

# MicroCamp

## ATmega8 Activity Kit manual



[www.inex.co.th](http://www.inex.co.th)  
[www.inexglobal.com](http://www.inexglobal.com)

# Content

---

<b>Chapter 1 MicroCamp Activity kit hardware.....</b>	<b>3</b>
<b>Chapter 2 Development software for MicroCamp kit.....</b>	<b>15</b>
<b>Chapter 3 C programming development for MicroCamp kit with AVR Studio and WinAVR C-compiler.....</b>	<b>23</b>
<b>Chapter 4 Library and Function of C programming .....</b>	<b>33</b>
<b>Chapter 5 Operator of WinAVR C-compiler.....</b>	<b>43</b>
<b>Chapter 6 Library and Specific command in MicroCamp kit.....</b>	<b>53</b>
<b>Chapter 7 Building robot with MicroCamp kit.....</b>	<b>63</b>
<i>Activity 1 Basic movement of MicroCamp robot.....</i>	<i>68</i>
<i>Activity 1-1 Forward and Backward movement</i>	
<i>Activity 1-2 Circle-shape movement control</i>	
<i>Activity 1-3 Square-shape movement control</i>	
<i>Activity 2 Object detection with Collision.....</i>	<i>71</i>
<i>Activity 2-1 Simple collision detection</i>	
<i>Activity 2-2 Trapped in a corner situation</i>	
<i>Activity 3 Line tracking robot.....</i>	<i>75</i>
<i>Activity 3-1 Testing black and white area</i>	
<i>Activity 3-2 Robot moves along the black line</i>	
<i>Activity 3-3 Line crossing detection</i>	
<b>MicroCamp libraries source program.....</b>	<b>83</b>

# Chapter 1

## MicroCamp Activity kit hardware

**MicroCamp** is a set of Microcontroller Activity kit for learning about Microcontroller 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"), Switch module, Infrared Reflector module, DC motor gearboxes and many other mechanical parts for building a programmable robot.

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

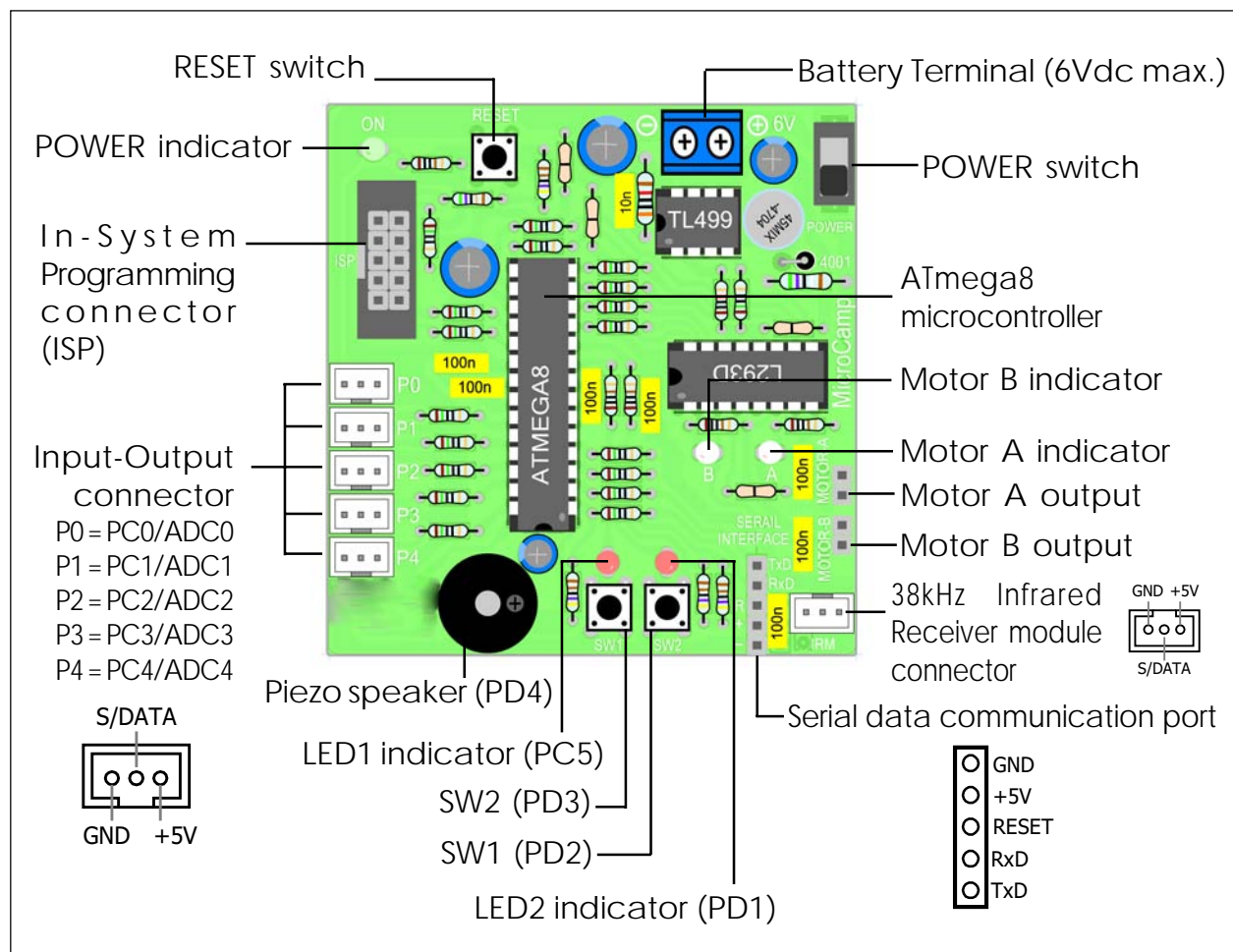


Figure 1-1 MicroCamp board layout

## 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 Digital 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 Digital 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.

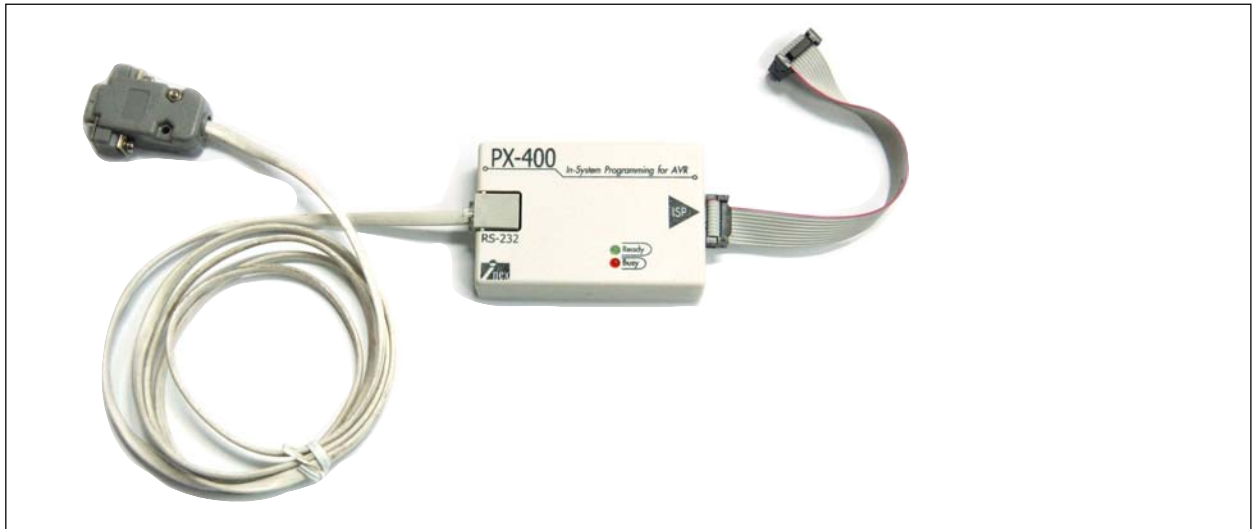


Figure 1-2 Shows PX-400 In-System Programmer box for AVR microcontroller.

- 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-Ospll 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

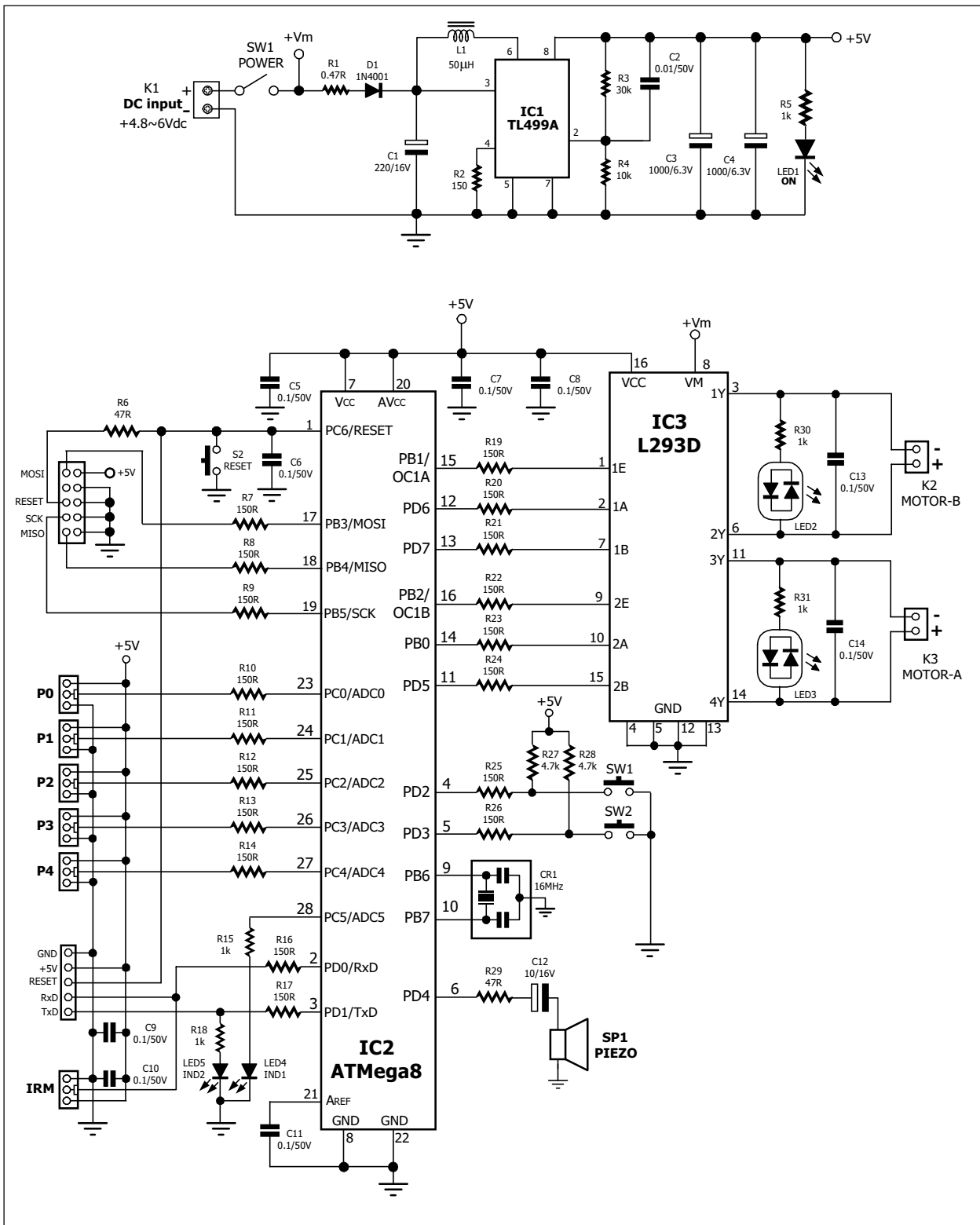


Figure 1-3 MicroCamp controller board schematic

## 1.2 MicroCamp controller board circuit description

The heart of this controller board is ATmega8 microcontroller. It runs on a 16MHz clock from crystal 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 be programmable to analog or digital input/output. Analog signal from these ports 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 connection with the external ISP programmer box.

PC6/RESET pin is connected with the RESET switch 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 directly 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 4.7k $\Omega$  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 10 $\mu$ F.

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 enable 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.

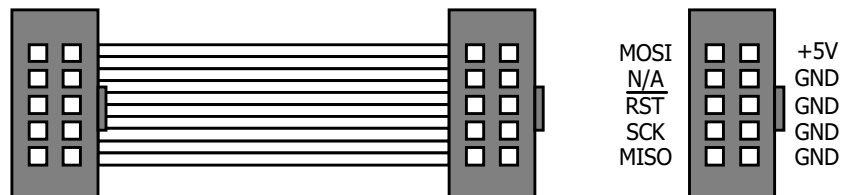
The Power supply circuit of this board is switching type circuit. TL499A is set to step-up +5V switching regulator for supply voltage to all microcontroller circuit except for the motor driver. With this circuit, it helps microcontroller voltage supply to be more stabilized. Although DC motors require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



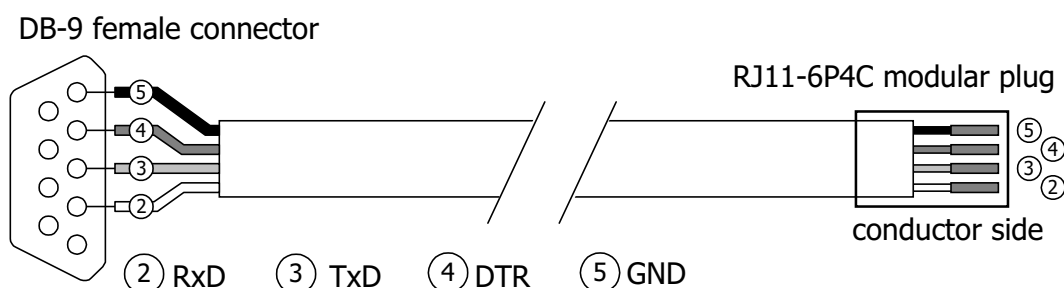
### 1.3.2 PCB3AA-8 cable

This is an INEX standard cable, 3-wires combined with 2mm. The PCB 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 EEPROM with 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.

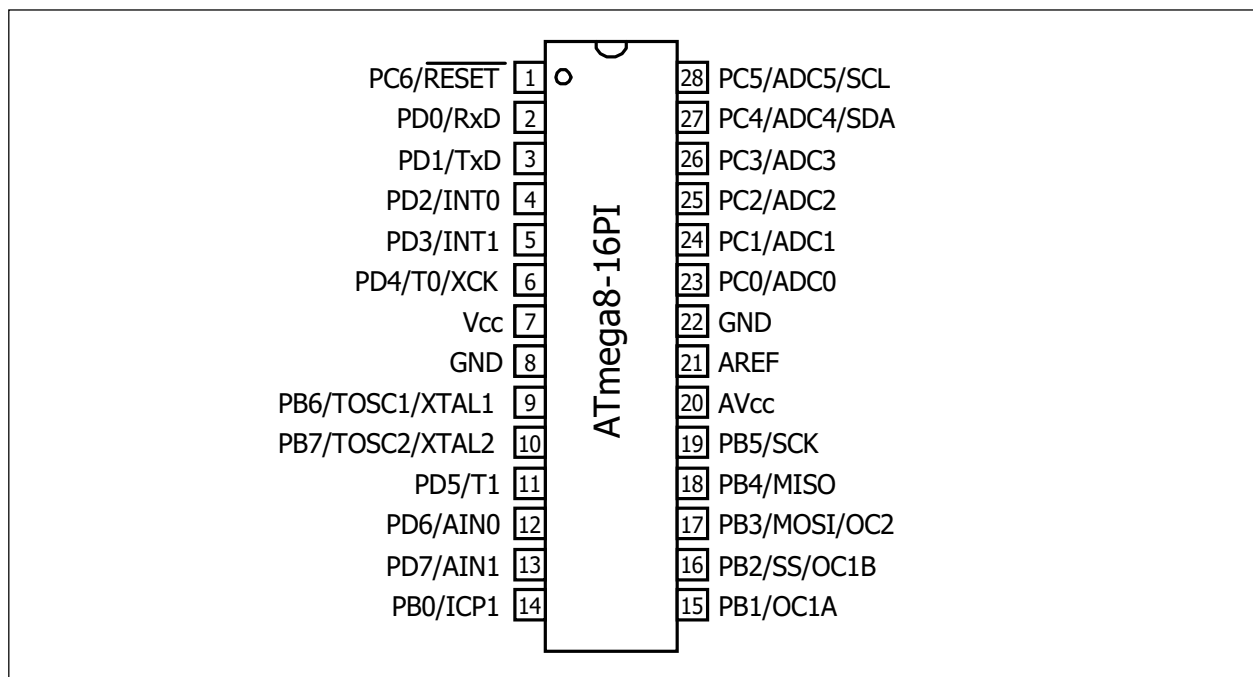


Figure1-4 ATmega8 microcontroller pin assignment

- 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.

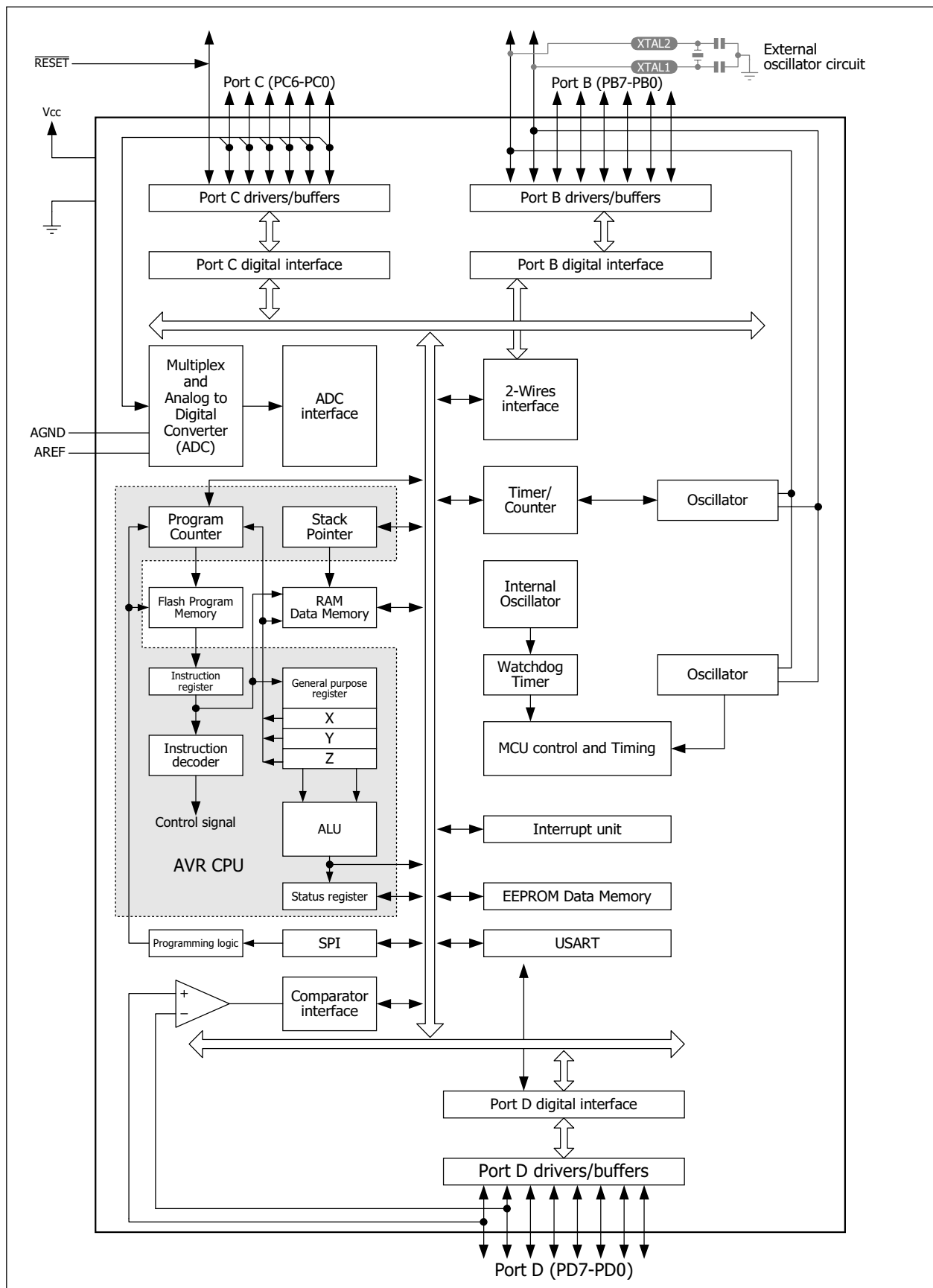


Figure 1-5 ATmega8 Block Diagram

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
<b>Port B</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 bus and 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
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 MicroCamp 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](http://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), avargcc (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.

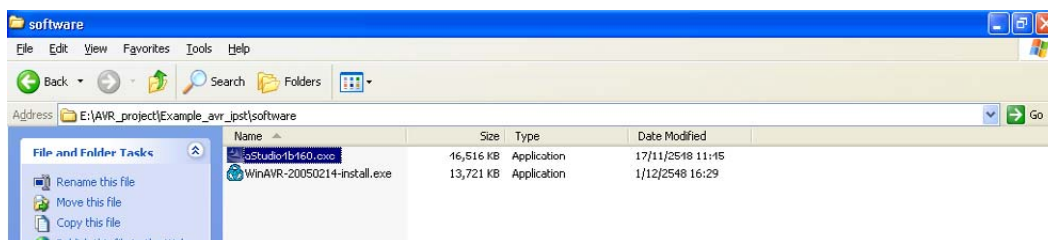
**3. Library** : These are the support files which allows the user to develop their C language program more comfortably. An example is the Port control library for controlling both Digital and Analog Input/Output, Motor control instructions, etc.

**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.

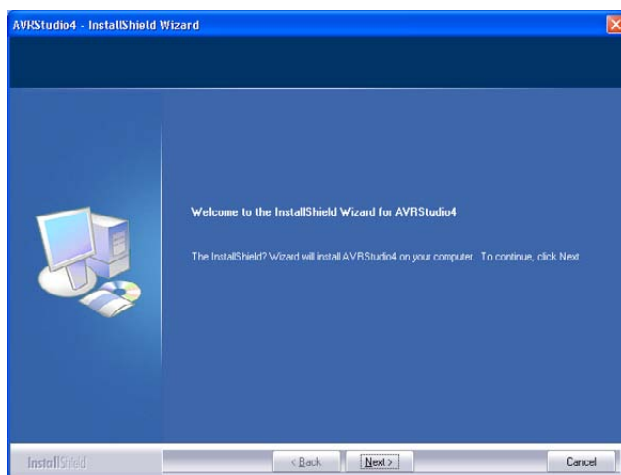
## 2.1 Installation AVR Studio

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.



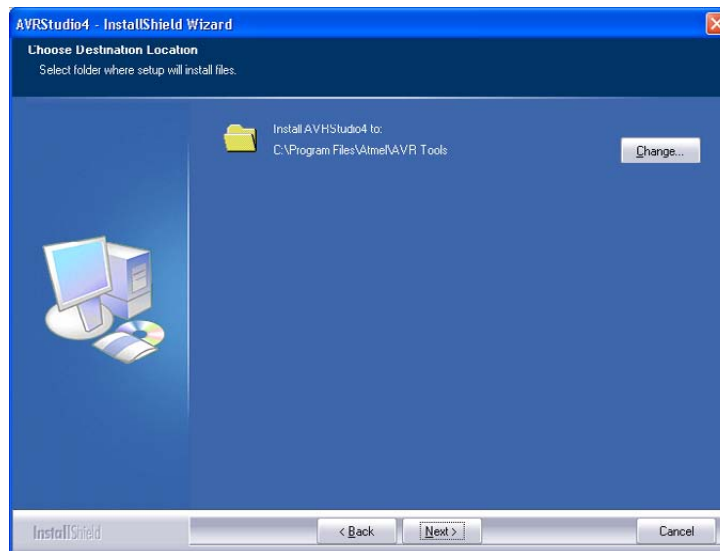
2.1.2 Enter Installation Wizard. Click on the **Next** button to continue.



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.

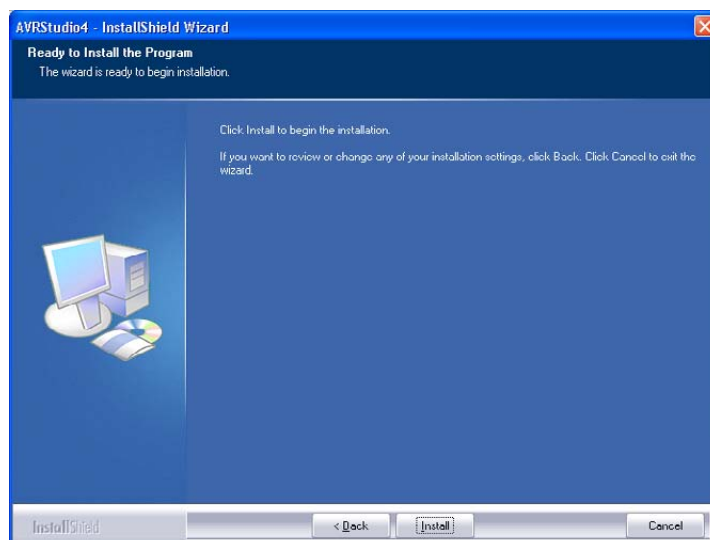


2.1.4 **Choose Destination Location** windows will appear. You can change the path by clicking on the **Change** button and setting the new path. After this, click on the **Next** button.



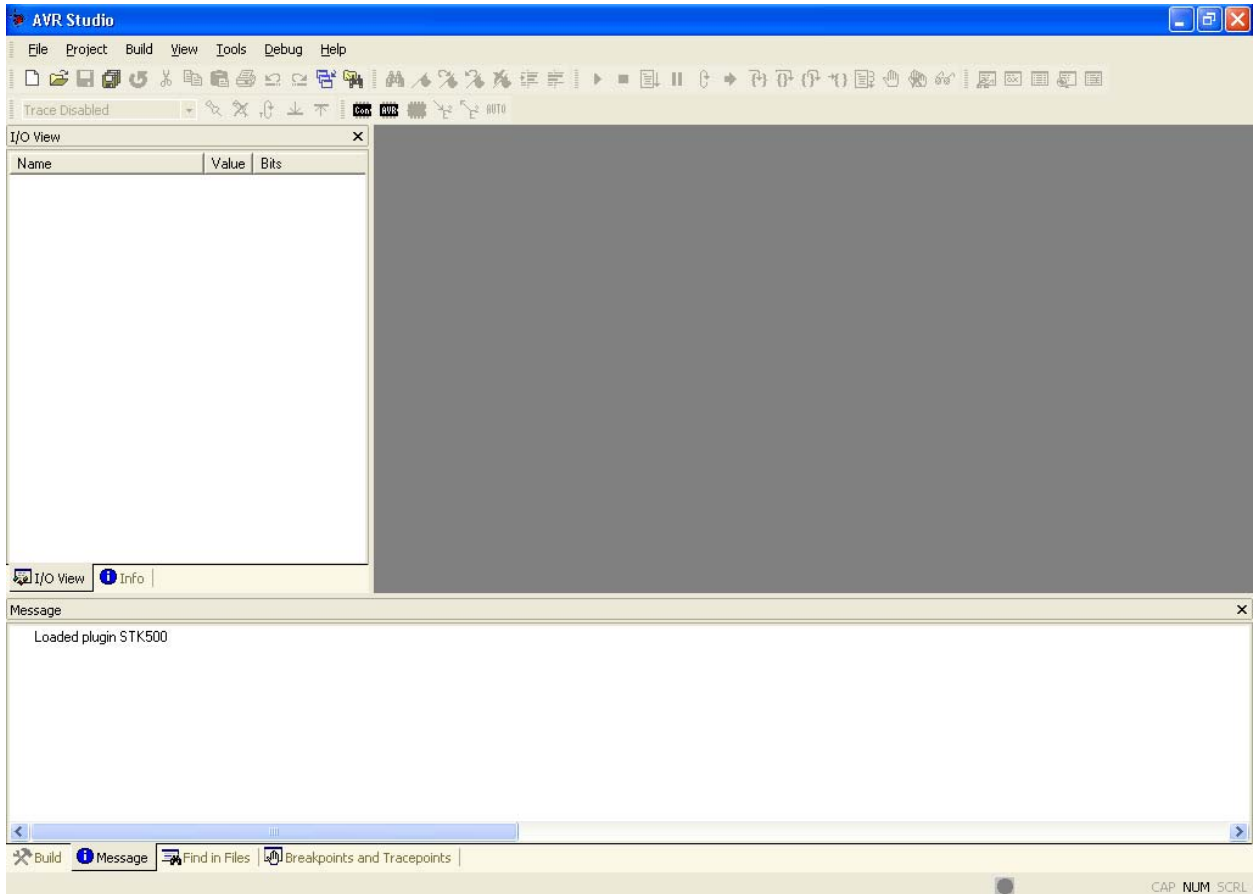
2.1.5 The Driver USB Upgrade window will now appear. Click on the **Next** button to pass this step.

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.

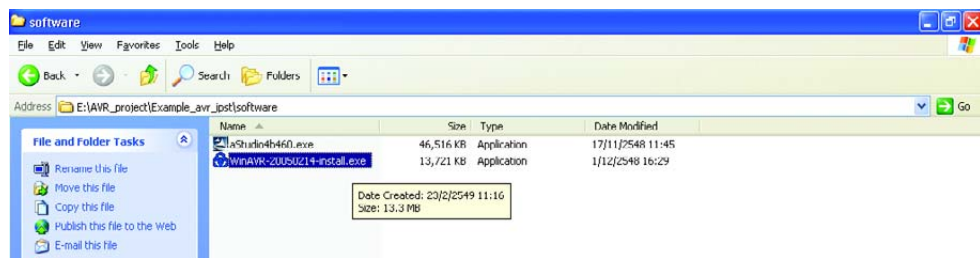


## 2.2 Instalaltion of WinAVR

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.



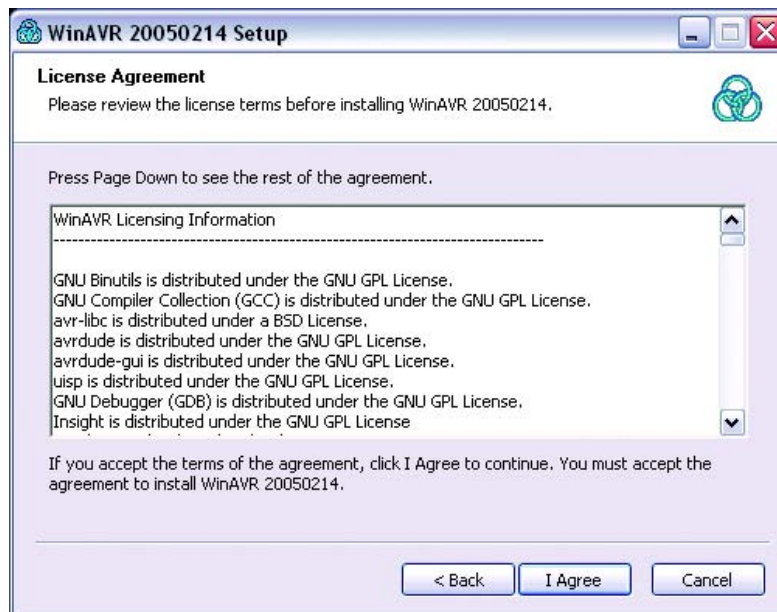
2.2.2 **Installation language** dialog box will appear for selection the language of this installation. Slect your preferred language from the sliding bar. After that click on the **OK** button.



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



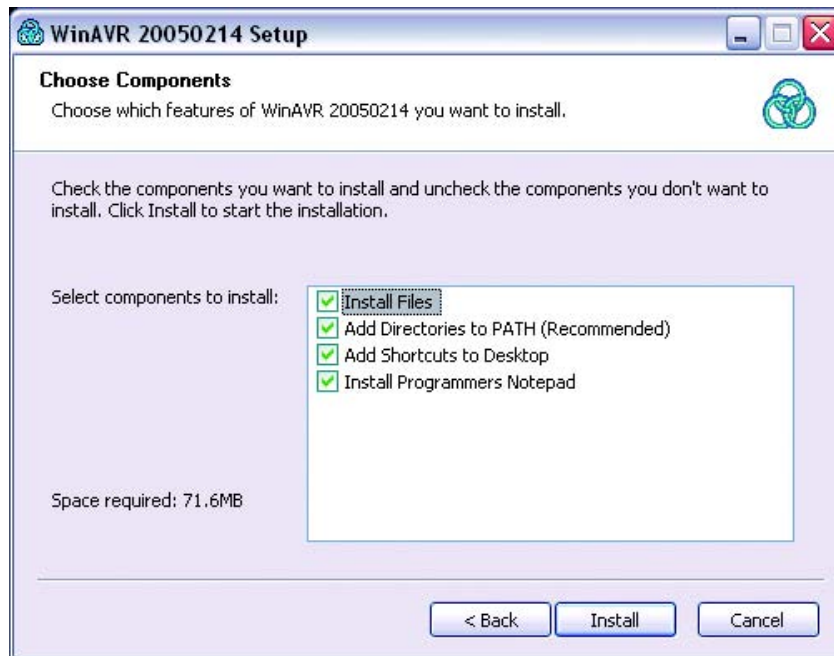
2.2.4 In the License agreement window, Click on the **I agree** button.



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 **MicroCamp\_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.

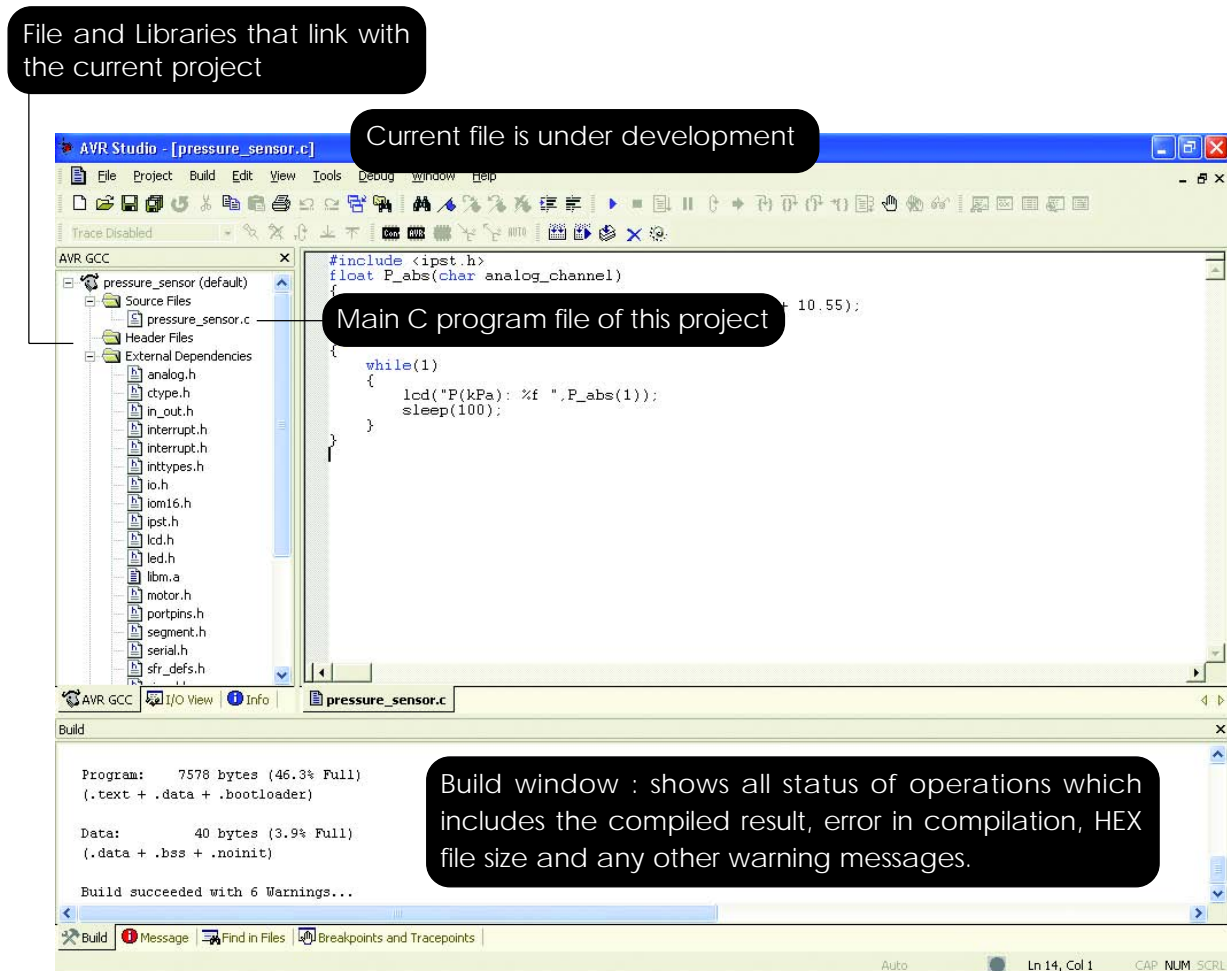
In the C programming development platform in AVR Studio, developers need to compile it into project file format. After the codes are being compiled into HEX file using the same name as the project filename, the file is needs to be downloaded into the ATMEGA8 Microcontroller.

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 :

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

### 3.3.2 Project menu

Includes the command as follows :

<b>Project Wizard</b>	Open the project wizard. You must close the current project first.
<b>New Project</b>	Open the new project dialog. You must close the current project first.
<b>Open Project</b>	Open a new project, either an APS project file or an object file.
<b>Save Project</b>	Save the current project with all settings
<b>Close Project</b>	Close the current project
<b>Recent Projects</b>	Show a list of recent project, select one to open
<b>Configuration Options</b>	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 :

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

### 3.2.4 Edit menu

Includes the command as follows :

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

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

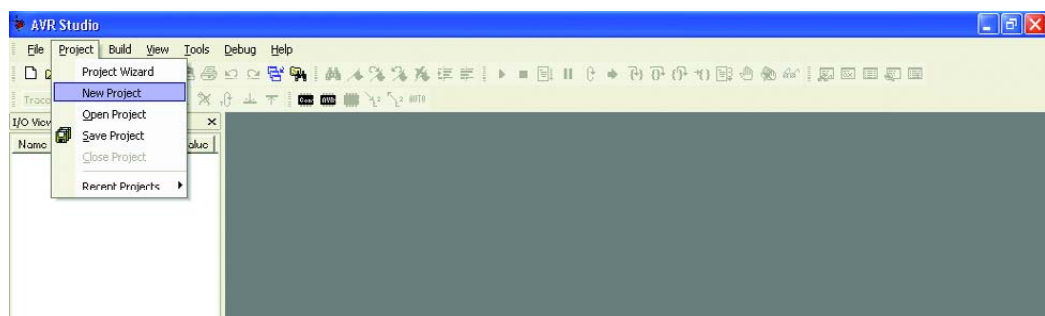
### 3.2.7 Debug menu

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**.

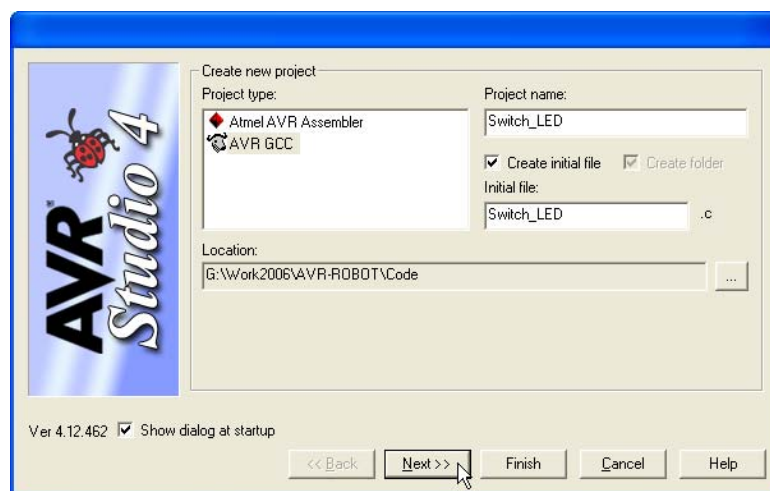


3.3.3 The properties project window will appear. Set the parameter as follows :

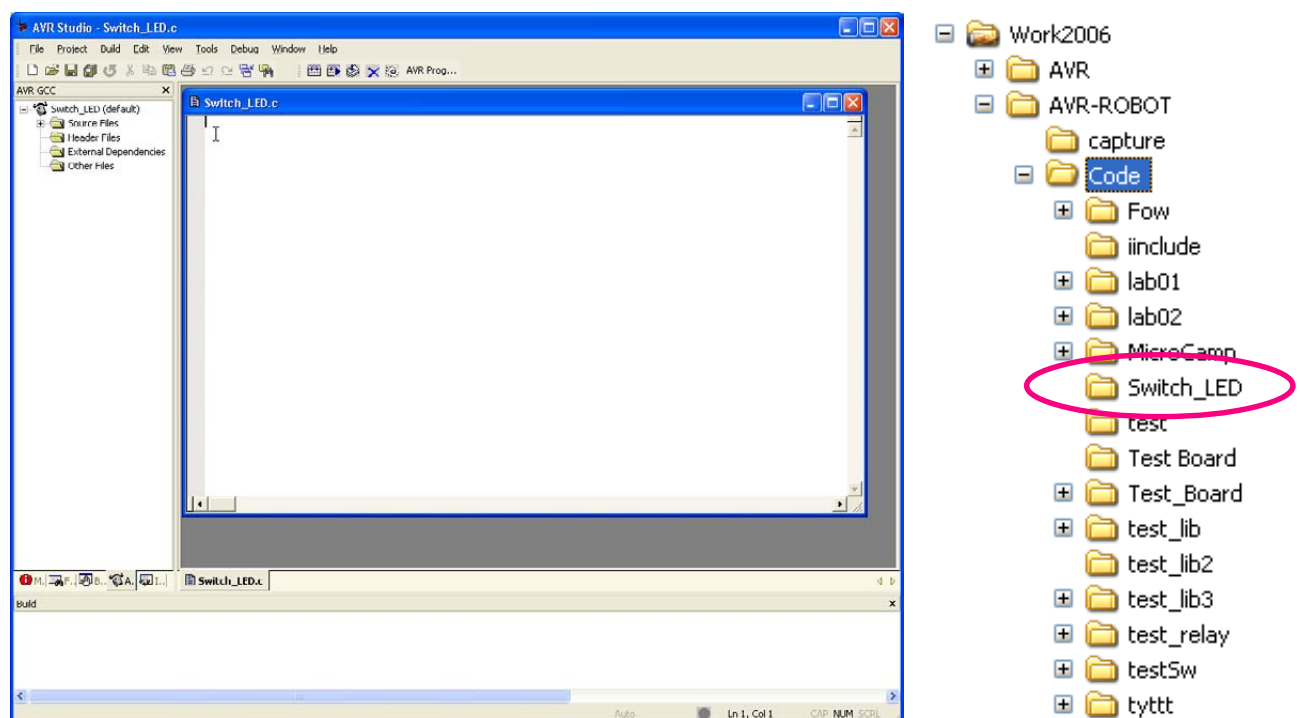
3.3.3.1 Click on this to select AVR GCC item within **Project type**: for select type of project file to program in C.

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.



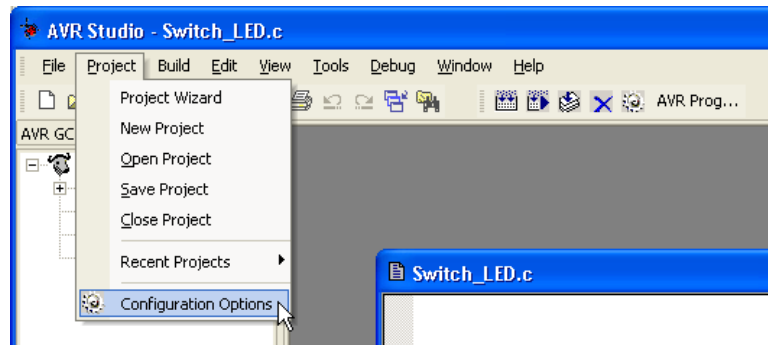
3.3.4 The **Switch\_LED** project environment will be created as shown in the diagram below.



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**

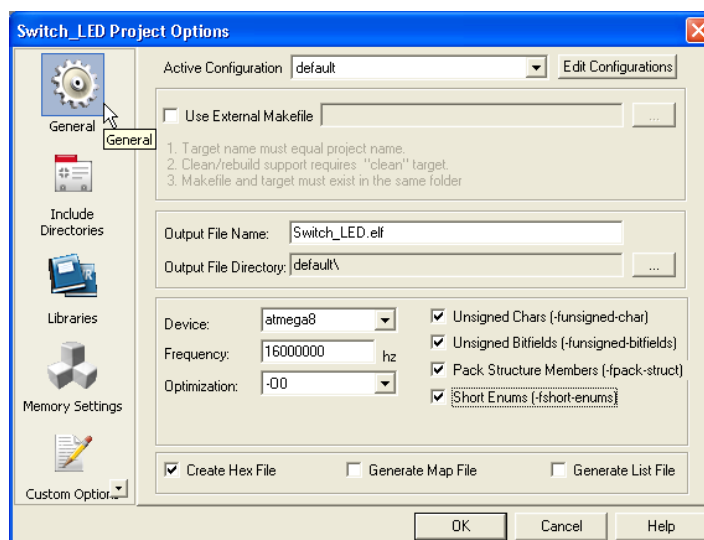


After that the window **Switch\_LED Project Options** will appear for setting the properties. See the left of this window. Developers will found 5 icons as :

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

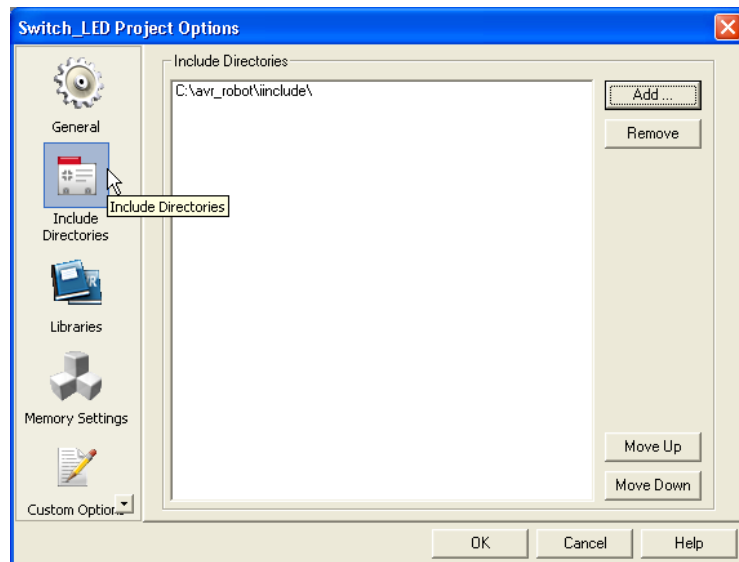
#### 3.3.5.2 At General icon, determine all data follows

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

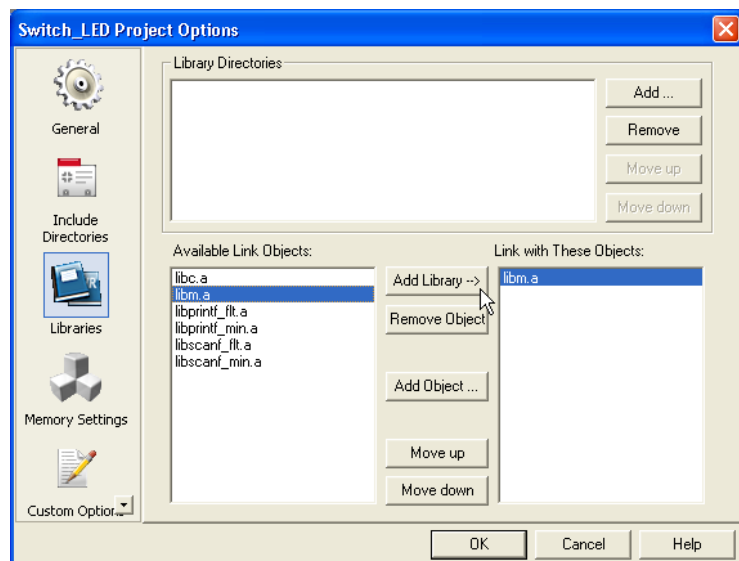


3.3.5.3 Click on this icon and the Include Directories for determining the path of library file. Find and select the library file and click on the **Add** button. For example is **C:\AVR\_ROBOT\include**. After determining the path, you will found the list for selection.

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



3.3.5.5 At the boxes , **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.



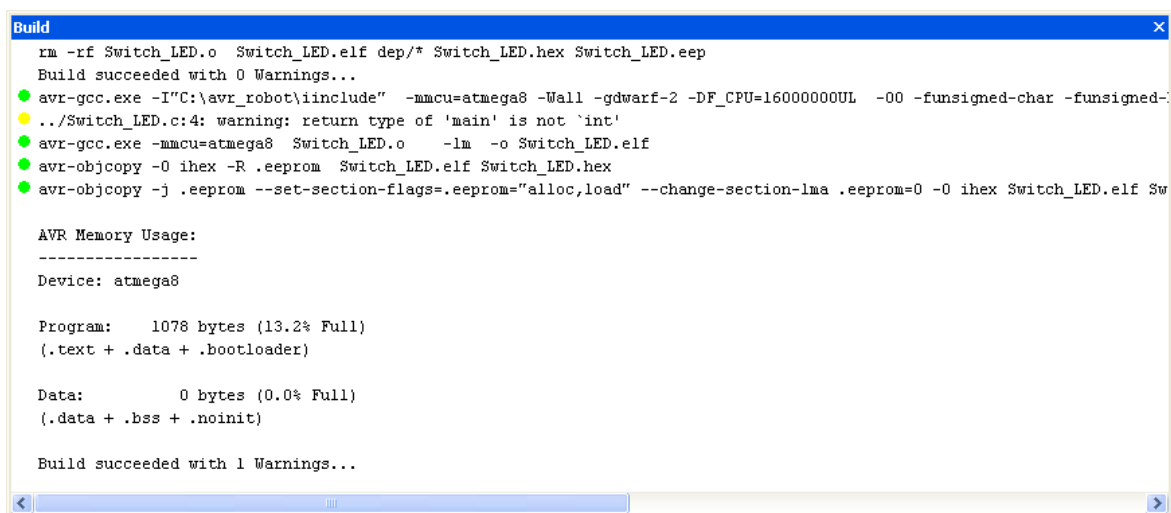
3.3.6 Next, write the C code in the **Switch\_LED.c** file. This file controls the microcontroller to On and off the LED when the switch is pressed. The details & codes of this file is shown in the Listing 3-1.

```
#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.*



```
Build
rm -rf Switch_LED.o Switch_LED.elf dep/* Switch_LED.hex Switch_LED.eep
Build succeeded with 0 Warnings...
avr-gcc.exe -I"C:\avr_robot\include" -mmcu=atmega8 -Wall -gdwarf-2 -DF_CPU=16000000UL -O0 -funsigned-char -funsigned-
../Switch_LED.c:4: warning: return type of 'main' is not 'int'
avr-gcc.exe -mmcu=atmega8 Switch_LED.o -lm -o Switch_LED.elf
avr-objcopy -O ihex -R .eeprom Switch_LED.elf Switch_LED.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex Switch_LED.elf Sw

AVR Memory Usage:
-----
Device: atmega8

Program: 1078 bytes (13.2% Full)
(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)

Build succeeded with 1 Warnings...
```

*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, prefix all errors and re-compile 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 controler 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.

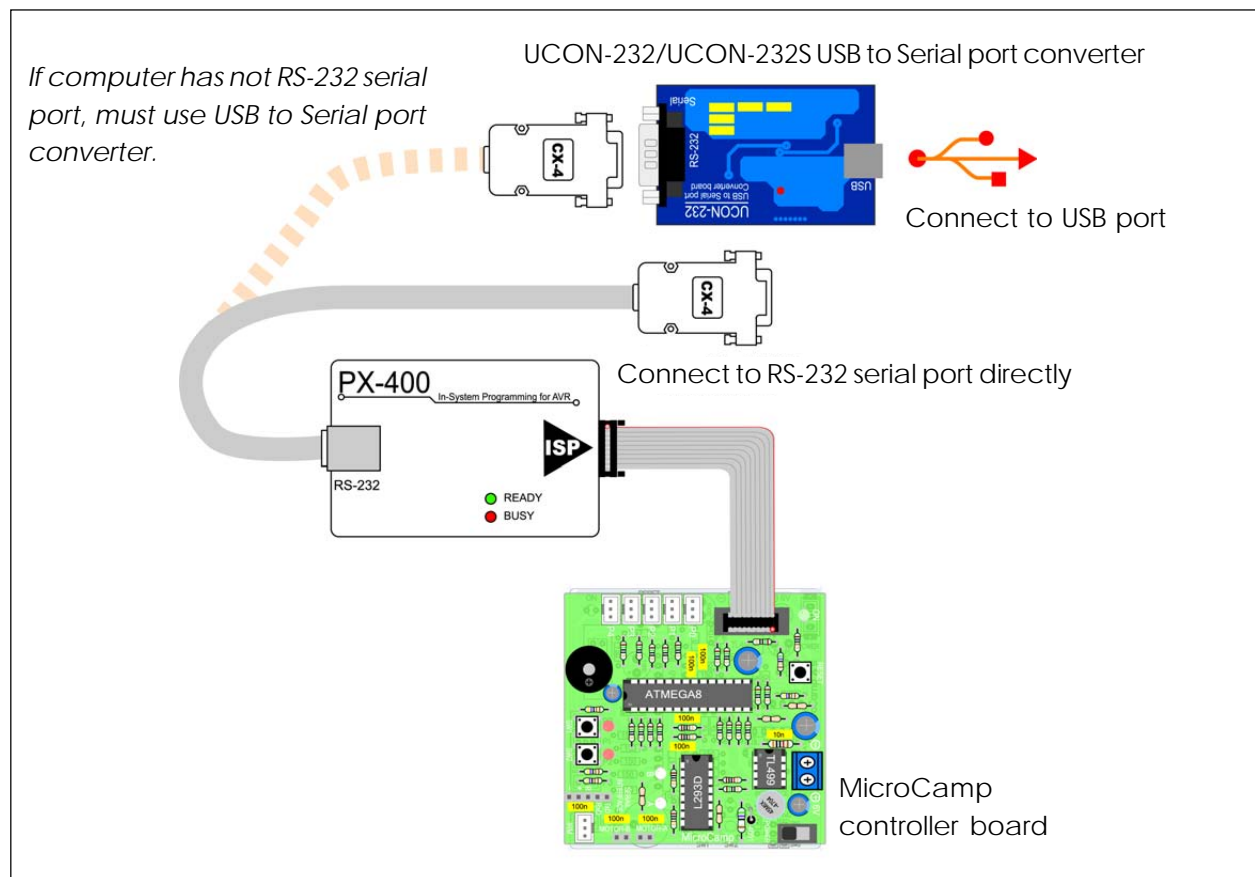
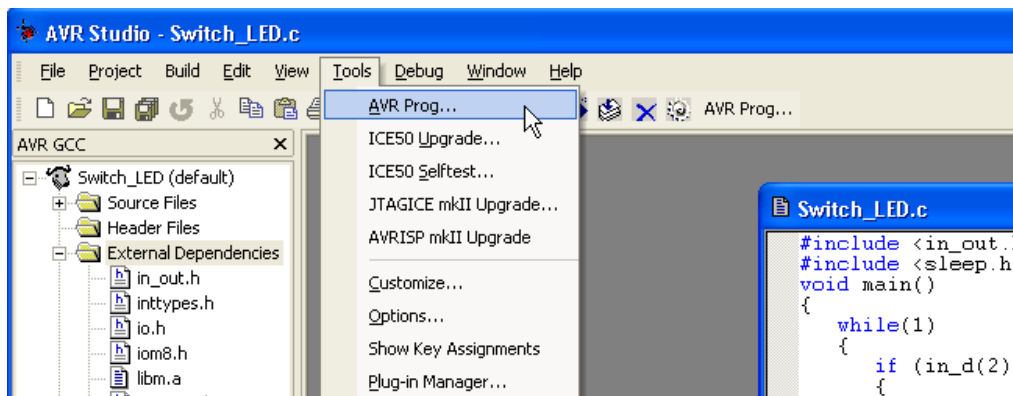


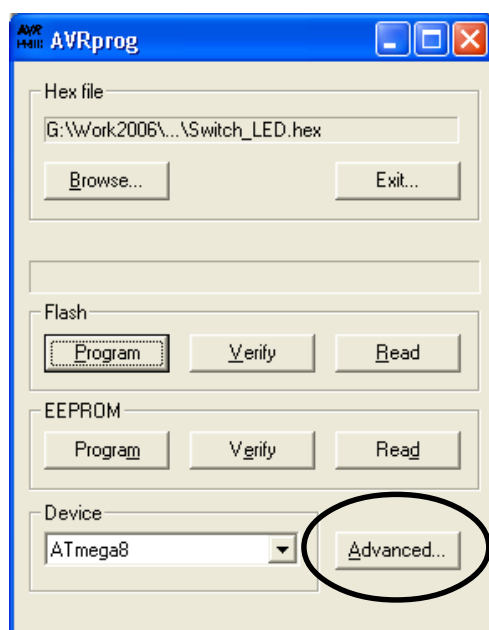
Figure 3-1 Connection diagram of PX-400 programmer box and MicroCamp controller board for downloading the program

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 advised **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 use of functions easy, convenient 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 function 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**). Another 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.hex** 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 :  $(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$   
 $= 146_{10}$

### Example 4-8

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

The calculation :  $(1 \times 2^{15}) + (1 \times 2^{14}) + (1 \times 2^{13}) + (1 \times 2^{12}) + (0 \times 2^{11}) + (1 \times 2^{10}) + (0 \times 2^9)$   
 $+ (0 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2)$   
 $+ (0 \times 2^1) + (1 \times 2^0) = 62701_{10}$

### Example 4-9

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

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

### Example 4-10

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

The calculation :  $(3 \times 16^1) + (1 \times 16^0) = 49_{10}$  and 0x31  $\rightarrow$  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 variable 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 x=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 addition 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 :**

$y = x/3; \rightarrow y = 10/3 \rightarrow y = 3$  (Returns an Integer)

$z = x\%3; \rightarrow z = 10\%3 \rightarrow z = 1$  (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
<i>a &gt; b</i>	<i>true.</i>	<i>1</i>
<i>a &gt; b</i>	<i>true</i>	<i>1</i>
<i>a &gt; c</i>	<i>false</i>	<i>0</i>
<i>a &gt;= c</i>	<i>true</i>	<i>1 (because 0xA0 = 10)</i>
<i>a != b</i>	<i>true</i>	<i>1</i>
<i>a != c</i>	<i>false</i>	<i>0</i>

or

<i>!(a &gt; b)</i>	<i>false</i>	<i>0</i>
<i>!(a &gt; c)</i>	<i>true</i>	<i>1</i>
<i>!(a &gt;= c)</i>	<i>false</i>	<i>0</i>
<i>!(a != b)</i>	<i>false</i>	<i>0</i>
<i>!(a != c)</i>	<i>true</i>	<i>1</i>

or

Operation	Result
<i>!(a &gt; b) &amp;&amp; (a &gt;= c)</i>	<i>false(0)</i>
<i>(a != b) &amp;&amp; (a &gt;= c)</i>	<i>true(1)</i>
<i>(a != b) &amp;&amp; !(a != b)</i>	<i>false(0)</i>

and

<i>!(a &gt; b)    (a &gt;= c)</i>	<i>true(1)</i>
<i>(a != b)    (a &gt;= c)</i>	<i>true(1)</i>
<i>(a != b)    !(a != b)</i>	<i>true(1)</i>
<i>!(a &gt;= c)    !(a != b)</i>	<i>false(0)</i>

## 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) &amp; (0000000001000110)

```
0000000010011100
      AND
0000000001000110
000000000000100    → 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)  $x1 = dat \ll 1;$

(2)  $x2 = dat \ll 2;$

**Solution :**

$dat = 0x93 \rightarrow 0000000010010011$

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 :  $a = (0000000001111010) \& (000000000111100)$

0000000001111010

AND

000000000111100

**000000000111000 → 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 :  $b = (000000000010110) |= (0000000001010001)$

000000000010110

OR

0000000001010001

**0000000001010111 → 0x0057 or 0x57**

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

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

0000000011111101

XOR

0000000011010000

**000000000101101 → 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 store 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

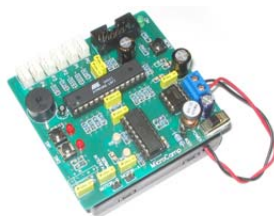
This chapter focus learning the applications of the MICROCAMP microcontroller. The building of a robot integrates knowledge and technology which includes electronics, programming, mechanical movements, and thinking process. The Microcamp Activity kit supports this concept. This kit includes all parts 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



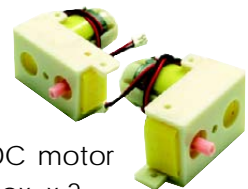
Circle base plate



MicroCamp board



Box holder x 1



48:1 DC motor gearbox x 2



Plastic spacer set x 1



Nut and Screw set x 1



Wheel and Tire set x 2



Ploastic joiners (Straight, Right angle and Obtuse)



Infrared reflector x 2



Swithc module x 2



2mm. Self-tapping screw x2



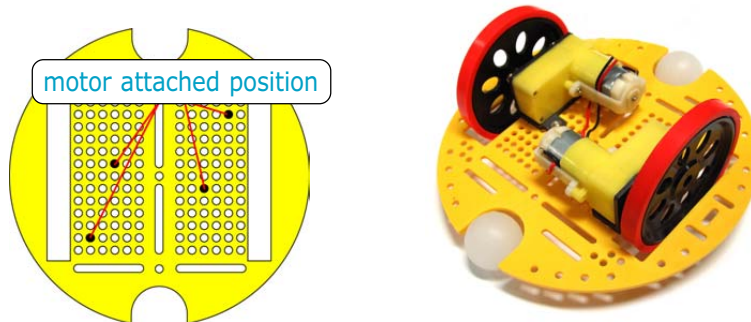
25mm. metal spacer x 2

## Construction

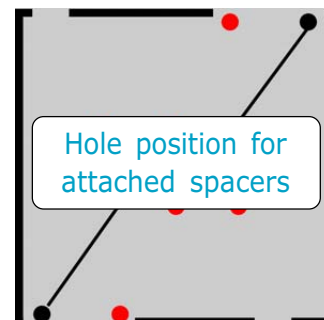
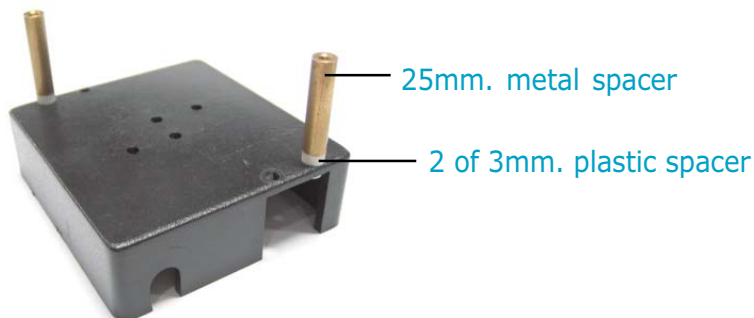
1. Fix on the 2 wheels with the rubber tires and attach them to the DC Gearbox with the 2 of the 2mm. self-tapping screws provided in the kit.



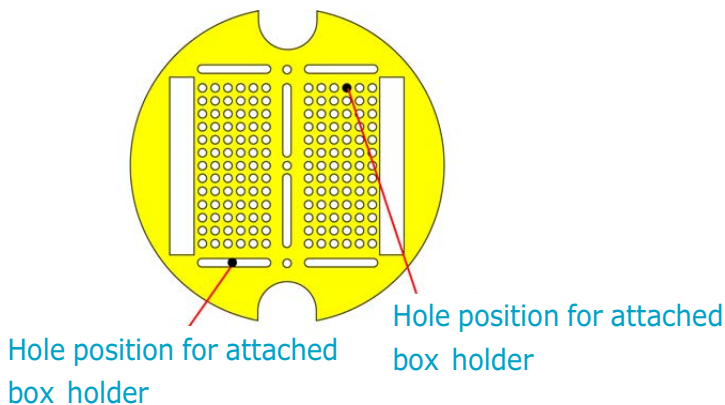
2. Install both the DC Gearboxes on the circular base plate at the specific positions shown in the picture with 4 of 3 x 6mm. machine screws.



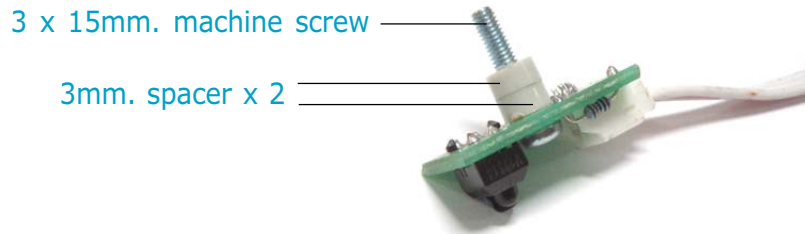
3. Insert the 3 x 10mm. machine screws through the hole at the corner of the Box holder with 25mm. and 2 of 3 mm. spacers.



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.

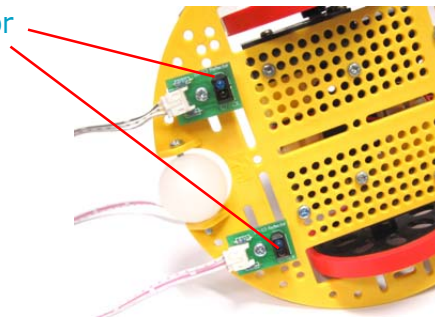
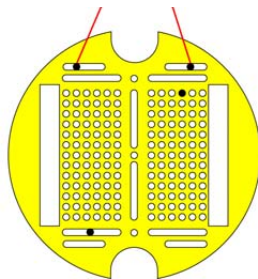


5. Insert a 3x15mm. machine screw through the Infrared Reflector sensor, followed by 2 of the 3mm. spacer. Do on both sides for this.

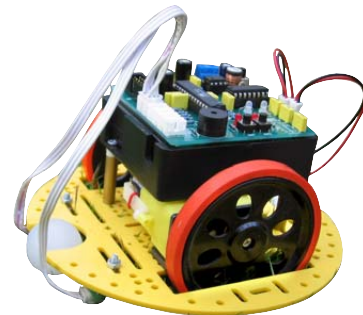
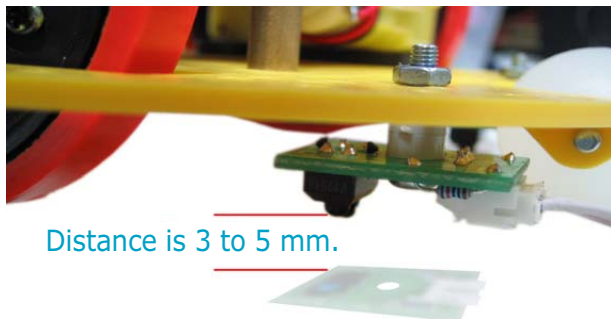


6. Attach both the Infrared Reflector structures from step 5 at the suitable holes at the bottom and front of the robot base. Tighten with a 3mm. nut.

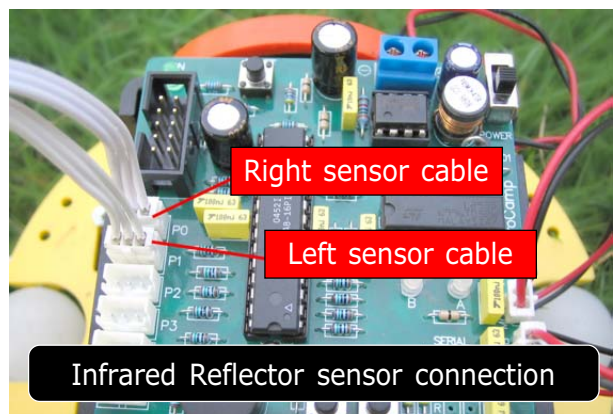
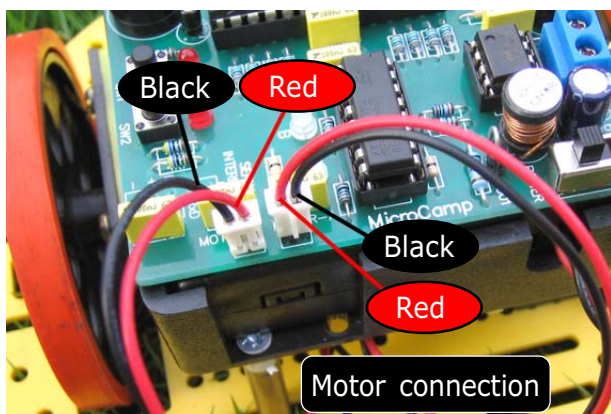
Hole position for attached Infrared Reflector sensor



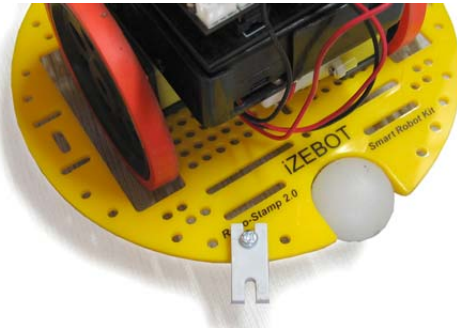
7. Observe the distance from the floor to the sensors. The suitable distance is 3 to 5 mm.



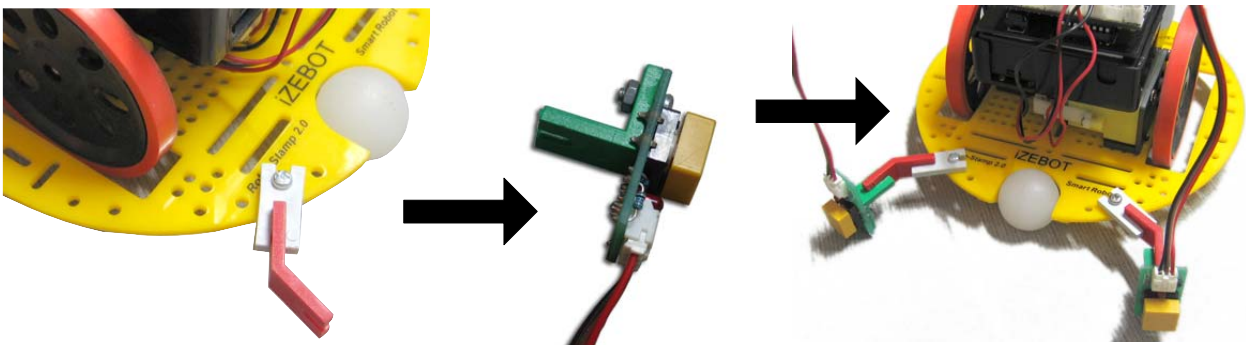
8. Place MicroCamp board on the box holder. Connect sensor cables and motor cables following the diagrams shown. (P0 for Right sensor and P1 for Left sensor).



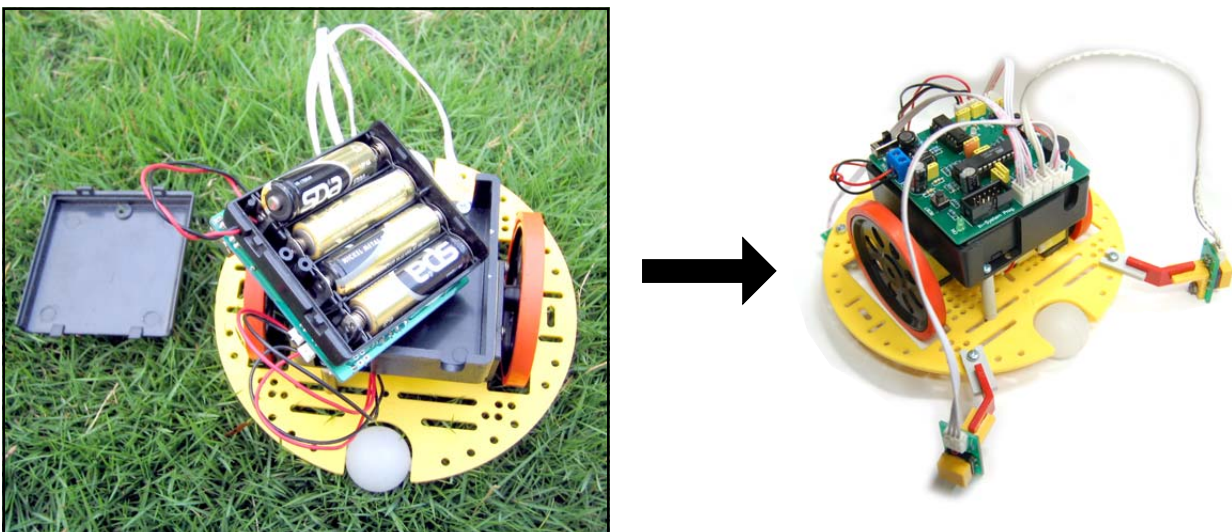
9. Attach the Straight joiner with robot base at front-right side by 3 x 10mm. machine screw and 3mm. nut. Attach 2 pieces.



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.

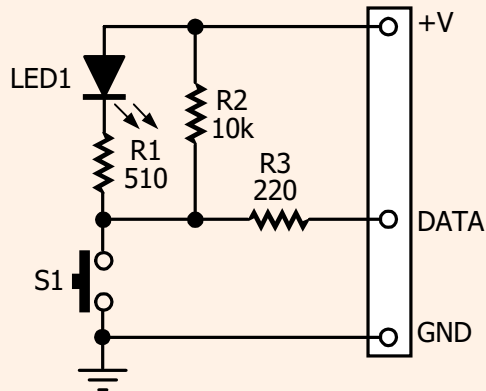


11. Connect the Left Switch module cable to the P2 (PC2) connector and the Right Switch module cable to the P3 (PC3) connector. Put 4 AA batteries into battery holder at the back of MicroCamp board. **The MicroCamp robot is ready for programming now.**



## Learning about the Switch circuit

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



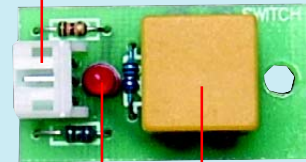
Pressing the switch results in two occurrences.

*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.

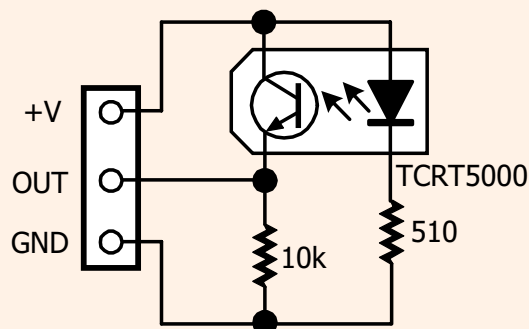
Signal connector



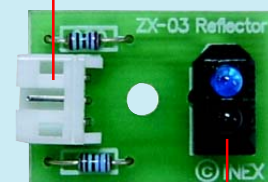
Indicator

Switch

## More information of Infrared Reflector



Signal connector



Infrared Reflector sensor

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. Download the HEX code to the robot.

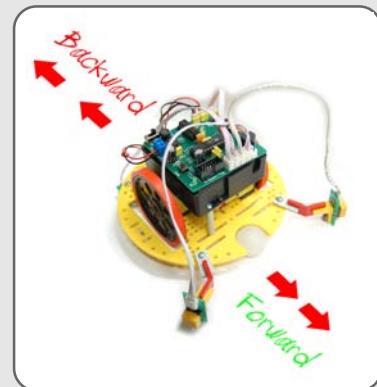
A1.3 Turn-off power and Remove the ISP cable.

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.*

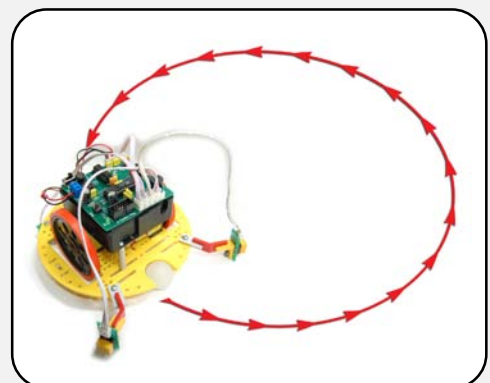
```
#include <in_out.h>
#include <sleep.h>
#include <motor.h>
void main()
{
    while(1)
    {
        while((in_d(2)==1));    // Loop for checking SW1 pressed
        motor(1,100);           // Apply full power for Motor 1
        motor(2,30);            // Apply 30% power for Motor 2
        while((in_d(3)==1));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

### Program description

In Listing A1-2, the forward and backward commands 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.**

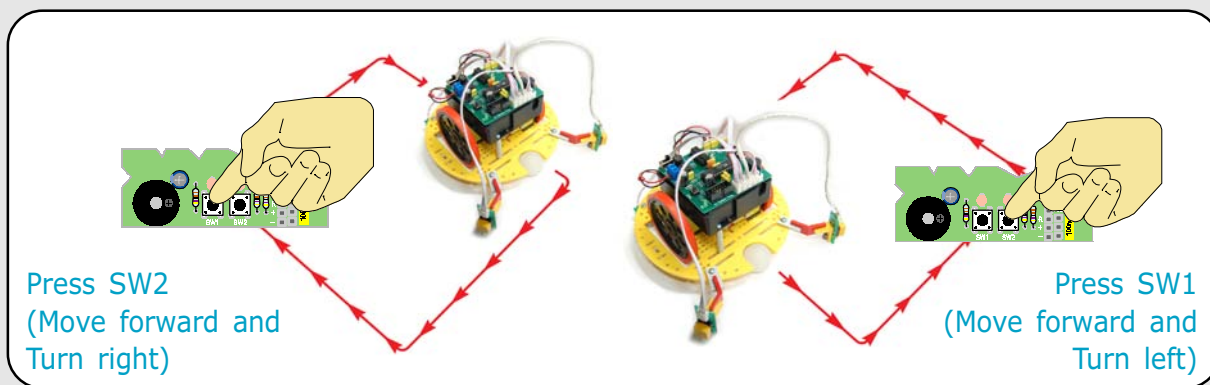
## Activity 1-3 Square-shape movement control

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.***

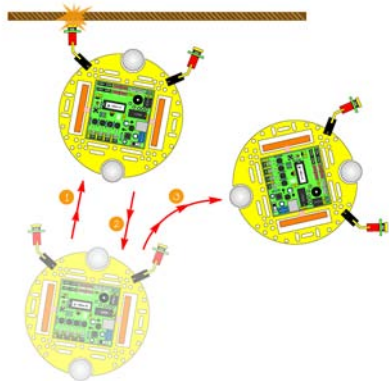
```

#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 continuous 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)

# Activity 3

## Line tracking robot

---

From the 2 first activities, these show how to read the digital input signal and to get the data to control robot movement. In this activity, there will be many activities about reading analog inputs and processing the data to detect black and white areas. It also detects Black and white line to control the robot to move along the line with 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 1023 in decimal number format.

C programming for this activity require a library file. It 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 1023 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.

### Activity 3-1 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 color surface.

#### (A) White surface testing

This sub-activity presents how to detect the white surface. The Listing A3-1 is C program for testing Infrared Reflector operation. After execution, the program will wait for SW1 or SW2 pressing.

*If press SW1 : select to read data from P0 or PC0 analog port*

*If press SW2 : select to read data from P1 or PC1 analog port*

After pressing the switch, the program will read data at the port pin continuously and will not switch to read another sensor unless the RESET button is pressed. Developers must press both switches to get the sensor's data.

```

#include <in_out.h>
#include <sound.h>
#include <analog.h>           // Analog reading library
void main()
{
    while(1)                  // Loop for waiting the key selection to
    {                          // read P0 or P1
        if ((in_d(2)==0))     // Check SW1 pressing
        {
            while(1)          // Repeat this loop
            {
                if (analog(0)>350) // Read the value from P0 and check the
                {                // white surface.
                    sound(3000,100); // If the white surface is deteced,
                                    // drive the sound to speaker.
                }
            }
        }
        if ((in_d(3)==0))     // Check SW2 pressing
        {
            while(1)          // Repeat this loop
            {
                if (analog(1)>350) // Read the value from P1 and check the
                {                // white surface.
                    sound(3000,100); // If the white surface is deteced,
                                    // drive the sound to speaker.
                }
            }
        }
    }
}

```

### Listing A3-1 The C program for MicroCamp robot in White surface testing activity

Compare the sensor's data with the reference data; 350.

**If data values are more than 350**, the color that is detected is white color.

**If data values are less than 350**, the color that is detected is black color.

After detect the white surface ready, program will send the sound signal to piezo speaker.

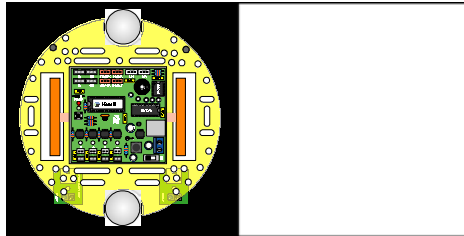
A3.1 Make the black & white testing sheet. The white surface area is 30 x 30 cm. and black surface is 30 x 30cm.

A3.2 Create the new project file and make the C program following the Listing A3-1. Build this project file.

A3.3 Connect the PX-400 programmer box with MicroCamp robot and download the HEX code to the robot.

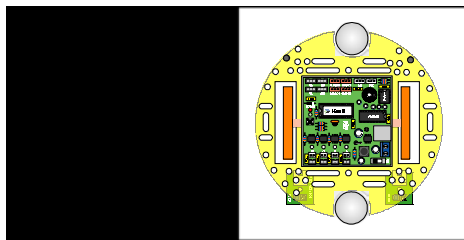
A3.4 Place the robot on the black surface first. Turn on the power and press SW1 switch.

*The Robot does not move.*



A3.5 Place the robot on the white surface and try to roll the robot.

*The robot produces sounds when its on a white surface.*



A3.6 Press the RESET switch. Place the robot on the black surface again. Press switch SW2 to test the operation of the Infrared Reflector at P1 port. Observe the robot's operation.

*After press SW2, robot will get data from sensors at P1 port and compare the reference value ; 350. **If the reading value more than 350**, means the robot detects the white area. It send sound signal to drive a piezo speaker. From step A3.6, robot detects the black surface then do not work anything.*

A3.7 Place the robot on the white surface and try to roll the robot.

*Robot drives sound always above the white surface.*

A3.8 If the robot cannot drive the signal when placed on the white surface in testing. The solution is

- (1) Decrease the reference value from 350 but not lower 100
- (2) Adjust the sensor position to decrease the distance from the floor.

A3.9 The Listing A3-2 is C program for testing the black surface. Developers can test with this program to check the black surface detection of robot to make sure the robot can detect white and black surface well. The procedure is same in step A3.4 to A3.8. But the decision criteria will change from higher 350 to lower 350 instead.

```

#include <in_out.h>
#include <sound.h>
#include <analog.h>           // Analog reading library
void main()
{
    while(1)                  // Loop for waiting key selection to read
    P0 or P1
    {
        if((in_d(2)==0))      // Check SW1 pressing
        {
            while(1)           // Repeat this loop
            {
                if (analog(0)<350) // Read the value from P0 and check the
                // black surface.
                {
                    sound(3000,100); // If the black surface is detected,
                    // drive the sound to speaker.
                }
            }
        }
        if((in_d(3)==0))      // Check SW2 pressing
        {
            while(1)           // Repeat the loop
            {
                if (analog(1)<350) // Read the value from P1 and check the
                // black surface.
                {
                    sound(3000,100); // If the black surface is detected,
                    // drive the sound to speaker.
                }
            }
        }
    }
}

```

**Listing A3-2** The C program for MicroCamp robot check the Black surface activity

## Activity 3-2 Robot moves along the black line

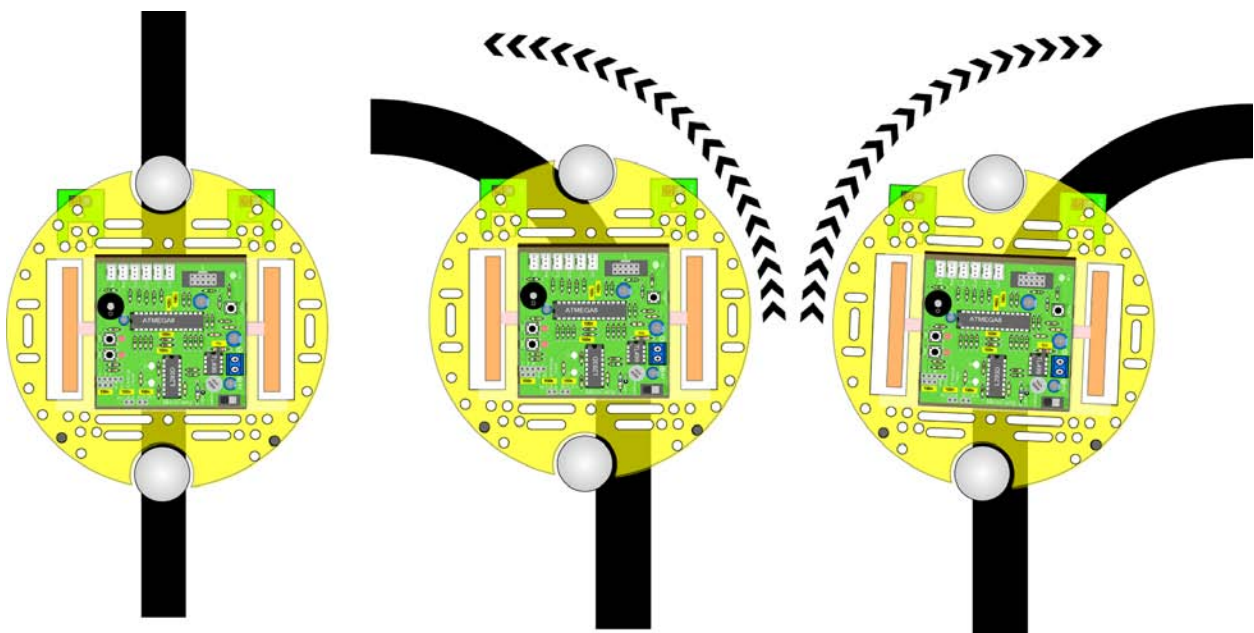
The robot moving along the line has 3 scenarios.

(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, can make the C program in the Listing A3-3.



**Scenario-1**

Both sensors put across

**Scenario-2**

Left sensor detects the line

**Scenario-3**

Right sensor detects the line

A3.10 Make the simple black line sheet. It has not the cross line. Most area is white color. Size of sheet can determine suitable for your robot.

A3.11 Create the new project file and make the C program following the Listing A3-3. Build this project file.

A3.12 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.

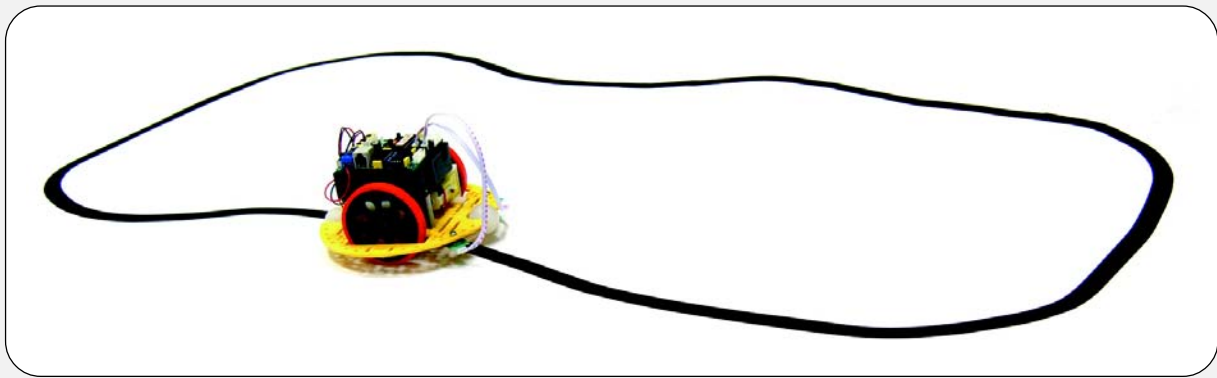
```

#include <in_out.h>
#include <sound.h>
#include <analog.h>
#include <motor.h>                                // Motor driver library
unsigned int AD0=350,AD1=350;                     // Determine the sensor reference
                                                    // value.

void main()
{
    while((in_d(2)==1));                          // Wait pressing SW1 to start the
                                                    // program

    while(1)
    {
        if ((analog(0)>AD0)&&(analog(1)>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 A3-3** The C program for controlling the MicroCamp robot to moves along the black line

A3.13 Place the robot across the black line on the sheet. Turn on power and press SW1 switch.

*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 3-3 Line crossing detection

From the activity 3-2, you can improve the MicroCamp robot so that it moves along the black line and detect the junction or line with same 2 sensors. One thing to do is edit the program.

When the robot moves to line crossing, both sensors will detect black line. You must add the program for support this scenario. The improved C program is shown in the Listing A3-4

A3.14 Improve the simple black line sheet from Activity 3-2. Add some cross lines. The number will depend on your inquiry.

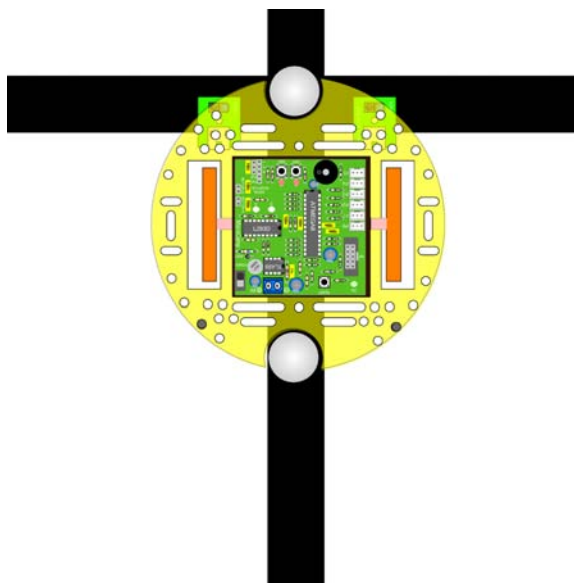
A3.15 Create the new project file and make the C program following the Listing A3-4. Build this project file.

A3.16 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.

A3.17 Place the robot across the black line on the sheet. Turn on power and press SW1 switch.

*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.*

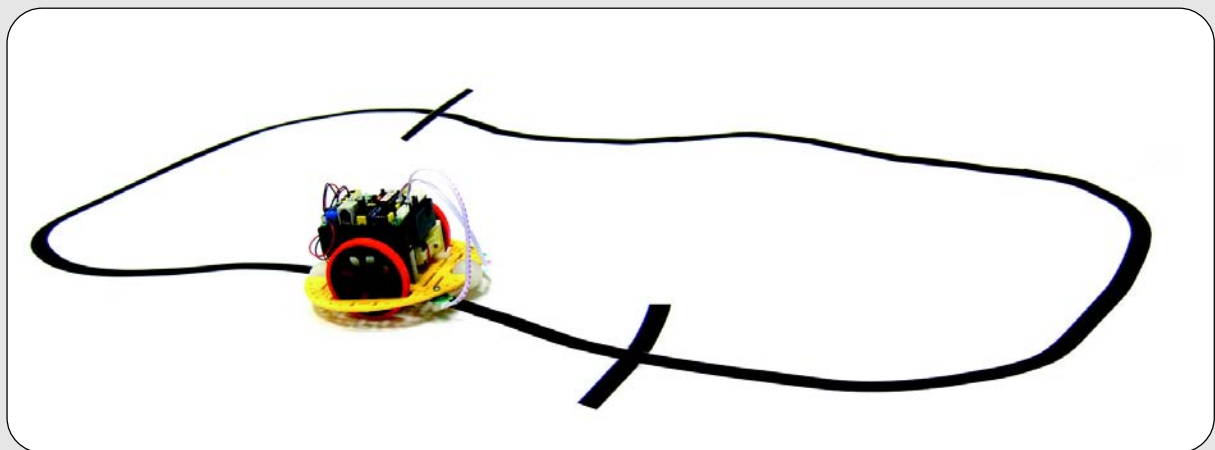
**Note :** In the motor brake operation, robot will stop and lock the motor's shaft immediately. But sometimes, this is not enough. You must program the robot to move backwards for a short time. This will cause the robot to stop at its position.



```

#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(0)<AD0)&&(analog(1)<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
                        // line-crossing detection
            {
                sound(2500,100);
                sleep(50);
            } // Drive sound
            forward(100); // Move forward to cross over
                        // the line.
            sleep(300);
        }
        if ((analog(0)>AD0)&&(analog(1)>AD1)) // Both sensor detect 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 A3-4 The C program for controlling the MicroCamp robot to moves along the black line and detect the line-crossing.

# MicroCamp 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()                   // Start Blinking LED2
{
    interval_init();
    LED |= (1<<1) ;
}

void led2_off()                  // Stop Blinking LED2
{
    LED &= ~_BV(1);
}
```

## 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,1MC=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); // Prescaler64,16MHz,1MC=64/16M=4us/count
}

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

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

## **COPYRIGHTS**

This documentation is copyright 2006-2007 by Innovative Experiment Co., Ltd. (INEX) By downloading or obtaining a printed copy of this documentation or software you agree that it is to be used exclusively with INEX products. Any other uses are not permitted and may represent a violation of INEX copyrights, legally punishable according to Federal copyright or intellectual property laws. Any duplication of this documentation for commercial uses is expressly prohibited by INEX. Duplication for educational use is permitted, subject to the following Conditions of Duplication:

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

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