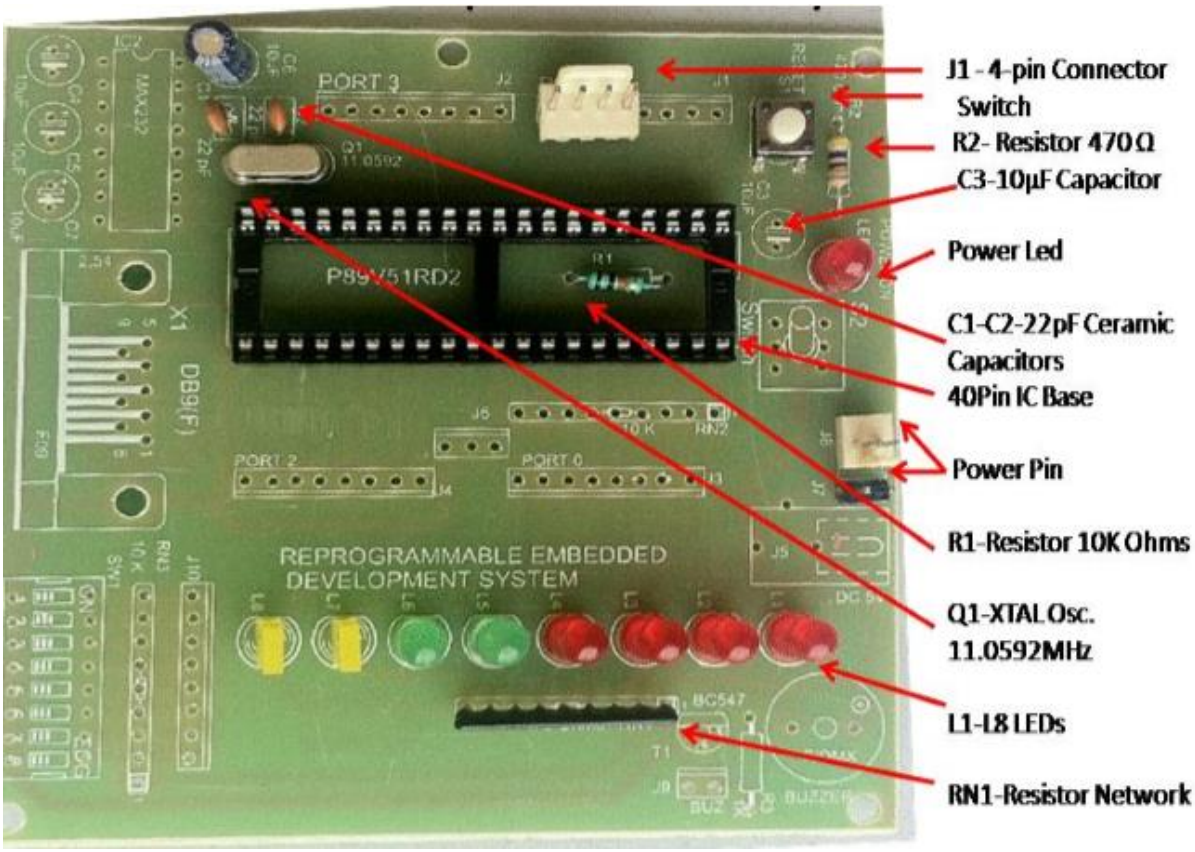
Other components include –

- Serial communication Interface – UART, MAX 232 Interface
- Oscillator – piezo electric crystal oscillator with frequency – 11.06592 MHz
- Connector (Max 232IC for serial and on board 8 bit connectors for parallel communication.
- Input / output Selection – through 8 bit on board connectors.
- Power supply – connector to external power supply. (DC 5V)

Figure below shows the bare board PCB and PCB with components.



Bare Board PCB



Assembled PCB for reprogrammable Embedded System

a. Programming
b. Executing and
c. Debugging

of the reprogrammable embedded system is explained in section Introduction to development environment.

The tools discussed are - Keil µVision 5 and Crossware C demonstration tool.

Practical 2.

(a)

Configure timer control registers of 8051 and develop a program to generate given time delay.

Problem Definition

Write an embedded C 8051 program to generate time delay for 1 second using the timer control register. Demonstrate the delay by interfacing 8 LEDs with parallel port P0, and blinking the LEDs with 1 s delay.

Algorithm

1. Initialize Port 0 as output port.
2. Send a low value to the port 0 to turn off the LED. i.e. Move the data value 00 to port 0. This will switch off the LEDs connected at port 0.
3. Call delay function which will wait for calculated time interval.
4. Send active high value to the port 0 to turn on the LED. i.e. Move the data value FF to port 0 this will glow all the LEDs connected at port 0.
5. Call delay
6. Repeat steps 2 – 5 in infinite loop.

Calculation of delay:

Consider the crystal frequency, that for 8051 is XTAL = 11.0592 MHz (12MHz).
Each machine cycle is divided into 12 clock cycles hence, clock frequency is –
11.0592 / 12 = 0.9216
Clock period = 1 / 0.9216 = 1.085 µsec

To generate a delay of specific time, a loop is executed, say n times.

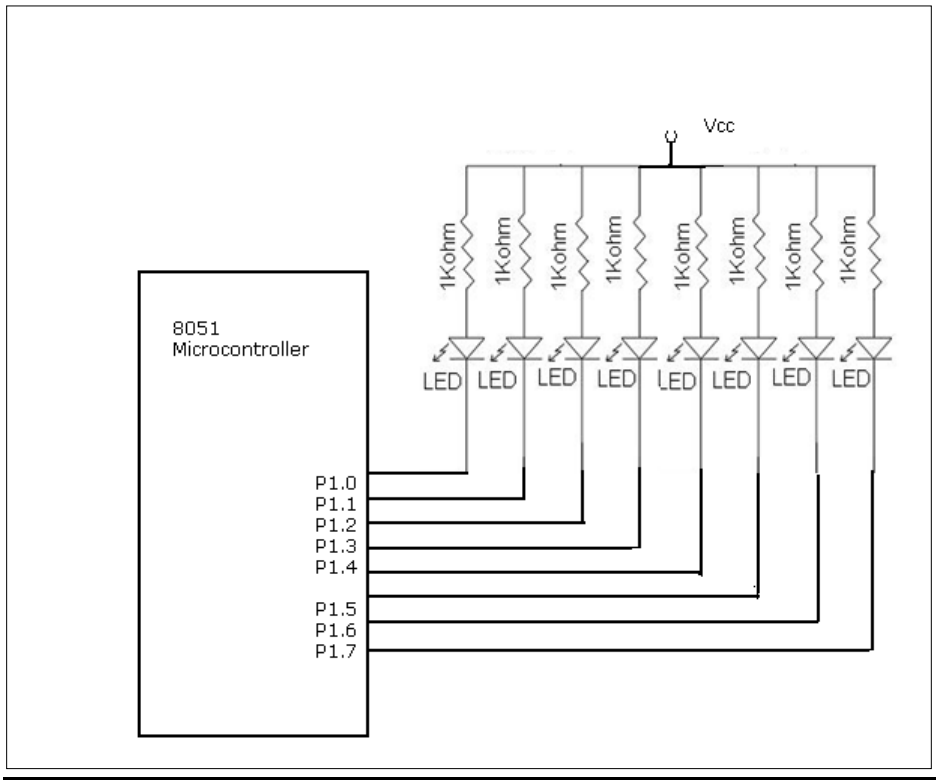
For 1s, n = 1000 ms / 1.085 µsec = 921.658

As 8051 microcontroller uses relative addressing, jump is made FFFF – n no of times. Thus,

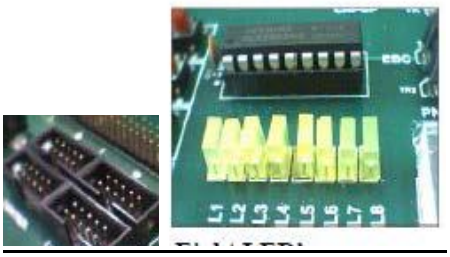
65536 (FFFF H) – n = N

For 1s, N = 65536 - 921.658 = 64614.34
convert N to hex yyxx
For 1 s hex value of N = FC65 H
Set TL = xx and TH = yy
For 1s TL = 65H and TH = FCH

Hardware connections



Circuit Dig of interfacing the LEDs with 8051 microcontroller using port1



Connectors used and the LED interface with buffer

FRC cable used to interconnect the LED interface and 8051 port1

Steps in Execution

Open suitable development environment and create a new project.

Add a new C file to the project and save it as timer.c

Write the appropriate code.

Build the hex file.

Use Flash Magic software to load the program in 8051 microcontroller.

Execute the program and observe output.

Embedded C Code

```
#include<REG51.H>
void T0Delay(void);
void main(void)
{
	while (1)
	{
		P1=0xFF;		//make high all bits of port 1
		ToDelay();

		P1=0x00;		// make low all bits of port 1
		ToDelay();	}
}
void ToDelay()			//delay function
{
	TMOD=0x01;			// use timer 0 , mode 1
//Load value "FC" in timer 0 lower byte & load //value "65" in timer 0 higher byte for 1 second //delay
```

```
TL0=0xFC;           // load TL0
        TH0=0x65;           //load TH0
        TR0=1;              // turn on Timer 0
while (TF0==0);    //wait for TF0(timer 0 overflow flag) to roll //over(overflow)
        TR0=0;              //turn off timer 0 run control bit
        TF0=0;              // clear timer 0 overflow flag bit.
}
```

Expected Output

(b)

To demonstrate use of general purpose port i.e. Input/ output port of two controllers for data transfer between them.

Problem Definition

Write an embedded C 8051 program to read a one bit data using input port (port 1) of the first 8051 microcontroller. Set port 2 as output port. Connect port 2 of the first microcontroller to input port (port 1) of second 8051 microcontroller. Set port 2 as output port and the received input should be redirected to output port (port 2) of second 8051 microcontroller.

Algorithm

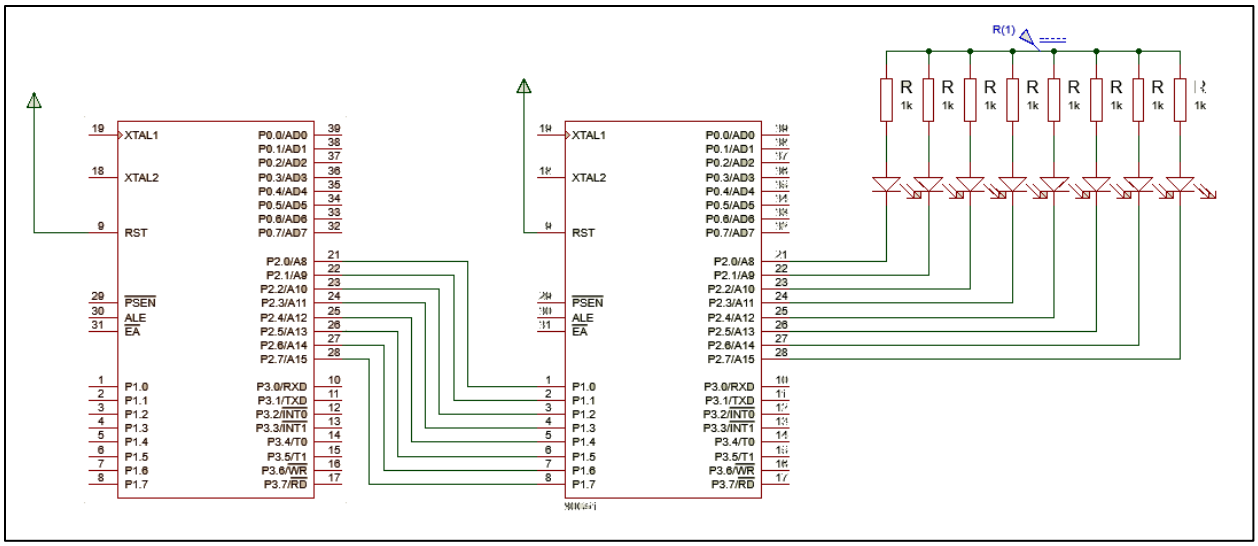
First Microcontroller

1. Set Port 2 as output port.
2. Set a Hex data value at port2.

Second Microcontroller

1. Set P1 as input port and port 2 as output port.
2. Redirect input from port 1 to port 2.

Hardware Connections



Code

Controller 1:

```
#include<reg51.h>
void main()
{
        unsigned int i;
        while (1)
                P2 = 0X55;              //Set up a hex value at port 2 of first controller
         for (i=0;i<1000;i++);
}
```

Controller 2 :

```
#include<reg51.h>
void delay(unsigned int time);
void main()
{
      while(1)
       {
              P2 = P3;
              delay(1000);
       }
}
void delay(unsigned int time)
{       unsigned int i;
       for(i=0;i<time;i++);
}
```

Practical 3

(a)

Port I / O: Use one of the four ports of 8051 for O/P interfaced to eight LED's. Simulate binary counter (8 bit) on LED's

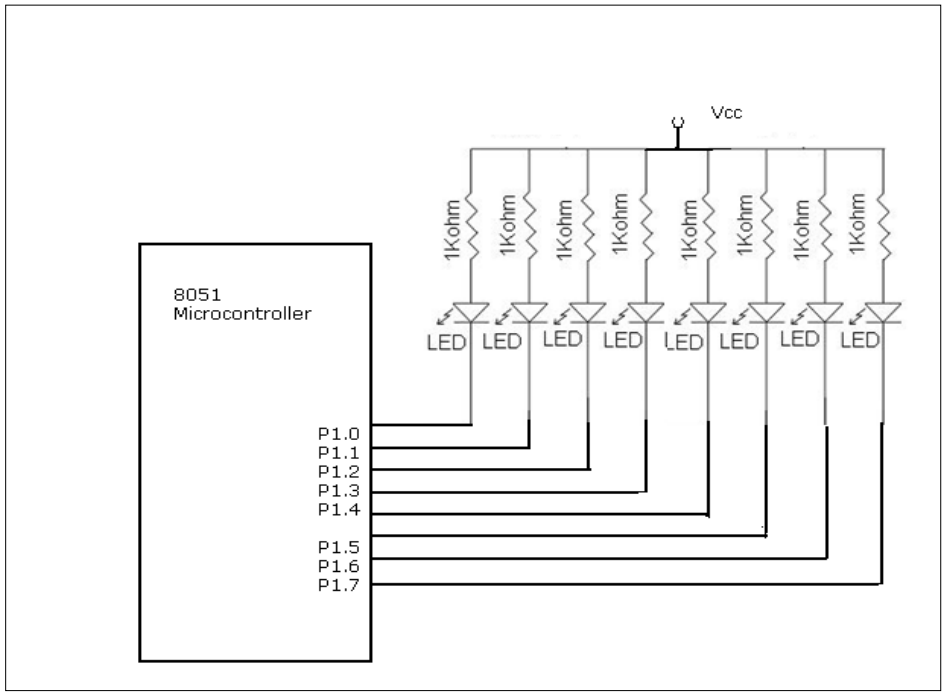
Problem Definition

Write an embedded C program to configure Port 1 of 8051 microcontroller as output port. Construct a binary counter output at port 1 of 8051. The counter should start counting from 00 H to FF H. Demonstrate the output using 8 bit LED interface.

Algorithm

1. Initialize port1 as output port.
2. Move the binary data value 00000000 to port1 this will switch off the LEDs connected at port1.
3. Call delay function which will wait for calculated time interval.
4. Increment the binary value by 1.
5. Call delay
6. Repeat steps 2 – 5 in infinite loop.

Hardware Connection

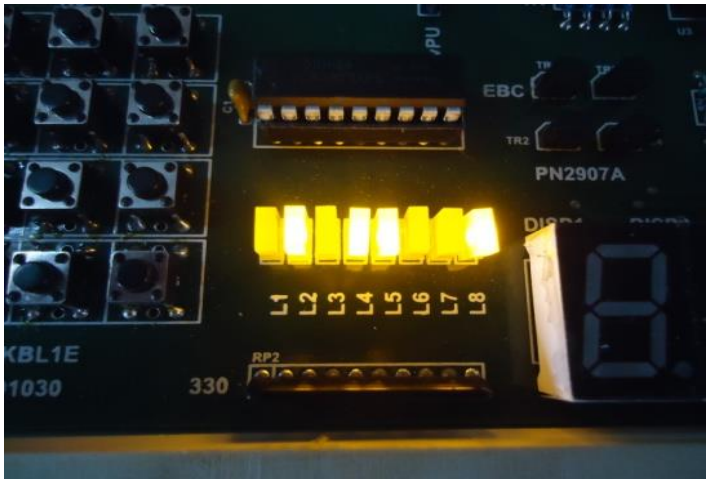


Actual Connections

Code

```
#include<reg51.h>
void delay(int time);

void main()
{
	P1 = 00000000;			// Initialize Port 1 as Output Port
	while(1)
	{
		P1++;			// Increment Port 1 (Binary Counter)
		delay(100);
	}
}

void delay(int time)
{
	int i,j;
	for(i=0;i<=time;i++)
		for(j=0;j<=23;j++);
}
```

Expected Output



Binary Counter with output 10011010 (9A)

(b)

To interface 8 LEDs at Input-output port and create different patterns.

Problem Definition

Configure port 1 of 8051 microcontroller as output port. Connect a 8 bit LED interface at port 1. Write an embedded C program to generate different patterns and display on the LED interface.

Algorithm

1. Set port 1 as output port.
2. set a value to generate a pattern. For example, 55 H will turn ON the alternate LEDs
3. Call delay.
4. Change pattern.
5. Repeat steps 2 to 4 in an infinite loop.

Hardware Connections

Same as 3 (a)

Code

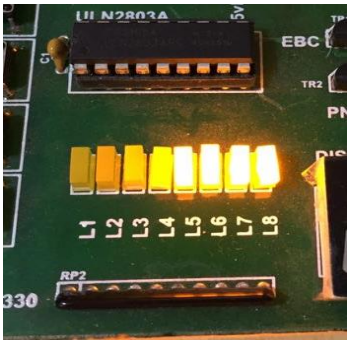
```
#include<REG51.H>
void ToDelay(void);                 // delay() function prototype

void main(void)
{
      while(1)
      {
            P1=0xAA;          // pattern to turn on alternate LEDs
            ToDelay();
            P1=0x55;          // Reverse the sequence.
            ToDelay();
      }
}

void ToDelay()
{
      unsigned int i,j;
      for (j =0; j<24; j++)
         for(i=0;i<1000;i++);
}
```

Expected Output

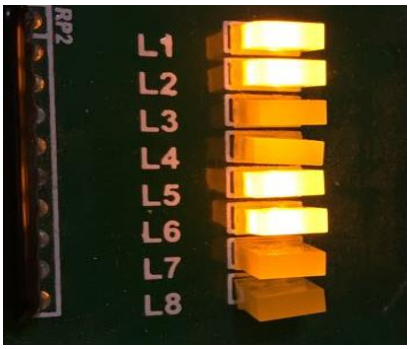
LED Interface showing alternate LEDs turned ON

Some other patterns can be –

Four LEDs ON and four LEDs OFF; or alternate nibbles. (F0H and 0FH)



Two LEDs ON and Two OFF (11001100 – 00110011 or CCH and 33 H)



(c)

Same as Practical 2 (a)

Practical 4

(a)

Serial Communication

Problem Definition

Write a program in Embedded C language to write a data 'YES' to the terminal of PC using Serial port.

Algorithm

1. Set Timer1 in 8 bit auto reload mode
2. For the above mode, set the baud rate of 9600
3. Configure Timer1 in mode1 (8bit data, 1 start bit and 1 stop bit).
4. Start Timer1.
5. Put the data in SBUF.
6. Transmit the serial data character by character.
7. TI flag will be set to 1 when transmission is over.
8. Wait indefinitely.

Hardware Configuration

No hardware connections needed, as virtual terminal will be used.

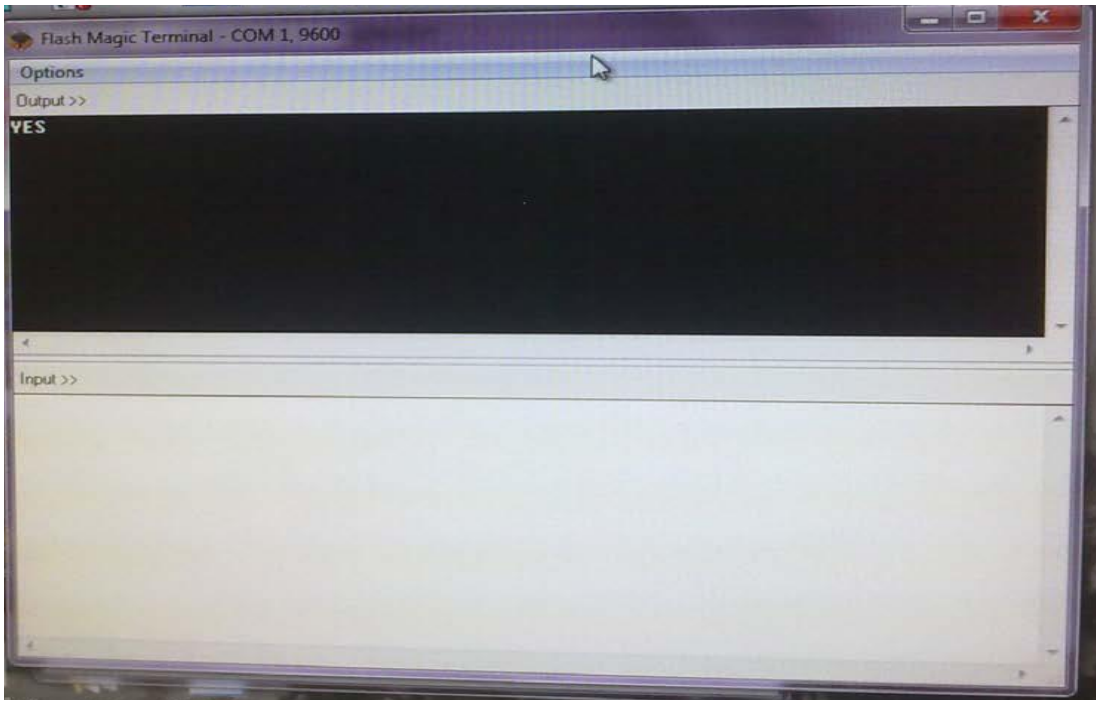
```
#include<reg51.h>
void send(char x);                  // Prototype of send() function.

void main(void)
{
    TMOD = 0x20;                    // 0x20 is stored in TMOD register to set Timer1 in 8-
                                    //Bit Auto-Reload Mode
    TH1 = 0xFD;                     // TH1 register is loaded with value 0xFD to generate
                                    //baud rate of 9600
     SCON = 0x50;                   // 0x50 is loaded into SCON register to configure
                                    //Timer1 in Mode 1.
       TR1 = 1;                     // When TR = 1.
       send('Y');
       send('E');
       send('S');
       while(1);
}
```

```
void send(char x)          // Send() function transmits the character passed to it
{
       SBUF = x;
        while(TI==0);
                           // Wait till transmission is finised i.e. wait until TI = 1
         TI=0;             // Clear TI flag
}
```

Steps in Execution

1. Using flash magic, burn the built hex file in the 8051 microcontroller.
2. Open virtual terminal from the Tools menu of Flash magic.
3. Select appropriate COM port.
4. Press he reset button to execute the program.
5. Character sequence should appear on the output of the virtual terminal.

Expected Output



(b)

To demonstrate interfacing of seven-segment LED display and generate counting from 0 to 99 with fixed time delay.

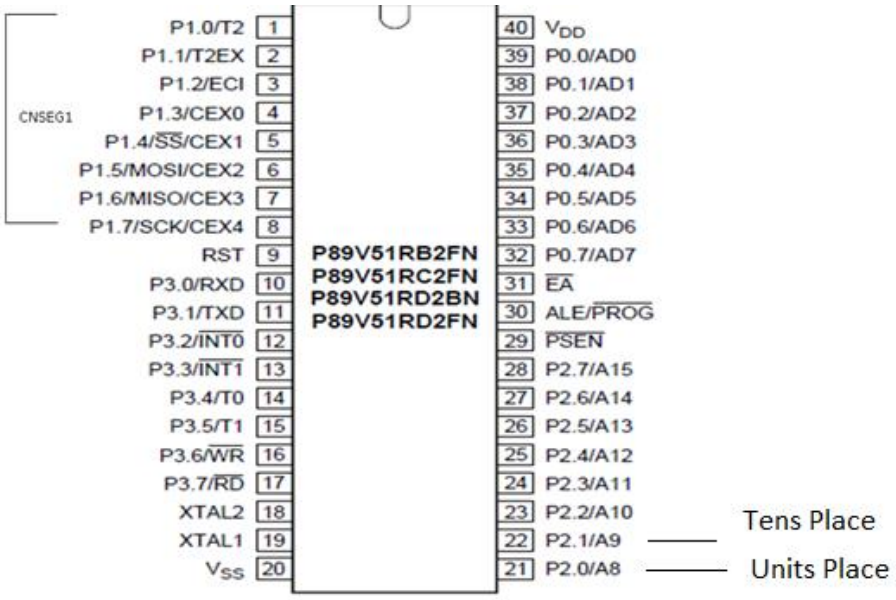
Problem Definition

Connect a seven segment display interface with port 1 of 8051 microcontroller. Set up port 1 as output port. Display the numbers 00 to 99 on the seven segment display interface, having a delay between display of two consecutive numbers. Counter should reset to 00 after it reaches 99.

Algorithm

1. Configure port 1 as output port.
2. Set P2^0 and P2^1 as select digit port pins to select the unit and tens place on seven segment display interface.
3. Set up look up table for displaying digits 0 to 9 on seven segment display interface.
4. Select the tens place display device.
5. Send the values of 0 to display.
6. Select the units place device
7. Send the value of 0 to display.
8. Repeat the value 0 at tens place.
9. Send value 1 at units place.
10. Repeat the process till count reaches 99.
11. Continue steps 4 to 10 continuously.

Hardware Connections



Actual Connection

Code

```
#include <reg51.h>
unsigned char look_up[] = {0X3f, 0X06, 0X5b, 0X4f, 0X66, 0X6d, 0X7d, 0X07,
0X7f, 0X6f, 0X77,0X7c, 0X39, 0X5e, 0X79, 0X71};
                          //display digit values stored in character array

void delay(int time);

sbit sd0 = P2^0;          //defining Port 2.0 to select digit in unit's place in 7-
                          //segment display
sbit sd1 = P2^1;          //defining Port 2.1 to select digit in ten's place in 7-
                          //segment display
void main()
{
      while(1)
      {
      unsigned char i, j;
      for(i=0;i<=9;i++)               //loop to display ten's place
      {
            for(j=0;j<=9;j++)         //loop to display one's place
```

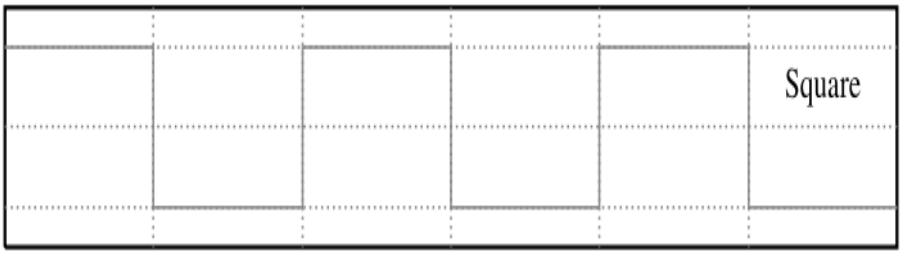
```
            {
                        sd0 = 1;
                        sd1=0;
                         P1 =look_up[i];
                        delay(500);
                        sd0 = 0;
                         sd1 = 1;
                        P1 = look_up[j];
                        delay(500);
                  }
      }
      }
}

void delay(int time)
{
      unsigned int i,j;
      for(i=0;i<100;i++)
      for(j=0;j<time;j++);
}
```

(c)

Interface 8051 with D/A converter and generate square wave of given frequency on oscilloscope.

Problem definition

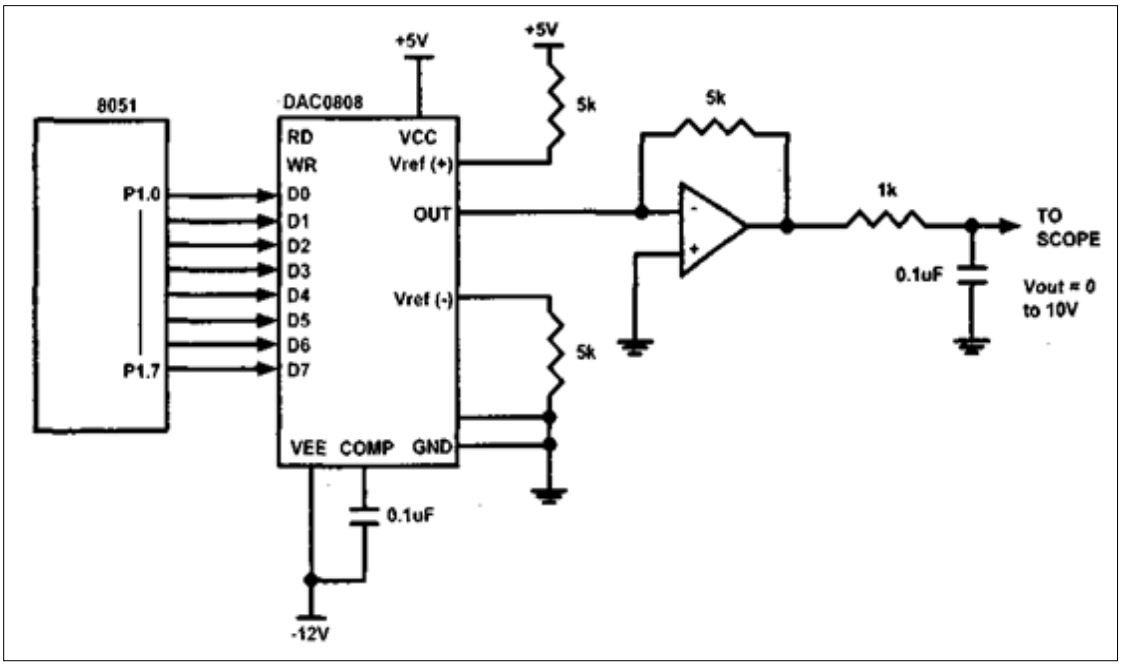
Write an embedded C program to generate a square wave of frequency 1KHz and duty cycle equal to 50% using 8051 microcontroller and a DAC. Connect the CRO through DAC interface to the output port of 8051 microcontroller to see the waveform and trace it.



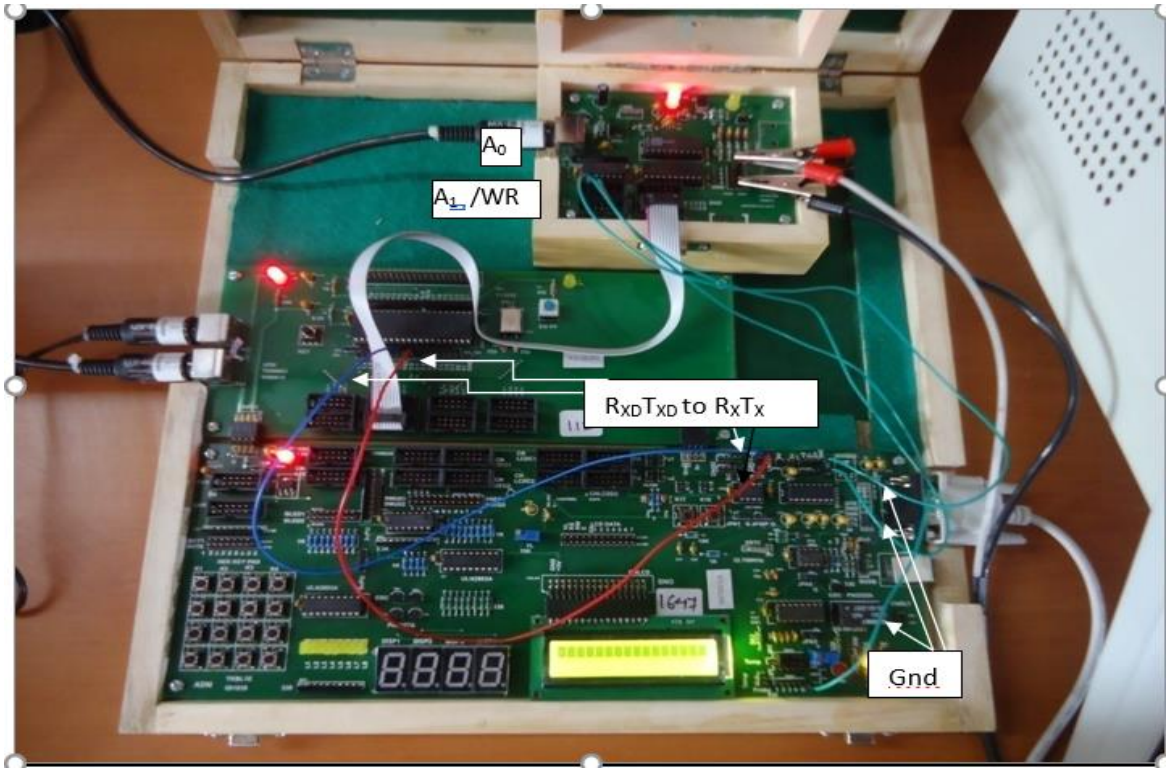
Algorithm

1. Set port 1 as output port.
2. Send maximum value at port 1
3. Call delay for calculated time interval go generate 1kHz square wave. i.e. time delay = 1/ 1000 = 1 ms
4. Send min value to port1
5. Call same delay to obtain 50% duty cycle.
6. Repeat in infinite loop.

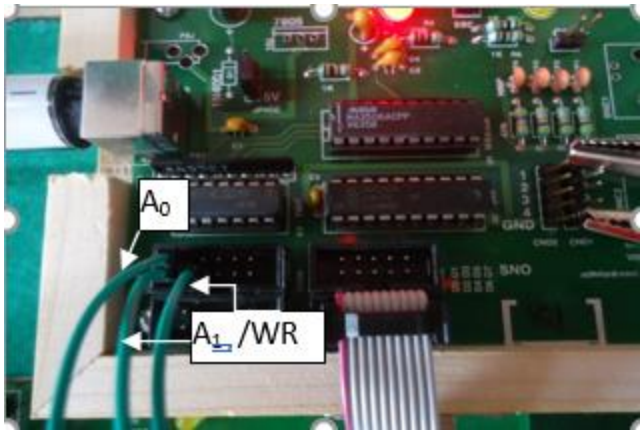
Hardware Connections



Actual Connections



Connections to Module



Code

```
#include<reg51.h>
void delay(int time);
void main()
{
	P1 = 0x00;
	while(1)
	{
		P1 = 0xFF;		// Send maximum value to Port1 for getting
					High //Period of Square Wave
		delay(1);

		P1 = 0x00;		 // Send maximum value to Port1 for getting
Low
					// Period of Square Wave
		delay(1);
	}
}
void delay(int time)			// delay of 1ms thus frequency of 1kHz.
{
	int i,j;
	for(i=0;i<=time;i++)
		for(j=0;j<=2;j++);
}
```

Expected Output

(a)

Interface 8051 with D/A converter and generate triangular wave of given frequency on oscilloscope

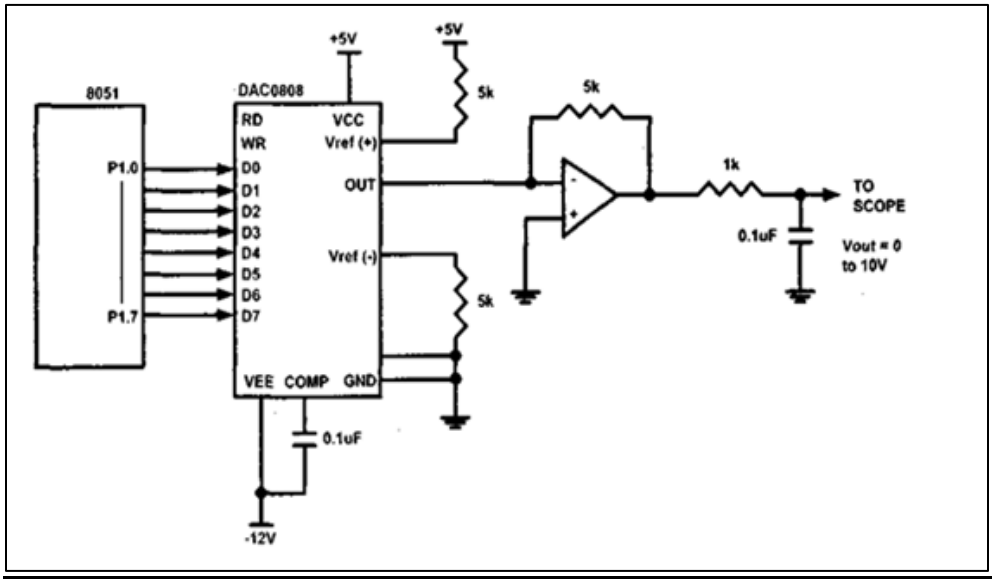
Problem Definition

Write an embedded C program to generate a triangular wave of frequency 1KHz using 8051 microcontroller and a DAC.

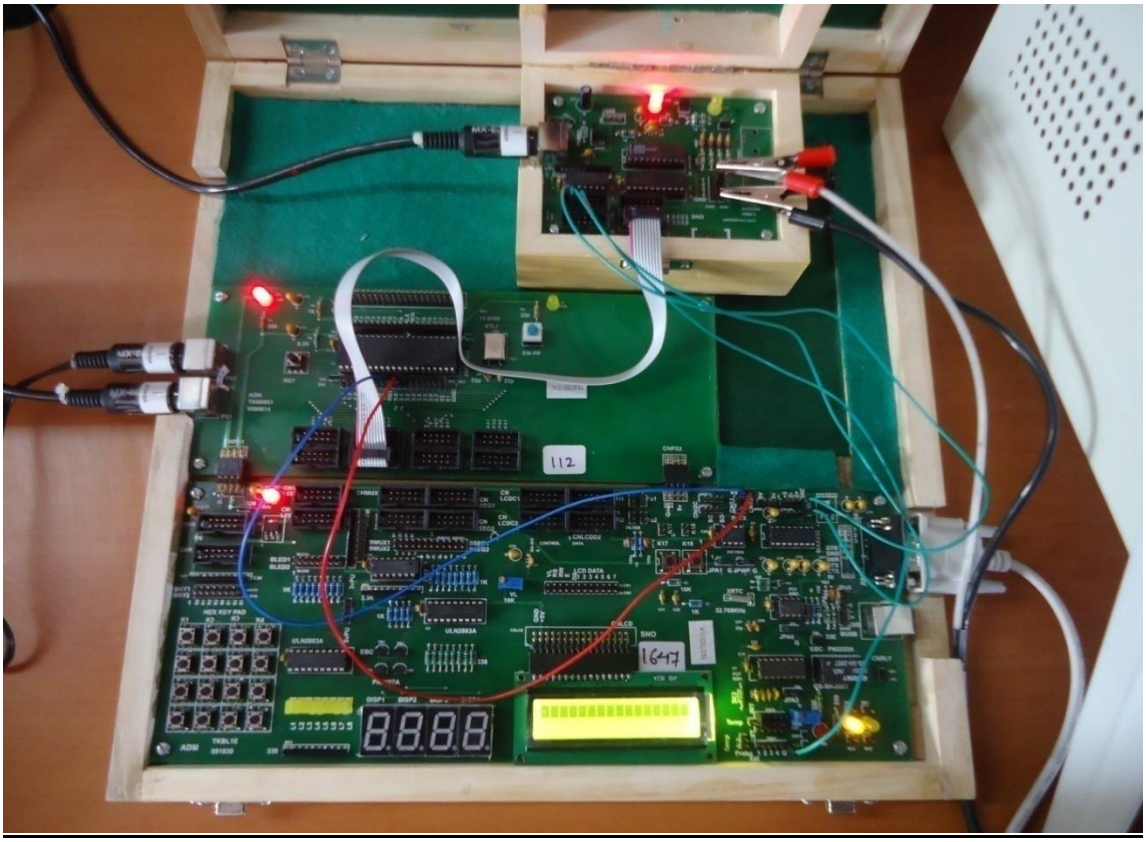
Algorithm

1. Set port 1 as output port.
2. Initialize port 1 as 00
3. Go on incrementing value till max value is reached at port1, for the upward going part of the triangular wave.
4. For 1kHz triangular wave, increase the value by 5 each time.
5. Once max value is reached, decrease the value in same steps till min value is reached.
6. Repeat in infinite loop.

Hardware Connection



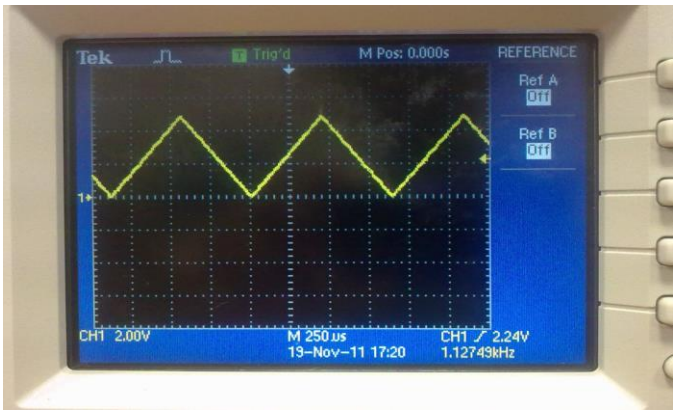
Actual Connection



Code

```
#include<reg51.h>                    //Header file for 8051
void main()                    // Start of main() function
{
P1 = 0x00;                           // Initialize Port 1 as Output
while(1)                              // Infinite Loop
{
do     // 1st do-while for upward portion of //triangular wave
{
P1 += 0x05;                          // Increment P1 to get upward portion
}
while(P1<0xFF);                      // stop loop after reaching max value
do     // 2nd do-while for downward portion of //triangular wave
{
P1 -= 0x05;                    // Decrement P1to get downward portion
}
while(P1>0x00);                      // stop loop after reaching lowest value
}
}
```

Expected Output



(b)

Using D/A converter generate sine wave on oscilloscope with the help of lookup table stored in data area of 8051.

Problem Statement

Write a program in embedded C language to generate a sine wave of 1 KHz at port1.

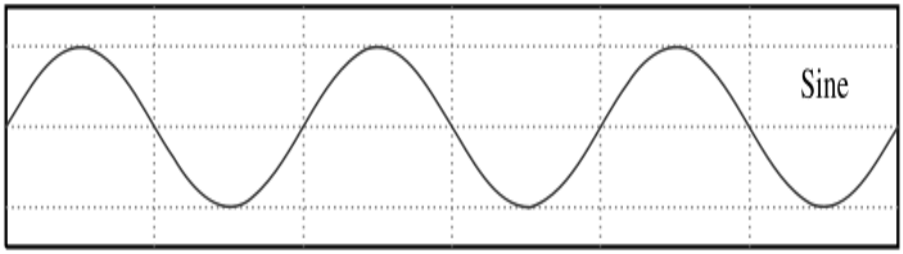
Concept

Sin wave takes values of magnitude of sine of angles between 0 and 360 degrees. Hence, sin function values vary from -1.0 to +1.0. these values along with the calculated voltage to be sent to the DAC are tabulated as under. The values show 30 degree increment for each angle. Assuming full scale voltage required for the DAC to be 10V, `Vout = 5V +(5* sin θ)`

To find values sent to DAC for various angles, we simply multiply the Vout voltage by 25.60 because there are 256 steps and full scale Vout is 10 volts . Therefore, 256 steps /10v = 25.6 steps per volt.
Vout of DAC for various angles is calculated and shown in table

Angle θ(Degrees)	Sin θ	Vout(Voltage magnitude) 5v + (5 v * sin θ)	Values sent to DAC (Decimal) (voltage Mag. * 25.6)
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

Lookup table for all values for generating sin wave



Desired Sin wave

Algorithm

7. Generate lookup table for all values given in above table.
8. Send these values one by one to the DAC.
9. Introduce appropriate amount of delay after every value is sent to DAC.
10. Repeat this indefinitely.

Hardware Connection

Same as 4 (c) and 5 (a)

Actual Connection

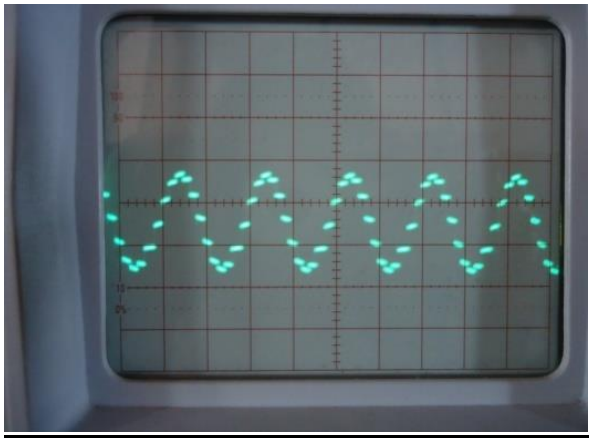
Same as 4 (c) and 5 (a)

Code

```
#include<reg51.h>
#include<intrins.h>                 //Header file for NOP function
void main()
{
        int WAVEVALUE[12] =
             { 128,192,238,255,238,192,128,64,17,0,17,64};

                                    // Create a look-up table (array) named
                                    //WAVEVALUE to send appropriate values to
                                    //DAC so that a sine-wave is generated.
        int i;
        while(1)
        {
               for(i=0;i<12;i++)
               {
                    P1 = WAVEVALUE[i];
                  _nop_();
                  _nop_();          // _nop_() function introduces delay.
                  _nop_();
```

```
                _nop_();            // Repeated use of _nop_() generates
                                    //wave of desired frequency
                _nop_0();
               _nop_();
               }
        }
    }
```

Expected Output



Practical 6

Interface stepper motor with 8051 and write a program to move the motor through a given angle in clock wise or counter clock wise direction

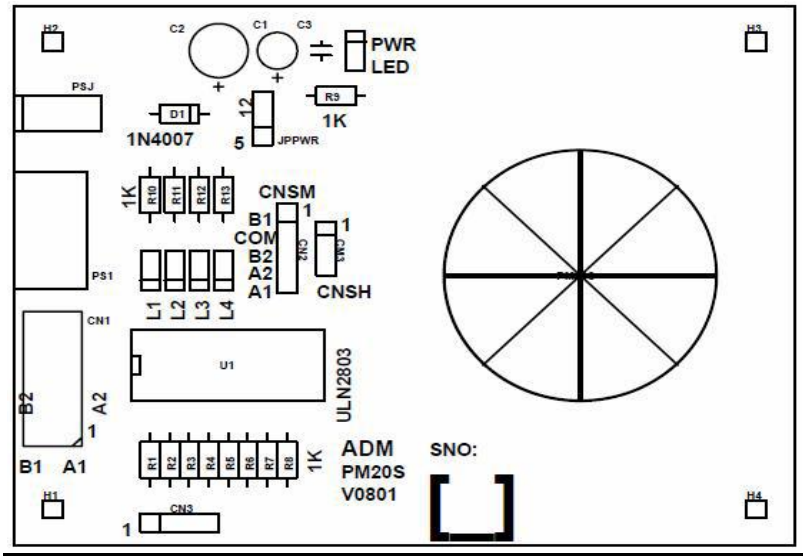
Problem Definition

Write a program in embedded C language to rotate the stepper motor in clockwise direction 10 times.

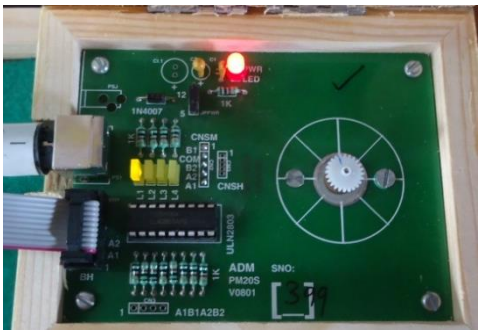
Algorithm

1. Initialize port 1 as output port
2. Fetch the address of internal data memory in a memory pointer.
3. If the data is 00 (initial position) , FF (Final position) or 01 (intermediate position), store the number 08H (1000)2 at port1. This will set the motor in first position. (as shown in Fig 7.1)
4. Else, go to the next position. This is achieved by rotating the number at port1 to right for clockwise (and to left for anticlockwise).
5. After every movement of motor, call delay.
6. Do this for 10 rotations.
7. Store the port data at the memory pointer.
8. Repeat infinitely.

Hardware Connection



Actual Connection



Stepper motor module

Actual Connection of Stepper Motor with TKBase51

Code

```
#include <reg51.h>
unsigned char j,k;
unsigned char *_data p = 0x56              //Pointer points to internal data memory
void delay (unsigned int);                 //Delay function prototype
void main ()
{
P1=*p;       //Assigns P1 with last P1 value stored in //internal data
for (j=0;j<2;j++)                          //For loop1
{
for (k=0;k<10;k++)                                 //For loop2
{
if(P1 == 0x01 || P1 == 0x00 || P1 == 0xff) //If P1=0x01or 0x00 or 0xff
{
P1 = 0x08;                                 //Assign 0x08 value to P1
delay(3000);
}
else
{
P1 = P1>>1;                                        //Rotate P1's value right by 1
bit
delay(3000);                                       //Call delay function
```

```
}
}
*p=P1;                                  //Stores P1 value in idata
delay (25000);                          //Call delay function
}
while (1);                              //Always true condition, jump here
}                                       //End of main program
void delay (unsigned int a)
{                                        //Delay function starts here
unsigned int i;                         //Variable
for (i=0;i<=a;i++);                     //For loop for delay
}                                       //End of delay function
```

Expected Ouput

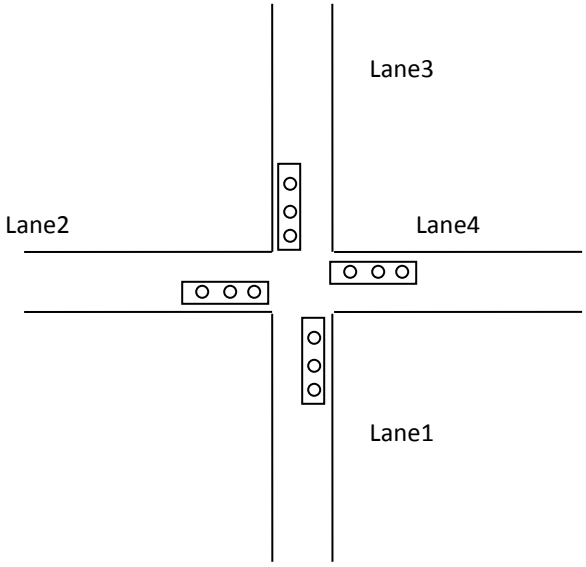
Same as Actual Connections with rotating motor.

Practical 7

Generate traffic signal.

Problem Definition

Following dig shows traffic lights on an interface road. There are four lanes and each lane has a traffic signal with Red, Orange and Green light. Lane1 and lane2 should work simultaneously and Lane3 and Lane4 should work simultaneously. Write a program in embedded C language to control a traffic light on an interface.



Write Embedded C program to glow the traffic lights in following sequence
Lane1 Green – Orange – Red
Lane2 Green – Orange – Red
Lane3 Green – Orange – Red
Lane4 Green – Orange – Red

Algorithm

1. Define constants for all LEDs (assuming Lane1 and Lane2 are needed simultaneously, lane1 and lane3 share same constant and lane2 and lane3 share same constants.)
2. Use a bit variables for lane1, lane2, lane3 and lane4. (lane1 and lane2 connected P1.6 and lane3 and lane4 are connected to P1.7)
3. Show Traffic light control of lane1 and lane2.
   i) Send data associated with Green LED of lane 1 to output port port1.
   ii) Latch this data at port1.

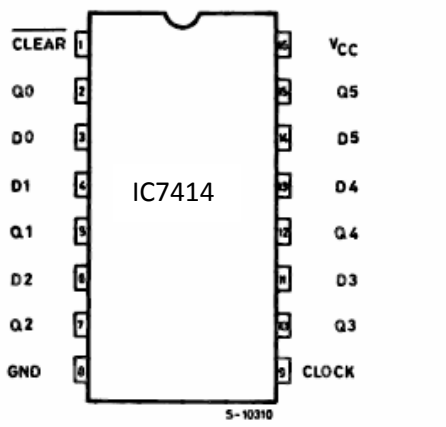
iii) Send data associated with Orange LED of lane1 to output port1.
iv) Latch this data at port1.
v) Send data associated with Red LED of lane 1 to output port port1.
vi) Latch this data at port1.
vii) Send data associated with Green LED of lane 2 to output port port1.
viii) Latch this data at port1.
ix) Send data associated with Orange LED of lane 2 to output port port1.
x) Latch this data at port1.
xi) Send data associated with Red LED of lane 2 to output port port1.
xii) Latch this data at port1.

4. Show traffic light control of lane3 and lane4.
   i) Send data associated with Green LED of lane3 to output port port1.
   ii) Latch this data at port1.
   iii) Send data associated with Orange LED of lane3 to output port1.
   iv) Latch this data at port1.
   v) Send data associated with Red LED of lane3 to output port port1.
   vi) Latch this data at port1.
   vii) Send data associated with Green LED of lane4 to output port port1.
   viii) Latch this data at port1.
   ix) Send data associated with Orange LED of lane4 to output port port1.
   x) Latch this data at port1.
   xi) Send data associated with Red LED of lane4 to output port port1.
   xii) Latch this data at port1.

5. Repeat in infinite loop.

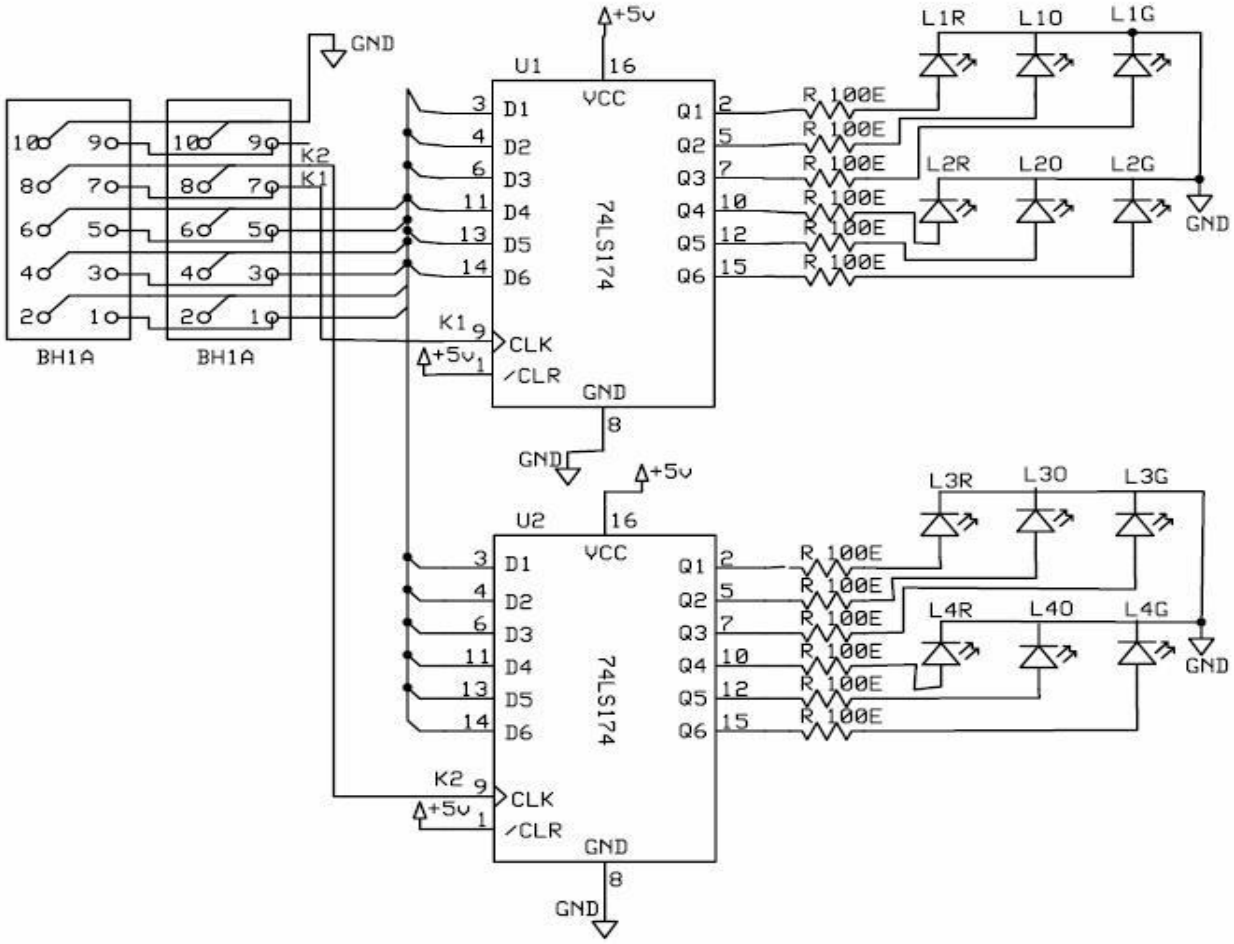
Hardware Connection

The system utilizes a latch, IC74147 for latching the data to be sent to the LEDs. Two ICs are used each to control two lanes. pin configuration of IC74147 is shown below –



Pin configuration of IC74147

Connect the power supply to the traffic light control module and connect the module to port 1 of the microcontroller using the FRC cable.



Schematic dig of traffic light module.

Actual Connection

Actual connection of TK base 51 kit with the stepper motor module.

Code

```
#include<reg51.h>

#define Lane1Red 0x01          // Define a labels for each combination of colour
                               //and lane. 4 lanes and 3 colors- 12 labels
#define Lane1Orange 0x02
#define Lane1Green 0x04
#define Lane2Red 0x08
#define Lane2Orange 0x10
#define Lane2Green 0x20
#define Lane3Red 0x01
#define Lane3Orange 0x02
#define Lane3Green 0x04
#define Lane4Red 0x08
#define Lane4Orange 0x10
#define Lane4Green 0x20
sbit Lane1Set = P1^6;          // Here the label 'Lane1Set' is used to refer pin
                               //PORT1.6

sbit Lane2Set = P1^6;
sbit Lane3Set = P1^7;          // When the Value is sent to Port1 it needs to be
                               //latched
```

```
sbit Lane4Set = P1^7;          // Each Lane has its own port pin to latch the data
                               // Latching of data is done by sending a Low to
                               //High signal on the respective Lane pins
void Reset();
void showLane12();
void showLane34();
void SetLane1();
void SetLane2();
void SetLane3();
void SetLane4();
void delay(int time);

void main()
{
   Reset();                    // Reset the Traffic Light Kit (clear previous data)

   while(1)
  {
     showLane12 ();            // Display Traffic lights of Lane 1,Lane 2
     Reset ();
     showLane34 ();            // Display Traffic lights of Lane 3,Lane 4
     Reset ();
   }
}
void Reset()
{
   P1 = 0x00;                        // Turn off all LEDs
   delay (100);
   Lane1Set = Lane2Set = 1;          // Latch data 0x00 to lane 1 & lane 2
   Lane3Set = Lane4Set = 1;          // Latch data 0x00 to lane 3 & lane 4
}

void showLane12()              // this function will turn on LEDs of Lane 1 and 2
{
  P1 = Lane1Green;             // send data to Port1 to turn on Green LED of
                               //Lane1
  SetLane1();                  // Latch data into the kit.Green LED of Lane1 will
                               //now turn on
  P1 = Lane1Orange;
  SetLane1();
  P1 = Lane1Red;
  SetLane1();
  P1 = Lane2Green;
  SetLane2();
  P1 = Lane2Orange;
  SetLane2();
  P1 = Lane2Red;
```

```
  SetLane2();
  }

void showLane34()          // this function will turn on LEDs of Lane3 & 4 one
                           //by one
{
  P1 = Lane3Green;                 // send data to Port1 to turn Green LED of L3
  SetLane3();              // Latch data into the kit. Green LED of Lane 3 will
                           //now turn on
  P1 = Lane3Orange;         // Repeat the steps for all the LEDs on Lane 3 & 4
  SetLane3();
  P1 = Lane3Red;
  SetLane3();
  P1 = Lane4Green;
  SetLane4();
  P1 = Lane4Orange;
  SetLane4();
  P1 = Lane4Red;
  SetLane4();
}

void SetLane1()            // This function Latches the data to Lane 1 of the
                           //kit which was sent on Port1
{
  Lane1Set = 0;            // 'Lane1Set' i.e pin P1.6 is set to Low
  Lane1Set = 1;            // 'Lane1Set' i.e pin P1.6 is set to High
  delay(2000);                     // A low to high signal will latch the data
}

void SetLane2()            // This function Latches the data to Lane2 of the kit
                           //which was sent on Port1
{
  Lane2Set = 0;            // 'Lane2Set' i.e pin P1.6 is set to Low
  Lane2Set = 1;            // 'Lane2Set' i.e pin P1.6 is set to High
  delay(2000);                     // A low to high signal will latch the data
}

void SetLane3()            // This function Latches the data to Lane3 of the kit
                           //which was sent on Port1
{
 Lane3Set = 0;             // 'Lane3Set' i.e pin P1.7 is set to Low
 Lane3Set = 1;             // 'Lane3Set' i.e pin P1.7 is set to High
 delay(2000);                      // A low to high signal will latch the data
}
```

```
void SetLane4()                 // This function Latches the data to Lane4 of the kit
                                //which was sent on Port1
{
 Lane4Set = 0;                  // 'Lane4Set' i.e pin P1.7 is set to Low
 Lane4Set = 1;                  // 'Lane4Set' i.e pin P1.7 is set to High
 delay(2000);                           // A low to high signal will latch the data
}

void delay(int time)
{
  int i,j;
  for(i=0;i<=time;i++)
    for(j=0;j<=23;j++);
 }
```

Expected Ouput

Same as actual connection with all lanes having leds turned on in the sequence green orange and then red with given delay

Practical 8

Implement temperature control.

Problem Definition

Write an embedded C program to detect temperature in room. If the temperature of the room goes above 50, LED connected to output port should glow. And when the room temperature falls below 50 degrees, LED should turn back OFF.

Algorithm

1. Set port1 as input port.
2. Set Port2 as output port.
3. If the temperature read at P1 is more than the absolute value (50), then set P2.0 to 1
4. Call delay
5. Else reset bit P2.0
6. Call delay

Hardware Connections

Hardware module used for the application is DT35. This device uses the semiconductor temperature sensor, LM335. LM335 series are precision, calibrated and integrated temperature sensors. Operating as two terminal zener, it has a breakdown voltage directly proportional to the absolute temperature at +10mv/^{o}K. default output of the device at room temperature 28^{o}C is approximately. (82.3 ^{o}F considered as 83H)

Output of temperature sensor is a analog value and requires to be converted into digital form. This is done with the help of ADC. TKADC084 is 8 bit microprocessor compatible ADC. Simple interface to TKADC084 is provided with 10 pin box connectors BH2A and BH2B

D1	D3	D5	D7	GND
D0	D2	D4	D6	NC

BH1A/B

WR	NC	NC	NC	GND
RD	CS	NC	NC	NC

BH2A/B

Box connectors

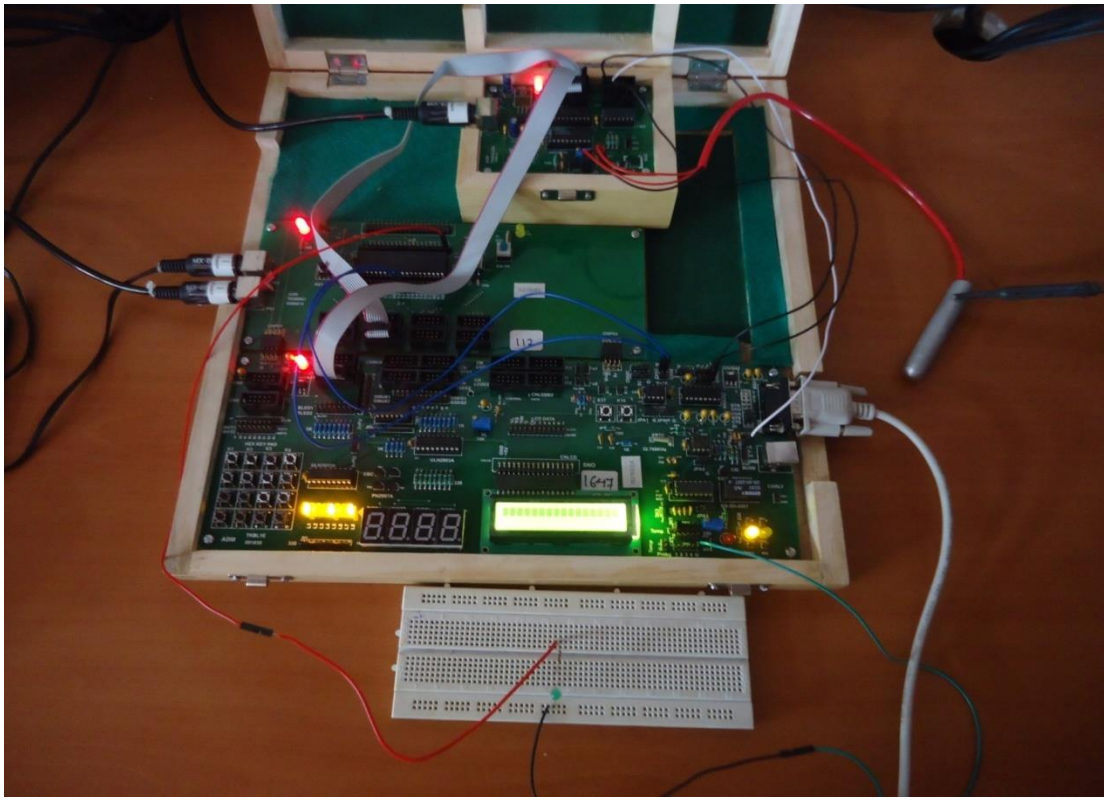
1. Connect BH1A to the LED interface on TKBase51. This will read the temperature in binary form on the LED interface.
2. To enable the ADC, connect RD and CS pins of BH2A connector to GND on the TKBase51
3. Connect the WR pin to +5V.
4. Connect ADC (connector BH1B) to input port of 8051 i.e. port1.

5. Connections of LM335.

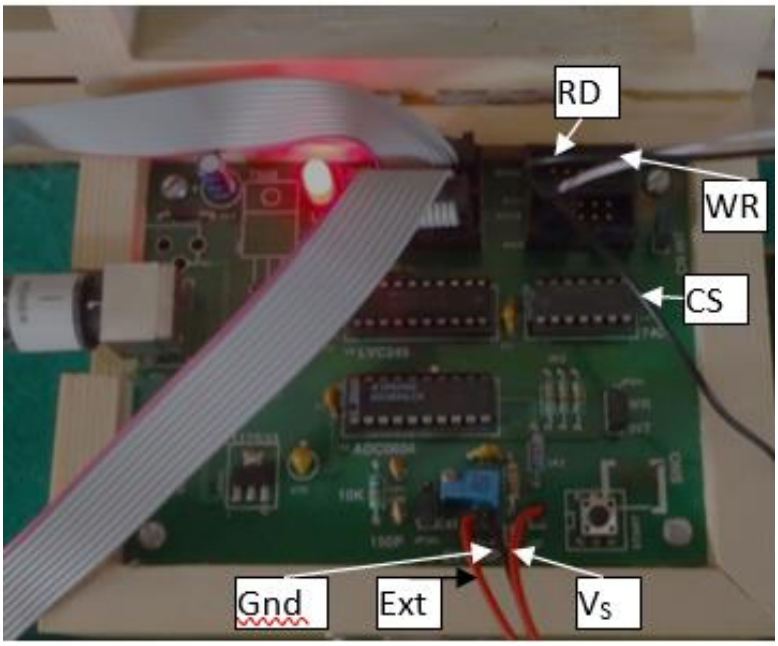
Bottom View of LM335.

   a. Connect the LM335 to the ADC by connecting $+V_S$ pin to $+V_{CC}$ on the ADC.
   b. Connect V_{OUT} of LM335 to Ext Input of ADC
   c. Connect GND pin of LM335 to GND of ADC.
6. Connect an external LED with pullup resistance setup on a bread board and connect the pin P2.0 to this external setup.

Actual Connections



Components: TK Base 51, ADC module, Temperature Sensor LM335, on board LED interface and External LED



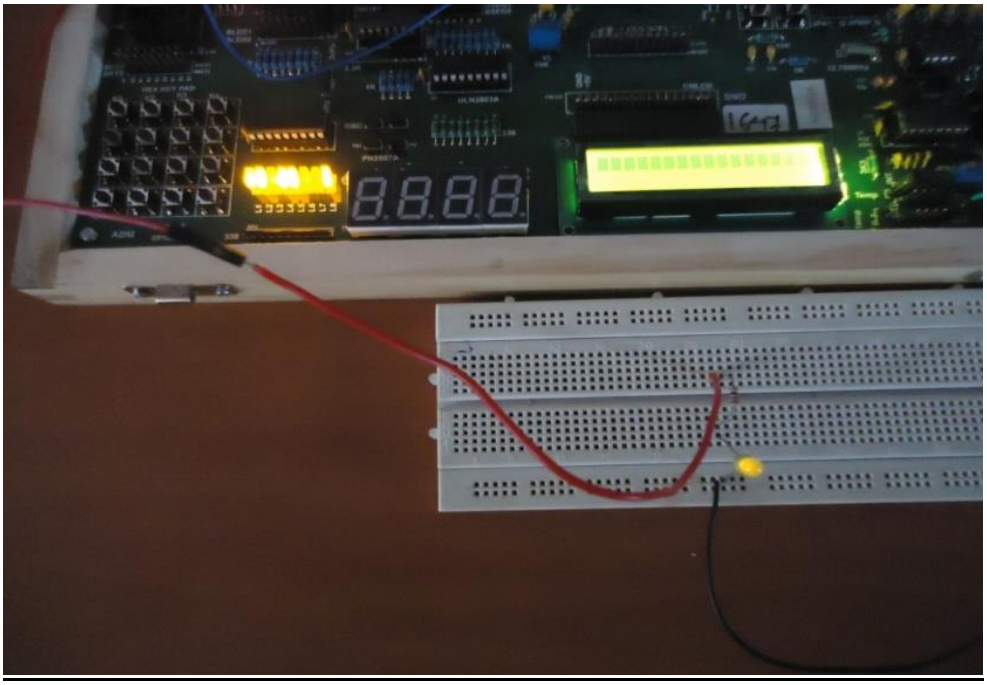
Connections for ADC Module

Code

```
#include <reg51.h>

void delay (void);
sbit IN = P2^0;                 //Defining P2.0 bit as an IN which is
                                //input pin to LED on bread board
void main (void)
{
 P1=0xFF;                       //P1 acts as an input port, reads the value of  ADC
 IN =0;                         // Turn OFF the external LED
 while (1)
 {
  if (P1>0x99)                  //if ADC value read by P1 exceeding the said
                                //temperature then turn ON the external LED

  {
    IN=1;
    delay ();
   }
  else
  IN=0;                         //if ADC value is below absolute value then turn
                                OFF //external LED

  }
 }
```

```
void delay (void)
{
  unsigned int i;
    for (i=0;i<=600;i++);                    //This for loop generates delay
}
```

Expected Output



Value of temperature is 9A H and external LED is turned ON

Practical 9

Implement Elevator control.

Problem Definition

Write a program in embedded C language to stimulate an elevator control system using a stepper motor. System includes a keypad to key in the floor, a seven segment display to show the current position of the elevator and a stepper motor which would act as the elevator itself.

Algorithm

1. Create a lookup table for displaying digit values.
2. Create a char array for generating motor sequence for a 4 axis stepper motor. i.e. 8,4,2,1 which will magnetize each axis.
3. Initialize the keyboard port as F0 to make all column values high (1111) and all row values low (0000).
4. Read the value of key pressed. (key scan function)
5. Call the display function for the pressed key value. (get key function)
6. Repeat in infinite loop.

Key scan function

1. Save the value of already pressed key to note the current position of the elevator.
2. Check if any key is pressed in each row, this is done by grounding the respective row. For eg F0 in port 0 has set all columns to high and all rows to ground, to check if row 1 has any key pressed, ground row1 and set all other rows to high. (P0 = 0xFE). For second row, P0 = 0xFD and so on.
3. If the key is pressed, newkey value is obtained from P0.
4. Reset the value of P0 to 0xF0 to read the next key.

Get key function

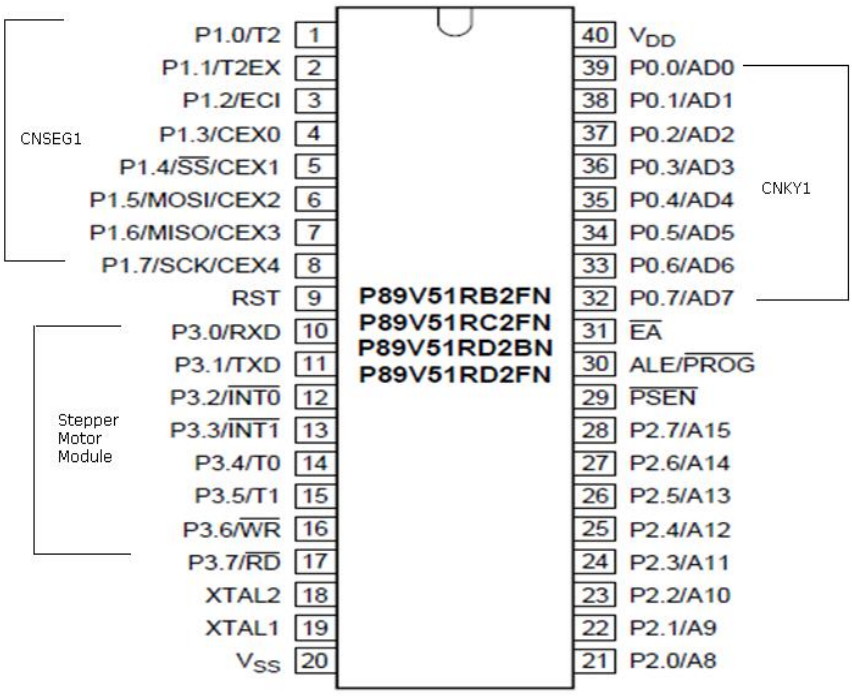
1. If newkey is EE, last column and last row is selected, call display function for 0.
2. Similarly, call display function for different digits.

Display Function

1. Get the difference between the newly pressed key and old key in tmp variable. This will give the direction of stepper motor (negative difference indicates anticlockwise movement so that the lift moves downwards and positive difference indicates clockwise movement so that lift moves upwards)

2. Get the value for display from the display lookup table and send it to the port1 where the 7 segment display is connected.
3. Get the value for motor sequence and send it to the stepper motor, which is connected at port3.
4. Call the delay.
5. Repeat this for 'tmp' times so that the elevator moves to the correct position.

Hardware Connection



Required connections on the I / O ports

Actual Connection

Components Hex Keyboard, Seven Segment Display and Stepper Motor

Code

```
#include <reg51.h>
unsigned char newkey,oldkey,p;          //newkey variable is for newly pressed
                                        //key; oldkey variable is to store the
                                        //previous pressed key;
                                        //p variable acts as a pointer.
signed char tmp;                        //tmp variable is to store the difference
                                        //between oldkey and newkey pressed
sbit sd = P2^0;                         //Defining P2.0 bit as an sd;

unsigned char look_up[]={0x3f,0x06,0x5b,0x4f,0x66,
                         0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};

unsigned char motor_sequence[]={8,4,2,1,8,4,2,1,8,4,2,1,8,4,2,1};

void key_scan (void);                   //Scan keyboard and check whether the key
                                        //is pressed?
void get_key (void);                    //Get the pressed key code
void display (unsigned char);           /Display pressed key and also rotate motor
void delay (unsigned int time);

void main (void)
{
      P0=0xf0;                          //Initializing keyboard (columns =high(1111)
                                        //and rows = low(0000))
```

```
	while (1)
	{
		key_scan ();
		get_key ();
	}
}


void key_scan (void)
{
	oldkey=newkey;			//stores previously pressed key in oldkey
	while (1)
	{
		P0=0xFE;		//Make 1st row ground
		P0 &=0xFE;		//Check whether key is pressed in 1st row?
		if (P0!=0xFE)		//if yes then
			break;		//Come out of loop, else

		P0=0xFD;		//Make 2nd row ground
		P0 &=0xFD;		//Check whether key is pressed in 1st row?
		if (P0!=0xFD)		//if yes then
			break;		//Come out of loop, else
		P0=0xFB;		//Make 3rd row ground
		P0 &=0xFB;		//Check whether key is pressed in 1st row?
		if (P0!=0xFD)		//if yes then
			break;		//Come out of loop, else

		P0=0xF7;		//Make 4th row ground
		P0 &=0xF7;		//Check whether key is pressed in 1st row?
		if (P0!=0xF7)		//if yes then
			break;		//Come out of loop, else

	}
	newkey=P0;			//Store keycode in newkey variable
	do {
		P0=0xF0;		//Again initialize P0 to its default value this
					//will allow user to press next key
		 }
		while (P0!=0xf0);
    }


void get_key (void)
{
	if (newkey==0xee)
		display (0);
	else if (newkey==0xed)
		display (1);
```

```
        else if (newkey==0xeb)
                display (2);
        else if (newkey==0xe7)
                display (3);
        else if (newkey==0xde)
                display (4);
        else if (newkey==0xdd)
                display (5);
        else if (newkey==0xdb)
                display (6);
        else if (newkey==0xd7)
                display (7);
        else if (newkey==0xbe)
                display (8);
        else if (newkey==0xbd)
                display (9);
        else if (newkey==0xbb)
                display (10);
        else if (newkey==0xb7)
                display (11);
        else if (newkey==0x7e)
                display (12);
        else if (newkey==0x7d)
                display (13);
        else if (newkey==0x7b)
                display(14);
        else if (newkey==0x77)
                display(15);
 }

void display (unsigned char x)
{
        newkey=x;
        tmp=newkey-oldkey;              //Take the difference between newkey and
                                        //previously pressed key
        sd=0;                           //Select the display
        if (tmp<0x00)                   //if tmp value is less than zero it
                                        //indicates that the newkey pressed is
                                        //smaller than the previous key pressed
        {
         for (p=oldkey;p>newkey;p--)
         {
                P3=motor_sequence[p];   //motor will rotate in steps
                                        //depends on tmp value, anti-clockwise
                P1=look_up[p];          //displays value on seven segment
                delay (10000);
                }
```

```
            P3=motor_sequence[p];    //motor will rotate once in aticlckwise
            P1=look_up[p];
            delay (10000);
      }
      else
      {
      for (p=oldkey;p<newkey;p++)
      {
            P3=motor_sequence[p];          //motor will rotate in steps depends
            on
                                           //tmp value, clockwise
                  P1=look_up[p];           //displays value on seven segment
                  delay (10000);           //software delay
            }
            P3=motor_sequence[p];   //motor will rotate once, clockwise
            P1=look_up[p];
            delay (10000);
      }
}

void delay (unsigned int time)
{
      unsigned int i;
      for (i=0;i<=time;i++);
}
```

Expected output

When a key on hex keypad is pressed, motor moves correspondingly. Also, the key pressed is the floor where elevator is going to and is displayed on the seven segment display.

Practical 10

(a)

To demonstrate the procedure for flash programming for reprogrammable embedded system board using Flash Magic

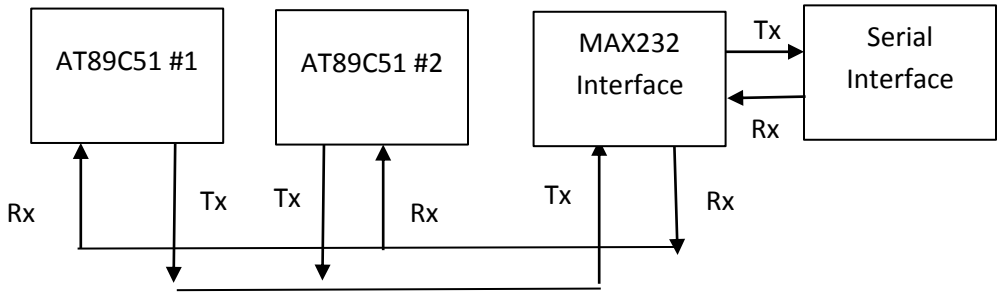
Discussed earlier in Practical 1.

(b)

To demonstrate the procedure and connections for multiple controllers programming of same type of controller with same source code in one go, using flash magic.

Steps

Connect multiple controllers of same types to the host PC through MAX232 interface. The connection can be done using Rx and Tx pins as shown in figure below. –



Block dig of multiple controller programming using flash magic

Start Flash magic software on the host PC. Select the Advanced Options menu.
Check the option – Disable device signature option.
Now use flash magic as discussed in practical 1.
Both controllers will get simultaneously programmed.