FIRE BIRD V

P89V51RD2
ROBOTIC RESEARCH PLATFORM
Software Manual

Designed By:

ERTS Lab, CSE, IIT Bombay
www.it.iitb.ac.in/~erts

NEX ROBOTICS
www.nex-robotics.com

Manufactured By: NEX Robotics Pvt. Ltd.

FIRE BIRD V

SOFTWARE MANUAL

Version 1.05
January 7, 2010

Documentation author
Sachitanand Malewar, NEX Robotics Pvt. Ltd.
Omkar Pradhan, NEX Robotics Pvt. Ltd.

Credits (Alphabetically)
Aditya Sharma, NEX Robotics
Amey Apte, NEX Robotics
Anant Malewar, EE, M.Tech, IIT Bombay
Ashish Gudhe, CSE, M.Tech, IIT Bombay
Behlul Sutarwala, NEX Robotics
Fawaz Mukar**i,** NEX Robotics
Gurulingesh R. CSE, M.Tech, IIT Bombay
Inderpreet Arora, EE, M.Tech, IIT Bombay
Prof. Kavi Arya, CSE, IIT Bombay
Prof. Krithi Ramamritham, CSE, IIT Bombay
Nandan Salunke, RA, CSE, IIT Bombay
Pratim Patil, NEX Robotics
Preeti Malik, RA, CSE, IIT Bombay
Prakhar Goyal, CSE, M.Tech, IIT Bombay
Raviraj Bhatane, RA, CSE, IIT Bombay
Rohit Chauhan, NEX Robotics
Rajanikant Sawant, NEX Robotics
Saurabh Bengali, RA, CSE, IIT Bombay
Vaibhav Daghe, RA, CSE, IIT Bombay
Vibhooti Verma, CSE, M.Tech, IIT Bombay

Notice

ATTENTION
OBSERVE PRECAUTIONS FOR HANDLING
ELECTROSTATIC SENSITIVE DEVICES

⚠ **Robot's electronics is static sensitive. Use robot in static free environment.**
⚠ **Read the hardware and software manual completely before start using this robot**

Recycling:

Almost all of the robot parts are recyclable. Please send the robot parts to the recycling plant after its operational life. By recycling we can contribute to cleaner and healthier environment for the future generations.

Index

1. Introduction to Fire Bird V P89V51RD2

Fire Bird V is designed to give you good exposure to the world of robotics and embedded systems. Thanks to its innovative architecture and adoption of the 'Open Source Philosophy' in its software and hardware design, you will be able to create and contribute to, complex applications that run on this platform, helping you acquire expertise as you spend more time with them.

Origins of the Fire Bird V

The Fire Bird V robot is the 5th in the Fire Bird series of robots. First two versions of the robots were designed for the Embedded Real-Time Systems Lab of Department of Computer Science and Engineering, IIT Bombay. Theses platforms were made commercially available form the version 3 onwards.

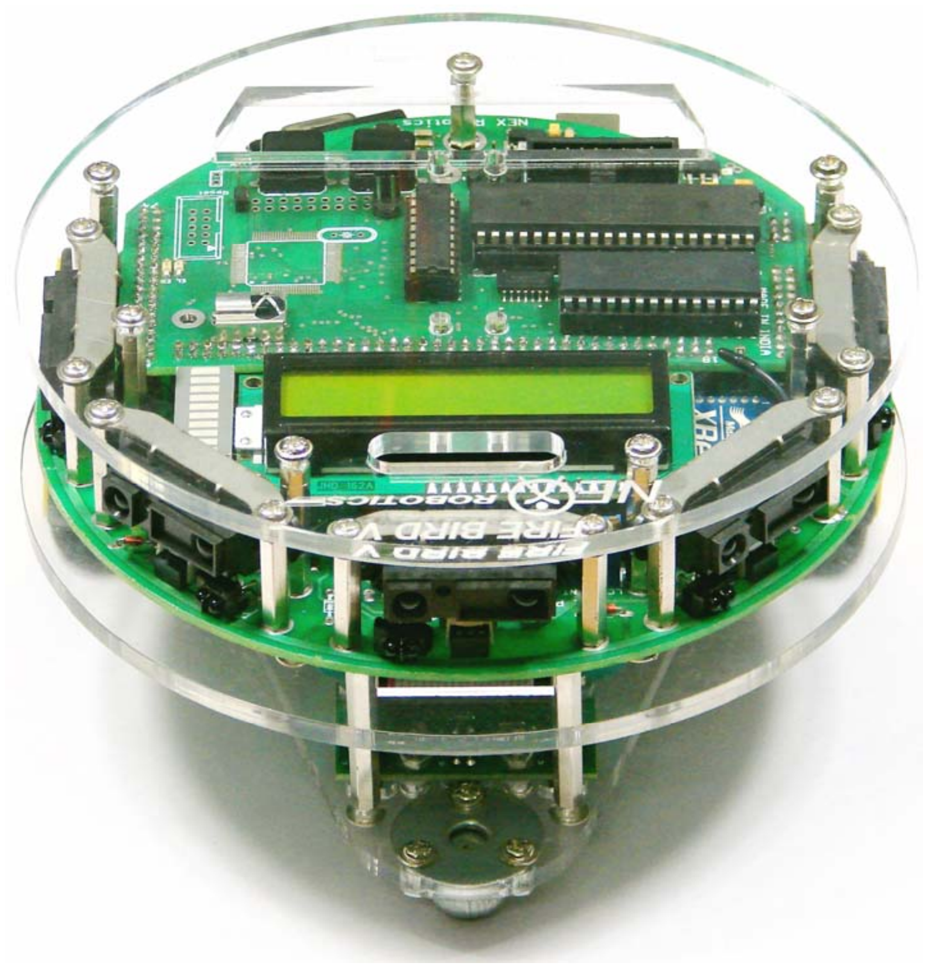


Figure 1.1: Fire Bird V P89V51RD2 robot

1.1 Fire Bird V Block Diagram:

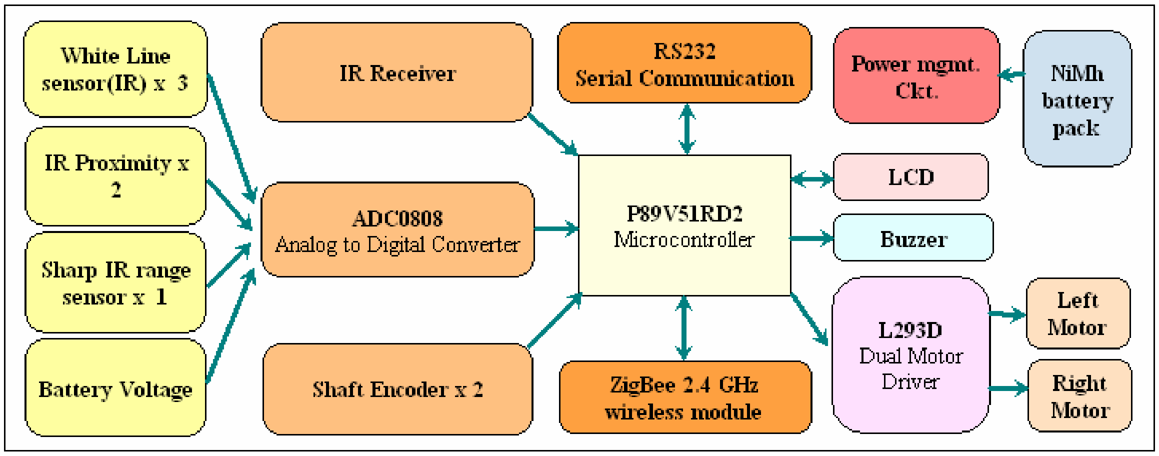

White Line sensor(IR) x 3
IR Proximity x 2
Sharp IR range sensor x 1
Battery Voltage
IR Receiver
ADC0808 Analog to Digital Converter
Shaft Encoder x 2
RS232 Serial Communication
P89V51RD2 Microcontroller
ZigBee 2.4 GHz wireless module
Power mgmt. Ckt.
NiMh battery pack
LCD
Buzzer
L293D Dual Motor Driver
Left Motor
Right Motor

Figure 1.2: Fire Bird V P89V51RD2 robot block diagram

1.2 Fire Bird V P89V51RD2technical specification

Microcontroller:
NXP P89V51RD2 (8051 architecture based Microcontroller)

Sensors:

Three white line sensors
One Sharp GP2D12C Infrared Range sensor with 80cm range
Three analog IR proximity sensors (TCRT5000)
Two position encoders (shaft encoders)
Battery voltage sensing

Indicators:

2 x 16 Characters LCD
Indicator LEDs
Buzzer

Control:

Autonomous Control
PC as Master and Robot as Slave in wired or wireless mode

Communication:

Wired RS232 (serial) communication
Simplex infrared communication (from infrared remote to robot)
Wireless ZigBee Communication (2.4GHZ)

Dimensions:

Diameter: 16 cm
Height: 10 cm
Weight: 1250 Gms.

Power:

12V, 2.1 Ah Nickel Metal Hydride (NiMh) battery pack.or Auxiliary power (external power) using battery charger.

Battery Life:

4 Hours while motors are operational at 75% of time

Locomotion:

Two DC geared motors in differential drive configuration and caster wheel at front as support

- Top Speed: 24 cm / second
- Wheel Diameter: 52mm
- Position encoder: 30 pulses per revolution
- Position encoder resolution: 5.44 mm

1.3 Pin Functionality

P89V51RD2 pin configuration

PIN No	Pin Function	Used For
1	T2/P1.0	Left Motor Direction Control(C)
2	T2EX/P1.1	Left Motor Direction Control(D)
3	ECI/P1.2	Right Motor Direction Control(B)
4	CEX0/P1.3	Pulse width modulation for the left motor (velocity control)
5	CEX1/SS/P1.4	Pulse width modulation for the right motor (velocity control)/SPI SS
6	MOSI/P1.5	SPI MOSI
7	MISO/P1.6	SPI MISO
8	SCK/P1.7	SPI SCK
9	RST	Microcontroller reset
10	RXD/P3.0	RS 232 serial communication
11	TXD/ P3.1	RS 232 serial communication
12	INT0 / P3.2	ADC End of Conversion (ADC EOC)
13	INT1 / P3.3	Position encoder left motor/ INT SW/ TV Remote
14	T0/P3.4	Right Motor Direction Control(A)
15	T1/ P3.5	Position encoder right motor
16	WR / P3.6	ALE for ADC
17	RD/P3.7	Output Enable for ADC
18	XTAL2	11.0592 MHz Crystal
19	XTAL1	
20	Vss	Ground
21	P2.0/A8	LCD Data Line DB4
22	P2.1/A9	LCD Data Line DB5
23	P2.2/A10	LCD Data Line DB6
24	P2.3/A11	LCD Data Line DB7
25	P2.4/A12	LCD Enable
26	P2.5/A13	LCD R/W
27	P2.6/A14	LCD RS
28	P2.7/A15	Buzzer
29	PSEN	NC
30	ALE/-PROG	Clock source for ADC0808 analog to digital converter
31	EA	Vcc (5V)
32	P0.7	ADC output data Lines (D7 to D0)
33	P0.6	
34	P0.5	
35	P0.4	
36	P0.3	
37	P0.2	
38	P0.1	
39	P0.0	
40	Vcc	Vcc (5V)

Table 1.1 P89V51RD2 microcontroller pin functionality

Microcontroller	ADC 0808
Pin 12 - INT0 / P3.2	Pin 7 - EOC (End of Conversion)
Pin 16 - WR / P3.6	Pin 22 - ALE (Address Latch Enable)
	Pin 6 - START (Conversion Start)
Pin 17 - RD/P3.7	Pin 9 - OE (Output Enable)
Pin 32 - P0.7	Pin 21 – D7 (MSB)
Pin 33 - P0.6	Pin 20 – D6
Pin 34 - P0.5	Pin 19 – D5
Pin 35 - P0.4	Pin 18 – D4
Pin 36 - P0.3	Pin 8 - D3
Pin 37 - P0.2	Pin 15 – D2 and Pin 23, ADD C
Pin 38 - P0.1	Pin 14 – D1 and Pin 24, ADD B
Pin 39 - P0.0	Pin 17 – D0 (LSB) and Pin 22, ADD A
Pin 40 - Vcc	Pin 11 - VCC (5V)

Table 1.2 ADC0808 and P89V51RD2 microcontroller pin interconnections

ADC 0808	Sensor
Pin 5 - IN 7	Right IR Proximity sensor no. 4
Pin 4 - IN 6	Right White Line sensor
Pin 3 - IN 5	Centre White Line sensor
Pin 2 - IN 4	Left White Line sensor
Pin 1 - IN 3	Left IR Proximity sensor no. 2
Pin 28 - IN 2	Front Sharp IR Range sensor
Pin 27 - IN 1	Battery Voltage
Pin 26 - IN 0	Front IR proximity sensor 3

Table1.3 ADC 0808 pin connections with analog sensors

2. Loading Firmware on Fire Bird V P89V51RD2

Programming Fire Bird V P89V51RD2 Robot

Fire Bird V P89V51RD2 robot programming involves two steps. First step is to write and compile the code and generate the "*.hex" file. Second step is to load this "*.hex" file on the microcontroller using Flash Magic software provided by NXP (formerly Phillips)

We are going to use Keil-U-Vision (Version 3) software for writing the code for the microcontroller. We can also use any other open source of proprietary software supporting P89V51RD2 microcontroller. Fire Bird V CD contains free version of the uVision 3 software. You can also download latest version from http://www.keil.com/dd/chip/3711.htm and click C51 Evaluation Software

2.1 Steps for writing program in uVision3

µVision® 3
Integrated Development Environment for
KEIL™ An ARM® Company
RealView® Tools by ARM®
Copyright © 1997-2005 Keil Software, 2005-2007 ARM Ltd. All rights reserverd.
This product is protected by US and international laws.

Figure 2.1

Step 1: Start with Keil-U-Vision
Go to project and start new project.

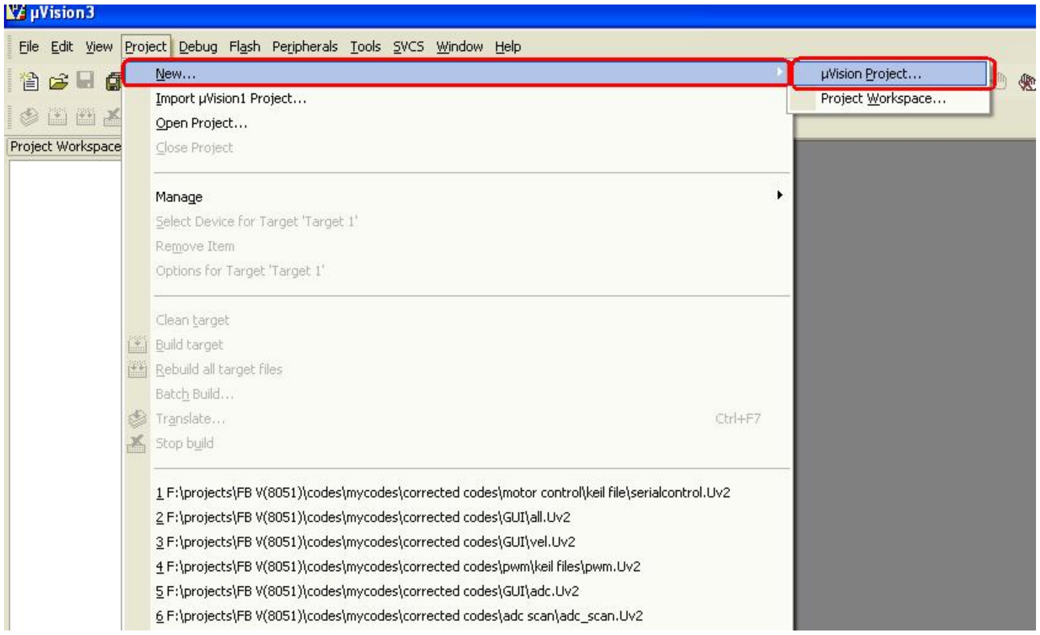

µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
New...
Import µVision1 Project...
Open Project...
Close Project
Manage
Select Device for Target 'Target 1'
Remove Item
Options for Target 'Target 1'
Clean target
Build target
Rebuild all target files
Batch Build...
Translate... Ctrl+F7
Stop build
1 F:\projects\FB V(8051)\codes\mycodes\corrected codes\motor control\keil file\serialcontrol.Uv2
2 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\all.Uv2
3 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\vel.Uv2
4 F:\projects\FB V(8051)\codes\mycodes\corrected codes\pwm\keil files\pwm.Uv2
5 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\adc.Uv2
6 F:\projects\FB V(8051)\codes\mycodes\corrected codes\adc scan\adc_scan.Uv2
µVision Project...
Project Workspace...

Figure 2.2

Step 2: Name your project and save it in your project folder. (always create new folder for new project)

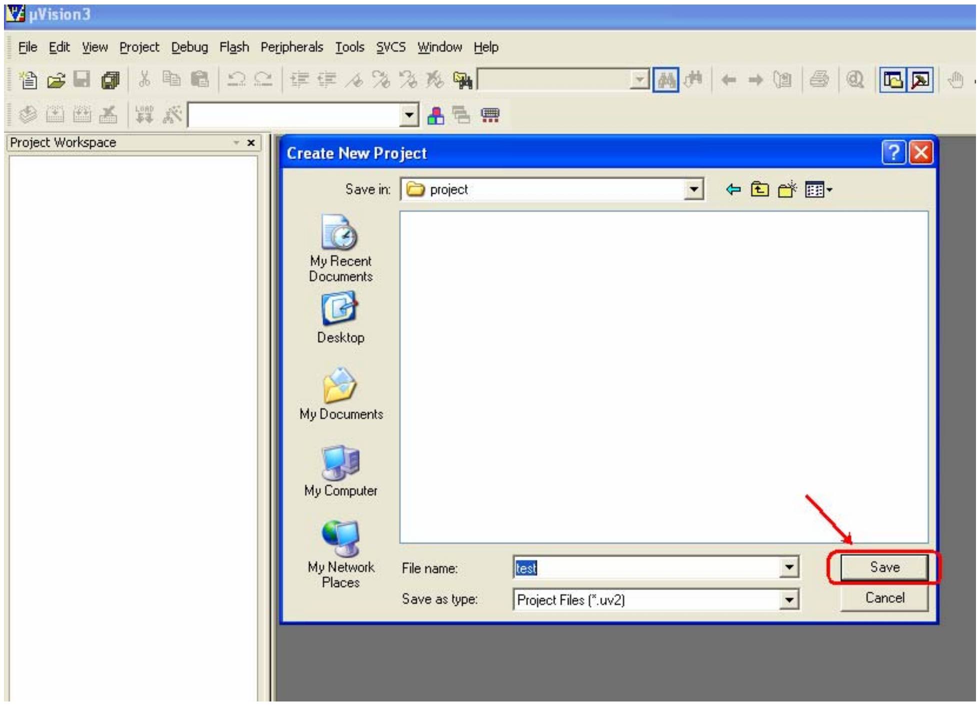


Figure 2.3

Step 3: A dialogue box will open asking you to select your device used. Select the appropriate device for e.g. P89V51RD2.Then click OK. P89V51RD2 can be found in the NXP (founded by Philips) directory.

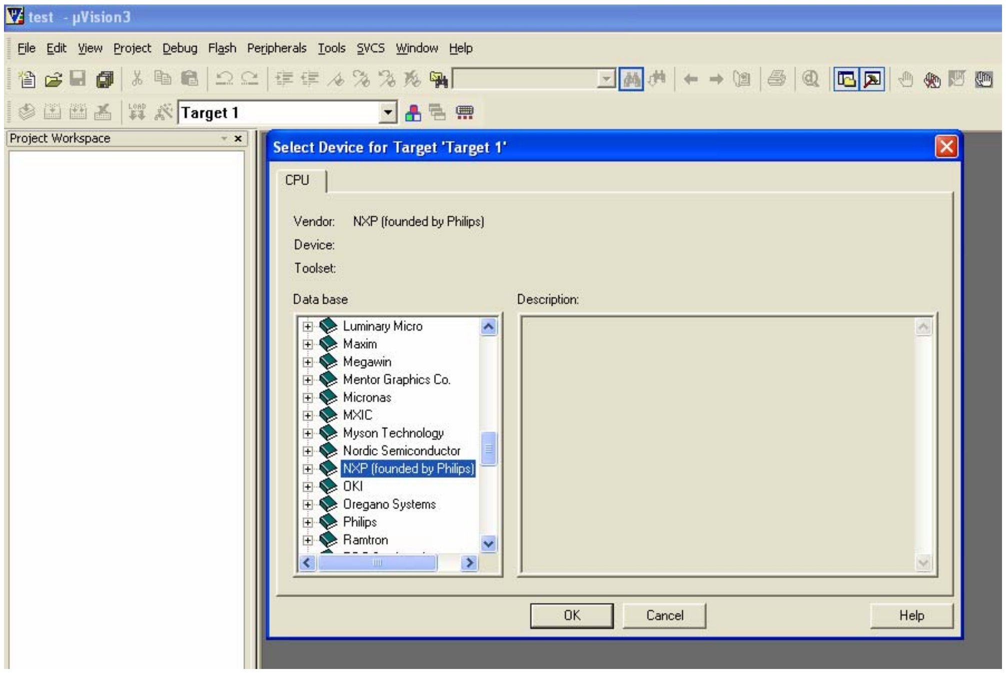
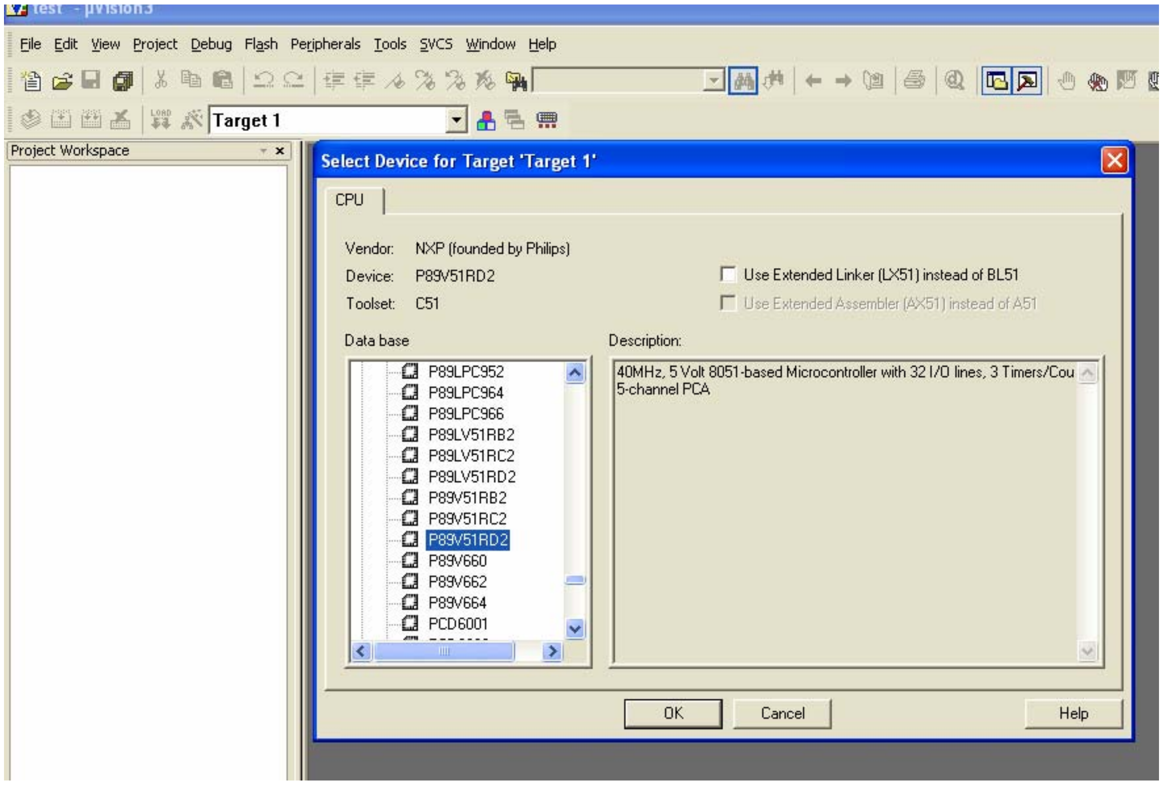


Figure 2.4


test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Select Device for Target 'Target 1'
CPU

Vendor: NXP (founded by Philips)
Device: P89V51RD2
Toolset: C51

Use Extended Linker (LX51) instead of BL51
Use Extended Assembler (AX51) instead of A51

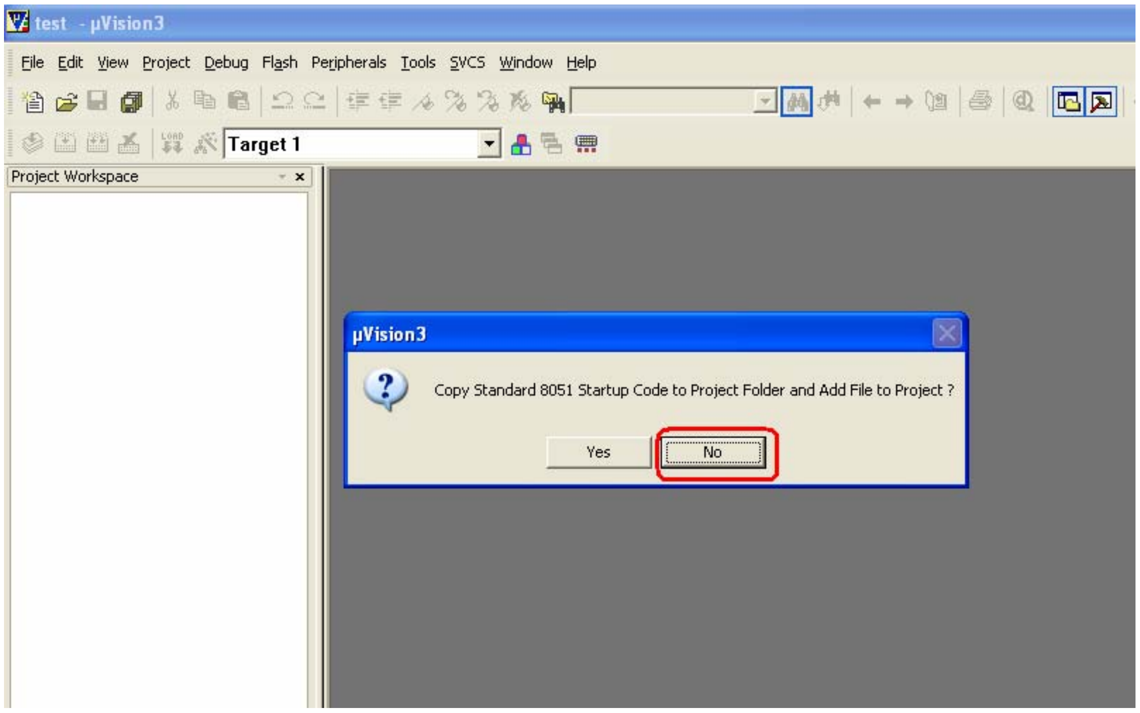
Data base

- P89LPC952
- P89LPC964
- P89LPC966
- P89LV51RB2
- P89LV51RC2
- P89LV51RD2
- P89V51RB2
- P89V51RC2
- P89V51RD2
- P89V660
- P89V662
- P89V664
- PCD6001

Description:
40MHz, 5 Volt 8051-based Microcontroller with 32 I/O lines, 3 Timers/Cou
5-channel PCA

Figure 2.5

Step 4: Next it will ask you if you want to add the A51STARTUP code.
Say “NO”,


test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
µVision3

Copy Standard 8051 Startup Code to Project Folder and Add File to Project ?

Figure 2.6

Step 5: In the Project Workspace right click on ‘Target 1’ and select options for target as shown. A dialogue box to choose different options will open.

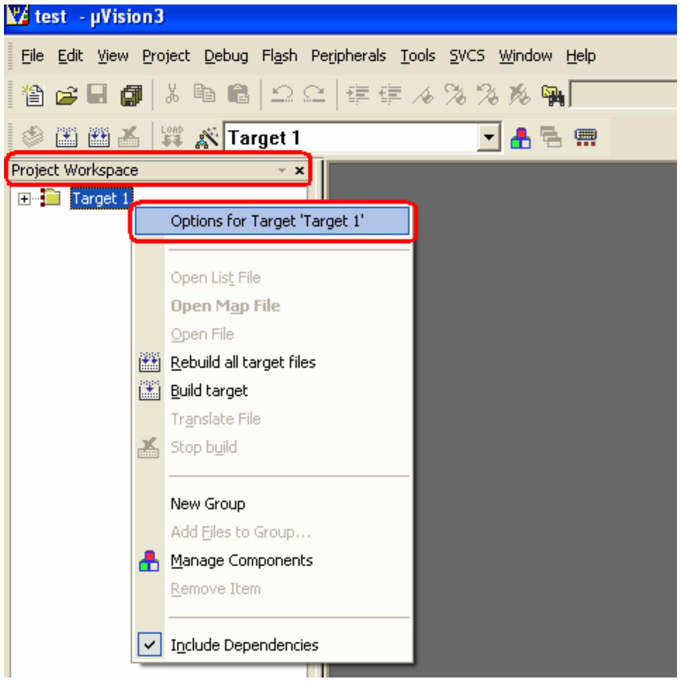
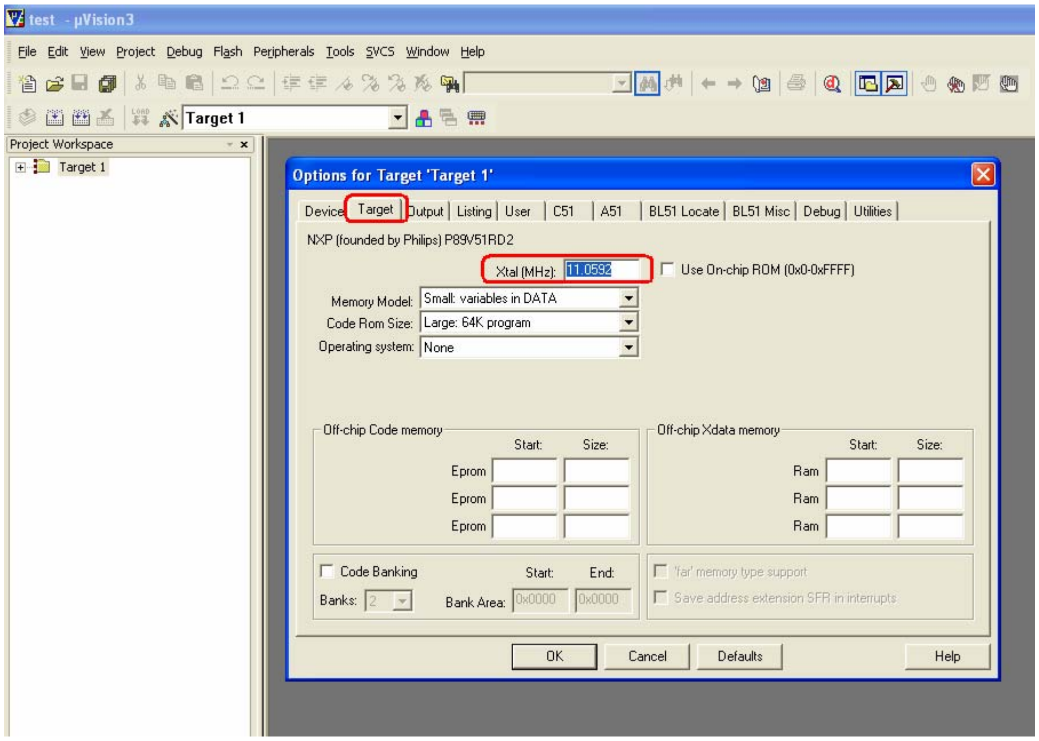


Figure 2.7

Step 6: Click on the the ‘Target’ tab. Enter the frequency of the crystal. For FIRE BIRD V its 11.0592 MHZ.


test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Options for Target 'Target 1'
Device Target Output Listing User C51 A51 BL51 Locate BL51 Misc Debug Utilities
OK Cancel Defaults Help

NXP (founded by Philips) P89V51RD2

Xtal (MHz): 11.0592 ☐ Use On-chip ROM (0x0-0xFFFF)

Memory Model: Small: variables in DATA
Code Rom Size: Large: 64K program
Operating system: None

Off-chip Code memory — Start: Size: — Eprom, Eprom, Eprom

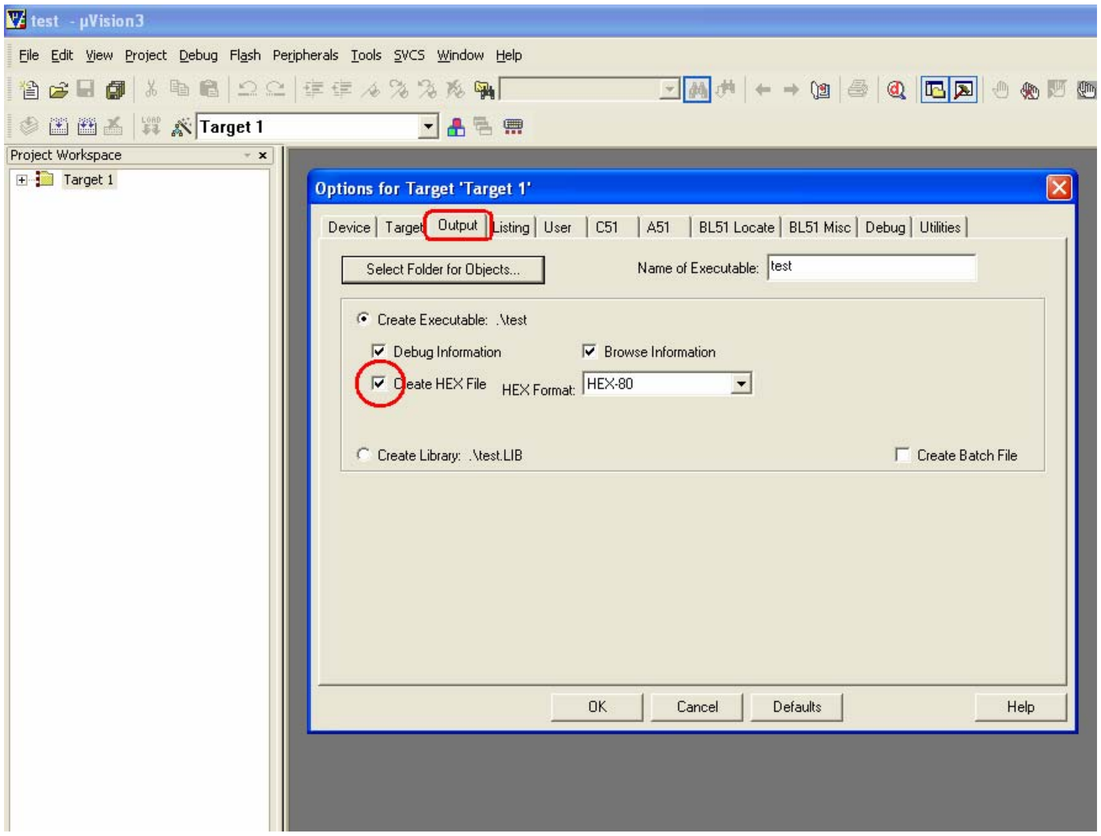
Off-chip Xdata memory — Start: Size: — Ram, Ram, Ram

☐ Code Banking Start: End:
Banks: 2 Bank Area: 0x0000 0x0000

☐ 'far' memory type support
☐ Save address extension SFR in interrupts

Figure 2.8

Step 7: Go to the output tab and tick on the file to create HEX-file. Then click OK to save your options. Other tabs can be left with the default options.


test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Options for Target 'Target 1'
Device Target Output Listing User C51 A51 BL51 Locate BL51 Misc Debug Utilities
Select Folder for Objects...
OK Cancel Defaults Help

Name of Executable: test

Create Executable: .\test

☑ Debug Information ☑ Browse Information

☑ Create HEX File HEX Format: HEX-80

Create Library: .\test.LIB ☐ Create Batch File

Figure 2.9

Step 8: After this is done, open a new file and save it with the project file as a C file i.e. with the extension ".c".

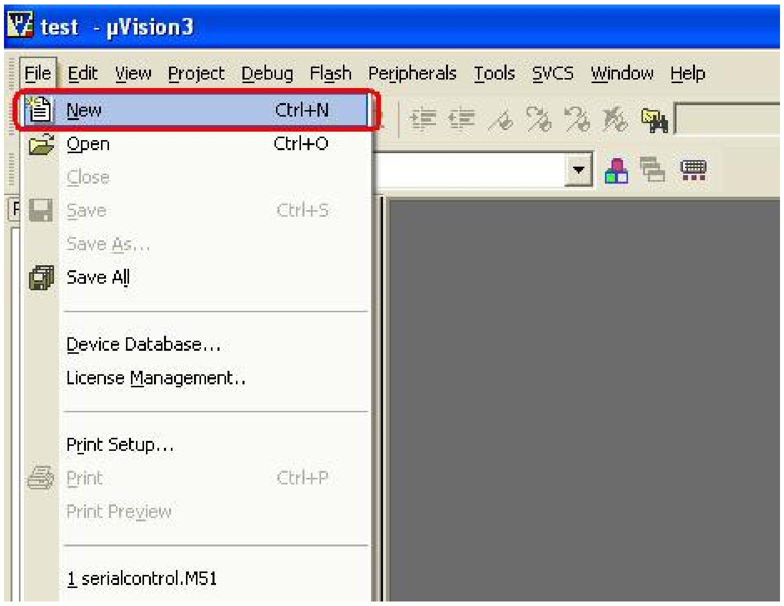


Figure 2.10

Step 9: Add this file to the project by Right-clicking on “Source Group” and choosing to add files to group. Select the appropriate ‘.C’ file to be added.

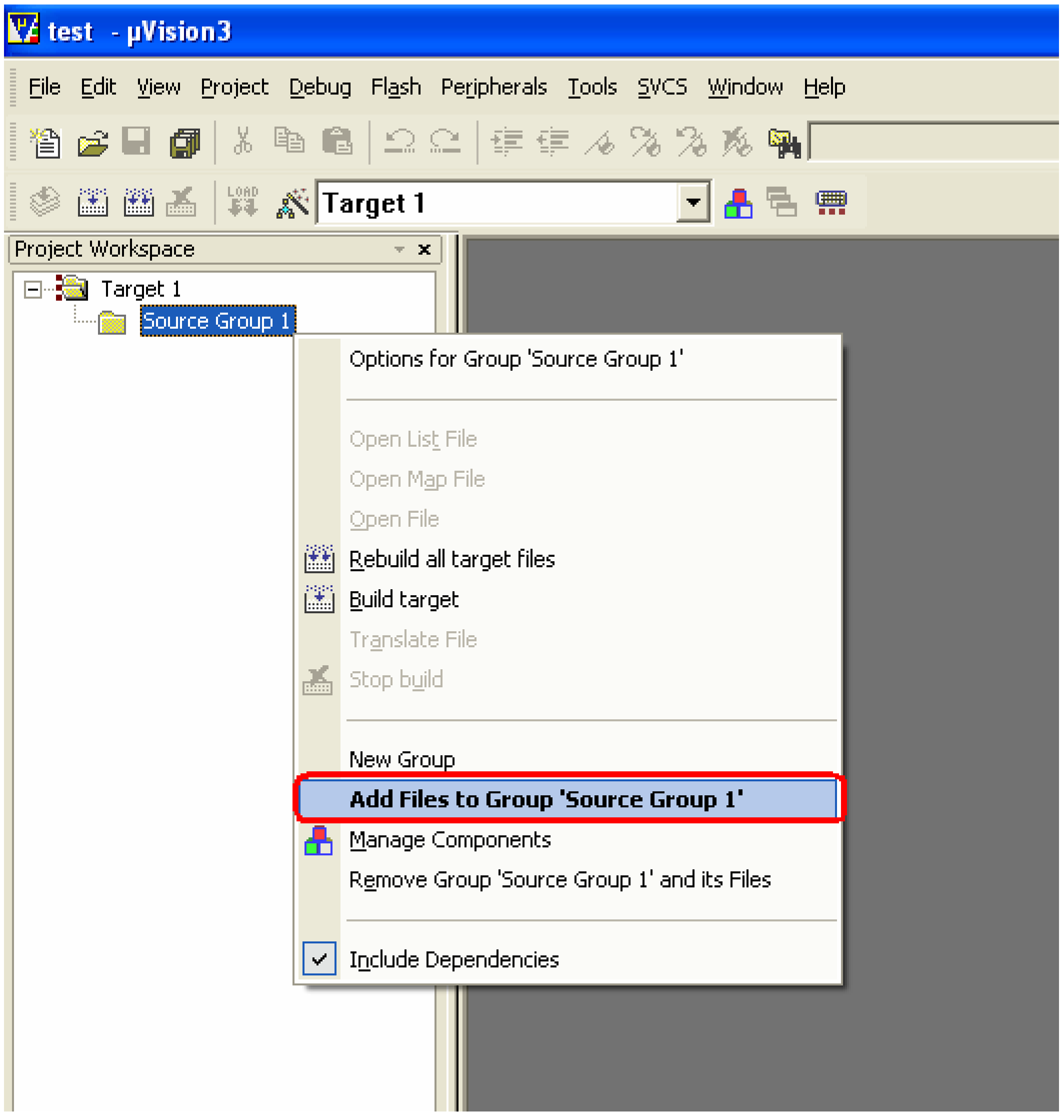
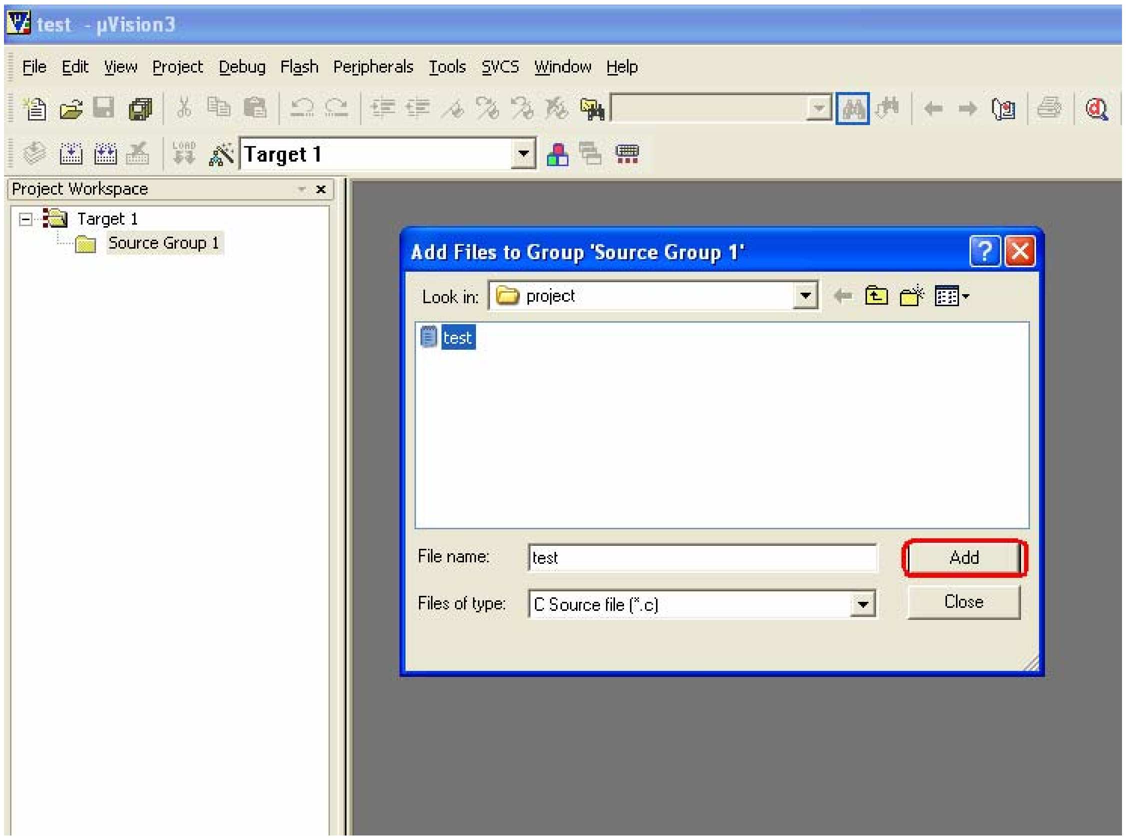


Figure 2.11


test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
Add Files to Group 'Source Group 1'
Look in:
project
test
Add
Close

File name: test

Files of type: C Source file (*.c)

Figure 2.12

Step 10: You can write macros and save them with an ".h" extension and add them to your project files. Write your 'C' code and save the file.
Add (write): #include <intrins.h>
Add (write): #include "p89v51rx2.h"

Add (write): #include "lcd_array_disp.h" If you want to display Data of LCD display. It is covered in chapter 6 in the software manual.

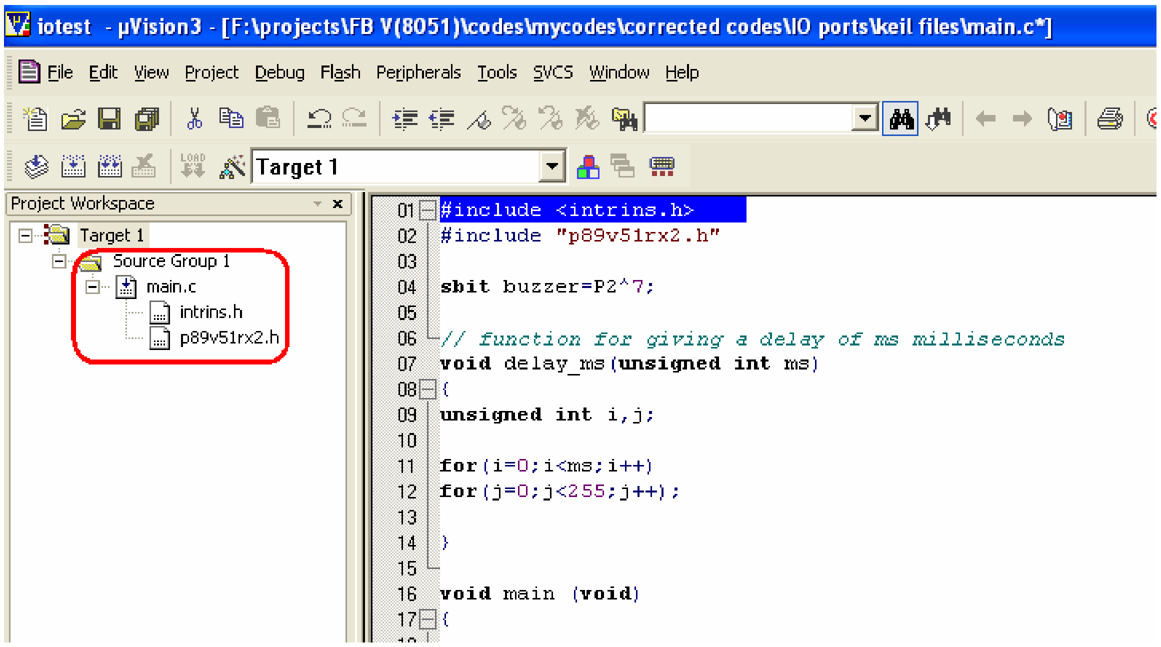
Copy file: 'p89v51rx2.h' from the folder 'Experiments' in the documentation folder and paste it in your project folder before building target.

If you want to display Data of LCD display Copy file: 'lcd_array_disp.h' from the folder 'Experiments' in the documentation folder and paste it in your project folder before building target.

Note: A declaration with the following syntax **#include** <intrins.h> would direct the compiler to look for the **.h** file in its own parent directory whereas the syntax **#include "**intrins.h**"** would direct the compiler to search for the **.h** file in the parent directory where the current project is stored. Therefore the double quotes **" "** are used to declare the files specific to the Microcontroller chip and its peripherals where as < > are used to declare more generic files which are compatible with multiple chips

Copy following code in your ‘C’ (#include <intrins.h> and #include "p89v51rx2.h" are already written in the ‘C’ file)

```
//code for buzzer beep

#include <intrins.h>
#include "p89v51rx2.h"

sbit buzzer=P2^7;  //buzzer = 1; buzzer off, buzzer = 0; buzzer on,

// function for giving a delay of ms milliseconds
void delay_ms(unsigned int ms)
{
unsigned int i,j;

for(i=0;i<ms;i++)
for(j=0;j<53;j++);

}

void main (void)
{

while(1)
{
buzzer=0;               //switch ON the buzzer
delay_ms(100);          //give delay of 100 milliseconds
buzzer=1;               //switch off the buzzer
delay_ms(100);
}
}
```


iotest - µVision3 - [F:\projects\FB V(8051)\codes\mycodes\corrected codes\IO ports\keil files\main.c*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
main.c
intrins.h
p89v51rx2.h

```
#include <intrins.h>
#include "p89v51rx2.h"

sbit buzzer=P2^7;

// function for giving a delay of ms milliseconds
void delay_ms(unsigned int ms)
{
unsigned int i,j;

for(i=0;i<ms;i++)
for(j=0;j<255;j++);

}

void main (void)
{
```

Figure 2.13

Step 11: Now click on the “Project” tab and choose to “Build Target” as shown.

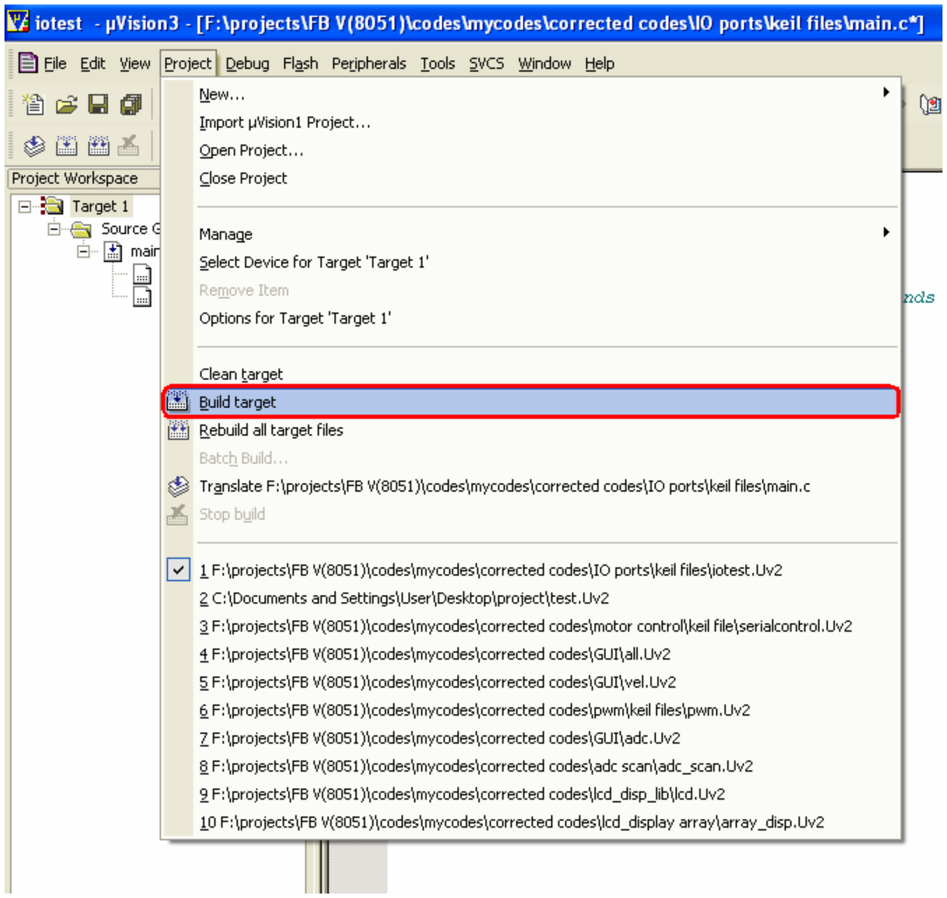

iotest - µVision3 - [F:\projects\FB V(8051)\codes\mycodes\corrected codes\IO ports\keil files\main.c*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
Target 1
Source G
main
New...
Import µVision1 Project...
Open Project...
Close Project
Manage
Select Device for Target 'Target 1'
Remove Item
Options for Target 'Target 1'
Clean target
Build target
Rebuild all target files
Batch Build...
Translate F:\projects\FB V(8051)\codes\mycodes\corrected codes\IO ports\keil files\main.c
Stop build
1 F:\projects\FB V(8051)\codes\mycodes\corrected codes\IO ports\keil files\iotest.Uv2
2 C:\Documents and Settings\User\Desktop\project\test.Uv2
3 F:\projects\FB V(8051)\codes\mycodes\corrected codes\motor control\keil file\serialcontrol.Uv2
4 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\all.Uv2
5 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\vel.Uv2
6 F:\projects\FB V(8051)\codes\mycodes\corrected codes\pwm\keil files\pwm.Uv2
7 F:\projects\FB V(8051)\codes\mycodes\corrected codes\GUI\adc.Uv2
8 F:\projects\FB V(8051)\codes\mycodes\corrected codes\adc scan\adc_scan.Uv2
9 F:\projects\FB V(8051)\codes\mycodes\corrected codes\lcd_disp_lib\lcd.Uv2
10 F:\projects\FB V(8051)\codes\mycodes\corrected codes\lcd_display array\array_disp.Uv2

Figure 2.14

- Check for any errors in the ‘Output Window’.
- If there are no errors then ‘Hex’ file will be created and stored in the Project folder.
- You can then download this file onto your microcontroller using the In System Programming (ISP) software i.e. using flash tools like Flash Magic or FLIP or parallel programming as supported by your microcontroller.

Note: Use ‘rebuild all target files’ option if you are including multiple header files

⚠**Warning:**
You need to set jumpers to select wired RS232 option between wired RS232 and Wireless communication options before loading hex file on the robot.

Make sure that jumper is configured to enable RS232 serial communication. Jumpers should be in the position as shown in the Figure 2.15.

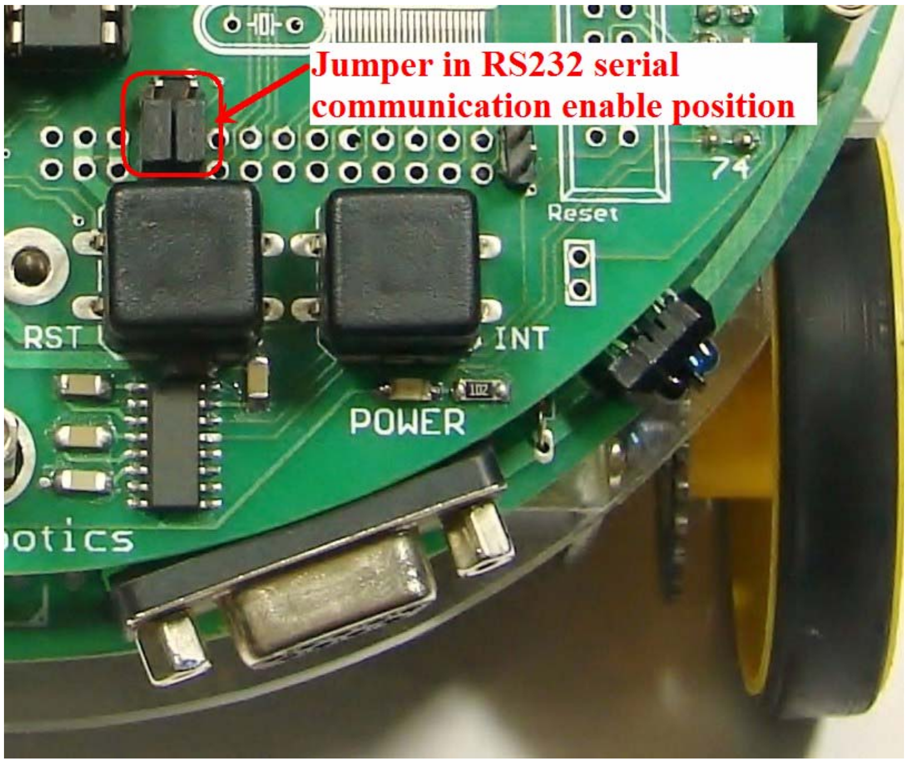


Figure 2.15: Enabling RS232 communication

2.2 Loading the generated Hex file on the microcontroller using Serial Port

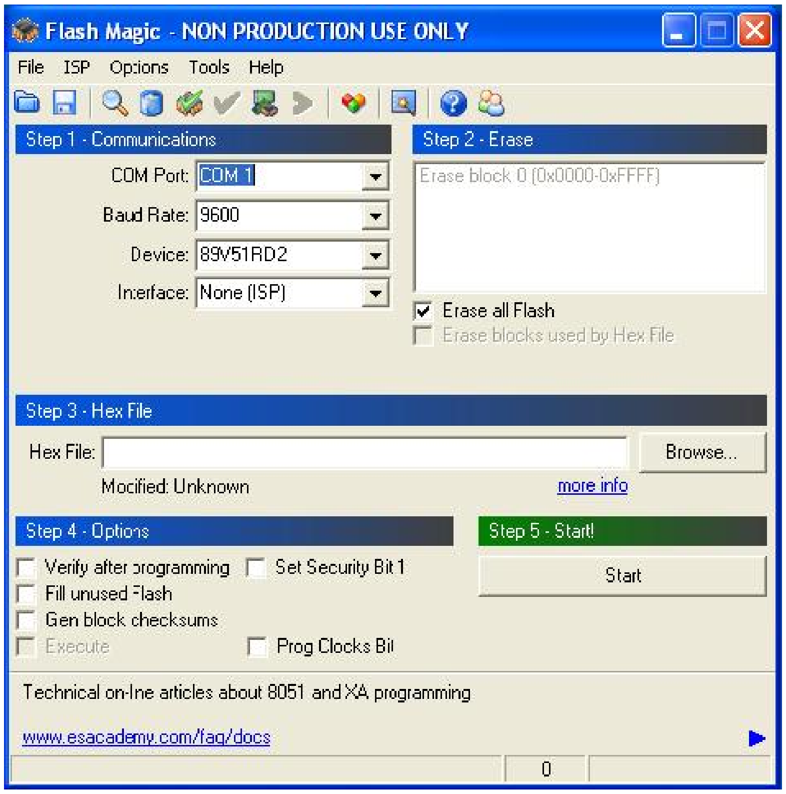
Figure 2.16

Flash Magic is Windows software which allows easy access to all the In System Programming (ISP) features provided by the devices.

These features include:
• Erasing the Flash memory (individual blocks or the whole device)
• Programming the Flash memory
• Reading Flash memory
• Reading the signature bytes
• Reading and writing the security bits
• Direct load of a new baud rate (high speed communications)
• Sending commands to place device in Boot loader mode

Flash Magic provides a clear and simple user interface to these features

Step 1: Go to '*Flash Magic*' Icon, it will open the main window as show below.



COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Erase block 0 (0x0000-0xFFFF)
[✓] Erase all Flash
[ ] Erase blocks used by Hex File

Hex File:
Modified: Unknown

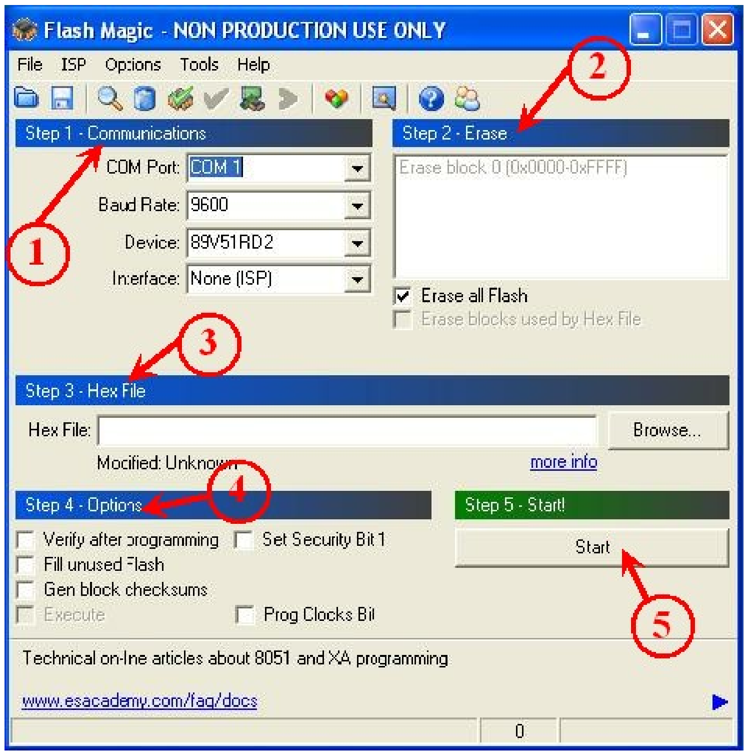
[ ] Verify after programming
[ ] Set Security Bit 1
[ ] Fill unused Flash
[ ] Gen block checksums
[ ] Execute
[ ] Prog Clocks Bit

Technical on-line articles about 8051 and XA programming
www.esacademy.com/faq/docs
0

Figure 2.17

In main window you can see five types of sections.

- Communications
- Erase
- Hex File
- Options
- Start



COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Erase block 0 (0x0000-0xFFFF)

☑ Erase all Flash
☐ Erase blocks used by Hex File

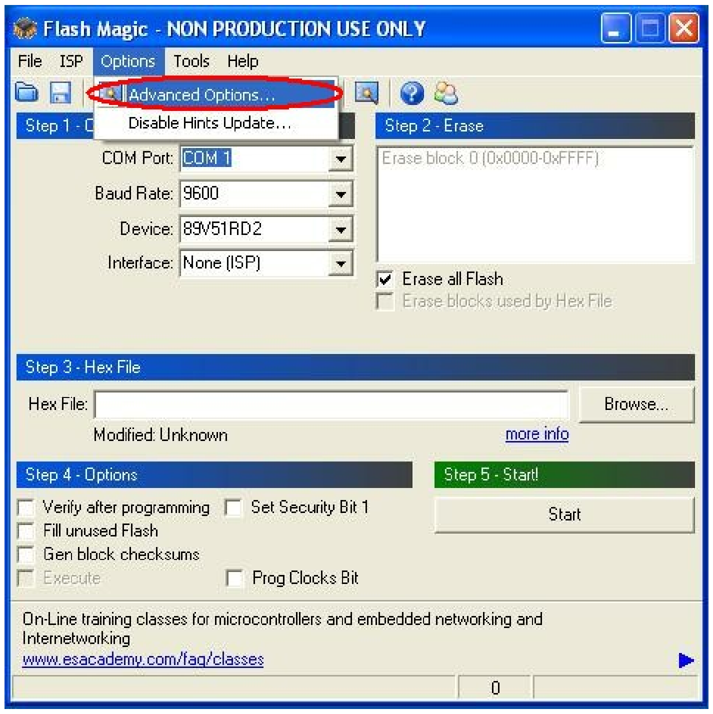
Hex File:
Modified: Unknown

☐ Verify after programming ☐ Set Security Bit 1
☐ Fill unused Flash
☐ Gen block checksums
☐ Execute ☐ Prog Clocks Bit

Technical on-line articles about 8051 and XA programming
www.esacademy.com/faq/docs

Figure 2.18

Step 2: Go to "*Option*" in the toolbar, and select first menu '*Advance options*.'



COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Erase block 0 (0x0000-0xFFFF)

☑ Erase all Flash
☐ Erase blocks used by Hex File

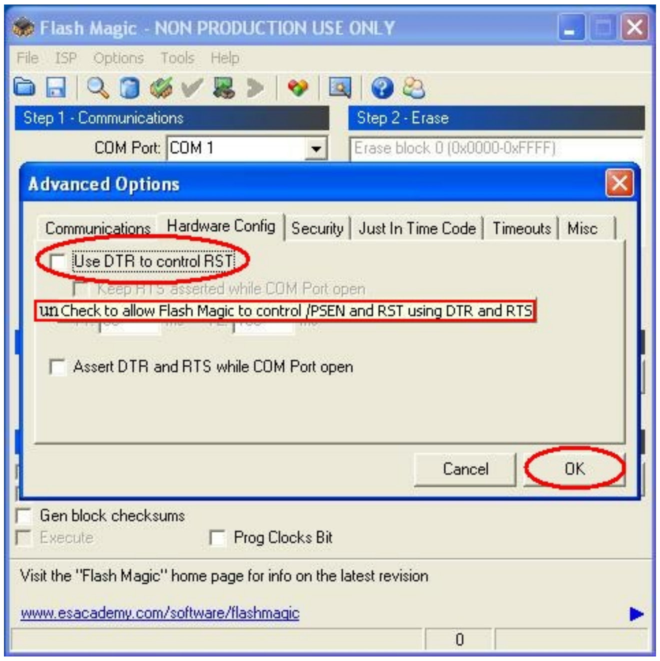
Hex File:
Modified: Unknown

☐ Verify after programming ☐ Set Security Bit 1
☐ Fill unused Flash
☐ Gen block checksums
☐ Execute ☐ Prog Clocks Bit

On-Line training classes for microcontrollers and embedded networking and Internetworking
www.esacademy.com/faq/classes

Figure 2.19

It will show you extracted window with many options.



COM Port: COM 1

Erase block 0 (0x0000-0xFFFF)

☐ Use DTR to control RST

☐ Keep RTS asserted while COM Port open

un Check to allow Flash Magic to control /PSEN and RST using DTR and RTS

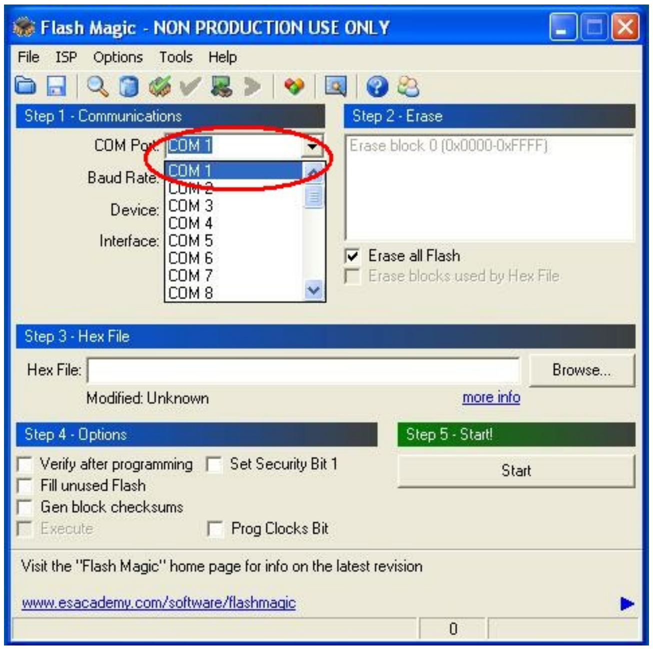
☐ Assert DTR and RTS while COM Port open

Visit the "Flash Magic" home page for info on the latest revision

Figure 2.20

Uncheck the box which is highlighted to don't allow Flash Magic to control PSEN & RST using DTR & RTS.

Step 3: Now go for Communication selection, select '*COM 1*' from '*COM PORT*' Option if you are using serial port. If you are using USB to serial converter from NEX Robotics then find out COM port number with the help of documentation provided with the USB to serial converter.



COM Port: COM 1

Erase block 0 (0x0000-0xFFFF)

☑ Erase all Flash

☐ Erase blocks used by Hex File

Modified: Unknown

☐ Verify after programming ☐ Set Security Bit 1

☐ Fill unused Flash

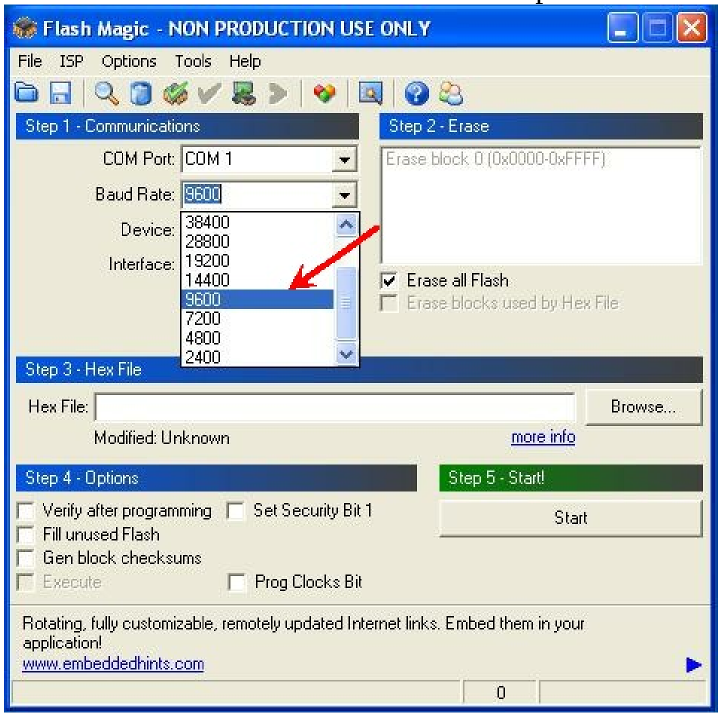
☐ Gen block checksums

☐ Execute ☐ Prog Clocks Bit

Visit the "Flash Magic" home page for info on the latest revision

Figure 2.21

Select ‘*Baud Rate*’ 9600 from Baud Rate option.


Flash Magic - NON PRODUCTION USE ONLY

Step 1 - Communications

COM Port: COM 1

Baud Rate: 9600

Step 2 - Erase

Erase block 0 (0x0000-0xFFFF)

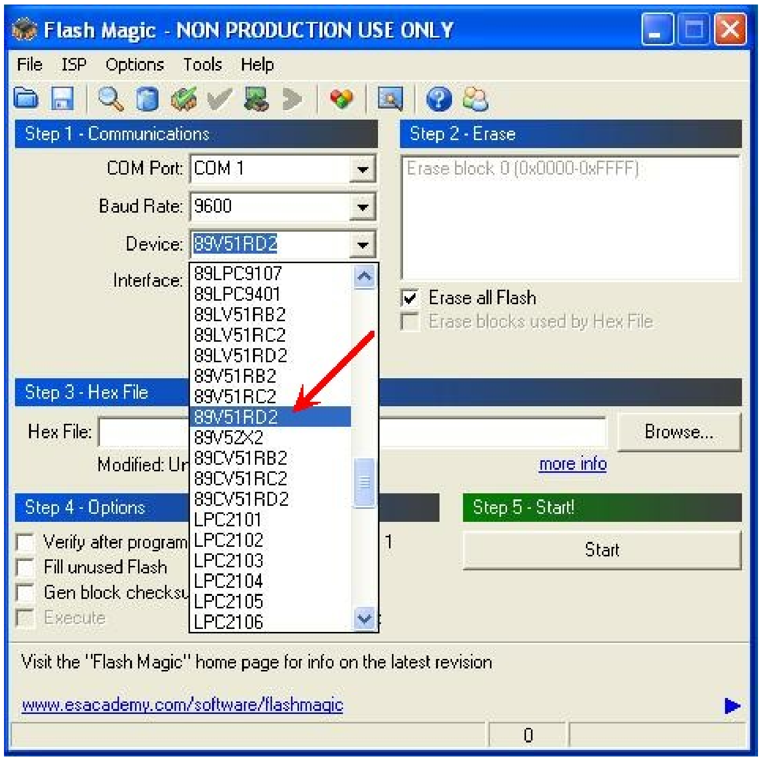
Erase all Flash: ✓

Rotating, fully customizable, remotely updated Internet links. Embed them in your application!

www.embeddedhints.com

Figure 2.22

Select proper Device ‘*89V51RD2*’ from Device option.


Flash Magic - NON PRODUCTION USE ONLY

Step 1 - Communications

COM Port: COM 1

Baud Rate: 9600

Device: 89V51RD2

Step 2 - Erase

Erase block 0 (0x0000-0xFFFF)

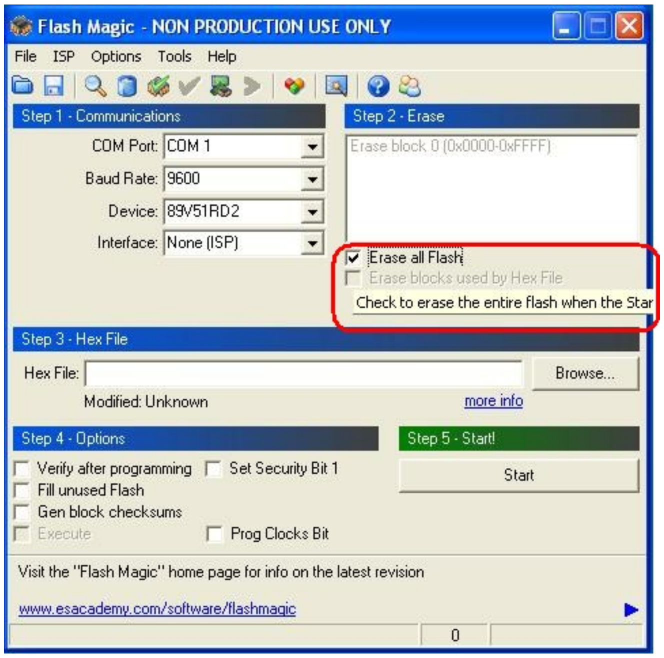
Erase all Flash: ✓

Visit the "Flash Magic" home page for info on the latest revision

www.esacademy.com/software/flashmagic

Figure 2.23

Step 4: Go to '*Erase*' section for erasing the all '*Flash*' or Blocks used by '*HEX File*'



COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Erase block 0 (0x0000-0xFFFF)
☑ Erase all Flash
☐ Erase blocks used by Hex File
Check to erase the entire flash when the Star

Hex File:
Modified: Unknown

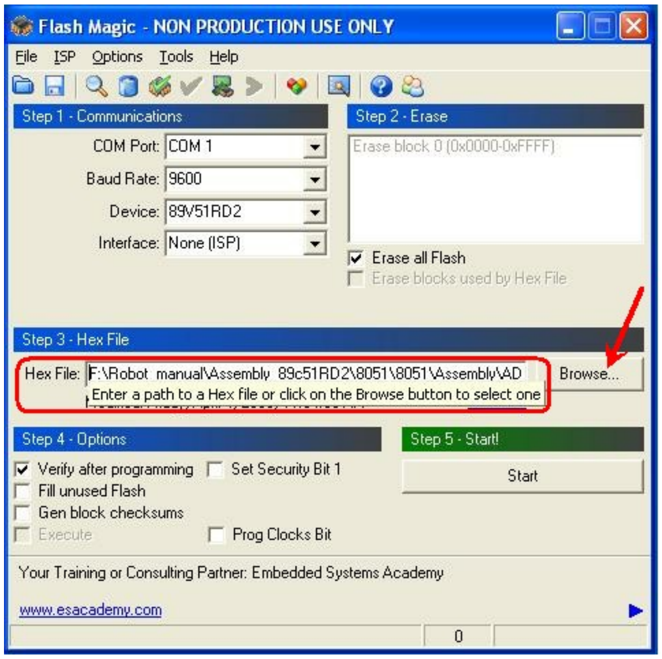
☐ Verify after programming ☐ Set Security Bit 1
☐ Fill unused Flash
☐ Gen block checksums
☐ Execute ☐ Prog Clocks Bit

Visit the "Flash Magic" home page for info on the latest revision
www.esacademy.com/software/flashmagic
0

Figure 2.24

Check on the check box to Erase all Flash.

Step 5: Go to '*HEX File*' section click '*Browse'* to select proper Hex file which is to be loaded into Fire Bird V P89V51RD2 Robot.



COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Erase block 0 (0x0000-0xFFFF)
☑ Erase all Flash
☐ Erase blocks used by Hex File

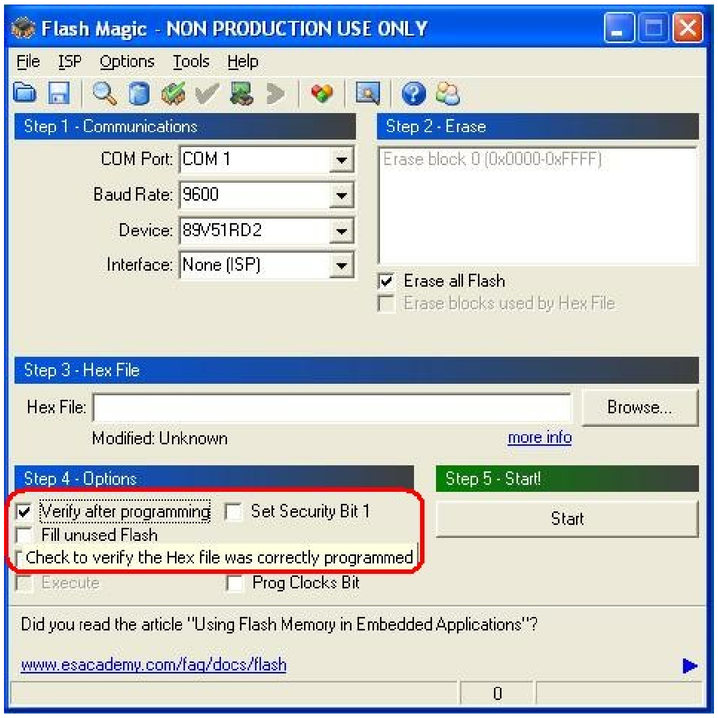
Hex File: F:\Robot manual\Assembly 89c51RD2\8051\8051\Assembly\AD
Enter a path to a Hex file or click on the Browse button to select one

☑ Verify after programming ☐ Set Security Bit 1
☐ Fill unused Flash
☐ Gen block checksums
☐ Execute ☐ Prog Clocks Bit

Your Training or Consulting Partner: Embedded Systems Academy
www.esacademy.com
0

Figure 2.25

Step 6: Go to '*Option*' section to select *'Verify after programming'* option. This will verify hex file after loading.



Flash Magic - NON PRODUCTION USE ONLY

Step 1 - Communications
COM Port: COM 1
Baud Rate: 9600
Device: 89V51RD2
Interface: None (ISP)

Step 2 - Erase
Erase block 0 (0x0000-0xFFFF)
☑ Erase all Flash
☐ Erase blocks used by Hex File

Step 3 - Hex File
Hex File:
Modified: Unknown

Step 4 - Options
☑ Verify after programming ☐ Set Security Bit 1
☐ Fill unused Flash
Check to verify the Hex file was correctly programmed
☐ Execute ☐ Prog Clocks Bit

Step 5 - Start!

Did you read the article "Using Flash Memory in Embedded Applications"?
www.esacademy.com/faq/docs/flash

Figure 2.26

Step 7: After doing all the required settings, connect serial cable between robot and PC. Turn on the Fire Bird V P89V51RD2 robot and click on the *'Start'*, it will ask to *'RESET TO DEVICE IN TO ISP MODE'*, now press RESET Switch on the Fire Bird V, Flash magic will load hex file on the robot and verify it for correctness.

If you are using NEX Robotic's USB to serial converter then please refer to Chapter 6. PC Based Control Using Serial Communication for how to use it and identify COM port number.

3. Input / Output Operations on the Robot

The P89V51RD2 is a 8051 architecture based microcontroller with 4, 8 bit general purpose I/O ports from P0 to P3. Input / out operations are the most basic, easy and essential operations.

We will need frequent I/O operations to do following tasks:

Function	**Pins**	**Input / Output**	**Recommended Initial State**
Robot direction and velocity control	P1.0 to P1.4 & P3.4	Output	Logic 0
LCD display control	P2.0 to P2.6	Input / Output	Logic 0
ADC0808	P0.0 to P0.7 &P3.2	Input / Output	-
Buzzer control	P2.7	Output	Logic 0
Position Encoder	P3.3 & P3.5	Input	Pulled up*

Table 3.1

*In P89V51RD2 all the ports except PORT 0 have internal pull-ups. In Fire Bird V P89V51RD2 pull-up of 10K ohm is connected to all the pins of PORT 0 to PORT 3.

When logic 1 is written to the Pn register (where n is port number ranging from 0 to 3) I/O pin is pulled up.

Note: Port 0 (P0) does not have internal pull-ups and floats in the input state. It can be used as a high impedance input in this state. (Reference: P89V51RD2 datasheet). Also External pull-up of 10K is attached to each port.

Setting Ports as Input or Output

To set a Port to act as input all pins need to be initialized to 1. When an input is externally pulled low (i.e. by some peripheral device) then that port pin will source current and thus a low level is detected when the pin is read. On the other hand no prior initialization is required for the port to behave as an output. All the I/O ports are bit addressable, i.e. individual bits can be set/reset separately.

Example 3.1.: The following piece of code- Sets Port 0 as an Input port and stores its value in the variable K.

```
{
int k;
P0=0xFF;
k=P0;
While (1);
}
```

Example3.2: The following code- Sets the 4th bit of Port 3 (i.e P3.3) to logic 1

```
{
P0^3=1;
While (1);
}
```

3.2 Buzzer control

Buzzer is located on the Fire Bird V robot's main board. In All the Fire Bird V robots except 89V51RD2 based robot buzzer will turn on when logic 1 is applied to the buzzer enable pin. However in the case of P89V51RD2 when reset switch of the microcontroller is pressed, all its pins become tri-stated (isolated). Since external pull-ups are connected to the all the pins of the 89V51RD2 microcontroller, when microcontroller is in reset state or it's being programmed buzzer would have remained on. To avoid this, an additional transistor based not gate is added in series with buzzer enable pin. Because of this in Fire Bird V P89V51RD2 robot buzzer gets turned on when 7th pin of the PORT 2 (P2.7) is pulled low. To turn buzzer off apply logic 1 to the P2.7.

Example Code
(Also available in the "Experiments \ 1 IO Operations Buzzer Beep" folder in the documentation CD)

```
//Buzzer is connected to the 8th pin of Port 2 i.e P2.7
// To turn it on we have to set P2^7 to zero

#include <intrins.h>
#include "p89v51rx2.h"
sbit buzzer=P2^7;

// function for giving a delay of ms milliseconds
void delay_ms(unsigned int ms)
{
unsigned int i,j;

for(i=0; i<ms; i++)
for(j=0; j<53; j++);

}

// main begins here
void main (void)
{
while(1)
{
buzzer=0;// turn on the buzzer
delay_ms(100); //give delay of 100 milliseconds
buzzer=1; // turn off the buzzer
delay_ms(100);
}

}// main ends here
```

Explanation of the example code:
In this code, first few lines represent the header file declaration. The # include directive is used for including header files in the existing code. The syntax for writing header file is as follows:

#include <intrins.h>

This # include directive will add the already existing intrins.h header file stored in 'INC' folder in the main 'keil' directory (for example: C:\\Keil\C51\INC\ INTRINS.H) The same way other header files can also be included in the main program so that we can use various utilities defined in the header files.

The **sbit** type defines a bit within a special function register (SFR). It is used in one of the following ways:
sbit *name* = *sfr-name* ^ *bit-position*;
sbit *name* = *sfr-address* ^ *bit-position*;
sbit *name* = *sbit-address*;

In this case the special function register is P2 (the I/O register for PORT 2) and bit position is 7 to which the buzzer has been connected.

The function delay_ms() provides a delay of 'ms' milliseconds where ms is the value passed to it by the calling function

In the function main(), the while loop toggles the buzzer ON and OFF with a finite delay of 100 milliseconds

Note: A declaration with the following syntax **#include <**intrins.h**>** would direct the compiler to look for the **.h** file in its own parent directory whereas the syntax **#include "**intrins.h**"** would direct the compiler to search for the **.h** file in the parent directory where the current project is stored. Therefore the double quotes **" "** are used to declare the files specific to the Microcontroller chip and its peripherals where as < > are used to declare more generic files which are compatible with multiple chips

3.3 Motor direction control

Robot's motors are controlled by L293D motor controller from ST Microelectronics. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are applied to IC L293D's direction pins. Velocity control is done using pulse width modulation (PWM). Velocity control using PWM will be covered in the Chapter 5: Timer / Counter operations on the Robot.

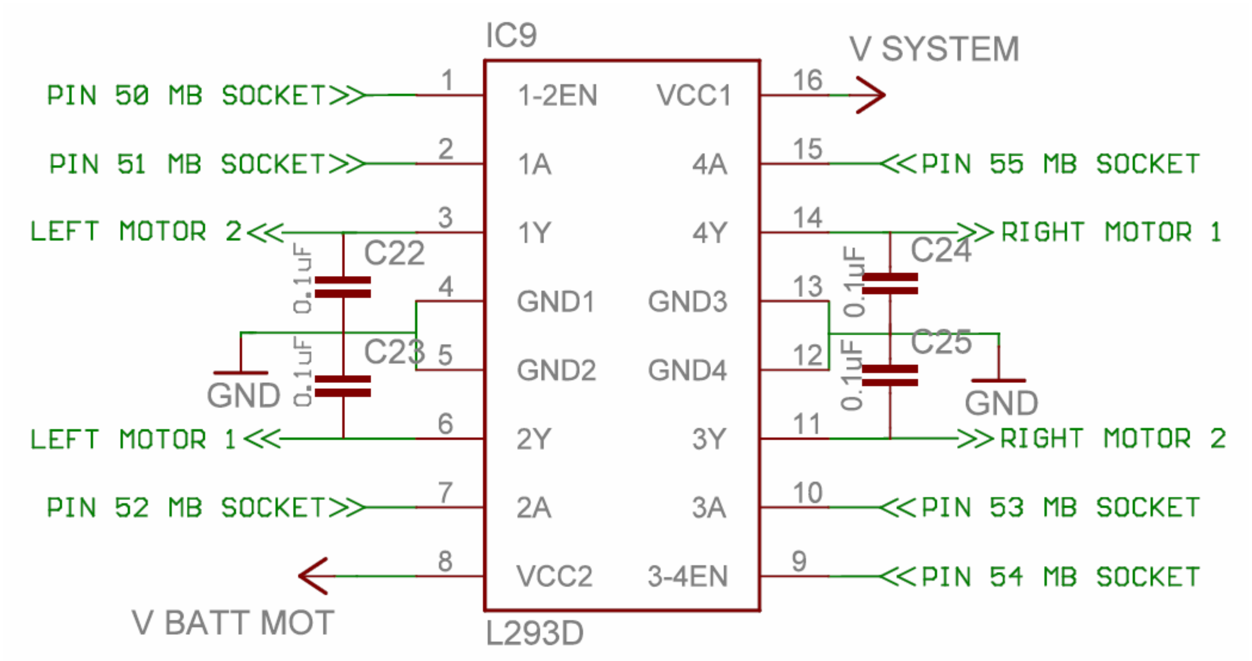


Fig: 3.1: Schematic of Motor Control circuit

Logic table for direction control of Motors:

DIRECTION	**LEFT BACKWARD (LB) P1.0**	**LEFT FORWARD (LF) P1.1**	**RIGHT FORWARD (RF) P1.2**	**RIGHT BACKWARD (RB) P3.4**	**PWM**
FORWARD	0	1	1	0	As per velocity requirement
REVERSE	1	0	0	1	As per velocity requirement
RIGHT *(Left wheel fwd, Right wheel bckwd)*	0	1	0	1	As per velocity requirement
LEFT*(Left wheel bckwd, Right wheel fwd)*	1	0	1	0	As per velocity requirement
SOFT RIGHT*(Left wheel fwd, Right wheel stop)*	0	1	0	0	As per velocity requirement

SOFT LEFT*(Left wheel stop, Right wheel fwd)*	0	0	1	0	As per velocity requirement
SOFT RIGHT 2 *(Left wheel stop, Right wheel bckwd)*	0	0	0	1	As per velocity requirement
SOFT LEFT 2 *(Left wheel bckwd, Right wheel stop)*	1	0	0	0	As per velocity requirement
HARD STOP	0	0	0	0	As per velocity requirement
SOFT STOP (Free running stop)	X	X	X	X	0

Table 3.2

Example Code
(Also available in the "Experiments \ 2 IO Operations Motor Direction Control" folder in the documentation CD)

```
#include <intrins.h>
#include "p89v51rx2.h
sbit buzzer = P2^7;  // buzzer is
connected to P2.7
//The appropriate bits have been defined
as per the logic table for motion control

sbit left_velocity=P1^3;
sbit right_velocity=P1^4;
sbit LB=P1^0;
sbit LF=P1^1;
sbit RF=P1^2;
sbit RB=P3^4;

// function for giving a delay of ms
milliseconds
void delay_ms(unsigned int ms)
{
unsigned int i,j;

for(i=0; i<ms; i++)
for(j=0; j<53; j++);

}

//these are the direction functions
void forward(void) //go forward
{
RF=1;
LF=1;
RB=0;
LB=0;
}
void backward(void) //go backward
{
RF=0;
LF=0;
RB=1;
LB=1;
}

void left(void) //go left ( hard left)
{
RF=1;
LF=0;
RB=0;
LB=1;
}


void right(void)  //go right ( hard right)
{
RF=0;
LF=1;
RB=1;
LB=0;
}
void stop(void) //stop ( soft stop )
```

```
{
RF= 0;
LF= 0;
RB= 0;
LB= 0;

}
// main begins here


void main (void)
{
Buzzer=1;
left_velocity=1;
right_velocity=1;

while(1)
{
forward();          //go forward
delay_ms(2000); //delay of 2 seconds

stop();                //stop
backward();       //go backward
delay_ms(2000); //delay of 2 seconds

stop();                //stop
delay_ms(1000); //delay of 1 second

left();                //go left
delay_ms(2000); //delay of 2 seconds

stop();                //stop
delay_ms(1000); //delay of 1 second

right();            //go right
delay_ms(2000); //delay of 2 seconds

stop();              //stop
delay_ms(2000); //delay of 2 seconds
}
}// main ends here
```

Explanation of the example code:

The above given program creates easy to use functions for motor direction control. In the main function a continuously repeating while loop is written which calls the motor forward, backward, right, left for a finite delay given by the delay_ms() function.

Note:

- In this case the right and left motor velocity have been set to the maximum value, that is the two enable pins of the motor driver L293D are always asserted. When velocity is to be varied a PWM signal should be applied at these two pins. We will apply PWM for velocity control at these two pins in the Chapter 5: Timer / Counter operations on the Robot.

- In the above example code if you remove function:
  ```
  right();            //go right
  delay_ms(2000); //delay of 2 seconds
  ```

- Between the changes of the motor directions robot might start behaving erratically and backlight of the LCD will flicker. This will happen only when robot's battery is half charged or it is running on the Auxiliary power. This is because sudden direction change of the motor will cause huge current surge, which will reset the microcontroller. That's why it is very important that while changing direction robot's motors should be in off state for a while (at least 250 to 500 millis econd). While debugging big code small hardware bug like this will make debugging very complicated. We always have to make sure that there is no sudden change in the motor direction to extend life of the motor and avoid problems arising due to current surges.

4. Robot Position Control using Interrupts

Fire Bird V P89V51RD2 uses various interrupt handling mechanisms such as timer overflow interrupts, timer compare interrupts, serial interrupts for wired & wireless communication and external hardware interrupts position control. In this chapter, we will have a brief overview of interrupt concept and will implement external hardware interrupt for position estimation of robots using position encoders.

Robot’s left wheel position encoder is connected to Interrupt 1 (INT1). An inverting schmitt trigger buffer is connected between encoder and the interrupt to remove spurious signal generated because of low slew rate of the encoder pulse and chatter.

Interrupts *interrupt* the flow of the program and cause it to branch to ISR (Interrupt Service Routine). ISR does the task that needs to be done when interrupt occurs. Whenever position encoder moves by one tick it interrupts the microcontroller and ISR does the job of tracking position count.

Each interrupt has a vector address assigned to it in the lowest section in program memory. The compiler places the starting address of the associated interrupt service routine and a relative jump instruction at the vector location for each interrupt. When the interrupt occurs, the program completes executing its current instruction and branches to the vector location associated with that interrupt. The program then executes the relative jump instruction to the interrupt service routine (ISR) and begins executing the ISR.

When an interrupt occurs, the return address is stored on the system stack. The RETI assembly language instruction causes the return address to be popped off the stack and continue program execution from the point where it was interrupted.

4.1 Robot position control using external interrupt INT1

Position encoder gives 30 pulses per revolution. Wheel has 52mm diameter. This means when robot moves by 5.44 mm, position encoder gives a pulse. We can estimate robot’s position as well as velocity using position encoder.

Position encoder on the left motor is connected to the External Interrupt 1 (INT 1) pin of the microcontroller. Every pulse (falling edge or low level, depending upon interrupt mode configured) from the encoder causes an interrupt to occur in the Microcontroller. In the ISR unsigned integer variable “left_shaft_count” is incremented every time an interrupt occurs. This count is used to calculate the actual distance traveled by the robot using the following formula.

Distance (in mm) = count * 5.44

Or

Required count = distance/5.44

By continuously comparing the current 'count' to the 'required count' as obtained above, position of the robot can be controlled.

Using Interrupts:
Interrupts needs to be initialized before they can be used. Initializing interrupts involves programming the following registers.

1. IEN0- Interrupt enable register 0:
Used to enable all interrupts servicing and enable external interupt

Bit	Symbol	Description	Bit Value
7	EA	Interrupt Enable Bit, EA = 1; Interrupt(s) servicing enabled	1
6	EC	PCA Interrupt Enable bit, EC = 0; PCA interrupt disabled	0
5	ET2	Timer 2 Interrupt Enable bit, ET2 = 0; Disable Timer 2 interrupt	0
4	ES	Serial Port Interrupt Enable, ES = 0; Disable serial port interrupt	0
3	ET1	Timer 1 Overflow Interrupt Enable, ET1 = 0; Timer 1 overflow interrupt disabled	0
2	EX1	External Interrupt 1 Enable, EX1 = 1; Enable external Interrupt 1	1
1	ET0	Timer 0 Overflow Interrupt Enable, ET0 = 0; Disable Timer 0 overflow interrupt	0
0	EX0	External Interrupt 0 Enable, EX0 = 0; External Interrupt 0 disable	0

Table 4.1: IEN0- Interrupt enable register 0 settings

IEN0 = 0x84;

2. TCON- Timer/counter control register:
Used to set Interrupt 1 trigger type to falling edge trigger.

Bit	Symbol	Description	Bit Value
7	TF1	Timer 1 overflow flag. Set by the hardware. Not used.	0
6	TR1	Timer 1 Run control bit, TR1 = 0; Timer 1 not used.	0
5	TF0	Timer 0 overflow flag. Set by the hardware. Not used.	0
4	TR0	Timer 0 Run control bit, TR0 = 0; Timer 0 not used.	0
3	IE1	Interrupt 1 Edge flag. Interrupt 1 Edge flag. Will be reset by the software at the end of the ISR.	0
2	IT1	Interrupt 1 Type control bit, IT1 = 1; Interrupt 1 falling edge trigger selected.	1
1	IE0	Interrupt 0 Edge flag. Set by the hardware. Not used.	0
0	IT0	Interrupt 0 Type control bit, IT0 = 0 or 1. Don't care.	0

Table 4.2: TCON- Timer/counter control register settings

TCON = 0x04;

Note: *Interrupt initialization should always be done in the following sequence:*
1. Select the trigger type (IT1) i.e. IT1 = 1;
2. Unmask/enable the specific interrupt (EX1) i.e. EX1 = 1;
3. Globally enable all unmasked interrupts (EA) i.e. EA = 1;
4. Make P3.3 INT 1 as input by writing 0 to P3.3 i.e. P3 = 0x08;
Can also be written as INT1 = 0; (refer to p89v51rx2.H)
5. In the ISR of Interrupt 1 clear the Interrupt 1 Edge flag (IE1) i.e. IE1 = 0;

Format for writing Interrupt Service Routine

After initializing interrupts, the next step will be to define the Interrupt Service Routine (ISR). The syntax to define the ISR is specific to the compiler and will vary if different compiler is used. The syntax acceptable in 'Keil' is as follows:

void int1_isr(void)**interrupt** 2

*Here the **'interrupt'** function attribute specifies that the associated function is an ISR. The interrupt attribute takes as an argument an integer constant (in this case 2). A table giving **'Interrupts to Integer constant'** is show below*

Interrupt	Integer Constant
EXTERNAL_INT 0	0
TIMER/COUNTER 0	1
EXTERNAL_INT 1	2
TIMER/COUNTER 1	3
SERIAL PORT	4

Table 4.3

Example code:
(Also available in the "Experiments \ 3 Position Control Using Interrupt 1" folder in the documentation CD)

This code counts pulses from left shaft encoder, checks if a particular distance has been covered by the robot, and stops the robot after this distance is traveled.

```
#include <intrins.h>
#include "p89v51rx2.h"


//program specific declarations
sbit LB=P1^0; //LEFT BACK Pin
sbit LF=P1^1; //LEFT FRONT Pin
sbit RF=P1^2; //RIGHT FRONT Pin
sbit RB=P3^4; //RIGHT BACK Pin

sbit left_velocity=P1^3; //Left velocity control pin
sbit right_velocity=P1^4; //Right velocity control
pin

unsigned int left_shaft_count=0; //left position
encoder count

void int1_setup()
{
TCON = 0x04; // set int 1 trigger type
IEN0 = 0x84; // enable all ISR servicing and
enable int 1
P3 = 0x08; // set P3.3 (INT 1) as input port
}

//External Interrupt 1 ISR
void int1_isr(void)interrupt 2
{
 left_shaft_count ++;   //Increment Encoder count
 IE1=0; //Reset Interrupt 1 flag
}

void forward(void) //go forward
{
RF=1;
LF=1;
RB=0;
LB=0;
}

void stop(void) //stop
```

```
{
RF= 0;
LF= 0;
RB= 0;
LB= 0;
}

void main()
{
unsigned int reqd_shaft_count_int=0;
unsigned int distance=0;
unsigned int left= 0;
distance=100; //Enter here the distance to be
traveled in mm

//This equation will calculate the count required
for distance to be traveled
 reqd_shaft_count_int =(unsigned int)(distance *
100 / 544);

//initialize external Interrupt 1
 int1_setup();

//set motor enable pins to 1
left_velocity =1;
right_velocity =1;

forward();//go forward

while(left_shaft_count<reqd_shaft_count_int);
//wait till required distance moved

 stop();
while(1);
}//main ends
```

Explanation of example code:

The function int1_setup() initializes the external Interrupt to be triggered at falling edge of shaft encoder pulse. int1_isr() is the external interrupt 1 ISR which increments the count: **'left_shaft_count'** everytime wheel moves one tick (i.e. 5.44 mm). While coming out of the ISR it also sets Interrupt 1 flag (IE1) to 0.

In the function main() required distance is converted in integer counts by using the formula reqd_shaft_count_int =(**unsigned int**)(distance * 100 / 544);

Here instrad of dividing by 5.44 we are dividing by 100/544. This is done to avoid floating point calculation and save RAM and computational resources.
The while() statement repeats continuously till the current wheel count is less than the required count. When current count is equal to or greater than the required count the while() statement becomes false and robot stops.

5. Timer / Counter Operations on the Robot

Timers/counters provide the basis for a number of indispensable tasks for the robot like, PWM generation, event capture, clock generation etc. P89V51RD2 based Fire Bird V uses them for the following purposes.

1. Serial Communication: For the transmit and receive baud rate generation (Timer 1)

2. PWM generation: Pulse Width Modulated pulses generated by the Programmable Counter Array (PCA) of the Microcontroller for motor velocity control.

3. Position control: Timer 1 is used as external event counter to measure pulses coming from the position encoder.

5.1 Timers / Counters:

Timers can be used in two different modes:

Timer mode: Acts as a binary up counter, which runs on internal clock frequency of the Microcontroller, thus counting the time periods applied to their input.

Counter mode: Acts as a binary up counter, counting certain events or occurrences at their input these events/occurrences appear as logic level changes at timer input which are detected by the internal circuit.

The P89V51RD2 has 3 timers/counters, Timer 0, 1, 2 and the Programmable Counter array (PCA). All of these are 16 bit up counters, i.e. they count from 0 to 65,535 then roll over to 0 and begin counting from the start again. Interrupts are also provided which are triggered when timer/counter rolls over.

Timer 0 and 1 can be operated in one of 4 available modes, i.e. mode 0,1,2 or 3 with each mode providing different functionality, like 8-bit timer, 16-bit timer, timer with automatic reload etc.

Timer 2 can also be operated in one of 4 available modes, which are different from those of Timers 0 & 1. These modes provide special functions, like Capture, Auto reload, Clock-out & baud rate generator for serial communication.

The PCA has a special 16-bit Timer that has five 16-bit capture/compare modules associated with it. Each module can be operated in one of 4 modes i.e. rising and/or falling edge detect (event counter), software timer, high speed output & pulse width modulator (PWM).

Note: The five modules of PCA have the same timer base, and thus are not separate timers.

5.2 Velocity control of motors using Pulse Width Modulation (PWM)

PWM (Pulse Width Modulation) is a process in which duty cycle of square wave is modulated to control motor velocity. Duty cycle is the ratio of **'T-ON/ T'**. Where '**T-ON'** is ON time and **'T'** is the time period of the wave. Power delivered to the motor is proportional to the **'T-ON'** time of the signal. PWM is used to control total amount of power delivered to load without power losses which generally occur in resistive loading. In case of PWM the motor reacts to the time average of the signal.

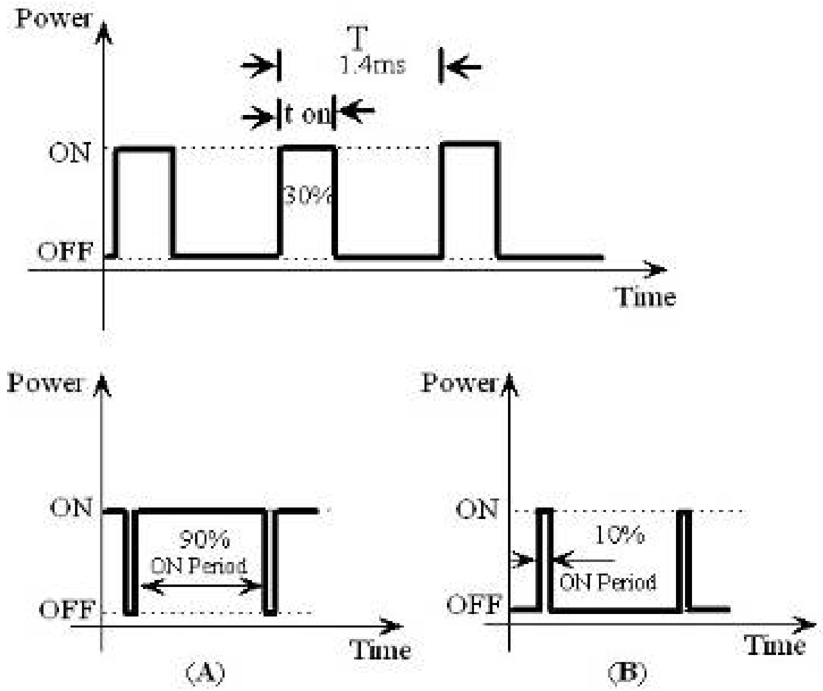


Figure 5.1: Pulse Width Modulation (PWM) for velocity control

In case (A), ON time is 90% of time period. This wave has more average value. Hence more power is delivered to the motor. In case (B), the motor will run slower as the ON time is just 10% of time period.

5.3 Velocity control by PWM using Programmable Counter Array (PCA)

All the 5 modules associated with the PCA can be used to generate PWM pulses by programming the PCA to operate in mode 4. All of the modules operate at the same frequency as they have the same timer base. However the duty cycle of each module can be varied using the capture registers CCAPnL and CCAPnH (where n denotes module number) associated with each module. The programmed module runs in 8-bit resolution in the PWM mode. The PCA continuously compares its current count value with the value loaded in the CCAPnL register. If the current count is less than the value in CCAPnL the output (Pin CEXn) is low, when it is equal to or greater than value in CCAPnL, the output (Pin CEXn) will be high. When the PCA count overflows from FF to 00, CCAPnL is again loaded with the value stored in CCAPnH. We are using CEX0 and CEX1 modules to generate PWM for the left and right motors.

PIN NO:	Pin name	USED FOR
1	T2/P1.0	Left Motor Direction Control(C)
2	T2EX/P1.1	Left Motor Direction Control(D)
3	ECI/P1.2	Right Motor Direction Control(B)
4	CEX0/P1.3	Pulse width modulation for the left motor (velocity control)
5	CEX1/SS/P1.4	Pulse Width Modulation for the right motor (velocity control)
14	T0/P3.4	Right Motor Direction Control(A)

Table 5.1: PIN functions of the microcontroller for motion control

Using PCA to generate PWM
PCA needs initialization before they can be used to generate PWM. Initializing PCA involves programming the following registers.

The following registers have to be programmed in order to use the PCA for PWM generation:

3. CMOD- PCA Counter Mode Register:
Used to select clock frequency source for PCA counter to generate PWM and disable counter overflow interrupt.

Bit	Symbol	Description	Bit Value
7	CIDL	Counter Idel Control: CDIL = 1; PCA counter gated off during idle	1
6	WDTE	Watchdog Timer Enable: WDTE = 0; disable Watchdog timer	0
5	-	Reserved for future use. Should be set to 0	0
4			0
3			0
2	CSP1	CSP1 = 0; CSP0 = 0; Clock source: f_{osc} / 6 (f_{osc} = 11.0592MHz)	0
1	CSP0		0
0	ECF	PCA Enable Counter Overflow Interrupt: ECF = 0; Disables CF bit in CCON to generate an interrupt.	0

Table 5.2: CMOD- PCA Counter Mode Register settings

CMOD = 0x80;
PWM frequency: Fosc / (6 x 255) = 7228.23Hz

2. CCON - PCA Counter Control Register
Used to start PCA counter

Bit	Symbol	Description	Bit Value
7	CF	PCA Counter Overflow Flag.	0
6	CR	PCA Counter Run Control Bit: CR = 1; Turn on PCA counter	1
5	-	Reserved for future use. Should be set to 0	0
4	CCF4	PCA Module 4 Interrupt Flag. (not used)	0
3	CCF3	PCA Module 3 Interrupt Flag. (not used)	0
2	CCF2	PCA Module 2 Interrupt Flag. (not used)	0
1	CCF1	PCA Module 1 Interrupt Flag. (not used)	0
0	CCF0	PCA Module 0 Interrupt Flag. (not used)	0

Table 5.3: CCON - PCA Counter Control Register settings

CCON = 0x40;

3. CCAPMn - PCA modules compare/capture register
In our case n = 0 for left motor and 1 for right motor. Used it enable PWM generation.

Bit	Symbol	Description	Bit Value
7	-	Reserved for future use. Should be set to 0	0
6	ECOMn	Enable Comparator. ECOMn = 1; enable the comparator function.	1
5	CAPPn	Capture Positive, CAPPn = 0; Disable positive edge capture.	0
4	CAPNn	Capture Negative, CAPNn = 0; Disable negative edge capture.	0
3	MATn	Match, MATn = 0; Disable flagging of the interrupt in CCFn bit in CCON register.	0
2	TOGn	Toggel, TOGn = 0; Disable toggeling of the CEXn when match of the PCA counter with this module's compare/capture register occours.	0
1	PWMn	Pulse Width Modulation Mode. PWMn = 1; Enables the CEXn pin to be used as a pulse width modulated output.	1
0	ECCFn	Enable CCF Interrupt, ECCFn = 0; Disables compare/capture flag CCFn in the CCON register to disable generation of an interrupt.	0

Table 5.4: CCAPMn - PCA modules compare/capture register settings

CCAPM1 = 0x42;
CCAPM0 = 0x42;

4. CCAPnH and CCAPnL registers for setting PWM duty cycle
There are two additional registers associated with each of the PCA modules. They are CCAPnH and CCAPnL and these are the registers that store the 16-bit count when a capture occurs or a compare should occur. When these modules are used in the PWM mode these registers are used to control the duty cycle of the output. We are using 8bit PWM mode.

To set the velocity of the left motor:
CCAP0L=0x00;
CCAP0H=0xFF; // Left motor duty cycle. 0 to 0xFF gives 0 to 100% duty cycle i.e. velocity from 0 to maximum speed. Any value in between 0x00 and 0xFF will give intermediate velocity.

To set the velocity of the right motor:
CCAP1L=0x00;
CCAP1H=0xFF; // Right motor duty cycle. 0 to 0xFF gives 0 to 100% duty cycle i.e. velocity from 0 to maximum speed. Any value in between 0x00 and 0xFF will give intermediate velocity.

Note:
CEX0 and CEX 1 modules will have the same PWM frequency of output because they all share one and only PCA timer. The duty cycle of each module is independently variable using the module's capture register CCAPnL. Here n is 0 or 1. When the value of the PCA CL SFR is less than the value in the module's CCAPnL SFR the output will be low, when it is equal to or greater than the output will be high. When CL overflows from FF to 00, CCAPnL is reloaded with the value in CCAPnH. this allows updating the PWM without glitches. Hence although CCAPnL register is used for comparison we load value for PWM in the CCAPnH register. This is also known as double buffering scheme. Also as value of the PCA CL SFR is less than the value in the module's CCAPnL SFR the output will be low, when it is equal to or greater than the output will be high we get inverted PWM output. I.E. when we load CCAPnH=0xFF; (where

n = 0 for left and n = 1 for right) output is 0% duty cycle and motors will not run and when value is 0x00 output will be of 100% duty cycle. I.E. motors will run at the full speed.

Example Code: This code generates PWM signal for motor velocity control
(Also available in the "Experiments \ 4 PWM using PCA (Timers and Counters)" folder in the documentation CD)

Code:

```
#include <intrins.h>
#include "p89v51rx2.h"

sbit LB=P1^0;   // Left Back
sbit LF=P1^1;   // Left Forward
sbit RF=P1^2;  // Right Forward
sbit RB=P3^4;  // Right Back

//start left motor with velocity vel(compare value)
void left_motor_vel(unsigned int vel)
{
CCAP0H=vel;
}
//start right motor with velocity vel(compare value)
void right_motor_vel(unsigned int vel)
{
CCAP1H=vel;
}
//these are the direction functions
void forward(void) //go forward
{
RF=1;
LF=1;
RB=0;
LB=0;
}
//initialise programmable counter array to generate
pwm
void pca_init(void)
{
//80 sets PCA counter to run at Fosc/6
 CMOD=0x80;
//start PCA counter
 CCON=0x40;
//Left motor duty cycle
 CCAP0L=0x00;
 CCAP0H=0xFF;
//Right motor duty cycle
 CCAP1L=0x00;
 CCAP1H=0xFF;
//enable PWM and ECOM bits for left motor
 CCAPM0=0x42;
//enable PWM and ECOM bits for right motor
CCAPM1=0x42;
}

void main()
{
pca_init();
//set the motor velocities over here. Use only values
in the range 0x00 – 0xFF
 left_motor_vel(0x1F);
 right_motor_vel(0x1F);
forward();
  while(1);
}
```

Explanation of example Code:
The pca_init() function sets the appropriate registers for inverted PWM generation using module 0 and module 1 of the Programmable Counter Array. The left_motor_vel() and right_motor_vel() functions load the required value into CCAPnH for velocity control. These values can be changed by passing the new value during function call in main(). CCAPnL will be loaded with the new value at the counter overflow immediately following the change. Since the output is low for counter value less than CCAPnL value, speed of the motors is more at lower values of CCAPnH than at higher values, for eg.. motor will run faster at 0x0F than at 0x5F.

5.4 Position control of the robot using Timer1 as counter

Output of the right position encoder is connected to the input pin of Timer 1 (T1 – P3.5). Timer 1 is used in the counter mode to count the pulses from the right position encoder. Timer 1 is set to operate as an event counter. It thus increments at every encoder pulse. By setting Timer 1 to operate in Mode 2 it acts as an 8 bit up counter.

For initializing Timer 1 as a counter the following registers have to be programmed

1. TMOD – Timer/counter mode control register

Bit	Symbol	Description	Bit Value
7	T1GATE	Gating control, T1GATE = 0; Timer 1 will be enabled only when TR1 bit is set to 1.	0
6	T1C/T^{-1}	Gating Timer or Counter Selector, T1C/T-1 = 1; Set for counter operation. Clock sourse is from the pin P3.5	1
5	T1M1	Timer 1 mode select, T1M1 = 1; T1M0 = 0; Timer 1 in 8 bit auto reload mode.	1
4	T1M0		0
3	T0GATE	Not used. Set to 0.	0
2	T0C/T^{-1}	Not used. Set to 0.	0
1	T0M1	Not used. Set to 0.	0
0	T0M0	Not used. Set to 0.	0

Table 5.5: TMOD – Timer/counter mode control register

TMOD = 0x60;

4. TCON – Timer/Counter control register:

Used to start Timer 1

Bit	Symbol	Description	Bit Value
7	TF1	Timer 1 overflow flag. Set by the hardware. Cleared by the hardware when processor vectors to Timer 1 ISR or can be cleared by the software.	0
6	TR1	Timer 1 Run control bit, TR1 = 1. Start Timer 1.	1
5	TF0	Timer 0 overflow flag. Not used. Set to 0.	0
4	TR0	Timer 0 Run control bit, Not used. Set to 0.	0
3	IE1	Interrupt 1 Edge flag. Not used. Set to 0.	0
2	IT1	Interrupt 1 Type control bit. Not used. Set to 0.	0
1	IE0	Interrupt 0 Edge flag. Not used. Set to 0.	0
0	IT0	Interrupt 0 Type control bit. Not used. Set to 0.	0

Table 5.6: SCON – Serial port control register settings

TCON = 0x40;

Note: While initializing Timer 1 initialize TH1 and TH0 to 0.

Example Code: Robot position control using Timer 1
(Also available in the "Experiments \ 5 Position Control Using Timer 1 as Counter" folder in the documentation CD)

```
#include <intrins.h>
#include "p89v51rx2.h"

//The appropriate bits have been defined as per
the logic table for motion control
sbit LB=P1^0;   // Left Back
sbit LF=P1^1;   // Left Forward
sbit RF=P1^2;  // Right Forward
sbit RB=P3^4;  // Right Back

sbit left_velocity=P1^3;
sbit right_velocity=P1^4;

unsigned int right_shaft_count=0; //pulse count
from right position encoder

void  timer1_setup(void)   //setting timer1 as
counter in mode 2
{
 TMOD=0x60; // Timer 1 in 8 bit external counter
mode
 TH1=0; //reset counter value to 0
 TL1=0; //reset counter value to 0
}

//motor control
void forward(void) //go forward
{
RF=1;
LF=1;
RB=0;
LB=0;
}

void stop(void) //stop
{
RF= 0;
LF= 0;
RB= 0;
LB= 0;
}


void main()
{

 unsigned int distance=0;
 unsigned int reqd_shaft_count_int=0;

timer1_setup(); //Timer 1 initialization

distance=100;  //Enter here the distance to be
traveled in mm

//This equation will calculate the count required
for distance to be traveled
reqd_shaft_count_int=(unsigned int)
(distance*100 / 544);

left_velocity=1; //enable left motor
right_velocity=1; //enable left motor

forward();//move forward
TR1=1;       //Start the timer
//loop until the required distance is travelled
 while(reqd_shaft_count_int > right_shaft_count)
 {
  right_shaft_count = TL1; //store the contents of
TL1 in left_shaft_count
 }

stop();//stop moving
while(1);
}//main ends
```

Example code Explanation:
The above code initializes the Timer 1 to run as an 8-bit counter. Pulses at the input of Timer 1 increment the timer and the value of the present count is stored in TLI register of Timer 1 (TH1&TL1 in case of 16 bit mode). In the function main ( ) a while ( ) loop continuously compares the counter count value with the required count. When the counter count value i.e. 'right_shaft_count' is equal to or exceeds the required count, the while () loop becomes false and the motor stops.

6. LCD Interfacing with the robot

6.1 Interfacing LCD with Microcontroller

To interface LCD with the microcontroller requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To save number of I/Os required for LCD interfacing we can use 3 control signals with 4 data lines. This is known as 4 bit mode and it requires 7 I/O lines. In this mode upper nibble and lower nibble of commands / data needs to be sent separately. We are using 4 bit interfacing mode to reduce number of I/O lines. Figure 6.1 shows basic LCD interfacing.

Fire Bird V P89V51RD2 uses the 4-bit interfacing mode. In this mode higher nibble and lower nibble of commands/data set needs to be sent separately.

Control lines:
LCD requires 3 control lines known as Enable (EN), Register Select (RS) and Read / Write (R/W)

1. Enable (EN):
This control line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again.

2. Register Select (RS):
Register Select control line is connected to P2.6. When RS is low (0), the data is treated as a command or special instruction by the LCD (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is treated as text data which should be displayed on the screen.

3. Read / Write (RW):
Read/Write control line is connected to P2.5. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading from) the LCD.

The data bus is bidirectional, 4 bit wide and is connected to PORT2 P2.0 to P2.3 of the microcontroller. The MSB bit (DB7) of data bus is used as a Busy flag. When the Busy flag is 1, the LCD is in internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/W = 1, the Busy flag is output on DB7. The next instruction must be written after ensuring that the busy flag is 0.

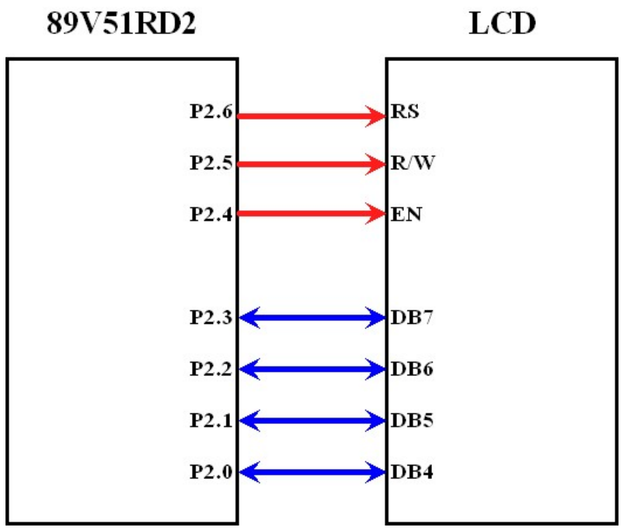


Figure 6.1: LCD interfacing with the microcontroller

Microcontroller	**LCD PINS**	**Description**
VCC	VCC	Supply voltage (5V).
GND	GND	Ground
P2.4	E (Control line)	Enable
P2.5	-R/W (Control line)	-READ /WRITE
P2.6	RS (Control line)	Resistor select
P2.0 to P2.3	D4 to D7 (Data lines)	Bidirectional Data Bus
	LED+, LED-	Backlight control

Table 6.1 LCD Pin mapping and functions

Communicating with the LCD:
Communication is done in the following sequence of steps
1. Select the register: Command register (RS = 1) or Data register (RS = 0)
2. Select read/write mode: Read (RW = 1) or Write (RW = 0)
5. Send or Receive byte from LCD.
4. Enable the LCD: (E = 1)
5. Disable the LCD: (E = 0)

Remember that data/command is clocked in to or out from the LCD only at the High to Low transition of the E pin.

The following timing diagram shows further explains the sequence of events for a write operation in the data register operation of the LCD.

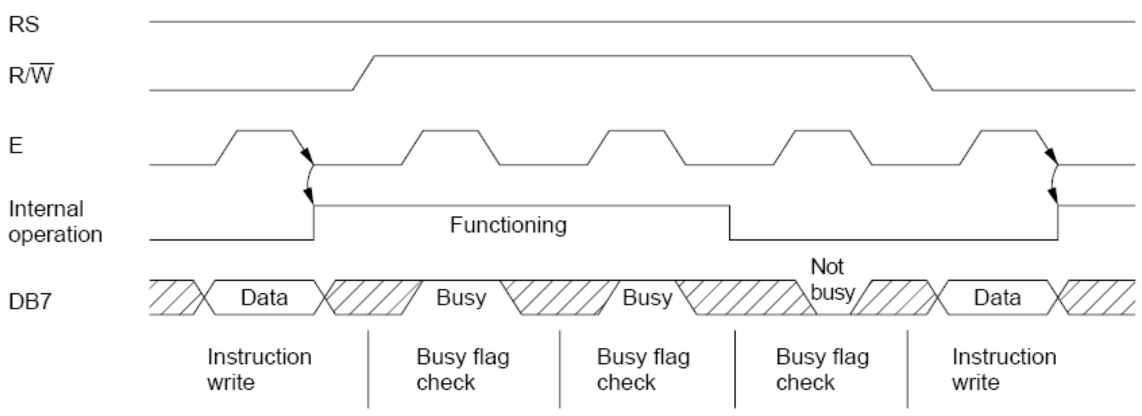


Figure 6.2: LCD timing diagram

Note:
LCD operates at slower speed than the processing speed of the microcontroller. In order to avoid sending data / commend before previous process is completed either busy flag (LCD DB7) should be checked before sending new data or inserting a finite delay in the code.

Initializing the LCD:
Before we can display any data on the LCD we need to initialize the LCD for proper operation.

1. The first instruction we send must tell the LCD whether we'll be communicating with it using an 8-bit or 4-bit data bus. Remember that the RS line must be low if we are sending a command to the LCD.
2. In the second and third instruction we reset and clear the display of the LCD.
3. The fourth instruction sets the cursor to move in incremental direction.
4. In fifth and sixth instruction we turn ON the display and place the cursor at the start.
LCD initialization is done in the above manner in the Init_LCD( ) function.

6.2 Application Example for displaying string on the LCD

Application example to display string on the two lines of the LCD is located in "Experiments \ 6 LCD Interfacing String Display".

Code is not written here because of large size of the code.

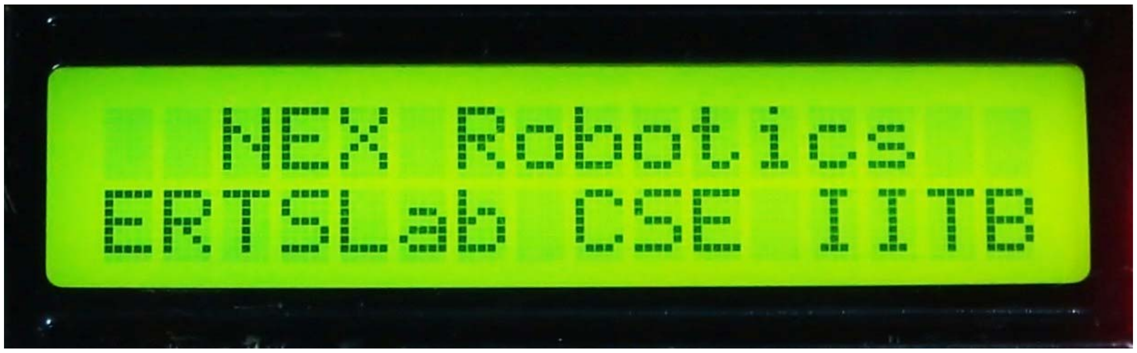


Figure 6.3: LCD string display

Function description of the Example Code

1. delay_ms(unsigned int ms):
Parameter passed: delay in milliseconds

Return data: None

Description:. The delay_ms(unsigned int ms) function introduces a delay of ms milliseconds (where ms is the parameter passed to it by the calling function).

2. ready ():
Description: The status of the LCD can be determined by checking the busy bit (DB7 of LCD). When the busy bit is 1 it means that the LCD is busy with internal operation. No further command/data can be accepted while the LCD is busy with internal operation. Thus the ready () function stalls the program till the busy bit goes low and only then proceeds with further instructions.

3. swap ()
Parameter passed: the data/command to be sent to LCD

*Return data: T*he upper nibble of the data to be sent is shifted into the lower nibble and returned to the calling function.

Description:
- Since Fire Bird V uses the LCD in 4 bit mode, 8 –bit data has to be sent separately as upper nibble and lower nibble.
- The swap () function uses an inbuilt right shift function _cror_(), to rotate the data and bring upper nibble to the lower nibble.
- The shifted data is saved in a temporary variable after masking the upper nibble.

- The returned variable is only the upper nibble, however placed in the least significant 4 bits.
- The swap () function thus separates the upper nibble from the lower nibble and returns the upper nibble.

4. commandsend():

Parameter passed: The command to be sent to set/modify LCD display properties is sent by the calling function to this function.

Return data: None

Description:

- The function first calls the swap function to separate out the upper nibble from the lower nibble of the data.
- The function then selects Instruction register(RS=0), Write function(RW=0) and sends data on the Port lines. A High to Low transition at the enable (E) pin gates in the command to the LCD.
- The function then send the lower nibble in the same way.

5. datasend():

Parameter passed: The data to be sent to appear on LCD is sent by the calling function to this function.

Return Data: None

Description:

- Similar to the commandsend() function, the datasend() function displays data onto the LCD screen.
- The datasend() function also calles the swap() function to separate out the upper nibble from the lower nibble.
- The function then selects data register (RS=1), write function(RW=0) and then outputs data onto the port lines. A High to Low transition at the enable (E) pin gates in the data to the LCD.
- The function then sends in the lower nibble in the same manner.

6. lcd_init():

Description:

- This function initializes the display sets the LCD to 4-bit mode (or 8-bit mode incase all 8 data lines of LCD have been used) and sets display properties like display ON/OFF, cursor ON/OFF, right/left shift, etc.

- The delay() function is used in this routine to allow the LCD driver sufficient time for proper initialization. This delay is as per the LCD driver datasheet (refer: CD_hd44780u_ datasheet provided in the CD).

- Sending Instructions to the LCD driver should follow the following sequence.
  1. Select the register (Command Register: RS=0, Data register: RS=1)
  2. Select Read/Write function (Read: RW=1; Write:RW=0)
  3. Place command/data onto the data lines (DB4 to DB7) through the port connected to them (P2.0 to P2.6)
  4. Assert the enable pin (EN=1)
  5. De-assert enable pin (EN=0)

- Set LCD ON; CURSOR OFF; BLINK ON.

Note: LCD accepts new command/data only on High to Low transition of the enable pin

7. lcd_display():

Description:

- Clear display and initialize auto increment cursor position

- The function defines two 16 element arrays which can store character strings.

- The first array lcd_data1 [ ] is displayed on the first line of the LCD and the second lcd_data2 [ ] on the second line.

6.3 Application Example for displaying character array onto the LCD

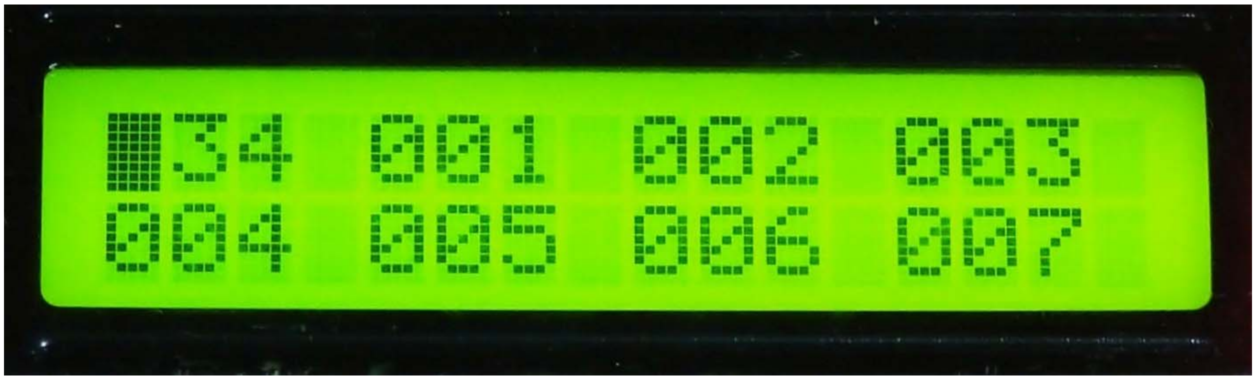
Application example to display character array on the two lines of the LCD is located in "Experiments \7 LCD Interfacing Data Array Display".

Code is not written here because of large size of the code.

'lcd_array_disp.h' will be extensively used by the codes involving Analog to Digital Conversion. All the analog data will be displayed on the LCD.

Using 'lcd_array_disp.h' you can show data from the globally defined unsigned character array 'data_array[8]' on the LCD in 3 digit group by calling function 'lcdprint(data_array)'.

LCD must be initalised in the 'main' function by calling function 'lcd_init()'.



```
█34 001 002 003
004 005 006 007
```

Figure 6.4: LCD array display

Explanation of Sample code:

- The above function is a very generic routine which display an array of maximum 8 elements.(each element being a 3 digit number)
- The function 'lcdprint ( )' accepts the name of the array as argument.
- Now the code displays this data character by character onto the LCD screen. Nested if statement in the 2nd for loop keeps track of the cursor position. When first line is full, it positions the cursor at the beginning of the 2nd line and displays further data from there.
- In the following chapter on ADC conversion, we will be using a more expanded from of the above function to display ADC values onto the LCD screen.

Note:
In the folder 'Experiments' header files 'lcd_array_disp.h' and 'lcd_display.h' located when can be used for displaying data on the LCD.

7. Analog to Digital Conversion

Fire Bird V P89V51RD2 has many analog sensors like, white line sensors, IR Proximity sensors sensors, Sharp IR Range sensors and battery voltage sensing etc. P89V51RD2 does not have onboard ADC (Analog to Digital Converter) for capturing analog signals. These analog signals are converted to digital form using ADC0808 which is interfaced with the microcontrollers PORT0.

7.1 ADC0808

The ADC0808 is an 8-bit analog to digital converter with a 8 channel analog multiplexer. The 8-bit A/D converter uses the successive approximation method for analog to digital conversion.

Functional Description:

The ADC0808 can be controlled with the help of 4 control signals:

- ALE (Address Latch Enable): On a Low to High transition at this pin the ADC latches the channel address on its multiplexed address lines.
- START: A High to Low transition at this pin will start the analog to digital conversion
- OUTPUT ENABLE: A High signal at this pin will latch the conversion output onto the output lines, which can be read by the Microcontroller.
- EOC (End of Conversion): The EOC pin goes low after a conversion has completed

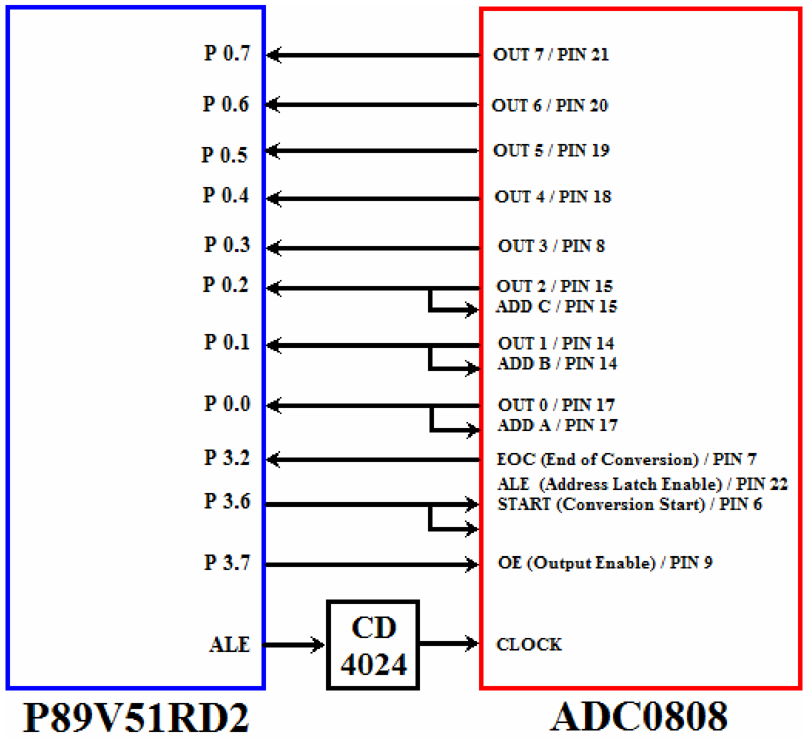


Figure 7.1: P89V51RD2 and ADC0808 interfacing

Note:

- As can be seen in above figure the three address lines have been multiplexed with three data lines so as to reduce the number of I/O lines required from the Microcontroller.
- ADC0808 need clock for operation. Clock is derived from ALE pin of the P89V51RD2 which is weakly pulsed at the 1/6th of the f_{osc} (11.0592MHz). It is divided by n using CD4024 frequency divider and provided to the clock of the ADC0808.

ADC Initialization and Conversion Sequence

An ADC conversion routine should be written by strictly following this sequence:

1. Select the ADC channel number and write it to the P0 which is connected to the address lines of the ADC.
2. Low to High transition at the ALE pin to latch the address into the ADC.
3. High to Low transition at the START pin to start ADC conversion.
4. Give a finite delay before next Instruction to allow the ADC enough time for conversion.
5. Set OUTPUT ENABLE pin of ADC 0808 to latch conversion result onto output lines.
6. Read ADC output into a variable
7. Reset OUTPUT ENABLE pin

Note:

- In FIRE BIRD V P89V51 robot START and ALE pin are tied together and connected to PORT 3, PIN 6 (P3.6). Once the Analog channel number written to the PORT0 as per step 1, P.6 is set as given in step 2. This latches the address onto the multiplexed data bus. Then, for starting conversion this same pin is reset as per step 3, which triggers the conversion process. The ALE well as START signals are edge triggered.

- Typically ADC 0808 requires 100 microseconds to complete one analog to digital conversion. Alternately the software can make use of external interrupt 0 connected to EOC pin of ADC 0808 to determine the end of conversion and then proceed with further instructions.

ADC resolution:

The ADC resolution is defined as the lowest analog value that can be represented by a corresponding unique digital value.
For an n-bit ADC resolution is given by:

$$\textbf{Resolution} = \frac{\textbf{Vref}}{\textbf{2\^{}n-1}}$$

Where **Vref** : Reference voltage used in successive approximation technique
n: number of bits used by ADC to store digital data.

For Fire Bird V,
Vref =5 V; n =8;

Thus the ADC resolution for Fire Bird V is 19.607 mV. It means that at the least a change of 19.607mV at the analog inputs can be detected by the ADC and represented by a change in the corresponding digital value.

7.2 Analog Sensors

Microcontroller	ADC 0808	Sensors
Pin 12 - INT0 / P3.2	Pin 7 - EOC (End of Conversion)	
Pin 16 - WR / P3.6	Pin 22 - ALE (Address Latch Enable)	
	Pin 6 - START (Conversion Start)	
Pin 17 - RD/P3.7	Pin 9 - OE (Output Enable)	
Pin 32 - P0.7	Pin 21 - 2^{-1} (MSB)	
Pin 33 - P0.6	Pin 20 - 2^{-2}	
Pin 34 - P0.5	Pin 19 - 2^{-3}	
Pin 35 - P0.4	Pin 18 - 2^{-4}	
Pin 36 - P0.3	Pin 8 - 2^{-5}	
Pin 37 - P0.2	Pin 15 - 2^{-6} / ADD C	
Pin 38 - P0.1	Pin 14 - 2^{-7} / ADD B	
Pin 39 - P0.0	Pin 17 - 2^{-8} (LSB) / ADD A	
Pin 40 - Vcc	Pin 11 - VCC (5V)	
	Pin 5 - IN 7	IR Proximity sensor 4
	Pin 4 - IN 6	Right White Line
	Pin 3 - IN 5	Centre White Line
	Pin 2 - IN 4	Left White Line
	Pin 1 - IN 3	IR Proximity sensor 2
	Pin 28 - IN 2	Front Sharp IR range sensor 3
	Pin 27 - IN 1	Battery Voltage
	Pin 26 - IN 0	IR Proximity sensor 3

Table 7.1 ADC0808's connection with P89V51RD2 microcontroller and sensors

7.3 Application Example for converting Analog data to digital form and displaying it onto the LCD

Application example to display string on the two lines of the LCD is located in “Experiments \ 8 ADC Sensor Display on LCD” folder in the documentation CD.

Code is not written here because of large size of the code.

Explanation of example code:

Important Functions in the code:

1. ADC_conversion():

Parameter passed: ADC Channel number

Return value: Digital value of the sensor at that channel

Description:

- This function accepts the ADC channel number as an input from calling function and selects appropriate channel by writing this channel number to P0 and asserting then start_conv bit (Low- High transition on ALE pin) High. The addressed is thus latched onto the ADD pins of ADC 0808.
- Then the function de-asserts the start_conv bit (High-Low transition on START pin) and ADC conversion begins.
- The function delay_adc_ms() provides a finite delay for conversion to complete. This can also be achieved by polling the EOC (End of Conversion) pin of ADC 0808 connected to Pin 12 of Microcontroller (External Interrupt 0 may also be used) , and checking for a High to Low transition.
- After this delay the function sets Port 0 as an input Port, asserts the Output Enable (OE) pin of ADC high, and reads the value of Port 0 into a variable.
- The function then de-asserts the OE pin and returns the ADC value to the calling function.

Other functions are covered in the chapter 6.

Data is displayed onto the LCD screen as shown below:

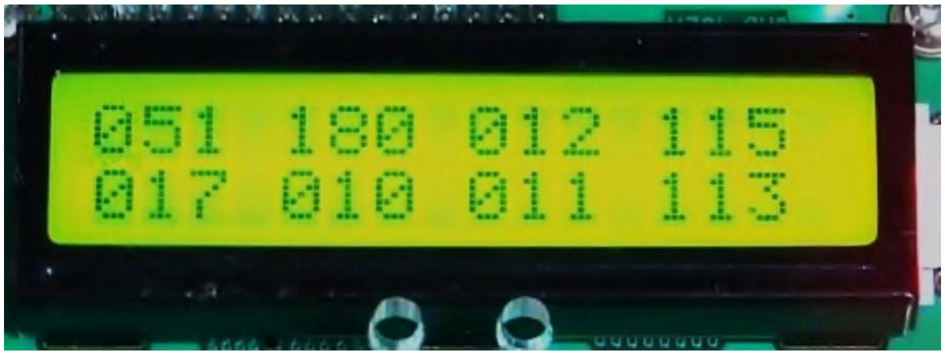


Fig 7.2 Sensor display on the LCD

ADC Channel No.	Position of value on LCD Screen	Corresponding Sensor
0	Top Row, 1st from left	IR Proximity sensor 3
1	Top Row,2nd from left	Battery voltage
2	Top Row, 3rd from left	Front Sharp IR Range sensor
3	Top Row, 4th from left	Left IR proximity sensor 2
4	Bottom Row, 1st from left	Left White Line sensor
5	Bottom Row, 2nd from left	Center White Line sensor
6	Bottom Row, 3rd from left	Right White Line sensor
7	Bottom Row, 4th from left	Right IR proximity sensor 2

Table 7.2 LCD dada display interpretation

Note:
All the sensor display is in 8bit unprocessed value.

To get the actual battery voltage use following formula:

Battery voltage = ((0.019607 * Battery voltage 8 bit value * (4700 ohms + 2200 ohms)) / 2200 ohms) + 0.7V

Battery voltage = (0.06149 * Battery voltage 8 bit value) + 0.7V

8. Serial Communication

The Fire Bird V can communicate with other robots / devices serially using either wired or wireless communication over serial port. We are using asynchronous mode of operation for serial communication. In asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization.

8.1 Serial Communication using UART (Universal Asynchronous data Receive / Transmit)

UART configuration for serial configuration consists of two parts
1. Set the UART mode using SCON – Serial Control Register.
2. Select Baud rate using Timer 1 TMOD – Timer mode control register.

UART Modes:

The P89V51RD2 UART operates in 4 modes which differ from one another in terms of number of bits transmitted received and baud rate (rate of data transfer)

Mode 0: 8 bits are exchanged at a fixed baud rate of 1/6th of the Microcontroller clock

***Mode 1*:** 10 bits are exchanged (start bit and stop bit and 8 data bits) at variable baud rate determined by Timer 1 / 2 overflow rate.

***Mode 2*:** 11 bits are exchanged (a programmable bit in addition to start, stop and 8 data bits) at fixed frequency of 1/16 or 1/32 of the Microcontroller clock (as determined by the SMOD 1 bit in PCON-Power Control Register).

Mode 3: 11 bits are exchanged (same as Mode 2) variable baud rate as determined by Timer 1 / 2 overflow rate.

Modes are selected by programming the SCON –Serial Port Control Register.

8.2 Application Example for data transfer between PC & Microcontroller using serial port

Character sent from the PC via the keyboard to Fire Bird V can be viewed onto the ‘terminal’.
For directions on how to use the terminal refer section 6.2 from ‘Chapter-6 PC Based Control Using Serial Communication’ in the hardware manual.

Application example for data transfer between PC & Microcontroller using serial port is located in “Experiments \ 9 Serial Communication echo” folder in the documentation CD.

The following registers needs to be programmed to enable serial communication:

5. SCON – Serial port control register:

Used to select UART mode

Bit	Symbol	Description	Bit Value
7	SM0/FE	SM0 = 0; SM1 = 1; UART mode 1. One start and stop bits + 8 data bits. Clock source is timer1	0
6	SM1		1
5	SM2	Multiprocessor communication mode, SM2 = 0; Disabled	0
4	REN	Enables serial reception, REN = 1; Enabled.	1
3	TB8	Not used. Set to 0.	0
2	RB8	Not used. Set to 0.	0
1	TI	Transmit interrupt flag. Set by hardware at the end of the stop bit in mode 1. Must be cleared by software.	0
0	RI	Receive interrupt flag. Set by hardware at the end of the stop bit in mode 1. Must be cleared by software.	0

Table 8.1: SCON – Serial port control register settings

SCON = 0x50;

6. TMOD – Timer/Counter mode control register:

Used to set timer 1 in mode 2

Bit	Symbol	Description	Bit Value
7	T1GAT E	Gating control, T1GATE = 0; Timer 1 is enabled whenever 'TR1' control bit is set.	0
6	T1C/T-1	Gating Timer or Counter Selector, T1C/T-1 = 0; Timer operation is selected.	0
5	T1M1	T1M1 = 1; T1M0 = 0; Timer 1 in mode 2. 8bit auto reload mode. 'TH1' holds a value which is to be reloaded into 'TL1' each time it overflows.	1
4	T1M0		0
3	T0GAT E	Not used. Set to 0.	0
2	T0C/T	Not used. Set to 0.	0
1	T0M1	Not used. Set to 0.	0
0	T0M0	Not used. Set to 0.	0

Table 8.2: SCON – Serial port control register settings

TMOD = 0x20;
TMOD - Timer/Counter mode control register
Baud Rate Generation:

Setting the Timer Mode

For Modes 1 and 3 the baud rate is determined by the overflow rate of Timer 1 or Timer 2. (we will be using Timer 1 in this case)

- Timer 1 is to be set to operate in Mode 2 (8-bit, Auto re-load mode). In this mode after overflow TL1 is re-loaded with the value stored in TH1 and timer starts counting again.

- Overflow rate can be varied by changing the value placed in TH1.

Determining re-load value in TH1

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON.7 is clear).

TH1 = 256 - ((Crystal frequency / 384) / Baud rate)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal frequency / 192) / Baud rate)

In Fire Bird V P89V51RD2 11.059 MHz crystal is used.

Hence TH1 = 256 - ((11.0592MHz / 384) / 9600)

TH1 = 253 decimal or 0xFD

7. TCON – Timer/Counter control register:

Used to start timer 1

Bit	Symbol	Description	Bit Value
7	TF1	Timer 1 overflow flag. Set by the hardware. Cleared by the hardware when processor vectors to Timer 1 ISR or can be cleared by the software.	0
6	TR1	Timer 1 Run control bit, TR1 = 1. Start Timer 1.	1
5	TF0	Timer 0 overflow flag. Not used. Set to 0.	0
4	TR0	Timer 0 Run control bit, Not used. Set to 0.	0
3	IE1	Interrupt 1 Edge flag. Not used. Set to 0.	0
2	IT1	Interrupt 1 Type control bit. Not used. Set to 0.	0
1	IE0	Interrupt 0 Edge flag. Not used. Set to 0.	0
0	IT0	Interrupt 0 Type control bit. Not used. Set to 0.	0

Table 8.3: SCON – Serial port control register settings

TCON = 0x40;

Explanation of Sample code:

- The uart_setup() function initializes registers for UART mode setup and baud rate generation.
- The function RxData() is called by the function main() only when RI (Receive Interrupt flag) is set, i.e. when some data is available in the serial buffer (SBUF). RxData() stores the value in SBUF into a variable and returns it to the calling function.
- Now main() calls the data transmit function TxData(), and passes the received data as argument. TxData () then accepts this data and puts it into the serial buffer

(SBUF). It waits till the Transmit Interrupt flag (TI) is set i.e. it waits till data transmission is complete and returns control to main()

Note:
RI (receive interrupt flag) & TI (transmit interrupt flag) are set by hardware when data receive or data transmit respectively is complete. They are used by the microcontroller to keep track of when UART has completed receive/transmit of previous data is ready to receive/transmit new data. However in order to use them for further data transfer they have to be reset every time by user software i.e. the hardware does not reset these flags.