MicroCamp2.0

ATmega8 Activity Kit
manual

INNOVATIVE EXPERIMENT

www.inex.co.th
www.inexglobal.com

Content

Chapter 1
MicroCamp Activity kit hardware

MicroCamp is a set of Microcontroller Activity kit for learning about Micro-controller operation via Robotic activities with C language programming. You will learn about simple operation of microcontroller and how to interface with external components in real word applications.

This activity kit includes Microcontroller board (will call "MicroCamp board"), Swtich module, Infrared Reflector module, DC motor gearboxes and many other mechnical parts for building a programmable robot.

Figure 1-1 shows the layout of MicroCamp main controller board.

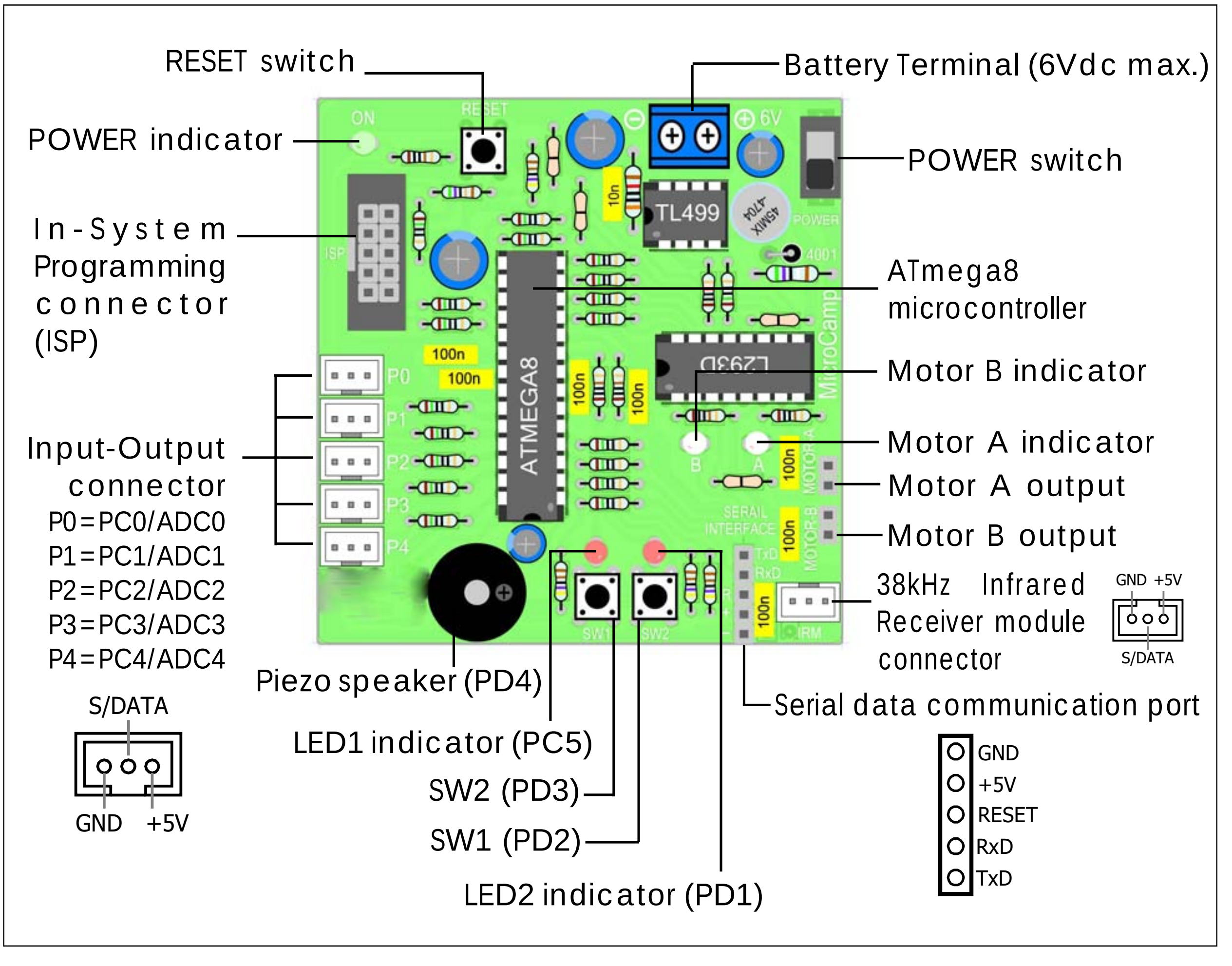


Figure 1-1 MicroCamp board layout

1.1 Hardware of MicroCamp Activity kit

1.1.1 MicroCamp controller board

● The main microcontroller is the 8-bit AVR microcontroller from Atmel; ATmega8. It has many features of modern microcontroller such as the 10-bit Analog to Digial Converter module (ADC), Flash program memory 8KB with 10,000 times erase-write cycles, Data EEPROM 512 bytes and RAM 512 bytes too.

● Main clock frequency 16MHz from Xtal.

● 5-channels Programmable 3-pin Input/Output port. User can programmable all port pins for usages as a Digial Input port, Digital Output port and an Analog input port. The 3-pins are Supply voltage (normally is +5V), Signal or Data and Ground respectively.

● Reserve a port for connecting 38kHz Infrared Receiver module. This port will be assigned to share with Serial Receiving signal (RxD) to external serial data communication device.

● Piezo speaker for sound beeps

● 2 Push-button switches

● RESET switch

● 2 LED indicators, active when logic is “High”

● 2-channels of DC motor drivers. They drive 4.5 to 6V 600mA DC motor with LED indicators

● Supply voltage of +4.8 to +6V from 4 of AA size batteries. Contain in battery holder at the back of controller board.

● On-board switching regulator circuit to maintain the +5V supply voltage when motors function and consume more current.

1.1.2 PX-400 The serial port interface In-System Programmer box

This programmer is used for programming the code into flash memory within the AVR microcontroller. It can work a wide variety of AVR microcontrollers.

Its features are :

● Connection with computer serial port via RS-232. If the computer has only USB port, a USB to Serial port converter can be used. The UCON-232S is highly recommended for this purpose.

● Program the AVR microcontroller via ISP cable. Supports Read, Write, Erase and Data protection functions.

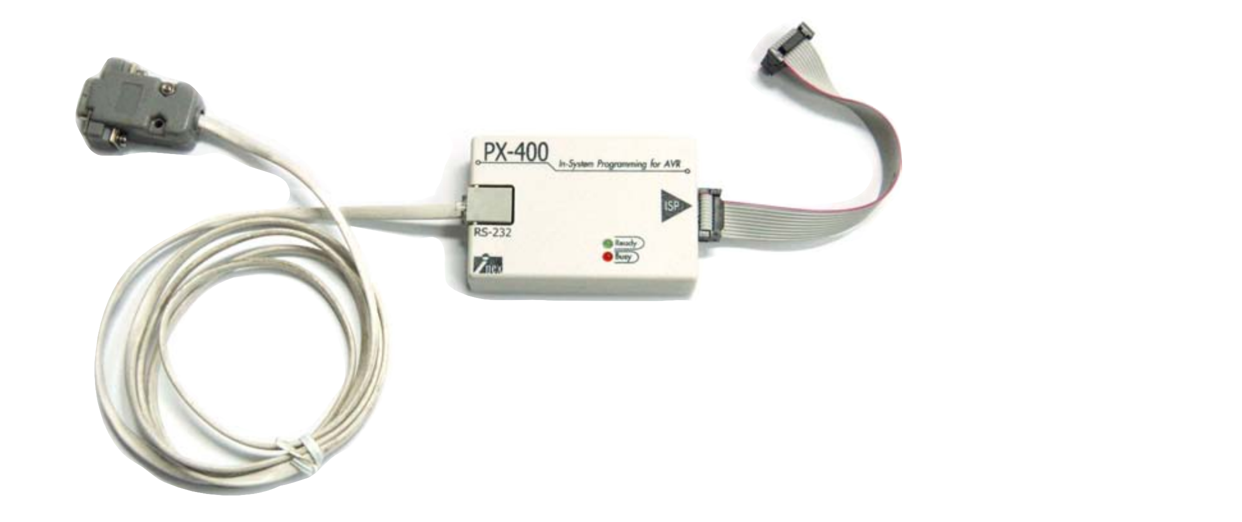


Figure 1-2 Shows PX-400 In-Systrem Programmer box for AVR microcontroller.

- Require +5V supply voltsge from target microcontroller board.
- Operate with AVR Prog software. This software is included in the AVR Studio and can be found in the tools menu and works with the Avr-OspII software as well.

Model Numbers of microcontroller supported in AVR Prog

AT90S1200, AT90S2313, AT90S2323, AT90S2343, AT90S4433 , AT90S8515 , AT90S8535 ,

ATmega128 , ATmega16 , ATmega161 , ATmega162 , ATmega163 , ATmega164P , ATmega165 , ATmega168 , ATmega32 , ATmega64 , ATmega8 , ATmega8515 , ATmega8535 ,

ATtiny12 , ATtiny13 , ATtiny15L , ATtiny2313, ATtiny26

Model Numbers of microcontroller supported in Avr-OSP II

AT90CAN128, AT90CAN32, AT90CAN64,

AT90PWM2, AT90PWM3,

AT90S1200, AT90S2313, AT90S2323, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8515comp, AT90S8535, AT90S8535comp,

ATmega103, ATmega103comp, ATmega128, ATmega1280, ATmega1281,

ATmega16, ATmega161, ATmega161comp, ATmega162, ATmega163, ATmega165, ATmega168, ATmega169,

ATmega2560, ATmega2561,

ATmega32, ATmega323, ATmega325, ATmega3250, ATmega329, ATmega3290,

ATmega406, ATmega48,

ATmega64, ATmega640, ATmega644, ATmega645, ATmega6450, ATmega649, ATmega6490,

ATmega8, ATmega8515, ATmega8535, ATmega88,

ATtiny11, ATtiny12, ATtiny13, ATtiny15,

ATtiny22, ATtiny2313,ATtiny24, ATtiny25, ATtiny26, ATtiny261, ATtiny28,

ATtiny44, ATtiny45, ATtiny461,

ATtiny84, ATtiny85, ATtiny861

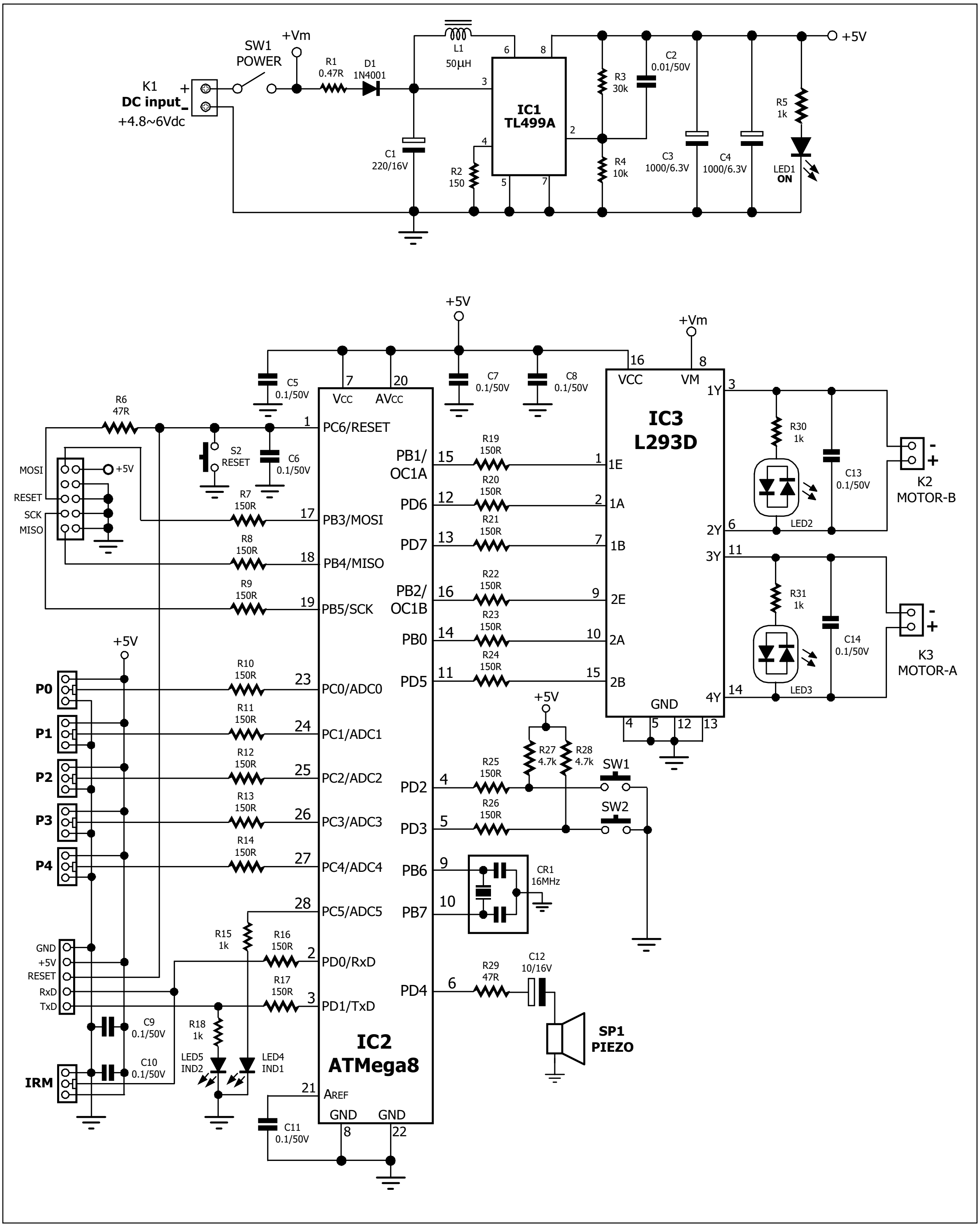


Figure 1-3 MicroCamp controller board schematic

1.2 MicroCamp controller board circuit description

The heart of this controller board is ATmega8 microcontroller. It runs on a 16MHz clock from crytal which is connected at PB6 and PB7 pin.

For PC0 to PC4 port is defined as the new name to P0 to P4. It is labeled on the circuit board for easy reference. All ports can programmable to analog or digital input/output. Analog signal from these port would pass through the Analog to Digital Converter module within ATmega8. The resolution conversion is at 10-bit.

PB3, PB4 and PB5 are In-System Programming port. They are connected to ISP connector for connect with the external ISP programmer box.

PC6/RESET pin is connected with the RESET swtich for resetting to restart the microcontroller operation from user.

PD0/RxD pin is the serial receiver pin. It is shared with IRM connector for 38kHz Infrared Receiver Module and 5-pin of Serial data communication port.

PD1/TxD pin is the serial transmit pin. It is shared to drive the LED5 (IND2 label) and TxD pin of 5-pin of Serial data communication port. For LED4 or IND1 is direct connected to PC5 of ATmega8 microcontroller with current-limit resistor.

The MicroCamp board is equipped with 2 Push-button switches. They are connected to PD2 and PD3 and connected 4.7kΩ resistor pull-up for setting the logic level to “High” in a normal operation and changing to logic “Low” or “0” when switch is pressed.

PD4 pin is connected with a Piezo speaker via coupling capacitor 10μF.

The MicroCamp controller board includes the DC motor driver circuit. It has 2 outputs. The driver IC is L293D H-Bridge driver. One DC motor driver circuit requires 3 signal pins to control :

A and **B** input for applying the signal to select the spin direction of motor.

E control pin is used for enble and stop operation of driver circuit. In addition, the user can control the motor speed with apply PWM signal to this pin. If the width of PWM is wide, it means the high level of voltage sent to motor output.

At the output of L293D, bi-color LED is connected to indicate the voltage pole at the output. Green color indicates forward. Red color indicates backward.

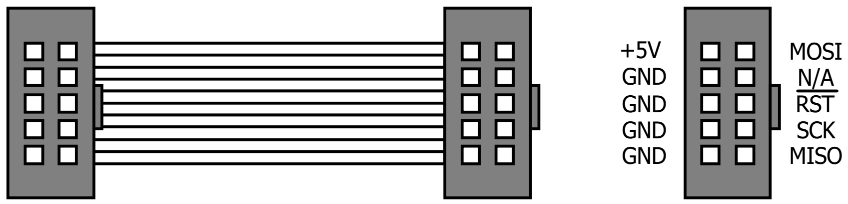
The Power suppy circuit of this board is switching type circuit. TL499A is set to step-up +5V switching regulator for supply voltage to all microcontroller circuit except for the motor driver. With this circuit, it helps microcontoller voltage supply to be more stabilized. Although DC motors require more power during operation but the supply voltage of microcontroller is still fixed at +5V.

1.3 MicroCamp activity kit's cable assignment

The MicroCamp activity kit includes some signal cables for the interfacing between the controller board, sensor module and the computer. They includes the ISP cable for programming the microcontroller, PCB3AA-8 cables for interconnection to the sensor module and a Serial port cable for interfacing with the computer.

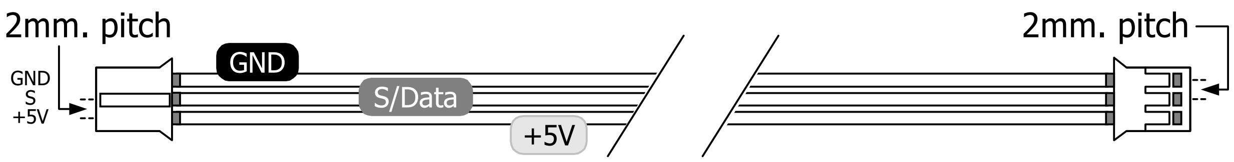
1.3.1 ISP cable

It is 10-wires ribbon cable. Both ends are attached to the female 10-pin IDC header. It is used for interfacing between ISP programmer box and Microcontroller board at ISP connector. This ISP cable's assignment is compatible with Atmel's programming tools standard. The wire assignment can show with the diagram below.



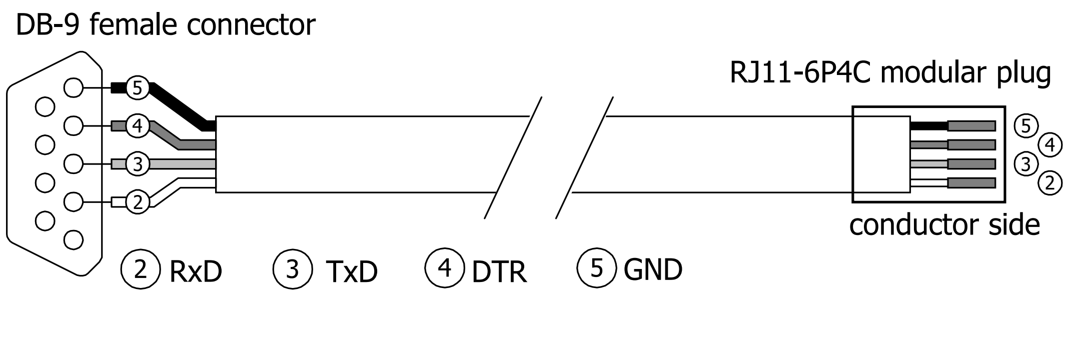
1.3.2 JST3AA-8 cable

This is an INEX standard cable, 3-wires combined with 2mm. The JST connector is at each end. 8 inches (20cm.) in length. Used for connecting between microcontroller board and all the sensor modules in MicroCamp kit. The wire assignment is shown in the diagram below.



1.3.3 CX-4 serial port cable

This is used to connect between the computer's RS-232 serial port and the target or external device such as a Microcontroller board, eg. The MicroCamp controller board. The connector's end uses a DB-9 female connector, and the other end uses a Modular plug RJ-11 6P4C (6-pins form and 4-contacts) Its Length is 1.5 meters. In the kit, this cable is used to connect between RS-232 serial port and PX-400 programmer box. The wire assignment is shown in the diagram below.



1.4 ATmega8 microcontroller Overview

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The ATmega8 which use in MicroCamp board is 28-pin DIP package. The pin assignment shows in the figure 1-4.

1.4.1 ATmega8 features

● It is a low-power 8-bit microcontroller based on the AVR RISC architecture.

● 8K bytes of In-System Programmable Flash with Read-While-Write capabilities 10,000 times erase cycle, 512 bytes of EEPROMwith 100,000 times erase cycle, 1K byte of SRAM and 32 general purpose working registers.

● 23 General I/O lines. manage to 3 groups

1. Port **B** (PB0 to PB7) : Use 2 pin (PB6 and PB7) for connect crystal for clock generator circuit. PB2 to PB5 normally are reserved for In-system porogramming port. Thus PB0 and PB1 free for general purpose application.

2. **Port C** (PC0 to PC6 : 7 pins) PC0 to PC5 are analog input pins. PC6 normally use for RESET pin.

3. **Port D** (PD0 to PD7 : 8 pins) This port can support general purpose application.

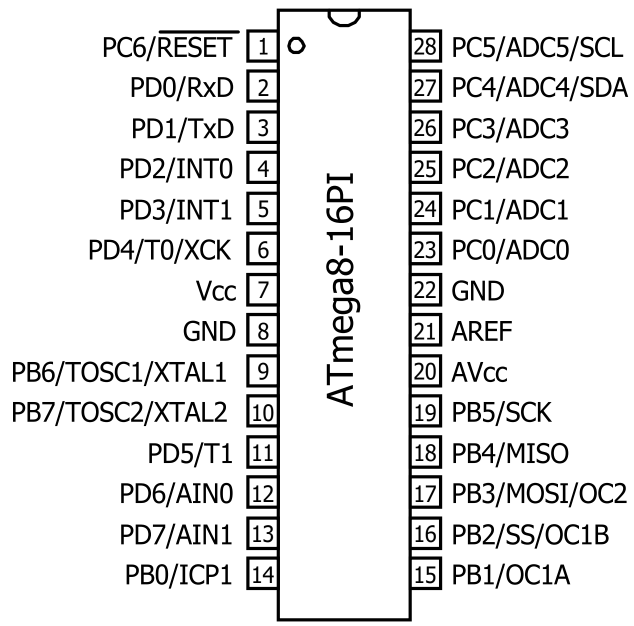


Figure1-4 ATmega8 microcontroller pin assignment

- Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
- 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Three PWM Channels
- 6-channel ADC, 10-bit Accuracy
- Byte-oriented Two-wire Serial Interface
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated RC Oscillator
- External and Internal Interrupt Sources
- 5 Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- Operating Voltage 4.5 - 5.5V
- Speed Grades 0 to 16 MHz

1.4.2 Block diagram of ATmega8

Figure 1-5 shows the ATmega8 microcontroller block diagram. The AVR core combines a risc instruction set with 32 general purpose working registers. The ATmega8 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, a 6-channel ADC with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning.

1.4.3 ATmega8 pin function

Table 1-1 is summary information about ATmega8 pin function.

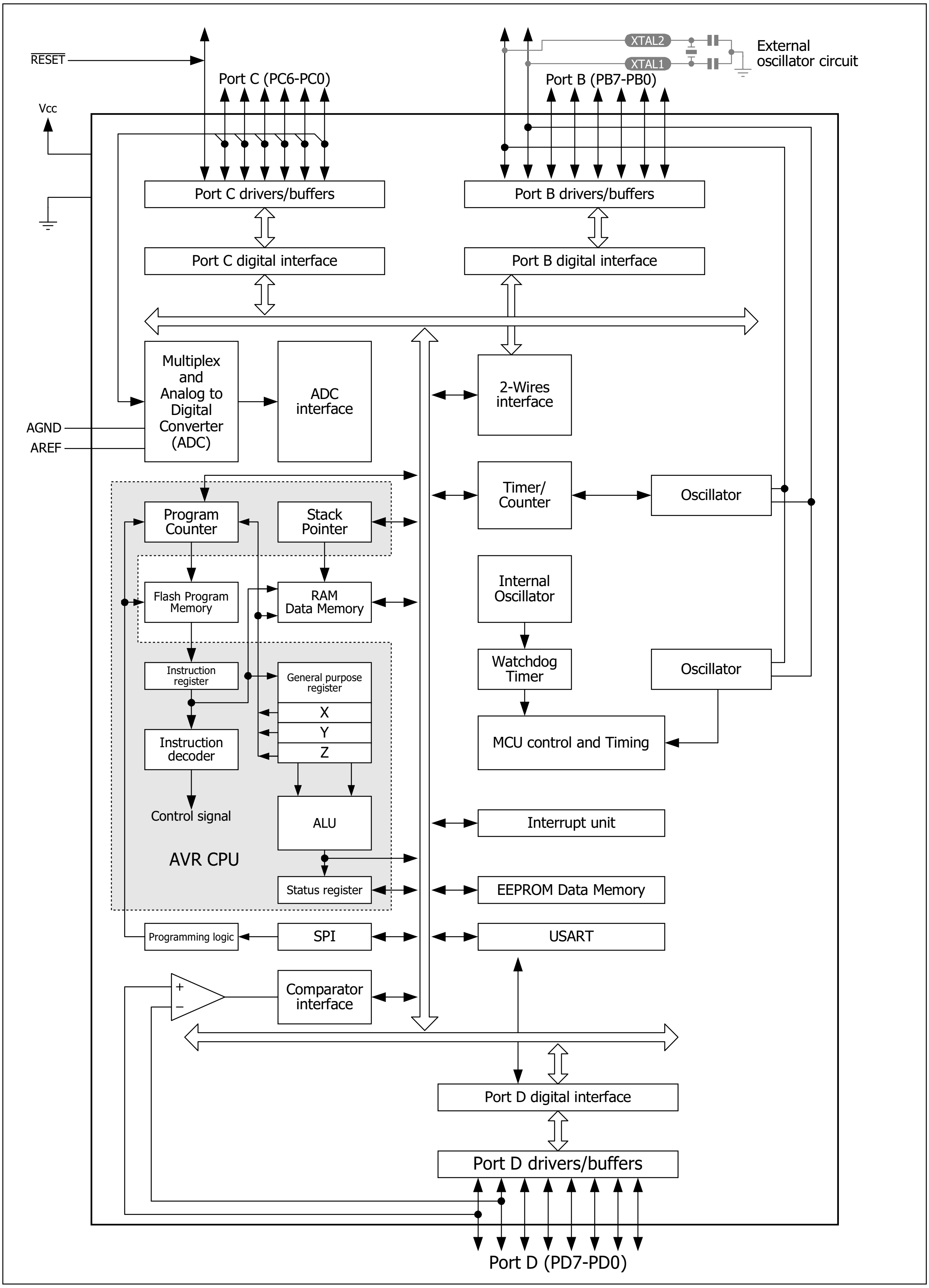


Figure 1-5 ATmega8 Block Diagram

Name	Pin number	Type	Description
Vcc	7	Input	- Supply voltage +4.5 to +5.5V
GND	8,22	Input	- Ground
AVcc	20	Input	- Supply voltage + 5V for ADC module of ATmega8
AREF	21	Input	- Reference voltage input for ADC module of ATmega8
Port B			
Name	Pin number	Type	Description
PB0	14	Input/Output	- PB0 Digital port
ICP1		Input	- Input Capture 1
PB1	15	Input/Output	- PB1 Digital port
OC1A		Output	- Output Compare/PWM 1A
PB2	16	Input/Output	- PB2 Digital port
OC1B		Output	- Output Compare/PWM 1B
SS		Input	- Slave input for SPI and In-System Programming (ISP)
PB3	17	Input/Output	- PB3 Digital port
OC2		Output	- Output Compare/PWM 2
MOSI		Input/Output	- Data input in Slave mode of SPI bus and ISP - Data output in Master mode of SPI busand ISP
PB4	18	Input/Output	- PB4 Digital port
MISO		Input/Output	- Data input in Master mode of SPI bus and ISP - Data output in Slave mode of SPI bus and ISP
PB5	19	Input/Output	- PB5 Digital port
SCK		Input/Output	- Clcok input in Slave mode of SPI bus and ISP - Clcok output in Master mode of SPI bus and ISP
PB6	9	Input/Output	- PB6 Digital port when config CPU operate with internal clock
XTAL1		Input	- External clock input, Connect with Crystal or Ceramic Resonator
TOSC1		Input	- Not use when config CPU operate iwth internal clock
PB7	10	Input/Output	- PB7 Digital port when config CPU operate with internal clock
XTAL2		Input	- Connect with Crystal or Ceramic Resonator
TOSC2		Output	- Clock output when config CPU operate with internal clock

Table 1-1 Pin function summary of ATmega8 microcontroller (continue)

Port C			
Name	Pin number	Type	Description
PC0	23	Input/Output	- PC0 port
ADC0		Input	- Analog input channel 0
PC1	24	Input/Output	- PC1 port
ADC1		Input	- Analog input channel 1
PC2	25	Input/Output	- PC2 port
ADC2		Input	- Analog input channel 2
PC3	26	Input/Output	- PC3 port
ADC3		Input	- Analog input channel 3
PC4	27	Input/Output	- PC4 port
ADC4		Input	- Analog input channel 4
SDA		Input/Output	- Serial data in 2-Wire bus interface
PC5	28	Input/Output	- PC5 port
ADC5		Input	- Analog input channel 5
SCL		Output	- Serial Clcok output in 2-Wire bus interface
PC6	1	Input/Output	- PC6 port
$\overline{\text{RESET}}$		Input	- External reset
Port D			
Name	Pin number	Type	Description
PD0	2	Input/Output	- PD0 Digital port
RxD		Input	-USART receiving input
PD1	3	Input/Output	- PD1 Digital port
TxD		Output	- USART transmit output
PD2	4	Input/Output	- PD2 Digital port
INT0		Input	- External interrupt channel 0
PD3	5	Input/Output	- PD3 Digital port
INT1		Input	- External interrupt channel 1
PD4	6	Input/Output	- PD4 Digital port
XCK		Input/Output	- USART external clock
T0		Input	- Timer 0 External input
PD5	11	Input/Output	- PD5 Digital port
T1		Input	- Timer 1 External input
PD6	12	Input/Output	- PD6 Digital port
AIN0		Input	- Analog comparator input channel 2
PD7	13	Input/Output	- PD7 Digital port
AIN1		Input	- Analog comparator input channel 1

Table 1-1 Pin function summary of ATmega8 microcontroller (finish)

Chapter 2
Development software for MicroCamp Activity kit

Programming development in MicriCamp Activity kit is C language. The software tools that are installed for programming are the following :

1. AVR Studio : This software tool is developed by Atmel Corporation. AVR Studio is a Development Tool for the AVR microcontrollers. AVR Studio enables the user to fully control execution of programs on the AVR In-Circuit Emulator or on the built-in AVR Instruction Set Simulator. AVR Studio supports source level execution of Assembly programs assembled with the Atmel Corporation's AVR Assembler and C programs compiled with WinAVR open-source C Compiler. AVR Studio runs under Microsoft Windows95 and Microsoft Windows NT. Now Windows XP SP2 is recommended. Free download this software at ***www.atmel.com***.

2. WinAVR : WinAVR is a set of tools for the C compiler, these tools include avrgcc (the command line compiler), avr-libc (the compiler library that is essential for avrgcc), avr-as (the assembler), avrdude (the programming interface), avarice (JTAG ICE interface), avr-gdb (the de-bugger), programmers notepad (editor) and a few others. These tools are all compiled for Microsoft Windows and put together with a nice installer program. Free download of the updated version is located at : ***http://sourceforge.net/projects/winavr/***.

For the MicroCamp Activity kit, C programming will be with **WinAVR V20050214**. User will need to install AVR Studio first and WinAVR after which. AVR Studio's mechanism integrates automatically with WINAVR. With this feature, it assist the user in the development of C language and programming on AVR Studio which is much easier and more powerful compared to WinAVR. The compiled file is a HEX file in which case, the user has to download it into the program memory of the AVR microcontroller Board.

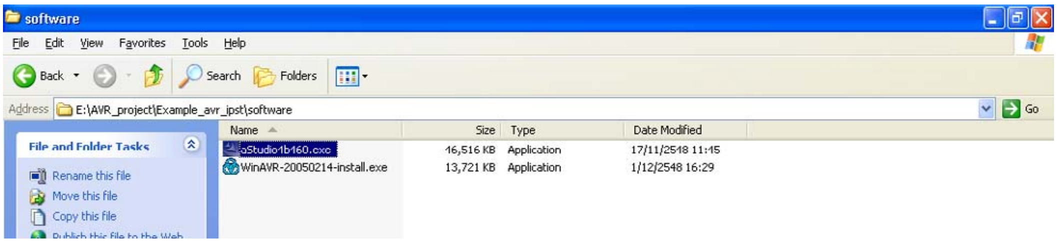
3. Library : These are the support files which allows the user to develop their C language program more comfortably. An example is the Port control library for controlling both Digital and Analog Input/Output, Motor control instructions, etc.

4. Programmer software : This software is used to download the compiled .HEX file to the AVR Microcontroller. Included in this kit is the **AVRProg**. It is Atmel's software and an add-in feature in AVR Studio. AVR Prog software works with the PX-400 Serial port In-system programmer box. The PX-400 programmer is bundled in the MicroCamp Activity kit.

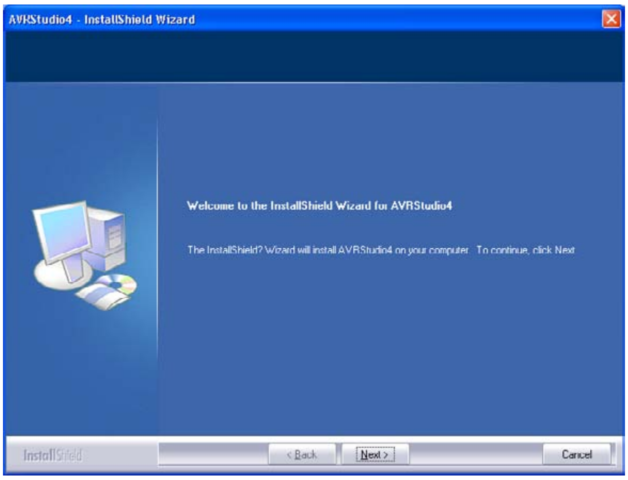
2.1 Installation AVR Studio

Installation of AVR Studio in Windows XP:

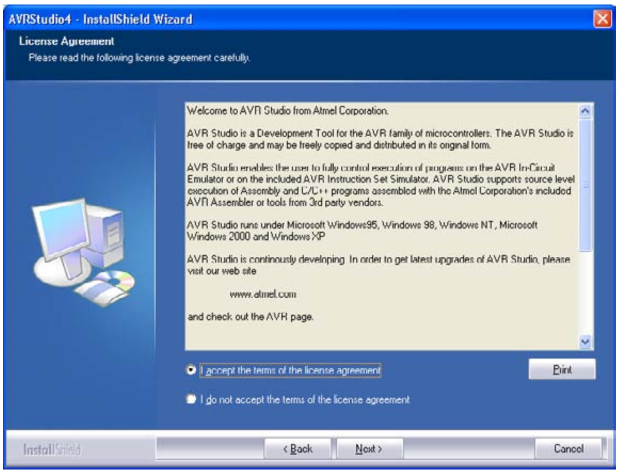
2.1.1 Insert the MicroCamp CD-ROM and look for this file in the AVR Studio directory; ***aStudio4b460.exe***. Double-click this file.



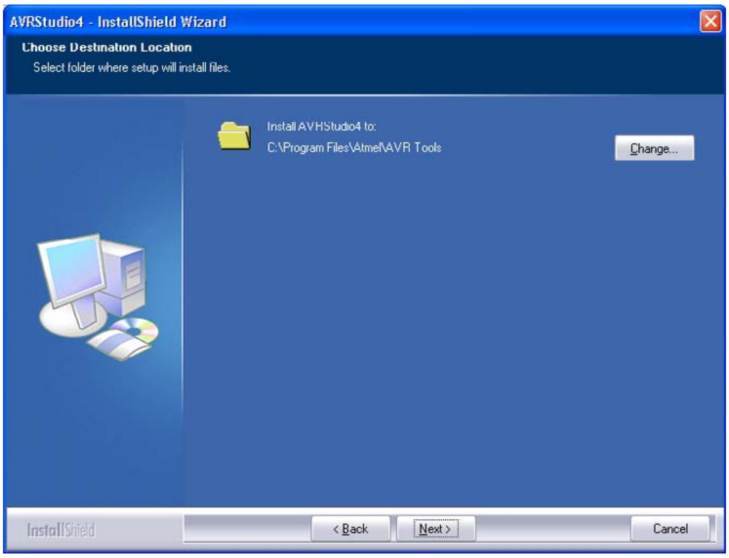
2.1.2 Enter Installation Wizard. Click on the **Next** button to continue.



2.1.3 In the license agreement window, Select the box : **I accept the terms of the license agreement** and Click on the **Next** button.

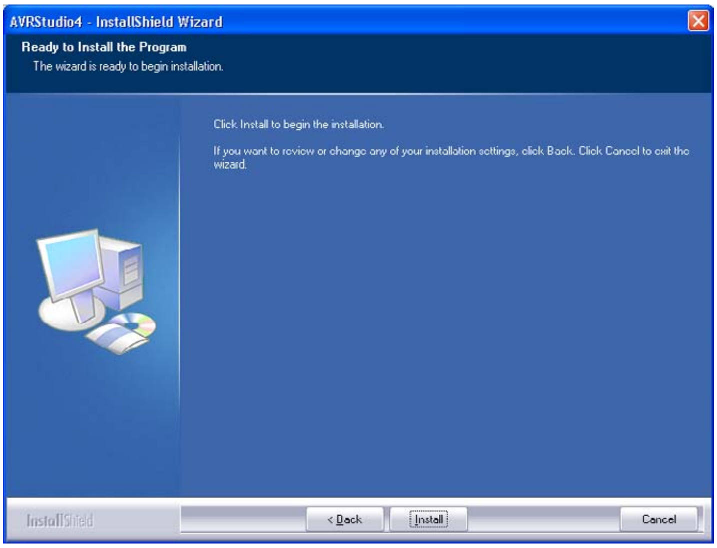


2.1.4 **Choose Destination Location** wondows will appear. You can change the path by clicking on the **Change** button and setting the new path. After this, click on the **Next** button.



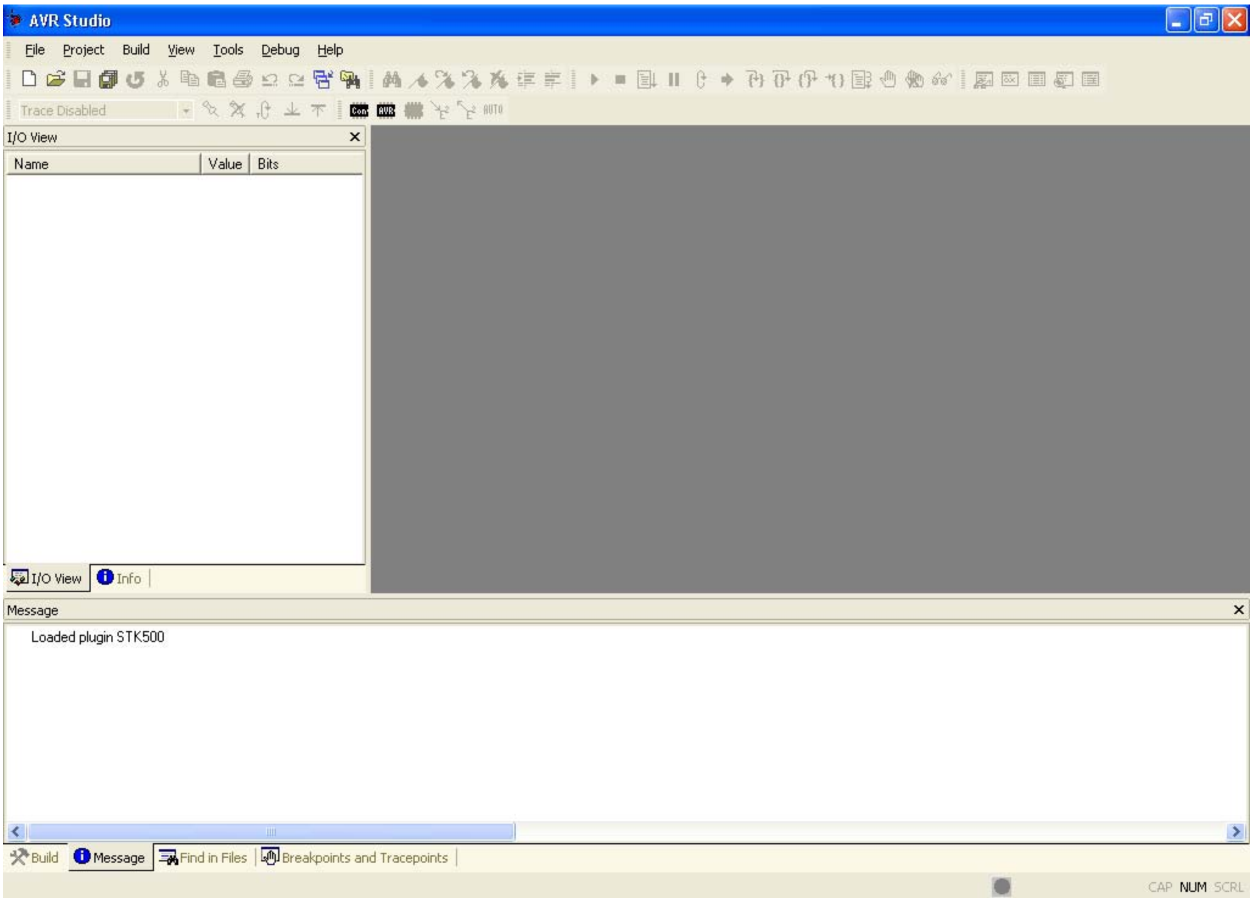
2.1.5 The Driver USB Upgrade window will now appear. Click on the **Next** button to pass this step.

2.1.6 In the begin installation window, click on the **Install** button to start installation.



2.1.7 After installation is complete, click on the **Finish** button to end the installation of AVR Studio.

2.1.8 To launch the AVR Studio program. Click on Start → Programs → Atmel AVR Tools → AVR Studio 4. The main window of the AVR Studio program will appear.

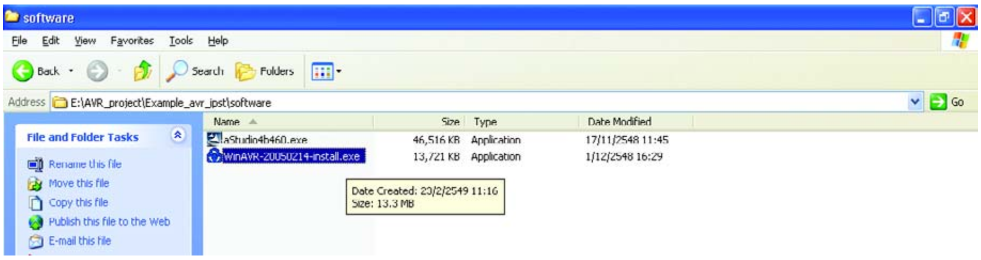


2.2 Installaltion of WinAVR

Please note that installation of WinAVR is done after the installation of AVR Studio. Please ensure this is being done before proceeding.

Installation of WinAVR in Windows XP :

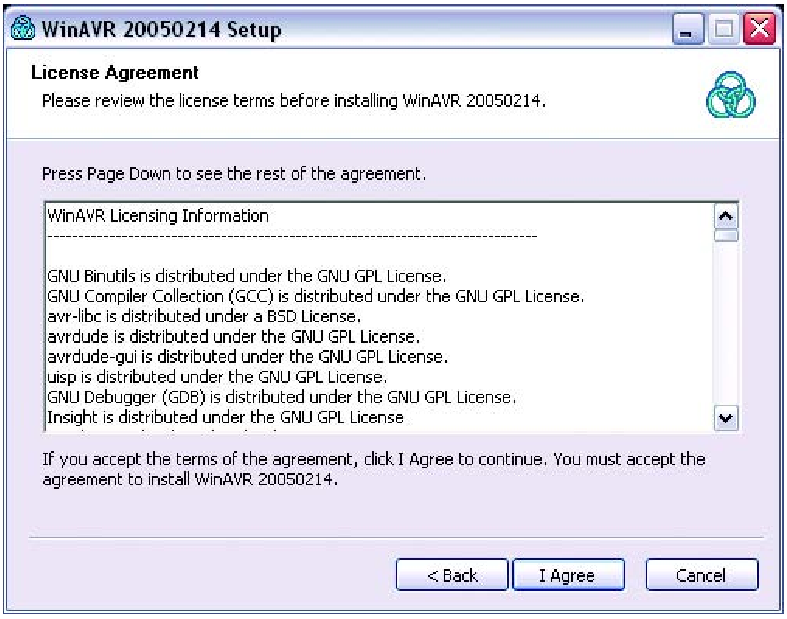
2.2.1 Insert the MicroCamp CD-ROM, and find the installation file of WinAVR; **WinAVR-20050214-install.exe**. Double-click this file.



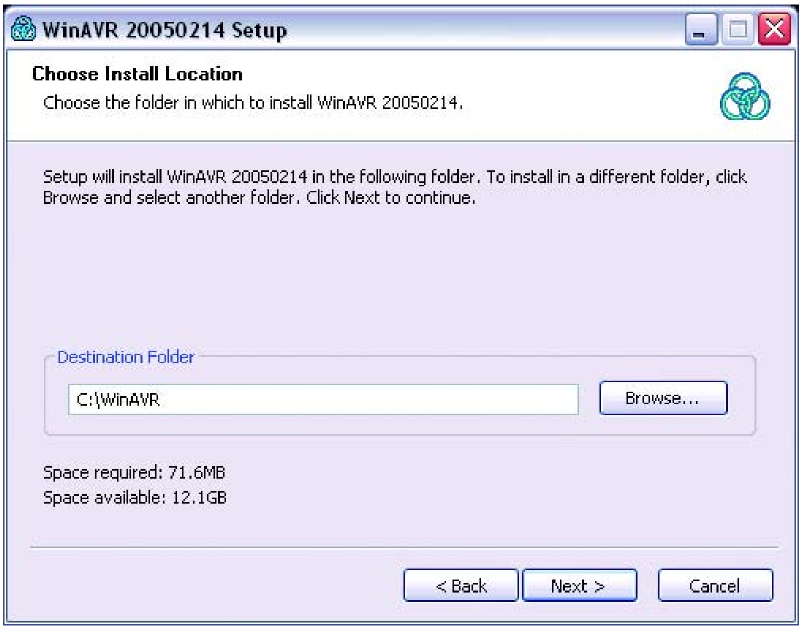
2.2.2 **Installation language** dialog box will appear for selection the language of this installation. Sleect your preferred language from the sliding bar. After that click on the **OK** button.

2.2.3 The Welcome installation software window appears and show the instalaltion information. Click on the **Next** button.

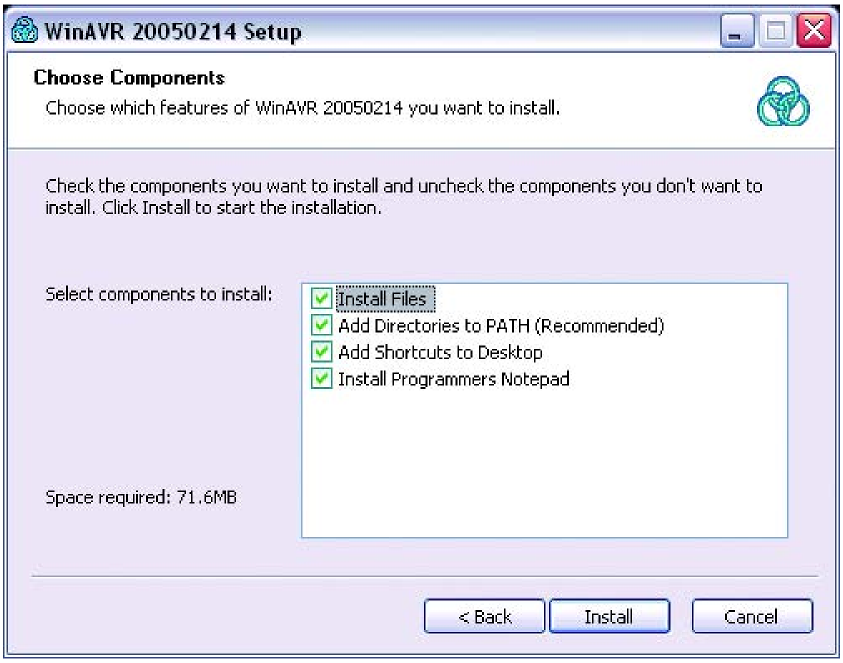
2.2.4 In the License agreement window, Click on the **I agree** button.



2.2.5 **Choose Install Location** window appears. User can change the path and the folder for installation of WinAVR by clicking at the **Browse** button and selecting the respective folder. The proposed folder is C:\WinAVR. After selection, click on the **Next** button to continue to the next step.



2.2.6 In the **Choose Components** window. select the components which you want to install or follow according to the below diagram. Click on the **Install** button to begin installation.



2.2.7 The installation process starts and reports the status back on the screen. The User needs to wait until the installation is complete. Click on the **Finish** button to end once its done.

2.3 Copying Library

You will need to copy the library file (.H file) from the ***MicroCamp_include*** folder in the Cd-ROM. It is better to copy these files to a folder where you save your programming codes.

During the program development of MicroCamp with AVR Studio and WinAVR, you will need to define or set the path of all the tools to integrate with the ***Micro Camp_include*** folder. Ensure that the path of the ***MicroCamp_include*** folder is correct. This is very important as if the path details are not clear or missing, the whole compilation process will have errors.

Chapter 3
C programming development for MicroCamp kit with AVR Studio and WinAVR compiler

3.1 The heart is the C compiler

In actual fact, writing of the C program for the microcontroller is not the actual code that is sent to the microcontroller's program memory. The real data is in the machine code which is being compiled from the written C code and compiled with the C Compiler software.

The steps in C programming development are as follows:

(1) Write the C programs with the text editor / Project IDE that is provided.

(2) Compile the C code into assembly Code for the microcontroller

(3) The Assembly Code will be converted into Machine Code into HEX file format.

(4) Download this code into the program memory of the microcontroller

(5) Run the microcontroller. Go back to step 1 if you have errors.

Steps (2) and (3) will not be shown as the C Compiler will do all of these in its background.

After installing AVR Studio and WINAVR software, the library files are required to be copied in order to support the MicroCamp kit. The MicroCamp Library files are contained in the ***MicroCamp_include*** folder in the CDROM that is included in this kit.

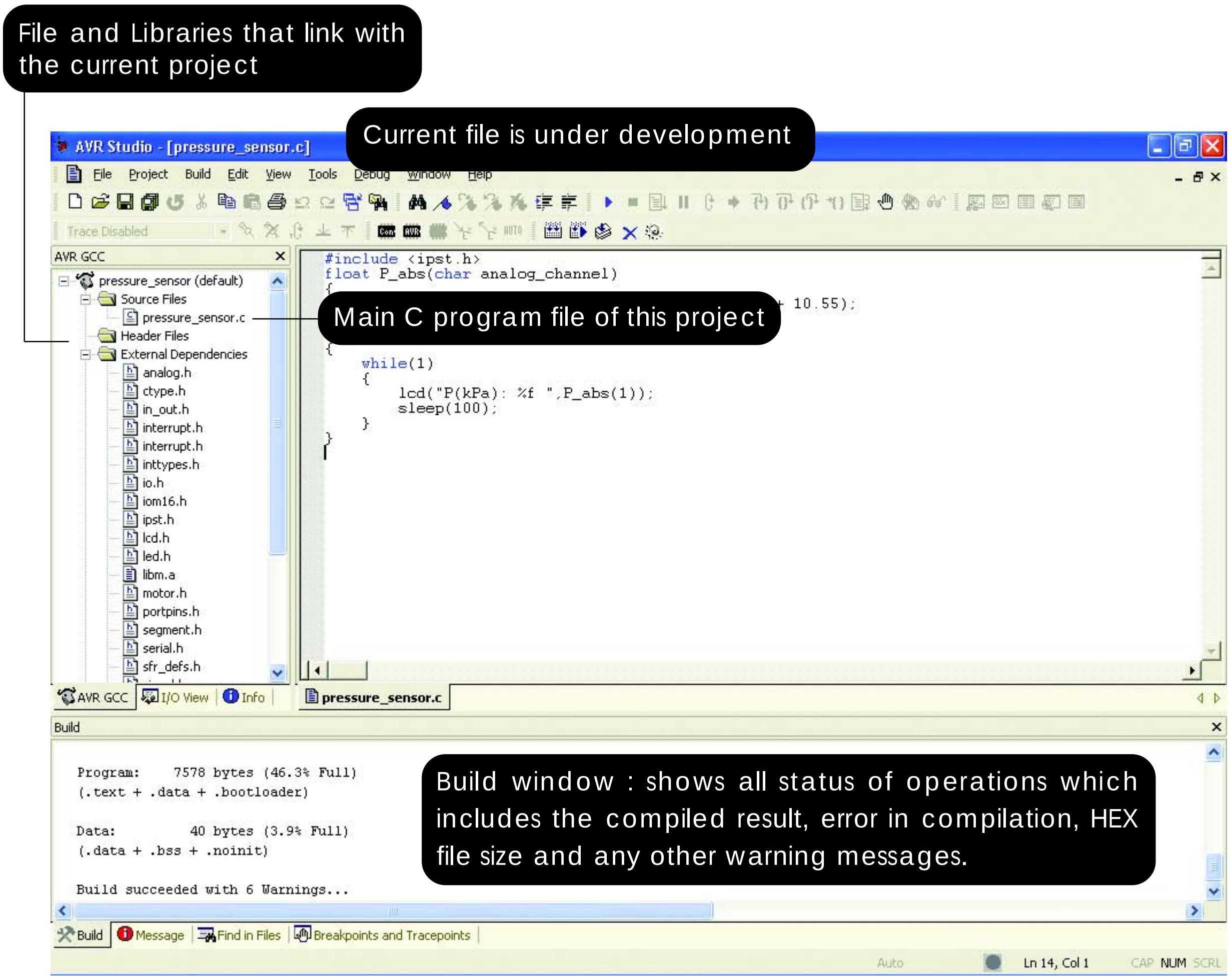
In the C programming development platform in AVR Studio, developers need to compile it into project file format. After the codes are being compiled into HEX file using the same name as the project filename, the file is needs to be downloaded into the ATMEGA8 Microcontroller.

For example :

Name the project file to ***test_segment***. After compiled, the result file is ***test_segment.hex***

3.2 The AVR Studio V4.0 windows details

The figure below shows the main components in the main window of the AVR Studio software.



3.2.1 File menu

Includes the command as follows :

New File	Create empty text file
Open File	Open a file in text editor or an object file for debugging
Close	Close the active text file
Save	Save current text file
Save As...	Save current text file under given name
Save All	Save all files and project settings
Print	Print active text file
Print Preview	Preview active text file
Print Setup	Setup printer
Exit	Exit AVR Studio, project are saved when exiting.

3.3.2 Project menu

Includes the command as follows :

Project Wizard	Open the project wizard. You must close the current project first.
New Project	Open the new project dialog. You must close the current project first.
Open Project	Open a new project, either an APS project file or an object file.
Save Project	Save the current project with all settings
Close Project	Close the current project
Recent Projects	Show a list of recent project, select one to open
Configuration Options	This option is only available when the project is a code writing project. E.g. an assembler or AVR GCC project. This command open the configuration dialog for the current project.

3.2.3 Build menu

Includes the command as follows :

Build	Build the current project
Rebuild All	Rebuild all the modules in the project
Build and run	Build, and if error free , start debugging session
Compile	Compile the current source file
Clean	Clean the current project
Export Makefile	Save the current settings in a new make file

3.2.4 Edit menu

Includes the command as follows :

Undo	Undo last editor action
Redo	Redo any undo action
Cut	Cut and copy selected text from editor
Copy	Copy selected text from editor
Paste	Paste any text from clipboard to the editor
Toggle Bookmark	Toggle bookmark on/off at the selected line in the editor
Remove Bookmarks	Remove all bookmarks
Find	Open a find dialog to search through the current source file.
Find in Files	Open a find in files dialog to search through all project files.
Next Error	Locate and jump to the next build error if any
Show whitespace	Toggle on/off whitespace markings
Font and color	Open a font dialog to view/edit font settings in the source editor

3.2.5 View menu

This menu includes the command as follows :

Toolbars	Sub menu toggles toolbars on/off, access to customize-dialog. Described here
Status Bar	Toggle status bar on/of (status bar is the line in the bottom of the screen)
Disassembler	Toggle on/off the disassembly window
Watch	Toggle on/off the watch view
Memory	Toggle on/off the memory view
Memory 2	Toggle on/off the memory view 2
Memory 3	Toggle on/off the memory view 3
Register	Toggle on/off the register view

3.2.6 Tools menu

This is the hardware interfacing command menu. AVR Studio can interface many hardware for development. For the MicroCamp kit, developers must select the AVRprog. This is the operating software for the PX-400 Serial Port In-System Programmer box.

Developers must connect the PX-400 box to their COM port before open the AVRprog software.

3.2.7 Debug menu

This menu have many commands that relates to the program simulation and debugging. The MicroCamp kit does not require much usage of this feature.

3.3 Building C project file in AVR Studio

3.3.1 Open the AVR Studio. If there is any project running, developers can close by select the menu **Project → Close Project**

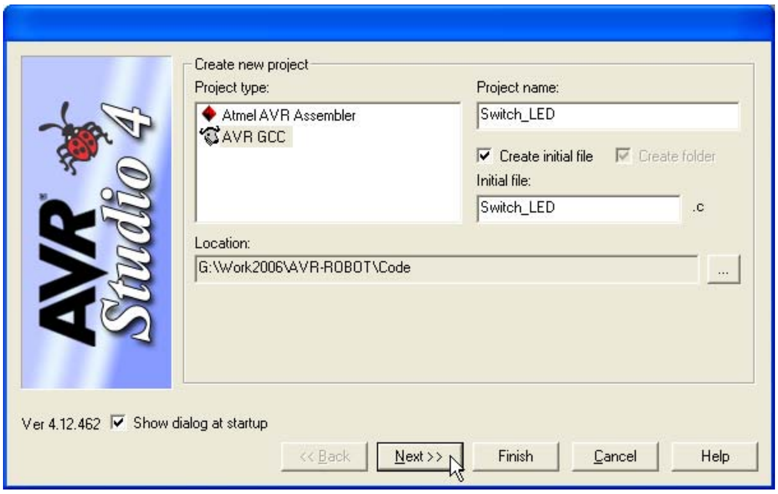
3.3.2 To create the new project. Select the command at menu **Project → New Project.**

3.3.3 The properties project window will appear. Set the parameter as follows :

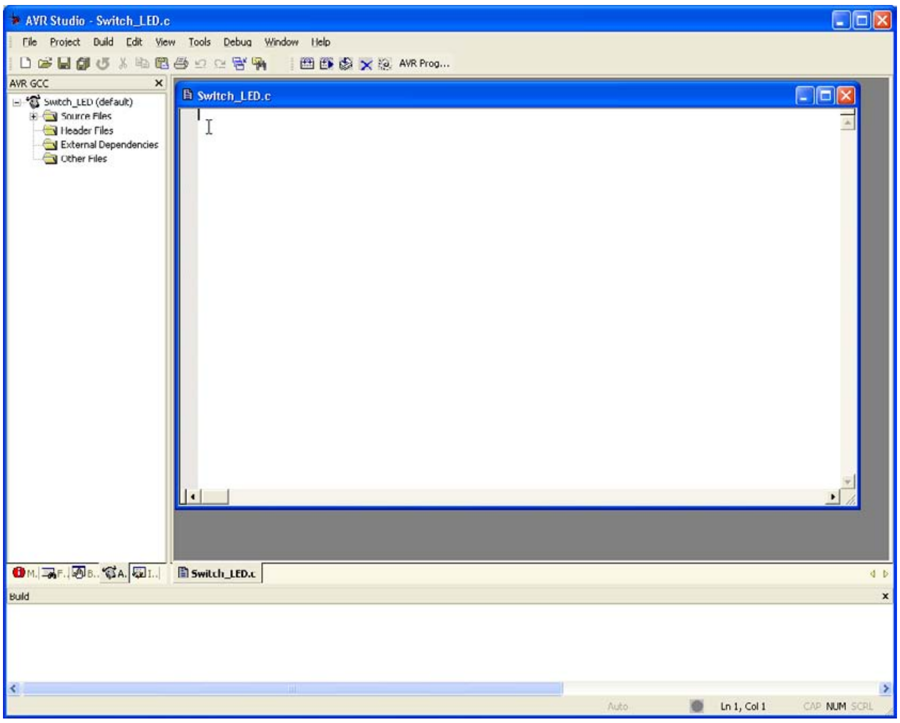
3.3.3.1 Click on this to select AVR GCC item within **Project type**: for select type of project file to program in C.

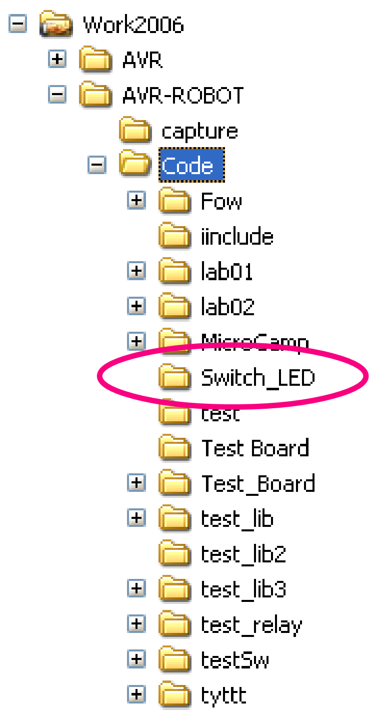
3.3.3.2 Set the project name as ***Switch_LED*** (an example name). This will cause the initial file section to be created. This project has a main C program file called, ***Switch_LED.c***.

3.3.3.3 Select the project's path in Location: Example is ***G:\Work2006\AVR-ROBOT\Code***. After this, click on the **Finish** button.



3.3.4 The ***Switch_LED*** project environment will be created as shown in the diagram below.

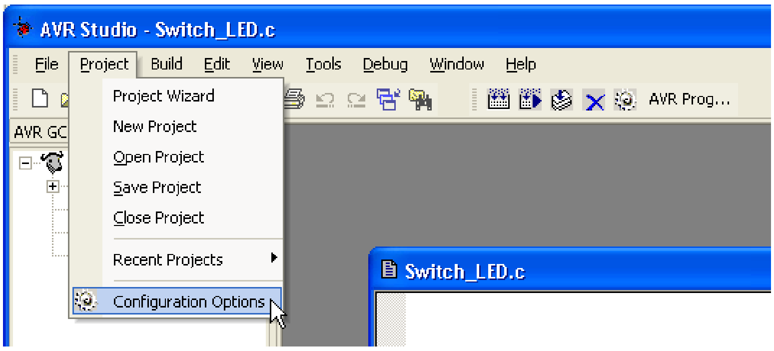




The folder ***Switch_LED*** will be created in ***G:\Work2006\AVR-ROBOT\Code***. In the same folder the file ***Switch_LED.aps*** and main C program file ***Switch_LED.c*** will be created.

3.3.5 Next step is to determine the microcontroller information and path of all the library file which is being used in this project.

3.3.5.1 Select the command at **Project → Configuration Options**

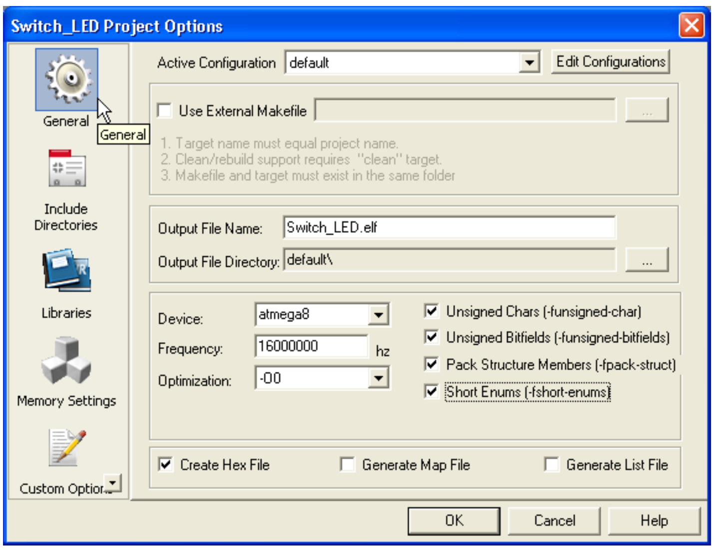


After that the window **Switch_LED Project Options** will appear for setting the properties. See the left of this window. Developers will found 5 icons as :

- General
- Include Directories
- Libraries
- Memory Settings
- Custom Options

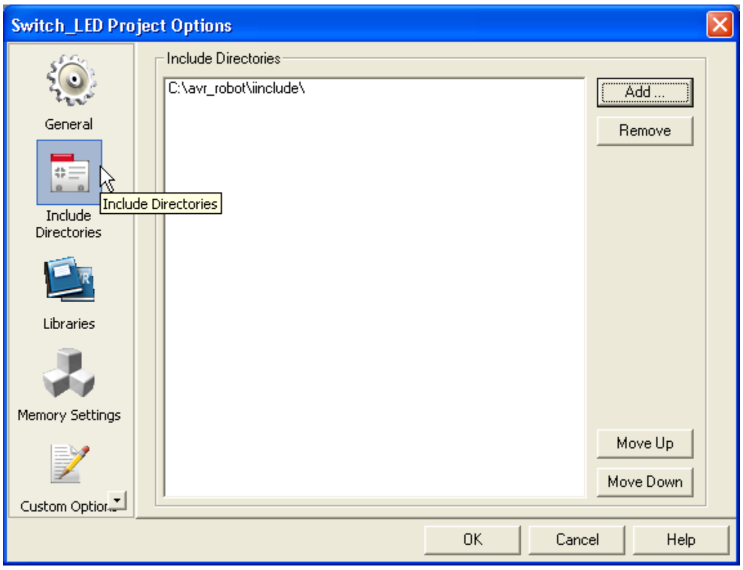
3.3.5.2 At General icon, determine all data follows

- Device : **atmega8**
- Frequency: **16000000 Hz**

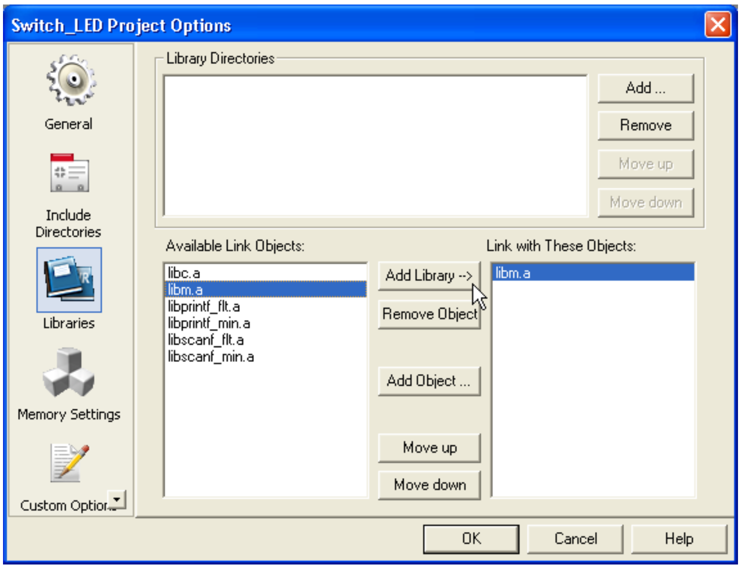


3.3.5.3 Click on this icon and the Include Directories for determining the path of library file. Find and select the library file and click on the **Add** button. For example is ***C:\AVR_ROBOT\include***. After determining the path, you will found the list for selection.

3.3.5.4 Select the icon **Libraries** to links to all the libraries with the main file.



3.3.5.5 At the boxs **, Available Link Objects:**, click to select the item ***libm.a*** and click the **Right Arrow** button to copy the item ***libm.a*** which appears at the Link with These Objects window. Click the **OK** button to finish.



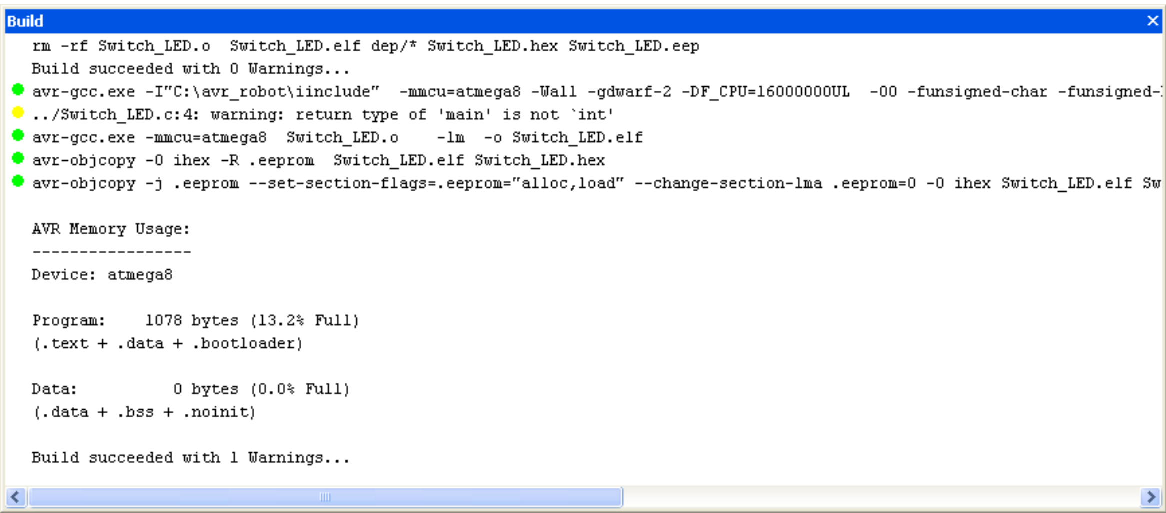
3.3.6 Next, write the C code in the ***Switch_LED.c*** file. TThis file controls the microcontroller to On and off the LED when the switch is pressed. The details & codes of this file is shown in the Listing 3-1.

```
#include <in_out.h>
#include <sleep.h>
void main()
{
  while(1)
  {
      if (in_d(2)==0)
      {
         toggle_c(5);
      }
      if (in_d(3)==0)
      {
         toggle_d(1);
      }
      sleep(200);
  }
}
```

Listing 3-1 C code of Switch_LED.c

3.3.7 Compile the target file to ***Switch_LED.hex*** by selecting the command at the menu Build → Build or press F7 button or click at button.

The status of this operation will be shown at the Build or Output window at the bottom of the main window of the AVR Studio as shown in the diagram.



If any error occurs, such as an illegal command or a link error, the Build Output window will appear. Developers need to edit the program,pfix all errors and re-comple the code until it is correct and the HEX file is being compiled properly.

After compilation, the file ***Switch_LED.hex*** will be made and stored in the folder of that project file. *For example* : The result file Switch_LED.hex is stored at the folder ***Switch_LED.hex*** is stored at ***G:\Work2006\AVRROBOT\Code\Switch_LED\default***.

3.4 How to develop the previously project file

Developers can open the previously project file for editing or improvement. Enter to menu **Project → Open Project** and select the path that store the target project file. The project file is saved as .aps file

Example **:** If would like to open the ***Switch_LED*** project file, select to **Project → Open Project** and access to the path or folder which contains the ***Switch_LED.aps*** file. Open this file for editing. Developers can save with the same name or different.

3.5 Downloading and Testing the program

The next step after compiling the project file is to download the HEX file to MicroCamp controlelr board. In this example the result file is saved as ***Switch_LED.hex***. The step of downloading and testing are as follows :

3.5.1 Turn on the POWER switch. The green LED at ON labeled is on.

3.5.2 Connect the download cable (ISP cable) from the PX-400 programmer box to the In-System Prog. (ISP) connector on MicroCamp controller board.

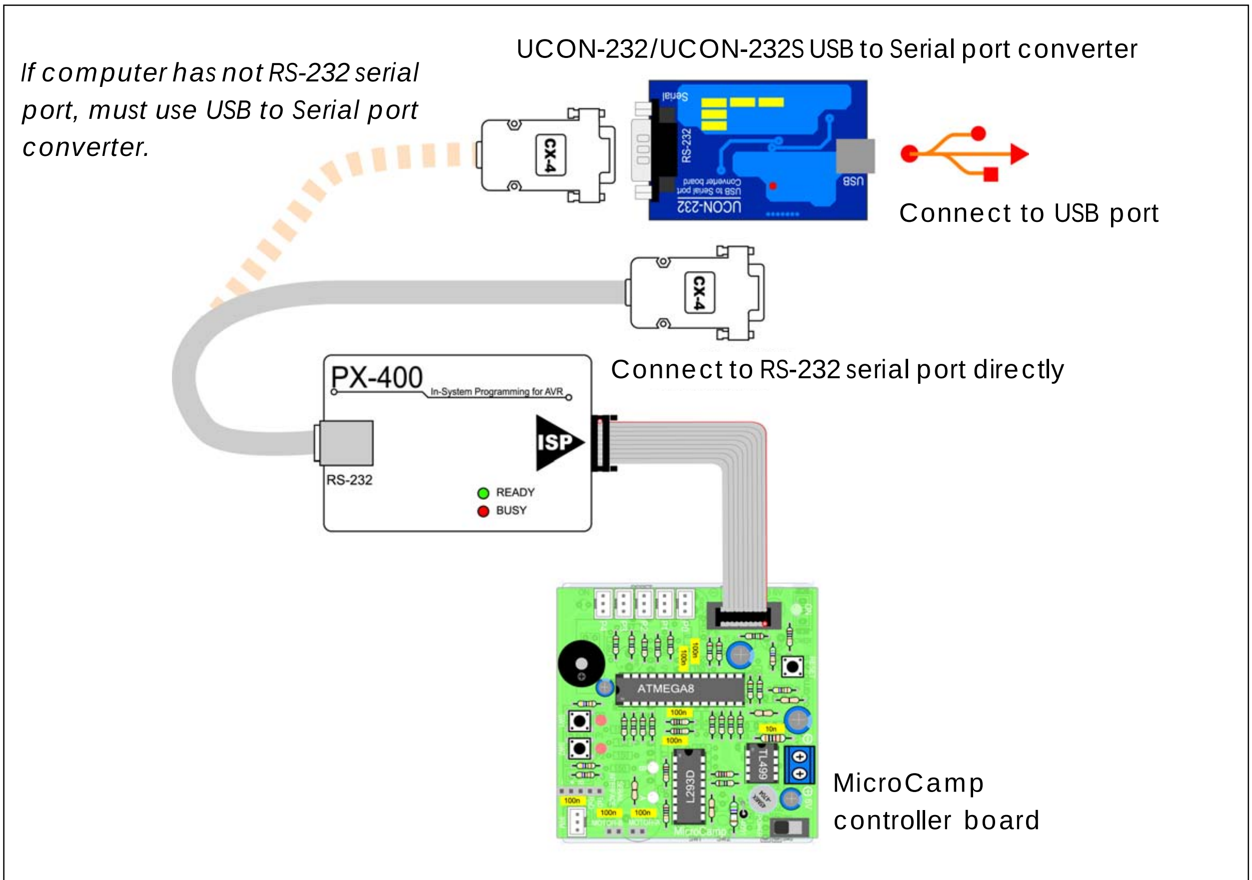
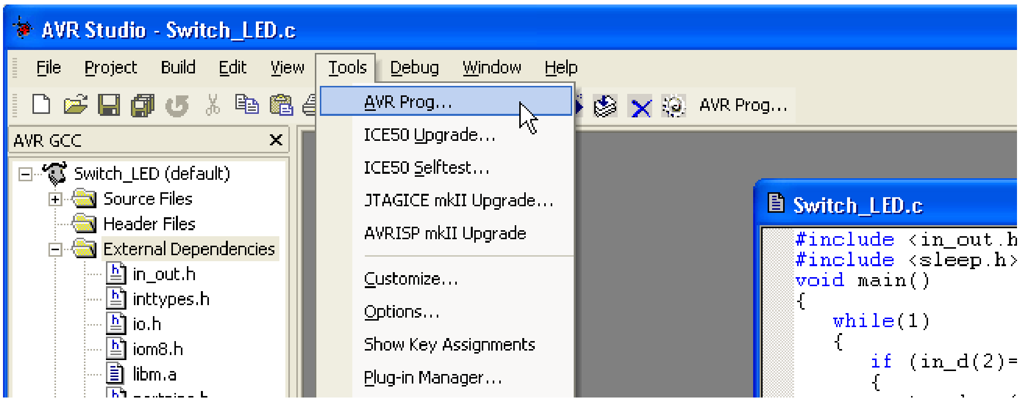


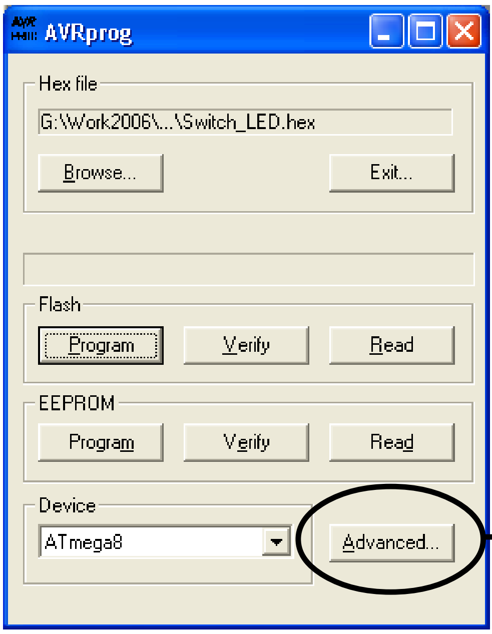
Figure 3-1 Connection diagram of PX-400 programmer box and MicroCamp controller board for downloading the program

3.5.3 Switch to AVR Studio program, select the command at menu **Tool → AVR Prog...**



3.5.4 The AVRprog window will not appear.

3.5.5 At the AVRprog window, click on the Browse button to find the path of ***Switch_LED.hex*** file for selection the HEX file require to download.



For novice users, it is adviced **NOT** to enter the advance button of this window as there are many advanced configuration which requires more experience for adjusting and changing.

If any setting are incorrect, ATMEGA8 will not be successfully programmed via ISP.

3.5.6 Click at the Program button in the Flash command. The ***Switch_LED.hex*** file will now be downloaded into the ATmega8 microcontroller in the MicorCamp controller board.

3.5.7 When the download is finished, the program will run automatic. Press the button swtich SW1 and SW2 on MicroCamp controller board. Observe the LED operation.

The LED will turn on and off when the switch is pressed and blink if the switch is released..

Chapter 4
Library and Function of C programming

In C, a function is equivalent to a subroutine , or a procedure. A function provides a convenient way to encapsulate some computation, which can then be used without worrying about its implementation. With properly designed functions, it is possible to ignore how a job is done; knowing what is done is sufficient. C makes the sue of functions easy, convinient and efficient; you will often see a short function defined and called only once, just because it clarifies some piece of code.

All C programs must have a '**main**' function that contains the code which will be run first when the program executes. Other sub C programs functions can be linked to this main function. Therefore function capability is a vital component in C programming.

4.1 Function declaration

It has general format :

```
return_type   function_name(parameter1, parameter2, ...)
{
   command_list 1;
   ....................
   ....................
   command_list n;
}
```

thus ;

function_name is the name of function

return_type is the type of the data resulting from each function. Within this function, the command `return(value)` is used for sending the result data. The target variable that the return value will be applied on must be the same as each other to avoid any variable mismatch. Any function without a return value, `void` parameter at the `return_type` must be.

parameter is a part of data or variable that relate with function. Some functions require many parameters, while some functions have none. If no parameters are required, a void can be declared. Some function need many parameter but some functrion not. In function that have not any parameter, can ignore or declare to void.

command_list 1...command_list n is a command within this function. At the end of each command, a semi-colon symbol is required to close and separate the commands.

4.2 How to using the function

All functions in the C program that are declared can be called in the "main" function and other functions as well. In the process of calling a function, developers are required to specify the name of function and put the suitable parameters or data which the function requires. The data which is passed to all parameters in each function is called an "Argument"

The calling function has this form :

```
function_name(agument1, agument2,...)
```

Thus;

`function_name` is the specific name of the function which was declared.

`agument` is the data which is passed from the function parameters. If the function has no parameters, no arguments are required.

Example 4-1

```
void tone(void)
{
      sound(3000,100);        // Sound generator function;
                              // generate 3kHz signal in 0.1 second
      sleep(1000);            // Delay 1 second
      sound(3000,100);        // Sound generator function;
                              // generate 3kHz signal in 0.1 second
}
```

from the code above, it is a declaration of a **tone** function. This function does not return the result and has no parameters. This function operation is to generate the sound signal of 3kHz for 0.1 second and repeat itself again after 1 second.

Developers can use this function inside a main function as follows :

```
void main()
{
      ................        // Any instruction
      tone();                 // Call tone function
      ...............         // Any instruction
}
```

Note :** This function requires 2 libraries to be included in the C program ; **sound.h** and **sleep.h

Example 4-2

```
void tone(unsigned int delay)
{
      sound(3000,100);        // Sound generator function;
      sleep(delay);           // Delay from parameter
      sound(3000,100);        // Sound generator function;
}
```

This example is different from the previous example at the Sleep function. The function needs the parameter "delay" which is being declared in the tone for setting the time delay in milliseconds.

4.3 Library

A Library is a file which includes one or many functions that operates similarly. Normally, the name of library file will according with the function for easy remembrance and referencing.

To use libraries, programmers need to declare the prototype of the library at the head of the main C program. The Correct path which contains the library file must be set when creating the AVR Studio Project File.

4.3.1 How to make library

The library file is similar to a C program but without any Main program or main functions. After write the codes , it must be saved as a .h file For example, create the library file ; ***led1.h***.

The steps for creating this file are as follows:

(1) Create the new file from **File → New File** to open the new editor window.

(2) Type in the code of the Blink function as follows :

```
void sleep(unsigned int ms)
{
      unsigned int i,j;
      for(i=0;i<ms;i++)
            for(j=0;j<795;j++);
}
void blink(unsigned int cnt)
{
      unsigned int _cnt=0;
      DDRC |= _BV(5);       // Set PC5 ==> Output
      while(_cnt < (cnt*2)) // Test Counter
            {
                  PORTC ^= _BV(5);      // Toggle PC5 bit
                  sleep(300);           // Delay 0.3 Second
                  _cnt++;
            }
}
```

(3) Save the file by selecting **File → Save As**... You need to save it as a .h file. Now the library file ***led1.h*** is created.

4.3.2 How to use library

After creating the library file, developers can call all functions inside the library files by including them into the head of the C program.

```
#include    <library_filename>
```

or

```
#include    "library_filename"
```

Directive `#include` helps the C program to recognize all functions inside the library file.

Example 4-6

(1) Create the new project in name ; ***test_lib***

(2) Type the C code below in the ***test_lib.c*** window

```
#include <in_out.h>         // Standard Library
#include <led1.h>           // get blink function from here
void main()
{
     blink(10);             // Blink LED 10 times
}
```

Description :

*The **test_lib** program will use 2 library files. One is the standard input/output port library of the ATmega8 microcontroller (**in_out.h**). Anothe one is the **led1.h** file that is created on your own. Inside the **led1** library has 2 functions ;* `blink` *and* `sleep()`. `blink` *function determines the PC5 port to output for driving LED and sending logic "1" and "0".* `sleep()` *function determines the delay time for the LED operation. The* `blink` *function works until it reaches the value that is being declared by the programmer.*

(3) Set the path for ***led1.h*** library from **Project → Configuration Options**. Select the icon **Include Directories**. After that, set the path for ***led1.h*** file.

(4) Build this project. The result file ***test_lib.hex*** will be created.

(5) Download the ***test_lib.he***x into the microcontroller.

(6) Observe the operation of the program in the MicroCamp controller board.

LED at PC5 pin of ATmega8 will blink 10 times.

4.4 Data type in C programming of WinAVR

WinAVR is a suite of executable, open source software development tools for the ATMEL AVR series of RISC microprocessors hosted on the Windows platform. Includes the GNU GCC compiler for C and C++. Thus, the Data types are compliant with AVR-GCC and the summary of all Data types are :

Data type	Size
`char`	8-bit Integer signed number. Range is -128 to +127.
`unsigned char`	8-bit Integer unsigned number. Range is 0 to +255.
`int`	16-bit Integer signed number. Range is -32,768 to +32,767.
`unsigned int`	16-bit Integer unsigned number. Range is 0 to +65535
`long`	32-bit Integer signed number. Range is -2,147,483,648 to +2,147,483,647
`unsigned long`	32-bit Integer unsigned number. Range is 0 to +4294967295
`long long`	64-bit Integer signed number. Range is -9223372036854775808 to + 9223372036854775807
`unsigned long long`	64-bit Integer unsigned number. Range is 0 to +18446744073709551616
`float` and `double`	32-bit floating point
`arrays`	Data or Variable group are same data types and store in address continue.
`pointers`	The index data to access the memory address.
`structures`	Data or Variable group are different data types.

4.5 Numerical system in C program of WinAVR

WinAVR compiler has 3 types of numerical system in C program.

1. **Decimal number**
2. **Binary number** The format is 0bBBBBBBBB. Thus, B is 0 or 1
3. **Hexadecimal number** The format is 0xHHHHHHHH. Thus, H is 0 to 9, A to F

Example 4-7

The 8-bit binary number ; 0b10010010 is equal to 146 in decimal number.

The calculation : $(1x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (0x2^3) + (0x2^2) + (1x2^1) + (0x2^0) = 146_{10}$

Example 4-8

The 16-bit binary number ; 0b1111010011101101 is equal to 62701 in decimal number.

The calculation : $(1x2^{15}) + (1x2^{14}) + (1x2^{13}) + (1x2^{12}) + (0x2^{11}) + (1x2^{10}) + (0x2^9)$

$+ (0x2^8) + (1x2^7) + (1x2^6) + (1x2^5) + (0x2^4) + (1x2^3) + (1x2^2)$

$+ (0x2^1) + (1x2^0) = 62701_{10}$

Example 4-9

The hexadecimal number ; 0xFF is equal to 255 in decimal number.

The calculation : $(15x16^1) + (15*16^0) = 255_{10}$ and 0xFF → 0b11111111 in binary number.

Example 4-10

The hexadecimal number ; 0x31 is equal to 49 in decimal number.

The calculation : $(3x16^1) + (1x16^0) = 49_{10}$ and 0x31 → 0b00111111 in binary number.

4.6 Variable declaration

Variable declaration in C program of WinaVR is similar to ANSI-C programming. The General form is

```
type variable_name;
```

Thus;

`type` is The result data type

`variable_name` is variable declared

such as :

```
int a;              // Declare a variable as int data type
long result;        // Declare result variable as long data type
float start;        // Declare start vairable as  float data type
int x,y;            // Declare 2 variables; x and y. Data types are int
float p,q,r;        // Declare 3 variables; p, q and r. Data types are float.
```

In addition, programmers can declare the variables and set the initial value such as

```
int =100;               // Declare x variable.
                        // Data type is an integer and the initial value is 100.
int x=15,y=78;          // Declare x and y variable.
                        // Data type is an integer and the initial value are
                        // x=15 and y=78.
long p=47L,q=31L;       // Declare p and q variable. Data type is long
                        // and initial value are p=47 and q=31.
```

4.7 Data type conversion

The general form of the conversion is

`(type)variable`

Thus; `type` is The result data type that is required

`variable` is the variable that is required to convert the data type

Example 4-11

```
int x=100;          // Declare x variable as integer type and set its initial value to 100.
float y=43.67,z;    // Declare y and z variable as float types and set y = 43.67.
z = y+(float)x ;    // Set the value of z to be the addtion of y and x.
                    // x data is originally int.
                    // It needs to be converted to a float with (float)x command.
                    // The result of z = 143.67.
```

Example 4-12

```
int a=50;           // Declare a variable as an integer type and set its initial value
                    // to 50.
long b=23L,c;       // Declare b and c variable as long data type and set b to 23.
c = b*(long)a;      // Set the value of c to be the multiplication between b and a.
                    // a data is originally int. It's different from b and c.
                    // It need s to be converted to a long data type with the
                    // (long)a command
                    // The result of c = 1150
```

4.8 Type of variable in WinAVR compiler

4.8.1 Array

4.8.1.1 One dimension Array

The declaration form of this one dimension array is :

```
type  name[size];
```

Thus ;

`type` is Data type of an Array variable

`name` is the Array variable name

`size` is the Numberof size of Array (optional)

Accessing the member of each array has the general form as follows :

```
name[index]
```

Thus ; `index` is the Index value for pointing to any member in array. This parameter can be a number or a variable, but these must be integer format.

Example 4-13

From declaration :

```
char  arr[4];
```

It means arr is an array variable. It has 4 sub-variables such as :

`arr[0]` : It is the first member but index value is '0'

`arr[1]` : It is the second member but index value is '1'

`arr[2]` : It is the third member but index value is '2'

`arr[3]` : It is the forth member but index value is '3'

`arr[0]`, `arr[1]`, `arr[2]` and `arr[3]` variable are char data type. All variable size are 1 byte. Thus declaration of the `arr` variable requires 4 bytes of space.

Example 4-14

```
char dat[8] = {1,3,5,7,9,11,13,15} ;
```

This declares the array ; dat. It is 8 cells and the value for each cell is as follows :

```
dat[0] = 1;
dat[1] = 3;
dat[2] = 5;
dat[3] = 7;
dat[4] = 9;
dat[5] = 11;
dat[6] = 13;
dat[7] = 15;
```

For calling of individual cells after which,

```
char i , j ;
i = 3;
j = dat[i]; // j = dat[i] ==> j = dat[3] ==> j = 7
            /* The result is j = 7 */
```

Example 4-15

```
char dat[4] = " abcd" ;
```

This declares the array ; dat. It has 4 cells and the value for each cell is as follows :

```
dat[0] = 'a';
dat[1] = 'b';
dat[2] = 'c';
dat[3] = 'd';
```

For calling of individual cells after which,

```
char i , j ;
i = 3;
j = dat[i];      // j = dat[i] ==> j = dat[3] ==> j = 'd'
                 /*The result is j = 'd' */
```

The array variable can be declared as a global variable or a local variable. It can be used in parameters when transferring of data into the function.

4.8.1.2 The 2-Dimension Array

The declaration form of this two dimension array is :

```
type name[x][y];
```

This command shows a 2 dimensional array type variable.

`type` is the Data type of Array variable

`name` is the Array variable name

`x` is the Number of row in the array

`y` is the Number of column in the array

For example :

```
int a[2][5];
```

It is declaring that "a" is a 2 dimensional array. It has integer types values in 10 cells.

```
a[0][0],  a[0][1],  a[0][2],  a[0][3],  a[0][4] ,
a[1][0],  a[1][1],  a[1][2],  a[1][3],  a[1][4]
```

For the setting of the cell values, this can be done as such:

```
int menu[3][4] ={{1,3,4,9} , {2,8,0,5}};
```

This would mean that :

```
menu[0][0] = 1    menu[0][1] = 3    menu[0][2] = 4    menu[0][3] = 9
menu[1][0] = 2    menu[1][1] = 8    menu[1][2] = 0    menu[1][3] = 5
menu[2][0] = 0    menu[2][1] = 0    menu[2][2] = 0    menu[2][3] = 0
```

Chapter 5
Operators of WinAVR compiler

The Operation in C program of Win AVR compiler can be divided into 3 groups, which are the Arithmetic operator, Relation & logic operator and Bitwise operator.

5.1 Arithmetic operator

This group can be summarized into the following:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
++	Increment
- -	Decrement
+=	Add with the Right-hand value
- =	Subtract with the Right-hand value
*=	Multiply by the Right-hand value
/=	Divide by the Right-hand value
%=	Modulo by the Right-hand value

5.1.1 Addition (+) and Subtraction (-)

Example 5-1

```
int  a = 12;
a = a + 3;
```

The result is a = 15

Operation : Begin with a = 12. Add a with 3 and store the result to a. It means 12+3 = 15, store 15 to a.

Example 5-2

```
int  a = 12;
a = a - 3;
```

The result is a = 9

Operation : Begin with a = 12. Subtract a with 3 and store the result to a. It means 12-3 = 9, store 9 to a.

5.1.2 / and % division

The different of both division is :

1. / is the division of numbers which will return an integer.

2. % is the division of numbers which will return with the remainder. Called as Modulo.

Example 5-3

```
int  x , y , z;
x = 10;
y = x/3;
z = x%3;
```

The result is y = 9 and z = 1

Operation :

`y = x/3;` → `y = 10/3` → y = 3 (Returns an Integer)

`z = x%3;` → `z = 10%3` → z = 1 (Returns only the remainder)

5.1.3 ++ and - - operation

Example 5-4

```
int  y = 5;
y++;
```

The result is y = 6

Operation : Begin with y = 5. Next, y+1 = 6 and store to y. Thus, `y++;` command gives the result similar to `y = y + 1;` command

Example 5-5

```
int  y = 5;

y--;
```

The result is y = 4

Operation : Begin with y = 5. Next, y-1 = 4 and store to y. Thus `y - -;` command gives the result similar to `y = y - 1;` command

5.1.4 += and - = operation

The operation of both operations can be summarized as follows:

`y +=a;` gives the result similar to `y = y + a;`

`y -=a;` gives the result similar to `y = y - a;`

Example 5-6

```
int  x = 100;
x += 10;
```

The result is x = 110

5.1.5 *= , /= and %= operation

The operation of all operators can be summarized as follows:

`y *=a;` gives the result similar to `y = y * a;`

`y /=a;` gives the result similar to `y = y/a;`

`y %=a;` gives the result similar to `y = y%a;`

Example 5-7

```
int  x, y, z;
x = y = z = 120;
x *= 4;
y /= 4;
z %= 4;
```

The result is x = 480 , y = 30 and z = 0

5.2 Relation & logic operator

The results of these operators are "1" if the condition is true and "0" if the condition is false. These operators can be summarized as follows :

Operator	**Meaning**
==	Equal
!=	Not equal
>	More than
<	Less than
>=	More than or Equal
<=	Less than or Equal
!	NOT
&&	AND
| |	OR

Example 5-8

a = 10, b = 4, c = 0xA0

Operation	***Condition***	***Result***
a>b	*true.*	*1*
a>c	*false*	*0*
a>=c	*true*	*1 (because 0xA0 = 10)*
a!= b	*true*	*1*
a!= c	*false*	*0*

5.2.1 ! , && and || operation

! (NOT) can be summarized as follows

Operation	**Result**
! false	true(1)
! true	false(0)

In summary, *the result of NOT is to reverse the value of the input.*

&&(AND) can be summarized as follows

Operation	**Result**
false && false	false(0)
false && true	false(0)
true && false	false(0)
true && true	true(1)

In summary, *the result of AND will be false if one of condition or both are false.*

||(OR) can be summarized as follows

Operation	**Result**
false || false	false(0)
false || true	true(1)
true || false	true(1)
true || true	true(1)

In Summary, *the result of OR will be true if one of the condition or both are true.*

Example 5-9

Determine a = 10, b = 4 and c = 0xA0

Operation	***Condition***	***Result***
a>b	*true.*	*1*
a>b	*true*	*1*
a>c	*false*	*0*
a>=c	*true*	*1 (because 0xA0 = 10)*
a != b	*true*	*1*
a != c	*false*	*0*

or

!(a>b)	*false*	*0*
!(a>c)	*true*	*1*
!(a>=c)	*false*	*0*
!(a != b)	*false*	*0*
!(a != c)	*true*	*1*

or

Operation	***Result***
!(a>b) && (a>=c)	*false(0)*
(a != b) && (a>=c)	*true(1)*
(a != b) && !(a != b)	*false(0)*

and

!(a>b) || (a>=c)	*true(1)*
(a != b) || (a>=c)	*true(1)*
(a != b) || !(a != b)	*true(1)*
!(a>=c) || !(a != b)	*false(0)*

5.3 Bitwise Operator

The operators can be summarized as follows

Operator	Meaning
~	Invert bit
&	Bit AND
|	Bit OR
^	Bit XOR
<<	Shift Left
>>	Shift Right
<<=	Shift left and Store the result to variable
>>=	Shift right and Store the result to variable
&=	AND operation with Store the result to variable
|=	OR operation with Store the result to variable
^=	XOR operation with Store the result to variable

5.3.1 Bit logic Operation

~ operation can summary as follows

Operation	Result
~ 0	1
~ 1	0

& operation can summary as follows

Operation	Result
0 & 0	0
0 &1	0
1 & 0	0
1 & 1	1

| operation can summary as follows

Operation	Result
0 | 0	0
0 | 1	1
1 | 0	1
1 | 1	1

^ operation can summary as follows

Operation	Result
0 ^ 0	0
0 ^1	1
1 ^ 0	1
1 ^ 1	0

Example 5-10

Determine :

```
int  x,y,result1,result2,result3,result4;
x  =  0x9C;
y  =  0x46;
```

Find the result of :

(1) `result1  =  x&y;`

(2) `result2  =  x|y;`

(3) `result3  =  x^y;`

(4) `result4  =  ~x;`

Solution :

Firstly, convert all value to a binary number.

x = 0x9C → 0000000010011100 (because int data type is 16-bit wide)

y = 0x46 → 0000000001000110

(1) result1 = (0000000010011100) & (0000000001000110)

0000000010011100
AND
0000000001000110
0000000000000100 → 0x0004 or 0x04

(2) result2 = (0000000010011100) | (0000000001000110)

0000000010011100
OR
0000000001000110
0000000011011110 → 0x00DE or 0xDE

(3) result3 = (0000000010011100) ^ (0000000001000110)

0000000010011100
XOR
0000000001000110
0000000011011010 → 0x00DA or 0xDA

(4) result4 = ~(0000000010011100) *Invert all bit*

1111111101100011 → 0xFF63

5.3.2 Shift bit operation

In the shifting bit operation, you must determine the number of shifting such as :

```
dat = dat<<4;
```

It means shifting to the left of the dat variable 4 times and storing the result into the dat again.

Another example is

```
dat = dat>>1;
```

It means shifting to the right of the dat variable 1 bit and storing the result into the dat again.

Example 5-11

```
int dat, x1, x2;
dat = 0x93;
```

Find the result of

(1) `x1 = dat<<1;`

(2) `x2 = dat<<2;`

Solution :

dat = 0x93 → 0000000010010011

```
dat    0000000010010011
X1     0000000100100110
X2     0000001001001100
```

(1) x1 is Shifting left 1 bit results in a dat vairable. Thus,

x1 = 0x0126 or 294 in decimal number.

(2) x2 is Shifting left 2 bits results in a dat vairable. Thus,

x2 = 0x024C or 588 in decimal number.

Example 5-12

```
int a , b , c;
a = 0x7A;
b = 0x16;
c = 0xFD;
```

Find the result of

(1) `a &= 0x3C;`

(2) `b |= 0x51;`

(3) `c ^= 0xD0;`

Solution :

(1) From `a &= 0x3C;`, it is equal `a = a & 0x3C;`. It means get the value of a (0x7A) AND with 0x3C and store the result back to a again.

It is equivalent to : a = (0000000001111010) & (0000000000111100)

0000000001111010

AND

0000000000111100

0000000000111000 → ***0x0038 or 0x38***

(2) From `b |= 0x51;`, it is equal `b = b |= 0x51;` . It means get the value of b (0x16) OR with 0x51 and store the result back to b again.

It is equivalent to : b = (0000000000010110) |= (0000000001010001)

0000000000010110

OR

0000000001010001

0000000001010111 → ***0x0057 or 0x57***

(3) From `c ^= 0xD0;`, it is equal `c = c ^= 0xD0;`. It means get the valuse of c (0xFD) XOR with 0xD0 and store the result back to c again.

It is equivalent to : c = (0000000011111101) ^= (0000000011010000)

0000000011111101

XOR

0000000011010000

0000000000101101 → ***0x002D or 0x2D***

Chapter 6
Library and Specific command in MicroCamp kit

The MicroCamp Activity kit comes with a lot of libraries to support developers and learners. It includes Input/Output port control library, Analog input reading library, Delay time library, Sound library and Motor control library.

The summary of all libraries are as follows :

- **in_out.h** Library for Sending digital data to the output port and Reading the Digital input port.
- **sleep.h** Delay function library
- **analog.h** Analog input reading library. Assist in reading of analog data from P0 to P4 port
- **led.h** LED control library
- **motor.h** DC motor control library
- **sound.h** Sound generator library
- timer.h Timer function library

All libraries must be stored in the same folder for proper linking of paths and to avoid any errors. Learners can see details of all libraries from the ***MicroCamp_include*** folder in CD-ROM which is bundled with the MicroCamp Activity kit.

6.1 Command in in_out.h library

6.1.1 Digital input port reading function

`in_b` : Port B input reading function

`in_c` : Port C input reading function

`in_d` : Port D input reading function

Function format :

```
char in_a(x)
char in_b(x)
char in_c(x)
char in_d(x)
```

Parameter :

`x` - determines the number of the input port that will be used. The value is 0 to 7.

Return value :

"0" or "1"

Example 6-1

```
char x=0;                // Declare x to store the result.
x = in_b(2);             // Get PB2 value to store in x
```

Example 6-2

```
char x=0;                // Declare x to store the result.
x = in_d(4);             // Get PD4 value to store in x
```

Example 6-3

```
#include <avr/io.h>      // Includes the Standard input/output port library
#include <in_out.h>      // Includes the Port control library
#include <sound.h>       // Includes the Sound generator library
void main()
{
    while(1)
    {
        if (in_d(2)==0)            // Check SW1 pressed ?
        {
            sound(3000,100);       // Generate sound if SW1 is pressed
        }
    }
}
```

6.1.2 Sending data to output port function

This function determines the port pin, configures it to output and sends the value to that port. These function does not return any values.

`out_b` **: Port B output sending function**

`out_c` **: Port C output sending function**

`out_d` **: Port D output sending function**

Function format :

```
out_b(char _bit,char _dat)
out_c(char _bit,char _dat)
out_d(char _bit,char _dat)
```

Parameter :

`_bit` - select port's pin. Range is 0 to 7.

`_dat` - determine the output value "0" or "1" to output pin

Example 6-4

```
out_c(5,0);                 // Send logic "0" to PC5 port.
out_d(1,1);                 // Send logic "1" to PD1 port.
```

Example 6-5

```
#include <avr/io.h>         // Includes the Standard input/output port library
#include <in_out.h>         // Includes the Port control library
#include <sound.h>          // Includes the Sound generator library
void main()
{
    while(1)                // Looping.
    {
        out_d(1,1);         // Outs logic "1" via PD1. The LED2 indicator on.
        sleep(300);         // Delays 0.3 second.
        out_d(1,0);         // Outs logic "0" via PD1. The LED2 indicator off.
        sleep(300);         // Delays 0.3 second.
    }
}
```

6.1.4 Invert logic output port function

toggle_b : Port B output invert logic function

toggle_c : Port C output invert logic function

toggle_d : Port D output invert logic function

Function format :

```
toggle_b(x)
toggle_c(x)
toggle_d(x)
```

Parameter :

x - Determines the port number. The value is 0 to 7.

Example 6-6

```
toggle_c(5);     // Invert logic at PC5 port.
toggle_d(1);     // Invert logic at PD1 port.
```

6.2 Delay function in sleep.h library

This library only has one function. It is the **sleep** function. Developers can use this function to pause or delay the operation in millisecond unit.

Function format :

```
void sleep(unsigned int ms)
```

Parameter :

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

Example 6-7

```
sleep(20);       // Delays 20 millisecond approximation.
sleep(1000);     // Delays 1 minute approximation.
```

6.3 analog.h library : Analog input reading library

6.3.1 analog function

This is reading of an analog value. It reads from PC0 to PC4 pins. Analog signals will pass through the Analog to Digital Converter inside ATmega8 microcontroller. The converter resolution is 10-bit. The digital output value in decimal number is 0 to 1,023 refer 0 to 5V DC voltage.

Function format :

```
unsigned int analog(unsigned char channel)
```

Parameter :

`channel` - select the analog input. Range is 0 to 4. It means PC0 to PC4

Return value :

The digital data from the conversion, range is 0 to 1,023 in decimal number.

Example 6-8

```
int adc_val=0;            // Set the variable for storing the analog reading data
adc_val = analog(0);  // Read from analog channel 0 (PC0) and sotre in adc_val.
```

6.4 LED blinking function in led.h library

The MicroCamp board provides 2 LEDs at PC5 (LED1) and PD1(LED2) pins. The LED blink operation is a very simple method which sends logic “0” and “1” toggle always. However developers can use a function to allow this operation to run concurrently with other functions, with using the LED blinking function in **led.h** library.

`led1_on()` : enable LED1 (PC5) blinking

`led1_off()` : disable LED1 (PC5) blinking

`led2_on()` : enable LED2 (PD1) blinking

`led2_off()` : disable LED2 (PD1) blinking

Example 6-9

```
void main()
{
      led1_on();          // LED1 still blink although the Main program execute finished
}
```

6.5 motor.h : Motor control library

6.5.1 motor function

This function is used for controlling the DC motor driver circuit on the MicroCamp controller board.

Function format :

```
void motor(char _channel,int _power)
```

Parameter :

`_channel` - select motor output channel. On MicroCamp control board has 2 channels; 1 and 2.

`_power` - determine the power apply for motor output.
Range is -100 to 100.
If value is positive (1 to 100), the motor will spin in a direction.
If value is negative (-1 to -100), the motor spin the other direction.
If the value is 0, motor will stop but this do not lock the motor's shaft.

Example 6-10

```
motor(1,60);     // Drive motor channel 1 with 60% of power.
................
motor(1,-60);    // Drive motor channel 1 with 60% of power in the opposite direction.
```

Example 6-11

```
motor(2,100);    // Drive motor channel 2 with full power (100%).
```

6.5.2 motor_stop function

It is the brake motor function. The motor's shaft will lock after active this function.

Function format :

```
void motor_stop(char _channel)
```

Parameter :

`_channel` - select motor output channel. This parameter has 3 values.
1 for braking motor at OUT1 channel
2 for braking motor at OUT2 channel
ALL for braking all motor channel

Example 6-12

```
motor_stop(1);       // Brake motor channel 1.
motor_stop(2);       // Brake motor channel 2.
motor_stop(ALL);     // Brake motor both channel (1 and 2).
```

6.5.3 motor_off function

This function is used for stopping the motor operation and to turn-off the voltage of all motor outputs. This function is similar to the motor function which sets the power value to 0.

Function format :

```
void motor_off()
```

6.5.4 forward function

This function is used for driving DC motor to move the robot in forward direction.

Function format :

```
void forward(int speed)
```

Parameter :

`speed` - determine the power applied to motor output. Range is 0 to 100.

6.5.5 backward function

This function is used for driving DC motor to move the robot in a backward direction.

Function format :

```
void backward(int speed)
```

Parameter :

`speed` - determine the power applied to motor output. Range is 0 to 100.

6.5.6 s_left function

This function is used for driving the DC motor to spin the robot in a left direction.

Function format :

```
void s_left(int speed)
```

Parameter :

`speed` - determine the power applied to motor output. Range is 0 to 100.

6.5.7 s_right function

This function is used for driving the DC motor to spin the robot in a right direction.

Function format :

```
void s_right(int speed)
```

Parameter :

`speed` - determine the power applied to motor output. Range is 0 to 100.

6.6 sound.h : Sound generator library

This function is used for setting the sound frequency which drives the piezo speaker on the MicroCamp controller board to produce sounds.

Function format :

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

Parameter :

`freq` - determine the frequency output in Hertz (Hz) unit.

`time` - determine the time value of sound output signal in millisecond unit.

Example 6-13

```
sound(2000,500);        // Generate 2kHz signal for 500 millisecond.
```

6.7 Counting time function in timer.h library

6.7.1 timer_start function

Determine the start point of the timer. After this function, the timer value will be cleared.

Function format :

```
void timer_start(void)
```

6.7.2 timer_stop function

This stops the timer and clears the counting value.

Function format :

```
void timer_stop(void)
```

6.7.3 timer_pause function

Pause timer counting. The value still remains.

Function format :

```
void timer_pause(void)
```

6.7.4 timer_resume function

Resume the counting after a pause from **timer_pause** function.

Function format :

```
void timer_resume(void)
```

6.7.5 msec function

Read the timer value in milliseconds.

Function format :

```
unsigned long msec()
```

Return value :

Time value is in millisecond. The data type is a “long” variable.

6.7.6 sec function

Read the timer value in seconds.

Function format :

```
unsigned long sec()
```

Return value :

Time value is in second. The data type is a “long” variable.

Example 6-14

```
#include <in_out.h>
#include <sleep.h>
#include <timer.h>
void main()                    // Main program
{
   timer_start();              // Set the startin point of timer
   while(1)                    // Endless loop
   {
      if(msec()>500)           // Check timer value. Is it more than 500 ?
      {
         timer_stop();         // Stop and clear the timer value.
         toggle_c(5);          // Toggle LED indicator every 0.5 second.
         timer_start();        // Start timer counting again.
      }
   }
}
```

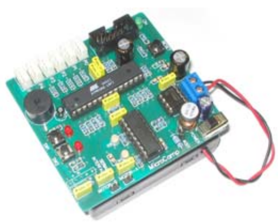
Chapter 7
Building robot with MicroCamp kit

This chapter focus learning the applications of the MICROCAMP microcontroller. The building of a robot integrates knowledge and technology which includes electronics, programming, mechanical movements, and thinking process. The Microcamp Activity kit supports this concept. This kit includes all parts for building a simple mobile robot. Users can learn about programming and how to apply the microcontroller aspects via robotic activities.

The Mobile robot in MICROCAMP has 2 DC Motor gearboxes for moving and 4 sensors for detecting external values. These are 2 touch sensors and 2 Infrared Reflector Line tracking sensors for use in black and white line following.

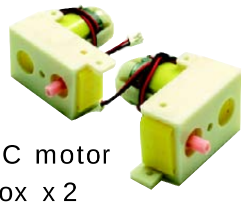
Part list

Circle base plate



MicroCamp board

Box holder x 1



48:1 DC motor gearbox x 2

Wheel and Tire set x 2

Plastic spacer set x 1

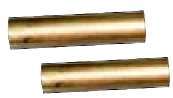
Nut and Screw set x 1

Ploastic joiners (Straight, Right angle and Obtuse)

Infrared reflector x 2

Swithc module x 2

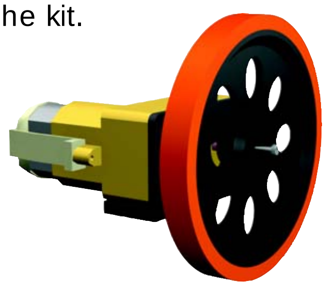
2mm. Self-tapping screw x2



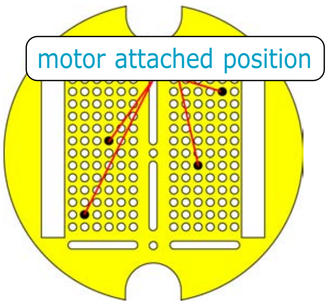
25mm. metal spacer x 2

Construction

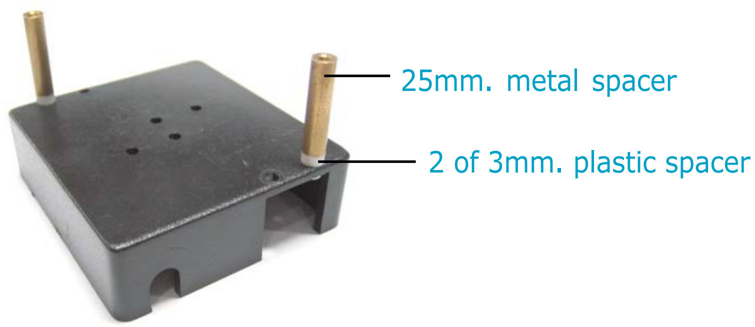
1. Fix on the 2 wheels with the rubber tires and attach them to the DC Gearbox with the 2 of the 2mm. self-tapping screws provided in the kit.

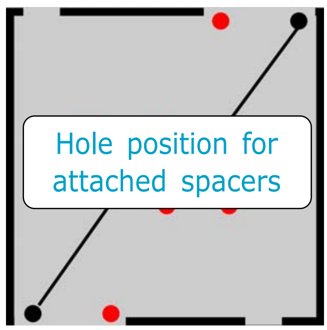


2. Install both the DC Gearboxes on the circular base plate at the specific positions shown in the picture with 4 of 3 x 6mm. machine screws.

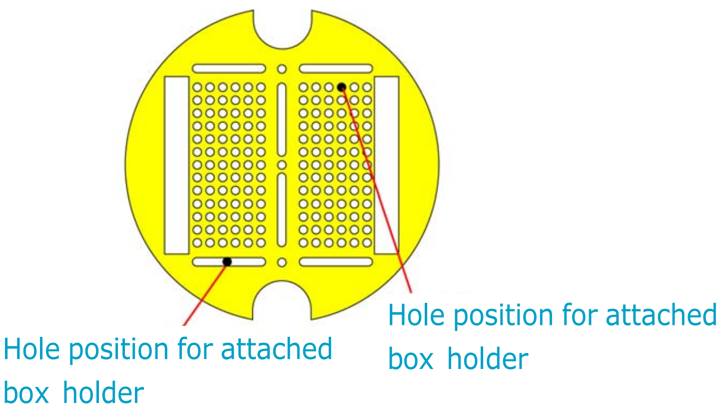


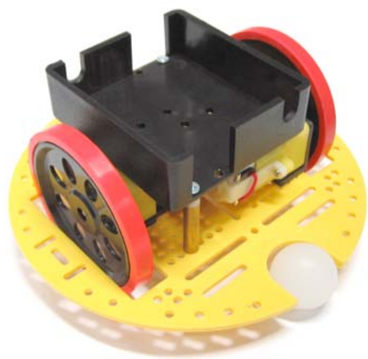
3. Insert the 3 x 10mm. machine screws through the hole at the corner of the Box holder with 25mm. and 2 of 3 mm. spacers.



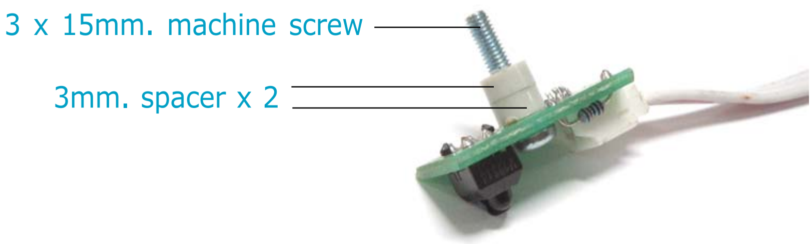


4. Place the Box holder from step 3 on the top of the Circle base plate and attach them with 3 x 10mm. screws at the specific positions.

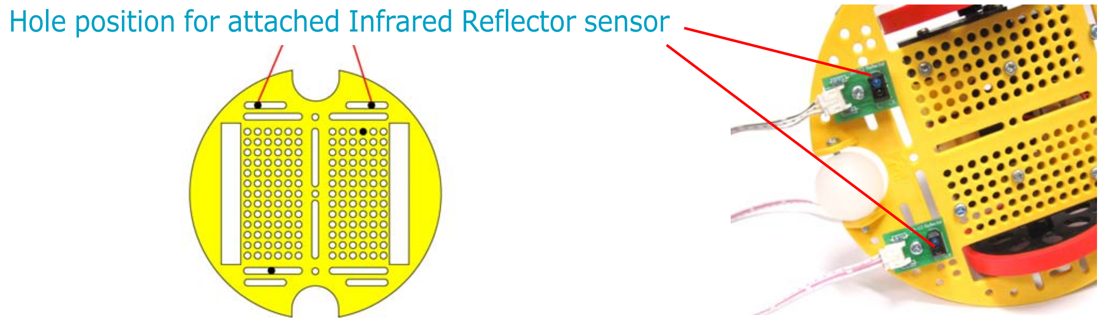




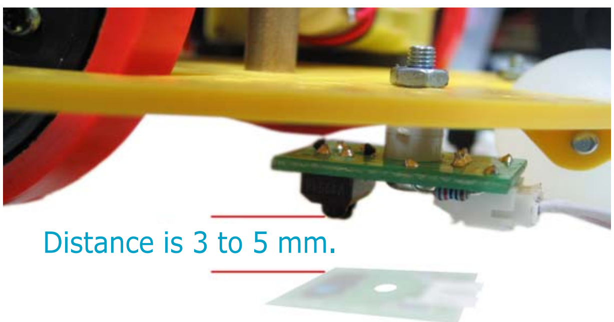
5. Insert a 3x15mm. machine screw through the Infrared Reflector sensor, followed by 2 of the 3mm. spacer. Do on both sides for this.

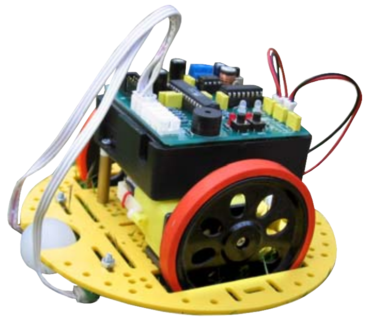


6. Attach both the Infrared Reflector structures from step 5 at the suitable holes at the bottom and front of the robot base. Tighten with a 3mm. nut.

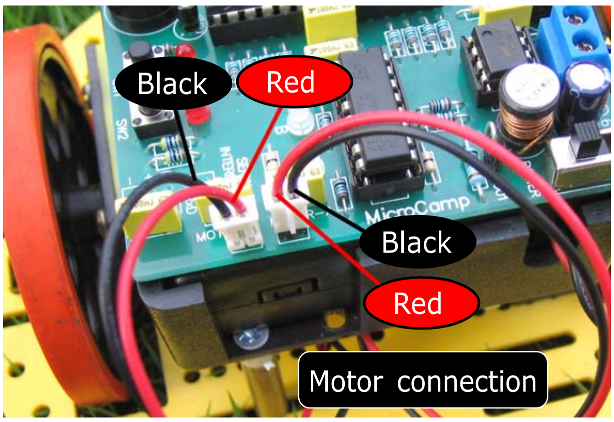


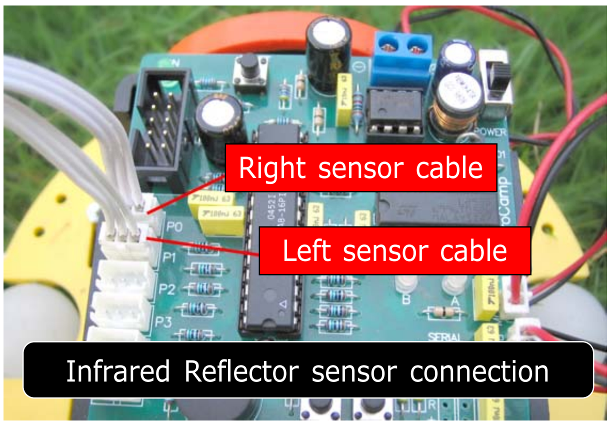
7. Observe the distance from the floor to the sensors. The suitable distance is 3 to 5 mm.



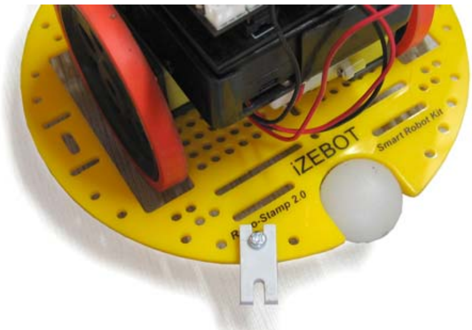


8. Place MicroCamp board on the box holder. Connect sensor cables and motor cables following the diagrams shown. (P0 for Right sensor and P1 for Left sensor).

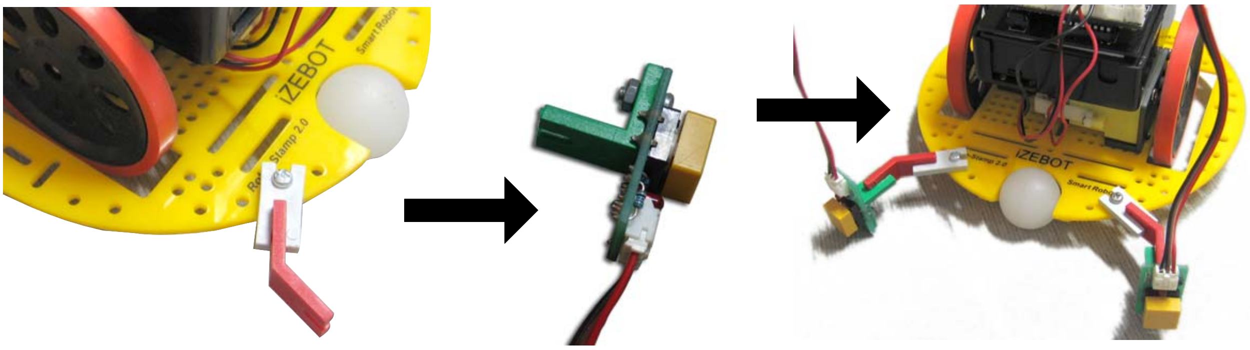




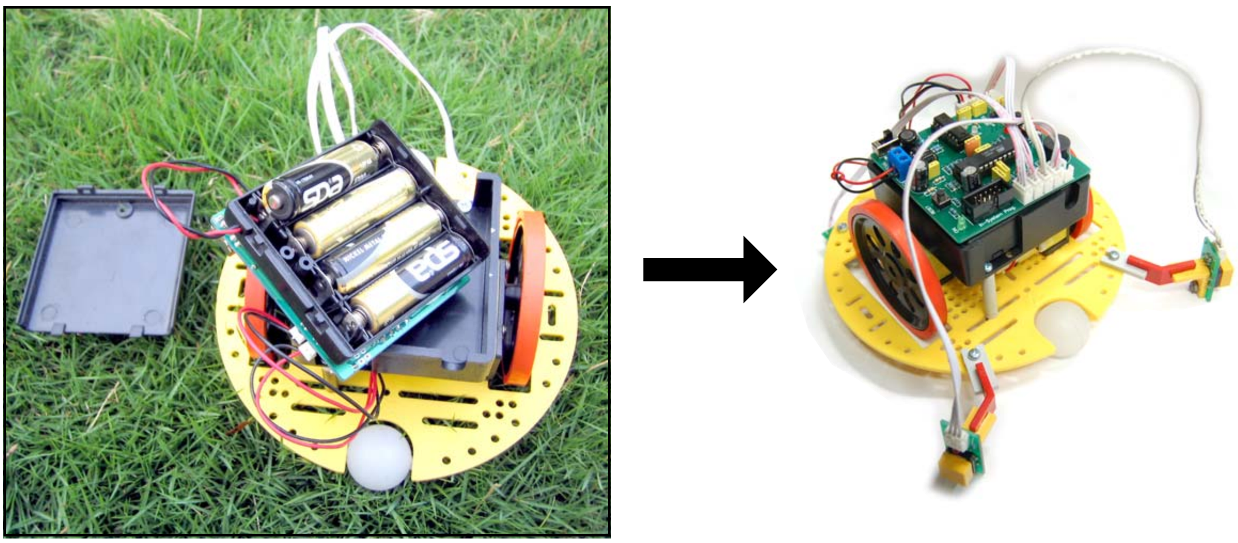
9. Attach the Straight joiner with robot base at front-right side by 3 x 10mm. machine screw and 3mm. nut. Attach 2 pieces.



10. Connect the Obtuse joiner at the end of Straight joiner. Attach the right angle joiner with Switch module by 3 x 10mm. machine screw and 3mm. nut. Make 2 sets. Bring these structures to connect at the end of the Obtuse joiner. Connect 2 sides.

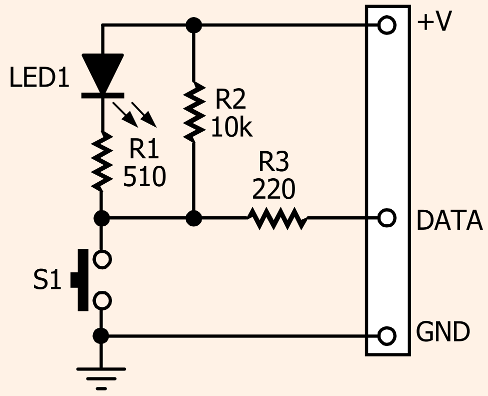


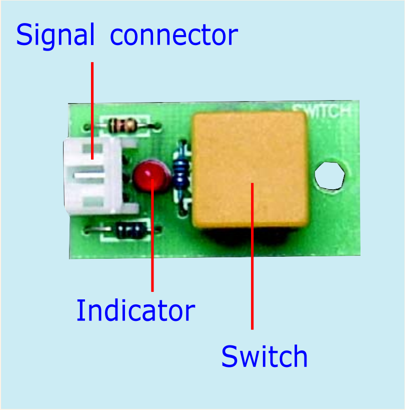
11. Connect the Left Switch module cable to the P2 (PC2) connector and the Right Switch module cable to the P3 (PC3) connector. Put 4 AA batteries into battery holder at the back of MicroCamp board. **The MicroCamp robot is ready for programming now.**



Learning about the Switch circuit

The switch that is used with the MicroCamp has the following schematic:





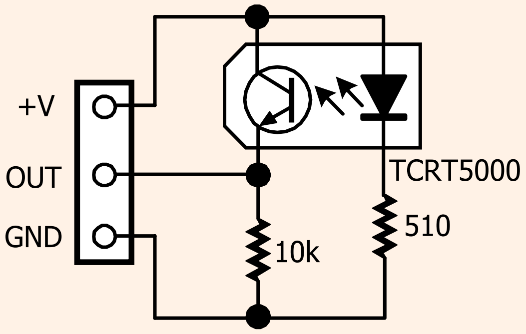
Pressing the switch results in two occurrences.

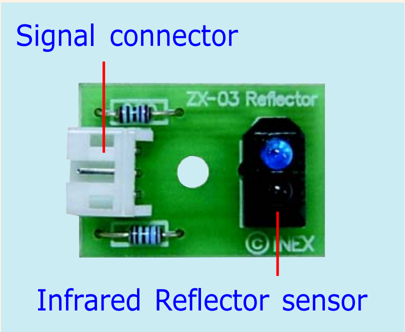
When the switch is not pressed, let the results be logic "1"

When the switch is pressed, let the results be logic "0", and LED1 light up.

Since the switch can give two results, it is considered to be a digital input component.

More information of Infrared Reflector





The heart of this sensor circuit is the sensor that detects reflections from infrared light. It consists of the Infrared LED which emits infrared light to the surface. Photo-transistors will then receive the reflected infrared lights. If no infrared light is received, the OUT terminal will have low voltage when measured. In the case that it receives infrared light, whether low or high current passes through the photo-transistor depends on the intensity of the light received which in turn varies according to the distance of the reflection. (Sensor TCRT5000 can be used at a distance of 0.1 – 1.5 centimeters).

Therefore, 0.5 – 5V can be measured at the OUT terminal, and *the MicroCamp will get a value of 30 to 1023.*

Activity 1
Basic movement of MicroCamp robot

Activity 1-1 Forward and Backward movement

A1.1 Open the AVR Studio to create the new project and write the C program following the Listing A1-1. Build this project.

A1.2 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

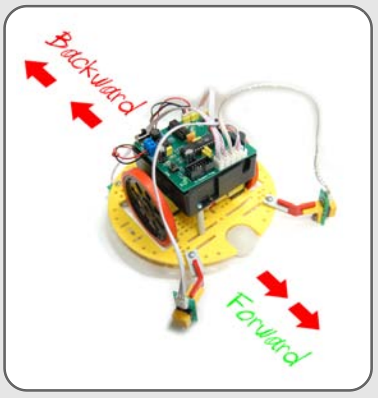
A1.3 Turn-off power and Remove the ISP cable.

A1.4 Make sure the robot is on a flat surface. Turn-on the power and observe the operation.

The MicroCamp robot moves forward. See both LED motor indicators light in green color. After 1 second, both indicators change color to red and the robot moves backward.

If this is incorrect *you will need to re-connect the motor cable to its opposite port / polarity. Do this until your robot moves correctly. Once its done, Use this motor port configuration for all your programming activities from now on. The robot will move forward and backward continually until you turn off its power.*

```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>   // Motor driver library
void main()
{
   while(1)          // Endless loop
   {
      forward(100);  // Move the robot forward.
      sleep(1000);   // Delays 1 second.
      backward(100); // Move the robot backward.
      sleep(1000);   // Delays 1 second.
   }
}
```



Listing A1-1 The C Program that allows the Microcamp Robot to move in circles.

Activity 1-2 Circle-shape movement control

A1.5 Create a new project file and write the following C Codes shown in A1-2.

A1.6 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A1.7 Turn-off power and Remove the ISP cable.

A1.8 Make sure the robot is on a flat surface. Turn-on the power and observe the robot.

The robot will be activated when you press SW1 and move in circles continually until you press the SW2 to stop the robot movement.

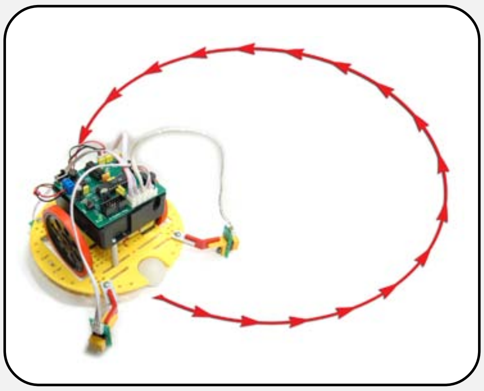
```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>
void main()
{
   while(1)
   {
     while((in_d(2)==1));     // Loop for checking SW1 pressed
     motor(1,100);            // Apply full power for Motor 1
     motor(2,30);             // Apply 30% power for Motor 2
     while((in_d(3)==1));     // Loop for checking if SW2 pressed
     motor_off();             // Stop all motors.
   }
}
```

Program description

In Listing A1-2, the forward and backward com--mands are not used for driving the robot. The MOTOR function is used instead. This function can control both motor outputs separately. This means that you can control both the motor's speed differently.

When both speeds are not equal, the robot will move towards the direction where the motor is of a lower speed. If the speed difference is great, the MicroCamp robot will move in circles.

The While command is used in this program. If SW1 at PD2 port is being pressed, the LOGIC value of “O” is returned. The first conditional loop is false. It then continues with the second conditional loop. If SW2 at PD3 port is press, the Program will stop both motors. The Robot will stop its movement.



Listing A1-2 The C program for MicroCamp robot move circle shape activity.

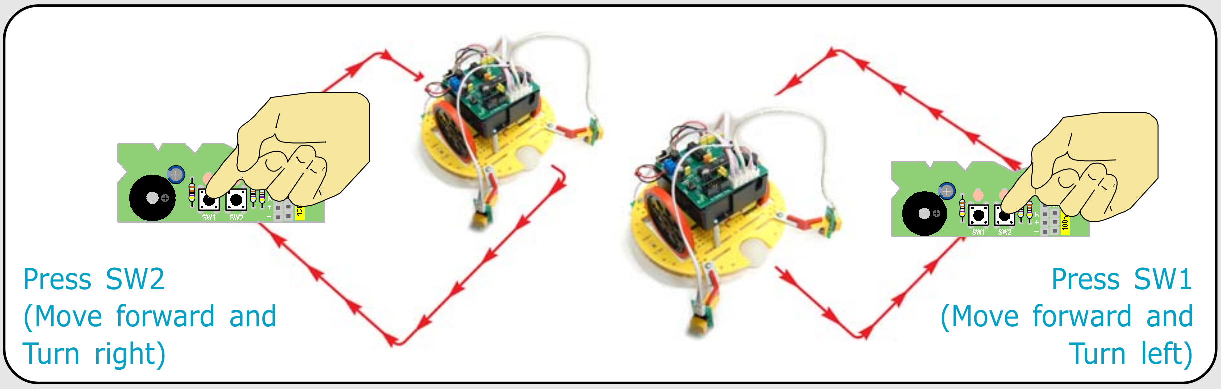
Activity 1-3 Square-shape movement control

A1.9 Create a new project file and write the following C Codes shown in A1-3. Connect the PX-400 programmer box to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A1.10 Turn-off power and Remove the ISP cable. Make sure the robot is on a flat surface. Turn-on the power and observe the robot.

The robot will be activated if SW1 or SW2 is being pressed. ***If you Press SW1****, the robot will move forward and turn left continually, making a square.* ***If you press SW2****, the operation is vice versa.*

```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>
void main()
{
   while(1)                       // Looping
   {
     if (in_d(2)==0)              // Check SW1 pressing
     {
        while(1)
        {
           forward(100);          // Move forward with full speed 0.9 second
           sleep(900);
           s_right(50);           // Turn right with 50% speed 0.3 second
           sleep(300);
        }
     }
     if (in_d(3)==0)              // Check SW2 pressing
     {
        while(1)
        {
           forward(100);          // Move forward with full speed 0.9 second
           sleep(900);
           s_left(50);            // Turn left with 50% speed 0.3 second
           sleep(300);
        }
     }
   }
}
```



Listing A1-3 The C Program for movement selection of the Microcamp Robot.

Activity 2
Object detection with Collision

Activity 2-1 Simple collision detection

This activity is program the robot to detect the collision of both switches at the front of the MicroCamp robot. After a collision is encountered, the robot will move backward and change the its direction of movement.

A2.1 Create a new project file and write the following C Codes shown in A1-4. Build this project.

A2.2 Connect the PX-400 programmer box to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot.

A2.3 Download the HEX code to the robot.

A2.4 Turn-off power and Remove the ISP cable.

A2.6 Prepare the demonstration area by placing and securing boxes or objects on the surface.

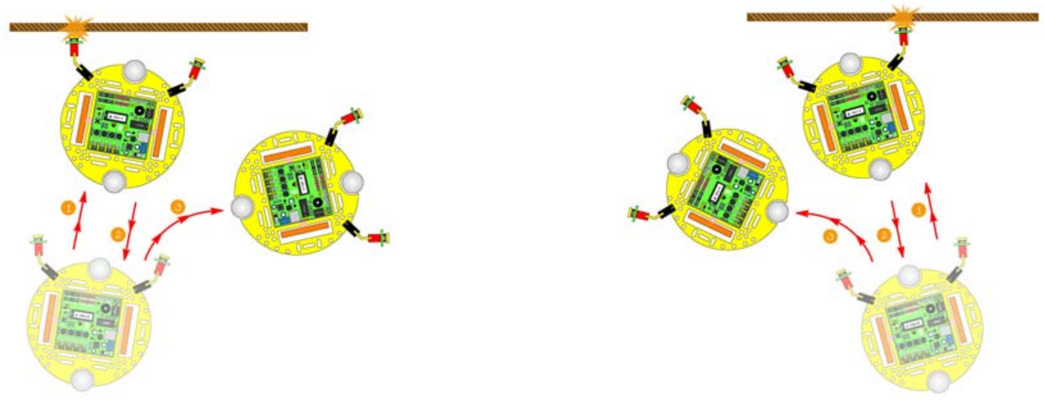
A2.7 Bring the robot into the demonstration area .Turn-on the power and observe the robot. The MicroCamp robot will read both switch status from PD2 and PC3 port. If any switch is pressed or touches some object, the result is logic “0”.

In a normal operation*, the robot will move forward continually.*

If the Left Switch module touches any object*, the robot will move backward and change its moving direction to its right to avoid the object.*

If the Right Switch module touches any object*, the robot will move backward and change its moving direction to its left to avoid the object.*

```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>
void main()
{
   while((in_d(2)==1));    // Loop until SW1 is pressed to start the program.
   while(1)                // Repeat loop
   {
     if (in_c(2)==0)       // Check status of the right switch.
     {
        backward(100);     // If ther is a collision, the robot moves backward
                           // for 0.4 second
        sleep(400);
        s_left(50);        // and turns left for 0.3 second.
        sleep(300);
     }
     else if (in_c(3)==0)  // Check status of the left switch.
     {
        backward(100);     // If ther is a collision, the robot moves backward
                           // for 0.4 second
        sleep(400);
        s_right(50);       // and turns right 0.3 second.
        sleep(300);
     }
     else
     {
        forward(100);      // No collision is deteced,
                           // the robot moves forward continually.
     }
   }
}
```



Robot attacks the object in the left.   Robot attacks the object in the right.

Listing A2-1 The C Program for Object Collision detection

Activity 2-2 Trapped in a corner situation

When the Robot is in a corner, it is caught in between whereby to the left or right is a wall. This causes continous hitting of the walls and thus trapping the robot in this corner. The solution is to modify your exiting C Code from A2-1 to that which is shown in A2-2.

A2.8 Create a new project file for making the C program according to Listing A2-2.

A2.9 Connect the PX-400 programmer box to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot.

A2.10 Prepare the demonstration area by placing and securing boxes or objects on the surface.

A2.11 Bring the robot into the demonstration area .Turn-on the power and observe the robot.

The robot will move forward and check for collision. If this happens over 5 times consecutively, the robot will spin 180 degrees to change its direction.

```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>
#include <sound.h>             // Sound library
void main()
{
   unsigned char cnt_=0;       // Declare variable for counting the number of
                               // collision both left and right.
   while((in_d(2)==1));        // Wait for SW1 is pressed to start operation
     sound(3000,100);          // Beep at once
   while(1)                    // Looping
   {
     if (in_c(2)==0)           // Check the right-side collision
     {
        if ((cnt_%2)==0)       // Check the counter as even number or not.
                               // If yes, means the previous collision is left-
                               // side collision.
        {cnt_++;}              // Increment the counter
        else                   // If not left-side collision,
        {cnt_=0;}              // clear the counter
        backward(100);         // Move backward 0.4 second
        sleep(400);            //
        s_left(50);            // Turn left
        if (cnt_>5)            // Check the counter over 5 or not.
        {
           sleep(700);         // If over, turn left more 0.7 second.
           sound(3000,100);    // Drive sound to piezo speaker
        cnt_=0;                // Clear counter
        }
```

Listing A2-2 The C program for MicroCamp robot in Trapping wall solution activity (continue..)

```
         else                    // If counter is less than 5,
         { sleep(300);}          // Set time value for turning to 0.3 second.
    }
      else if (in_c(3)==0)      // Check the leftt-side collision
      {
         if ((cnt_%2)==1)        // Counter is odd number or not.
                                 // If yes, the previous collision is right-side.
         {cnt_++;}               // Increment counter
         else
         {cnt_=0;}               // If not, clear counter
         backward(100);          // Robot move backward for 0.4 second
         sleep(400);             //
         s_right(50);            // Turn right for 0.3 second
         sleep(300);             //
      }
      else                       // If not collision, move forward.
      {forward(100);}
    }
}
```

Listing A2-2 The C program for MicroCamp robot in Trapping wall solution activity (final)

Chapter 8
Serial LCD module activity with MicroCamp

The SLCD16x2 is the 16 characters 2 lines LCD module that communicates by serial interface. It receives data serially and displays on the LCD. Accept serial data at 2400 or 9600 baud rate and accepts either TTL or RS-232 levels by selection of 2 jumpers. Supports the standard LCD controller HITACHI HD44780 or SEIKO EPSON SED1278 compatible. Both 1/8 Duty and 1/16 Duty of 1x16 LCD Module can be used by jumper selection too.

Normally LCD interfacing requires at least 6 wires but SLCD16x2 need only one signal wire. This display module is suitable for MicroCamp robot.

8.1 SLCD16x2 information

8.1.1 Features

- Serial Input RS-232 or Invert/Non-invert TTL/CMOS logic level.
- 1/8 or 1/16 Duty can be selected by jumper.
- Scott Edwards's LCD Serial Backpack™ command compatible addition with Extended Command that make LCD control easier.
- Easy to interface with the microcontroller
- Operation with +5 to 12 Vdc supply

8.1.2 Setting up

In the figure 8-1, it shows the detail of SLCD16x2 backside. User will see 4 jumpers to configuration as follows

(1) **Mode command jumper :** Select the command modes. SLCD16x2 has 2 modes. One is Standard command (ST). This mode compatible with Scott Edwards's LCD Serial Backpack™ . Another mode is Extended mode command (EX). ***For MicroCamp activities select Standard command mode (ST).***

(2) **Lines jumper** : Select the line displays ; 1/8 and 1/16 Duty. 1/8 Duty means displaying 8 digit per line. 1/16 Duty means displaying 16 digit per line or more. Normally set to ***1/16.***

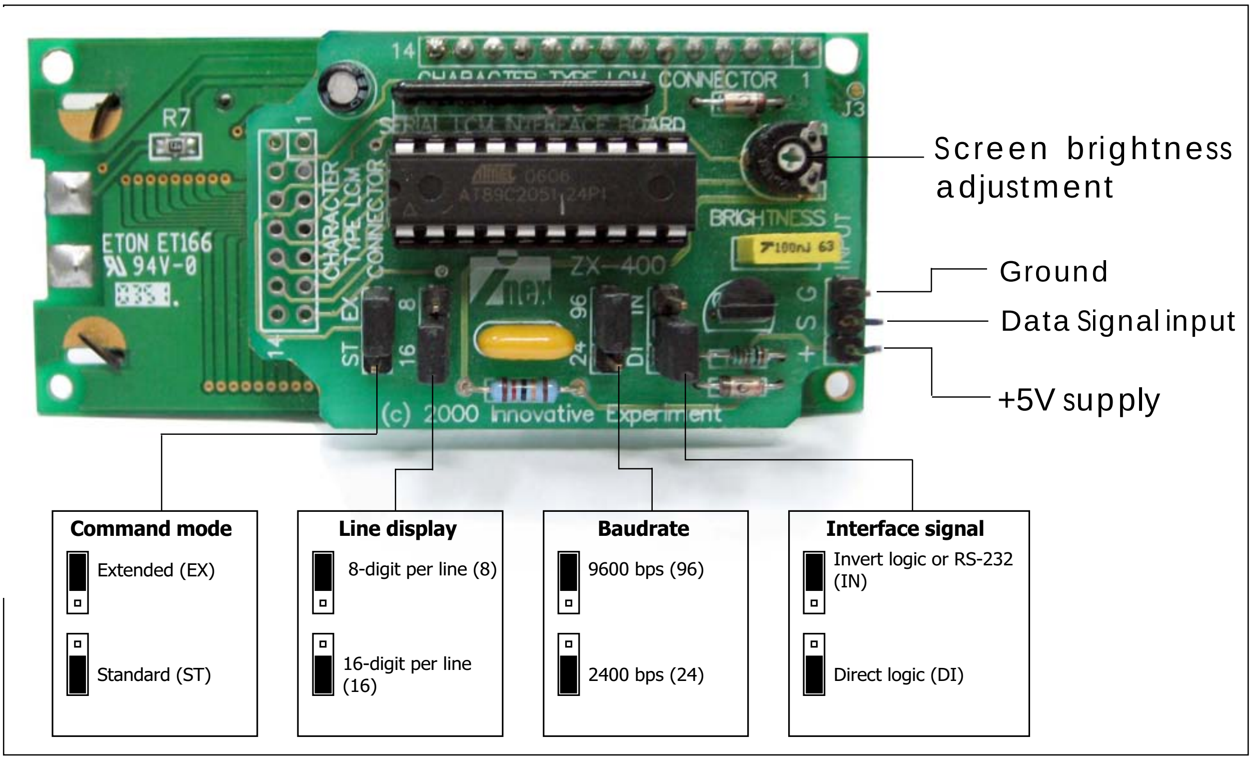


Figure 8-1 Details of SLCD16x2's jumper selections

(3) **Baudrate select jumper** : 2 selections as 2400 and 9600 bps (bit per second) with 8N1 data format (8-bit data, no parity bit and 1 stop bit)

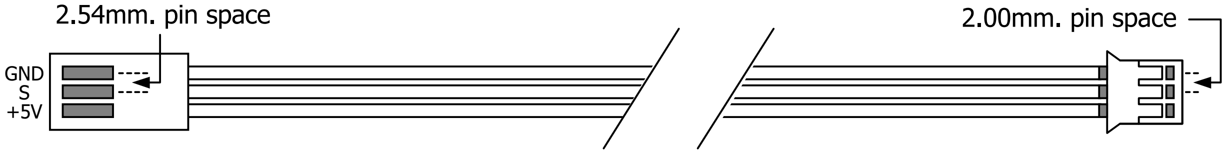
(4) **Interface signal jumper :** 2 selections as Invert logic TTL/CMOS level or RS-232 (IN) and Direct logic TTL/CMOS level (DI).

SLCD16x2 provides a brightness adjustment with variable resistor at **BRIGHTNESS** position.

Interfacing connector has 3 pins : +5V Supply voltage (**+**), Serial data input (**S**) and Ground (**G**).

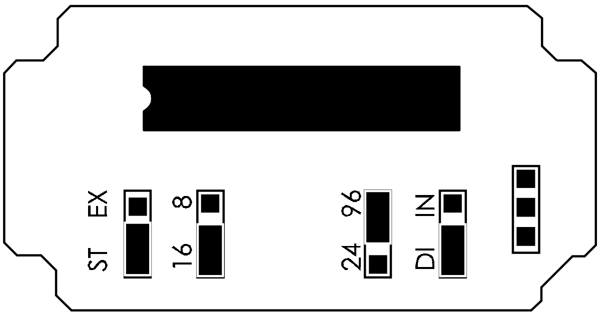
8.1.3 Interfacing SLCD16x2 with MicroCamp

The JST3AB-8 cable is required for connecting between SLCD with MicroCamp controller board. This cable wire assignment can show below.



The JST3AB-8 cable has the one end as 2.54mm. housing (call B-end) and another end as 2.00mm. housing (call A-end) The A-end will be connect to JST connector of any port (P0 to P4) of MicroCamp controller board. The B-end is connected to the input connector of SLCD16x2

After conecting, set all jumpers following the figure below.



- Select command mode to Standard (ST).
- Select the lines display to 16-digit per line (16).
- Select baudrate to 9600 bps (96).
- Select the interface siganl to Direct (DI).

8.1.4 Data and Command sending

Once the SLCD16x2 is properly connected and configured, data and command can be sent serially. For data sending, you can send any message such as “Hello” via serial I/O directly, “Hello” message will be shown on your LCD.

For command sending, you can send standard instruction set to LCD (see Figure 8-2) by precede it with the instruction prefix character, ASCII 254 (0FE hex or 11111110 binary). SLCD16x2 treats the byte immediately after prefix as an instruction, then automatically returns to data mode.

An example: To clear screen on LCD, clear instruction is 00000001 binary (or ASCII 1), send [254] and [1] to SLCD16x2 (where parentheses in [ ] symbols mean single bytes set to these values)

COMMAND\DATA BIT	D7	D6	D5	D4	D3	D2	D1	D0
1. Initial LCD	0	0	0	0	0	0	0	0
2. Clear LCD	0	0	0	0	0	0	0	1
3. Returm Home	0	0	0	0	0	0	1	*
4. Entry Mode Setting	0	0	0	0	0	1	I / D	S
5. Display Setting	0	0	0	0	1	D	C	B
6. Shift Display	0	0	0	1	S / C	R / L	*	*
7. Function Setting	0	0	1	*	N	F	*	*
8. Set CGRAM Address	0	1	A5	A4	A3	A2	A1	A0
9. Set DDRAM Address	1	A6	A5	A4	A3	A2	A1	A0

Standard instrction command set summary
(except Initial LCD is addition command.
Initialize make I/D=1, S=0, D=1, C=0, B=0, N=1, F=0, DDRAM Address=00

* Don't care bit

S 0=Automatic cursor shift after byte
1=Cursor not moved
I/D 0=After byte, decrease cursor position
1=After byte, increase cursor position
(when S=1, cursor won't be shifted .)

D 0=Display OFF, 1=Display ON
C 0=Cursor OFF, 1=Cursor ON
B 0=Cursor not blink, 1=Cursor blink

S/C 0=Cursor shift, 1=Display Shift
R/L 0=Left shift, 1=Right shift

N 0=1/8 Duty, 1=1/16 Duty
(not recommend to set this bit, use jumper setting instead)
F 0=5x7 dot size, 1=5x10 dot size

A0 to A7 are CGRAM or DDRAM Address

Serial input timing diagram

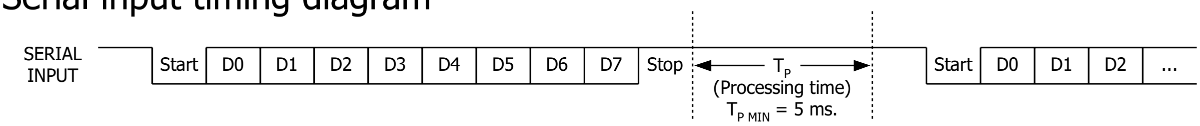
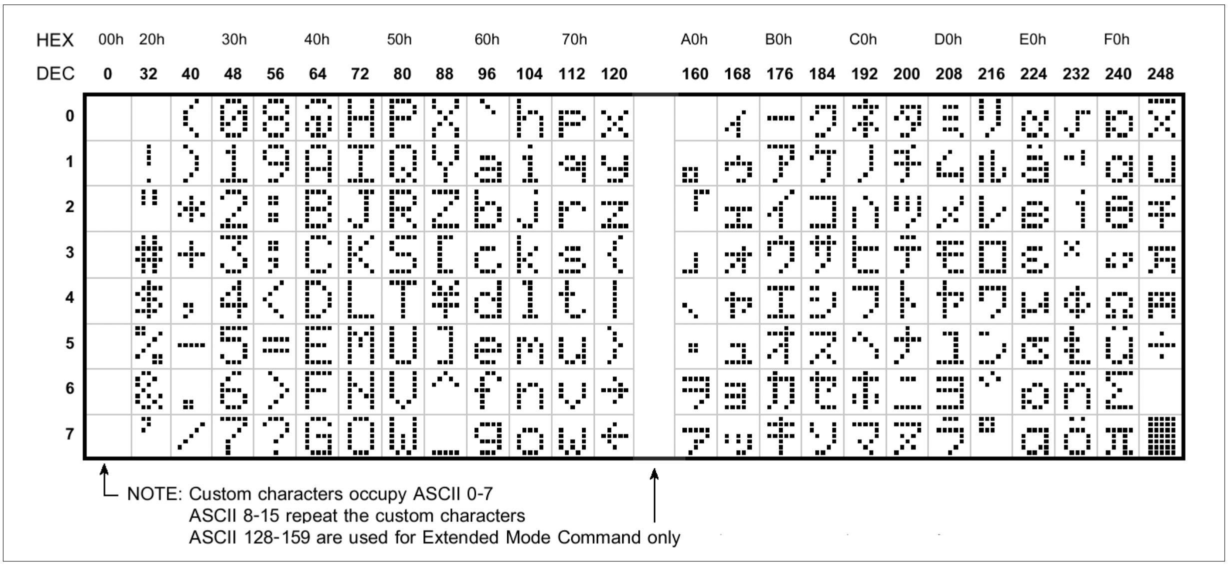


Figure 8-2 SLCD16x2 command summary and timing diagram

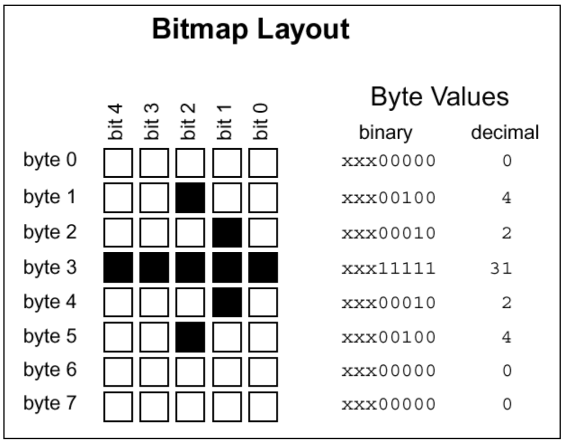
8.1.5 LCD Characters

Most of the LCD characters (Figure E) cannot be changed because they are stored In the ROM. However, the first eight symbols, corresponding to ASCII 0 through 7, are stored in the RAM. By Writing new values to the character-generator RAM (CGRAM), you can alter these characters as you want in 5x8 dots size.



LCD character set. (Built-in character on HD44780A or SED1278F0A)

Create your symbols by pointing to the CGRAM location, then write the first line whose bits form the desired pattern, and point to next CGRAM address to write bits later. Repeat this procedure until 8 times (one character), your character is ready to use now. CGRAM 0 is located on CGRAM Address 00h-07h, CGRAM 1 on 08h-0Fh, CGRAM 2 on 10h-17h, ...until CGRAM 7 on 38h-3Fh. See figure below



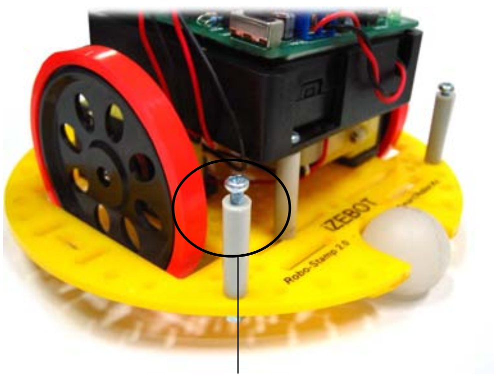
Defining custom symbols.
Example: Load arrow symbol on CGRAM 3, a program would send the following bytes to the SLCD controller.

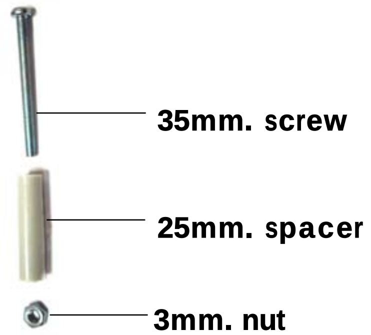
[254] , [01011000 b] , [0] ,
[254] , [01011001 b] , [4] ,
[254] , [01011010 b] , [2] ,
[254] , [01011011 b] , [31] ,
[254] , [01011100 b] , [2] ,
[254] , [01011101 b] , [4] ,
[254] , [01011110 b] , [0] ,
[254] , [01011111 b] , [0]

Activity 3
Installation SLCD16x2

A3.1 Connect the B-end of JST3AB-8 cable to SLCD16x2 input connector. Becareful the connection pin must correct and attach the straight joiner with both holes of SLCD16x2 by 3x10mm. screws and 3mm. nuts.

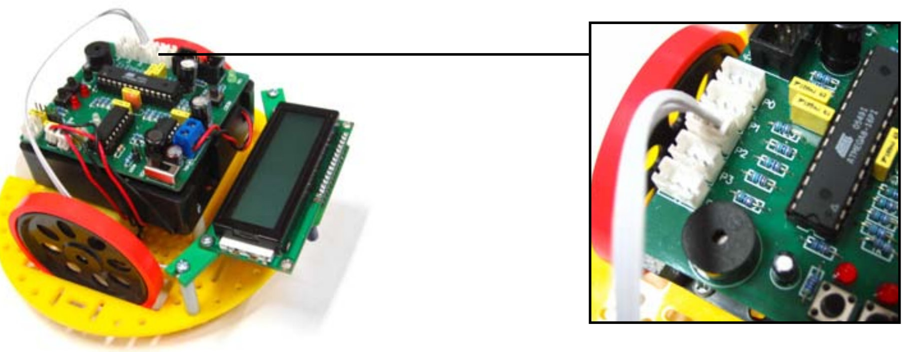
A3.2 Insert 3x35mm. screw through the 25mm. spacer and attach with MicroCamp body with 3mm. nut on the opposite side which attached the Swtiches. Make two sets and do not tight the nut.





Insert screw and tight the nut tempolary

A3.3 Attach SLCD16x2 with the screw and spacer from the previous step by inserted the end of straight joiner between head screw and the top of spacer. After that tighten the screw. Connect the A-end of JST3AB cable to P2 port on the MicroCamp controller board.



8.2 `soft_serout` library

For more comfortable to interface SLCD16x25 with MicroCamp, the serial data communication library is nescessary. The suitable library is soft_serout.h file. This library is serial data transfering from host to any slave serial device in 8N1 format. The full sourcecode of this library is shown in Listing 8-1.

User can select any MicroCamp port (P0 to P4) for interfacing the serial device. This library contains 3 functions as follows.

8.2.1 soft_serout_init()

This function is used for setting Baudrate value. The maximum is 9600 bit per second.

Syntax

```
void soft_serout_init(unsigned long baud_)
```

Example :

```
soft_serout_init(9600);
```

Set the baudrate to 9600 bit per second.

8.2.2 serout_byte()

This function is used for sending one byte data from any port.

Syntax

```
void serout_byte(char tx,unsigned char dat)
```

Thus; `tx` is MicroCamp port. The value is 0 to 4 for P0 to P4 port

`dat` is the sending 8-bit data

Example :

```
serout_byte(2,0x80);
```

Send 0x80 data to P2 port.

8.2.3 `serout_text()`

This function is used to send the string data or many the byte data to the specific port.

Syntax

```
void serout_text(char tx,unsigned char * p)
```

Thus; `tx` is MicroCamp port. The value is 0 to 4 for P0 to P4 port

`* p` is the sending string or data

Example :

```
serout_text(2,"MicroCamp");
```

Send the message “MicroCamp” to P2 port of MicroCamp.

```
#include <avr/io.h>
#include <in_out.h>
#ifndef _soft_serout_
#define _soft_serout_

#define PRESCALER_1  (1<<CS20)                 // (1/16M) 0.0625 us per MC
#define PRESCALER_8  (1<<CS21)                 // (8/16M) 0.5 us per MC
#define PRESCALER_32 (1<<CS21) | (1<<CS20)     // (32/16M) 2 us per MC
#define PRESCALER_64 (1<<CS22)                 // (64/16M) 4 us per MC
#define OFFSET_DELAY1  20                      // for out function used 20 us
#define OFFSET_DELAY2  18                      // for out function used 20 us
unsigned int base=0;
unsigned char base_start_rcv=0;
unsigned char TCCR2_cal=0;
unsigned int base;
unsigned int baud=9600;

void soft_serout_init(unsigned long baud_)     // Config.and Start up timer 2
{
   unsigned long tick=0;
   if(baud_<=4800)
   {
     tick = ((1000000/baud_)-OFFSET_DELAY1)/4;  // Calculate Delay for baudrate
     TCCR2_cal = PRESCALER_64;
   }
   else if(baud_>4800 && baud_<=9600)
   {
     tick = ((1000000/baud_)-OFFSET_DELAY2)/2;  // Calculate Delay for baudrate
     TCCR2_cal = PRESCALER_32;
   }

   TCCR2 = 0;                                   // Stop timer
   TIFR |= 1<<TOV2;                             // Ensure Clear Overflow flag
   base = 255-tick;
   base_start_rcv = 255-(tick/2);
}

// Delay For baudrate

void delay_baud(unsigned int _tick)
{
   TCNT2 = _tick;                    // Load Prescaler form calculate
   TCCR2 = TCCR2_cal;                // Load interval
   while(!(TIFR & (1<<TOV2)));       // Wait until count success
   TIFR |= 1<<TOV2;                  // Ensure Clear Overflow flag
   TCCR2 = 0;                        // Stop timer 2
}

// Send Data  1 Byte
void serout_byte(char tx,unsigned char dat)
{
   int i;
   out_c(tx,0);                      // start bit
   delay_baud(base);                 // Delay for start bit
   for(i=0;i<8;i++)
   {
     out_c(tx,dat & 0x01);           // Send data bit
```

Listing 8-1 The soft_serout library file sourcecode (continue)

```
    delay_baud(base);              // Delay for calculate base
    dat=dat>>1;                    // Shift for next bit
  }
  out_c(tx,1);                     // stop bit
  delay_baud(base);                // Delay for stop bit
}

// Send More Than 1 byte

void serout_text(char tx,unsigned char * p)
{
  while(*p)
  {
    serout_byte(tx,*p++);
  }
}

#endif
```

Listing 8-1 The soft_serout library file sourcecode (final)

Activity 4
SLCD16x2 simple programming

A4.1 Open the AVR Studio to create the new project and write the C program following the Listing A4-1.

A4.2 Add the `soft_serout` library file into the project file. Build this project.

A4.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A4.4 Turn-off power and Remove the ISP cable.

A4.5 Turn-on the power and observe the SLCD16x2 operation.

The SLCD16x2 show message MicroCamp *on the top line.*


```
#include <soft_serout.h>
#include <sleep.h>

void main()                         // Main Program
{
      sleep(1000);                  // Delay 1 Sec
      soft_serout_init(9600);       // Initital 9600 8N1
      serout_text(2,"MicroCamp");   // Send Text "MicroCamp to SLCD
      while(1);                     // Stop
}
```

Program description

This code will refer 2 libraries; soft_serout.h for sending the serial data and sleep.h for delaying. The program operation are :

1. Delay 1 second for SLCD16x2 initial.
2. Set the baudrate. In this program is 9600 bit per second.
3. Send mthe message “MicroCamp” to P2 port to display on the SLCD16x2
4. Loop operation.

Listing A4-1 The simple program for sending the message to display on the SLCD16x2

Activity 5
Control the SLCD16x2 with command

User can control many the display operation of SLCD16x2 such as set the line display, clear screen, select the display format etc. with seding the control command to SLCD16x2. For the Standard command mode, start byte must start with 0xFE and following the command. User can see the LCD command in SLCD information topic in this chapter.

A5.1 Open the AVR Studio to create the new project and write the C program following the Listing A5-1 and build this project.

A5.2 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A5.3 Turn-off power and Remove the ISP cable.

A5.4 Turn-on the power and observe the SLCD16x2 operation.

The SLCD16x2 show many message displaying following the specific by program.

```
#include <soft_serout.h>
#include <sleep.h>

void main()                                      // Main Program
{
   unsigned char i=0;
   while (1)
   {
      sleep(1000);                               // Delay 1 Sec
1     soft_serout_init(2,9600);                  // Initial Serial Comm 9600 8N1
      serout_byte(2,0xFE);serout_byte(2,0x00);   // Command Initial LCD Module
---------------------------------------------------------------------------------
      serout_byte(2,0xFE);serout_byte(2,0x80);   // Command First Line ,First Char
      serout_text(2,"MicroCamp");                // Show Text "MicroCamp"
2     serout_byte(2,0xFE);serout_byte(2,0xC0);   // Command Second Line, Fisrt Char
      serout_text(2,"Microcontroller");          // Show Text "Microcontroller"
      sleep(2000);
---------------------------------------------------------------------------------
      serout_byte(2,0xFE);serout_byte(2,0x01);   // Command Clear Screen
      serout_byte(2,0xFE);serout_byte(2,0x85);   // Command First Line, 5'th Char
3     serout_text(2,"From");                     // Show Test "From"
      sleep(500);
---------------------------------------------------------------------------------
      serout_byte(2,0xFE);serout_byte(2,0x07);   // Shift Text Left and Inc Address
      for (i=0;i<9;i++)                          // 9 Time Loop
      {
4        serout_byte(2,0x20);                    // <<<<<<  Shift Data Left
         sleep(200);
      }
---------------------------------------------------------------------------------
      serout_byte(2,0xFE);serout_byte(2,0x05);   // Shift Text Right and Inc Address
      for (i=0;i<9;i++)
      {
5        serout_byte(2,0x20);                    // Shift Data Right 9 Time
         sleep(200);
      }
---------------------------------------------------------------------------------
      for (i=0;i<9;i++)                          // Blinking 9 Time
      {
         serout_byte(2,0xFE);serout_byte(2,0x08); // Display Off
6        sleep(200);
         serout_byte(2,0xFE);serout_byte(2,0x0C); // Display ON
         sleep(200);
      }
---------------------------------------------------------------------------------
      serout_byte(2,0xFE);serout_byte(2,0x00);   // Command Initial LCD
7     serout_byte(2,0xFE);serout_byte(2,0x80);   // Command First Line, First Char
      serout_text(2,"Innovative");               // Show Text "Innovative "
      serout_byte(2,0xFE);serout_byte(2,0xC0);   // Commmand Second Line, First Char
      serout_text(2,"Experiment");               // Show Text "Experiment"
   }
}
---------------------------------------------------------------------------------
```

Listing A5-1 The experiment code about testing the SLCD16x2 display in many formats. Do not type the broken line in the listing because it is used for program description reference (continue)

Program description

Part 1 Initial the communication module in microcontroller and SLCD16x2

Part 2 Select the target line to display. The top line (0x80) is set to show MicroCamp message. The bottom line (0xC0) is set to show Microcontroller message.

Part 3 Send the Clear screen command (0x01) and defind the first letter at 5th digit on the top line of LCD (0x85) to show From message.

Part 4 Send the shift left command (0x07) and loop to shift the From message to left direction.

Part 5 Send the shift right command (0x05) and loop to shift the From message back to start.

Part 6 Loop for sending the Turn-off display command (0x08) and Turn-on display command (0x0C) and swap. It cause the message From will be blink.

Part 7 The opearion is same Part 2 but chage the message on top line as Innovative and bottom line as Experiment

Listing A5-1 The experiment code about testing the SLCD16x2 display in many formats. (final)

Chapter 9
MicroCamp robot with Line tracking activities

From the activities in chapter 7, these show how to read the digital input signals and data to control the robot's movement. In this activity, there will be many activities about reading analog inputs and processing the data for the detection of black and white areas. It also detects Black and white line to control the robot to move along the line with variable conditions.

The MicroCamp robot has 5 analog inputs that directly connects to the PC0 to PC4 of ATmega8 microcontroller. This microcontroller contains the 10-bit analog to digital converter (ADC) module. The digital conversion data is 0 to 1,023 in decimal number format.

C programming for this activity requires a library file. This is the **analog.h** file. Functions in this library will define relate the input port to the analog input and reads data from ADC module to store in its memory. The resulting data range is 0 to 1,023 in decimals or 0000H to 03FFH in hexadecimals.

The important devices in this activity is the 2 Infrared Reflector modules. They are installed at the bottom of the robot base. They are used to detect the surface's color (black and white) including the white and black line. The Line tracking robot activity is the classic activity. It shows the basic robot's programming performance.

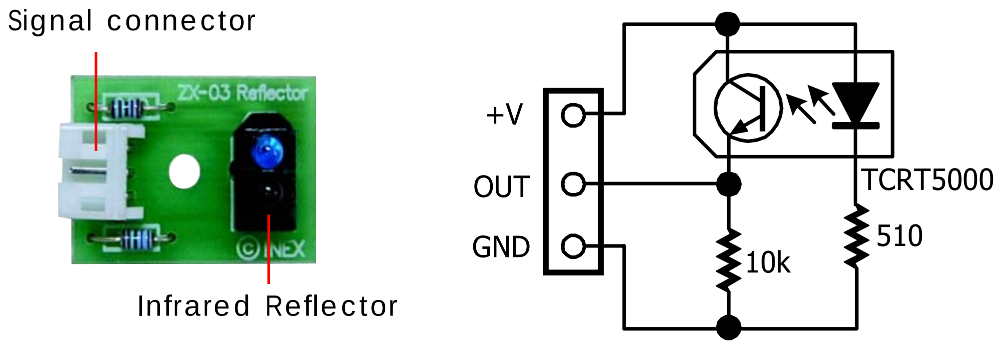


Figure 9-1 : ZX-03 Infrared Reflector information

9.1 ZX-03 Infrared Reflector

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 from10-bit A/D converter is 20 to 1,000. Thus, ZX-03 will suitable to apply to line tracking sensor.

9.2 `analog` : Read analog signal function of analog.h library

This function is suitable for reading the analog data from PC0 to PC5 pin. These ports are set to analog input. The analog to digital converter resolution is 10-bit. The conversion result is during 0 to 1,023 for scaling voltage 0 to 5V.

Syntax

```
unsigned int analog(unsigned char channel)
```

Parameter

`channel` : Select the required analog input. The value is 0 to 4 for P0 to P4 port of MicroCamp.

Return value

0 to 1023 from the analog to digital converter module within microcontroller.

Activity 6
Testing black and white area

The MicroCamp robot is attached with 2 of Infrared Reflector module at bottom of the robot base ready. Thus, this activity will only dwell on programming.

Before develop the robot to track the line, developers must program the robot to detect the difference between black and white surface.

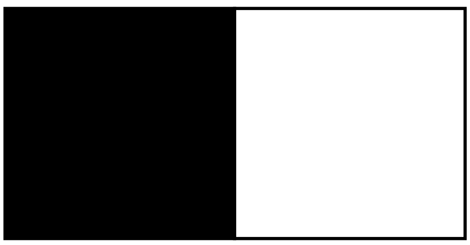
A6.1 Open the AVR Studio to create the new project and write the C program following the Listing A6-1.

A6.2 Add the `analog` library file into the project file. Build this project.

A6.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A6.4 Turn-off power and Remove the ISP cable.

A6.5 Make the black & white testing sheet. The white surface area is 30 x 30 cm. and black surface is 30 x 30cm as shown below.



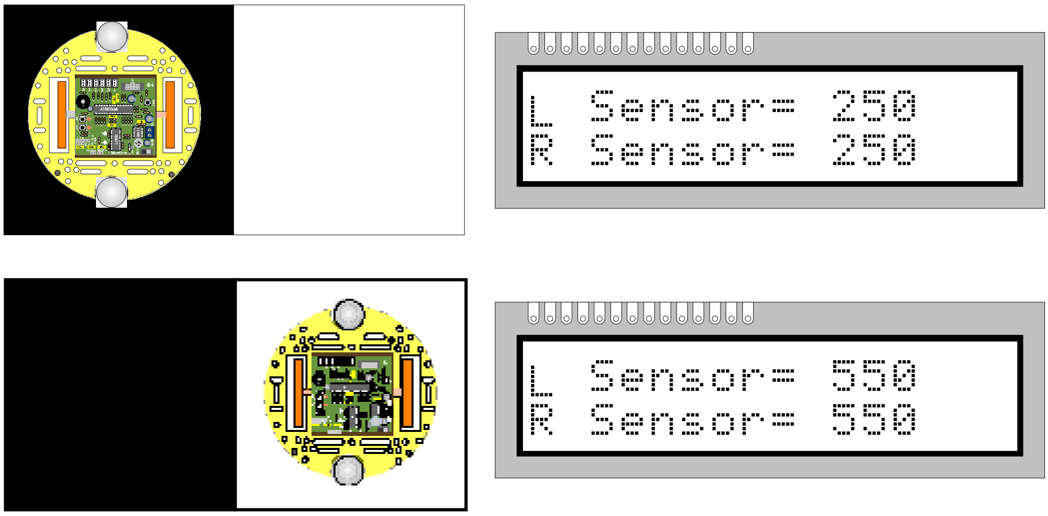
```
#include <stdlib.h>          // For convert type of data
#include <soft_serout.h>     // Software serial communication For LCD
#include <analog.h>
#include <motor.h>
#include <sleep.h>
void lcd_command(unsigned char pin,unsigned char command)
{
  serout_byte(pin,0xFE);serout_byte(pin,command);
}
int main()
{
  unsigned int l_sensor=0,r_sensor=0;
  unsigned char dec1[4],dec2[4];
  sleep(1000);                        // Delay 1 Sec
  soft_serout_init(2,9600);           // Initial Serial Comm 9600 8N1
  while(1)
  {
    l_sensor = analog(3);             // Read Left Sensor
    r_sensor = analog(4);             // Read Right Sensor
    utoa(l_sensor,dec1,10);           // Convert Integer to decimal Ascii
    utoa(r_sensor,dec2,10);           // Convert Integer to decimal Ascii
    lcd_command(2,0x80);              // Command First Line
    serout_text(2,"L Sensor=       ");
    lcd_command(2,0x8A);              // Command First Line
    serout_text(2,dec1);              // Show Ascii data
    lcd_command(2,0xC0);              // Command First Line
    serout_text(2,"R Sensor=       ");
    lcd_command(2,0xCA);              // Command First Line
    serout_text(2,dec2);              // show centimeter data
    sleep(300);
  }
}
```

Program description

The ATmega8 microcntroller in MicroCamp robot would read the result data from the A/D converter module that get from both Infrared reflector sensors at P3 and P4 port. The data is displayed on the SLCD16x2 screen.

Listing A6-1 : The program code for testing black and white area.

A6.6 Place the MicroCamp robot that programmed ready from step A6.4 above the white surface of the testing chart. Turn on the robot. See the reading value at SLCD screen and record it. After that change to read value of black surface and record the value also.



The result is :

The white surface value is during 600 to 950

The black surface value is during 100 to 300

The example reference value for detecting the line is (600+100) /2 = 350.

Activity 7
Robot moves along the black line

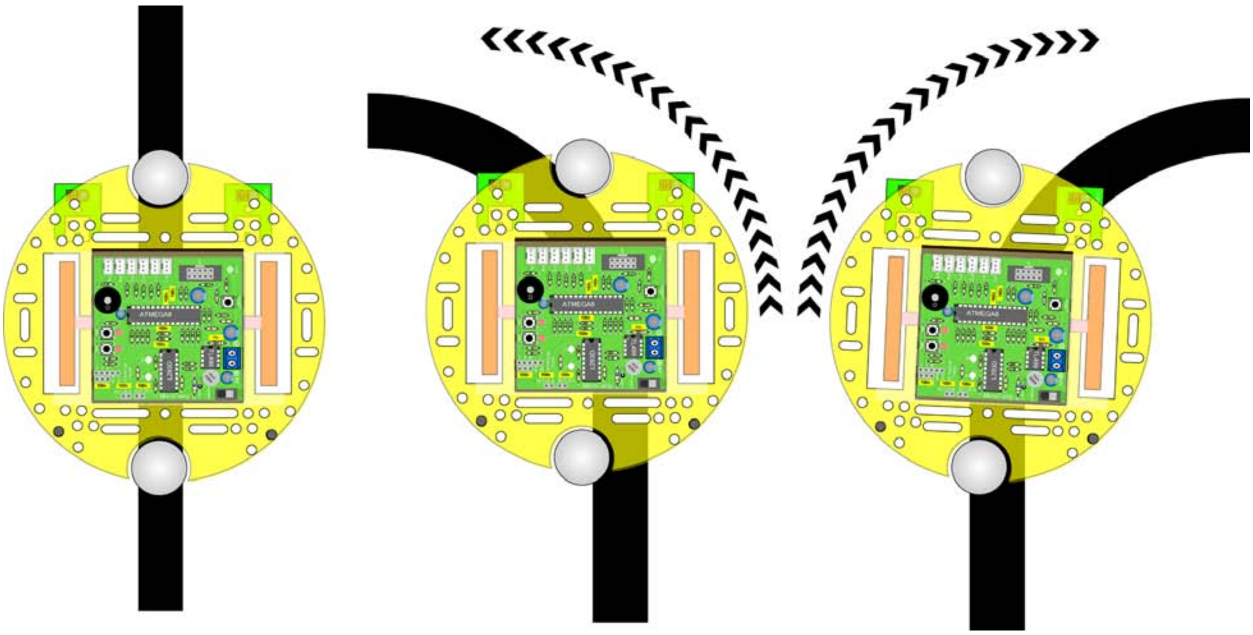
The robot moving along the line has 3 scenarios.

(1) Both sensors read values that are white : The robot will move forward. Thus, this program is written so that the robot moves forward normally.

(2) The left sensor reads black while the right sensor reads white : This occurs when the robot is slightly turned to the right. Thus, the program is written for the robot to move back left to resume its normal path.

(3) The left sensor read white while the right sensor reads black : This occurs when the robot is slightly turned to the left. Thus, the program is written for the robot to move back to the right to resume its normal path.

From all scenarios,, you can make the C program as follows in the listing A7-1



Scenario-1
Both sensors put across

Scenario-2
Left sensor detects the line

Scenario-3
Right sensor detects the line

```
#include <in_out.h>
#include <sound.h>
#include <analog.h>
#include <motor.h>                               // Motor diver library
unsigned int AD0=350,AD=350;                     // Determine the sensor reference
                                                 // value.
void main()
{
   while((in_d(2)==1));                          // Wait SW1 pressing to start the
                                                 // program
   while(1)
   {
     if((analog(3)>AD0)&&(analog(4)>AD1))        // Both sensor detect the white
                                                 // surface
        forward(100);                            // Move forward
     if (analog(0)<AD0)                          //  Left sensor detects black line.
        s_left(100);                             // Turn left
     if (analog(1)<AD1)                          // Right sensor detects black line.
        s_right(100);                            // Turn right
   }
}
```

Listing A7-1 : The C program code for controlling the MicroCamp robot to move along a black line.

A7.1 Make a simple black path sheet. You can use any conventional black insulating tape on a white cardboard paper. Leave most of the surface area white with only the black line significant. You may use a standard black marker and white paper as well.

A7.2 Create the new project file and make the C program following the Listing A7-1. Build this project file.

A7.3 Connect the PX-400 programmer box with MicroCamp robot and download the HEX code to the robot. Turn off power and unplug ISP cable from the robot.

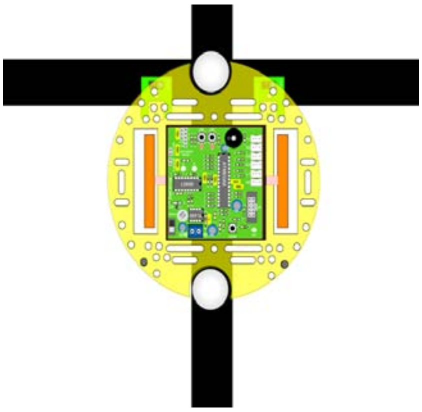
A7.4 Place the robot across the black line on the sheet. Turn on power and press SW1 switch.

The Microcamp Robot will move along the black line. It is possible that the robot moves out of the line. You can improve the precision by editing the program with adjusting the sensor reference value and adjust to the position of both the Infrared Reflector sensors.

Activity 8
Line crossing detection

From the activity 7, you can improve the MicroCamp robot so that it moves along the black line and detects the junction or line with the same 2 sensors. All you have to do is to edit your program code.

When the robot moves to the black line T junction , both sensors will detect the black line. You must add the program for support this scenario. The improved C program is shown in the Listing A8-1.



A8.1 Improve the simple black line sheet from Activity 7. Add some cross lines. Add as many junctions as you like. However, make sure that they are at least 2 robots width apart.

A8.2 Create the new project file and make the C program following the Listing A8-1. Build this project file.

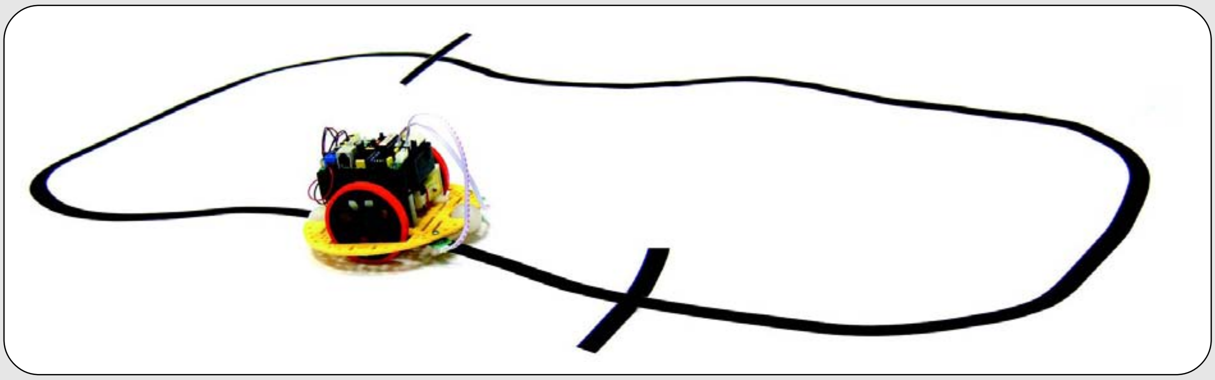
A8.3 Connect the PX-400 programmer box with MicroCamp robot and download the HEX code to the robot. Turn off power and unplug ISP cable from the robot.

A8.4 Place the robot across the black line on the sheet. Turn on power and press SW1 switch.

The Robot will move along the black line. When the robot detects the crossing, it will brake and drive sound once. When it finds the second crossing, the robot will drive sound twice and this will increase for the subsequent crossings.

Note : In the motor brake operation, robot will stop and lock the motor's shaft immediately. But sometimes, this is not enough. You must program the robot to move backwards for a short time. This will cause the robot to stop at its position.

```
#include <in_out.h>
#include <sound.h>
#include <analog.h>
#include <motor.h>                              // Motor control library
unsigned int AD0=350,AD1=350;                   // Sensor reference value
unsigned char i=0,j=0;                          // Crossing counter variable
void main()
{
   while((in_d(2)==1));                         // Wait for SW1 to be pressed to
                                                // start
     while(1)
   {
     if((analog(3)<AD0)&&(analog(4)<AD1))       // Detect line-crossing
     {
        j++;
        backward(30);                           // Move backward for a short time
                                                // to brake.
        sleep(10);
        motor_stop(ALL);                        // Motor brake function
        for  (i=0;i<j;i++)                      // Repeat the loop of crossing detection
        {
           sound(2500,100);
           sleep(50);
        }                                       // Drive sound
        forward(100);                           // Move forward to closs over
        sleep(300);                             // the line.
     }
     if((analog(3)>AD0)&&(analog(4)>AD1))       // Both sensor detect white surface.
        forward(100);                           // Move forward
     if  (analog(3)<AD0)                        // Left sensor detects black line.
        s_left(100);                            // Turn left
     if  (analog(4)<AD1)                        // Right sensor detects black line.
        s_right(100);                           // Turn right
   }
}
```



Listing A8-1 The C program code for controlling the MicroCamp robot to moves along the black line and detect the line-crossing.

Chapter 10
MicroCamp robot with IR Ranger Capability

One of the special sensors in robotics is the. It is an Infrared Distance sensor. Some people call it the IR Ranger. With the GP2D120 module, it adds the distance measuring and Obstacle detection using infrared light feature to your robot. Your MicroCamp robot can avoid obstacles without having to make any physical contact.

10.1 GP2D120 features

- Uses Infrared light reflection to measure range
- Can measure a range from 4 to 30 cm.
- 4. 5 to 5 V power supply and 33mA electric current
- The output voltage range is 0.4 to 2.4V when supplied by +5V

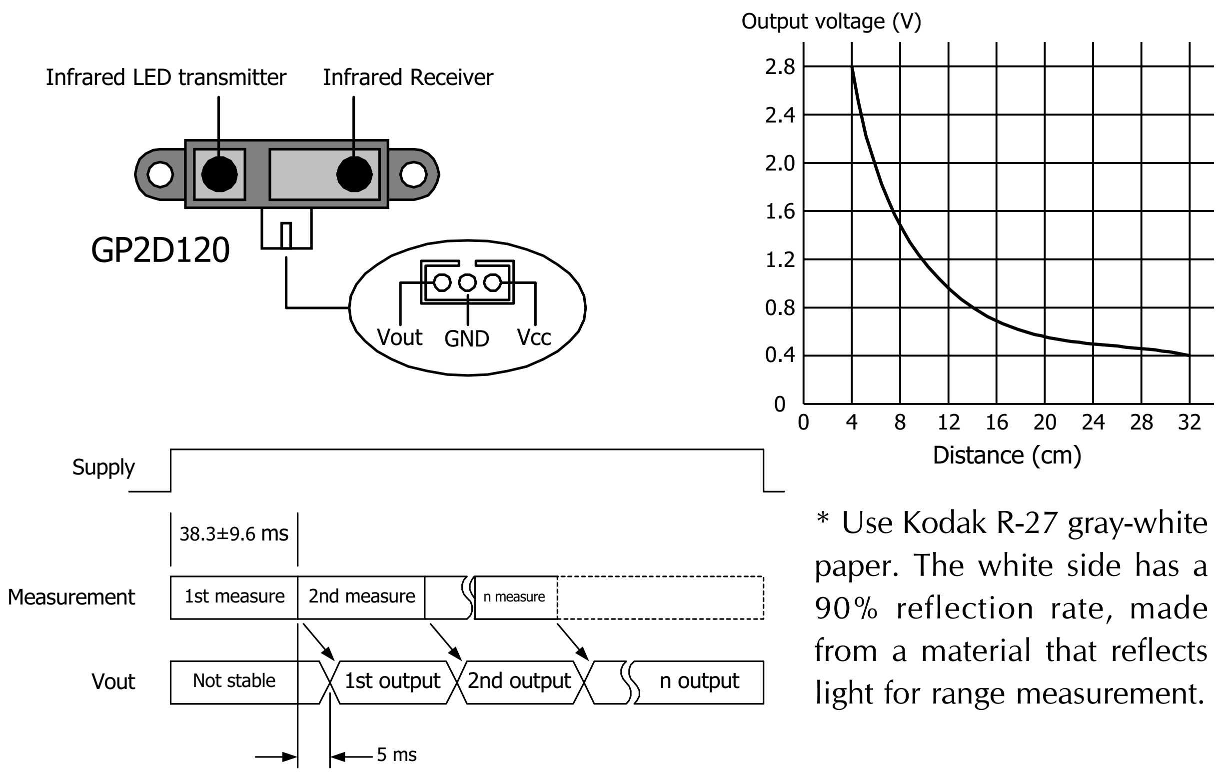


Figure 10-1 : GP2D120 pin assignment, operation and characteristic curve

** GP2D120 module is optional component for MicroCamp2.0 Beginner kit and bundled in Standard kit.*

GP2D120 Infrared Ranger module has 3 terminals: Power input (Vcc), Ground (GND) and Voltage output (Vout). To read the voltage values from the GP2D120, you must wait till after the acknowledgement period which is around 32 to 52.9 ms.

The output voltage of GP2D120 at a range of 30 cm and +5V power supply is between 0.25 to 0.55V, with the mean being 0.4V. At the range of 4 cm., the output voltage will change at 2.25V± 0.3V.

10.2 How the IR Ranger Module works

Measuring range can be done in many ways. The easiest to understand is through ultra sonic where sound waves are sent to the object and the time it takes to reflect back is measured. This is because sounds waves do not travel fast, and can be measured by present day equipment. However, in the case of infrared light, the time it takes to hit an obstacle and reflect back can not be measured because infrared light travels fast. No measurement equipment is available yet. Therefore, the following theory must be used.

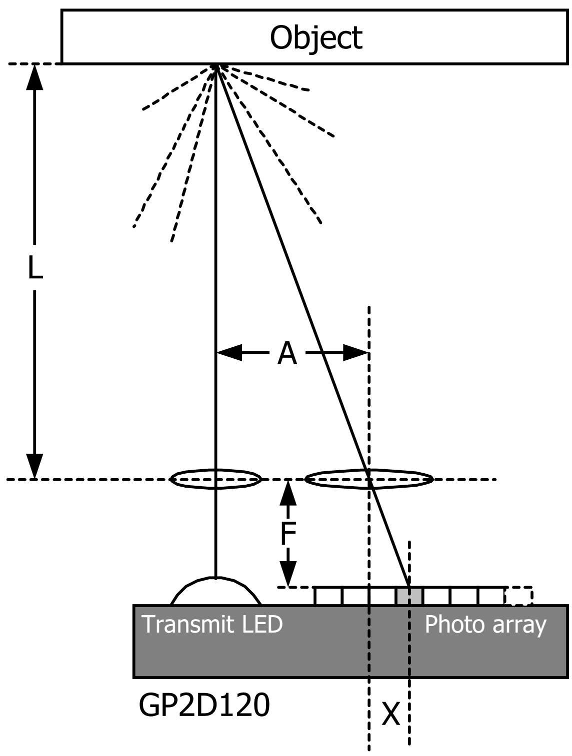
The infrared light is sent out from a transmitter to the object in front, by passing through a condense lens so that the light intensity is focused on a certain point. Refraction occurs once the light hits the surface of the object. Part of the refracted light will be sent back to the receiver end, in which another lens will combine these lights and. determine the point of impact. The light will then be passed on to an array of photo-transistors. The position in which the light falls can be used to calculate the distance (L) from the transmitter to the obstacle using the following formula:

$$\frac{L}{A} = \frac{F}{X}$$

Therefore, L equals

$$L = \frac{F \times A}{X}$$

Thus, the distance value from the phototransistors will be sent to the Signal Evaluation Module before it is changed to voltage, resulting in a change of voltage according to the measured distance.



10.3 Reading GP2D120 with A/D converter

The GP2D120's output voltage will change acoording to the detection distance. For example, Vout 0.5V is equal 26cm. distance and Vout 2V is equal 6cm. distance. The table 10-1 shows the summary of GP2D120's Vout and Distance relation.

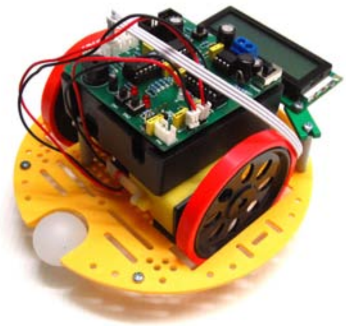
For interfacing with A/D converter module within microcontroller, the result is raw data from the A/D conversion. The user will need to use the software to convert the raw data to the exact distance. For example, see the Table10-1. The raw data from conversion is 307. It is equal 8cm. distance.

GP2D120's Vout	Raw data	Distance
0.4	82	33
0.5	102	26
0.6	123	22
0.7	143	19
0.8	164	16
0.9	184	14
1.0	205	13
1.1	225	12
1.2	246	11
1.3	266	10
1.4	287	9
1.5	307	8
1.6	328	8
1.7	348	7
1.8	369	7
1.9	389	6
2.0	410	6
2.1	430	6
2.2	451	5
2.3	471	5
2.4	492	5
2.5	512	5
2.6	532	4

Table 10-1 : Shows the relation about the GP2D120's voltage output, A/D conversion raw data and the calculation distance.

Activity 9
Installation GP2D120 with MicroCamp

Part list



MicroCamp that install SLCD ready

GP2D120 module (bundled in MicroCamp2.0 Standard kit)

Angled joiner	x 2
Obtuse joiner	x 2
Straight joiner	x 4
3x10mm. screw	x 4
3mm. nut	x 4

Warning for the signal cable of the GP2D120

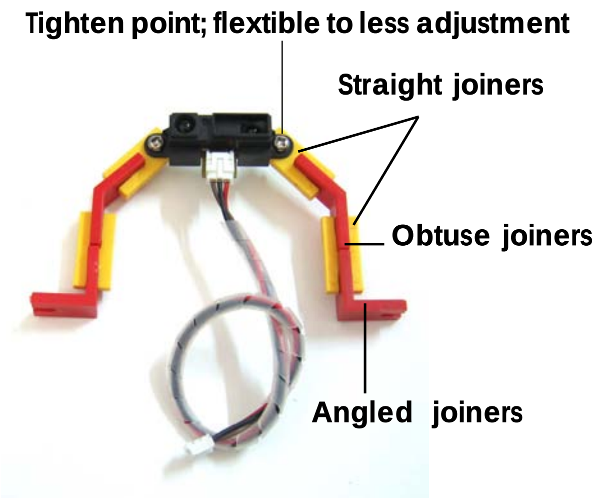
The GP2D120 module has a different pin arrangement then that of the MicroCamp board, even though it looks similar. Therefore, a special signal cable has already been connected to the GP2D120 module. The user just needs to connect the other end of the cable to the connection points of the MicroCamp board. DO NOT remove the cable from the module, and do not replace it with signal cables from other sensor modules.

A9.1 Screw in 2 of 3 x 10mm. machine screws into the installation slot on the GD2D120 module loosely followed by 3mm. nuts. Do not tighten.

A9.2 Place the straight joiner between the screw and nut, and tighten the screw. (Leave it loose enough to change the angle)

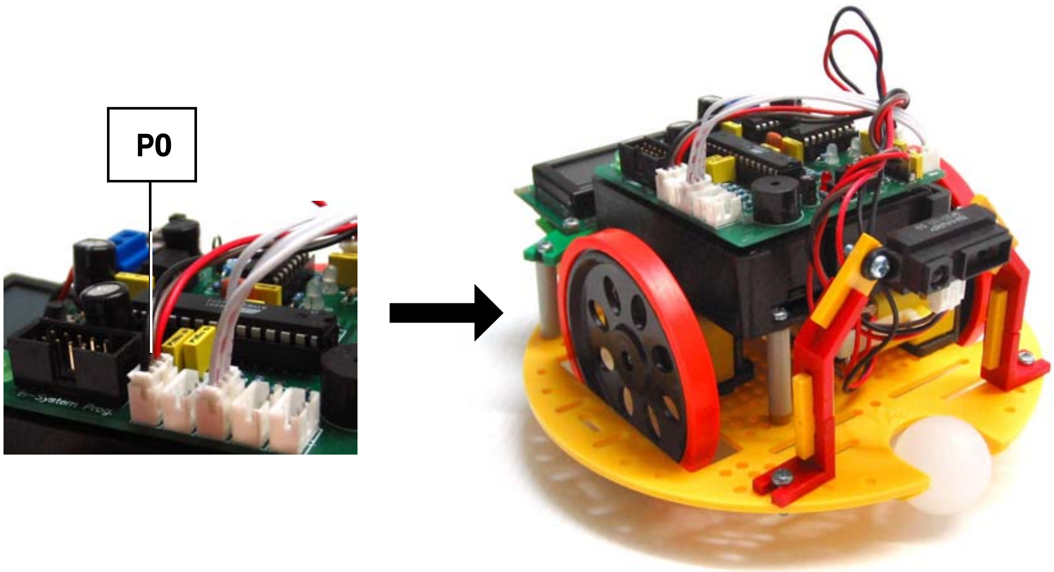
A9.3 Attach an obtuse joiner to the other end of the straight joiner. Repeat for both sides.

A9.4 Attach a straight joiner to the other end of the obtuse joiner. Then place an angled joiner to the other end of the straight joiner.



A9.5 Remove the Switch's structure at the front of robot first. After that, install the GP2D120 structure instread by use a 3 x 10mm. machine screw to screw it in loosely with a 3mm. nut and on the right and left sides.

A9.6 Connect the GPD120's cable to P0 port of MicroCamp robot as seen in the picture below.



Activity 10
Measure distance with GP2D120

The MicroCamp robot can read the data from GP2D120 with **analog()**function in C programming. To convert the A/D conversion raw data to the distance value, the suitable formula will be used as follows :

R = (2933 / (V + 20)) - 1

Thus; R as Distance in Centimetre unit

V as the raw data from A/D conversion. The range is 0 to 1,023.

A10.1 Open the AVR Studio to create the new project and write the C program following the Listing A10-1.

A10.2 Add the `analog` library file into the project file. Build this project.

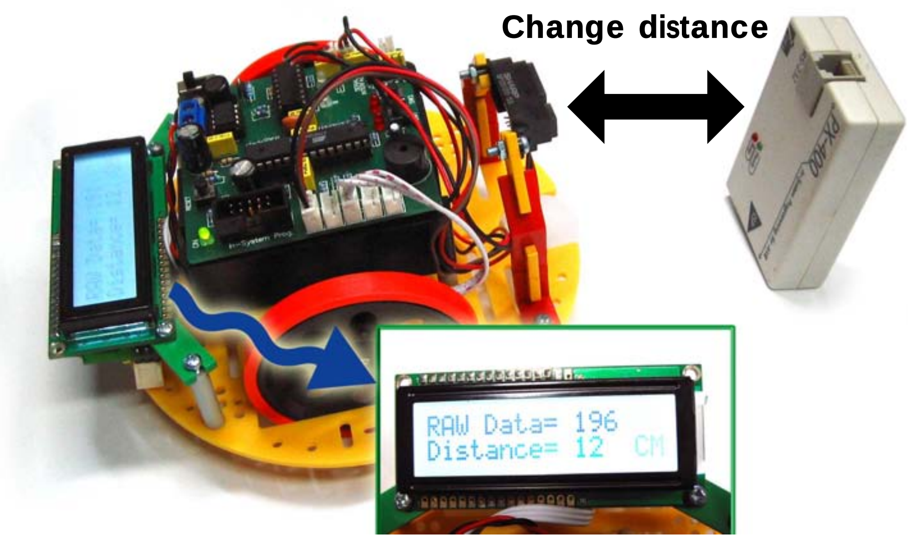
A10.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A10.4 Turn-off power and Remove the ISP cable.

A10.5 Turn-on the power and place some object at the front of GP2D120 module. Observe the SLCD16x2 operation.

A10.6 Adjust the distance of the object from MicroCamp robot and observe the result.

From testing, you will found the GP2D120 can detect the object in range 4 to 30cm. correctly.



```
#include <stdlib.h>           // For CONVERT type of data
#include <soft_serout.h>      // Software serial communication
#include <sleep.h>            // Delay
#include <analog.h>           // Analog to Digital Converter
#define m 2933                // Contance for convert data to centimeters.
#define b 20
#define k 1
void main()                   // Main Program
{
   unsigned char dec[4],dec2[4];    // for save ascii after convertion
   unsigned int gp2=0,cm=0;         // Read data from adc
   sleep(1000);                     // Delay 1 Sec
   soft_serout_init(2,9600);        // Initial Serial Comm 9600 8N1 For SLCD

   while(1)
   {
      gp2   = analog(0);            // Read
      cm = (m/(gp2+b)) - k;         // Convert Data to Centimeter
      utoa(gp2,dec,10);             // Convert Integer to decimal Ascii
      utoa(cm,dec2,10);             // Convert Integer to decimal Ascii
      serout_byte(2,0xFE);serout_byte(2,0x00);     // Command Clear Screen
      serout_byte(2,0xFE);serout_byte(2,0x80);     // Command First Line
      serout_text(2,"RAW Data= ");
      serout_text(2,dec);                          // Show Ascii data
      serout_byte(2,0xFE);serout_byte(2,0xC0);     // Command Second Line
      serout_text(2,"Distance= ");
      serout_text(2,dec2);                         // show centimeter data
      serout_byte(2,0xFE);serout_byte(2,0xCE);     // Command 2nd line
                                                   // position 14
      serout_text(2,"CM");
      sleep(500);
   }
}
```

Program description

(1) Intitial the serial data communication and SLCD16x2.

(2) Loop to read the analog data at P0 port.

(3) Convert the raw data to Centimetre unit by `cm = (m/(gp2+b)) - k;` formula.

(4) Covert the calculation data to ASCII for displaying at SLCD16x2.

(5) Loop the operaition every 0.5 second.

Listing A10-1 : The C program for reading the GP2D120's raw data and convert to distance unit and display on the SLCD16x2

Activity 11

Non-Contact object avoidance Robot

A11.1 Open the AVR Studio to create the new project and write the C program following the Listing A11-1.

A11.2 Add the `analog` library file into the project file. Build this project.

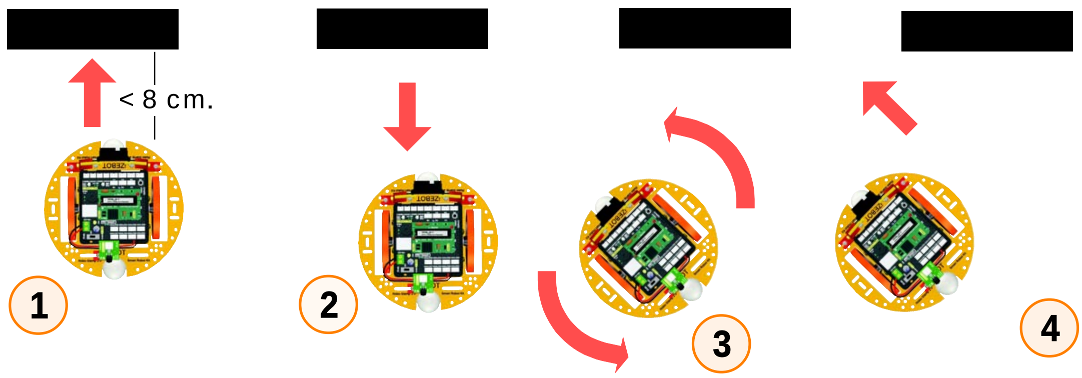
A11.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot (must install GP2D120 and SLCD ready) at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A11.4 Turn-off power and Remove the ISP cable.

A11.5 Place the robot on the floor. Turn-on the power and observe its operation.

A11.6 Try to place any object at the front of the robot and see its operation.

The robot will check the distance of the object in 8cm. range. If not any obstacle, robot will move forward continue. If found the object, it will move backward, turn left and move forward again.



```
#include <stdlib.h>                    // For convert type of data
#include <motor.h>                     // Motor control
#include <sleep.h>                     // Delay
#include <sound.h>
#include <analog.h>                    // Analog to Digital Converter

void main()                            // Main Program
{
  unsigned int sensor=0;
  unsigned char i=0;
  sleep(200);sound(4000,50);           // start with beep
  while(1)
  {
    sensor=0;
    for (i=0;i<5;i++)
    {
      sensor=(sensor+analog(0));       // Read GP2D120 5 Times
    }
    sensor=(sensor/5);                 // Average Data
    if (sensor>260)                    // data more than 10 CM ?
    {
      backward(50);sleep(800);         // away from object
      s_left(50);sleep(600);
    }
    else
    {
      forward(50);                     // do not any object "Let'go"
    }
  }
}
```

Program description

(1) Start with beep a sound.

(2) Read the data from GP2D120 and store into sensor variable. Read 5 times to calculate the average for more precision.

(3) Check the value of sensor more than 320 or not. If more than, it means the obstacle is far from the robot less than 8cm. Control the robot to move backward 0.8 second and turn left 0.6 second. The speed is set at 50%. You can change to any value fro more suitable operation.

(4) If the detect value is less than 320, the robot still move forward.

(5) Loop the opertion.

Listing A11-1 : The C program code for controlling the MicroCamp robot to avoid the obstacles without any physical contact with the application of GP2D120 module.

Chapter 11
MicroCamp robot with Remote control

In chapter 8, we introduced the simple serial communication activity by using the SLCD module. It is a sending activity only. This chapter is different. The new component is introduced as the Infrared remote control ER-4. This remote control works with the serial communication. The button data will be modulated with 38kHz carrier frequency. The MicroCamp robot must connect the 38kHz Infrared receiver module to demodulate and get the serial data to processing late

You can use the ER-4 remote control to control the robot's movement. Some activity in this chapter will show this operation.

11.1 ER-4 : 4-ch. Infrared 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)

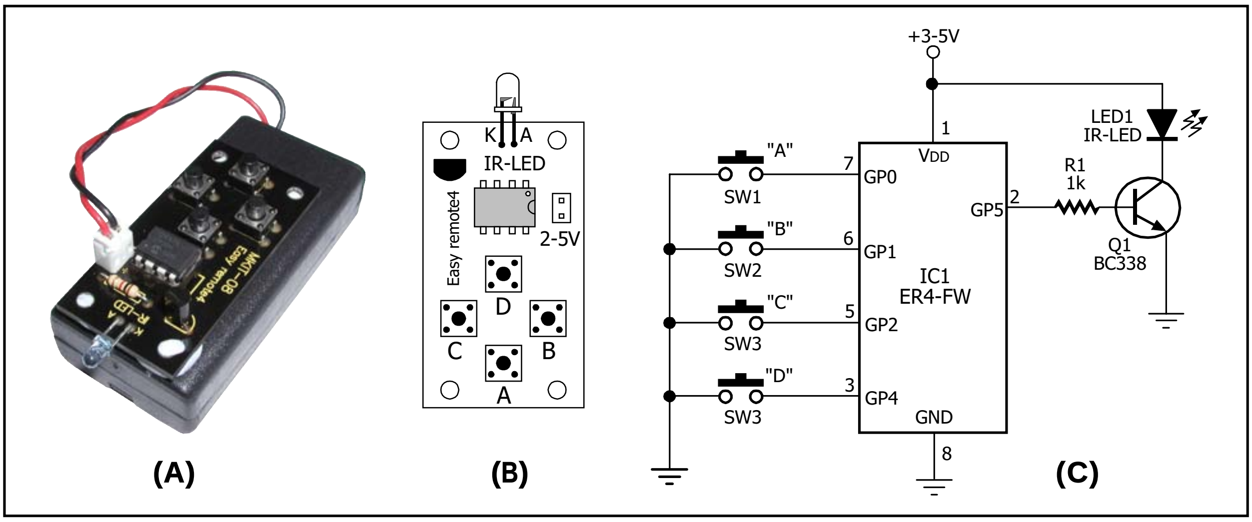


Figure 11-1 : Shows the photo, board layout and Schematic of ER-4 remote control

** ER-4 remote control and ZX-IRM 38kHz Infrared receiver module are optional componnets of MicroCamp2.0 Beginner kit and bundled in Standard kit.*

11.1.1 Format of data sent by Easy Remote4

To make it easier for the receiver to read the switch value from the remote control, the ER-4 transmit serial data is 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 11-1

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.

11.2 ZX-IRM : Infrared Receiver module

In transmitting the data modulated with infrared light for long distance is about 5 to 10 meters similar TV remote control. The carrier frequency is 38kHz.Thus, the 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”.

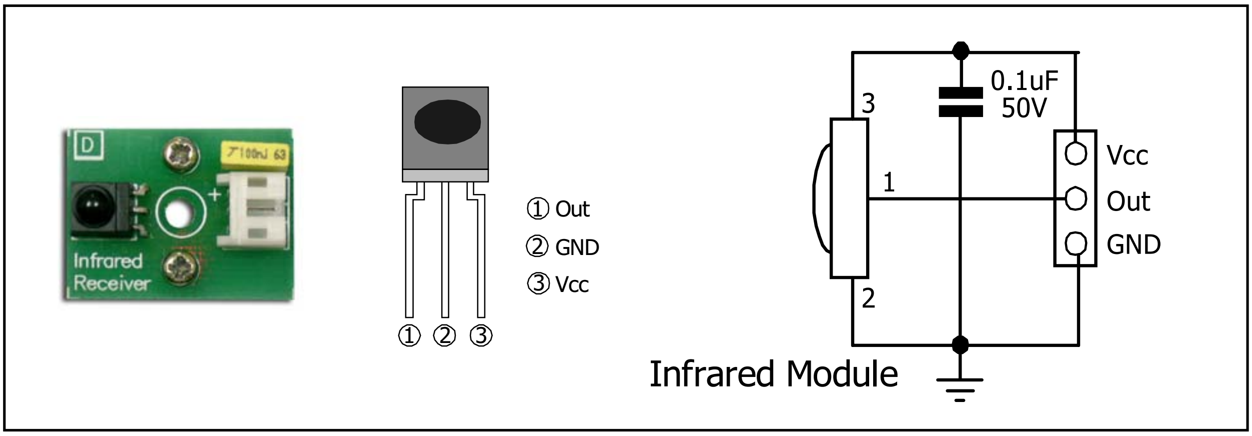


Figure 11-2 : Shows the photo of 38kHz Infrared Receiver module, pin assignment and schematic diagram

Activity 12

Installation 38kHz Infrared receiver module

Part list

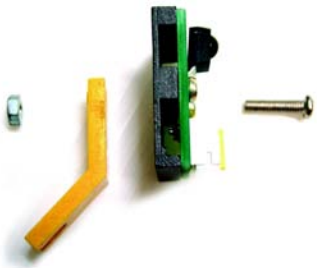
3x10mm. screw and 3mm. nut x 2

Angled joiner x 1

Obtuse joiner x 1

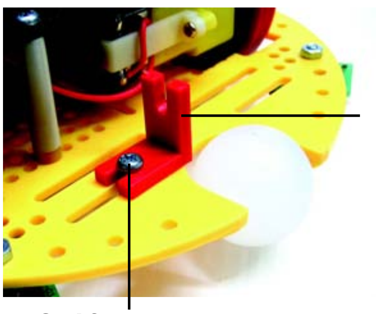
ZX-IRM 38kHz Infrared Receiver module x 1

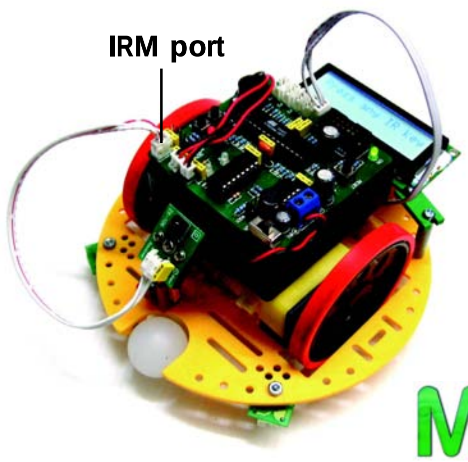
A12.1 Insert a 3x10mm screw through the ZX-IRM and the Obtuse joiner. Tighten with 3mm. nut.



A12.2 Fix the Angled joiner at the front of the MicroCamp robot opposite side of SLCD installed with 3x10mm. screw. However the position could be change to the best location for receiveing the infrared light from ER-4 remote control.

A12.3 Insert the ZX-IRM structure from step A12.1 with the end of the right angle joiner of step A12.2. Connect the ZX-IRM cable to IRM port on the MicroCamp controller board following the photo below.





11.3 Serial data receiving function

C programming for interfacing the 38kHz Infrared Receiver module requires a specific library. It is serial.h file. This library file contains the function about receive and transmit data via UART module within ATmega8 microcontroller.

The functions consist of `uart_set_baud()` and `uart_get_key()`. The detail can describe as follows.

11.3.1 uart_set_baud();

It is baudrate setting function for UART module of ATmega8 microcontroller.

Syntax

```
void uart_set_baud(unsigned int baud)
```

For working with ER-4 Remote control, must set the baudrate as 1200 bit per second. The example of this function is :

```
uart_set_baud(1200);
```

11.3.2 uart_get_key();

It is 1-byte receiving data function at RxD pin of ATmega8 microcontroller. User can set the waiting time at **timeout** parameter. If set to 20,000, the timeout is 30ms.

Syntax

```
char uart_getkey(unsigned int timeout)
```

Example

```
key=uart_getkey(20000);
```

Get data 1-byte and store to key variable.

Activity 13

Getting data from ER-4 Remote control

A12.1 Open the AVR Studio to create the new project and write the C program following the Listing A12-1.

A12.2 Add the `serial.h` library file into the project file. Build this project.

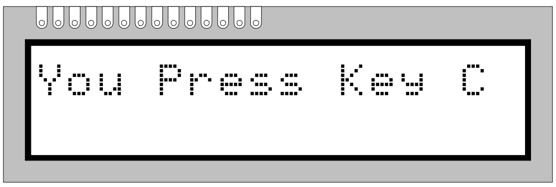
A12.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot (must install the ZX-IRM and SLCD ready) at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A12.4 Turn-off power and Remove the ISP cable.

A12.5 Put the 2 of AA batteries inot Batter holder of ER-4 Remote control.

A12.6 Turn-on power. Press the button switch on ER-4 Remote control to send data to ZX-IRM at the MicroCamp robot. Observe the operation at SLCD screen.

If all correct, SLCD screen would be show the button data that received.



```
#include <stdlib.h>                          // Standard Library
#include <soft_serout.h>                     // For SLCD Module
#include <sleep.h>                           // For Delay
#include <serial.h>                          // For Receiving Data from Remote Control
#include <sound.h>                           // For Generate Sound
#include <motor.h>                           // For Control Motor

unsigned char key,flag=0;
unsigned char dec[4],bin[9];

void main()
{
   sleep(1000);                              // Delay 1 Sec
   soft_serout_init(0,9600);                 // Initial Serial Comm 9600 8N1
   uart_set_baud(1200);                      // Set IR remote control Baudrate
   sound(2000,200);                          // Status Sound
   serout_byte(0,0xFE);serout_byte(0,0x01);     // Clear Screen SLCD
   serout_byte(0,0xFE);serout_byte(0,0x80);     // Show Text on First Line
   serout_text(0,"You Press Key ");

   while(1)                               // Infinite Loop
   {
     key=uart_getkey(20000);              // read key from remote control 33 mS
     if  ((key!=flag)&&(key>0x40)&&(key<0x7F))
                                          // Test Key in Range and not same key
     {
        serout_byte(0,0xFE);
        serout_byte(0,0x8E);              // Show on SLCD
        serout_byte(0,key);
        flag=key;
     }
   }
}
```

Program description

This code requires 2 imporatant libraries include **serial.h** and **soft_serout.h**. The operation step as follows

(1) Delay 1 second to wait for SLCD initailize.

(2) Set the baudrate for SLCD communication to 9,600 bit per second.

(3) Set the baudrate for receiving data from ER-4 Remote control to 1,200 bit per second

(4) Generate the sound for starting the operation.

(5) Send the clear screen command to SLCD and send message You Press Key to display.

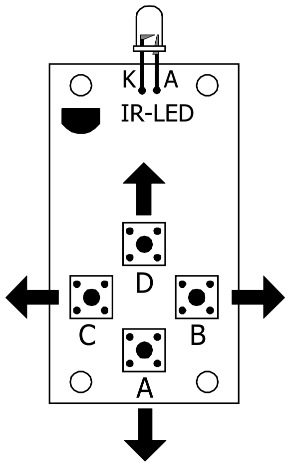
(6) Loop the program for checking the button pressing at ER-4 Remote control. If press the illegal button or do not press any button, program will not get the button data.

(7) If the pressing is correct, program will get the button data to display on SLCD screen.

Listing A12-1 : The C program code for getting the button data from ER-4 Remote control.

Activity 14
Infrared Remote control Robot

From the Activity 12, MicroCamp robot can get data from ER-4 Remote control 8 values as A, B, C, D and a, b, c, d. The button's position on ER-4 are designed to suitable for direction movement control as follows :



Move forward - top button or D or d button

Move backward - bottom button or A or a button

Move left - left button or C or c button

Move right - right button or B or b button

Get all button data to create the program for controlling the robot's movement with remote contol.

A13.1 Open the AVR Studio to create the new project and write the C program following the Listing A13-1.

A13.2 Add the `serial.h` library file into the project file. Build this project.

A13.3 Connect the PX-400 programmer to the MicroCamp board on The MicroCamp robot (must ZX-IRM and SLCD ready) at the In-System Prog. connector. Turn-on the Robot. Downlaod the HEX code to the robot.

A13.4 Turn-off power and Remove the ISP cable.

A13.5 Put the 2 of AA batteries inot Batter holder of ER-4 Remote control.

```
#include <stdlib.h>                         // Standard Library
#include <soft_serout.h>                    // For SLCD Module
#include <sleep.h>                          // For Delay
#include <serial.h>                         // For Receieve Data from ER-4
#include <sound.h>                          // For Generate Sound
#include <motor.h>                          // For Control Motor
unsigned char key,flag=0;

void main()
{
   sleep(1000);                             // Delay 1 Sec
   soft_serout_init(0,9600);                // Initial Serial Comm 9600 8N1
   uart_set_baud(1200);                     // Set IR remote control Baudrate

   sound(2000,200);                         // Status Sound
   serout_byte(0,0xFE);serout_byte(0,0x01); // Clear Screen SLCD
   serout_byte(0,0xFE);serout_byte(0,0x80); //  Show Text on First Line
   serout_text(0,"Press any IR Key");
   while(1)                              // Infinite Loop
   {
     key=uart_getkey(65000);             // wait and read key from ER-4 in 33ms
     if ((key=='a')||(key=='A'))       //  "A" key for Backward
     {
        backward(100);
        if (flag!=1)                 // Show "Backward" on SLCD
        {
           serout_byte(0,0xFE);
           serout_byte(0,0xC0);
           serout_text(0,"Backward   ");
           flag=1;
        }
     }
     else if ((key=='d')||(key=='D'))       // "D" key for Forward
     {
        forward(100);
        if (flag!=2)                        // Show "Forward" on SLCD
        {
           serout_byte(0,0xFE);
           serout_byte(0,0xC0);
           serout_text(0,"Forward    ");
           flag=2;
        }
     }
     else if ((key=='c')||(key=='C'))       // "C" key for Forward
     {
        s_left(100);
        if (flag!=3)                        // Show "Turn Left" on SLCD
        {
           serout_byte(0,0xFE);
           serout_byte(0,0xC0);
           serout_text(0,"Turn Left ");
           flag=3;
        }
     }
     else if ((key=='b')||(key=='B'))       // "D" key for Forward
```

Listing A13-1 : The C program for Microcamp robot with Remote control activity (continue)

```
      {
         s_right(100);
         if (flag!=4)                          // Show "Turn Right" on SLCD
         {
            serout_byte(0,0xFE);
            serout_byte(0,0xC0);
            serout_text(0,"Turn Right");
            flag=4;
         }
      }
      else
      {motor_off();flag=0;}                    // Stop motor if no key press
   }
}
```

Program description

This code is develped from Listing A12-1 to control the robot's movement with ER-4 Remote control. The code check the button's data that received as A, B, C D or a, B, C, d. In this code does not support case sensitive.

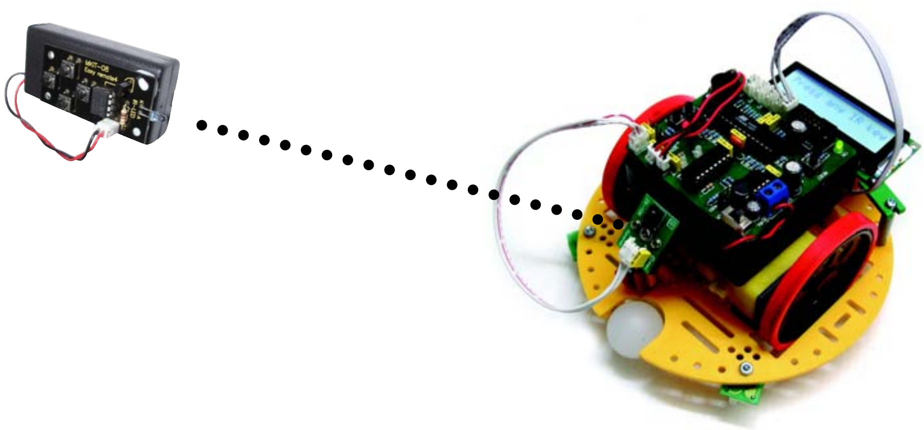
During the robot moves, microcontroller shows the operation message on SLCD screen too. The message will change if the button pressing is chenged.

Listing A13-1 : The C program for Microcamp robot with Remote control activity (final)

A13.6 Place the robot on the floor. Turn-on power.

A13.7 Press the ER-4's button to send data to MicroCamp robot. The direction of sending light must straight. The communication will be complete. Observe the robot's movement.

MicroCamp robot moves following the button's function.



MicroCamp 2.0 libraries source program

in_out.h : Read and Write the digital data with any port

```
#ifndef _IN_OUT_
#define _IN_OUT_

#define toggle_b(x)     DDRB |= _BV(x);   PORTB ^= _BV(x);
#define toggle_c(x)     DDRC |= _BV(x);   PORTC ^= _BV(x);
#define toggle_d(x)     DDRD |= _BV(x);   PORTD ^= _BV(x);
char in_b(char _bit)
{
   DDRB &= ~(1<<_bit);
   return((PINB & _BV(_bit))>>_bit);
}
char in_c(char _bit)
{
   DDRC &= ~(1<<_bit);
   return((PINC & _BV(_bit))>>_bit);
}
char in_d(char _bit)
{
   DDRD &= ~(1<<_bit);
   return((PIND & _BV(_bit))>>_bit);
}
void out_b(char _bit,char _dat)
{
   DDRB |= _BV(_bit);
   if(_dat)
     PORTB |= _BV(_bit);
   else
     PORTB &= ~_BV(_bit);
}
void out_c(char _bit,char _dat)
{
   DDRC |= _BV(_bit);
   if(_dat)
     PORTC |= _BV(_bit);
   else
     PORTC &= ~_BV(_bit);
}
void out_d(char _bit,char _dat)
{
   DDRD |= _BV(_bit);
   if(_dat)
     PORTD |= _BV(_bit);
   else
     PORTD &= ~_BV(_bit);
}

#endif
```

sleep.h : Delay function library

```
#ifndef _sleep_
#define _sleep_
void sleep(unsigned int ms)
{
   unsigned int i,j;
   for(i=0;i<ms;i++)
   for(j=0;j<795;j++);
}
#endif
```

analog.h : Analog reading input library

```
unsigned int analog(unsigned char channel)
{
   unsigned int adc_val;
   ADMUX = 0x40;
   ADMUX |= channel;                  // Single end mode
   ADCSRA = 0xC6;
   while((ADCSRA & (1<<ADSC)));
   adc_val = ADCL;
   adc_val += (ADCH*256);
   return(adc_val);
}
```

sound.h : Sound generator library

```
#include <in_out.h>
#include <sleep.h>
void delay_sound(unsigned int ms)
{
    unsigned int i,j;
    for(i=0;i<ms;i++)
    for(j=0;j<200;j++);
}

void sound(int freq,int time)
{
    int dt=0,m=0; // Keep value and
     dt = 5000/freq;  // Keep active logic delay
    time = (5*time)/dt; // Keep counter for generate sound
    for(m=0;m<time;m++) // Loop for generate sound(Toggle logic P0.12)
    {
      out_d(4,1);
      delay_sound(dt);  // Delay for sound
      out_d(4,0);
      delay_sound(dt);        // Delay for sound
    }
}
void sound_cnt(unsigned char cnt,int freq,int time)
{
    unsigned char i;
    for (i=0;i<cnt;i++)
    {
      sound(freq,time);
      sleep(300);
    }
}
```

led.h : LED control library

```
// Library for LED indicator by Timer 2 interrupt every 5 ms
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <in_out.h>
unsigned char LED=0;
unsigned char LED_cnt;

SIGNAL (SIG_OVERFLOW2)         // Interval 10 ms
{
   TCNT2 = 178;                // Reload interval 10 ms(TCNT2 = 178)
   LED_cnt++;                  // Increment Counter
   if (LED_cnt>30)             // Check Counter 10 ms X 30
   {
      LED_cnt=0;               // Clear Counter
      if (LED==1)              // Check LED1 Enable
      {
         toggle_c(5);
      }
      else if (LED==2)         // Check LED2 Enable
      {
         toggle_d(1);
      }
      else if (LED==3)         // Check LED1 and LED2 Enable
      {
         toggle_c(5);
         toggle_d(1);
      }
   }
}

void interval_init()           // Config. and Start up timer 0
{
    TCCR2 |= (1<<CS22)|(1<<CS21)|(1<<CS20);
                               // Prescaler 1024,16 MHz,
                               // 1 MC = 1024/16M = 64us/count
   TIFR |= 1<<TOV2;            // Clear TOV2 / clear
   TIMSK |= 1<<TOIE2;          // Enable Timer2 Overflow Interrupt
   TCNT2 = 178;                // Interval 10 ms
   sei();                      // Enable all interrupt
}

void led1_on()                 // Start Blinking LED1
{
   interval_init();
   LED |= (1<<0) ;
}
void led1_off()                // Stop Blinking LED1
{
   LED &= ~_BV(0);
}
void led2_on()
{
   interval_init();            // Start Blinking LED2
   LED |= (1<<1) ;
}
void led2_off()
{
   LED &= ~_BV(1);             // Stop Blinking LED2
}
```

motor.h : Dc motor control library

```
/* Hardware Configuration
   MOTOR1
   - PD7 Connect to 1B   port
   - PD6 Connect to 1A   port
   - PB1 Connect to 1E   port

   MOTOR2
   - PB0 Connect to 2A port
   - PD5 Connect to 2B port
   - PB2 Connect to 2E port

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#define ALL 3                                   // Clear all motor
#define all 3                                   // Clear all motor
unsigned char _duty1=0,_duty2=0;                // duty cycle variable
char pwm_ini =0;                                // Flag for check initial ?

SIGNAL (SIG_OVERFLOW1)                          // Interval 1 ms
{
   OCR1AL = _duty1;                             // Duty Cycle 1 Read
   OCR1BL = _duty2;                             // Duty Cycle 2 Read
}

void pwm_init()
{
   TCCR1A |= (1<<WGM10);
   TCCR1B = (1<<CS12)|(1<<CS10)|(1<<WGM12);     // Set Prescaler
// TCCR1B = (1<<CS12)|(1<<WGM12);               // Set Prescaler
   TIFR |= 1<<TOV1;                             //Clear TOV0 / clear
   TIMSK |= 1<<TOIE1;                     //Enable Timer0 Overflow Interrupt
                                          //timer_enable_int(_BV(TOIE1));
   sei();
}
void pwm(char channel,unsigned int duty)
{
   duty = (duty*255)/100;                       // Convert 0-100 to 0-255
   if(pwm_ini==0)                               // PWM Initial ?
   {
      pwm_init();                               // If no Intitial it
      pwm_ini=1;                                // show now initial
   }
   if(channel==2)
   {
      TCCR1A |= _BV(COM1A1);
      DDRB |= _BV(PB1);
      OCR1AL = duty;
      _duty1 = duty;
   }
   else if(channel==1)
   {
      TCCR1A |= _BV(COM1B1);
      DDRB |= _BV(PB2);
      OCR1BL = duty;
      _duty2 = duty;
   }
```

```
   else if(channel==3)
   {
      TCCR1A |= _BV(COM1A1);
      DDRB |= _BV(PB1);
      OCR1AL = duty;
      _duty1 = duty;
      TCCR1A |= _BV(COM1B1);
      DDRB |= _BV(PB2);
      OCR1BL = duty;
      _duty2 = duty;
   }
}

void motor(char _channel,int _power)
{
   if(_power>0)
   {
      pwm(_channel,_power);
      if(_channel==2)
      {
         out_d(7,1);
         out_d(6,0);
      }
      else if(_channel==1)
      {
         out_d(5,0);
         out_b(0,1);
      }
   }
   else
   {
      pwm(_channel,abs(_power));
      if(_channel==2)
      {
         out_d(7,0);
         out_d(6,1);
      }
      else if(_channel==1)
      {
         out_d(5,1);
         out_b(0,0);
      }

   }
}

void motor_stop(char _channel)
{
   pwm(_channel,100);
   if(_channel==2 ||_channel==3)
   {
      out_d(7,0);
      out_d(6,0);
   }
   if(_channel==1||_channel==3)
   {
      out_d(5,0);
      out_b(0,0);
   }
}
```

```
void motor_off()
{
   pwm(3,0);
   out_d(7,0);
   out_d(6,0);
   out_d(5,0);
   out_b(0,0);
}

void forward(int speed)
{
   motor(1,speed);
   motor(2,speed);
}

void backward(int speed)
{
   motor(1,speed*-1);
   motor(2,speed*-1);
}

void s_left(int speed)
{
   motor(1,speed);
   motor(2,speed*-1);
}

void s_right(int speed)
{
   motor(1,speed*-1);
   motor(2,speed);
}
```

timer.h : Timer library

```
#include <C:/WinAVR/avr/include/avr/interrupt.h>
#include <C:/WinAVR/avr/include/avr/signal.h>

/******************* Timer 0  Interrupt *********************************/
/****************** Interval 1 ms ****************************************/

unsigned long _ms=0;

SIGNAL (SIG_OVERFLOW0)           // Interval 1 ms
{
   TCNT0 = 6;                    // Interval 1 ms
   _ms++;

}
void timer_start(void)           // Config. and Start up timer 0
{

   TCCR0 = (1<<CS01)|(1<<CS00);  // Prescaler 64,16MHz,1 MC = 64/16M = 4us/
count
   TIFR |= 1<<TOV0;              // Clear TOV0 / clear
   TIMSK |= 1<<TOIE0;            // Enable Timer0 Overflow Interrupt
   TCNT0 = 6;                    // Interval 1 ms

   sei();                        // Enable all interrupt
   _ms = 0;
}
void timer_stop()
{
   TCCR0 = 0;                    // Stop timer and
   TCNT0 = 0;
   TIMSK &= ~_BV(TOIE0);         // Clear bit TOIE0
   _ms = 0;                      // Clear time
}

void timer_pause()
{
   TCCR0 = 0;                    // Stop timer and not clear time
}

void timer_resume()
{
   TCCR0 = (1<<CS01)|(1<<CS00);  // Prescaler 64,16 MHz,
                                 // 1 MC = 64/16M = 4us/count
}

unsigned long msec()
{
   return(_ms);
}

unsigned long sec()
{
   return(_ms/1000);
}
```

serial.h : ATmega8's UART Serial data communication library

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define F_OSC 16000000                /* oscillator-frequency in Hz */
#define UART_BAUD_CALC(x,F_OSC) ((F_OSC)/((x)*16l)-1)
#define even_uart_rec() SIGNAL(SIG_UART_RECV)

//------------------- Condition test parameter data type for displa -----------//
#ifndef TEST_CHAR_TYPE(x)
#define TEST_CHAR_TYPE(x) *x=='%' && (*(x+1)=='c' || *(x+1)=='C')
#endif

#ifndef TEST_INT_TYPE(x)
#define TEST_INT_TYPE(x) *x=='%' && (*(x+1)=='d' || *(x+1)=='D')
#endif

#ifndef TEST_LONG_TYPE(x)
#define TEST_LONG_TYPE(x) *x=='%' && (*(x+1)=='l' || *(x+1)=='L')
#endif

#ifndef TEST_FLOAT_TYPE(x)
#define TEST_FLOAT_TYPE(x) *x=='%' && (*(x+1)=='f' || *(x+1)=='F')
#endif

#ifndef TEST_STRING_TYPE(x)
#define TEST_STRING_TYPE(x) *x=='%' && (*(x+1)=='s' || *(x+1)=='S')
#endif

#ifndef F_PREC
#define F_PREC 3
#endif

unsigned int _baud=9600;
char uart_ini=0;
char _key=0;

#ifndef USE_EVEN_UART_REC

SIGNAL(SIG_UART_RECV)
{
   _key = UDR;
}
#endif

void uart_set_baud(unsigned int baud)
{
   _baud = baud;
   uart_ini==1;
// Set baud rate
   UBRRH = (unsigned int)(UART_BAUD_CALC(baud,F_OSC)>>8);
   UBRRL = (unsigned int)UART_BAUD_CALC(baud,F_OSC);
// UBRRH = 00; //for 9600 bps
// UBRRL = 51;
```

```
// Enable receiver and transmitter; enable RX interrupt
   UCSRB |= (1 << RXEN) | (1 << TXEN) | (1 << RXCIE);

// Asynchronous 8N1
   UCSRC |= (1 << URSEL) | (3 << UCSZ0);
   sei();  // enable interrupts

}

unsigned int uart_gets_baud()
{
   return(_baud);
}

void uart_putc(unsigned char c)
{
   if(uart_ini==0)
   {
      uart_ini==1;
      uart_set_baud(_baud);
   }
   while(!(UCSRA & (1 << UDRE)));
   UDR = c;    // send character
}

void uart_puts(char *s)
{
   while (*s)
   {
      uart_putc(*s);
      s++;
   }
}

void uart(char *p,...)
{
   char *arg,**pp;                // Pointer of point
   char *ptr,buff[16];            //  s_arg_offset=0,s_arg_i=0//;
   pp = &p;
   ptr = p;                       // Copy address
   arg = pp;                      // Copy address of p point

   arg += 2;                      // Cross 2 time go to Origin of first parameter

   if(uart_ini==0)
   {
      uart_ini==1;
      uart_set_baud(_baud);
   }

   while(*ptr)                          // Check data pointer = 0?
   {
      if(TEST_CHAR_TYPE(ptr))
      {
         uart_putc(toascii(*arg));
         arg+=2;                        // Cross address char type
         ptr++;                         // Cross %d parameter
      }
      else  if(TEST_INT_TYPE(ptr))
```

```
      {
         p = ltoa(*(unsigned int *)arg,&buff[0],10);
         uart_puts(p);
         arg+=2;                        // Cross address int type
         ptr++;                         // Cross %d parameter
      }
      else  if(TEST_LONG_TYPE(ptr))
      {
         p = ltoa(*(long *)arg,&buff[0],10);
         uart_puts(p);
         arg+=4;                        // Cross address long type
         ptr++;                         // Cross %l parameter
      }
      else  if(TEST_FLOAT_TYPE(ptr))
      {
         p = dtostrf(*(float *)arg,2,F_PREC,&buff[0]);
                                        // Convert float to string(used libm.a)
         uart_puts(p);
         arg+=4;                        // Cross address long type
         ptr++;                         // Cross %l parameter
      }
      else
      {
         uart_putc(*ptr);               // Send data to LCD
      }
      ptr++;                            // Increase address 1 time
   }
}
/*
char uart_getkey()
{
   char _c=_key;
   if(uart_ini==0)
   {
      uart_ini==1;
      uart_set_baud(_baud);
   }
   _key = 0;
   return(_c);
}
*/

char uart_getkey(unsigned int timeout)
{
   unsigned int cnt=1;
   char _c=0;
   if(uart_ini==0)
   {
      uart_ini==1;
      uart_set_baud(_baud);
   }
   while(!_key&&(cnt<timeout))
   {cnt++;}
   _c = _key;
   _key = 0;
   return(_c);
}
```

soft_serout.h : Serial data output library for any port of ATmega8 microcontroller (not use UART)

```
#include <avr/io.h>
#include <in_out.h>
#ifndef _soft_serout_
#define _soft_serout_

#define PRESCALER_1  (1<<CS20)              // (1/16M) 0.0625 us per MC
#define PRESCALER_8  (1<<CS21)              // (8/16M) 0.5 us per MC
#define PRESCALER_32 (1<<CS21) | (1<<CS20)  // (32/16M) 2 us per MC
#define PRESCALER_64 (1<<CS22)              // (64/16M) 4 us per MC
#define OFFSET_DELAY120                     // for out function used 20 us

#define OFFSET_DELAY218                     // for out function used 20 us
unsigned int base=0;
unsigned char base_start_rcv=0;
unsigned char TCCR2_cal=0;
unsigned int base;
unsigned int baud=9600;

void soft_serout_init(char tx,unsigned long baud_)
                                            // Config and Start up timer 2
{
   unsigned long tick=0;
   out_c(tx,1);
   if(baud_<=4800)
   {
     tick = ((1000000/baud_)-OFFSET_DELAY1)/4;  // Calculate delay for baudrate
     TCCR2_cal  = PRESCALER_64;
   }
   else if(baud_>4800 && baud_<=9600)
   {
     tick = ((1000000/baud_)-OFFSET_DELAY2)/2;  // Calculate Delay for baudrate
     TCCR2_cal  = PRESCALER_32;
   }
   TCCR2  = 0;                                  // Stop timer
   TIFR |= 1<<TOV2;                             // Ensure Overflow flag clearing
   base = 255-tick;
   base_start_rcv = 255-(tick/2);
}

// Delay for baudrate
void delay_baud(unsigned int _tick)
{
   TCNT2  = _tick;                              // Load Prescaler from calculate
   TCCR2  = TCCR2_cal;                          // Load interval
   while(!(TIFR & (1<<TOV2)));                  // Wait until count success
   TIFR |= 1<<TOV2;                             // Ensure Clear Overflow flag
   TCCR2  = 0;                                  // Stop timer 2
}
```

```
// Send Data  1 Byte
void serout_byte(char tx,unsigned char dat)
{
   int i;
   out_c(tx,0);               // start bit
   delay_baud(base);          // Delay for start bit
   for(i=0;i<8;i++)
   {
     out_c(tx,dat & 0x01);    // Send data bit
     delay_baud(base);        // Delay for calculate base
     dat=dat>>1;              // Shift for next bit
   }
   out_c(tx,1);               // stop bit
   delay_baud(base);          // Delay for stop bit
}

// Send More Than 1 byte
void serout_text(char tx,unsigned char *p)
{
   while(*p)
   { serout_byte(tx,*p++); }
}
#endif
```

INNOVATIVE EXPERIMENT