Robotics experiment with
MCS-51 microcontroller

based-on Robo-51 robot kit

(C) Innovative Experiment Co., Ltd.
Contents

Chapter 1 Part list of Robo-51 and Introduce software tools.................5
  1.1 Robo-51 part list
  1.2 Hand tools for making robot kit
  1.3 Software development tools for Robot programming

Chapter 2 Introduction FLIP the in-system programming software ..........9

Chapter 3 The RBX-51AC2 controller board ..........................................25
  3.1 Introduce the RBX-51AC2 robot controller board
  3.2 Preparing the software development tools
  3.3 Interface LED programming
    Activity 1 : Driving the LED
  3.4 How to generate sound signal
    Activity 2 : Sound driving
  3.5 Displaying with LCD
    Activity 3 : LCD experiment

Chapter 4: Building Robo-51 kit..............................................................47
  4.1 Robo-51 features
    Activity 4 : Make the Robo-51
  4.2 DC motor driving programming
    Activity 5 : Driving the DC motor
  4.3 Robo-51 Library
  4.4 Robo-51 movement programming
    Activity 6 : Simple movement control of Robo-51

Chapter 5 : Sensor interfacing................................................................75
  5.1 Simple digital sensor interfacing
    Activity 7 : Reading the Switch with polling method
    Activity 8 : Reading the Switch with interrupt
  5.2 Analog sensor interfacing
    Activity 9 : Reading the Analog signal
Chapter 6 Line following mission ................................................................. 85

6.1 The Infrared reflector sensor
6.2 Reading the sensor value
   Activity 10 : Reading IR Reflector value
6.3 Robo-51 versus the black line
   Activity 11 : Robo-51 finds the black line
   Activity 12 : Robo-51 PING-PONG
6.4 Robo-51 moves along the line programming
   Activity 13 : Robo-51 tracks the black line

Chapter 7 Remote control experiment ...................................................... 101

7.1 ER-4 Infrared remote control 4 channels
7.2 The 38kHz Infrared receiver module
7.3 Making the Remote control library file
   Activity 14 : Reading the ER-4 Remote control data
   Activity 15 : IR control Robo-PICA movement

Chapter 8 Contactless object detection of Robo-51 ................................... 111

8.1 Introducing ZX-08 Infrared object detector
8.2 Infrared objector library file
   Activity 16 : Testing Infrared objector
   Activity 17 : Contactless object detection robot
Robotics Experiment with MCS-51 microcontroller
Chapter 1
Part list of Robo-51 and Introduce software tools

1.1 Robo-51 part list

There is 3 groups as:

1.1 Robo-51 part list

1.2 Hand tools for making robot kit

1.3 Software development tools for Robot programming

1.1.1 Mechanical parts

**Motor Gearbox** - Uses a 4.5-9V and 180mA DC motor with a ratio of 48:1; torque 4kg/cm; comes with support plates.

**Track wheel set** - includes 3-length of Track wheel, many support wheels and sprockets, axels and shaft base

**The Plate set** and 3-types of the color-mixed Plastic Joiner (20 of Strength Joiner, 20 of Right-angle Joiner and 20 of Obtuse Joiner)

**Many sizes of Screw and Nut** (4 of 3x6mm., 30 of 3x10mm., 4 of 3x15mm., 4 of 3x25 mm., 4 of 3x35mm. Screw and 30 of 3mm. nuts), 2 of Flat head screws, **Set of Spacers** (4 of 5mm., 4 of 10mm, 4 of 15mm. and 4 of 25 mm.)
1.1.2 Electronic parts

**RBX-51AC2** The T89C51AC2 Robotic experiment board

**ZX-03Q**
Infrared Reflector x2

**ZX-08**
Infrared Objector x2

38kHz Infrared Receiver module

**ZX-01**
Switch input x2

**ZX-08**
Infrared Objector x2

**ER-4**
Infrared Remote Control

**AA size 4-pieces Battery holder**
with power cable

**16-Characters 2-lines LCD module**
with back light

1.2 Hand tools for making robot kit

**Cutter Plier**
A sharp-tipped hobby knife or Handy Cutter

**Philips Screwdriver**

**Computer**
Install Windows98SE or higher (Windows XP is recommended) and has a free RS-232 serial port at least. If not, the USB to Serial port converter is required. UCON-232S is recommended.
1.3 Software development tools for Robot programming

In Robo-51 kit use MCS-51 microcontroller, T89C51AC2 thus builders can write the controlled program in assembly, BASIC and C language. Only BASIC and C program must use compiler software.

However in this kit select the C program in all examples. The C compiler which suggest in here is Raisonance 8051 in name Rkit-51. This tools includes many software to support the C programming development such as RIDE - IDE for making the code in C and assembly, RC51 - C compiler, MA51 - Assembler, LX51 - Linker tools, UB51 - Library Manager and RIDE Simulator/Debugger.

The demo version of Rkit-51 is selected for this robot kit. Limit is 4KB code size.

Builders who need to develop the advance program, please purchase the full version from Raisonance at their website; www.raisonance.com. However in the Robo-51 robot kit contains this software in a bundled CD-ROM.

Builders can see the instruction manual of Rkit-51 and all related software tools after installation. In this manual does not describe all instructions.

![Image of RIDE IDE]

Figure 1-1 The main window of RIDE the IDE of Rkit-51 software tools for Robo-51 programming development
Figure 1-2. The main window of Flip the flash programming software for T89C51AC2 microcontroller from Atmel. Experimenters must use this software in flash programming with Robo-51 kit.

After compiling the C project file, use will get the result file in HEX format (.hex). Must download this code to program memory in T89C51AC2 microcontroller. The downloading software is FLIP from Atmel Corporation. Download free of charge at www.atmel.com. However in the Robo-51 robot kit contains this software in a bundled CD-ROM ready.
Chapter 2

Introduction the software tools for Robo-51 kit

2.1 Introduction to Rkit-51

The Robo-51 educational robotic kit programming concentrate to C language. The suggestion development tool is Raisonance 8051 from French’s developer, Raisonance S.A. The software suite is called “Rkit-51”. The features of this software tools are:

- The best price/performance for professional tools
- Excellent code density (even better with CodeCompressor(TM))
- Optimizing C Compiler
- Intuitive IDE (one click to start the simulator and load the program)
- Powerful IDE (Scripts, Multiproc. Simulation and Peripheral Dev. Kit)
- Accurate Simulators (bit level representation of the UART)
- Hardware and Software managed from the IDE (no compatibility problems)
- Prompt support (help and fixes)
- Support many numbers of MCS-51 microcontroller from leader manufacturer such as AT89xxx from Atmel, P89C51Rxx and P89LPC9xx from Philips or ADuC8xx from Analog Device etc.

- Raisonance provides the comprehensive evaluation version. User can test and evaluate the operation nearly the commercial version. 4KBytes code size limit is good enough for all robotic experiment in this manual and user can learn and practice the C programming very well.

User who need more information or order the full version, please contact Raisonance at www.raisonance.com
2.1.1 Rkit-51 Tools set

Rkit-51 integrates 5 software tools under RIDE (Raisonance Integrated Development Environment). RIDE is a Window program that allows the user to create projects, easily call the Compiler, Assembler and Linker to build the project and either simulate or debug the project. The following is a list of the tools included in the Rkit-51 development kit with a short overview of each one:

1. **RC-51** ANSI C Compiler: compile C code to Assembly code
2. **MA-51** Assembler: convert the assembly code to machine code.
3. **LX-51** Linker/locator: combines the object files generated by the Compiler and Linker and produces a different kind of object file. The Linker also decides where certain types of Data and Code are located in memory.
4. **LIB-51** Library Manager: take object files generated by the Compiler or Assembler and create a library that is included in other projects.
5. **OH51XA** Object-HEX Converter: converts an object file generated by the linker and generates an Intel Hex file, compatible with most device programmers.

The diagram in the figure 2-1 shows the relationship between the tools.

![Diagram of RIDE tools relationship](image)

Figure 2-1 shows the relationship between the tools in Rkit-51.
2.1.2 Installation

If the software is being installed from CD then the installation program should automatically run when the CD is inserted.

If the CD autorun feature is turned off or you have downloaded the software from a web site then the software may be installed simply by running INSTALL.EXE.

Minimum System Requirements for Rkit-51 is

- Windows 98/NT/2000/XP
- Pentium Processor
- 30Mb Hard Drive Space
- 64Mb RAM

2.1.3 Getting start with Rkit-51

Starting RIDE is very easy. Select “Ride IDE” from the Start → Programs → Raisonance Kit menu. You will be presented with the following splash screen:

After a few moments the main window will open. The main window is described in the next section.
2.1.3.1 Creating a Project

The first thing we will do is create a new project. We will then take a look at the main RIDE window and become familiar with it.

(1) Choose **Project → New** and you will be presented with a window which looks like the following:

![Project Window](image)

(2) The Name field shows the path to the project file that will be created. The Type field shows the microcontroller family type the project will use. Depending on which development kit you are using the Type field will show either: 80C51, XA or ST6. In here select **80C51**.

(3) Click on the **Browse** button and browse to the folder where the project is to be created. Example will use the location: **C:\ride\work**

(4) In the Filename field the name of the project is entered. Enter **test** and click on the **Next** button. The Target window will appear to select the target microcontroller.

![Target Window](image)
(5) At the **Device** tab, select **Atmel → T89C51AC2**. After that select **Properties** tab to select **Harvard Architecture**. Click the **Finish** button to create the project. The main RIDE window should now look the following:

![RIDE window](image)

The Project window acts like a project manager, showing which source files are in the project and giving instant access to each one in both the editor and debugger. If you look at the Project window you will see one entry with the following pathname: `C:\WORK\TEST.AOF`. This entry represents the project as a whole and the .aof file will be the result of building the project.

### 2.1.3.2 Creating and Adding a Source File

The next step we will take is to create a new, basic source file and add it to the project. The following section will show how to build the project. We will then know that all the tools are working and you will then have a starting point for all future projects.

(6) To create a new source file choose **File → New** followed by “**C Files**” from the pop-up menu that appears. A blank window will open.

(7) Enter the following into the new window:

```c
void main (void)
{
   while (1) ;
}
```
(8) To save the source file:

(8.1) Choose File → Save As. A standard Save As window will open.
(8.2) Enter main.c into the Filename field.
(8.3) Click on Save.

(9) To add the file to the project:

(9.1) To menu Project → Add note Source/Application or press Alt + Insert

(9.2) Select the file which required to add (main.c). Click the Open button.
(9.3) The source file should now appear in the Project window under the .aof file:

![Project window screenshot]

2.1.3.3 Building the Project

(10) To build the project simply click on the Make All button on the toolbar Or choose Project → Make All

![Make window screenshot]

(11) Once the project has been built, the Make window will show the result of the build process in tree form:

![Make window output]

Once you have the project successfully built we are ready to start !!!
2.2 Introduction to Flip

**Flip** is a flexible PC-application which lets you program and configure Atmel's microcontroller devices in-system. This new major version of Flip offers the following capabilities:

- Perform In-System Programming through RS232, USB or CAN interfaces.
- May be used through its intuitive Graphical User Interface or launched from a DOS window (see the batchisp manual), from an embedded software IDE like KEIL's uVision2, or even from your own application (see the ISP Functions Library manual).
- Runs under Windows 9x / Me / NT / 2000 / XP.
- Supports Intel MCS-86 Hexadecimal Object, Code 88 file format for data file loading and saving.
- Buffer editing capabilities: fill, search, copy, reset, modify, goto address.
- Target device memory control: erase, blank check, program, verify, read, security level and special bytes reading and setting.
- ISP hardware conditions may be set by software.
- The demo mode emulates ISP operations without any target hardware.

The suitable version of Flip for Robo-51 experiment is V1.8 or higher. Because this version will support T89C51AC2 microcontroller. The Flip function is downloading the HEX code from computer to T89c51AC2 microcontroller on the RBX-51AC2 controller board. This controller board is brain of Robo-51.

2.2.1 Installation

If the software is being installed from CD then the installation program should automatically run when the CD is inserted. Find the `setup.exe` file.

If the CD autorun feature is turned off or you have downloaded the software from a website then the software may be installed simply by running `setup.exe`.

After installation complete, run this program from **Programs → Atmel → Flip 2.2.4 → Flip** (the number of version could be change)
2.2.2 Getting start with Flip

After run the FLIP, the main window of FLIP will appear. See all detail in the figure 2-1

In the FLIP main window, you can see three frames from left to right:

- Operations flow
- Buffer information
- Device parameters

A message log window, a progress bar area and a communication information report are displayed in the bottom of the main window.

2.2.2.1 Important command menu

**File**

- **Load Hex file**: Select the HEX file to download.
- **Save Hex file**: Store the current HEX data in buffer to HEX file

**Buffer**

- **Edit**: Check and change data in buffer.
- **Option**: Set some option parameter related to buffer.

**Device**

- **Select**: Select the target microcontroller.
- **Erase**: Erase all data in flash memory of target microcontroller.
- **Blank check**: Check the empty memory.
- **Read**: Read or get data from microcontroller to buffer.
- **Program**: Program or download the HEX code in the buffer into flash program memory of the target microcontroller.
- **Verify**: Check the flash programming by comparing with buffer.

**Settings**

- **Communication**: Select the communication port between PC and the target microcontroller
- **Command Window**: Select to show the result window.

**Help**: See all instruction and how to use FLIP software.
1. Select the target device or microcontroller button.

2. Set communication or the computer port button which connect with the target microcontroller.

3. Erase the flash program memory within the target device.

4. Blank check button.

5. Program the target device button.

6. Verify device button.

7. Read device button; read data from the target microcontroller to show in the buffer.

8. Edit buffer button; determine the value to change in buffer.

9. Load HEX file; import the hex file from compiling or assembling and store in buffer.

Figure 2-1 shows detail of FLIP software (continue)
10. Save HEX file button.

11. Help button : see some information of FLIP software.

12. Instruction list for Automatic programming in the Operation flow. You need which function, check the box in front of that instruction as follows : Erase/Blank check/ Program/Verify. After setting, click the RUN button to operate. Click the Clear button if not require the auto programming.

13. Label for showing the HEX file is selected.

14. Status box

15. Progress bar

16. Select and setting buffer information

17. Summary information of the target microcontroller; include :
   
   17.1 Inform the number of target microcontroller.
   
   17.2 Signature byte
   
   17.3 Device Boot IDs
   
   17.4 Hardware byte
   
   17.5 Bootloader Ver.
   
   17.6 BLJB (Bootloader Jump Bit) : this parameter must selected for enable the In-System Programming
   
   17.7 X2 : this parameter is clock operation. If checked, the 6 clock per cycle will act. But if not selected, 12-clock operation will work. In Robo-51 kit must select this parameter.
   
   17.8 Level 0-2 Code protection level.

Figure 2-1 shows detail of FLIP software (final)
2.2.2.2 How to download file to microcontroller with FLIP

(1) Connect the RBX-51AC2 board to the PC RS232 port.

(2) Turn the power supply on. Press MODE switch to select Program mode. The LED PGM will lights and reset the target microcontroller. We assume that the device to be programmed contains a FLIP compliant bootloader program. Hardware ISP will be discussed further in this manual; this introduction guide covers software ISP only.

(3) Run the FLIP

(4) Select a device from the device list:

   (4.1) From the top menu bar, execute Device → Select. The device selection dialog box pops up. Select a device from the devices list box and click OK.

   (4.2) As soon as the device is selected, the Device parameters area is updated and lets you see the device special bytes. The Buffer information area is updated as well with device dependent information.
(5) Select a communication medium:

(5.1) From the top menu bar, execute **Settings → Communication** and select **RS-232**.

(5.2) The RS-232 setup dialog box pops up. Adjust the communication parameters, and click **Connect**. FLIP starts a synchronization sequence with the target device bootloader software. After the synchronization sequence completion, FLIP reads the target device special bytes and updates the main window frame on the right.

The right box will active after the connection is succeed.
(6) Select a HEX data file:

(6.1) From the top menu bar, execute **File → Load HEX File...** or click at button.

(6.2) Select a HEX file from the file browser.

(7) At the Operation Flow box in the left, click the instructions in order to from top to bottom as follows **Erase → Blank Check → Program → Verify.** After that click the **RUN** button.

**FLIP** will erase all data in flash program memory, blank check, write the HEX code into flash and EEPROM memory and verify. All operation works automatically. User can bring the chip to working.
2.2.2.3 Addition instruction

Addition the flash programming, FLIP have more instructions similar the professional flash programming software as follows:

(8) Open the buffer edition window:

(8.1) From the top menu bar, execute Buffer → Edit.

(8.2) The Edit Buffer window pops up. You may now perform many operations onto the buffer contents.

(9) Open the buffer options window:

(9.1) You may open the buffer options window from the FLIP main window, or from the Edit Buffer dialog box.

(9.2) From the main window menu bar, execute Buffer → Options. The Buffer Options dialog box pops up. The main buffer options are:

- buffer size
- initial contents
- address programming range

(10) Verify the target device:

From the top menu bar, execute Device → Verify.

For detailed descriptions of the possible operations, please see the Operations Summary in Help file within FLIP software.
Chapter 3

The RBX-51AC2 controller board

3.1 Introduce the RBX-51AC2 robot controller board

Robo-51 robotic kit is controlled by the RBX-51AC2 (T89C51AC2 Robot Experiment board). The main microcontroller is the Atmel’s T89C51AC2. Figure 3-1 shows the board layout for explaining the important component of this controller board. Figure 3-2 shows operating diagram of RBX-51AC2 board. In this chapter will present the RBX-51AC2 board and some example experiment. Summary of technical feature as follows:

- Controlled by T89C51AC2 Microcontroller with 32Kbyte memory
- Download the program via Serial port, selected by Mode switch
- LCD16x2 display
- Piezo speaker
- 3-LED monitor
- Drive 2-DC motors 4.5V to 6V and 3-RC Servo motors (in range 4.8 to 6V)
- 5-Programmable digital input/output ports
- Supply voltage from 4 of AA batteries (Rechargeable 1700mAh recommended)
- 2.375 x 6.25 Inches size

Figure 3-1 Layout of RBX-51AC2 Robot controller board for Robo-51 robotic kit
Figure 3-2 The completely schematic diagram of RBX-51AC2 controller board
3.2 Preparing

3.2.1 Preparing the software development tools

Before experiment with RBX-51AC2 board, Users must prepare the software as follows:

(1) Run the Window Explorer. Enter to RIDE folder at C:\RIDE. Enter to folder INC \ Atmel. Copy the c51ac2.h and c51ac2.inc file and paste to C:\RIDE\INC.

(2) If users have any include file that use INCLUDE directive, must copy all include files into C:\RIDE\INC.

3.2.2 Preparing the RBX-51AC2 controller board

Before experiment with RBX-51AC2 board, Users must prepare the hardware as follows:

(1) Put 4 pieces of AA Battery in the battery holder.

(2) Connect the battery holder's power cable to power connector on the RBX-51AC2 controller board.

(3) Connect the serial port cable between RBX-51AC2 controller board and COM or RS-232 port of computer. If computer has only USB port, the USB to Serial port converter is required. INEX’s UCON-232S converter is recommended.

3.3 Step of hardware experiments with the RBX-51AC2 controller board

The step of hardware experiment can summarize as follows:

(1) Open RIDE software. If have any project still open, must close first by select menu Project → Close

(2) Create the new project. To menu Project → New Project. → New. The Project setting window appears. Put the project name into Name box and select type of microcontroller as 80C51. Click the OK button.

(3) At the Device tab, select Atmel → T89C51AC2. After that select Properties tab to select Harvard Architecture. Click the Finish button to create the project.

(4) Create the C file by enter to File → New → C file. A blank window will open.

(5) Type the C source code into the new window. Save in .c file with same project's name in step 2.
(6) Add file into the project with select to menu **Project → Add note Source/**

**Application**

(7) Select the file which required to add (.c). Click the **Open** button.

(8) Select to menu **Option Project**. The Option window appear and select

Directories. Go to Include (.h, .inc) box for setting parameter as

```
%RIDEDIR%\Inc;%RIDEDIR%\Inc or C:\RIDE\INC
```

(9) Make project with press F9 key or select to menu **Project à Make All**. At

Make output windows will shows the making project result. If all complete, many files will be create include .aof, .dbi, .bak, .lst, .xrf and .hex file. The hex file is resulting file for programming into the microcontroller with FLIP software.
3.4 Interface LED programming

Firstly, experiment with RBX-51AC2 controller board is driving LED. The RBX-51AC2 board has 3 LED that is connected with P3.5 to P3.47 of T89C51AC3 microcontroller. The circuit only this interface is shown in figure 3-3.

3.4.1 How to control LED on RBX-51AC2 board

Refer circuit in figure 3-3, LED will be connected in sink current type. The 1kW resistor is limit current resistor for LED. After power-on reset, all port initial to logic “HIGH”. All LED will off. Thus, driving LED to on must send logic “LOW” to each port.

In programming, must make the program to sending logic “LOW” to port and delay for maintain the logic “LOW” for human seeing. If not delay, human cannot see the LED operation because microcontroller operation works very fast. Delay is very important. However in C programming tools do not provide Delay function. Users must make with their own and save as library file.

3.4.2 Delay_robo51 library

Listing 3-1 is source code of delay_robo51.h. The delay library file which use in all experiments in Robo-51 kit. User can make with type the program in notepad or any editor and save as in .h file. Store into c:\ride\inc.

![Figure 3-3 Driving LED circuit in RBX-51AC2 controller board](image-url)
3.4.2.1 Function description

(1) **delay_ms**: Delay in millisecond unit

**Function format:**

```c
void delay_ms(unsigned int ms)
```

**Parameter:**

ms - Delay time in millisecond. Range is 0 to 65,535

(2) **delay_100us**: Delay in 100 microsecond or 0.1 milliseconds unit

**Function format:**

```c
void delay_100us(unsigned int us)
```

**Parameter:**

us - Delay time in 100 microsecond unit. Range is 0 to 65,535

**Example 3-1**

```c
delay_100us(5);             // Delay 500 microsecond
delay_100us(20);           // Delay 2000 microseconds or 2 millisecond
delay_ms(100);             // Delay 100 microsecond
delay_ms(2000);            // Delay 2 minutes
```
Activity 1
Driving the LED

A1.1 Open RIDE and create the new project. Make C code in Listing A1-1 and save as act01.c file. Build project to HEX file.

A1.2 Apply the supply voltage to RBX-51AC2 controller board.

A1.3 Download the hex file from step A1.1 with FLIP software and run. Observe the operation.

3-points Running light LED will operate. It start from P3.5 to P3.7. LED will on with 2 seconds.

Listing A1-1 The C program for driving LED on te RBX-51AC2 controller board
3.5 How to generate sound signal

The microcontroller creates these beeps by sending rapid high/low signals to various types of speakers. The rate of these high/low signals is called the frequency, and it determines the tone or pitch of the beep. Each time a high/low repeats itself, it is called a cycle. You will often see the number of cycles per second referred to as Hertz, and it is abbreviated Hz. For example, one of the most common frequencies for the beeps that help machines get your attention is 2 kHz. That means that the high/low signals repeat at 2000 times per second.

3.5.1 Introducing the Piezoelectric Speaker

In this activity, you will experiment with sending a signal to a common, small, and inexpensive speaker called a piezoelectric speaker.

How the Piezoelectric Speaker Circuit Works?

When a guitar string vibrates, it causes changes in air pressure. These changes in air pressure are what your ear detects as a tone. The faster the changes in air pressure, the higher the pitch, and the slower the changes in air pressure, the lower the pitch. The element inside the piezo speaker’s plastic case is called a piezoelectric element. When high/low signals are applied to the speaker’s positive terminal, the piezoelectric element vibrates, and it causes changes in air pressure just as a guitar string does. As with the guitar string, your ear detects the changes in air pressure caused by the piezoelectric speaker, and it typically sounds like a beep or a tone.

3.5.2 Driving a sound signal on RBX-51AC2 controller board

On the RBX-51AC2 controller board has a Piezo speaker. It is generate the signal through P2.7 of T89C51AC2. The signal inputs to the simple transistor amplifier circuit and drive signal to the piezo speaker. The interface circuit is shown in figure 3-4.

In programming, microcontroller make the “HIGH” / “LOW” signal at P2.7 port. The simple square wave signal is shown in figure 3-5. If you can make this signal continue, you can determine period of sound generating.

The signal has 100 microsecond cycle. You can calculate the frequency as follows

\[
f = \frac{1}{T} = \frac{1}{2 \times 100 \times 10^{-6} \times \text{dt}} = \frac{5000}{\text{dt}}
\]

and

\[
\text{dt} = \frac{5000}{f} \quad \text{...................................................(3.1)}
\]
You can determine the duration of the sound signal in programming with set the number of period. In figure 3-6 the signal is generated continue with time parameter in millisecond unit. You can find the number of period from

\[
\text{Number of period or cycle} = \frac{\text{time} \times 10^{-3}}{2 \times 100 \times 10^{-6} \times \frac{dt}{\text{time}}} = \frac{5 \times \text{time}}{dt} \quad ........(3.2)
\]
3.5.3 Sound_robo51 library

Listing 3-2 is source code of delay_robo51.h. The delay library file which use in all experiments in Robo-51 kit. User can make with type the program in notepad or any editor and save as in .h file. Store into c:\ride\inc.

3.5.3.1 Function description

(1) delay_sound : Set the delay time for the sound signal generation

**Function format:**

```c
void delay_sound(unsigned int ms)
```

**Parameter:**

- `ms` - Delay time in millisecond. Range is 0 to 65,535

(2) sound : Generate the sound signal

**Function format:**

```c
void sound(int freq,int time)
```

**Parameter:**

- `freq` - Target frequency in Hz
- `time` - Duration time in second unit

(3) beep : Generate a beep signal at 500Hz frequency 0.1 second duration.

**Function format:**

```c
void beep(void)
```

**Example 3-2**

```c
sound(700,200); // Generate the 700Hz signal 200 milisecond duration
beep(); // Generate a beep signal
```
/*—————————————————————————————————————*/
// Program : Generate sound by frequency
// Description: Generate sound function
// Filename : sound_robo51.h
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/

sbit s_bit = P2^7;
#pragma DISABLE // for disable all interrupt
void delay_sound(unsigned int ms)
{
    unsigned int x,a; // Keep for counter loop
    for(x=0;x<ms;x++)
    {
        for(a=0;a<75;a++); // Loop for delay 100 us per unit
    }
}
#pragma DISABLE
void sound(int freq,int time)
{
    int dt=0,m=0; // Keep value and keep active logic delay time
    dt = 5000/freq;
    time = (5*time)/dt; // Keep counter for generating sound
    for(m=0;m<time;m++)
    {
        s_bit = 1; // P2.7 = high
        delay_sound(dt); // Delay for sound
        s_bit = 0; // P2.7 = low
        delay_sound(dt); // Delay for sound
    }
}
#pragma DISABLE // Disable all interrupt before call ‘beep’ function
void beep(void)
{
    sound(500,100); // Generate sound at default frequency
}

Note : 
In using all function of this library do not disturb from any interrupt. Because put
#pragma DISABLE directive before call each function.

Listing 3-2 Sourcecode of sound_robo51.h library file. This library function
is generate and control the sound signal to the piezo speaker on the RBX-51AC2 controller board
Activity 2

Sound driving

A2.1 Open RIDE and create the new project. Make C code in Listing A2-1 and save as act02.c file. Build project to HEX file.

A2.2 Download the hex file from step A2.1 with FLIP software and run. Observe the operation.

LED at P3.5 to P3.7 light and listen the sound in different frequency.

P3.5 is 1kHz, P3.6 is 800Hz and P3.7 is 500Hz.

Listing A2-1 The C program for driving LED and sound on the RBX-51AC2 controller board

```c
/*—————————————————————————————————————*/
// Program : LED and sound
// Description: drive LED P3.5 , P3.6 and P3.7 by delay interval 2 sec and
// generate sound when
// : change LED
// Filename: act0401_sound.c
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
#include <C51ac2.h> // Declare T89C51AC2’s register
#include <delay_robo51.h> // Declare Delay library
#include <sound_robo51.h> // Declare Sound generator library
sbit led1 = P3^5; // Define led1 to P3.5
sbit led2 = P3^6; // Define led2 to P3.6
sbit led3 = P3^7; // Define led3 to P3.7

void main()
{
    while(1) // Endless loop
    {
        sound(1000,400); // Generate sound at 1000Hz for 0.4 second
        led1 = 0; // Turn-on LED at P3.5
        delay_ms(2000); // Delay 2 seconds (call function from Delay library)
        led1 = 1; // Turn-off LED at P3.5

        sound(800,200); // Generate sound at 800Hz for 0.2 second
        led2 = 0; // Turn-on LED at P3.6
        delay_100us(20000); // Delay 2 seconds (call function from Delay library)
        led2 = 1; // Turn-off LED at P3.6

        beep(); // Generate "beep" signal at 500Hz for 0.1 second
        led3 = 0; // Turn-on LED at P3.7
        delay_100us(20000); // Delay 2 seconds
        led3 = 1; // Turn-off LED at P3.7
    }
}
```
3.6 Displaying with LCD

The RBX-51AC2 board provides the connector to interface the LCD module. It interfaces with LCD module in 4-bit mode. The important technique in this mode is sending 3H data to LCD module 2 times continue and following 2H data at once. It is initialize the LCD module to 4-bit interfacing mode.

3.6.1 LCD interface in the RBX-51AC2 controller board

The interface circuit is shown in figure 3-7. Interfacing the 6 port pins include P0.4 to P0.7 for data pin D4-D7 in mode 4-bit interface, P0.2 to RS pin and P0.3 to E pulse pin. R/W pin of LCD is connected to ground for only writing all data to LCD only.

![Diagram of LCD interface circuit](image-url)

Figure 3-7  LCD module interface circuit of the RBX-51AC2 controller board
### LCD module instruction set

1. **Clear screen**
   - Instruction data is $1.

2. **Return Home**
   - Instruction data is $2 or $3. Recommended is $2. This instruction move the cursor to home position and display not change.

3. **Entry mode set**
   - Instruction data format is
     
     - **I/D bit**: Set the DDRAM address after write or read data.
       - "0" - Decrease address
       - "1" - Increase address
     
     - **S bit**: Set entry mode
       - "0" - Automatic cursor shift right after byte
       - "1" - Cursor not move. The new character move left instead
     
   The populated instruction data is $6. It means Cursor shift right when the new character is entered and DDRAM address increase.

4. **Control display**
   - Instruction data format is
     
     - **D bit**: ON/OFF screen
       - "0" - Off LCD screen
       - "1" - On LCD screen
     
     - **C bit**: ON/OFF Cursor
       - "0" - Off the cursor
       - "1" - On the cursor
     
     - **B bit**: Cursor blinking control
       - "0" - The cursor does not blink
       - "1" - The cursor blink
     
   The populated instruction data is $0C. It means On the LCD screen, Off the cursor. Another is $0F. It means On the LCD screen, On the cursor and blink the cursor.

5. **Shifting control**
   - Instruction data format is

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>I/D</td>
<td>S</td>
</tr>
</tbody>
</table>

   - **S/C**
   - **R/L**: shifting format
   - **Inst. data**
     
     - "0" - 4-bit mode
     - "1" - 8-bit mode

   - **S/C**
   - **R/L**
   - **shifting format**
   - **Inst. data**: $10-$13
   - $14-$17
   - $18-$1B
   - $1C-$1F

6. **Function control**
   - Instruction data format is

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DL</td>
<td>N</td>
<td>F</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

   - **DL bit**: Set the interface mode
     - "0" - 4-bit mode
     - "1" - 8-bit mode

   - **N bit**: Select line display
     - "0" - 1-line
     - "1" - 2-line or more

   - **F bit**: Select character resolution
     - "0" - display in 5x7 points
     - "1" - display in 5x10 points
3.6.2 **lcd_robo51 library**

Listing 3-3 is source code of *lcd_robo51.h*. The LCD library file which use in all experiments in Robo-51 kit. User can make with type the program in notepad or any editor and save as in .h file. Store into c:\ride\inc.

```c
/* Program : Module function control LCD display
// Description: Call function control LCD display group type 4 bit
// Filename : lcd_robo51.h
// C compiler : RIDE 51 V6.1
*/
#define line1 0x80 // Define constant
#define line2 0xC0 // Define constant
sbit rs = P0^2; // Bit control command for LCD
sbit e = P0^3; // Bit control pulse command for LCD
enum {Hex, Dec}; // Format display integer

/***************** Function delay 1 ms per unit ******************************/
void lcd_delay(unsigned int ms)
{
    unsigned int x, a; // Keep for counter loop
    for(x=0; x<ms; x++)
    {
        for(a=0; a<878; a++); // Loop for delay 1 mS per unit
    }
}

/***************** Function send command to LCD format 4 Bit *******************/
void lcd_command(unsigned char com)
{
    unsigned char buff; // For Keep command send to LCD
    buff = com & 0xF0; // Keep 4 bit high byte before send to LCD
    rs = 0; // Select send command to LCD
    e = 1; // Start generate pulse clock LCD
    P0 = (P0 & 0x0F)|buff; // Send data to LCD port
    lcd_delay(1); // Delay 1 ms
    e = 0; // Stop generate pulse clock LCD
    lcd_delay(1); // Delay 1 ms
    buff = (com & 0x0F)<<4; // Keep 4 bit low byte before send to LCD
    rs = 0; // Select send command to LCD
    e = 1; // start generate pulse clock LCD
    P0 = (P0 & 0x0F)|buff; // Send data to LCD port
    lcd_delay(1); // Delay 1 ms
    e = 0; // Stop generate pulse clock LCD
    lcd_delay(1); // Delay 1 ms
}

/*********************** Function send data to LCD format 4 Bit *****************/
void lcd_text(char text)
{
    unsigned char buff; // For Keep data send to LCD
    printf("%c", text); // Start send data to LCD
    printf("\n"); // Start send command to LCD
    lcd_command(0x01); // Start new line
    lcd_command(0x08); // Start move cursor
    lcd_command(0x04); // Start show
    lcd_text("Welcome to Robo-51"); // Start send data to LCD
    lcd_command(0x02); // Start move cursor
    lcd_text("Microcontroller Programming Library"); // Start send data to LCD
    lcd_command(0x02); // Start move cursor
    lcd_text("C:\ride\inc\lcd_robo51.h"); // Start send data to LCD
    lcd_command(0x02); // Start move cursor
    lcd_text("Listing 3-3 Sourcecode of *lcd_robo51.h* library file. This library function is control the LCD module 16x2 on the RBX-51AC2 controller board (continue)"
Listing 3-3 Sourcecode of `lcd_robo51.h` library file. This library function is control the LCD module 16x2 on the RBX-51AC2 controller board (continue)
buff[4] = (((value%10000)%1000)%100)%10 | 0x30;
    // Convert data to ascii 1’th
buff[5] = 0; // End of data
break; // Out block for decimal format case

    case Hex : buff[0] = (value & 0xF000)>>12; // Filter 4 bit 1’th
        if(buff[0]<10){buff[0] |= 0x30;} else {buff[0] = (buff[0]-9) | 0x40;}
        // Convert data to ascii
buff[1] = (value & 0x0F00)>>8; // Filter 4 bit 2’th
        if(buff[1]<10){buff[1] |= 0x30;} else {buff[1] = (buff[1]-9) | 0x40;}
        // Convert data to ascii
buff[2] = (value & 0x00F0)>>4; // Filter 4 bit 3’th
        // Convert data to ascii
buff[3] = (value & 0x000F); // Filter 4 bit 4’th
        // Convert data to ascii
buff[4] = 0; // End of data
break; // Out block for hexadecimal format case

    default :break; // Out block for default case
}
if(buff[0]==0x30) // if data 1’th = 0 none display
    {index = 1;} // Shift to display 2’th
if(buff[0]==0x30 && buff[1]==0x30) // if data 1’th = 0 and 2’th = 0 none display
    {index = 2;} // Shift to display 3’th
if(buff[0]==0x30 && buff[1]==0x30 && buff[2]==0x30) // if data 1’th = 0, 2’th = 0 and 3’th = 0
    {index = 3;} // no display
if(buff[0]==0x30 && buff[1]==0x30 && buff[2]==0x30 && buff[3]==0x30) // if data 1’th = 0, 2’th = 0, 3’th = 0 and
    {index = 4;} // 4’th = 0 none display
lcd_putstr(line_sel,&buff[index]); // Send integer to LCD

Listing 3-3 Sourcecode of **lcd_robo51.h** library file. This library function is control the LCD module 16x2 on the RBX-51AC2 controller board (final)
3.6.2.1 **Function description**

(1) **lcd_init**: Initial LCD module to 4-bit mode, 2-lines display and 5*7 points resolution

**Function format**:  
```c
void lcd_init(void)
```

**Example 3-3**
```c
lcd_init();  // Initial LCD module
```

(2) **lcd_command**: Send the display control instruction to LCD module

**Function format**:  
```c
lcd_command(unsigned char com)
```

**Parameter**:  
- `com` - Instruction data 1 byte

**Example 3-4**
```c
lcd_command(0x01);  // Clear LCD screen
lcd_command(0x02);  // Return home
lcd_command(0x80);  // Jump to DDRAM address 00H or $00
lcd_command(0xC6);  // Jump to DDRAM address 46H or $46
```

(3) **lcd_text**: Send character to display

**Function format**:  
```c
void lcd_text(char text)
```

**Parameter**:  
- `text` - ASCII code of text 1 byte

**Example 3-5**
```c
lcd_text('A');  // Display "A" at LCD module
lcd_text(0x30);  // Display number "0" at LCD module
```
4. **lcd_putstr**: Send message to display

**Function format:**

```c
void lcd_putstr(unsigned char line, char *p)
```

**Parameter:**

- `line`: Start address for message. It is 1-byte data
- `p`: Array pointer of function. Get the message that is pointed to display with `lcd_text` function

**Example 3-6**

```c
lcd_putstr(0x81, "hello world"); // Shows message "hello world" at LCD module
```

5. **inttolcd**: Send the integer to display

**Function format:**

```c
void inttolcd(unsigned char line_sel, int value, int format)
```

**Parameter:**

- `line_sel`: Start address for message. It is 1-byte data
- `value`: Integer value to display
- `format`: Numerical format, Dec or Hex

**Example 3-7**

```c
inttolcd(0x82, 244, Dec); // Shows 244 number in decimal, start at DDRAM address $02
inttolcd(0x82, 244, Hex); // Shows F4 number in hex, start at DDRAM address $02
```

**Example 3-8**

```c
int number; // Declare number variable
inttolcd(0xC5, number, Dec); // Shows value of number in decimal, // start at DDRAM address $45
```
Activity 3

LCD experiment

A3.1 Open RIDE and create the new project. Make C code in Listing A3-1 and save as act03.c file. Build project to HEX file.

A3.2 Download the hex file with FLIP software and run. Observe the operation.

Start..LCD module shows the title message below in 4 seconds:

Hello
RBX-51AC2

After that, LED at P3.5 to P3.7 will act the running lights and LCD screen change to shows the message of LED’s position which turn-on. Piezo speaker drive a beep when any LED turn-on.

Listing A3-1 The C program for driving LED, sound and display message at LCD module on the RBX-51AC2 controller board (continue)
Listing A3-1 The C program for driving LED, sound and display message at LCD module on the RBX-51AC2 controller board (final)
Robotics Experiment with MCS-51 microcontroller
Chapter 4
Building Robo-51 kit

4.1 Robo-51 features

This chapter describes about how to building the Robo-51 robot kit. The Robot kit in this chapter is consist of the basic sensor, ZX-03 Infrared reflector. The features of Robo-51 robot kit are as follows:

- Driving with DC motor gearboxes and Track wheel
- Controlled by PIC16F877 microcontroller
- 8KWords program memory
- Re-programmable at least 10,000 times for flash program memory
- Support many types of sensor and detectors such as
  - ZX-01 Switch input board for attacking detection,
  - ZX-03 Infrared Reflector for line tracking and area,
  - ZX-05 Infrared receiver module for remote controlling,
  - ZX-08 Infrared Objector for contactless object avoiding,
  - GP2D120 Infrared distance sensor,
  - SRF05 Ultrasonic sensor,
  - CMPS03 Digital compass,
  - Memsic 2125 Accelerometer sensor
  and more...
- Provides Character LCD moduel 16x2 and LED status for displaying thre robot operation.
Activity 4
Make the Robo-51

Main sprocket wheel x 2
Large support wheel x 2
Hub x 8
3x6 mm. Screw x 3
Track wheel set (4 of 8-joint, 4 of 10-joint and 2 of 30-joint)
3x10 mm. Screw x 16
Plastic spacer (4 pieces of 3mm., 10mm., 15 mm. and 25 mm.)
Metal axel x 4
2 mm. Wood screw x 2
3x25 mm. Screw x 3
3 mm. Nut x 20
DC motor gearbox with mounting x 2

RBX-51AC2 T89C51AC2 Robot Experiment board x 1

Medium support wheel x 6
3x10 mm. Screw x 16

3x25 mm. Screw x 3

3 mm. Nut x 20

Plastic joiner (Right angle, Strength, Obtuse) x 60

Metal axel x 4

2 mm. Wood screw x 2

3x25 mm. Screw x 3

3 mm. Nut x 20

DC motor gearbox with mounting x 2

Main sprocket wheel x 2
Large support wheel x 2
Hub x 8
3x6 mm. Screw x 3
Track wheel set (4 of 8-joint, 4 of 10-joint and 2 of 30-joint)
3x10 mm. Screw x 16
Plastic spacer (4 pieces of 3mm., 10mm., 15 mm. and 25 mm.)
Metal axel x 4
2 mm. Wood screw x 2
3x25 mm. Screw x 3
3 mm. Nut x 20
DC motor gearbox with mounting x 2

RBX-51AC2 T89C51AC2 Robot Experiment board x 1

Medium support wheel x 6
3x10 mm. Screw x 16

3x25 mm. Screw x 3

3 mm. Nut x 20

Plastic joiner (Right angle, Strength, Obtuse) x 60

Metal axel x 4

2 mm. Wood screw x 2

3x25 mm. Screw x 3

3 mm. Nut x 20

DC motor gearbox with mounting x 2

Main sprocket wheel x 2
Large support wheel x 2
Hub x 8
3x6 mm. Screw x 3
Track wheel set (4 of 8-joint, 4 of 10-joint and 2 of 30-joint)
3x10 mm. Screw x 16
Plastic spacer (4 pieces of 3mm., 10mm., 15 mm. and 25 mm.)
Metal axel x 4
2 mm. Wood screw x 2
3x25 mm. Screw x 3
3 mm. Nut x 20
DC motor gearbox with mounting x 2

RBX-51AC2 T89C51AC2 Robot Experiment board x 1

Medium support wheel x 6
3x10 mm. Screw x 16

3x25 mm. Screw x 3

3 mm. Nut x 20

Plastic joiner (Right angle, Strength, Obtuse) x 60

Metal axel x 4

2 mm. Wood screw x 2

3x25 mm. Screw x 3

3 mm. Nut x 20

DC motor gearbox with mounting x 2

4-AA size Battery holder with power cable
(Battery not included. 1700mAH or higher Rechargeable battery recommended)

Short angled shaft base x 2

Universal Plate x 1

Long angled shaft base x 2

16-Characters2-Lines LCD module with connector x 1

ZX-03 Infrared reflector with cable x 2

Figure A4-1 Shows all parts of Robo-51 robot kit
A4.1 Create two track belts by putting the different size tracks together. One track would consist of the following: One 30-joint track, one 10-joint track, and two 8-joint track. Connect the 10-joint track to the end of the 30-joint track. Next connect the two 8-joint tracks together. Take one end and connect it to the other end of the 10-joint track. Then take the other end of the 8-joint track and connect it to the remaining end of the 30-joint track to form one complete loop. Repeat the steps to make two track sets. If the track is too tight or loose, you can either adjust the length of the track or adjust the position of the short angled shaft base until the track has a good fit.

A4.2 Attach the motor gearbox sets to the universal plate using a 3 x 6 mm. and 3 x 10 mm. screw. Position the Motor Gearbox sets as shown in the picture below. Attach only 3 points because 1 point remainder will be use for attach a Long angled shaft base.
A4.3 Insert the Main Sprocket Wheel into the motor gearbox and screw it in tightly with a 2 mm. self-tapping screw.

A4.4 Attach the Long angled shaft base with the plate by 3x10 mm. Screw. One position tighten a screw into the remainder hole from step A2.2. Another end, insert 3x10 mm. Screw from above through the plate’s hole and tighten by 3 mm. Nut. Attach another Long angled shaft base in same method.

A4.5 Attach the short angled shaft base at the end of the plate (opposite the motor attached end) as shown in the picture by counting the hole from the end 4 holes. Use 3x10 mm. Screw and 3 mm. Nut to tighten the short angled shaft base. Make them both side. Next, insert the metal axel through the hole of the short angled shaft base. Insert two Large support wheels to both end of axel and close by Hubs.
A4.6 Insert the metal axel into the holes of the long angled shaft in the hole positions of 1, 4, and 7 (Counting from any side). Place the Medium track support wheels over the metal axel as shown in the pictures. Insert the hubs over the wheels so that the wheels and the axels are connected tightly.

A4.7 Insert 3x15 mm. Screw through the ZX-03 Infrared reflector and 2 of 3 mm. Plastic spacer. Make 2 sets. Next, attach both sensors with the front of bottom base (opposite the attached motor end). Distance between both sensor is about 3 to 4 cm. Tighten with 3 mm. Nuts. The sensor will far from the floor about 5 mm.

A4.8 Open the cover of Battery holder. Insert 2 of 3x25 mm. Screws through the Battery holder and insert 15 mm. Plastic spacer via both screws. Attach the Battery holder with the Robot base. Turn the cable end in same direction with motor. Because the motor output connector and battery is near. Tighten by 3 mm. Nuts.
A4.9 Attach 2 of Right angle joiners at front position of robot base with 3x10 mm. Screws and 3 mm. Nuts. Tighten all together. At the back end, attach a Right angle joiner at the corner of Robot base. All 3-position of Right angle joiner will relate with the attachment holes of RBX-51AC2 board. Connect the Strength joiner at the end of 3 Right angle joiners for support the RBX-51AC2 board attachment.

A4.10 Attach 3 of Right angle joiners into the RBX-51AC2 board at the straight position from the Right angle joiner attachment from step A2.9 by 3x10 mm. Screws and 3 mm. Nuts.

A4.11 Load 4 of AA size batteries into the Battery holder and close the cover. Connect the RBX-51AC2 board that attached Right angle joiners with the Robot base from step A4.9. With this technique, builders can remove the board for changing the batteries easier.

A4.12 Plug the motor cables to M-1 and M-2 connector on the RBX-51AC2 board. The connection can change for adjust the motor direction to correct in the movement activity. Plug power cable from the Battery holder to Battery connector on RBX-51AC2 board. Keep all cables be neat and tidy.

A4.13 Attach the tracks to the supporting wheels of the robot.

A4.14 Plug ZX-03 Infrared Reflector’s cable to AN5 (for left sensor) and AN4 (for right sensor) of RBX-51AC2 board.

**Robo-51 ready to program and Run it !!!**
4.2 DC motor driving programming

The Robo-51 has 2 channels of DC motor drivers. You can control the speed and direction of DC motor rotation with software. Because DC motor is driven by PWM (Pulse width modulation) signal. In this section describe how to drive DC motor with PWM, circuit diagram, how to generate PWM signal of T89C51AC2 microcontroller and related library file in C programming.

4.2.1 Basic operation of driving DC motor with PWM

By changing (modulating) the width of the pulse applied to the DC motor we can increase or decrease the amount of power provided to the motor, thereby increasing or decreasing the motor speed. Notice that, although the voltage has a fixed amplitude, it has a variable duty cycle. That means the wider the pulse, the higher the speed.

Refer figure 4-1, the Vs supplies PWM signal to DC motor. The speed is depended on Ton time (ON time of motor). At this time, DC motor will receive the full voltage; Vm. If Ton’s width is more, DC motor is received more voltage. It rotate in high speed. The ratio of Ton time in percentage with period (T) is called Duty cycle. You can calculate as follows:

\[
\text{\% duty cycle} = 100 \times \frac{\text{Ton}}{\text{Ton} + \text{Toff}} \quad \text{.................................................................(3.3)}
\]

\[
\text{PWM frequency} = \frac{1}{\text{Ton} + \text{Toff}} = \frac{1}{T} \quad \text{.................................................................(3.4)}
\]

Average DC motor voltage drop = Supply voltage x duty cycle (%) ........(3.5)

![Figure 4-1 The PWM signal for driving DC motor](image)
Although the duty cycle is determine the motor speed. But DC motor can operate at limit frequency. If the PWM frequency is over the limit, DC motor will stop because its operation reach to saturation point. The figure 3-9 shows the operation of DC motor with the PWM duty cycle. The example PWM signal in figure 3-9 has 20 milliseconds period and 50Hz frequency.

In figure 4-2 (A) the PWM duty cycle is 20%. Motor will rotate with lowest speed because the voltage drop is only 0.9V. When increase the duty cycle in figure 4-2 (B) and (C), voltage is applied to DC motor increase. Its speed is increase too.

In figure 4-2 (D) the voltage is applied to DC motor full level because duty cycle is 100%. Thus, controlling the PWM duty cycle is a method of motor speed control.

Figure 4-2 shows the relation between the different duty cycle and voltage across the DC motor.
4.2.2 DC motor driver circuit of Robo-51’s controller board

The circuit is shown in figure 4-3. T89C51AC2 microcontroller generates PWM signal at P1.6 and P1.7 port. The PWM resolution is 8-bit, generated from PCA (Programmable Counter Array) module of T89C52AC2 microcontroller.

The PWM signal output to L293D, the H-bridge motor driver IC. It can drive 2 DC motors. T89c51AC2’s port P2.0 and P2.1 are connected to 1A and 2A input of L293D for direction control of motor output A. P2.2 and P2.3 are used control direction of motor output B. The indicator are bi-color LEDs.

**LED indicator shows Green color; it means motor rotate forward.**

**LED indicator shows Red color; it means motor rotate backward.**

If the direction is different from above, user must convert the connection to opposite and maintain the correct connection throughout all activities from here.

---

**Figure 4-3** DC motor driver circuit of the RBX-51AC2 board in Robo-51
4.2.3 T89C51AC2’s PCA module register in PWM operation

4.2.3.1 CCAPM PCA Compare/Capture Module Mode registers

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECOMn</td>
<td>CAPPn</td>
<td>CAPNn</td>
<td>MATn</td>
<td>TOGn</td>
<td>PWMn</td>
<td>ECCFn</td>
</tr>
</tbody>
</table>

T89C51Ac2 has 5 registers for supporting 5 of PCA modules. Address are 0DAH to 0DEH. 0DAH is address of CCAPM0 register and 0DEH is for CCAPM4. They are used for select the operation mode of PCA module which make a summary in Table 4-1. This register could not access in bit. User must access and set the values with byte.

In PWM operation, PCA can out PWM 5 channels as:

- P1.3 for PWM0 channel
- P1.4 for PWM1 channel
- P1.5 for PWM2 channel
- P1.6 for PWM3 channel
- P1.7 for PWM4 channel

However on the RBX-51AC2 controller board select only 2 channels to use; PWM3 and PWM4. Port pin are P1.6 for motor output A and P1.7 for motor output B. The selected register are CCAPM3 and CCAPM4. They are set to PWM mode by setting PWM and ECOM bit in both register. The C sample code is:

\[
\text{CCAPM3} = 0x42; \quad \text{// Set PCA module 3 to PWM mode}
\]

\[
\text{CCAPM4} = 0x42; \quad \text{// Set PCA module 4 to PWM mode}
\]

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No operation</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Capture mode Positive</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Capture mode Negative</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Capture mode both edge</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Compare mode</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>High speed output mode</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8-bit PWM mode</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>Watchdog timer</td>
</tr>
</tbody>
</table>

Table 4-1 The summary of selection PCA mode by setting CCAPM register. (n means number of PCA module, range is 0 to 4)
4.2.3.2 CCAPn : PCA Compare/Capture Module n Register (n=0..4)

It is 16-bit wide register. It is separated 2 parts as CCAPnH for storing high byte data and CCAPnL for storing low byte data. Each PCA module has a CCAP register. Data in CCAP register is used for determination the duty cycle of PWM signal when PCA moduel operate in PWM mode.

The related register which work with this register in PWM generator are CH and CL. Both register works to 16-bit PCA counter register. CH keeps high byte counting data and CL keeps low byte data. CH and CL will increase following the condition that select from CMOD register. If CH and CL values reach to CCAP’s values, the interrupt will occurred. User can use this operation to generates the PWM signal.

4.2.3.3 CMOD : PCA Mode register

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIDL</td>
<td>WDTE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CPS1</td>
<td>CPS0</td>
<td>ECF</td>
</tr>
</tbody>
</table>

It is 8-bit wide register. Address location is 0D9H. It has 2 bit relate PWM generation; CPS1 and CPS0. They are used for PCA clock source selection. Detail of selecttion as:

<table>
<thead>
<tr>
<th>CPS1</th>
<th>CPS0</th>
<th>Clock source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Clock frequency is 1/6 of main clock frequency in 6 cycle mode</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Clock frequency is 1/2 of main clock frequency in 6 cycle mode</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Internal clock from Timer0 overflow</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>External clock from P1.2/ECI pin</td>
</tr>
</tbody>
</table>

4.2.3.4 CCON : PCA control register

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF</td>
<td>CR</td>
<td>-</td>
<td>CCF4</td>
<td>CCF3</td>
<td>CCF2</td>
<td>CCF1</td>
<td>CCF0</td>
</tr>
</tbody>
</table>

It is 8-bit wide register bit accessable. Address location is 0D8H. Its function is controlling counter and shows interrupt status include overflow flag. Only one bit of this register that use in PWM generation. It is CR bit (bit 6 of CCON). This bit is used for turning on and off the counter in PCA module. Thus, this bit is enable bit of the PWM generation.
4.2.4 How to generate PWM from PCA module

All the PCA modules can be used as PWM outputs. The output frequency depends on the source for the PCA timer. All the modules will have the same output frequency because they all share the PCA timer. The duty cycle of each module is independently variable using the module’s capture register CCAPnL. The operation diagram is shown in figure 4-4. The CEX pin is set to output. PWM duty cycle in each module is set from the value of CCAPnL register. Enabling the PWM generation is determined from setting PWM and ECOM bits in CCAPM register.

When the value of the PCA CL register is less than the value in the module’s CCAPnL register the output will be low, when it is equal to or greater than it, the output will be high. When CL overflows from FFH to 00H, CCAPnL is reloaded with the value in CCAPHn. This allows the PWM to be updated without glitches. The PWM and ECOM bits in the module’s CCAPMn register must be set to enable the PWM mode.

Figure 4-4 PWM operating diagram in PCA module of T89C51AC2 microcontroller in Robo-51 controller board
4.2.5 Creating the motor driver function of C programming

4.2.5.1 Pin assignment

Refer the DC motor driver circuit of Robo-51, the port pin of T89C51AC2 that is connected with L293D h-Bridge driver has 6 pins as follows:

P2.0 and P2.1 are connected with IN1A and IN2A of L293D for control the motor output 1 (1Y and 2Y pin)

P1.6 is connected with 12EN the enable pin of L293D. P1.6 pin send the PWM signal to control the speed of motor output A at 1Y and 2Y pin of L293D.

P2.2 and P2.3 are connected with IN4A and IN3A of L293D for control the motor output 2 (3Y and 4Y pin)

P1.7 is connected with 34EN the enable pin of L293D. P1.7 pin send the PWM signal to control the speed of motor output B at 3Y and 4Y pin of L293D.

In creating the motor driver function for Robo-51 must refer some assignment as:

See the robot top view, Motor driver 1 or Motor A means the left side motor and Motor driver 2 or Motor B means the right side motor.

Back side of the Robo-51 is the side that attach DC motor gearboxes

Front side of the Robo-51 is the side that near 3-LED indicators

The figure 4-5 shows the reference position.

Figure 4-5 shows the reference position of Robo-51 for making the movement function.
4.2.5.2 Forward movement function

Driving the Robo-51 to moves forward direction, must drive both motors to forward. All movements programming in Robo-51 should make to functions, they help you to make the program easier and more powerful.

For the Forward movement, will make the motor_fd function. Listing 4-1 shows sourcecode of this function. This function need 2 parameters in operation as:

- select_fd : select motor output channel
  - "1" - Motor output 1 (M-1)
  - "2" - Motor output 2 (M-2)
- speed : determine the motor speed. Range is 0 to 255. At 255 is highest speed.

```c
#define motor_pulse 115 // Define constant
sbit dir_a1 = P2^0; // bit drive motor0
sbit dir_a2 = P2^1; // bit drive motor0
sbit dir_b1 = P2^2; // bit drive motor1
sbit dir_b2 = P2^3; // bit drive motor1

void motor_fd(unsigned char select_fd, unsigned char speed)
{
    TMOD &= 0xF0; // Refresh mode timer 0
    TMOD |= 0x02; // Setup mode timer 0 (8 bit auto reload)
    TH0 = motor_pulse; // Reload value for timer 0
    TL0 = motor_pulse; // Initial value for count of timer 0
    TR0 = 1; // Start timer 0
    CMOD = 0x40; // Set CMOD PCA count freq. by pulse overflow timer0
    if(select_fd==1)
    {
        CCAPM3 = 0x42; // Set CCAP Module 3 as 8 bit PWM
        CCAP3L = 255-speed; // Set CCAP3L initial value by speed variable
        // (set duty cycle)
        CCAP3H = 255-speed; // Set CCAP3H initial value by speed variable
        // (set duty cycle)
        dir_a1 = 0; // Set direction forward
        dir_a2 = 1; // Set direction forward
    }
    else if(select_fd==2)
    {
        CCAPM4 = 0x42; // Set CCAP Module 3 as 8 bit PWM
        CCAP4L = 255-speed; // Set CCAP4L initial value by speed variable
        // (set duty cycle)
        CCAP4H = 255-speed; // Set CCAP4H initial value by speed variable
        // (set duty cycle)
        dir_b1 = 0; // Set direction forward
        dir_b2 = 1; // Set direction forward
    }
    CCON = 0x40; // Set PCA counter run
}
```

Listing 4-1 Sourcecode of motor_fd function for forward movement
4.2.5.3 Backward movement function

Driving the Robo-51 to moves backward direction, must drive both motors to backward. For the Backward movement, will make the motor_bk function. Listing 4-2 shows sourcecode of this function. This function need 2 parameters in operation as:

- **select_bk**: select motor output channel
  
  “1” - Motor output 1 (M-1)
  
  “2” - Motor output 2 (M-2)

- **speed**: determine the motor speed. Range is 0 to 255. At 255 is highest speed.

```c
#define motor_pulse 115 // Define constant
sbit dir_a1 = P2^0; // bit drive motor0
sbit dir_a2 = P2^1; // bit drive motor0
sbit dir_b1 = P2^2; // bit drive motor1
sbit dir_b2 = P2^3; // bit drive motor1

void motor_bk(unsigned char select_bk ,unsigned char speed)
{
    TMOD &= 0xF0; // Refresh mode timer 0
    TMOD |= 0x02; // Setup mode timer 0 (8 bit auto reload)
    TH0 = motor_pulse; // Reload value for timer 0
    TL0 = motor_pulse; // Initial value for count of timer 0
    TR0 = 1; // Start timer 0
    CMOD = 0x04; // Set CMOD PCA count freq. by pulse overflow timer0
    if(select_bk==1)
    {
        CCAPM3 = 0x42; // Set CCAP Module 3 as 8 bit PWM
        CCAP3L = 255-speed; // Set CCAP3L initial value by speed variable
            // (set duty cycle)
        CCAP3H = 255-speed; // Set CCAP3H reload value by speed variable
            // (set duty cycle)
        dir_a1 = 1; // set direction backward
        dir_a2 = 0; // set direction backward
    }
    else if(select_bk==2)
    {
        CCAPM4 = 0x42; // Set CCAP Module 3 as 8 bit PWM
        CCAP4L = 255-speed; // Set CCAP4L initial value by speed variable
            // (set duty cycle)
        CCAP4H = 255-speed; // Set CCAP4H initial value by speed variable
            // (set duty cycle)
        dir_b1 = 1; // set direction backward
        dir_b2 = 0; // set direction backward
    }
    CCON = 0x40; // Set PCA counter run
}
```

Listing 4-2 Sourcecode of motor_bk function for backward movement
4.2.5.4 Brake function

Braking movement the Robo-51 need 2 operaitons. One is send logic “1” or “HIGH” to direction input of motor drive circuit for locking the motor’s shaft and Stop PWM signal.

This function is motor_stop. Listing 4-3 shows sourcecode of this function. It needs only one parameters as:

select_stop : select motor output channel to brake

select_stop = 1 → Brake only M-1 channel

select_stop = 2 → Brake only M2 channel

select_stop = 3 → Brake both channels

sbit dir_a1 = P2^0; // bit drive motor0
sbit dir_a2 = P2^1; // bit drive motor0
sbit dir_b1 = P2^2; // bit drive motor1
sbit dir_b2 = P2^3; // bit drive motor1

void motor_stop(char select_stop)
{
    if(select_stop==1)
    {
        dir_a1 = 0; // Break motor channel 1
        dir_a2 = 0; // Break motor channel 1
        CCAPM3 &= 0xFD; // Stop generate PWM at pin P1.6
    }
    else if(select_stop==2)
    {
        dir_b1 = 0; // break motor channel 2
        dir_b2 = 0; // Break motor channel 2
        CCAPM4 &= 0xFD; // Stop generate PWM at pin P1.7
    }
    else if(select_stop==3)
    {
        dir_a1 = 0; // Break motor channel 1
        dir_a2 = 0; // Break motor channel 1
        CCAPM3 &= 0xFD; // Stop generate PWM at pin P1.6
        dir_b1 = 0; // Break motor channel 2
        dir_b2 = 0; // Break motor channel 2
        CCAPM4 &= 0xFD; // Stop generate PWM at pin P1.7
    }
}

Listing 4-3 Sourcecode of motor_stop function for braking movement
4.2.6 dc_motor.h : Motor control library for Robo-51

From 3 motor function in section 4.2.5, you can combine to create the new library file. Ut is dc_motor.h. Listing 4-4 shows in detail of code. You can make it with text editor program and save as in dc_motor.h file. Store this library file at folder C:\RISE\INC

The function description of this library can summarize as follows:

(1) motor_bk : Backward movement function

Function format:

void motor_bk(bit select_bk ,unsigned char speed)

Parameter:

select_bk - Select motor channel. Range is 1 and 2.
speed - Determine motor speed. Range is 0 to 255

(2) motor_fd : Forward movement function

Function format:

void motor_fd(bit select_fd, unsigned char speed)

Parameter:

select_fd - Select motor channel. Range is 1 and 2.
speed - Determine motor speed. Range is 0 to 255

(3) motor_stop : Brake motor function

Function format:

void motor_stop(char select_stop)

Parameter:

select_stop - Select motor channel to stop. Range is 1, 2 and 3
(3 means select both channels)
Robotics Experiment with MCS-51 microcontroller

Listing 4-4  Sourcecode of dcMotor.h library file. This library function is control direction of DC motor driver in Robo-51 (continue)
TR0 = 1; // Start timer 0
CMOD = 0x04; // Set CMOD PCA count freq. by pulse overflow timer0
if(select_fd==1)
{
    CCAPM3 = 0x42; // Set CCAP Module 3 as 8 bit PWM
    CCAP3L = 255-speed; // Set CCAP3L initial value by speed variable
    // (set duty cycle)
    CCAP3H = 255-speed; // Set CCAP3H initial value by speed variable
    // (set duty cycle)
    dir_a1 = 0; // Set direction forward
    dir_a2 = 1; // Set direction forward
}
else if(select_fd==2)
{
    CCAPM4 = 0x42; // Set CCAP Module 3 as 8 bit PWM
    CCAP4L = 255-speed; // Set CCAP4L initial value by speed variable
    // (set duty cycle)
    CCAP4H = 255-speed; // Set CCAP4H initial value by speed variable
    // (set duty cycle)
    dir_b1 = 0; // Set direction forward
    dir_b2 = 1; // Set direction forward
}
CCON = 0x40; // Set PCA counter run

/{************************** Break motor *****************************************/
void motor_stop(char select_stop)
{
    if(select_stop==1)
    {
        dir_a1 = 0; // Break motor channel 1
        dir_a2 = 0; // Break motor channel 1
        CCAPM3 &= 0xFD; // Stop generate PWM at pin P1.6
    }
    else if(select_stop==2)
    {
        dir_b1 = 0; // Break motor channel 2
        dir_b2 = 0; // Break motor channel 2
        CCAPM4 &= 0xFD; // Stop generate PWM at pin P1.7
    }
    else if(select_stop==3)
    {
        dir_a1 = 0; // Break motor channel 1
        dir_a2 = 0; // Break motor channel 1
        CCAPM3 &= 0xFD; // Stop generate PWM at pin P1.6
        dir_b1 = 0; // Break motor channel 2
        dir_b2 = 0; // Break motor channel 2
        CCAPM4 &= 0xFD; // Stop generate PWM at pin P1.7
    }
}
Activity 5
Driving the DC motor

A5.1 Open RIDE and create the new project. Make C code in Listing A5-1 and save as act05.c file. Build project to HEX file.

A5.2 Turn-on Robo-51 and download hex file to it. Turn power off and unplug the download cable.

A5.3 Lift the robot above the floor. Turn power on and select mode to RUN. Observe the operation.

Robot-51 will beep at once after that the motor gearbox at M-1 output turn forward. LED indicator lights in Green color. In same time, Motor M-2 turn backward. LED lights in Red color. Both motor will turn in 10 seconds to stop and beep again for ending.

Listing A5-1 The activity C program for simple driving DC motor of Robo-51

```c
/* ——————————————————————————————————*/
// Program : Basic driving motor
// Description: Drive motors on forward and backward for checking direction
// Filename : act05.c
// C compiler : RIDE 51 V6.1
/* ——————————————————————————————————*/
#include <C51ac2.h> // Declare T89C51AC2’s register
#include <sound_robo51.h> // Declare Sound library
#include <dc_motor.h> // Declare DC motor library
#include <delay_robo51.h> // Declare Delay library

void main()
{
    beep(); // Beep for beginning
    motor_fd(1,100); // Drive motor channel 1 forward
    motor_bk(2,100); // Drive motor channel 2 backward
    delay_ms(10000); // Delay 10 seconds for observation motor turning
    motor_stop(all); // Stop all motors
    beep(); // Beep for finishing
    while(1); // Endless loop
}
```
4.3 **Robo-51 Library**

From Chapter 3 to the previous section introduce many library files. This section will summarize all basic library files which is used in Robo-51 movement control. There is 4 main library files: *dc_motor.h*, *delay_robo51.h*, *sound_robo51.h* and *lcd_robo51.h*

### 4.3.1 *dc_motor.h* : Driving motor library

- **motor_fd** - Drive the one or two motors to forward direction with 255 speed level.
- **motor bk** - Drive the one or two motors to backward direction with 255 speed level.
- **motor_stop** - Brake the one or two motors (1, 2 and 3 parameyet value)

### 4.3.2 *delay_robo51.h* : Delay time library

- **delay_100us** - Delay time in 100 microsecond unit (0 to 65,535). The time value is approximation.
- **delay_ms** - Delay time in 1 millisecond unit (0 to 65,535). The time value is approximation.

### 4.3.3 *sound_robo51.h* : Sound generation library

- **sound** - Generates the sound signal in Hertz unit with duration.
- **beep** - Generates a beep at 500Hz frequency 0.1 second duration.

### 4.3.4 *lcd_robo51.h* : LCD interface library

- **lcd_command** - Send the command byte to LCD module.
- **lcd_text** - Send the text byte to LCD module.
- **lcd_init** - Initail LCD module to 4-bit mode interfacing.
- **lcd_putstr** - Send message to LCD module.
- **inttolcd** - Convert number to numerical text and send to LCD module.
4.4 Robo-51 movement programming

In real robot movement, there is 5 simple movements include Move Forward, Backward, Turn Left, Turn Right and Brake. You can modify and improve the dc_motor.h library file to add more functions. This section should be present.

4.4.1 run_fd : Move forward function

This function will drive the both motors to forward direction. A parameter in this function is time_spin. It is time duration of forward movement in millisecond unit. The C code of this function is shown below.

```c
#define pow 200
void run_fd(int time_spin)
{
    motor_fd(2,pow); // Motor channel 2 forward
    motor_fd(1,pow); // Motor channel 1 forward
    delay_ms(time_spin); // Delay time for robot driving forward
}
```
4.4.2 run_bk: Move backward function

This function will drive the both motors to backward direction. A parameter in this function is \texttt{time\_spin}. It is time duration of backward movement in millisecond unit. The C code of this function is shown below.

```c
#define pow 200
void run_bk(int time_spin)
{
    motor_bk(2,pow);  // Motor channel 2 backward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay\_ms(time\_spin);  // Delay time for robot driving backward
}
```

4.4.3 turn\_left: Turn left movement function

This function will drive the both motors in different direction. M-2 motor is driven to forward and M-1 motor is driven to backward direction. A parameter in this function is \texttt{time\_spin}. It is time duration of turn left movement in millisecond unit. The C code of this function is shown below.

```c
#define pow 200
void turn\_left(int time\_spin)
{
    motor_fd(2,pow);  // Motor channel 2 forward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay\_ms(time\_spin);  // Delay time for robot spining turn left
}
```
4.4.4 turn_right : Turn right movement function

This function will drive the both motors in different direction. M-2 motor is driven to backward and M-1 motor is driven to forward direction. A parameter in this function is time_spin. It is time duration of turn right movement in millisecond unit. The C code of this function is shown below.

```c
#define pow 200
void turn_right(int time_spin)
{
    motor_bk(2,pow); // Motor channel 2 backward
    motor_fd(1,pow); // Motor channel 1 forward
    delay_ms(time_spin); // Delay time for robot spinning turn right
}
```

4.4.5 Stop : Stop the robot movement function

This function will brake both motor operations. The C code of this function is shown below.

```c
void stop(void)
{
    motor_stop(3); // Brake both motor outputs
    // (call from dc_motor.h library file)
}
```
Activity 6
Simple movement control of Robo-51

A6.1 Open RIDE and create the new project. Make C code in Listing A6-1 and save as act0601.c file. Build project to HEX file.

A6.2 Turn-on Robo-51 and download hex file to it. Turn power off and unplug the download cable.

/*---------------------------------------------*/
// Program : Basic Move
// Description: Robo-51 movement by simple function
// Filename : act0601.c
// C compiler : RIDE 51 V6.1
/*---------------------------------------------*/
#include <C51ac2.h>    // Header include register of P89C51AC2
#include <lcd_robo51.h> // LCD Module function
#include <dc_motor.h>   // DC motor driving module function
#include <sound_robo51.h> // Sound generation module function
#include <delay_robo51.h> // Delay time module function
#define pow 200        // Define constant for DC motor power driving.
void run_fd(int time_spin)
{
    motor_fd(2,pow);    // Motor channel 2 forward
    motor_fd(1,pow);    // Motor channel 1 forward
    delay_ms(time_spin); // Delay time for forward movement
}
void run_bk(int time_spin)
{
    motor_bk(2,pow);    // Motor channel 2 backward
    motor_bk(1,pow);    // Motor channel 1 backward
    delay_ms(time_spin); // Delay time for backward movement
}
void main()
{
    while(1)    // Infinite loop
    {
        run_fd(2000);    // Run forward 2 seconds
        beep();          // Beep at once
        run_bk(2000);    // Run backward 2 seconds
        beep();          // Beep at once
    }
}
Listing A6-1 The C program for Robo-51 basic movement activity
/* ————————————————————————————————————————————————— ——————————————————————————————————*/
// Program : Basic Move -2
// Description : Robo-51 movement with display operation on LCD
// Filename : act0602.c
// C compiler : RIDE 51 V6.1
/* ————————————————————————————————————————————————— ——————————————————————————————————*/
#include <C51ac2.h>  // Header include register of P89C51AC2
#include <lcd_robo51.h>  // LCD Module function
#include <dc_motor.h>  // DC motor driving module function
#include <sound_robo51.h>  // Sound generation module function
#include <delay_robo51.h>  // Delay time module function
#define pow 200  // Define constant for power drive DC motor

void run_fd(int time_spin)
{
    motor_fd(2,pow);  // Motor channel 2 forward
    motor_fd(1,pow);  // Motor channel 1 forward
    delay_ms(time_spin);  // Delay time for forward movement
}

void run_bk(int time_spin)
{
    motor_bk(2,pow);  // Motor channel 2 backward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay_ms(time_spin);  // Delay time for backward movement
}

void turn_left(int time_spin)
{
    motor_fd(2,pow);  // Motor channel 2 forward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay_ms(time_spin);  // Delay time of turn left
}

void turn_right(int time_spin)
{
    motor_bk(2,pow);  // Motor channel 2 backward
    motor_fd(1,pow);  // Motor channel 1 forward
    delay_ms(time_spin);  // Delay time of turn right
}

void main()
{
    lcd_init();  // Initial LCD module
    while(1)  // Infinite loop
    {
        lcd_command(0x01);  // Clear display
        lcd_putstr(line1," Forward");  // Show message
        run_fd(2000);  // Run forward 2 seconds
        lcd_command(0x01);  // Clear display
        lcd_putstr(line1," Turn Left");  // Show message
        turn_left(500);  // Turn left 0.5 second
        lcd_command(0x01);  // Clear display
        lcd_putstr(line1," Forward");  // Show message
        run_fd(2000);  // Run forward 2 seconds
        lcd_command(0x01);  // Clear display
        lcd_putstr(line1," Turn Right");  // Show message
        turn_right(500);  // Turn Right 0.5 second
    }
}
A6.3 Place the robot on the floor. Turn power-on and select mode to RUN. Observe the operation.

Robo-51 moves alternate to forward 2 seconds and backward 2 seconds. Every change movement the beep sound is driven.

If the operation incorrect, must change the direction of motor cable connection to RBX-51AC2 board. Change until the operation correct. In forward movement, both motor’s indicators are Green and Red in backward movement.

After all correct, must fix this motor connection. Because we must use in all activities from here.

A6.4 Create the new project. Make C code in Listing A6-2 and save as act0602.c file. Build project to HEX file.

A6.5 Turn-on Robo-51 and download hex file to it. Turn power off and unplug the download cable.

A6.6 Place the robot on the floor. Turn power-on and select mode to RUN. Observe the operation.

Robo-51 moves following the pattern belo and shows the movement message at LCD screen.

1. Move forward 2 seconds and shows message Forward.
2. Turn left 0.5 second and shows message Turn Left.
3. Move forward 2 seconds and shows message Forward again.
4. Turn right 0.5 second and shows message Turn Right.
5. Loop to step (1) to (4) continue.
Robotics Experiment with MCS-51 microcontroller
Chapter 5
Sensor interfacing

There are a wide variety of sensors used in mobile robots. In Robo-51 kit provides many sensors such as Touch sensor, Infrared Reflector sensor and Object detector. Two kinds of sensor will describe in this chapter. There are the Digital sensor and Analog sensor.

5.1 Simple digital sensor interfacing

The basically digital sensor is Switch or Touch sensor. Because this sensor gives the digital signal “HIGH” or “LOW” for detection.

Figure 5-1 is shown the schematic diagram of ZX-01 Switch input module. The switch is the mechanical extension and the ground electrical connection of a normally open, singlepole, single-throw switch. Normally it sends logic “HIGH” at output. When press or touch the switch, it will connect to ground. Output is logic “LOW” and LED indicator lights up.

The T89C51AC2 microcontroller on Robo-51 can be programmed to detect when a switch is pressed. I/O pins connected to each switch circuit monitor the voltage at the 10 kΩ pull-up resistor.

All I/O pins default to input every time a program starts. This means that the I/O pins connected to the switch will function as inputs automatically. As an input, an I/O pin connected to a switch circuit will cause its input register to store a 1 if the voltage is 5 V (switch not pressed) or a 0 if the voltage is 0 V (switch pressed).

The Robo-51 prepares 5 digital I/O for interfacing the digital sensors as P3.2, P3.3, P3.4, P4.0 and P4.1. Only P3.2 and P3.3 pin support the external interrupt from the switches.
Activity 7

Reading the Switch with polling method

A7.1 Open RIDE and create the new project. Make C code in Listing A7-1 and save as act07.c file. Build project to HEX file.

A7.2 Turn-on Robo-51 and download hex file to it. Turn power off and unplug the download cable.

A7.3 Connect the ZX-01 switch module to P3.2 on the RBX-51AC2 controller board of Robo-51.

A7.4 Turn power on. The program will run. Try to press the switch and observe the operation.

At LCD module shows message Input P3.2 Read at above line and show sensor status (“0” or “1”) at bottom line. If switch not pressed, the status shows “0” and “1” if switch is pressed.

Listing A7-1 The C program for reading digital sensor activity of Robo-51
Activity 8
Reading the Switch with interrupt

A8.1 Open RIDE and create the new project. Make C code in Listing A8-1 and save as act08.c file. Build project to HEX file.

```c
/* ————————————————————————————————————*/
// Program : Reading digital input
// Description: Reading digital signal from switch input board by interrupt
// Filename : act08.c
// C compiler : RIDE 51 V6.1
/* ————————————————————————————————————*/
#include <C51ac2.h> // Header include register of P89C51AC2
#include <lcd_robo51.h> // Module function LCD display
#include <delay_robo51.h> // Module function Delay

unsigned int count=0; // Declare count variable
void service_external0(void) interrupt 0 // External interrupt service routine
{
    lcd_clear(); // Clear display
    lcd_putstr(line1," Count:"); // Show "Count" message
    count = 0; // Clear counter
}

void main(void)
{
    EX0 = 1; // Enable external interrupt on P3.2 (EX0)
    IT0 = 1; // Select active edge of input signal at P3.2
    EA = 1; // Enable global interrupt
    lcd_init(); // LCD initialize
    lcd_putstr(line1," Count:"); // Show "Count:"
    while(1) // Loop
    {
        inttolcd(0x88,count,Dec); // Show counter on LCD
        delay_ms(100); // Delay 0.1second
        count++; // Increase counter
    }
}

Program description

Program begin with count up from 0 to 65,535 every 0.1 second approximation. The count value store in count variable and send to display at LCD module. If switch at P3.2 pin is pressed during counting. The interrupt is occurred. At the header of program, enable the external interrupt at P3.2 or EX0. When the interrupt is occurred, CPU will jump to interrupt service routine at service_external0.

At interrupt service routine, begins with clear screen of LCD module command following send message Count to show at LCD screen and reset the counting value to zero (count = 0). After that CPU will back to main program to start again.

Listing A8-1 The C program for reading switch sensor with interrupt method
A8.2 Turn-on Robo-51 and download hex file to it. Turn power off and unplug the download cable.

A8.3 Connect the ZX-01 switch module to P3.2 on the RBX-51AC2 controller board of Robo-51.

A8.4 Turn power on. The program will run. Try to press the switch and observe the operation.

When program start, LCD module shows the counting up value.

\[
\text{Count: } 0 \rightarrow \ldots \rightarrow \text{Count: 65535}
\]

If switch is pressed, the value reset to zero and re-start counting.

Before switch is pressed \hspace{1cm} \text{Count: 100 or the other value.}

After switch is pressed \hspace{1cm} \text{Count: 0 and count again.}
5.2 Analog sensor interfacing

Robo-51 has 6 analog inputs for interfacing with analog sensors. The analog to digital converter module within microcontroller of controller board can accept voltage not over 3V. But with attenuator circuit can help the controller board receives the voltage reach to +5V. Thus, Robo-51 can interface with many analog sensors. The conversion resolution is 10-bit. It means you can get the result in range 0 to 1,023. Totally is 1,024 values (calculated from $2^{10} = 1024$).

The math relation between $V_{\text{in}}$ (analog voltage input) with the conversion data ($A$) from Analog to Digital Converter can show below:

$$ V_{\text{in}} = A \times \frac{5}{1023} $$

by $5$ is the supply voltage +5V

1023 is maximum value of conversion data.

All 6 analog input of Robo-51 will connected the analog port of T89C51AC2 as follows:

<table>
<thead>
<tr>
<th>Analog input channel</th>
<th>Port pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>P1.0</td>
</tr>
<tr>
<td>1</td>
<td>P1.1</td>
</tr>
<tr>
<td>2</td>
<td>P1.2</td>
</tr>
<tr>
<td>3</td>
<td>P1.3</td>
</tr>
<tr>
<td>4</td>
<td>P1.4</td>
</tr>
<tr>
<td>5</td>
<td>P1.5</td>
</tr>
</tbody>
</table>

5.2.1 Control register of A/D converter

5.2.1.1 ADCF : ADC Configuration register

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH7</td>
<td>CH6</td>
<td>CH5</td>
<td>CH4</td>
<td>CH3</td>
<td>CH2</td>
<td>CH1</td>
<td>CH0</td>
</tr>
</tbody>
</table>

It is 8-bit wide register. Address is F6H. This register is set to use P1.x as A/D converter input and clear to use P1.x as standard digital input/output port.
5.2.1.2 ADCON : ADC Control Register

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5</th>
<th>bit 4</th>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSIDLE</td>
<td>ADEN</td>
<td>ADEOC</td>
<td>ADSST</td>
<td>SCH2</td>
<td>SCH1</td>
<td>SCH0</td>
</tr>
</tbody>
</table>

It is 8-bit wide register. Address is F6H. Detail of each bit are:

**PSIDLE** (bit 6) : **Pseudo Idle Mode (Best Precision)**

“0” - Clear to convert without idle mode.

“1” - Set to put in idle mode during conversion

**ADEN** (bit 5) : **Enable/Standby Mode**

“0” - Standby mode (power dissipation 1 mW).

“1” - enable ADC

**ADEOC** (bit 4) : **End Of Conversion**

“0” - ADC module still in conversion

“1” - ADC convert ready

Set by hardware when ADC result is ready to be read. This flag can generate an interrupt. Must be cleared by software.

**ADSST** (bit 3) : **Start and Status**

“0” - Cleared by hardware after completion of the conversion

“1” - Start an A/D conversion

**SCH2:0** (bit 2:0) : **Selection of Channel to Convert**

<table>
<thead>
<tr>
<th>SCH2</th>
<th>SCH1</th>
<th>SCH0</th>
<th>Analog input channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>AN0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>AN1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>AN2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>AN3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>AN4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>AN5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>AN6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>AN7</td>
</tr>
</tbody>
</table>
5.2.1.3 IEN1: Interrupt Enable Register

It is 8-bit wide register. Address is E8H. There is only one bit in using A/D converter. It is **EADC** (ADC Interrupt Enable bit - bit 1). The interrupt is occurred when ADC module convert signal complete and the ADEOC flag bit is set. If this bit is not set, the interrupt cannot occur.

The ADC interrupt vector address is 0043H. The interrupt number is 8 when make the interrupt service routine with C programming.

5.2.1.4 ADDH and ADDL: ADC data register

They are registers which store the data result from conversion. ADDH address is F5H and F4H for ADDL register.

ADDH register stores 8-bit upper (bit 9 to 2) of the conversion result.

ADDL register stores 2-bit lower (bit 1 and 0) of the conversion result.

5.2.2 Reading the A/D conversion

5.2.2.1 Conversion flag checking

This reading can summarize the step as:

1. Set the port to ADC input via ADCF register. The example is
   
   ```
   ADCF = 0x01;  // Select AN0
   ADCF = 0x03;  // Select AN0 and AN1
   ADCF = 0x84;  // Select AN7 and AN2
   ADCF = 0xFF;  // Select all analog input
   ```

2. Set ADEN bit to enable the ADC. The ADEN is bit 5 of ADCON register. The sample C code is
   
   ```
   ADCON = 0x20;
   ```

3. Select the input reading via SCH2 to SCH0 bits of ADCON register. The sample C code is

   ```
   ADCON &= 0xF8;  // Clear all SCH2 to SCH0 bits
   ADCON |= channel;  // Select input with the value of channel. The value is 0 to 7.
   ```

   If select to read AN3 input, the C code is

   ```
   ADCON &= 0xF8;
   ADCON |= 3;
   ```

4. Set ADSST bit to start the conversion. ADSST is bit 3 of ADCON register. After conversion complete, ADSST bit will clear automatically. The sample C Code is

   ```
   ADCON |= 0x08;
   ```
(5) After step (4), ADC module need time to convert the analog signal to digital data and store the data to ADDH and ADDL register. After conversion finish, the ADEOC flag (bit 4 of ADCON register) is set automatically. Thus, in C programming must make the program to polling the status of this flag. When ADEOC flag is set, it means the conversion finish. Must clear ADEOC flag before read the conversion data. The sample C code is

```c
while((ADCON & 0x10) != 0x10);
    // Wait until ADEOC flag is set.
ADCON &= 0xEF; // Clear ADEOC flag before read the result.
```

(6) Read the 10-bit conversion data from ADDH and ADDL register by

- **Data in ADDH register is 8-bit upper or bit 9 to 2.**
- **Data in ADDL register is 2-bit lower or bit 1 and 0**

In reading full 10-bit, the variable must declare to integer. The sample C code is

```c
int data; // Declare the result variable as integer.
data = (ADDH<<2) + ADDL;
    // Read data to store in dat variable
```

### 5.2.2.2 Reading with interrupt

This reading can summarize the step as:

1. Set the port to ADC input via ADCF register.
2. Set ADEN bit to enable the ADC.
3. Enable the global interrupt and ADC interrupt with set EA and EADC bit.
   
   The sample C code is
   ```c
   EA = 1; // Enable global interrupt
   EADC = 1; // Enable ADC interrupt.
   ```
4. Select the input reading via SCH2 to SCH0 bits of ADCON register.
5. Set ADSST bit to start the conversion.
6. After step (4), ADC module need time to convert the analog signal to digital data and store the data to ADDH and ADDL register. After conversion finish, the interrupt is occurred. CPU will jump to execute at interrupt service routine. In that routine must clear ADEOC bit first. After that read the 10-bit conversion result. The sample of interrupt service routine is shown as follows

```c
```
int adc_dat; // Declare the conversion variable
void ADC_service() interrupt 8 // Interrupt number is 8.
{
    EADC = 0; // Disable interrupt
    ADCON &= 0xEF; // Clear ADEOC bit.
    adc_dat = (ADDH<<2)+(ADDL); // Read data to store at adc_dat
    ADCON |= 0x48; // Set ADEN and ADSST bit
                   // for next conversion
    EADC = 1; // Enable interrupt
               // for next conversion.
}

5.2.3 Reading A/D converter function

From all step that describe in previous section, can make the suitable function for
reading the A/D conversion and save as the library file adc_robo51.h into C:\ RIDE\ INC. The full source
code is shown in Listing 5-1.

Example 5-1

int dat; // Declare integer variable for getting data from
          // analog function
data = analog(2); // Read the analog input 2 (P1.2) and store to dat
          // variable.

Listing 5-1 Sourcecode of adc_robo51.h library file. This library function is
reading the A/D conversion
Activity 9
Reading the Analog signal

A9.1 Open RIDE and create the new project. Make C code in Listing A9-1 and save as act09.c file. Build project to HEX file.

A9.2 Turn-on Robo-51 and download hex file. Turn power off and unplug the download cable.

A9.3 Connect the ZX-Potentiometer module to AN0 input on the RBX-51AC2 controller board of Robo-51.

A9.4 Turn power on. The program will run. Try to turn the potentiometer shaft and observe the operation.

The conversion result is shown at LCD module following adjustment of the potentiometer. The data are 0 to 1023. However the maximum result may be not reach to 1023 depend on battery level.

Listing A9-1 The C program for reading A/D conversion activity of Robo-51

```c
/*—————————————————————————————————————*/
// Program : Reading digital data from A/D conversion
// Filename: act09.c
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
#include <C51ac2.h>  // Header include register of P89C51AC2
#include <lcd_robo51.h>  // Module function LCD display
#include <delay_robo51.h>  // Module function Delay
#include <adc_robo51.h> // Module function analog to digital converter
void main(void)
{
    unsigned int val;  // Declare analog data variable, integer type is recommended
    lcd_init();  // Initial LCD module
    while(1)  // Looping
    {
        lcd_putstr(line1," Analog CH0 Read");  // Show title message
        val = analog(0);  // Read digital data from conversion to variable
        inttolcd(0xC7,val,Dec);  // Convert digital data to show on LCD display
        delay_ms(1000);  // Delay 1 second
        lcd_clear();
    }
}
```
Chapter 6

Line following mission

The popular activity of the educational robot is Line following or Line tracking. Purpose of this activity is learn about how to interface analog sensor. In Robo-51 robot kit prepares a pair of Infrared reflector sensor for this activity. Add senses to the Robo-51 so that it can detect and move following the line, by using the IR Reflector Sensor. Two IR Reflector Sensors will be installed at the bottom of the Robo-51 so that it can detect both white and black lines.

6.1 The Infrared reflector sensor

The heart of this sensor is TCRT5000 reflective object sensor. It is designed for close proximity infrared (IR) detection. There's an infrared diode behind its transparent blue window and an infrared transistor behind its black window. When the infrared emitted by the diode reflects off a surface and returns to the black window, it strikes the infrared transistor's base, causing it to conduct current. The more infrared incident on the transistor's base, the more current it conducts.

When used as an analog sensor, the ZX-03 can detect shades of gray on paper and distances over a short range if the light in the room remains constant.

The suitable distance from sensor to line or floor is during 3 to 8 mm. The output voltage is during 0.1 to 4.8V and digital value from 10-bit A/D converter is 20 to 1,000. Thus, ZX-03 will suitable to apply to line tracking sensor.

![Infrared Reflector Sensor Diagram](image)

Figure 5-1 ZX-03 Infrared Reflector information
Activity 10
Reading IR Reflector value

A10.1 Open RIDE and create the new project. Make C code in Listing A10-1 and save as act10.c file. Build project to HEX file.

A10.2 Turn-on Robo-51 and download hexfile. Turn power off and unplug the download cable.

A10.3 Connect the left ZX-03 Infrared Reflector module to AN5 and The right at AN4 input on the RBX-51AC2 controller board of Robo-51 (ready connection from building activity).

A10.4 Place the robot on the black area. Turn power on. The program will run. Observe the operation at LCD module. After that change the area to white color and run again.

At LCD module shows the value of AN4 input.

If the value over the reference ; 400, Robo-51 will drive a beep. Keep this operation for using in the following the line activity.

Listing A10-1 The C program for reading Infrared reflector sensor

```c
/*—————————————————————————————————————*/
// Program : Reading IR reflect sensor
// Description: Reading white and black surface from IR reflect sensor
// Filename : act10.c
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
#include <C51ac2.h>    // Header include register of P89C51AC2
#include <lcd_robo51.h> // Module function LCD display
#include <delay_robo51.h> // Module function Delay
#include <adc_robo51.h> // Module function Analog to digital converter
#include <sound_robo51.h> // Module function Sound
#define ref 400   // Define reference value as 400
void main(void)
{
    unsigned int reflect_dat; // Declare reflect_dat for keeping analog data
    lcd_init();    // Initialize LCD module
    lcd_putstr(line1,"Reflect CH4"); // Show message “Reflect CH4” on LCD
    while(1)    // Looping
    {
        reflect_dat = analog(4); // Read value from IR reflect sensor
        inttolcd(0xC7,reflect_dat,Dec); // Display analog value on LCD module
        if(reflect_dat>ref) // Compare analog value with reference
            beep(); // Beep if reflect_dat > reference
        delay_ms(500); // Delay 0.5 second
    }
}
```
6.2 Robo-51 versus the black line

This section presents how to detect the black line of Robo-51. First, read the value from Infrared Reflector sensor to calculate to find the decision value of black line and white surface. The sample C code is shown in Example 6-1.

**Example 6-1**

Determine the Infrared Reflector is connected with AN4 as Right sensor and AN5 is Left sensor.

```c
#include <C51ac2.h>
#include <delay.h>
#include <adc_robo51.h>
#include <lcd_robo51.h>
void main()
{
  lcd_init();
  while(1)
  {
    lcd_command(1);
    {  // Show letter “L” for the Left sensor
      lcd_putstr(0x80,"L");
      inttolcd(0x82,analog(5),Dec); // Read value from the Left sensor
      lcd_putstr(0xC0,"R");       // Show letter “R” for the Right sensor
      inttolcd(0xC2,analog(4),Dec); // Read value from the Right sensor
      delay_ms(100);               // Delay for display
    }
  }
}
```

From testing, in Black line detection the result is during 30 to 100 and white surface is during 600 to 800. **Therefore the reference value for detect the black line is 400.**

The sensor which detect the value lower 400 means found the black line or area. If higher 400 means sensor can detect the white area.
Activity 11
Robo-51 finds the black line

In this activity presents the finding and detection black line of Robo-51. The robot move forward always until detect the black line. After detection, robot will stop and drive a beep signal. See the figure A11-1 about the operation.

A11. Make the demonstration field by stick the 1-Inches width black tape 8-Inches long on the white surface.

Figure A11-1 shows the operation of Robo-51 for finding and detection the black line in this activity.
A11.2 Open RIDE and create the new project. Make C code in Listing A11-1 and save as act11.c file. Build project to HEX file.

```c
/*—————————————————————————————————————*/
// Program : Detect black line
// Description: Drive robot for finding the black line by IR reflect sensor
// Filename : act11.c
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
#include <C51ac2.h> // Header include register of T89C51AC2
#include <lcd_robo51.h> // LCD module function
#include <dc_motor.h> // DC motor module function
#include <sound_robo51.h> // Sound module function
#include <delay_robo51.h> // Delay time function
#include <adc_robo51.h> // ADC module function
#define pow 100 // Define constant for driving robot
#define ref 400 // Define constant the reference value for detection line

/***************** Function drive robot forward ******************************/
void run_fd(int time_spin)
{
    motor_fd(2,pow); // Drive left motor forward
    motor_fd(1,pow); // Drive right motor forward
    delay_ms(time_spin); // Delay for robot spin
}

/***************** Function drive robot backward ******************************/
void run_bk(int time_spin)
{
    motor_bk(2,pow); // Drive left motor backward
    motor_bk(1,pow); // Drive right motor backward
    delay_ms(time_spin); // Delay for robot spin
}

/***************** Function drive robot turn left ******************************/
void turn_left(int time_spin)
{
    motor_fd(2,pow); // Drive left motor forward
    motor_bk(1,pow); // Drive right motor backward
    delay_ms(time_spin); // Delay for robot spin
}

/***************** Function drive robot turn right ******************************/
void turn_right(int time_spin)
{
    motor_bk(2,pow); // Drive left motor backward
    motor_fd(1,pow); // Drive right motor forward
    delay_ms(time_spin); // Delay for robot spin
}

/********************** Main function ***********************************/
void main()
{
    int left=0,right=0; // Define variable for keep input analog value
    lcd_init(); // Initial LCD module
    beep(); // Sound beep 1 time
    while(1) // Infinite loop
```
Robotics Experiment with MCS-51 microcontroller

```c
{ left = analog(5); // Read input analog channel 5(connect left sensor)
  right = analog(4); // Read input analog channel 4(connect right sensor)

  if(left<ref || right<ref) // if both sensors detect the white area
  { 
    motor_stop(all); // Stop robot
    beep(); // Beep when both sensors detect the line
    lcd_clear(); // Clear LCD display
    if(left<ref)
    { 
      lcd_putstr(line1,"Left:>> Line"); // Shows Left sensor detects line
    } else
    { 
      lcd_putstr(line1,"Left:>> Floor"); // Shows Left sensor detects floor
    }
    if(right<ref)
    { 
      lcd_putstr(line2,"Right:>> Line"); // Shows Right sensor detects line
    } else
    { 
      lcd_putstr(line2,"Right:>> Floor"); // Shows Right sensor detects floor
    }
    while(1); // Break program
  }
  else
  { 
    run_fd(10); // Drive robot forward
  }
}
```

**Program description**

When program start, Robo-51 drive a beep for beginning. Robo-51 polls to read the value from both analog inputs which connect the Infrared Reflectors. The value from Left sensor stores to left variable and Right sensor to right variable. Compare both value with the reference value; ref. This is comparative condition:

- **left<ref**
  
  "True" means the Left sensor detects the black line.
  
  "False" means the Left sensor detects the white surface.

- **right<ref**
  
  "True" means the Right sensor detects the black line.
  
  "False" means the Right sensor detects the white surface.

From condition testing of `if(left<ref || right<ref)`, if the result is **true** the meaning will be

Listing A11-1 The C program for detection the black line of Robo-51 (continue)
Case #1: left<ref is “true” and right<ref is “false”
Case #2: left<ref is “false” and right<ref is “true”
Case #3: left<ref is “true” and right<ref is “true”
Robo-51 move forward until detects the black line. It stop and drive a beep and reports the detected condition from Case #1 to #3 on LCD module. At the upper line, it shows the operation of Left sensor and bottom line shows the operation of Right sensor.

Listing A11-1 The C program for detection the black line of Robo-51 (final)

A11.3 Turn-on Robo-51 and download hex file. Turn power off and unplug the download cable.
A11.4 Place the robot above the white surface at the position as follows (place one position to test first and change the position to testing in the next):
   (A) Right angle with the black line from step A11.1
   (B) 45 degree angle on the left with the black line
   (C) 45 degree angle on the right with the black line
Turn power on to run. Observe the robot movement and LCD displaying
The Robo-51 drive a beep and moves forward to find the black line always. At LCD module shows the result message of detection. Any sensor can detect the black line, LCD will shows the word “Line” when sensor can detect the line. After that, Robo-51 stop movement.
Activity 12
Robo-51 PING-PONG

In this activity presents how to control the Robo-51 to move in ZIG-ZAG format. The black line is turning point following the figure below. The Robo-51 moves forward until found the black line to change the direction. Loop this operation always. Thus, robot will move within space between both black lines.

Programming concept

The steps of programming in this activity are:

1. Find the decision value between the white surface and black line.
2. Read the sensor values and store to compare
3. Check the condition as follows
   - **Case #1**: Both sensor detect the white surface
     - **Action**: Robot moves forward.
   - **Case #2**: Left sensor detected the black line but Right sensor detected the white surface.
     - **Action**: Robot moves with rotate right by suitable value for changing direction to move to the black lines opposite side.
   - **Case #3**: Left sensor detected the white surface but Right sensor detected the black line.
     - **Action**: Robot moves with rotate left by suitable value for changing direction to move to the black lines opposite side.
   - **Case #4**: Out of 3 conditions above
     - **Action**: Control the robot to move following programmer purpose.
4. Back to step (2).
Making the demo field

Prepare the flat white surface size 2 metres or longer and 60 centimetre width or more. Stick two of the 1-Inche width and 2 metres long black lines parallel. The distance between the lines is at least 40 centimetre.

Activity procedure

A12.1 Open RIDE and create the new project. Make C code in Listing A12-1 and save as act12.c file. Build project to HEX file.

A12.2 Turn-on Robo-51 and download hexfile. Turn poweroff and unplug the download cable.

A12.3 Place the Robo-51 at start point. Turn around the robot to 45 degree following the demonstration figure in this activity.

A12.4 Turn power on and select to RUN mode. Observe the robot operation.

In this program, determine the turning time to 1 second approximation. It cause the turning angle $\phi$ (see the figure below). Programmer can adjust the time value to make the suitable angle for control the movement path correctly.

If the time value is more, the $\phi$ angle is narrow. It cause the robot moves back to same path. In the other hand, the time value is less, the robot should move out of the line or possible to moves parallel the lines with do not detect the lines.
Listing A12-1 The C program for PING-PONG movement of Robo-51 (continue)
Listing A12-1 The C program for PING-PONG movement of Robo-51 (final)
6.3 Robo-51 moves along the line programming

Next step of Robo-51 operation relate the line detection is Move along the line or following the line. This operation has 4 scenario as follows.

**Scenario-1**  Both sensor above the white surface

**Sensor operation :**
- Left sensor gives value over 400. It detected the white surface.
- Right sensor gives value over 400. It detected the white surface.

**Action :**
- This scenario can interpret the robot move bestride the line. The robot must be controlled so that it moves forward and delays briefly.

**Scenario-2**  Robot moves rightwards away of the line.

**Sensor operation :**
- Left sensor gives value lower 400. It detected the black line.
- Right sensor gives value over 400. It detected the white surface.

**Action :**
- The robot must be controlled so that it turns left slowly and delays briefly.
**Scenario-3**  Robot moves leftwards away of the line.

Left sensor gives value over 400. It detected the white surface.
Right sensor gives value lower 400. It detected the black line.

*Action*:

The robot must be controlled so that it turns right slowly and delays briefly.

---

**Scenario-4**  The robot stay on the across line.

Left sensor gives value lower 400. It detected the black line.
Right sensor gives value lower 400. It detected the black line.

*Action*:

The robot has many options to choose from whether it is to move forward, turn left or turn right.
Activity 13
Robo-51 tracks the black line

A13.1 Open RIDE and create the new project. Make C code in Listing A13-1 and save as act13.c file. Build project to HEX file.

A13.2 Turn-on Robo-51 and download hex file. Turn power off and unplug the download cable.

A13.3 Prepare the demonstration field with the 1-inch width black line on the white surface. You can create the line pattern with your idea.

A13.4 Place the Robo-51 over the black line. Turn power on and select to RUN mode. Observe the robot operation.

```c
/* Program : Tracking line robot V1 */
// Description: Drive robot to tracking the black line
// Filename : act13.c
// C compiler : RIDE 51 V6.1 */
#include <C51ac2.h> // Header include register of T89C51AC2
#include <dc_motor.h> // DC motor module function
#include <sound_robo51.h> // Sound module function
#include <delay_robo51.h> // Delay time function
#include <adc_robo51.h> // ADC module function
#define pow 250 // Define constant power drive robot
#define ref 400 // Define constant reference compare track line
int left=0,right=0; // Define variable for keep input analog value

void run_fd(int time_spin)
{
    motor_fd(2,pow); // Drive left motor forward
    motor_fd(1,pow); // Drive right motor forward
    delay_ms(time_spin); // Delay for robot spin
}

void run_bk(int time_spin)
{
    motor_bk(2,pow); // Drive left motor backward
    motor_bk(1,pow); // Drive right motor backward
    delay_ms(time_spin); // Delay for robot spin
}

void turn_left(int time_spin)
{
    motor_fd(2,pow); // Drive left motor forward
    motor_bk(1,pow); // Drive right motor backward
    delay_ms(time_spin); // Delay for robot spin
}
```

Listing A13-1 The C program for Robo-51 line tracking activity (continue)
Robotics Experiment with MCS-51 microcontroller

*************** Function drive robot turn right ***********************
void turn_right(int time_spin)
{
    motor_bk(2,pow); // Drive left motor backward
    motor_fd(1,pow); // Drive right motor forward
    delay_ms(time_spin); // Delay for robot spin
}

************************** Main function ******************************
void main()
{
    beep(); // Sound beep 1 time
    while(1) // Infinite loop
    {
        left = analog(5); // Read input analog channel 5 (connect left sensor)
        right = analog(4); // Read input analog channel 4 (connect right sensor)

        if((left>ref && right>ref)
        { // If left sensor and right sensor detect white
            run_fd(20); // Drive forward
        }
        else if((left<ref && right>ref) // If left sensor detect black and
        { // right sensor detect white
            turn_left(20); // Drive turn left
        }
        else if((left>ref && right<ref) // if left sensor detect white and
        { // right sensor detect black
            turn_right(20); // Drive turn right
        }
        else if((left<ref && right<ref) // if both sensors detect black line
        { //
            run_fd(20); // Drive forward
            turn_left(20); // Drive turn left
        }
    }
}

Program description

The condition of decision in this program are 4 conditions as follows:

Listing A13-1 The C program for Robo-51 line tracking activity (continue)
if(left>ref && right>ref) // Both sensor detect the white
// surface.
{
    run_fd(20); // Move forward 20 millisecond
}

else if(left<ref && right>ref)
    // The Left sensor detects the line
    // but Right sensor detects the white
    // surface
{
    turn_left(20); // Turn left 20 millisecond
    // to adjust the both sensors bestride
    // the line.
}

else if(left>ref && right<ref)
    // The Left sensor detects the white
    // area but Right sensor detects the
    // black line
{
    turn_right(20); // Turn right 20 millisecond
    // to adjust the both sensors bestride
    // the line.
}

else if(left<ref && right<ref)
    // Both sensors detect the
    // black line
{
    run_fd(20); // Move forward 20 millisecond
    turn_left(20); // Turn left after found the
    // cross line
}

Listing A13-1 The C program for Robo-51 line tracking activity (final)
Chapter 7
Remote control experiment

Another feature of automatic robots is that it can receive commands from a far distance by using infrared light. This is similar to a remote-controlled robot except that the commands received are through serial communication.

Robo-51 provides an Infrared Remote control, called ER-4. ER-4 remote control will modulate serial data with infrared light. Robo-51 must install a 38kHz infrared receiver module for receiving.

7.1 ER-4 Infrared remote control 4 channels

It is a 4-channel general purpose infrared remote control. The signal output is the serial data which modulated with 38kHz carrier frequency. All operation is controlled by only one microcontroller. Figure 7-1 shows the photo, board layout and schematic of ER-4 remote control.

7.1.1 Features of ER-4 Remote control

- Operational distance is 4 to 8 meters in open space.
- The 4-channel switch operates in an on/off mode
- Uses low power; Automatically resumes power-save mode once data is sent
- Uses only 2.4-3.0 V from two AA batteries - both regular and rechargeable.
- Transmits serial data using the RS-232 standard with 1200 bps baud rate and 8N1 data format (8 data bit, no parity, 1 stop bit)

Figure 7-1 shows the photo, board layout and schematic of ER-4 remote control.
7.1.2 Circuit description

The schematic of ER-4 Remote Control is shown in Figure 7-1 (C). It included less component. 4 of push-button switch are connected with GP0, GP1, GP2 and GP4 of IC1; PIC12F629 pre-programmed microcontroller. IC1 function is reading switch status and create 38kHz carrier frequency. It modulate all signal and send to Q1 transistor to drive signal via infrared LED.

7.1.3 Format of data sent by ER-4 Remote Control

To make it easier for the receiver to read the switch value from the remote control, the ER-4 transmit serial data according to the RS-232 standard, with a baud rate of 1,200 bps and 8N1 format. Characters are transmitted according to what switch is pressed on the remote. The switch positions are displayed in Figure 7-1 (B)

- **Press switch A**, the large cap A, followed by small cap A (a) is sent.
- **Press switch B**, the large cap B, followed by small cap B (b) is sent
- **Press switch C**, the large cap C, followed by small cap C (c) is sent
- **Press switch D**, the large cap D, followed by small cap D (d) is sent.

The reason that we have to alternate large cap and small cap letters is so that the receiver can differentiate if a user presses continuously or if the user represses. If a user represses, the large cap character will be sent the first time. If the user represses the same button again, the small cap character will be sent the second time. If the user presses continually, the last character will be sent repeatedly.

7.2 The 38kHz Infrared receiver module

In transmitting the data modulated with infrared light for long distance about 5 to 10 meters similar TV remote control. The carrier frequency is 38kHz. Thus, receiver must demodulate 38kHz carrier frequency. After this transfer serial data to microcontroller.

If the sensor does not detect the 38kHz frequency with the infrared light, the output will be logic “1”. Otherwise, if it detects the 38kHz frequency, the output logic is “0”.

![Infrared receiver module diagram](image)

Figure 7-2 shows the photo of 38kHz Infrared Receiver module, pin assignment and schematic diagram.
7.3 Making the Remote Control library file

For easier to interface Robo-51 with ER-4 Remote Control with C programming, the library file is required. This library function is decoder the switch data from ER-4 remote control.

7.3.1 Concept

Interfacing with ER-4 in Robo-51 assign the connection 38kHz Infrared Receiver module with RBX-51AC2 controller via P3.2. This port is external interrupt input. When detected the signal, interrupt is occurred. It is synchronize with Start bit of serial data. With this technique, the microcontroller can receive data 8-bit completely.

From this concept, we can make the C library file; remote.h. Listing 7-1 is full source code of this library file.

7.3.2 Function description of remote.h library

(1) get_remote : Decode the input data from ER-4 Remote Control

Function format:

    unsigned char get_remote(void)

Return value:

The switch identification value of ER-4 Remote Control in pattern as:

a, A, b, B, c, C, d and D

(2) clear_remote : Clear value of remote control buffer

Function format:

    void clear_remote(void)

There is 2 important things which programmer must know in using this library file. One is hardware connection of 38kHz Infrared Receiver module. Must connect with P3.2 of RBX-51AC2 controller board in Robo-51.

Two is buffer variable. All function in this library file need a variable. It is ir_command. All operation will work via this parameter.
/*—————————————————————————————————————*/
// Filename : remote.h
// Description: receive command from Easy remote to control robot
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
sbit irm_data = P3^2; // Define port input remote sensor
unsigned char ir_command=0; // Define for buffer command from Easy remote
/******************** Function Delay for baudrate 1200 *********************/
void delay_baud1200() // Delay baudrate 1200 bps
{
    unsigned int x,a; // Keep for counter loop
    for(x=0;x<1;x++)
    {
        for(a=0;a<731;a++); // Loop for delay 833 us
    }
}
/******************** Function Delay for baudrate 1200(for start bit) **********/
void start_1200() // Delay baudrate 1200 while start bit
{
    unsigned int x,a; // Keep for counter loop
    for(x=0;x<1;x++)
    {
        for(a=0;a<365;a++); // Loop for delay 416 us
    }
}
/************* Service interrupt routine for receive command *************/
void service_ex0(void) interrupt 0
{
    unsigned char i; // Define for counter loop
    if(irm_data==0) // Check start bit true?
    {
        start_1200(); // Delay for start bit
        for(i=0;i<8;i++) // For loop count 8 time(for receive data 8 bit)
        {
            delay_baud1200(); // Delay for data 1 bit
            ir_command = ir_command>>1; // Shift data bit to right 1 time
            if(irm_data) // If data bit = high
                ir_command = ir_command | 0x80; // Config data bit = "1"
        }
        delay_baud1200(); // Delay for stop bit
    }
}
/******************** Function read Command from Easy remote ***************/
unsigned char get_remote(void)
{
    return(ir_command); // Return command
}
/******************** Function clear command *****************************/
void clear_remote()
{
    ir_command = 0; // Clear command
}
/******************** Function initial for Easy remote *******************/
void remote_init(void)
{
    irm_data = 1; // Configuration input P3.2
    EX0 = 1; // Enable External interrupt0
    IT0 = 1; // Detect falling edge
    EA = 1; // Enable interrupt all
}
Listing 7-1 Sourcecode of remote1.h library file.
Activity 14

Reading the ER-4 Remote Control data

A14.1 Use a 3 x 10 mm screw and place it into the Infrared Receiver Module, followed by the 3 mm plastic spacer. Then place the obtuse joiner at the back of the module and used a 3 mm nut to screw it all in together tightly.

A14.2 Attach the Infrared Receiver module from step A14.1 in front of the robot by 3x10 mm. screw and nut.

A14.3 Connect the 38kHz Infrared Receiver module’s cable to P3.2 connector on RBX-51AC2 controller of Robo-51.

A14.4 Open RIDE and create the new project. Make C code in Listing A14-1 and save as act14.c file. Build project to HEX file.

A14.5 Turn-on Robo-51 and download hex file.

A14.6 Put 2 of AA battery in ER-4 Remote Control.

A14.7 Select the MODE switch on the RBX-51AC2 controller board of Robo-51 to RUN mode.
/* Program : Test Infrared receiving 
// Description: Control LED at P3.5 P3.6, P3.7 and sound by ER-4 Remote control 
// Filename: act14.c 
// C compiler : RIDE 51 V6.1 */

#include <C51ac2.h> // Header include register of P89C51AC2 
#include <delay_robo51.h> // Delay time module function 
#include <remote.h> // ER-4 Remote Control function 
#include <sound_robo51.h> // Sound module function 

sbit led1 = P3^5; // Define LED on P3.5 port 

sbit led2 = P3^6; // Define LED on P3.6 port 

sbit led3 = P3^7; // Define LED on P3.7 port 

void main()
{
    remote_init(); // Initialize ER-4 remote control 
    while(1) // Looping 
    {
        switch(get_remote()) // Check command from ER-4 remote control 
        {
            case 'a' : led1 = 0; // LED P3.5 on 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'A' : led1 = 1; // LED P3.5 off 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'b' : led2 = 0; // LED P3.6 on 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'B' : led2 = 1; // LED P3.6 off 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'c' : led3 = 0; // LED P3.7 on 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'C' : led3 = 1; // LED P3.7 off 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'd' : beep(); // Beep 1 time 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
            case 'D' : beep(); // Beep 1 time 
                        clear_remote(); // Clear command 
                        break; // Out of from case 
        }
    }
}

Program description

This program controls the microcontroller to get data from ER-4 and control the 3-LEDs at P3.5 to P3.7 port and Piezo speaker. The library file remote.h is included for interfacing the ER-4 Remote Control with get_remote and clear_remote function.

Program begins with initialize system for receiving the serial data with remote_init function. After that, program will read the button data from the return value of get_remote function with switch-case command. In each case has a break command to clear all buffer with clear_remote function. After clearing, program can run continue to get the new button.

Listing A14-1 The C program for reading ER-4 Remote control serial data
A14.8 Turn the ER-4 direction to Infrared Receiver module at Robo-511. Press any switch on ER-4 Remote control. Observe the operation.

Press **A** button at once:

*LED P3.5 on the RBX-51AC2 is ON.*

Press **A** button continue:

*LED P3.5 is ON and OFF.*

Press **B** button at once:

*LED P3.6 on the RBX-51AC2 is ON.*

Press **B** button continue:

*LED P3.6 is ON and OFF continue until release the switch*

Press **C** button at once:

*LED P3.7 on the RBX-51AC2 is ON.*

Press **C** button continue:

*LED P3.7 is ON and OFF continue until release the switch.*

Press **D** button at once:

*Piezo drives a beep.*

Press **D** button continue:

*Piezo drives beep signal continue until release the switch.*
Activity 15
IR control Robo-51
movement

A15.1 Open RIDE and create the new project. Make C code in Listing A15-1 and save as act15.c file. Build project to HEX file.

A15.2 Turn-on Robo-51 and download hex file. Turn power off and unplug the download cable.

A15.3 Select the MODE switch on the RBX-51AC2 controller board of Robo-51 to RUN mode.

A15.4 Turn power on. Use the ER-4 to control the Robo-51 robot. Observe the operation.

The operation of Robo-51 can summarize as follows:

- **Press D button**
  Robo-51 moves forward continue until press any button.

- **Press A button**
  Robo-51 moves backward continue until press any button.

- **Press C button**
  Robo-51 turn left. If still press the button, Robo-51 will turn left continue until release the button.

- **Press B button**
  Robo-51 turn right. If still press the button, Robo-51 will turn right continue until release the button.
/*—————————————————————————————————————*/
// Program : Remote control robot
// Description: Robot controlled by ER-4 remote control Robot receive command from remote control by serial communication at baud 1200 bps
// Filename : act15.c
// C compiler : RIDE 51 V6.1
/*—————————————————————————————————————*/
#include <C51ac2.h>  // Header include register of T89C51AC2
#include <delay_robo51.h>  // Delay time module function
#include <remote.h>  // ER-4 Remote Control function
#include <sound_robo51.h>  // Sound module function
#include <dc_motor.h>  // DC motor module function
#include <lcd_robo51.h>  // LCD module function
#define pow 200  // Define constant for power drive DC motor

void run_fd(int time_spin)
{
    motor_fd(2,pow);  // Motor channel 2 forward
    motor_fd(1,pow);  // Motor channel 1 forward
    delay_ms(time_spin);  // Delay time for robot drive forward
}

void run_bk(int time_spin)
{
    motor_bk(2,pow);  // Motor channel 2 backward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay_ms(time_spin);  // Delay time for robot drive backward
}

void turn_left(int time_spin)
{
    motor_fd(2,pow);  // Motor channel 2 forward
    motor_bk(1,pow);  // Motor channel 1 backward
    delay_ms(time_spin);  // Delay time for robot spin turn left
}

void turn_right(int time_spin)
{
    motor_bk(2,pow);  // Motor channel 2 backward
    motor_fd(1,pow);  // Motor channel 1 forward
    delay_ms(time_spin);  // Delay time for robot spin turn right
}

void main()
{
    beep();  // Beep 1 time
    remote_init();  // Initial remote
    while(1)  // Infinite loop
    {
        switch(get_remote())  // Check command for receive
        {
            case 'a': run_bk(100);  // Drive robot backward when receive "a"
                clear_remote();  // Clear command
                break;  // Out from case
            case 'A': run_bk(100);  // Drive robot backward when receive "A"
                clear_remote();  // Clear command
                break;  // Out from case
            case 'b': turn_right(100);  // Turn right when receive "b"
                motor_stop(all);  // Robot stop
                clear_remote();  // Clear command
                break;  // Out from case

Listing A15-1 The C program for remote controlling Robo-51 with ER-4 (continue)
case 'B' : turn_right(100);    // Turn right when receive "B"
    motor_stop(all);         // Robot stop
    clear_remote();          // Clear command
    break;                   // Out from case

    case 'c' : turn_left(100);  // Turn right when receive "c"
        motor_stop(all);     // Robot stop
        clear_remote();      // Clear command
        break;              // Out from case

    case 'C' : turn_left(100);  // Turn right when receive "C"
        motor_stop(all);     // Robot stop
        clear_remote();      // Clear command
        break;              // Out from case

    case 'd' : run_fd(100);    // Turn right when receive "d"
        clear_remote();      // Clear command
        break;              // Out from case

    case 'D' : run_fd(100);    // Turn right when receive "D"
        clear_remote();      // Clear command
        break;              // Out from case

    default : break;          // Out from case

Listing A15-1 The C program for remote controlling Robo-51 with ER-4 (final)
Chapter 8
Contactless object detection of Robo-51

The one popular application of intelligent educational robot is contactless object avoiding. Robo-51 also supports the application. The new sensor will suggest in this chapter is ZX-08 Infrared objector.

8.1 Introducing ZX-08 Infrared object detector

This sensor has both the infrared light receiver and transmitter within itself. The ZX-08 sensor can detect obstacles at a maximum distance of 6 centimeters. Figure 8-1 shows the schematic of ZX-08 Infrared Objector module.

The Rx signal cable must be connected to the digital input, while the Tx must be connected to any digital output on the RBX-51AC2 board. Once the signal cables are connected, sending logic "1" to Tx will make the infrared LED on the ZX-08 module light up. If there is an obstacle blocking in front, the infrared light will send reflect that object back to the infrared receiver, causing a logic "0" to be sent to the Robo-51 input port.

Figure 8-1 ZX-08 Infrared objector schematic
8.2 Infrared objector library file

8.2.1 Hardware connection assignment

In using ZX-08 Infrared Objector with Robo-51 and library file object.h should connect hardware as follows:

Connect Tx point of the left objector with P4.0 of RBX-51AC2 controller board
Connect Rx point of the left objector with P4.1 of RBX-51AC2 controller board
Connect Tx point of the right objector with P3.3 of RBX-51AC2 controller board
Connect Tx point of the right objector with P3.4 of RBX-51AC2 controller board

8.2.2 Detail of object.h library

Listing 8-1 is the source code of object.h library file of C programming with Robo-51. User should add into folder C:\RIDE\INC

The object.h should call some function in delay_robo51.h. Thus, programmer must include the delay_robo51.h with main program.

8.2.3 Function description of object.h library

(1) obj_left: Send the left detection

Function format:

unsigned char obj_left(void)

Return value:

"1" - Object detected
"0" - Not object to detected

(2) obj_right: Send the right detection

Function format:

unsigned char obj_right(void)

Return value:

"1" - Object detected
"0" - Not object to detected

Example 8-1

int left,right; // Declare 2 status variables
left = obj_left(); // Read the left objector value
    // to store in left variable
right = obj_right(); // Read the right objector value
    // to store in right variable
/* —————————————————————————————————————— */
// Program : Detect object
// Description: Module function for detect object 2 channel(fixed port)
// Filename : object.h
// C compiler : RIDE 51 V6.1
/* —————————————————————————————————————— */
sbit ir_l = P4^0; // Define port send infrared signal left channel
sbit irm_l = P4^1; // Define port receive infrared signal left channel
sbit ir_r = P3^3; // Define port drive infrared signal right channel
sbit irm_r = P3^4; // Define port receive infrared signal right channel

unsigned char obj_left(void)
{
    unsigned char count1=0, round1; // counter variable
    irm_l = 1; // Config. input port for receive infrared signal left channel
    ir_l = 1; // Enable infrared signal left channel
    for(round1=0; round1<5; round1++) // Loop for check 5 time
    {
        delay_ms(2); // Delay 2 ms (from delay_robo51.h)
        if(irm_l==0) // receive infrared signal is OK?
            count1++; // increase counter 1 time
    }
    if(count1>=3)
    {
        return(1); // if receive infrared signal is OK return 1
    }
    else
    {
        return(0); // if receive infrared signal is not OK return 0
    }
    ir_l = 0; // Disable infrared signal left channel
}

unsigned char obj_right(void)
{
    unsigned char count2=0, round2; // counter variable
    irm_r = 1; // Config. input port for receiving IR signal right channel
    ir_r = 1; // Enable infrared signal right channel
    for(round2=0; round2<5; round2++) // Loop for check 5 time
    {
        delay_ms(2); // Delay 2ms (from delay.h)
        if(irm_r==0) // receive infrared signal is OK?
            count2++; // increase counter 1 time
    }
    if(count2>=3)
    {
        return(1); // if receive IR signal OK, return 1
    }
    else
    {
        return(0); // if not receive IR signal, return 0
    }
    ir_r = 0; // Disable IR signal right channel
}

Listing 8-1 Sourcecode of object.h library file. It works with ZX-08 Infrared Objector and Robo-51 robot.
Activity 16
Testing Infrared objector

Part list

- 3x10 mm. Screw
- 3 mm. Nut
- 3 mm. Plastic spacer
- Obtuse joiner
- Straight joiner
- ZX-08 Infrared objector

Procedure

A16.1 Make the sensor structure before. Insert 3x10 mm. screw through ZX-08 board, 3 mm. spacer and an Obtuse joiner. Tighten by 3 mm. nut. Connect the other end of Obtuse joiner with Straight joiner following another Obtuse joiner finally. Make this ZX-08 structure 2 sets.

A16.2 Loosen a screw that attach the RBX-51AC2 board with Right angle joiner and insert ZX-08 structure from step A16.1 to space between screw and RBX-51AC2 board. Tighten this screw again for locking the structure. Make same method with another ZX-08 structure.

A16.3 Connect ZX-08 cables to RBX-51AC2 Controller board as follows:

- Connect Tx point of the left object to P4.0
- Connect Rx point of the left object to P4.1
- Connect Tx point of the right object to P3.3
- Connect Tx point of the right object to P3.4
Robotics Experiment with MCS-51 microcontroller

Listing A16-1 The C program for testing Infrared Objector

```c
#include <C51ac2.h> // Header include register of P89C51AC2
#include <delay_robo51.h> // Module function Delay
#include <sound_robo51.h> // Module function Sound
#include <lcd_robo51.h> // Module function LCD on Robo-51
#include <object.h> // Module function IR-object detector

void main(void)
{
    unsigned char left, right; // Declare bit variable for object detection
    // in left and right side
    lcd_init(); // Initialize LCD module
    lcd_putstr(line1, " Obj Left ="); // Show " Obj Left =" message on LCD
    lcd_putstr(line2, " Obj Right ="); // Show " Obj Right =" message on LCD
    while(1) // Looping
    {
        left = obj_left(); // Read result from Objector #1 then
        // store to left variable
        right = obj_right(); // Read result from Objector #2 then
        // store to right variable
        inttolcd(0x8C, left, Dec); // Show the Objector #1 result to LCD
        inttolcd(0xCC, right, Dec); // Show the result from Objector #2 to LCD
        if(left || right) // If any detector active, condition is true.
            beep(); // Beep 1 time if the condition is true.
        delay_ms(100); // Delay 0.1 second
    }
}
```

A16.4 Open RIDE and create the new project. Make C code in Listing A16-1 and save as act16.c file. Build project to HEX file.

A16.5 Turn-on Robo-51 and download hex file.

A16.6 Select the MODE switch on the RBX-51AC2 controller board of Robo-51 to RUN mode.

A16.7 Turn power on. Bring the object to delude at front of Robo-51 robot near both sensors. Observe the operation.

(A) Observe LED of the sensor module ON or not when the object is near sensor. If not ON, must calibration at a small variable resistor on the sensor module. Normal detection range is 6 cm. approximation.

(c) See the operation at LCD module

To beginning, LCD screen show message

```
Obj Left = 0
Obj Right = 0
```

If any sensor detected object, the value message is change from 0 to 1 and make a beep.
Activity 17
Contactless object detection robot

A17.1 Open RIDE and create the new project. Make C code in Listing A17-1 and save as act17.c file. Build project to HEX file.

A17.2 Turn-on Robo-51 and download hex file. Turn power off and unplug the download cable.

A17.3 Select the MODE switch on the RBX-51AC2 controller board of Robo-51 to RUN mode.

```c
/*—————————————————————————————————————*/
// Program : Robo-51 Object detector
// Description: Robo-51 can avoid the object by IR-object detector
// Filename: act17.c
// C compiler : RIDE 51 V6.1
/*————————————————————————————————————*/
#include <C51ac2.h> // Header include register of T89C51AC2
#include <delay_robo51.h> // Delay time module function
#include <remote.h> // ER-4 Remote Control function
#include <sound_robo51.h> // Sound module function
#include <dc_motor.h> // DC motor module function
#include <lcd_robo51.h> // LCD module function
#include <object.h> // Module function detect object
#define pow 200 // Define constant for power drive DC motor

void run_fd(int time_spin)
{
    motor_fd(2,pow); // Motor channel 2 forward
    motor_fd(1,pow); // Motor channel 1 forward
    delay_ms(time_spin); // Delay time for robot drive forward
}

void run_bk(int time_spin)
{
    motor_bk(2,pow); // Motor channel 2 backward
    motor_bk(1,pow); // Motor channel 1 backward
    delay_ms(time_spin); // Delay time for robot drive backward
}

void turn_left(int time_spin)
{
    motor_fd(2,pow); // Motor channel 2 forward
    motor_bk(1,pow); // Motor channel 1 backward
    delay_ms(time_spin); // Delay time for robot spin turn left
}
```

Listing A17-1 The C program for contactless object detection of Robo-51 (continue)
void turn_right(int time_spin)
{
    motor_bk(2,pow);      // Motor channel 2 backward
    motor_fd(1,pow);      // Motor channel 1 forward
    delay_ms(time_spin);  // Delay time for robot spin turn right
}

void main()
{
    lcd_init();            // Initial LCD module
    beep();
    //__________________________________________//
    //Detect Object______________________________//
    while(1)
    {
        if(obj_left()==1 && obj_right()==1) // If both sensors seek object
        {
            lcd_command(0x01); // Clear display
            lcd_putstr(line1,"  Turn Left"); // Show message"  Turn Left"
            run_bk(300);         // Backward 0.3 sec
            turn_right(200);     // Turn right 0.2 sec
        }
        else if(obj_left()==1 && obj_right()==0) // If sensor left seek object and sensor right is blank
        {
            lcd_command(0x01); // Clear display
            lcd_putstr(line1,"  Turn Right"); // Show message"  Turn Right"
            run_bk(300);       // Backward 0.3 sec
            turn_right(200);   // Turn right 0.2 sec
        }
        else if(obj_left()==0 && obj_right()==1) // If sensor left is blank and sensor right seek object
        {
            lcd_command(0x01); // Clear display
            lcd_putstr(line1,"  Turn Left"); // Show message"  Turn Left"
            run_bk(300);        // Backward 0.3 sec
            turn_left(200);     // Turn left 0.2 sec
        }
        else if(obj_left()==0 && obj_right()==0) // If both sensors not detect
        {
            lcd_command(0x01); // Clear display
            lcd_putstr(line1,"  Forward"); // Show message"  Forward"
            run_fd(50);        // forward 50 ms
        }
    }
}
A17.3 Place the robot on the floor. Turn power on. Bring the object to delude at front of Robo-51 robot near both sensors. Observe the operation.

To beginning, ROBO-51 moves forward. If it detects any object, it can move to avoid that object. The criteria in any decision are:

1. If both sensors detect object, Robo-51 moves backward and turn left. LCD shows message **Turn Left**.
2. If the left sensor detected object, Robo-51 moves backward and turn right. LCD shows message **Turn Right**.
3. If the right sensor detected object, Robo-51 moves backward and turn left. LCD shows message **Turn Left**.
4. If both sensors do not detect any object, Robo-51 still moves forward always and LCD screen shows message **Forward**.
COPYRIGHTS

This documentation is copyright 2005-2007 by Innovative Experiment Co., Ltd. (INEX).
By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with INEX products. Any other uses are not permitted and may represent a violation of INEX copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by INEX. Duplication for educational use is permitted, subject to the following Conditions of Duplication:

INEX grants the user a conditional right to download, duplicate, and distribute this text without INEX’s permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with INEX products, and the user may recover from the student only the cost of duplication.

All text and figure is subject to publisher’s approval. We are not responsible for mistakes, misprints, or typographical errors. Innovative Experiment Co., Ltd. (INEX) assumes no responsibility for the availability.