FIRE BIRD V

ATMEGA2560
ROBOTIC RESEARCH PLATFORM
Software Manual

Designed By:

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

NEX ROBOTICS
www.nex-robotics.com

Manufactured By: NEX Robotics Pvt. Ltd.

FIRE BIRD V

SOFTWARE MANUAL

Version 2.00
August 15th, 2012

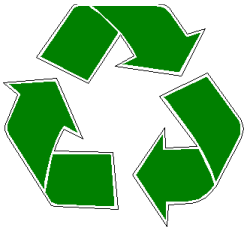
Documentation author
Sachitanand Malewar, NEX Robotics Pvt. Ltd.
Dr. Anant Malewar, NEX Robotics Pvt. Ltd.

Credits (Alphabetically)
Aditya Sharma, NEX Robotics
Amey Apte, NEX Robotics
Amit Yadav, NEX Robotics
Ashish Gudhe, CSE, M.Tech, IIT Bombay
Behlul Sutarwala, NEX Robotics
Gaurav Lohar, NEX Robotics
Gurulingesh R. CSE, M.Tech, IIT Bombay
Inderpreet Arora, EE, M.Tech, IIT Bombay
Prof. Kavi Arya, CSE, IIT Bombay
Prof. Krithi Ramamritham, CSE, IIT Bombay
Kunal Joshi, NEX Robotics
Nandan Salunke, RA, CSE, IIT Bombay
Pratim Patil, NEX Robotics
Preeti Malik, RA, CSE, IIT Bombay
Prakhar Goyal, CSE, M.Tech, IIT Bombay
Raviraj Bhatane, RA, CSE, IIT Bombay
Rohit Chauhan, NEX Robotics
Rajanikant Sawant, NEX Robotics
Saurabh Bengali, RA, CSE, IIT Bombay
Vaibhav Daghe, RA, CSE, IIT Bombay
Vibhooti Verma, CSE, M.Tech, IIT Bombay
Vinod Desai, NEX Robotics

Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, NEX Robotics welcomes your corrections. You can send us your queries / suggestions at info@nex-robotics.com

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



Recycling:

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

Important:

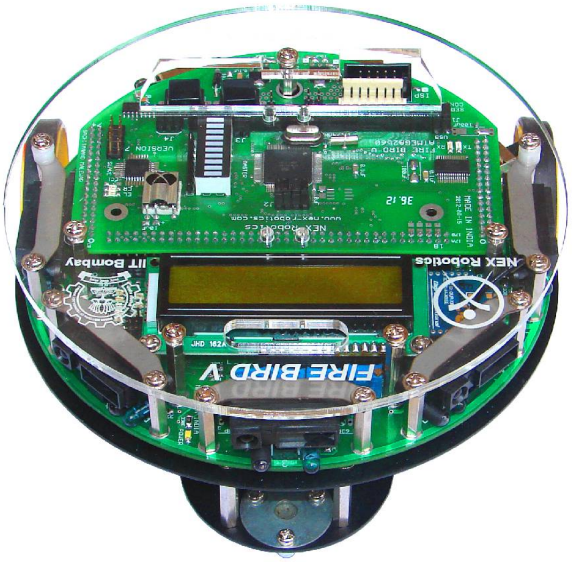
1. User must go through the Fire Bird V's Hardware and Software manuals before using the robot.

2. Crystal of the ATMEGA2560 microcontroller is upgraded to 14.7456MHz from 11.0592 MHz in all the Fire Bird V ATMEGA2560 robots delivered on or after 1st December 2010. This documentation is made considering crystal frequency as 14.7456MHz.

Index

1. Fire Bird V ATMEGA2560

The Fire Bird V robot is the 5th in the Fire Bird series of robots. First two versions of the robots were designed for the Embedded Real-Time Systems Lab, Department of Computer Science and Engineering, IIT Bombay. Theses platforms were made commercially available form the version 3 onwards. All the Fire Bird V series robots share the same main board and other accessories. Different family of microcontrollers can be added by simply changing top microcontroller adaptor board. Fire Bird V supports ATMEGA2560 (AVR), P89V51RD2 (8051) and LPC2148 (ARM7) microcontroller adaptor boards. This modularity in changing the microcontroller adaptor boards makes Fire Bird V robots very versatile. User can also add his own custom designed microcontroller adaptor board.



Fire Bird V ATMEGA2560 (AVR)

Fire Bird V P89V51RD2 (8051)

Fire Bird V LPC2148 (ARM7 TDMI)

Figure 1.1: Fire Bird V Robots

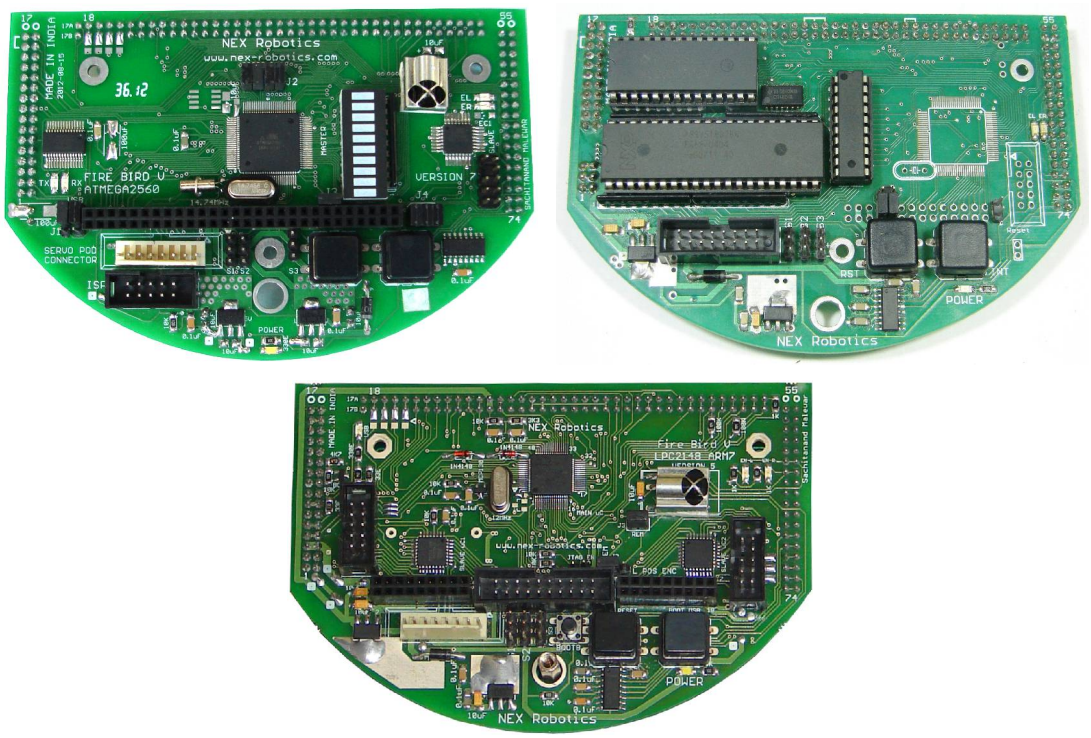


Figure 1.2: ATMEGA2560 (AVR), P89V51RD2 (8051) and LPC2148 ARM7 microcontroller adaptor boards for Fire Bird V

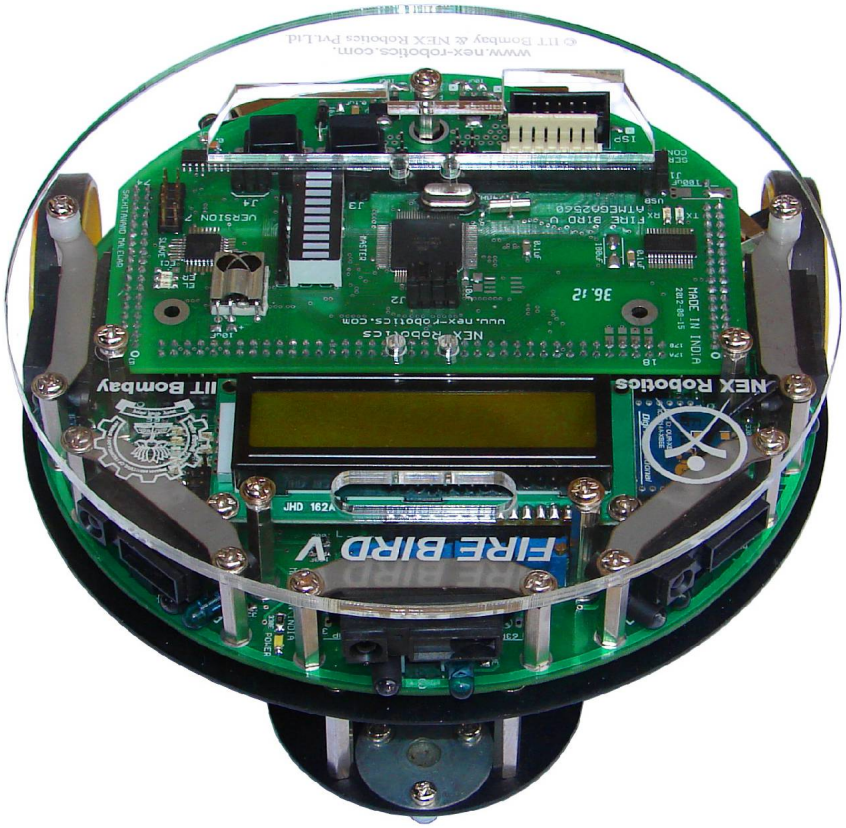
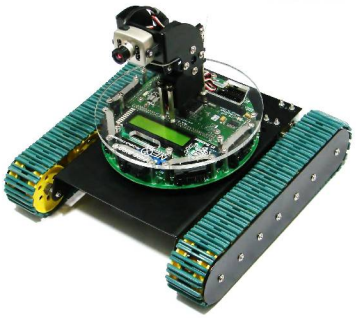


Figure 1.3 Fire Bird V ATMEGA2560 robot

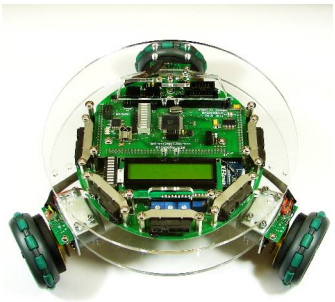
1.1 Avatars of Fire Bird V Robot

All Robots use the same main board and microcontroller adaptor board. All Fire Bird V Robots share the same unified architecture.

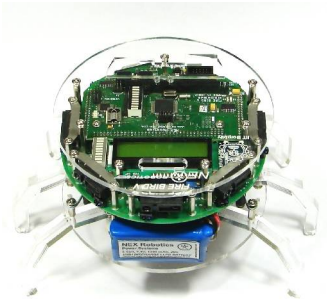
Fire Bird V



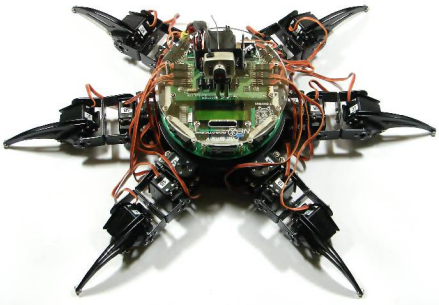
Fire Bird V Tank



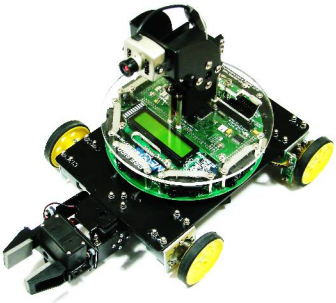
Fire Bird V Omnidirectional Robot



Fire Bird V Insect



Fire Bird V Hexapod



Fire Bird V 4WD with Gripper

Figure 1.4: Avatars of Fire Bird V Robot

1.2 Fire Bird V Block Diagram:

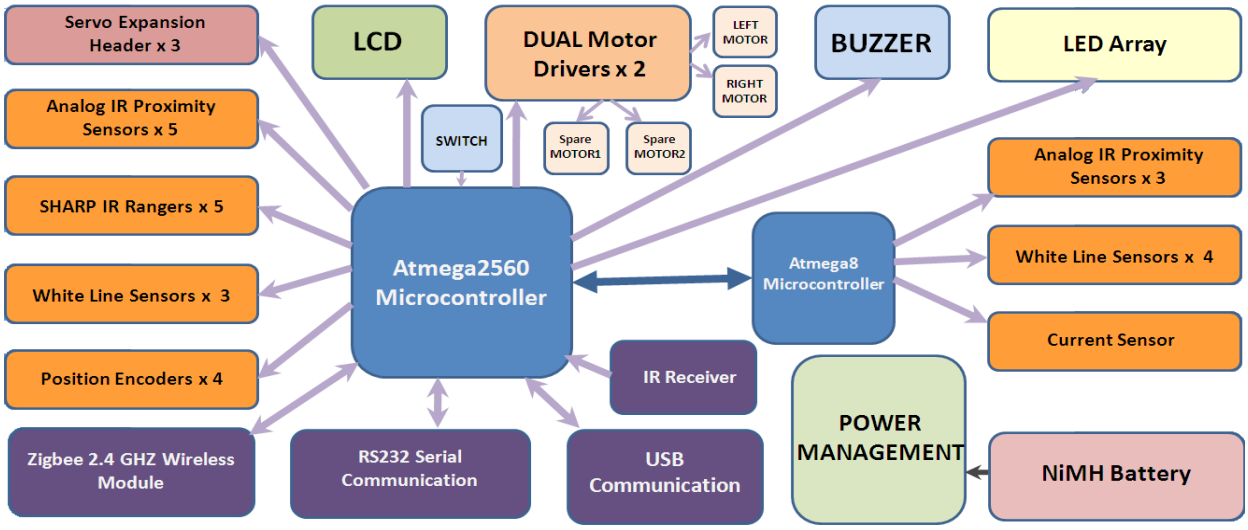


Figure 1.5: Fire Bird V ATMEGA2560 robot block diagram

1.3 Fire Bird V ATMEGA2560 technical specification

Microcontroller:
Atmel ATMEGA2560 as Master microcontroller (AVR architecture based Microcontroller)
Atmel ATMEGA8 as Slave microcontroller (AVR architecture based Microcontroller)

Sensors:
Three white line sensors (extendable to 7)
Five Sharp GP2Y0A02YK IR range sensor (One in default configuration)
Eight analog IR proximity sensors
Two position encoders (extendable to four)
Battery voltage sensing
Current Sensing (Optional)
Five MaxBotix Ultrasonic Range Sensors (Optional)

Indicators:
2 x 16 Characters LCD
Buzzer and Indicator LEDs

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

Communication:
USB Communication
Wired RS232 (serial) communication
Wireless ZigBee Communication (2.4GHZ) (if XBee wireless module is installed)
Wi-fi communication
Bluetooth communication
Simplex infrared communication (From infrared remote to robot)

Dimensions:
Diameter: 16cm
Height: 8.5cm
Weight: 1100gms

Power:
9.6V Nickel Metal Hydride (NiMH) battery pack and external Auxiliary power from battery charger.
On Board Battery monitoring and intelligent battery charger.

Battery Life:
2 Hours, while motors are operational at 75% of time

Locomotion:
Two DC geared motors in differential drive configuration and caster wheel at front as support
Top Speed: 24 cm / second
Wheel Diameter: 51mm
Position encoder: 30 pulses per revolution
Position encoder resolution: 5.44 mm

2. Programming the Fire Bird V ATMEGA2560 Robot

There are number of IDEs (Integrated Development Environment) available for the AVR microcontrollers. There are free IDEs which are based on AVR GCC like AVR Studio from ATMEL and WIN AVR and proprietary IDEs like ICC AVR, Code vision AVR, IAR and KEIL etc. IDEs like ICC AVR and code vision AVR are very simple to use because of their GUI based code generator which gives you generated code. Almost all the proprietary IDEs works as full version for first 45 days and then there code size is restricted to some size. We have used AVR Studio from ATMEL which is feature rich free to IDE for the robot. In this manual we are going to focus on the AVR studio from the ATMEL. It uses WIN AVR open source C compiler at the back end. It has many attractive features like built-in In-Circuit Emulator and AVR instruction set simulator. After writing and compiling the program it gives ".hex" file. This ".hex" file needs to be loaded on the robot using In System Programmer (ISP).

IDE Installation

Since AVR studio uses WIN AVR compiler at the back end we need to install WIN AVR first before installing AVR studio. By doing so AVR Studio can easily detect the AVRGCC plug-ins.

2.1 Installing WIN AVR

Copy "WIN AVR 2009-03-13" from the "Software and Drivers" folder of the documentation CD on the PC and click on WinAVRxxxx.exe file.

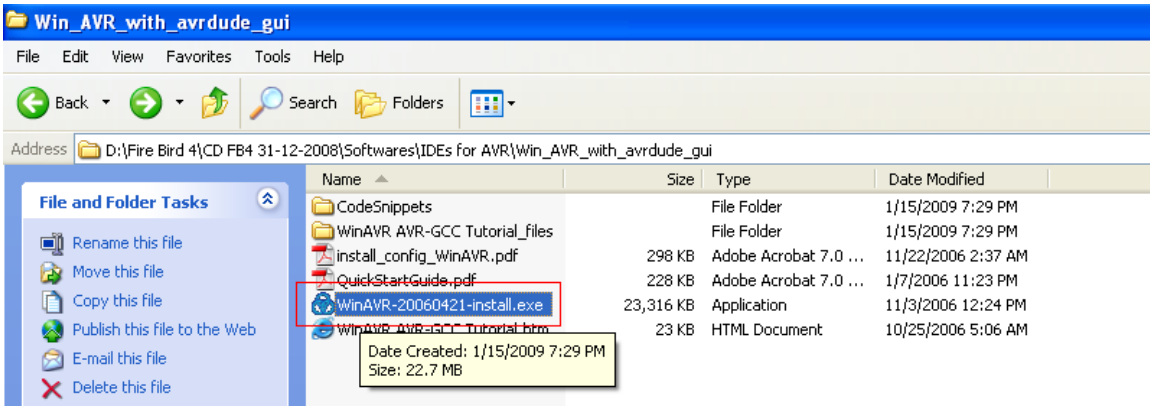


Figure 2.1

The following window with WIN AVR installation package will open. Choose language as "English".

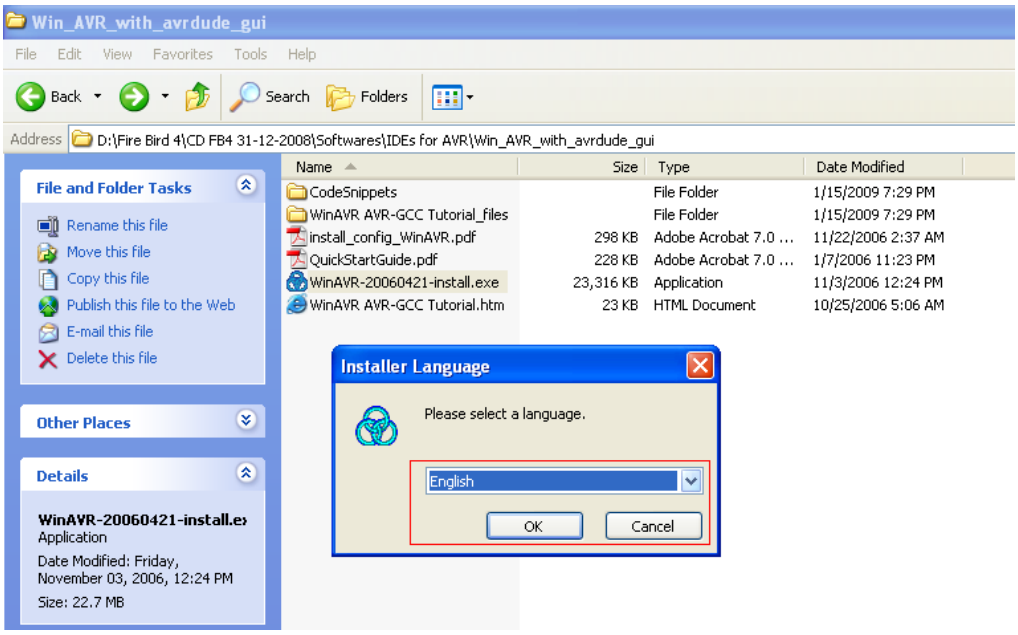


Figure 2.2

Click next in the WIN AVR setup wizard.

Figure 2.3

Press “I Agree” after going through license agreement.

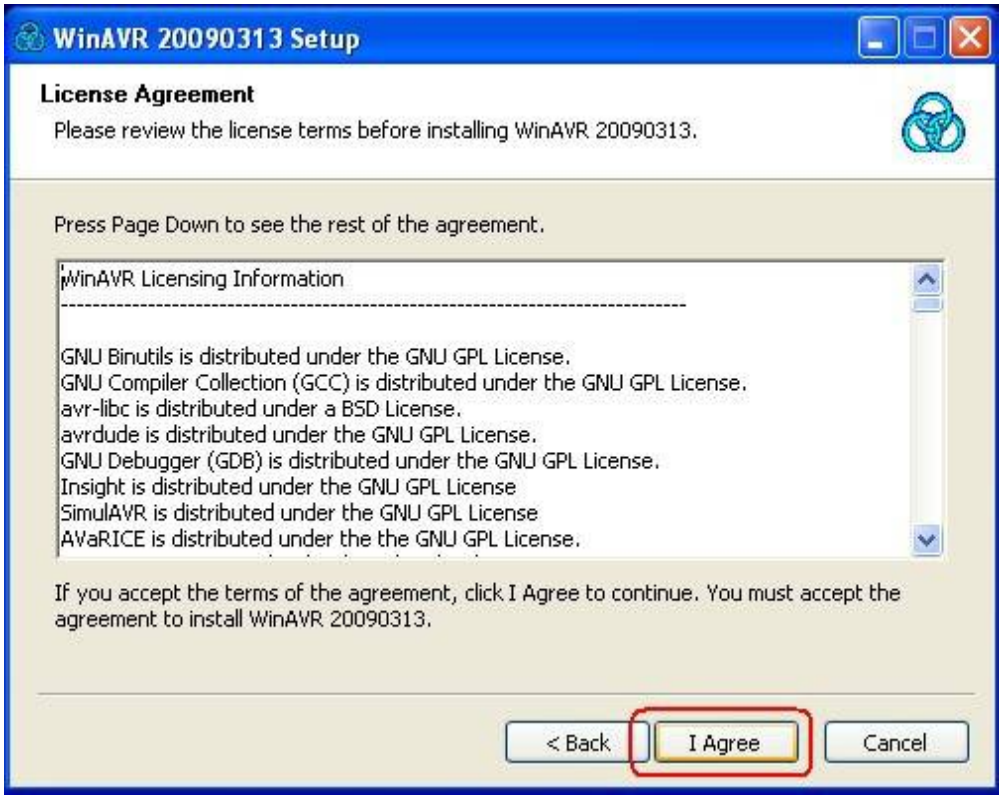


Figure 2.4

Make sure that you have to select the drive on which operating system is installed.

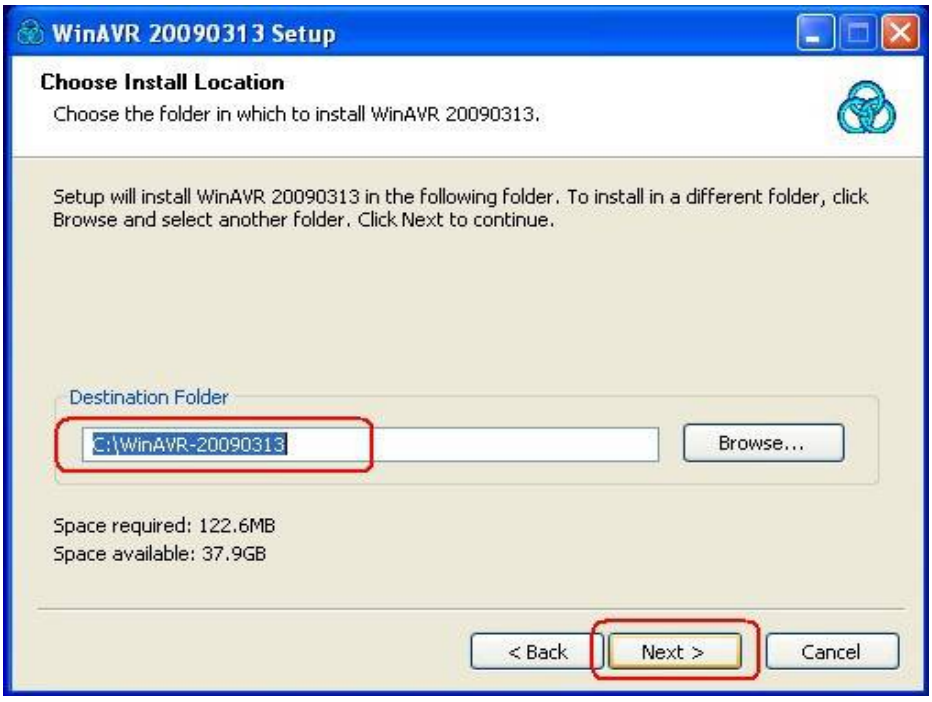


Figure 2.5

Select all the components and press “Install”.

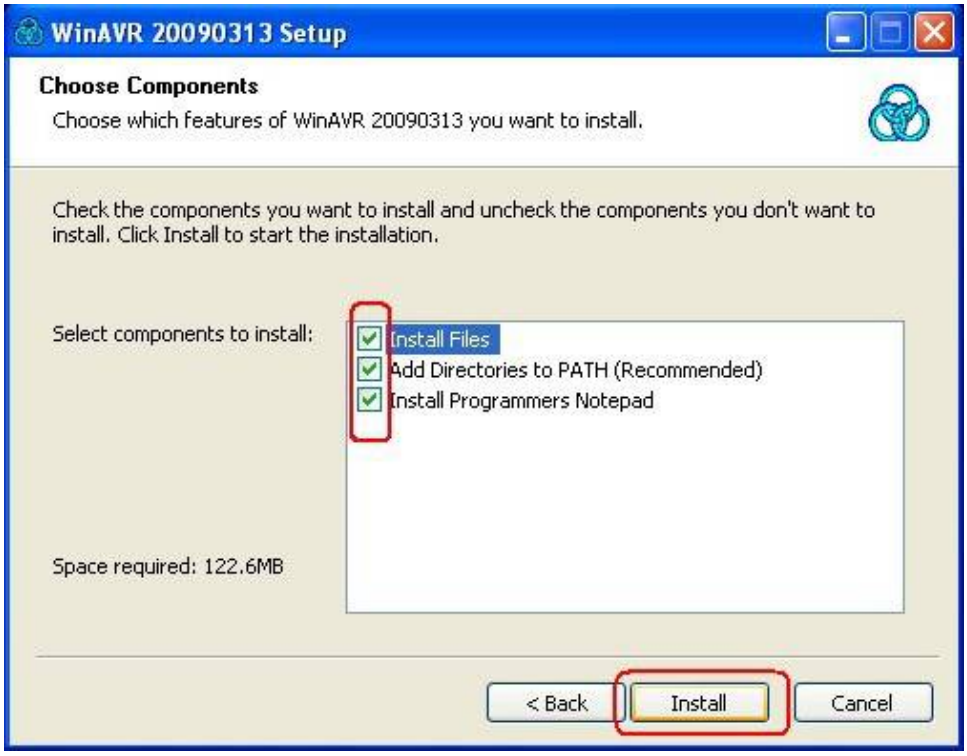


Figure 2.6

Click “Finish” to complete WIN AVR installation

Figure 2.7

2.2 Installing AVR Studio

Go to “Software and Drivers” folder from the documentation CD, copy folder “AVR Studio 4.17” on the PC and click on “AvrStudio417Setup.exe” to start the installation process.

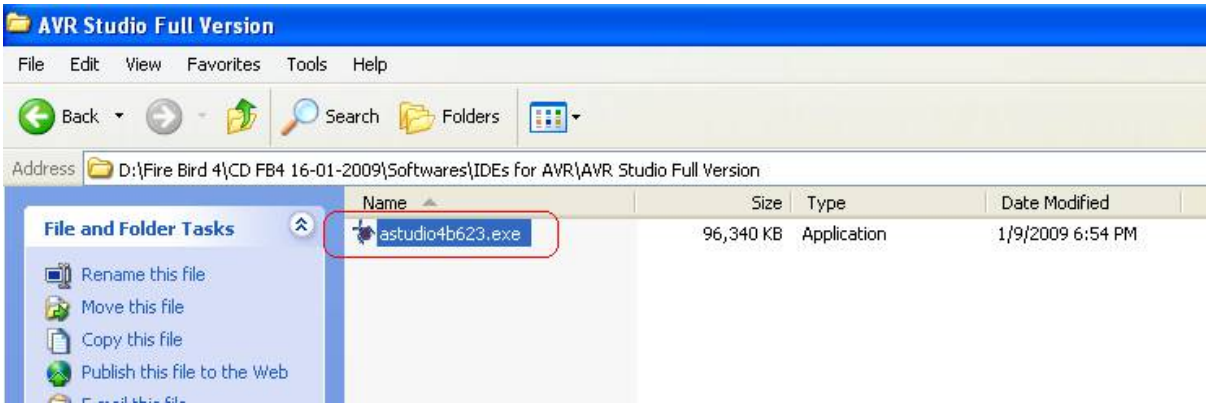


Figure 2.8

Click on “Run”

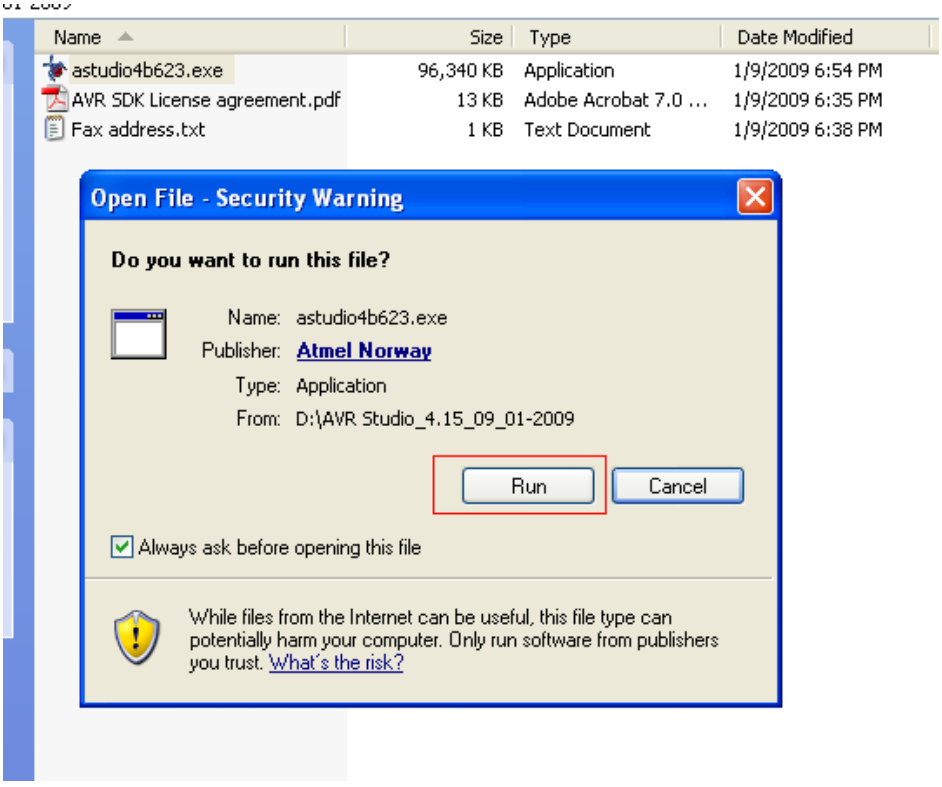


Figure 2.9

Click “Next” to start installation of AVR Studio 4

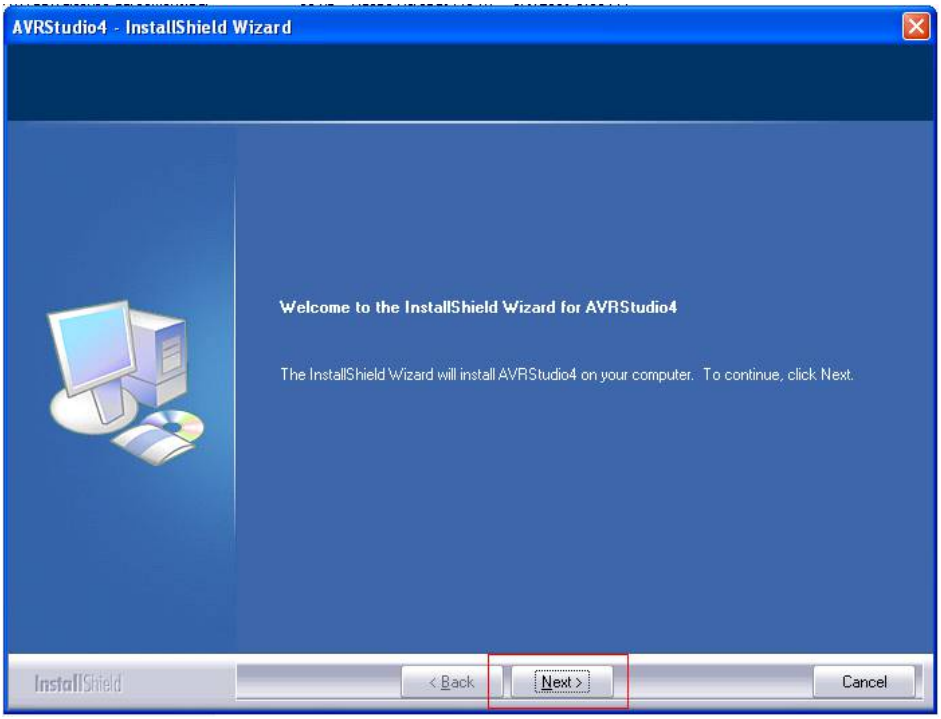


Figure 2.10

After clicking “Next”, go through the license agreement. If it is acceptable then click “Next”

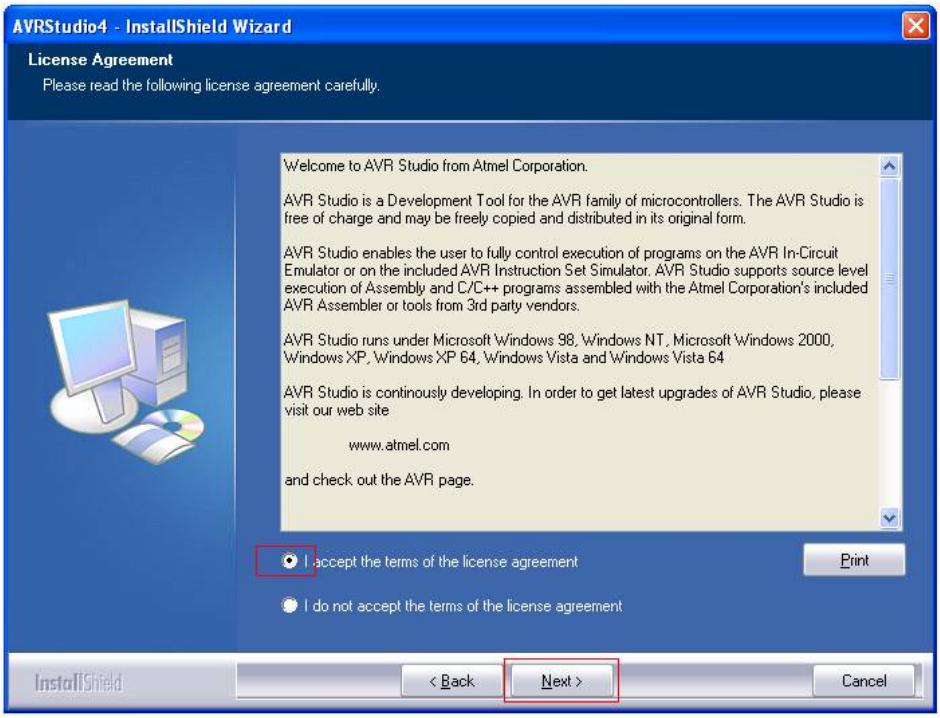


Figure 2.11

Now choose the destination drive. Select the same drive in which your operating system and WINAVR is installed.

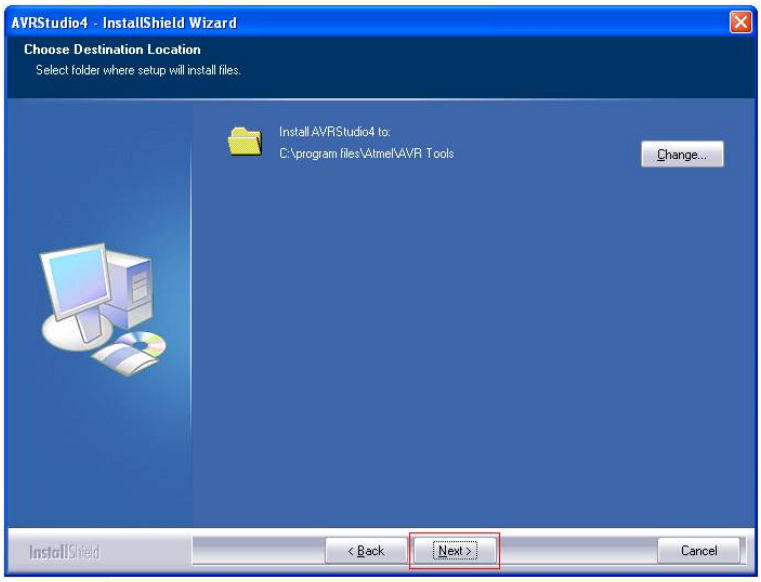


Figure 2.12

Select for the "Install / upgrade Jungo USB Driver" to support In System Programming (ISP) by AVRISP mkII

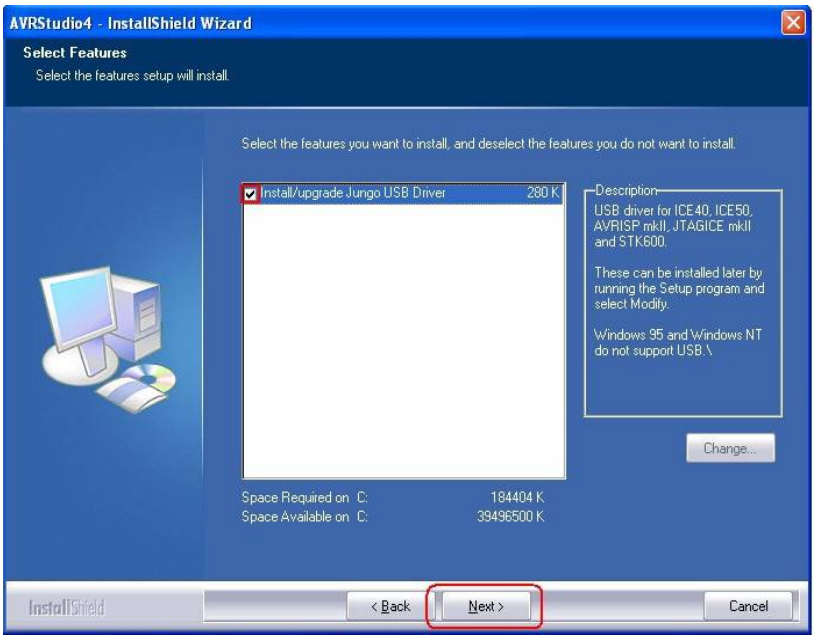


Figure 2.13

Important: If "Install / upgrade Jungo USB Driver" is not selected then AVRISP mkII programmer will not work with the AVR Studio.

Click “Next” to start the installation process.

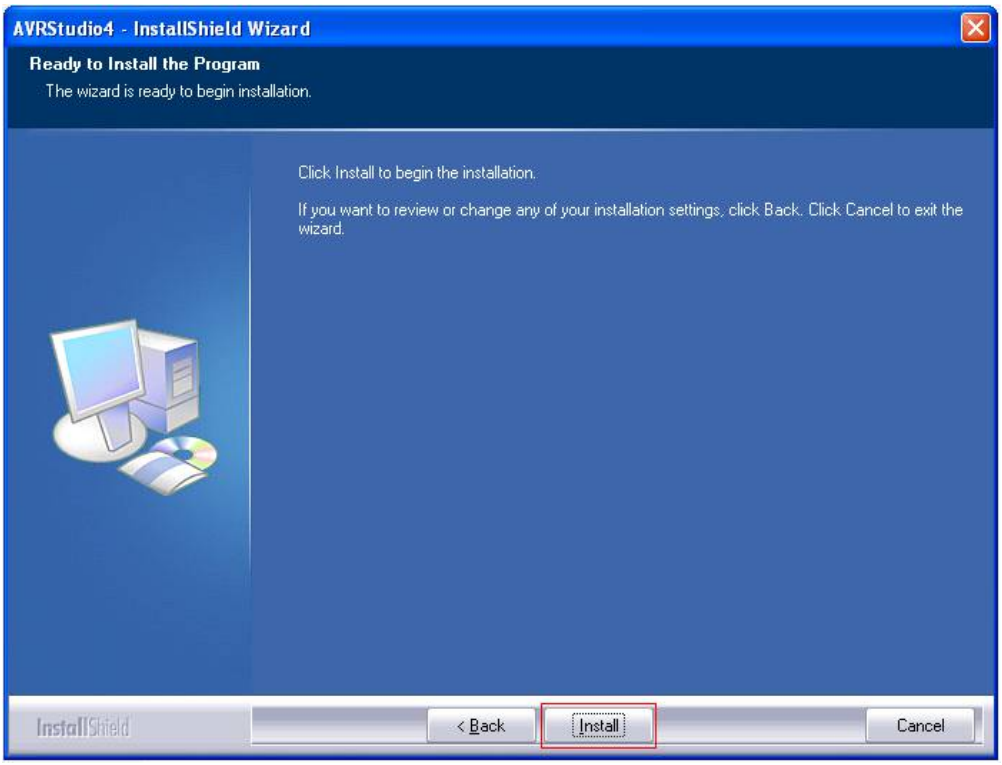


Figure 2.14

Click “Finish” to complete the installation process.

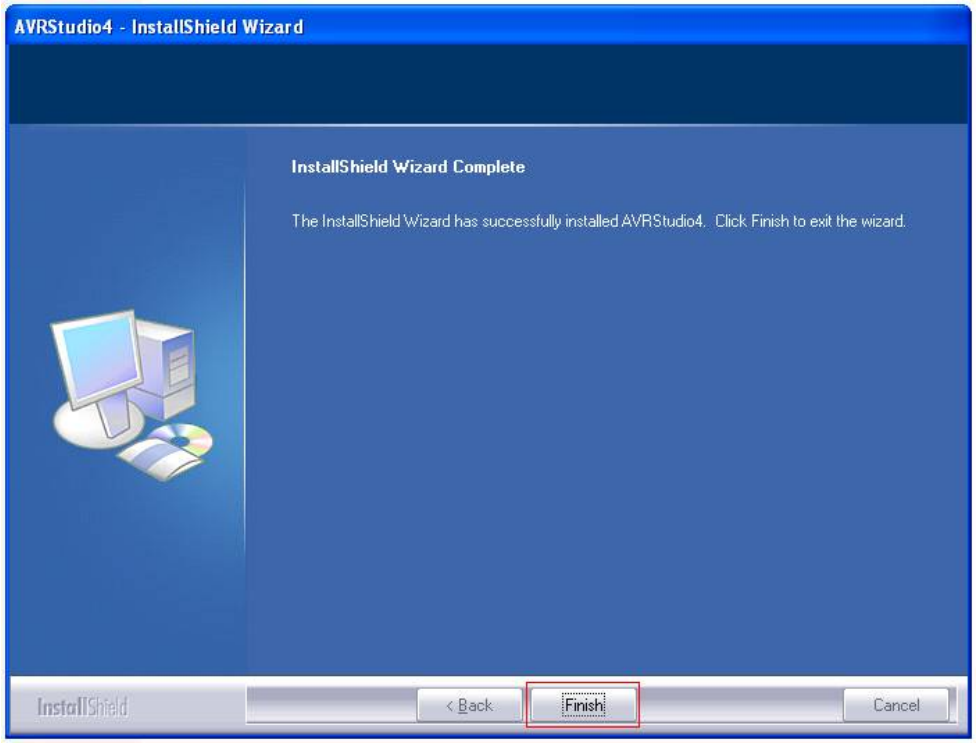


Figure 2.15

2.3 Setting up Project in AVR Studio

AVR studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications. As a code writing environment, it supports includes AVR Assembler and any external AVR GCC compiler in a complete IDE environment.

AVR Studio gives two main advantages:

1. Edit and debug in the same application window. Faster error tracking.
2. Breakpoints are saved and restored between sessions, even if codes are edited.

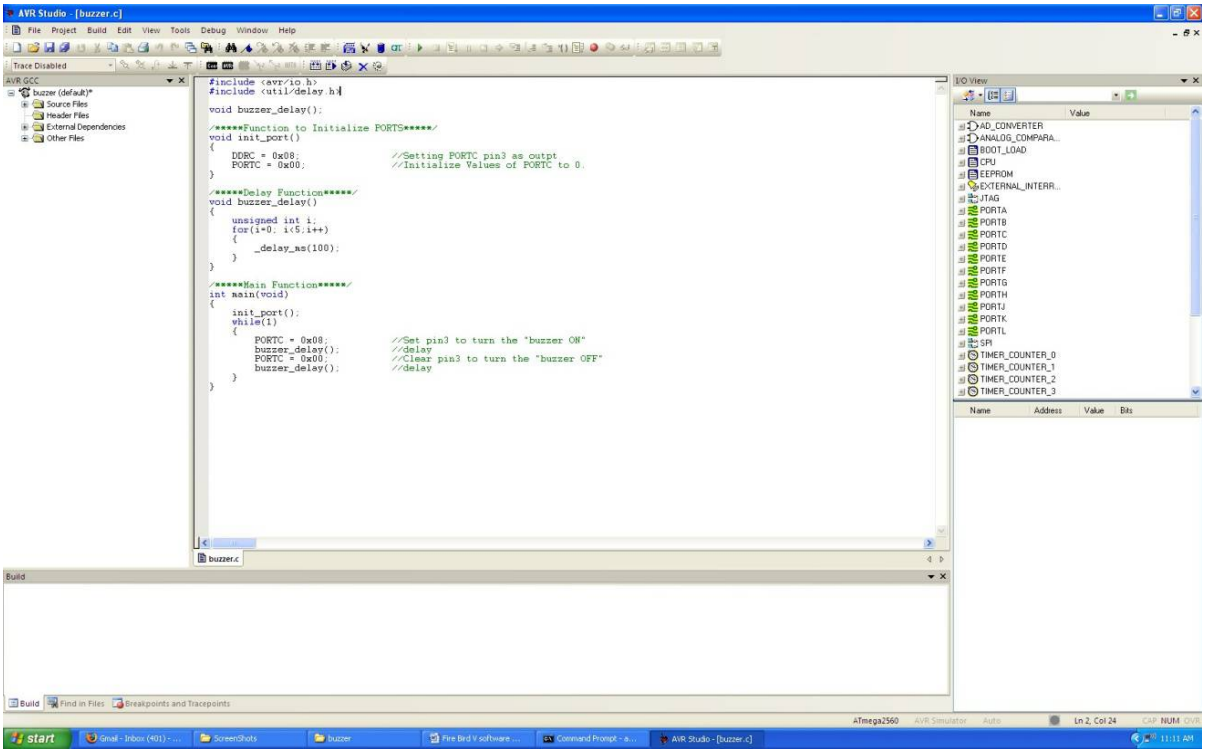


Figure 2.16

Middle window shows current code under development. Window on the left side shows view of source files, header files, External dependencies, and other files. Right side window shows all the ports and other peripheral's status. Bottom window is known as Build window. It shows results of the compilation, errors, HEX file size and other warning messages etc.

1. Open AVR Studio. If any project is running it can be closed by clicking on Project in the menu bar and select Close Project.

2. To create a new project, click on Project in the menu bar and select “New Project”.

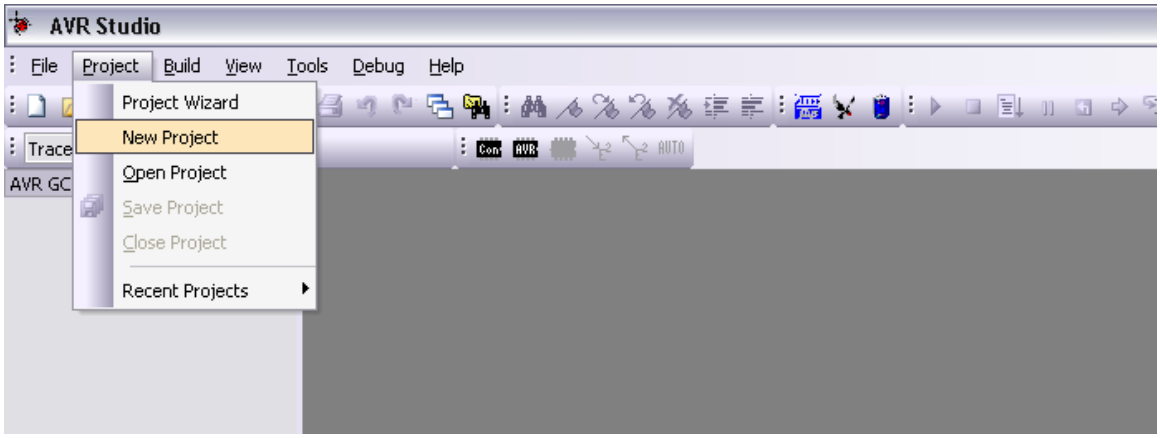


Figure 2.17

3. Select Project Type as “AVR GCC”. Type project name in the “Project name” window. In this case it is “buzzer_test”. Also check on Create initial file and Create folder check box. This will create all the files inside the new folder. In the Location window select the place where would like to store your project folder and then click “Next”.

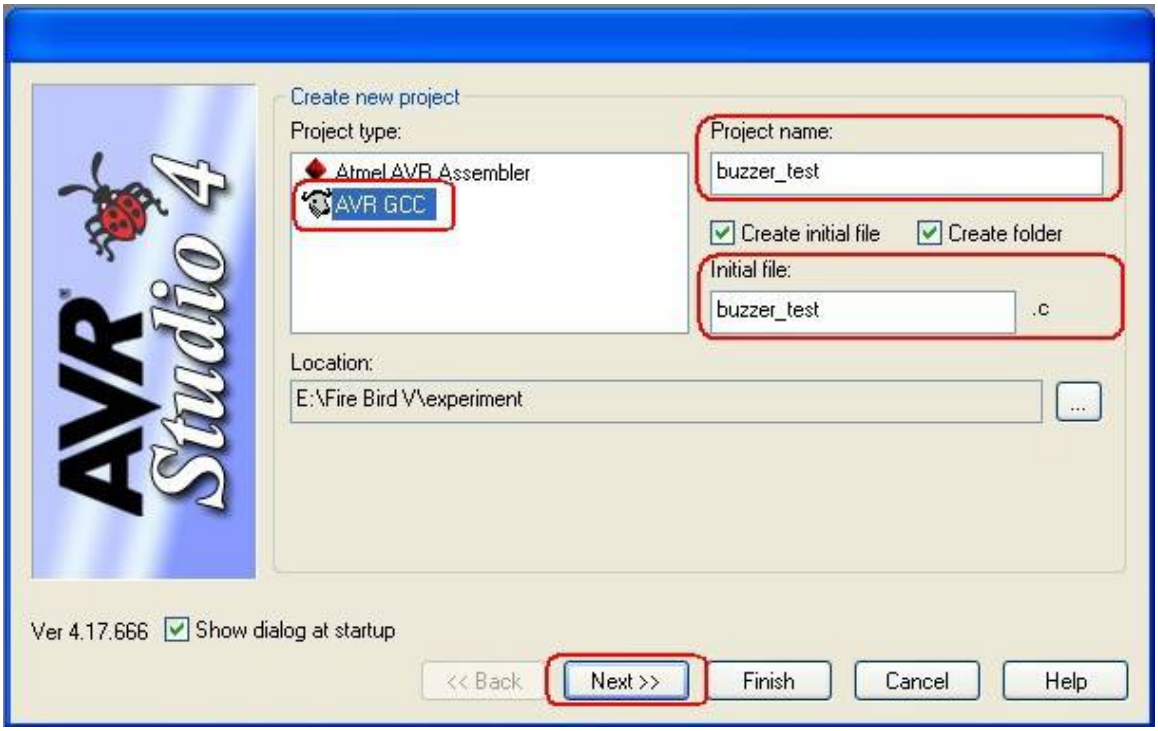


Figure 2.18

4. Select debug platform and Device. In this case we have selected “AVR simulator” and “ATMEGA2560” microcontroller and click finish.

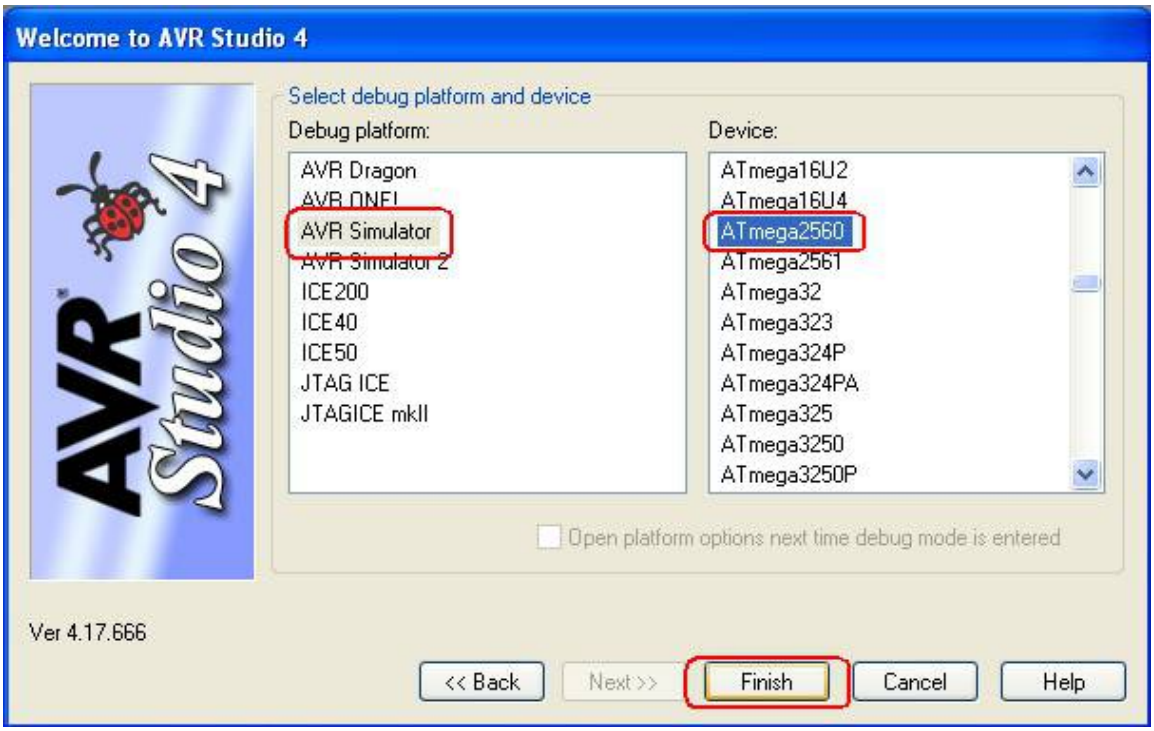


Figure 2.19

5. Now we are almost ready to write our first code. Before we start coding we will check other setting to make sure that they are set properly.

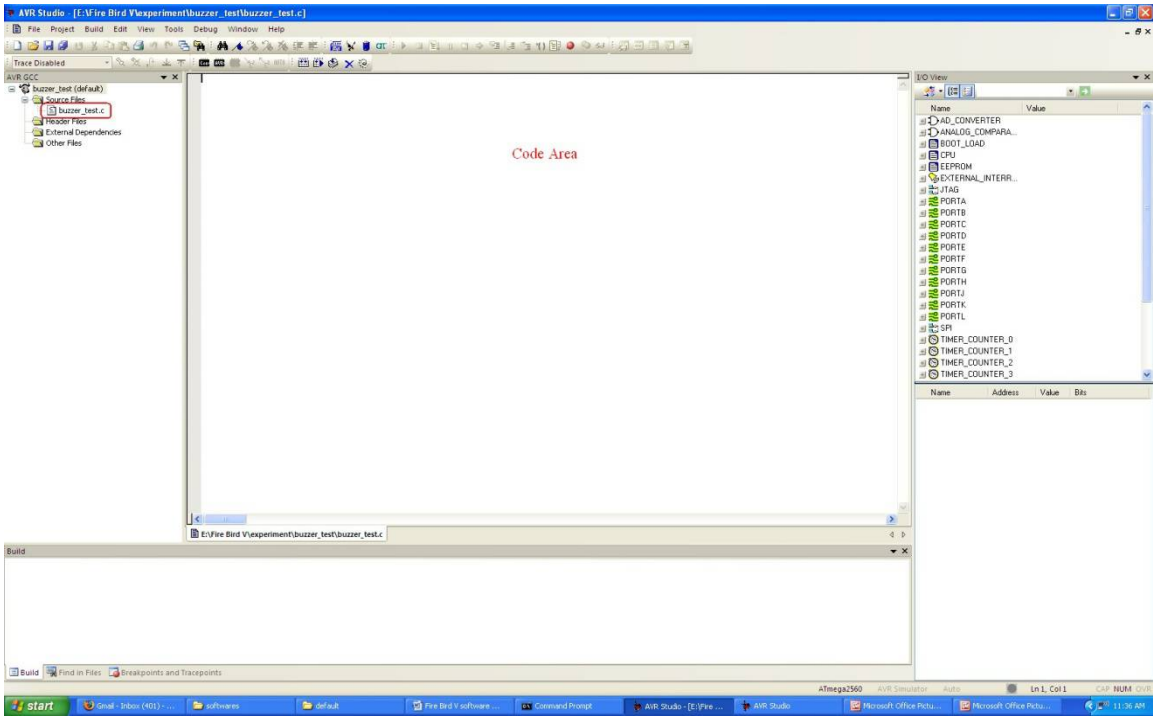


Figure 2.20

6. Open Project menu and click on the Configuration option.

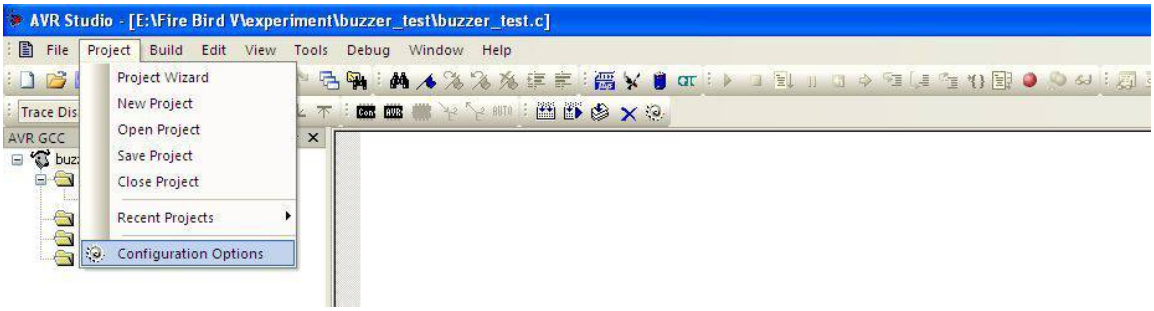


Figure 2.21

7. In the Project Option "General" tab will open. Select device as "ATMEGA2560" and frequency (Crystal Frequency) as 14.7456MHz i.e. 14745600Hz. Set the optimization level be at "-O0".

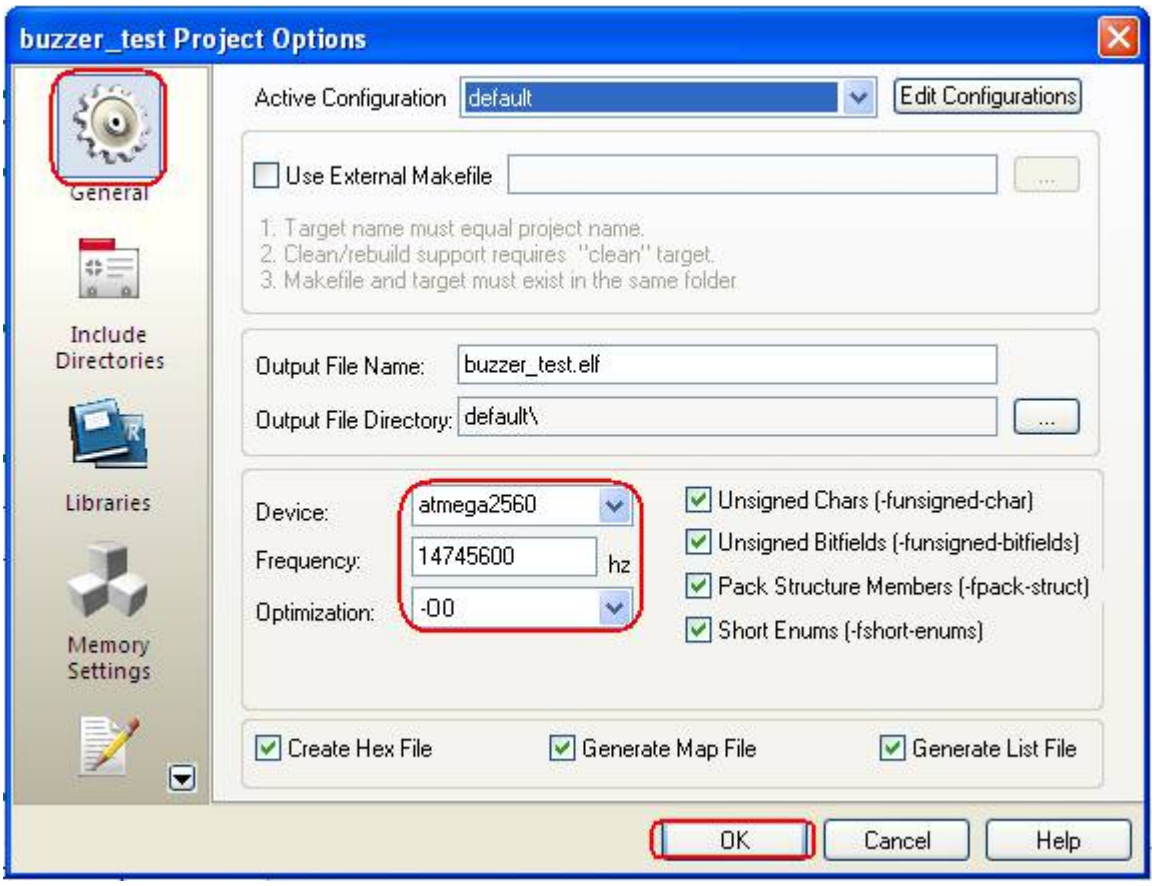


Figure 2.22

Selecting proper optimization options

"Optimization" option defines the optimization level for all files. Higher optimization levels will produce code that is harder to debug. Stack variables may change location, or be optimized away, and source level debugging may "skip" statements because they too have been optimized away.

The levels of optimization are:

- -O0 No optimization. This is the same as not specifying any optimization.
- -01 Optimize. Reduces code size and execution time without performing any optimizations that take a great deal of compilation time.
- -O2 Optimize even more. avr-gcc performs almost all optimizations that don't involve a space-time tradeoff.
- -O3 Optimize yet more. This level performs all optimizations at -O2 along with -finline-functions and -frename-registers.
- -Os Optimize for size. Enables all -O2 optimizations that don't increase code size. It also performs further optimizations designed to reduce code size.

For more information on optimization, see the 'man' pages for avr-gcc.

Important: During the coding choose appropriate optimization option. If you feel that code is not working properly as it should be then turn off all optimization by selecting optimization option as "-O0". Once you know that your code is properly working then you can incrementally increase optimization level.

We suggest that always use optimization level as "-O0" at the beginner level.

8. Make sure that in the External Tools, proper path for avr-gcc.exe and make.exe are given and press ok.

Now we are ready to write our first code.

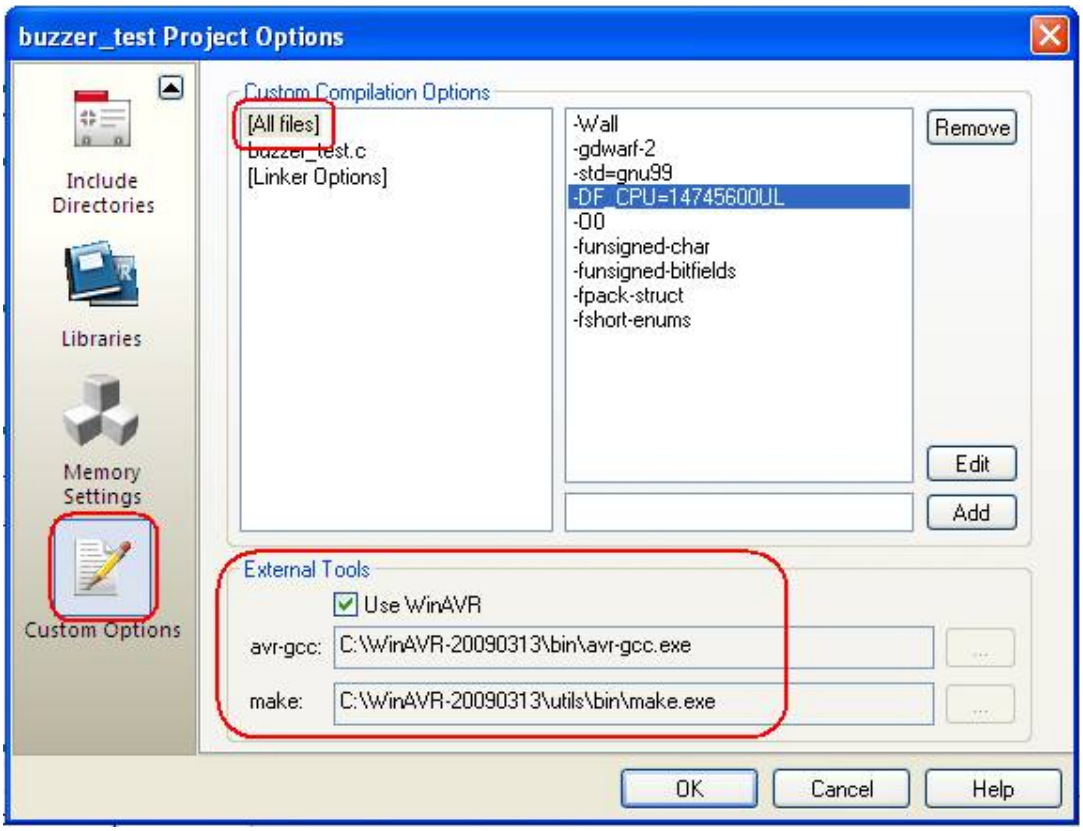


Figure 2.23

2.4 Writing your first code in AVR Studio

This program will make robot's buzzer beep.

Copy the following code in window "code area" as shown in figure 2.25. We will see how this code works in the next chapter.

```
//Buzzer is connected at the third pin of the PORTC
//To turn it on make PORTC 3rd (PC3 )pin logic 1
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//Function to initialize Buzzer
void buzzer_pin_config (void)
{
 DDRC = DDRC | 0x08;                    //Setting PORTC 3 as output
 PORTC = PORTC & 0xF7;   //Setting PORTC 3 logic low to turnoff buzzer
}

void port_init (void)
{
 buzzer_pin_config();
}

void buzzer_on (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore | 0x08;
 PORTC = port_restore;
}

void buzzer_off (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore & 0xF7;
 PORTC = port_restore;
}

void init_devices (void)
{
 cli();      //Clears the global interrupts
 port_init();
 sei();      //Enables the global interrupts
}

//Main Function
int main(void)
{
        init_devices();
        while(1)
        {
                buzzer_on();
                _delay_ms(1000);   //delay
                buzzer_off();
                _delay_ms(1000);   //delay
        }
}
```

We are now going to compile this code to generate the hex file and we will load same on the Robot's microcontroller. Select "Build" menu and click on "Rebuild All". It will compile the "buzzer_test.c" code and will generate "buzzer_test.hex" file for the robot's microcontroller.

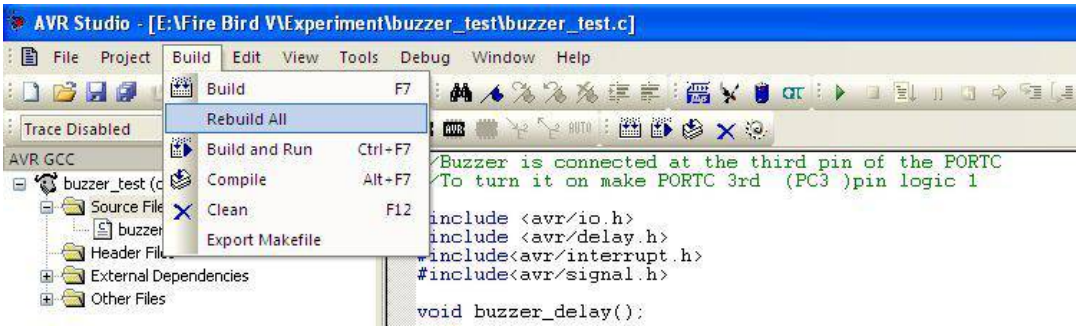


Figure 2.24

You can verify successful compilation in the bottom most "Build" window of the AVR Studio.

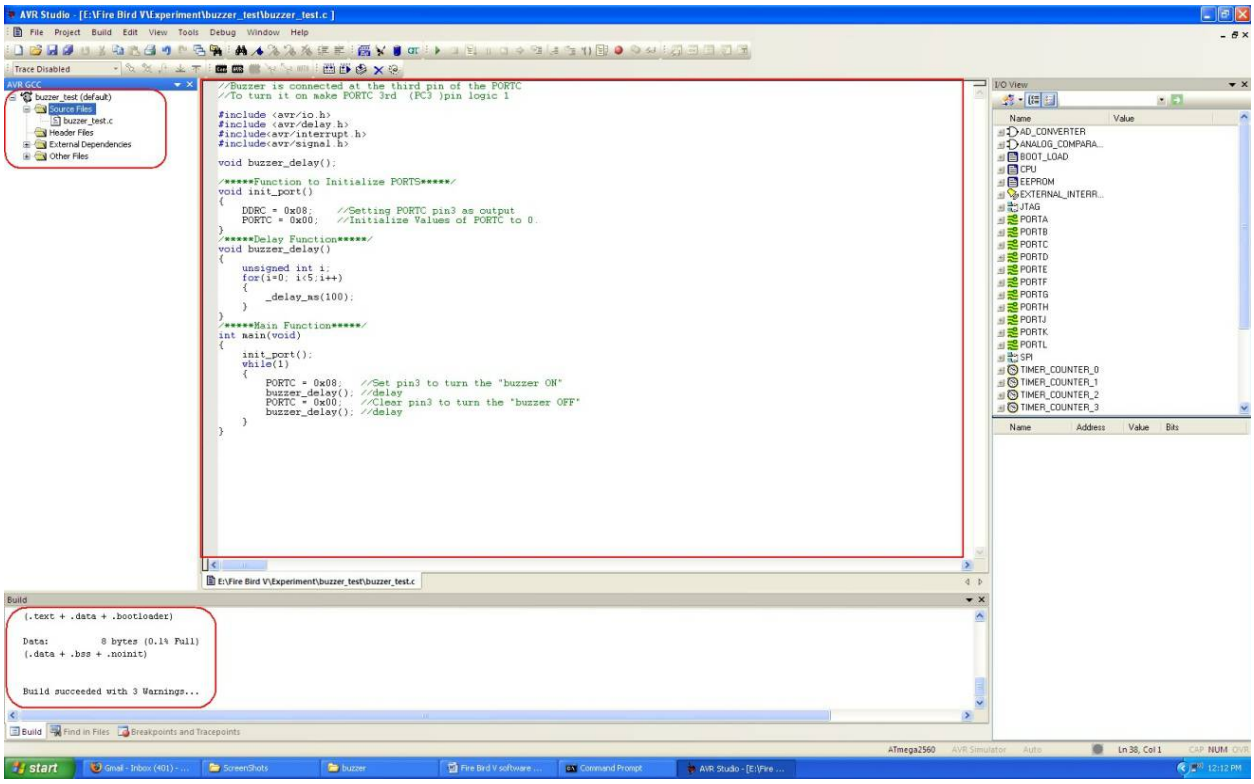


Figure 2.25

You can also verify that "buzzer_test.hex" file is generated in the "default" folder inside the project folder you have selected.

2.5 Debugging the code in AVR studio

After successful compilation of the code we can debug the code by AVR Debugger provided by AVRStudio. Here is the illustration of debugging of code given in Exp1 (buzzer ON-OFF folder).

Click on Debug tab in the menu and click on “Start Debugging”.

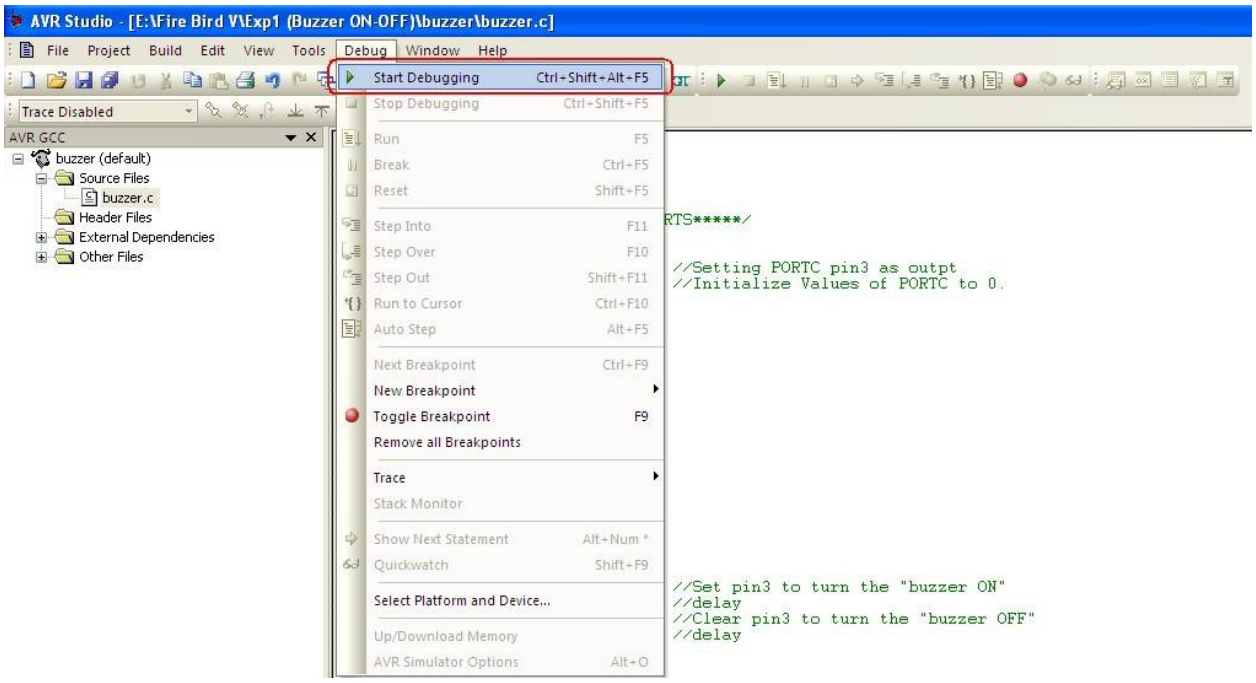


Figure 2.26

Now debugging mode is started and an arrow is visible at the first line of our main function from where the debugging will start.

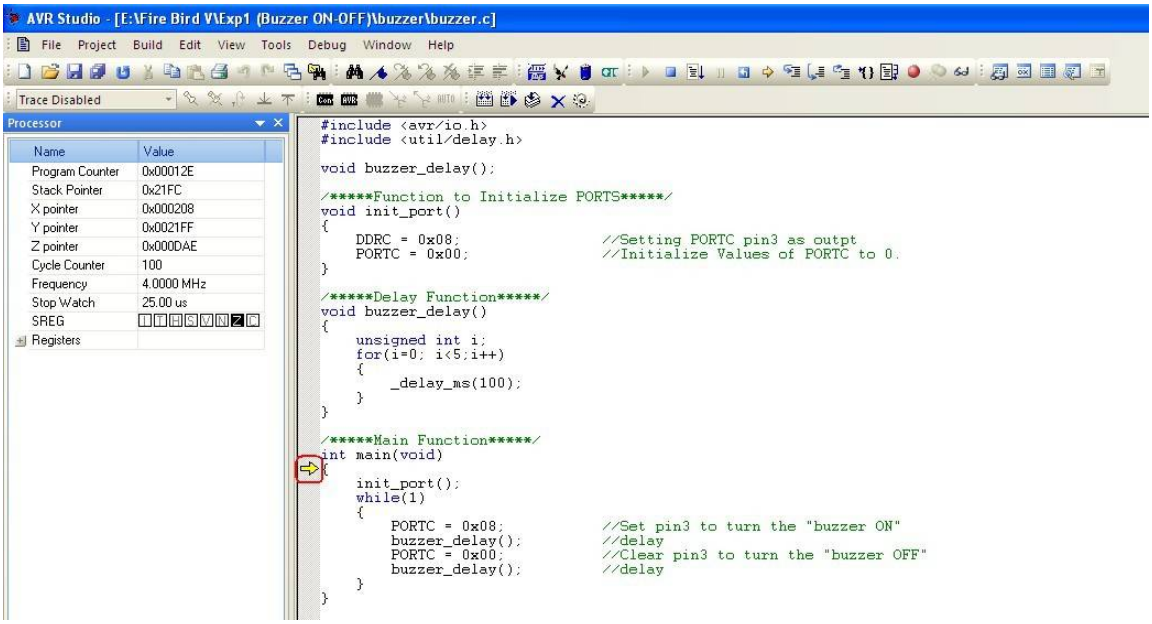


Figure 2.27

Press “F11” key or “Step into” button  from the toolbar to start debugging statement by statement. Processor details are visible at left window and the I/O port status is displayed at the rightmost window.

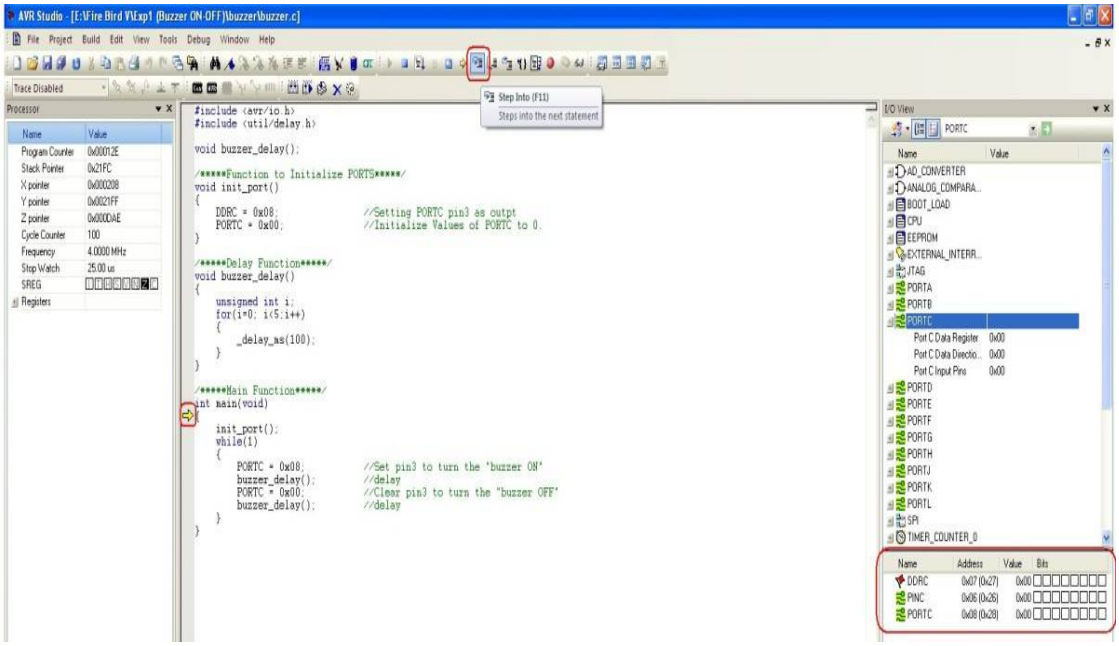


Figure 2.28

By this way we can continuously monitor the bit changes in any of the registers of microcontroller and debug the code before actually burning it to the microcontroller. PORTC bits changes as per our commands and these changes can be seen in right window. After debugging is done, select “Stop Debugging” from Debug tab.

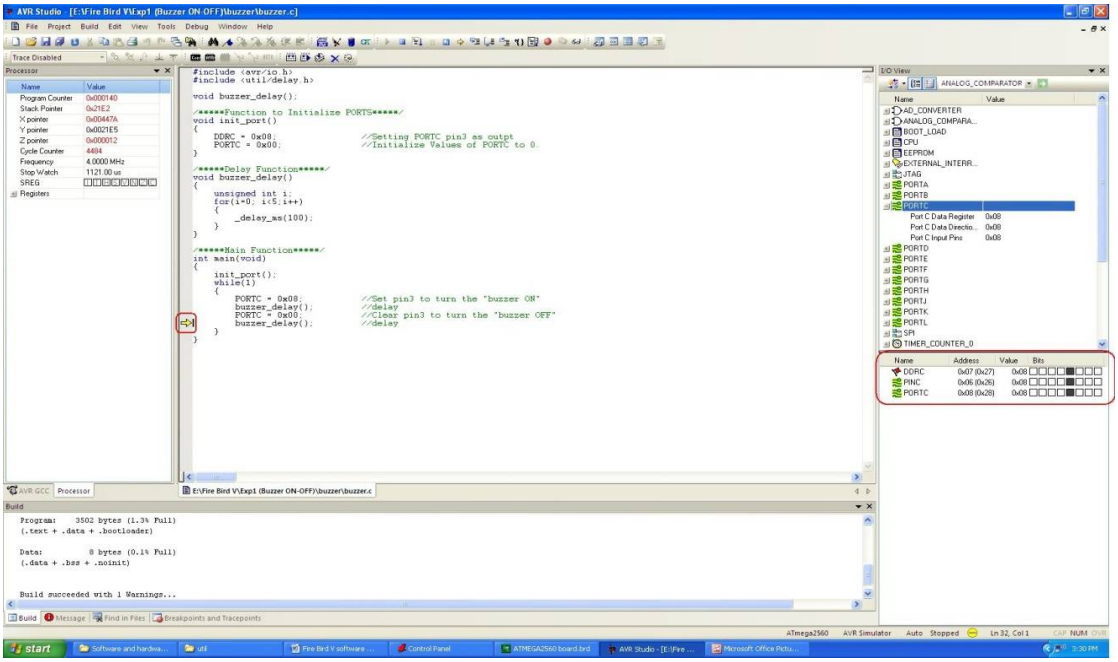


Figure 2.29

2.6 Loading your code on robot using AVR Boot loader from NEX Robotics

All AVR microcontrollers can be programmed using In System Programming (ISP), external programmer or using boot loader. Advantage with the boot loader is that you don't need any external hardware to load .hex file on the microcontroller. It also protects robot's hardware from possible damage due to static electricity and prevents any accidental changes in the fuse settings of the microcontroller.

Important:

All the Fire Bird V ATMEGA2560 robots are factory shipped with this boot loader. We strongly recommend to use this mode for the robot programming. If you load .hex file on the robot using any ISP programming then this boot loader will get erased and it needs to be reloaded again using ISP programmer.

2.6.1 Boot loader operating principle

If Bootloader firmware is loaded on the microcontroller, it allows in system programming directly via serial port without any need for the external ISP programmer. Code responsible for In System Programming via serial port (boot loader) resides in the configurable boot memory section of the microcontroller. When signaled using external switch while resetting the microcontroller it gets active and waits for communication from the PC for copying .hex file on the microcontroller's flash memory. PC sends the .hex file to the microcontroller. Code residing in the boot section loads the .hex file on the microcontroller's flash memory. After the boot loading process is complete, newly loaded code can be executed by pressing reset. Once the code is loaded on the microcontroller UART is free and can be used for other applications. Bootloader get invoked only if boot switch is kept pressed while microcontroller is reset using reset switch.

Note:
Bootloader code is loaded on the ATMEGA2560 of the Fire Bird V robot using ISP programmer. It is recommended that you only use Bootloader for the robot. If you use ISP programmer with the robot then boot section code might get erased and robot will no longer support boot loading.

2.6.2 Programming the robot via Bootloader from NEX Robotics

.hex file can be loaded on the robot via serial port or USB port of the Fire Bird V robot. Robot is shipped with the bootloader to program robot via USB port.

USB port of the Fire Bird V robot is connected to the UART 2 of the ATMEGA2560 microcontroller via FT232 USB to serial converter. You need to install drivers for FT232 USB to serial converter on the PC before bootloading.

2.6.3 Installing drivers of FT232 USB to Serial Converter

Steps to install drives for FT232 USB to Serial Converter are covered in detail in the section 6.5 of the Hardware Manual. To identify and change the COM port number, refer to section 6.6 of the Hardware Manual.

Important:

1. If COM port number is set to more than 8 by the PC then you have to change it in the range of COM 2 to COM 8 else AVR Bootloader will not program the robot. For changing COM port number, refer to section 6.6 of the Hardware Manual.

2. When using USB port for the communication, for proper operation first turn on the robot then insert the USB cable in the robot. We have to follow this sequence because USB to serial converter chip is powered by USB. If any fault occurs then turn off the robot and remove the USB cable and repeat the same procedure.

2.6.4 Installing Bootloader GUI on the PC

Step 1:
Copy "AVR Bootloader" folder which is located in the "Software and Drivers" folder of the documentation CD to the PC and click on the "setup" application file (not on AVR Bootloader Setup).

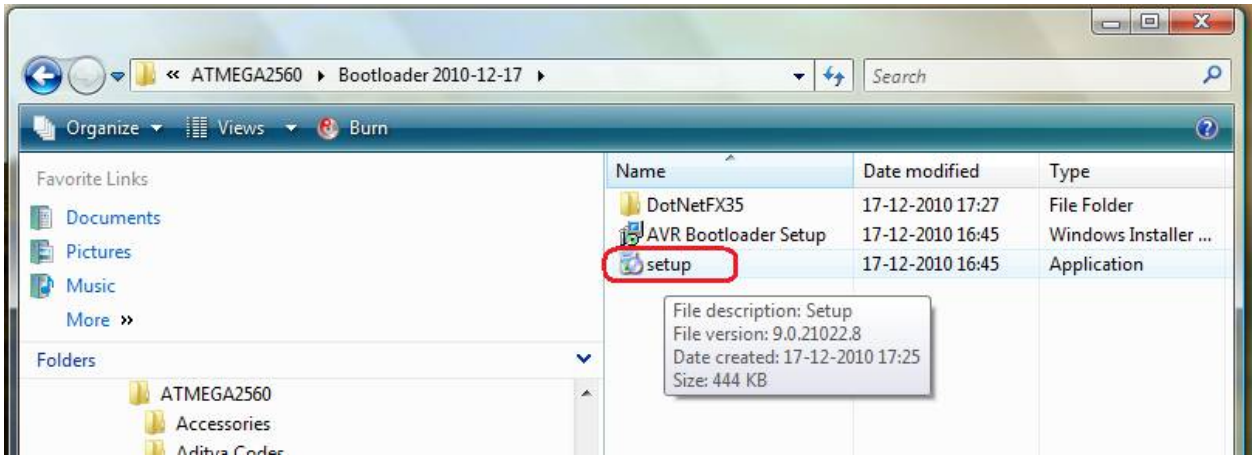


Figure 2.30

Step 2:
Follow the installation steps to complete the installation.

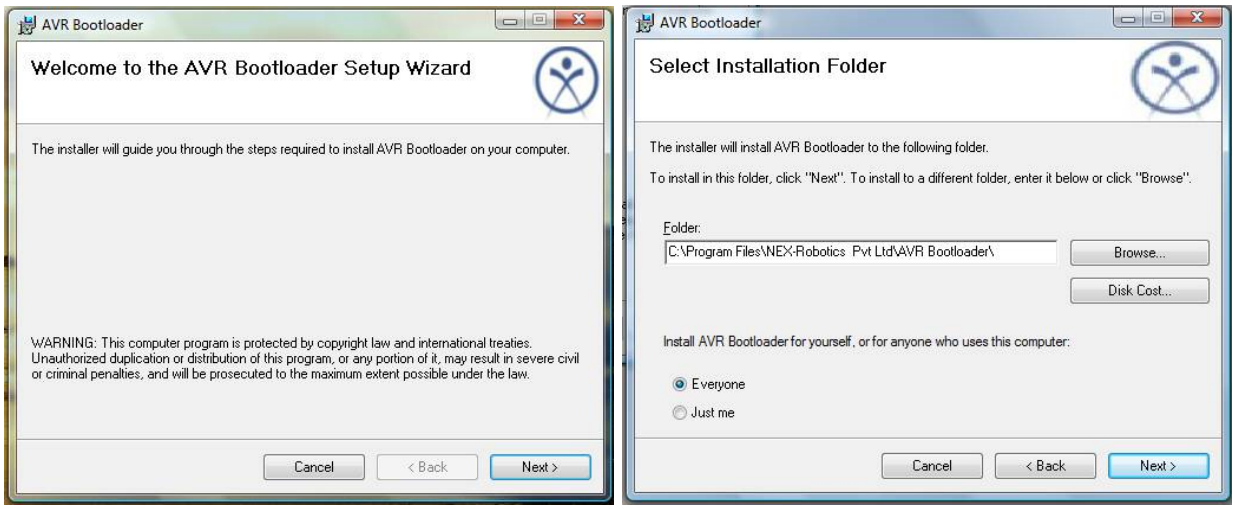


Figure 2.31

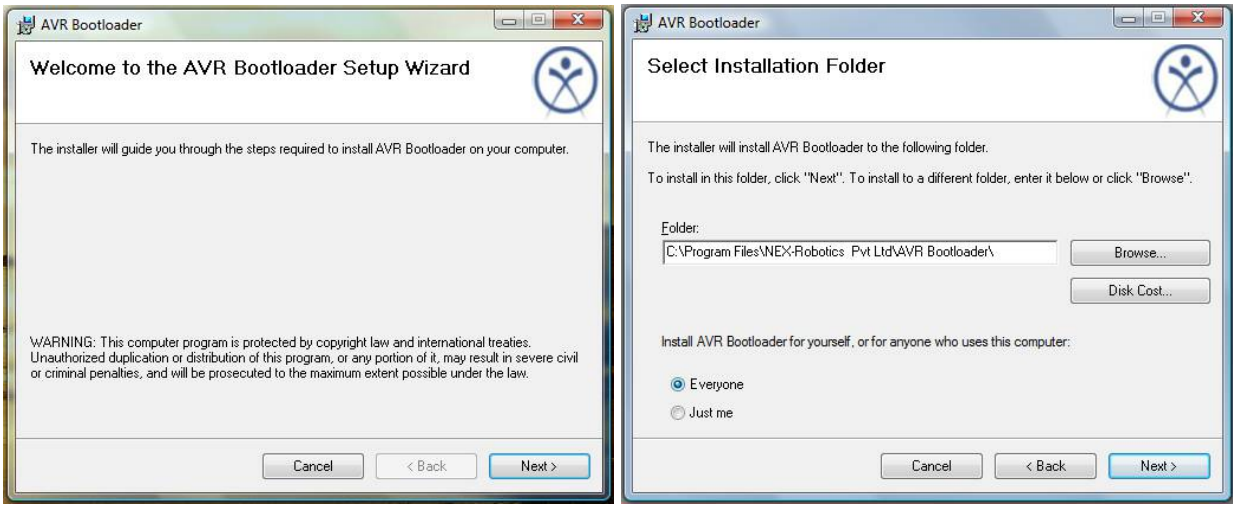


Figure 2.32

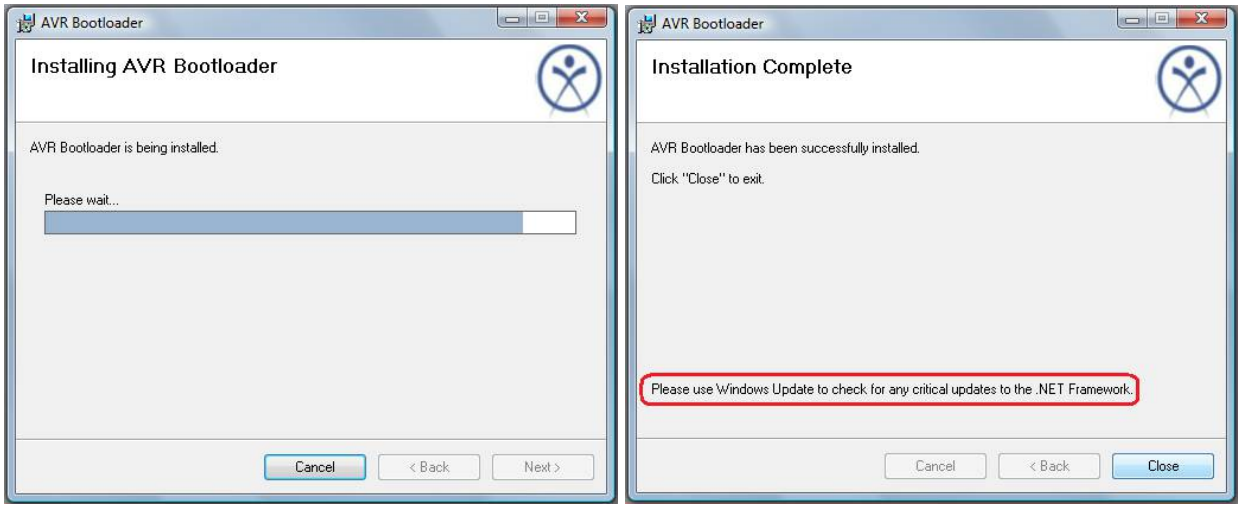


Figure 2.33

2.6.5 Using AVR Bootloader

Step 1:
Go to Start Menu and click on "AVR Bootloader". AVR Bootloader software will start.

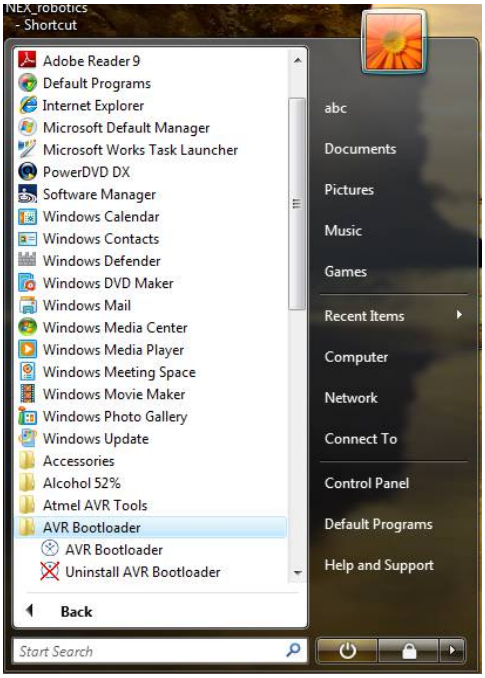


Figure 2.34

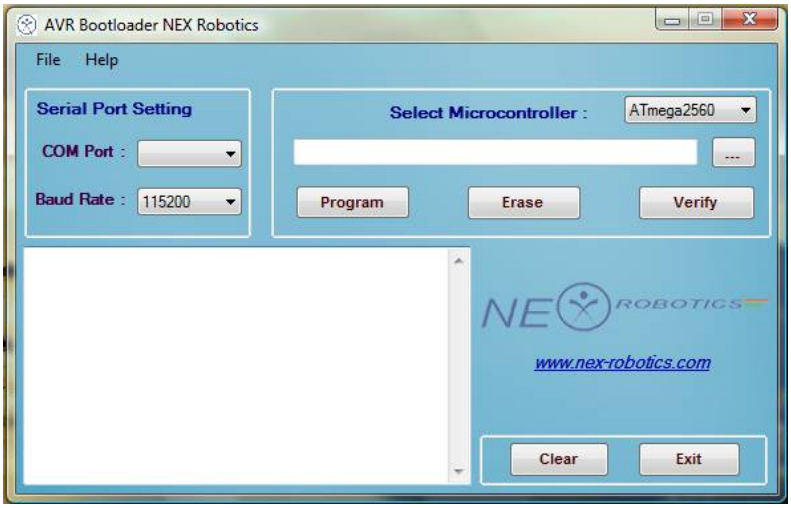


Figure 2.35

Step 2:

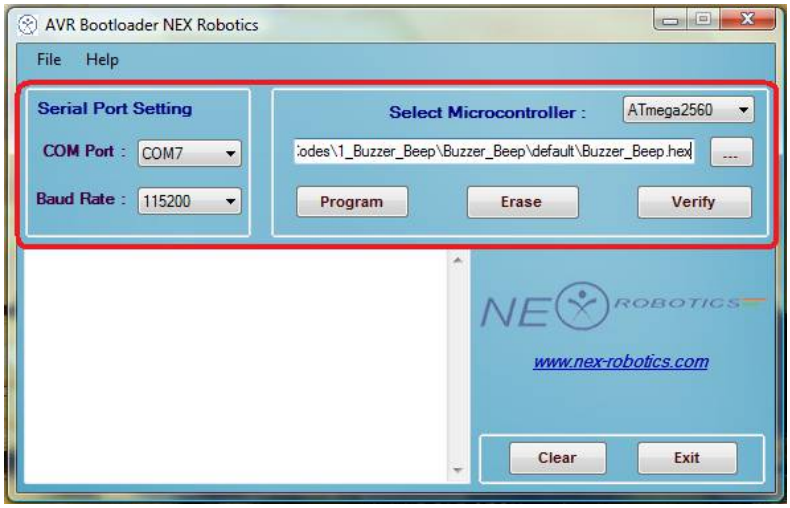


Figure 2.36

In this step we will configure port settings, select the microcontroller and .hex file which is to be loaded on the robot.

1. Make sure that drivers for FT232 USB to serial converter are installed.
2. FT232 is connected to the UART2 of the microcontroller by jumper J1 on the ATMEGA2560 microcontroller board. For more details refer to section 6.3 of the Hardware Manual.
3. Turn on the Robot.
4. Connect USB wire between Robot and the PC and wait for 2 seconds.
5. AVR Bootloader software will auto detect the COM port number.
6. Click on the COM port number. It will show the detected COM port numbers. If multiple COM port numbers are detected then to identify COM port number associated with the Robot, refer to section 6.6 in the Hardware Manual.
7. If Robot's COM port number is more than COM8 then change it to any COM port number between COM1 to COM8.
8. Set the Baud rate at 115200 bps.
9. Select ATMEGA2560 microcontroller
10. Browse for the target .hex file.

Now we are ready to load .hex file on the robot.

Step 3:

In this step we will load the .hex file on the robot.

1. First press the boot switch (switch on the right). Press and release reset switch with time interval of 1 second while continuously pressing the boot switch. Bootloader is written is such a way that when reset switch is pressed while holding PE7 low, ATMEGA2560 will go in to bootloading mode.
2. Press program button on the GUI. With in 2 seconds .hex file loading will start. You can see the activity on the TX and RX LEDs of the FT232 USB to Serial Converter on the ATMEGA2560 microcontroller adaptor board. These LEDs are located just below the FT232 USB to Serial Converter chip. To find the location of the TX and RX LEDs refer to figure 3.58 in the Hardware Manual.
3. Figure 2.37 shows the comments in the message box after successfully programming the ATMEGA2560 microcontroller on the robot.

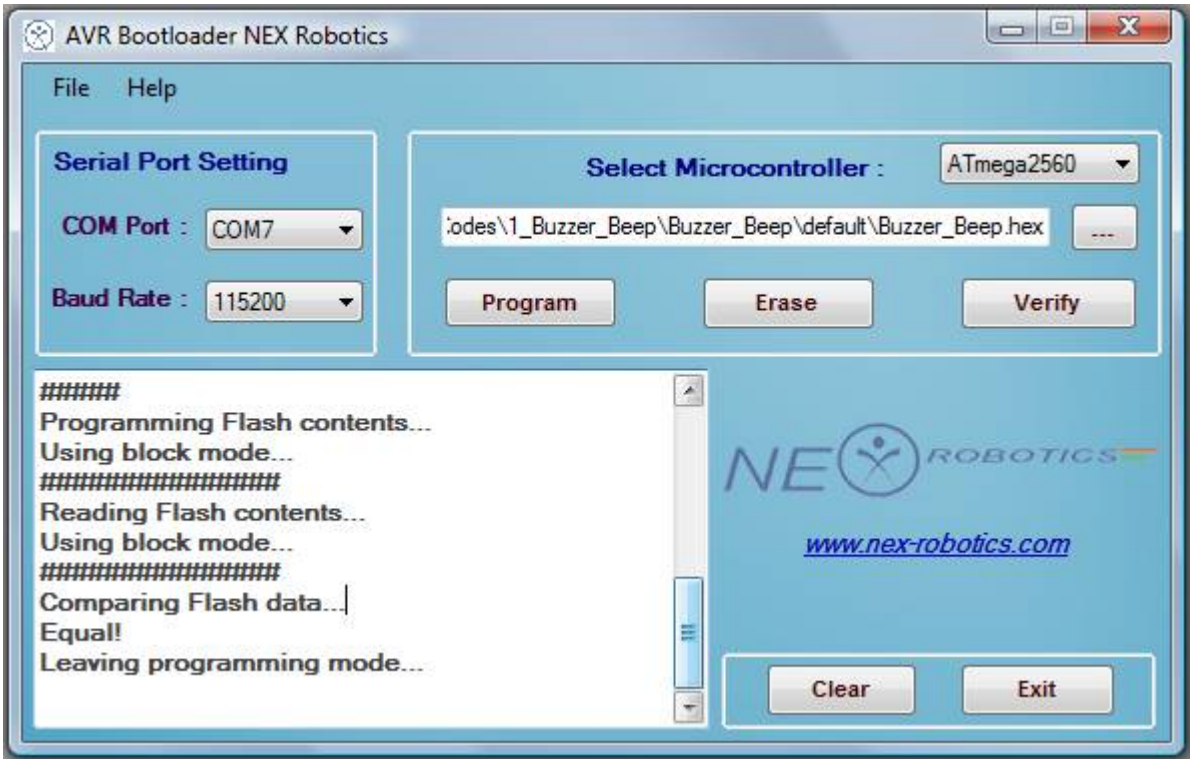


Figure 2.37

After successfully programming following text will appear in message box:

```
Serial port timeout set to 5 sec.
Found AVRBOOT on COM7!
Entering programming mode...
Parsing XML file for device parameters...
Parsing '.\ATmega2560.xml'...
#######
Saving cached XML parameters...
Signature matches device!
Erasing chip contents...
Reading HEX input file for flash operations...
################################################################################
################################################################################
################################################################################
#####
Programming Flash contents...
Using block mode...
################
Reading Flash contents...
Using block mode...
################
Comparing Flash data...
Equal!
Leaving programming mode...
```

Note:
Loading bootloader on the ATMEGA2560 microcontroller will be covered in the Section 2.11 after introduction to ISP programmers for the ATMEGA2560 microcontroller.

2.7 Correct Jumper setting before loading hex file on the robot using ISP programmer

All AVR microcontrollers can be programmed by using external In System Programmer. In the manual we are going to cover AVRISP mkII from ATMEL and AVR USB programmer NR-USB-006 from NEX Robotics for In System Programming (ISP).

For ATMEGA2560 (master) and ATMEGA8 (slave) ISP is done via their SPI port. Both microcontrollers also talk with each other using SPI bus where ATMEGA2560 acts as master and ATMEGA8 acts as slave. Order to load the .hex file on these microcontrollers; we need to disconnect the SPI bus connection between these microcontrollers by removing three Jumpers marked by J4 on the ATMEGA2560 microcontroller socket. For more details on the jumpers refer to section 3.19.6 of the Hardware Manual.

Refer to figure 2.38 and 2.39 for the correct jumper settings before proceeding to ISP. Figure 2.40 shows the ISP socket for the ISP programming.

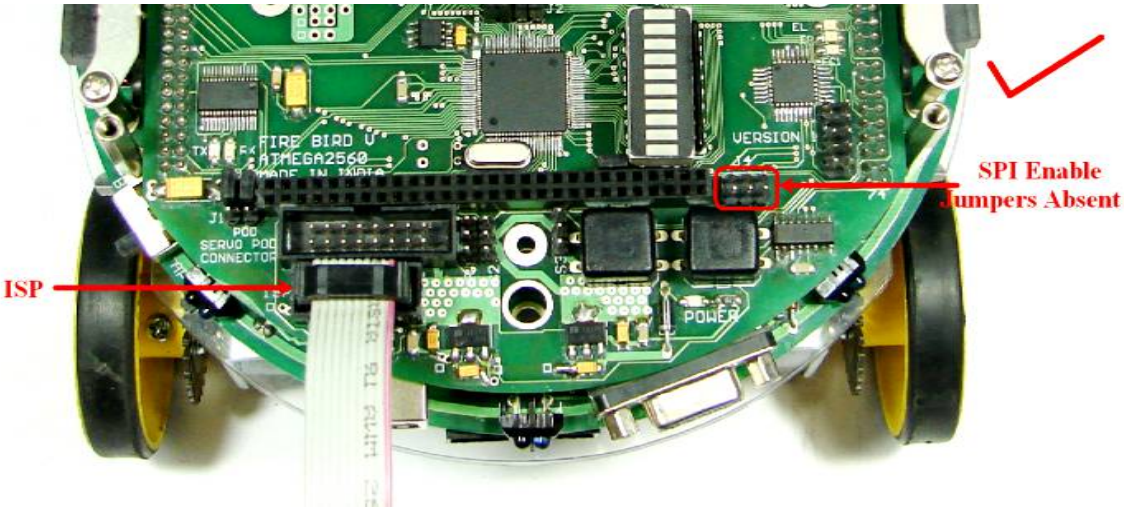


Figure 2.38: Correct jumper setting before proceeding to ISP

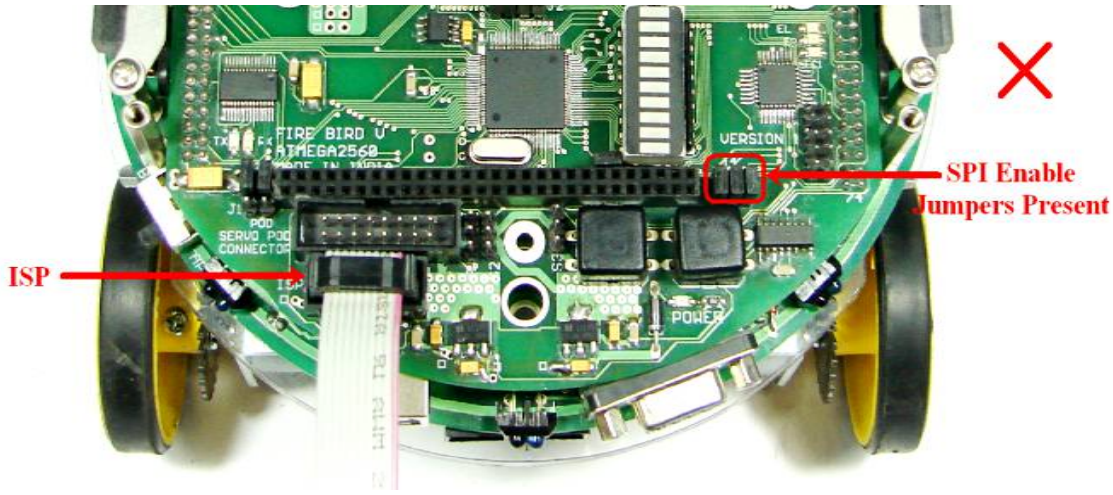


Figure 2.39: Incorrect jumper setting before proceeding to ISP

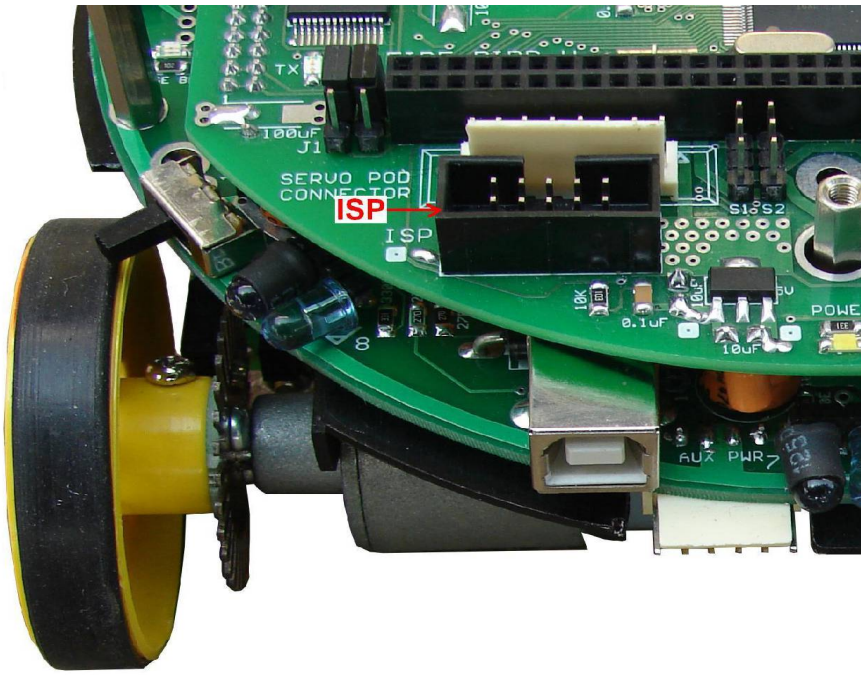


Figure 2.40: ISP socket on the ATMEGA2560 microcontroller adaptor board

Note:
ATMEGA8 slave microcontroller is used for collecting analog data from IR proximity sensors 6, 7, 8, Robot current sense (if ACS712 current sensor is installed), extended white line sensor channels 4, 5, 6, 7 and pin on the servo expansion port. If you are not using these sensors then for convenience you can keep jumper J4 disconnected. If you want to access these sensor values then you have to connect jumper J4.

2.8 Loading your code on the robot using ATMEL's AVRISP mkII programmer

AVRISP mkII programmer from the ATMEL is the most versatile programmer. It is very easy to use and has more features. It is the most recommended programmer for the robot after AVR Bootloader from NEX Robotics.

Figure 2.41: AVRISP mkII

Step 1:

1. Connect AVRISP mkII to the PC. It will install driver automatically provided that USB driver installation option is selected while installing AVR Studio. For more details refer to figure 2.13.
2. Start AVR Studio
3. Go to "Tools" tab and click on "Program AVR". Select connect option.
4. Window as shown in figure 2.43 will open.

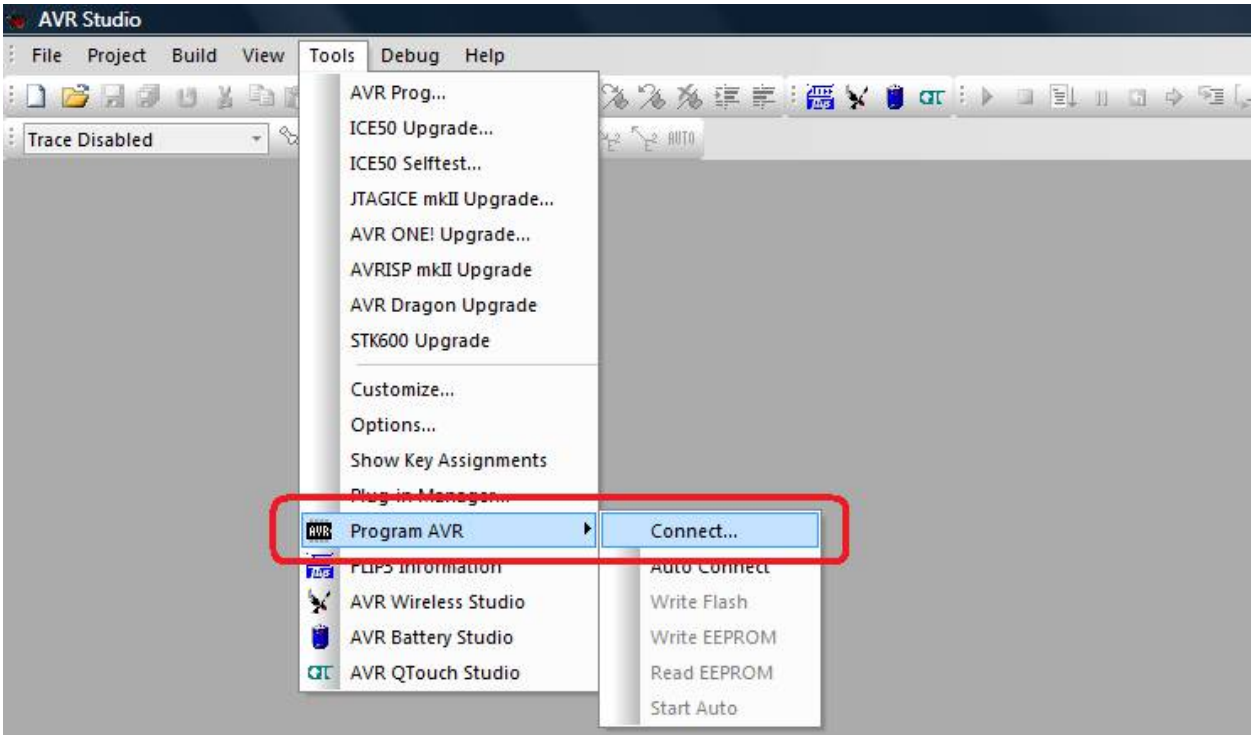


Figure 2.42

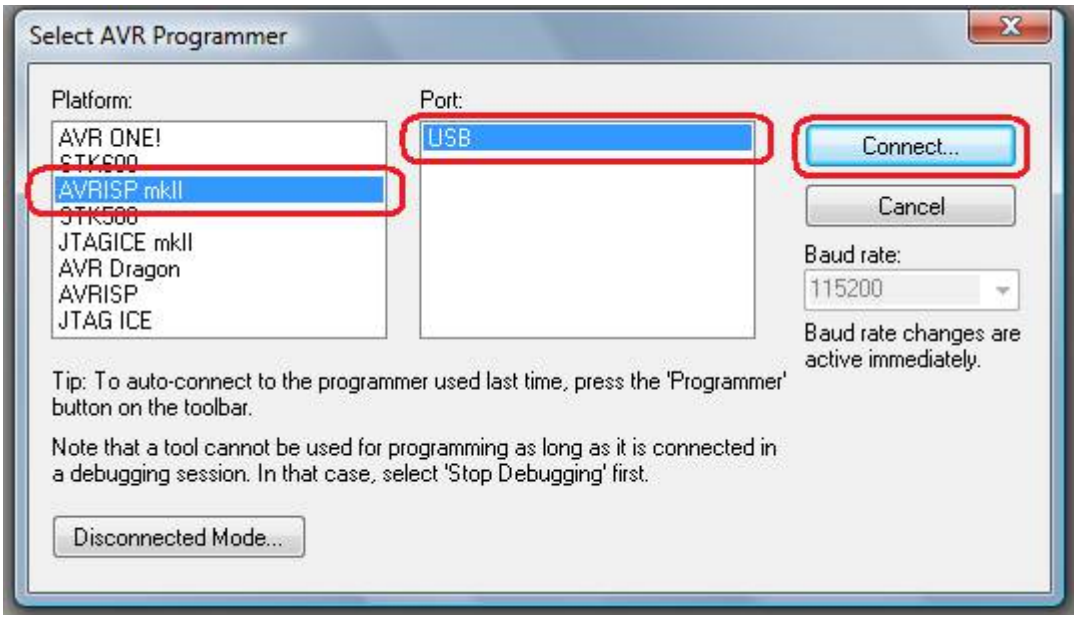


Figure 2.43

5. Select platform as AVRISP mkII and Port as USB Port and press connect.
6. Window as shown in Figure 2.44 will appear.

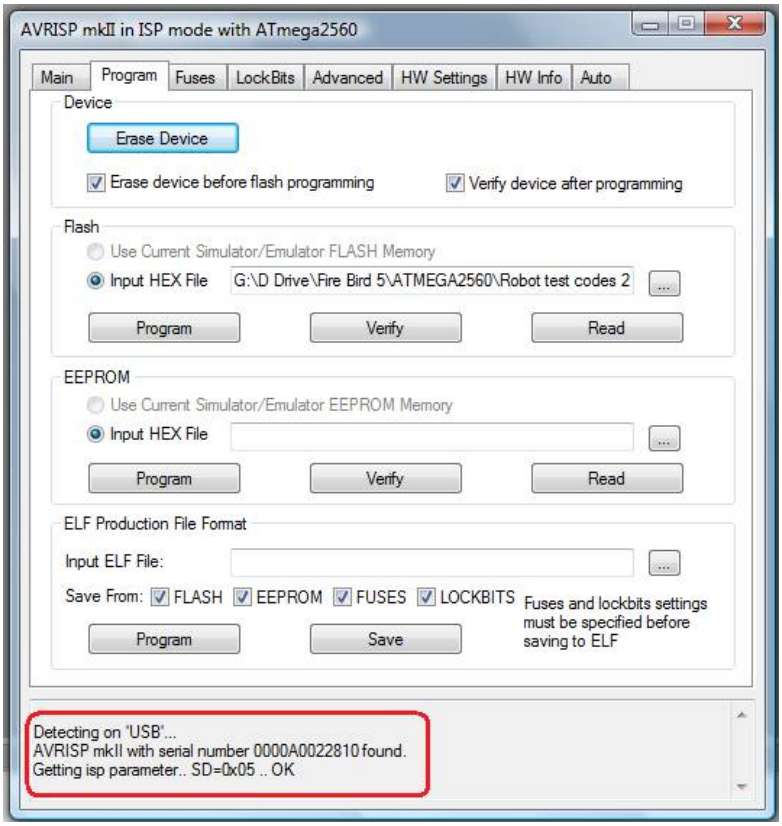


Figure 2.44

Step 2:

Connecting AVRISP mkII with the robot

1. AVRISP mkII use 6 pin FRC connector for ISP, while Fire Bird V robot uses 10 pin FRC connector for programming. We need to use AVRISP adaptor to convert 6 pin to 10 pin connector. Figure 2.45 shows the 6 to 10 pin converter for the In System Programming.
2. Connect AVRISP adaptor between robot and AVRISP mkII. Insert 10 pin FRC connector in the Fire Bird V ATMEGA2560 robot and turn the power on. For ISP port location and other precautions refer to section 2.7.
3. Turn on the robot

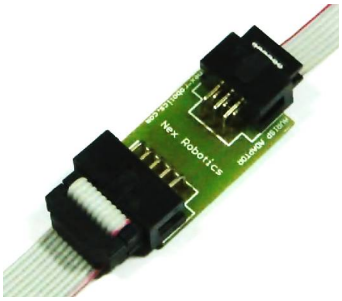


Figure 2.45: In System Programming

Step 3:

Reading the microcontroller signature

Follow Step 4 if you are connecting AVRISP mkII with the robot for the first time or if any problem occurs during programming.

1. Go to Main tab
2. Select “ATMEGA2560” microcontroller.
3. Click on the “Read Signature” button.
4. It will read the signature and if its matches with the microcontroller signature, we will get the confirmation as “Signature matches selected device” as shown in the below window. Now we are ready to load hex file on the robot.

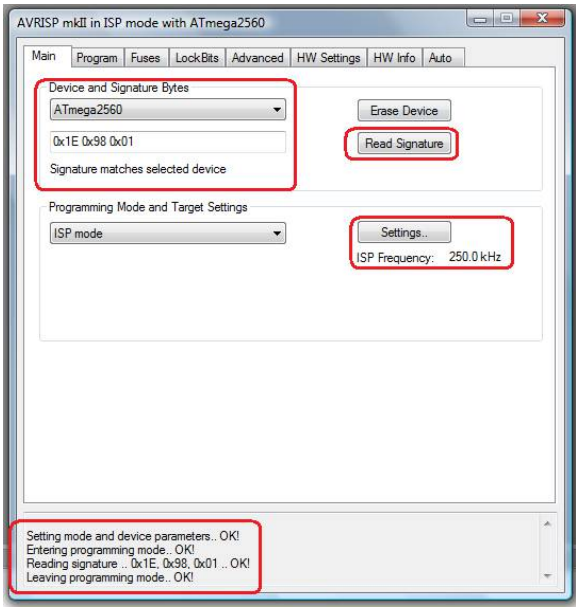


Figure 2.46

If ISP does not work properly then try to reduce the ISP frequency and try it again by clicking on the “Settings” button which is located inside the “Programming Mode and Target Settings frame”. Refer to figure 2.47.

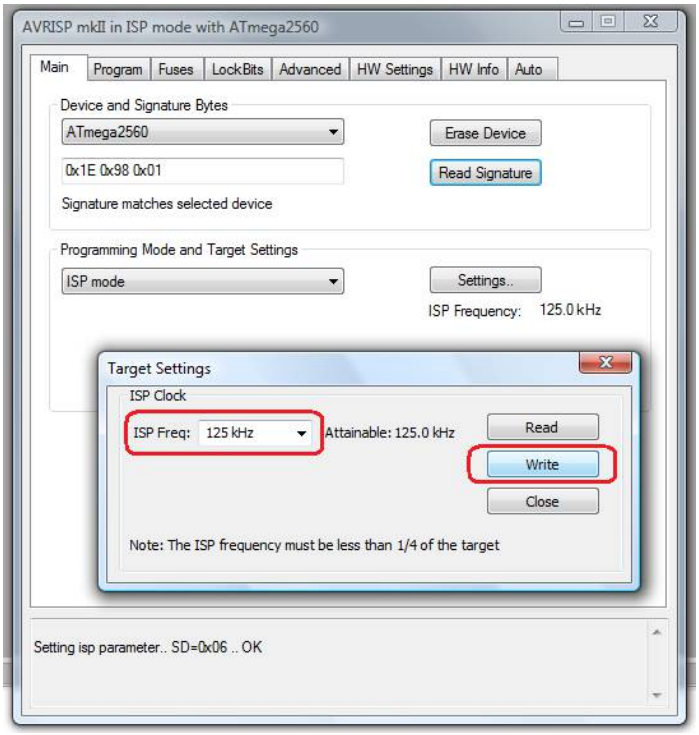


Figure 2.47: Changing ISP frequency if required

Note:
If you want to load program at faster speed you can increase the ISP frequency.
If you notice any instability while programming then reduce the ISP frequency.

Step 4:

Loading .hex file on the microcontroller

1. Go to “Program” tab.
2. Check on Erase device before programming and Verify device after programming check box.
3. Browse and select the desired hex file in the flash section
4. Press “Program” button
5. Look at the comments at the bottom to verify that hex file is loaded in the flash.

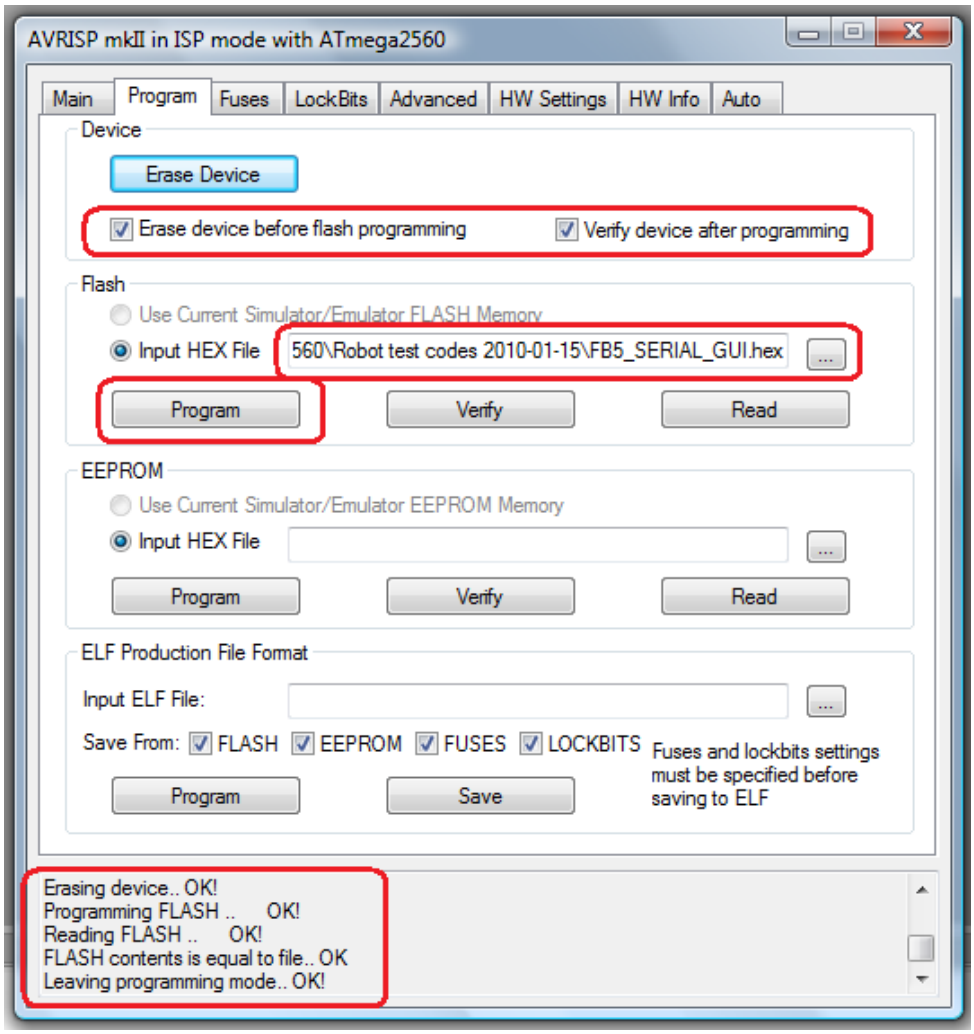


Figure 2.48

2.9 Fuse settings for ATMEGA2560 (Master) microcontroller

All the microcontroller's Fuse settings are factory set at the NEX Robotics before shipping the robot. Do not change them. This information is only given for the reference.

All the Fuse settings are done using AVRISP mkII programmer. For starting AVRISP mkII refer to section 2.8.

To check the fuse settings click on the Fuse tab. Following window shown in figure 2.49 will appear. To verify the fuse settings press "Read". To write fuse settings after modifications press "Program". Make sure that "Auto Read", "Smart warning" and "Verify after programming" are checked.

Upon clicking on the "SUT CKSEL" it will extend the clock options as shown in figure 2.50. In the "SUT CKSEL" select "Ext. Crystal Osc. 8.0- MHz; Start-up time: 16K CK + 65ms"

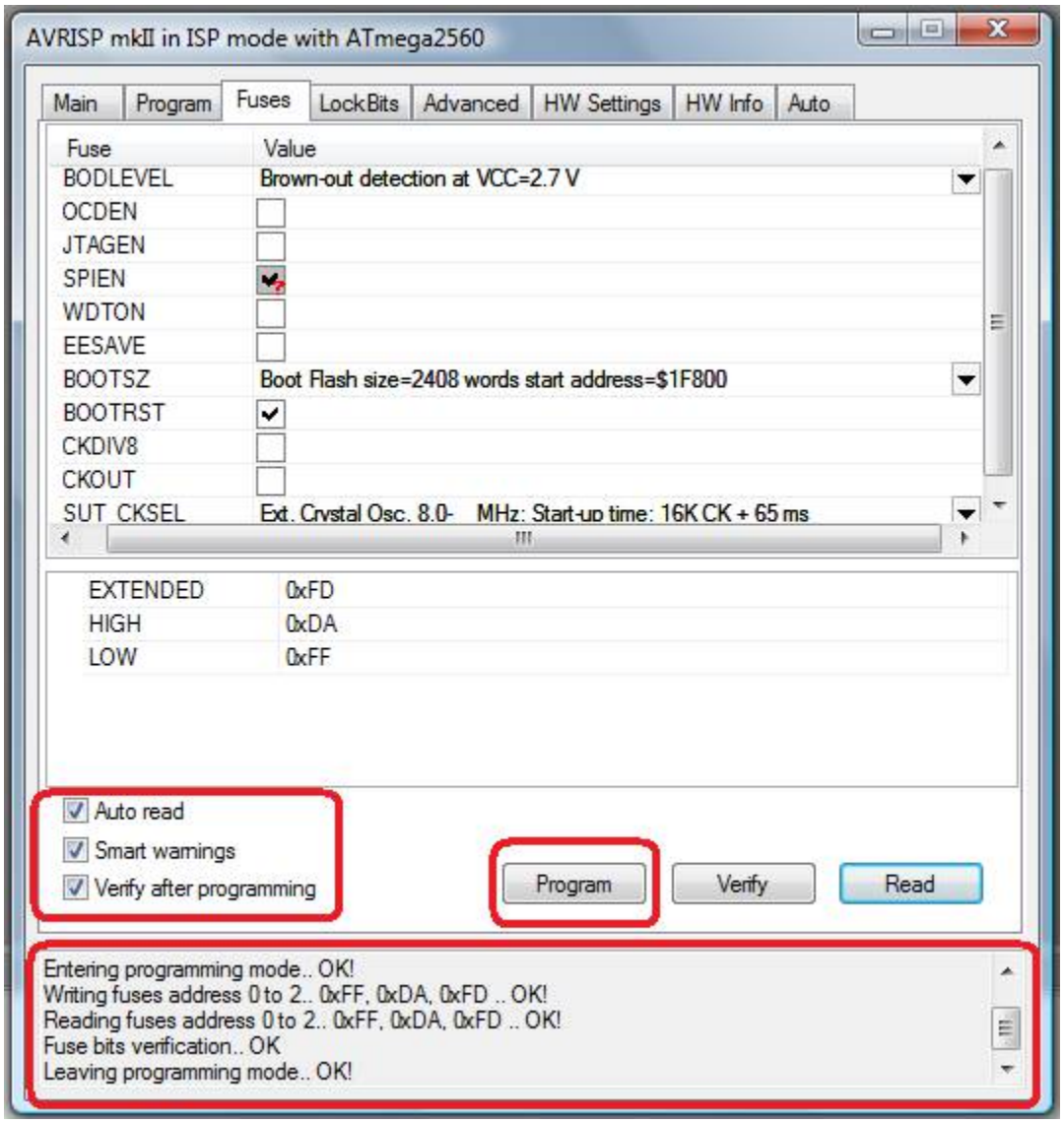


Figure 2.49

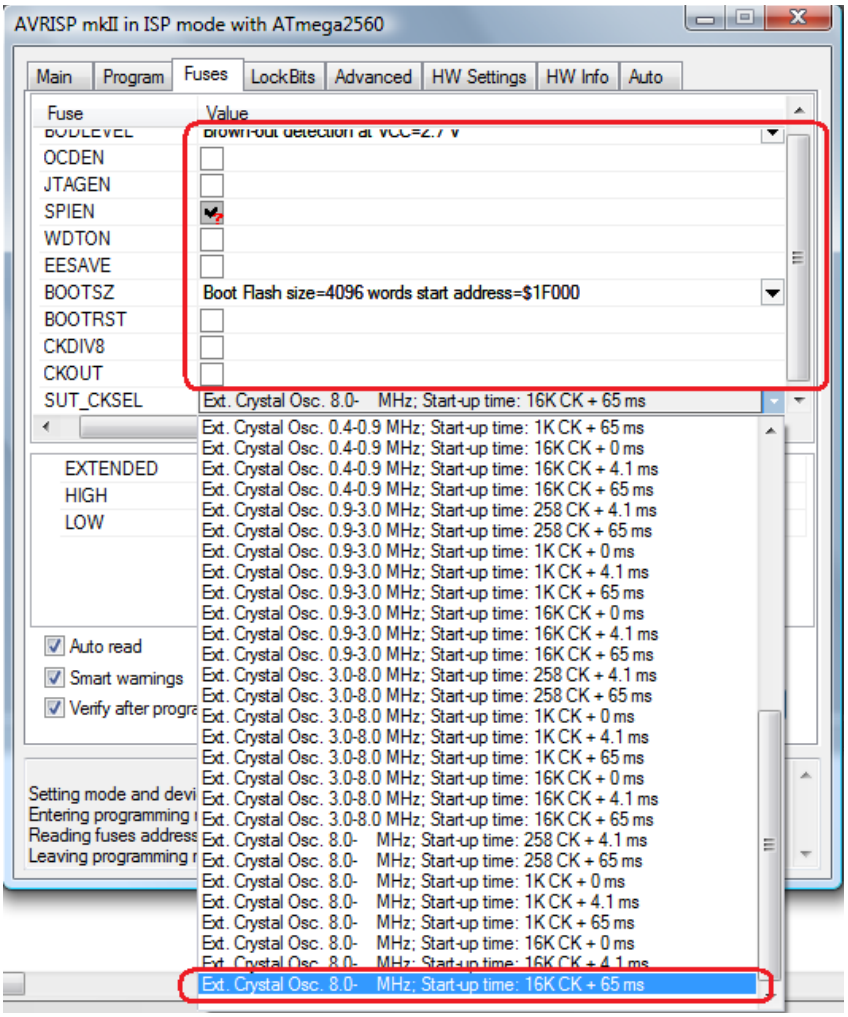


Figure 2.50: Fuse settings of ATMEGA2560 microcontroller

Following fuse settings are done:

1. Brown-out detection set at 2.7V (checked)
2. JTAG enabling is disabled (JTAGEN) (unchecked)
3. Boot size is selected at 2408 bytes.
4. BOOTRST is enabled. This will enable the boot mode detection at the PE7 of the microcontroller it PE7 is held logic low while microcontroller is reset. (checked)
5. Clock option (SUT CKSEL) set as external crystal of more than 8MHz i.e. “Ext. Crystal Osc. 8.0- MHz; Start-up time: 16K CK + 65ms”

2.10 Fuse settings for ATMEGA8 (Slave) microcontroller

All the microcontroller's Fuse settings are factory set at the NEX Robotics before shipping the robot. Do not change them. This information is only given for the reference.

All the Fuse settings are done using AVRISP mkII programmer. For starting AVRISP mkII refer to section 2.8.

To check the fuse settings click on the Fuse tab. Following window shown in figure 2.51 will appear. To verify the fuse settings press "Read". To write fuse settings after modifications press "Program". Make sure that "Auto Read", "Smart warning" and "Verify after programming" are checked.

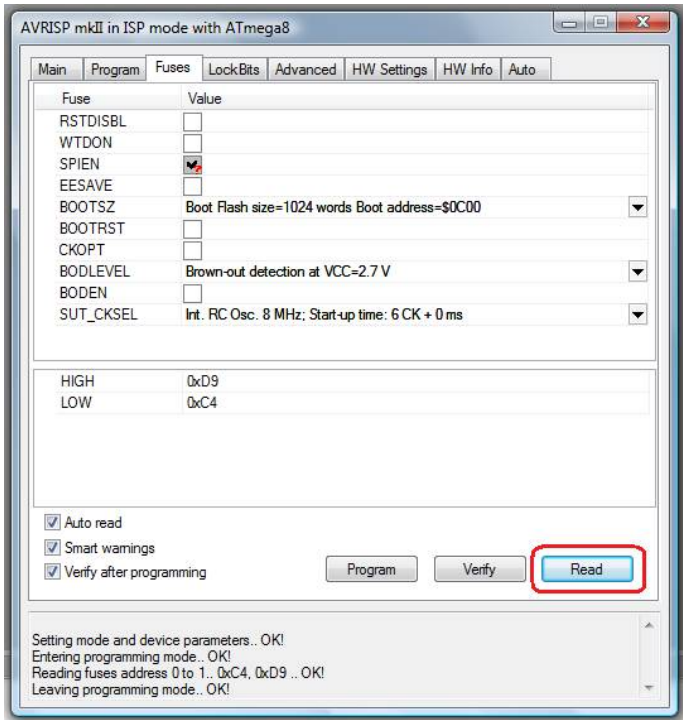


Figure 2.51

Following fuse settings are done:

1. Brown-out detection set at 2.7V (checked)
2. Clock option (SUT CKSEL) set as internal RC oscillator set at 8MHz i.e. "Int. RC Osc. 8MHz; Start-up time: 6CK + 0ms"

Important:

1. Never ever ever ever … select external crystal oscillator option in "SUT_CKSEL" for the ATMEGA8 microcontroller. As ATMEGA8 doesn't have the crystal oscillator it will not able to respond to anything from any device and we have to replace the microcontroller.
2. Before loading hex file or reading fuse settings ensure that jumpers at J4 are open as shown in figure 2.38 and are not as shown in figure 2.39.

Firmware for the ATMEGA8 (Slave microcontroller)

Firmware for the ATMEGA8 microcontroller is located in the folder "GUI and Related Firmware \ ATMEGA8 hex file"

2.11 Loading bootloader code on the ATMEGA2560 microcontroller

Fire Bird V ATMEGA2560 robot is factory shipped with the bootloader. If you do In System Programming using any ISP programmers then bootloader will get erased. To load the bootloader you will need the ISP programmer.

You need to load "M2560-14_7456MHz. USB-UART 2.a90" on the ATMEGA2560 microcontroller. This file is located in the documentation CD with the following path: "\Software and Drivers\AVR Bootloader\AVR Bootloader Microcontroller firmware"

To load this file follow the exactly same procedure as described in the step 4 of the section 2.8. Make sure that ATMEGA2560 has exactly the same fuse settings as described in the section 2.9.

2.12 Loading your code on the robot using AVR USB programmer from NEX Robotics

USBasp, is an USB port based programmer from NEX Robotics. From 10th March 2012 onwards we have stopped supplying this programmer with the robot. New robots comes with STK500V2 programmer from NEX Robotics.

This programmer requires avrdude.exe file which comes with WINAVR. So installation of WINAVR is necessary for this programmer. How to load this hex file is covered in detail in the programmer's documentation. It is located in the "AVR USB Programmer Documentation" folder inside the documentation CD.

2.13 Loading your code on the robot using STK500V2 AVR USB programmer from NEX Robotics

NEX AVR USB ISP STK500V2 is a high speed USB powered STK500V2 compatible In-System USB programmer for AVR family of microcontrollers. The compatibility with different windows platform is given in table below. For more information on how to use this programmer, refer to its manual located in the folder "AVR USB ISP STK500V2" in the robot's documentation CD. The drivers can be found in the same folder.

Compatibility Chart

Operating System	AVR Studio (CDC)	Avrdude (HID)
Windows XP	YES	YES
Windows Vista	X	YES
Windows 7	X	YES

Table 2.1

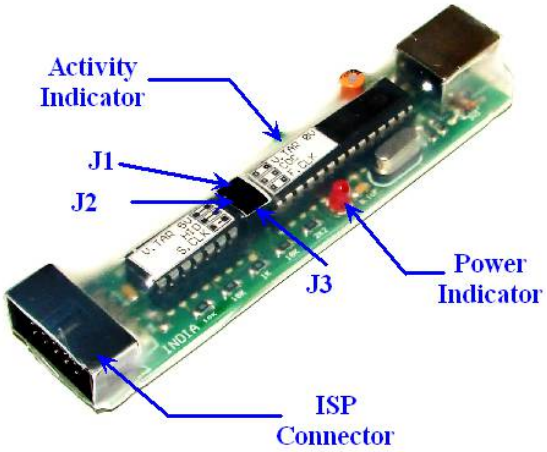


Figure 2.52: NEX AVR USB ISP STK500V2

3. Input / Output Operations on the Robot

ATMEGA2560 microcontroller has ten 8 bit ports from PORT A to PORT F and PORT H to PORT L (no PORT I) and PORT G has 6 bits. Input/output operations are the most essential, basic and easy operations.

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

Function	Pins	Input / Output	Recommended Initial State
Robot Direction control	PA3 to PA0	Output	Logic 0
LCD display control	PC0 to PC2 PC4 to PC7	Input / Output	Logic 0
On Board Interrupt Switch	PE7(INT7)	Input	Pulled up*
Buzzer control	PC3	Output	Logic 0
Sharp & White line sensor control	PG2	Output	Logic 1 **
Side Sharp Enable	PH2	Output	Logic 1***
IR Proximity Analog sensor enable	PH3	Output	Output **** Logic 1

Table 3.1

Note:

* In the AVR microcontrollers while pin is used as input it can be pulled up internally by using software enabled internal pull-up resistor. This internal pull-up as name indicates pulls up the floating pin towards Vcc. This makes input pin less susceptible to noise.

** Power to the Sharp IR range sensor 2, 3, 4 and red LEDs of the white line sensor can be turned on permanently by jumper 1 marked as J1 on the main board. In order to control illumination by microcontroller, jumper J1 must be removed. For more details refer to the section 3.10, 3.11 and 3.12 of the Fire Bird V Hardware Manual.

*** Power to the Sharp IR range sensor 1, 5 can be turned on permanently by jumper 3 marked as J3 on the main board. In order to control illumination by microcontroller jumper J3 must be removed. For more details refer to the section 3.10, 3.11 and 3.12 of the Fire Bird V Hardware Manual.

**** Fire Bird V has eight IR proximity Analog sensors out of which five are interfaced directly to ATMEGA2560. Power to these eight IR proximity sensors can be turned on permanently by jumper 2 marked as J2 on the main board. In order to control illumination by microcontroller jumper J2 must be removed. For more details refer to the section 3.10, 3.11 and 3.12 of the Fire Bird V Hardware Manual.

By disabling these sensors we can reduce current consumption by about 300mA and also allow many robots work in the same field without interfering with each others sensors by turning them on and off in a predetermined schedule which can be synchronized via wireless communication between these robots using XBee wireless module.

3.1 Registers for using PORTs of the ATMEGA2560 microcontroller

Each pin of the port can be addressed individually. Each pin individually can be configured as input or as output. While pin is input it can be kept floating or even pulled up by using internal pull-up. While pin is in the output mode it can be logic 0 or logic 1. To configure these ports as input or output each of the port has three associated I/O registers. These are Data Direction Register (DDRx), Port Drive Register (PORTx) and Port pins register (PINx) where 'x' is A to L (except I) indicating particular port name.

A. Data Direction Register (DDRx)

The purpose of the data direction register is to determine which bits of the port are used for input and which bits are used for output. If logic one is written to the pin location in the DDRx, then corresponding port pin is configured as an output pin. If logic zero is written to the pin location in the DDRx, then corresponding port pin is configured as an input pin.

```
DDRA = 0xF0; //sets the 4 MSB bits of PORTA as output port and
             //4 LSB bits as input port
```

B. Port Drive Register (PORTx)

If the port is configured as output port, then the PORTx register drives the corresponding value on output pins of the port.

```
DDRA = 0xFF; //set all 8 bits of PORTA as output
PORTA = 0xF0; //output logic high on 4 MSB pins and logic low on 4 LSB pins
```

For pins configured as input, we can instruct the microcontroller to apply a pull up register by writing logic 1 to the corresponding bit of the port driver register.

```
DDRA = 0x00; //set all 8 bits of PORTA as input
PORTA = 0xF0; //pull-up registers are connected on 4 MSB pins and 4 LSB pins are floating
```

C. Port pins register (PINx)

Reading from the input bits of port is done by reading port pin register

```
x = PINA; //read all 8 pins of port A
```

DDRx	PORTx	I/O	Pull-up	Comments
0	0	Input	No	floating input
0	1	Input	Yes	Will source current if externally pulled low
1	0	Output	No	Output Low (Sink)
1	1	Output	No	Output High (source)

Table 3.2

Note:

- 'X' represents port name – A, B, C, D, E, F, G, H, J, K, L.
- Tri-State is the floating pin condition.
- For more details, refer to ATMEGA2560 datasheet which is located in the "Datasheets" folder in the documentation CD.

Example:
Make PORTA 0-3 bits as output and PORTA 4-7 bits input.
Add pull-up to pins PORTA 4 and PORTA 5.
Output of PORTA 0 and PORTA 2 = 1; PORT A 1 and PORT A 3 = 0;

Pin	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
DDRA	0 (i/p)	0 (i/p)	0 (i/p)	0 (i/p)	1 (o/p)	1 (o/p)	1 (o/p)	1 (o/p)
PORTA	0	0	1 (↑)	1 (↑)	0	1	0	1
Status	i/p Floating	i/p Floating	i/p Pull-up	i/p Pull-up	o/p Low	o/p High	o/p Low	o/p High

Table 3.3

```
{
unsigned char k;
DDRA = 0x0F; //Make PA4 to PA7 pins input and PA0 to PA3 pins output
PORTA = 0x35; //Make PA7, PA6 floating; PA5, PA4 pulled-up; PA2, PA0 logic 0,
PA3, PA1 logic 1;
k = PINA; //Reads all the data from PORTA
while (1);
}
```

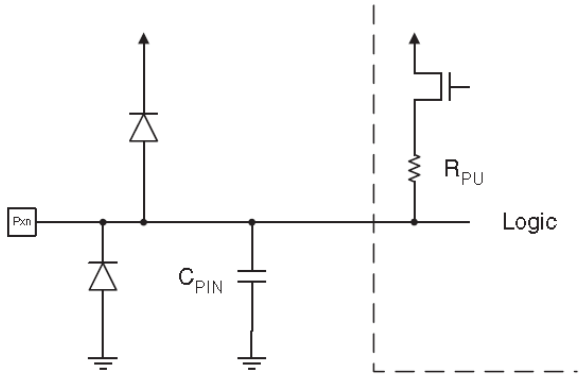


Figure 3.1: I/O pin equivalent schematic.
Source: ATMEGA2560 datasheet

All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both VCC and Ground as indicated in Figure.

D. To disable pull-ups of all the ports we need to set Bit 2 of SFIOR to logic one.

Special Function I/O register – SFIOR

Pin	TSM	-	-	-	ACME	PUD	PSR0	PSR321
Read/ Write	R/W	R	R	R	R/W	R/W	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0

Table 3.4

Bit 2-PUD: Pull-Up Disable

When this bit is written to one, the pull-ups in all the I/O ports are disabled even if the DDRxn and PORTxn Registers are configured to enable the pull-ups ({DDRxn, PORTxn} = 0b01).

E. Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port. Where ‘x’ is the port name and ‘n’ is the bit number.

3.2 ATMEGA2560 microcontroller pin configuration

PIN NO	Pin name	USED FOR	Status
1	(OC0B)PG5	Slave Select (SS) of the SPI expansion port on the main board (refer to figure 3.5)	--
2	RXD0/PCINT8/PE0	UART 0 receive for XBee wireless module (if installed)	Default
3	TXD0/PE1	UART 0 transmit for XBee wireless module (if installed)	Default
4	XCK0/AIN0/PE2	GPIO* (Available on expansion slot of the microcontroller socket)	
5	OC3A/AIN1/PE3	PWM output for C2 motor drive	Output
6	OC3B/INT4/PE4	External Interrupt for the left motor's position encoder	Input
7	OC3C/INT5/PE5	External Interrupt for the right motor's position encoder	Input
8	T3/INT6/PE6	External Interrupt for the C2 motor's position encoder	Input
9	CLK0/ICP3/INT7/ PE7	External Interrupt for Interrupt switch on the microcontroller board, External Interrupt for the C1 motor's position encoder, Connection to TSOP1738 if pad is shorted, can also be used as Boot loading switching ********	Input
10	VCC	5V	
11	GND	Ground	
12	RXD2/PH0	UART 2 receives for USB Communication. For more details refer to section 3.19.7	Default
13	TXD2/PH1	UART 2 transmit for USB Communication. For more details refer to section 3.19.7	Default
14	XCK2/PH2	IR proximity sensors 1 to 8 enable / disable. Turns off these sensors when output is logic 1 *******	Output
15	OC4A / PH3	Sharp IR ranges sensor 1and 5 enable / disable. Turns off these sensors when output is logic 1 *******	Output
16	OC4B / PH4	Connected to Rx pin of 1st Ultrasonic range sensor to trigger the ultrasonic triggering if sensor is mounted. Also Available on expansion slot of the microcontroller socket	--
17	OC4C / PH5	Available on expansion slot of the microcontroller socket	--
18	OC2B / PH6	Available on expansion slot of the microcontroller socket	--
19	SS/PCINT0/PB0	ISP (In System Programming), SPI Communication with ATMEGA8 **, Connection to the SPI port on the main board. Also available on expansion slot of the microcontroller socket	
20	SCK/PCINT1/PB1		Output
21	MOSI/PCINT2/PB2		Output
22	MISO/PCINT3/PB3		Input
23	OC2A/PCINT4/PB4	Servo Pod GPIO	--
24	OC1A/PCINT5/PB5	PWM for Servo motor 1. ***	Output
25	OC1B/PCINT6/PB6	PWM for Servo motor 2. ***	Output
26	OC0A/OC1C/PCINT7/PB7	PWM for Servo motor 3. ***	Output
27	T4/PH7	GPIO (Available On Expansion Slot)	--
28	TOSC2/PG3	RTC (Real Time Clock)****	
29	TOSC1/PG4		
30	RESET	Microcontroller reset	
31	VCC	5V	
32	GND	Ground	
33	XTAL2	Crystal 14.7456 MHz	
34	XTAL1		
35	ICP4/PL0	Connected to RSSI pin of XBee module. Also Available on expansion slot of the microcontroller socket.	--

36	ICP5/PL1	Available on expansion slot of the microcontroller socket.	--
37	TS/PL2	Available on expansion slot of the microcontroller socket.	--
38	OC5A/PL3	PWM for left motor.	Output
39	OC5B/PL4	PWM for right motor.	Output
40	OC5C/PL5	PWM for C1 motor.	Output
41	PL6	GPIO* (Available on expansion slot of the microcontroller socket)	--
42	PL7		--
43	SCL/INT0/PD0	I2C bus / GPIOs (Available on expansion slot of the microcontroller socket)	--
44	SDA/INT1/PD1		--
45	RXD1/INT2/PD2	UART1 receive for RS232 serial communication	Default
46	TXD1/INT3/PD3	UART1 transmit for RS232 serial communication	Default
47	ICP1/PD4	GPIO* (Available on expansion slot of the microcontroller socket)	--
48	XCK1/PD5		--
49	T1/PD6		--
50	T0/PD7		--
51	PG0/WR	GPIO* (Available on expansion slot of the microcontroller socket)	--
52	PG1/RD		--
53	PC0	LCD control line RS (Register Select)	Output
54	PC1	LCD control line RW(Read/Write Select)	Output
55	PC2	LCD control line EN(Enable Signal)	Output
56	PC3	Buzzer	Output
57	PC4	LCD data lines (4-bit mode)	Output
58	PC5		
59	PC6		
60	PC7		
61	VCC	5V	
62	GND	Ground	
63	PJ0/RXD3/PCINT9	LED bargraph display and GPIO* (Available on expansion slot of the microcontroller socket)	Output
64	PJ1/TXD3/PCINT10		
65	PJ2/XCK3/PCINT11		
66	PJ3/PCINT12		
67	PJ4/PCINT13		
68	PJ5/PCINT14		
69	PJ6/PCINT15		
70	PG2/ALE	Red LEDs of white line sensor enable/disable. ******* Turns off these sensors when output is logic 1	Output
71	PA7 C2-2	Logic input 2 for C2 motor drive	Output
72	PA6 C2-1	Logic input 1 for C2 motor drive	Output
73	PA5 C1-2	Logic input 2 for C1 motor drive	Output
74	PA4 C1-1	Logic input 1 for C1 motor drive	Output
75	PA3	Logic input 1 for Right motor (Right back)	Output
76	PA2	Logic input 2 for Right motor (Right forward)	Output
77	PA1	Logic input 2 for Left motor (Left forward)	Output
78	PA0	Logic input 1 for Left motor (Left back)	Output
79	PJ7	LED Bar Graph and GPIO* (Available on expansion slot of the microcontroller socket)	

80	VCC	5V	
81	GND	Ground	
82	PK7/ADC15/PCINT23	ADC Input For Servo Pod 2	Input (Floating)
83	PK6/ADC14/PCINT22	ADC Input For Servo Pod 1	Input (Floating)
84	PK5/ADC13/PCINT21	ADC input for Sharp IR range sensor 5	Input (Floating)
85	PK4/ADC12/PCINT20	ADC input for Sharp IR range sensor 4	Input (Floating)
86	PK3/ADC11/PCINT19	ADC input for Sharp IR range sensor 3	Input (Floating)
87	PK2/ADC10/PCINT18	ADC input for Sharp IR range sensor 2	Input (Floating)
88	PK1/ADC9/PCINT17	ADC input for Sharp IR range sensor 1	Input (Floating)
89	PK0/ADC8/PCINT16	ADC input for IR proximity analog sensor 5	Input (Floating)
90	PF7(ADC7/TDI)	ADC input for IR proximity analog sensor 4*****	Input (Floating)
91	PF6/(ADC6/TD0)	ADC input for IR proximity analog sensor 3*****	Input (Floating)
92	PF5(ADC5/TMS)	ADC input for IR proximity analog sensor 2*****	Input (Floating)
93	PF4/ADC4/TCK	ADC input for IR proximity analog sensor 1*****	Input (Floating)
94	PF3/ADC3	ADC input for white line sensor 1	Input (Floating)
95	PF2/ADC2	ADC input for white line sensor 2	Input (Floating)
96	PF1/ADC1	ADC input for white line sensor 3	Input (Floating)
97	PF0/ADC0	ADC input for battery voltage monitoring	Input (Floating)
98	AREF	ADC reference voltage pin (5V external) ******	
99	GND	Ground	
100	AVCC	5V	

Table 4.1: ATMEGA2560 microcontroller pin connections

* Not used pins are by default initialized to input and kept floating. These pins are available on the expansion slot of the ATMEGA2560 microcontroller adapter board. Some pins are especially reserved for servo motor interfacing for the Fire Bird V Hexapod robot.
** MOSI, MISO, SCK and SS pins of ATMEGA2560 are associated to the ISP (In System programming) port as well as the SPI interface to ATMEGA8. J4 needs to be disconnected before doing ISP. To communicate with ATMEGA8 jumper J4 needs to be in place. For more details refer to section 3.19.6.

*** PORTB pin5, 6, 7 are OC1A, OC1B, OC1C of the Timer1. These pins are connected to the servo motor sockets S1, S2, S3 on the microcontroller adapter board.

**** External Crystal of 32 KHz is connected to the pins PG3 and PG4 to generate clock for RTC (Real Time Clock).

***** For using Analog IR proximity (1, 2, 3 and 4) sensors short the jumper J2. To use JTAG or interface external analog sensors via expansion slot of the microcontroller socket remove these jumpers.

****** AREF can be obtained from the 5V microcontroller or 5V analog reference generator IC REF5050 (optional). For more details refer to section 3.19.9.

******* Sensor's switching can be controlled only is if corresponding jumpers are open. For more details refer to section 3.11 and 3.12.
Sharp jumper on J1 of main Board
IRP jumper on J1 of main Board
White Line jumper on J1 of main Board

******** External interrupt from the position encoder C1 is disabled by removing short on pad P1 near CD40106 Schmitt trigger inverter buffer to avoid its wire ANDing with the interrupt switch. Refer section 3.9 and 3.19.3 for details.

3.3 Application example Buzzer Beep

Located in the folder “Experiments \ Buzzer_Beep” folder in the documentation CD.

In the previous chapter, we have loaded buzzer beep code in Fire Bird V. Now we will see in detail the structure of this code.

This experiment demonstrates the simple operation of Buzzer ON/OFF with one second delay. Buzzer is connected to PORTC 3 pin of the ATMEGAM2560

Concepts covered: Output operation, generating exact delay

Note: Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega2560
Frequency: 14745600
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

```
//Buzzer is connected at the third pin of the PORTC
//To turn it on make PORTC 3rd pin logic 1

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//Function to initialize Buzzer
void buzzer_pin_config (void)
{
 DDRC = DDRC | 0x08;              //Setting PORTC 3 as output
 PORTC = PORTC & 0xF7;                    //Setting PORTC 3 logic low to turnoff buzzer
}

void port_init (void)
{
 buzzer_pin_config();
}

void buzzer_on (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore | 0x08;
 PORTC = port_restore;
}

void buzzer_off (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore & 0xF7;
```

```
 PORTC = port_restore;
}

void init_devices (void)
{
 cli(); //Clears the global interrupts
 port_init();
 sei(); //Enables the global interrupts
}

//Main Function
int main(void)
{
        init_devices();
        while(1)
        {
                buzzer_on();
                _delay_ms(1000);                //delay
                buzzer_off();
                _delay_ms(1000);                //delay
        }
}
```

In this code, first three lines represent the header file declaration. The # include directive is used for including header files in the existing code. The syntax for writing header file is as follows: **#include <avr/io.h>**

This # include directive will add the already existing io.h header file stored in avr folder under WINAVR folder. The same way other header files are also included in the main program so that we can use various utilities defined in the header files.

In all the codes we will configure pins related to any particular module in the *xxxx_pin_config()* functions. In this example code we have used the function *buzzer_pin_config()*. Buzzer is connected to the PORTC 3 pin of the microcontroller. PORTC 3 is configured as output with the initial value set to logic 0 to keep buzzer off at the time of port initialization. All the *xxxx_pin_config()* functions will be initialized in the *port_init()* function in all the codes as a convention. Function *init_devices()* will be used to initialize all the peripherals of the microcontroller as a convention.

In the above code buzzer is turned on by calling function *buzzer_on()*.
_delay_ms(1000) introduces delay of 1 second.
Buzzer is turned off by calling function *buzzer_off()*.
Again *_delay_ms(1000)* introduces delay of 1 second.
All these statements are written in *while(1)* loop construct to make buzzer on-off periodically.

3.4 Application example Simple Input – Output operation

Located in the folder "Experiments \ I-O Interfacing" folder in the documentation CD.

This experiment demonstrates simple Input and Output operation. When switch is pressed buzzer and bargraph LED display is turned on. When switch is opened buzzer and bargraph display are turned off. Refer to folder "Experiments \ I-O Interfacing" folder in the documentation CD to look at the program.

Concepts covered: Input and Output operations

Connections:
Buzzer: PORTC 3
LED bargraph: PORTJ 7 to PORTJ 0
Interrupt switch: PORTE 7

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code

    Microcontroller: ATMEGA2560
    Frequency: 14745600
    Optimization: -O0
    (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. Jumper J3 is in place to enable LED bargraph display on the ATMEGA2560 microcontroller adaptor board

3.5 Robot direction control

Located in the folder "Experiments \ Motion_Control_Simple" folder in the documentation CD.

Hardware aspects of the motion control are covered in detail in the section 3.8 and 3.9 of the Hardware Manual. Robot's motors are controlled by L293D motor controller from ST Microelectronics. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are applied to IC L293D's direction pins. Velocity control is done using pulse width modulation (PWM) applied to Enable pins of L293D IC. For more information, refer to section 3.8 and 3.9 of the Hardware Manual.

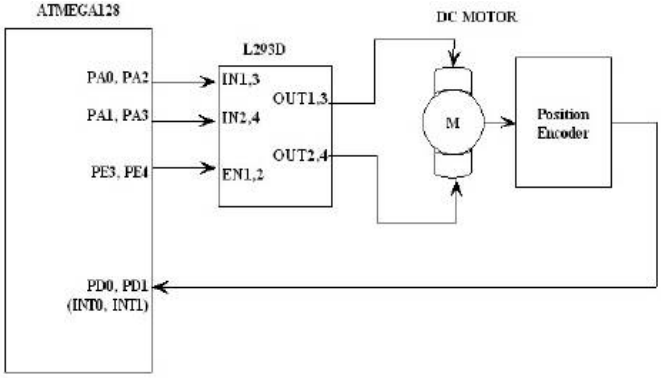


Figure 3.2

DIRECTION	LEFT BWD (LB) PA0 (L1)	LEFT FWD(LF) PA1 (L2)	RIGHT FWD(RF) PA2 (R1)	RIGHT BWD(RB) PA3 (R2)	PWM PL3 (PWML) for left motor PL4 (PWMR) for right motor
FORWARD	0	1	1	0	As per velocity requirement
REVERSE	1	0	0	1	As per velocity requirement
RIGHT *(Left wheel forward, Right wheel backward)*	0	1	0	1	As per velocity requirement
LEFT*(Left wheel backward, Right wheel forward,)*	1	0	1	0	As per velocity requirement
SOFT RIGHT*(Left wheel forward,, Right wheel stop)*	0	1	0	0	As per velocity requirement
SOFT LEFT*(Left wheel stop, Right wheel forward,)*	0	0	1	0	As per velocity requirement
SOFT RIGHT 2 *(Left wheel stop, Right wheel backward)*	0	0	0	1	As per velocity requirement
SOFT LEFT 2 *(Left wheel backward, Right wheel stop)*	1	0	0	0	As per velocity requirement
HARD STOP	0	0	0	0	As per velocity requirement
SOFT STOP (Free running stop)	X	X	X	X	0

Table 3.4: Logic table for motor direction control

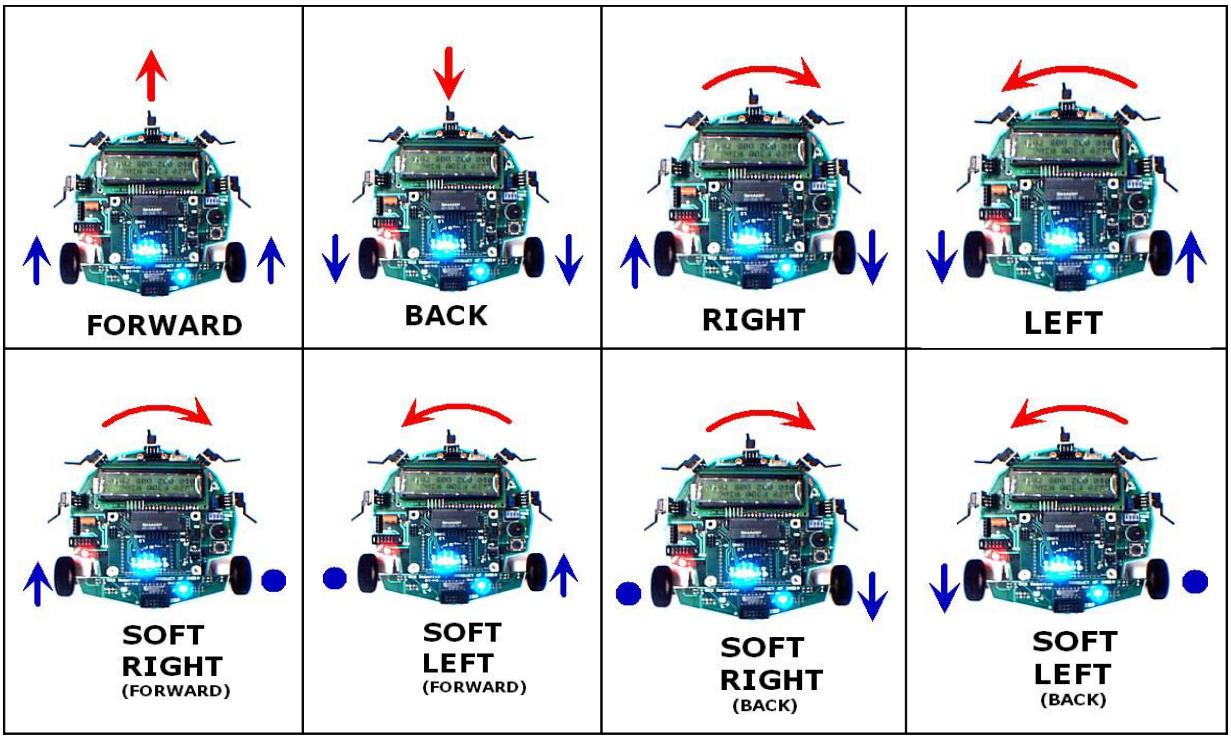


Figure 3.3

Note:

- All the soft turns should be used when you need more accuracy during turning
- Soft left 2 and Soft right 2 motions are very useful in grid navigation.

Application example: Robot direction control

Located in the folder "Experiments \ Motion_Control_Simple" folder in the documentation CD.

This experiment demonstrates simple motion control.

Concepts covered: Simple motion control using I-O interfacing

There are two components to the motion control:
1. Direction control using pins PORTA0 to PORTA3
2. Velocity control by PWM on pins PL3 and PL4 using OC5A and OC5B of timer 5.

In this experiment for the simplicity PL3 and PL4 are kept at logic 1.

Connections:

Microcontroller Pin	Function
PL3 (OC5A)	Pulse width modulation for the left motor (velocity control)
PL4 (OC5B)	Pulse width modulation for the right motor (velocity control)
PA0	Left motor direction control
PA1	Left motor direction control
PA2	Right motor direction control
PA3	Right motor direction control

Table 3.7: Pin functions for the motion control

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code

    Microcontroller: ATMEGA2560
    Frequency: 14745600
    Optimization: -O0
    (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. Auxiliary power can supply current up to 1 Ampere while Battery can supply current up to 2 Ampere. When both motors of the robot changes direction suddenly without stopping, it produces large current surge. When robot is powered by Auxiliary power which can supply only 1 Ampere of current, sudden direction change in both the motors will cause current surge which can reset the microcontroller because of sudden fall in voltage. It is a good practice to stop the motors for at least 0.5seconds before changing the direction. This will also increase the useable time of the fully charged battery.

3.6 Functions used by the robot for configuring various ports of the ATMEGA2560 microcontroller

3.6.1 Buzzer

Buzzer is connected to the PORTC 3 pin of the microcontroller.
Buzzer is turned on of logic 1 is applied at the PORTC 3 pin. For more information on the hardware refer to section 3.13 in the Hardware Manual.

3.6.1.1 buzzer_pin_config()

PORTC 3 pin is configured as output with the initial state set at logic 0 to keep the buzzer off.

```
void buzzer_pin_config (void)
{
 DDRC = DDRC | 0x08;                //Setting PORTC 3 as output
 PORTC = PORTC & 0xF7;              //Setting PORTC 3 logic low to turnoff buzzer
}
```

3.6.1.2 buzzer_on()

Turns on the buzzer by setting PORTC 3 pin to logic 1.

```
void buzzer_on (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore | 0x08;
 PORTC = port_restore;
}
```

3.6.1.3 buzzer_off()

Turns off the buzzer by setting PORTC 3 pin to logic 0.

```
void buzzer_off (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore & 0xF7;
 PORTC = port_restore;
}
```

3.6.2 Interrupt switch

Interrupt switch is connected to the PORTE 7 pin of the microcontroller. It has 10Kohm external pull-up resistor. When switch is pressed PORTE 7 pin is connected with the ground. Fore more details on the hardware refer to the section 3.19.10 in the Hardware Manual.

Interrupt switch is used as general purpose input device. Interrupt switch is configured as input with its internal pull-up resistor enabled.

```
void interrupt_switch_config (void)
{
 DDRE = DDRE & 0x7F;  //PORTE 7 pin set as input
 PORTE = PORTE | 0x80; //PORTE7 internal pull-up enabled
}
```

3.6.3 Bargraph LED display

Bargraph LED display is connected to the port J of the microcontroller. It can be used as general purpose LED display to display data or information for debugging.

```
void LED_bargraph_config (void)
{
 DDRJ = 0xFF;  //PORT J is configured as output
 PORTJ = 0x00; //Output is set to 0
}
```

Note: To use this display make sure that Jumper J3 is enabled on the microcontroller adaptor board. For more details, refer to section 3.19.6 of the Hardware Manual.

3.6.4 Robot motion control

For more information on the hardware, refer to section 3.8 and 3.9 from the Hardware Manual and table 3.7 for the hardware connection details, table 3.6 for control logic from this manual.

3.6.4.1 motion_pin_config()

Sets the directions and logic levels of the pins involved in the motion control.

```
void motion_pin_config (void)
{
 DDRA = DDRA | 0x0F; //set direction of the PORTA 3 to PORTA 0 pins as output
 PORTA = PORTA & 0xF0; // set initial value of the PORTA 3 to PORTA 0 pins to logic 0
 DDRL = DDRL | 0x18;   //Setting PL3 and PL4 pins as output for PWM generation
 PORTL = PORTL | 0x18; //PL3 and PL4 pins are for velocity control using PWM
}
```

3.6.4.2 motion_set()

Used for setting appropriate logic values for controlling robots direction. It is called by other functions to set robot's direction.

```
void motion_set (unsigned char Direction)
{
 unsigned char PortARestore = 0;

 Direction &= 0x0F;            // removing upper nibbel as it is not needed
 PortARestore = PORTA;         // reading the PORTA's original status
 PortARestore &= 0xF0;                 // setting lower direction nibbel to 0
 PortARestore |= Direction;            // adding lower nibbel for direction command and
                                       // restoring the PORTA status
 PORTA = PortARestore;         // setting the command to the port
}
```

3.6.4.3 Robot direction set functions

Sets robot's direction

```
void forward (void) //both wheels forward
{
 motion_set(0x06);
}

void back (void) //both wheels backward
{
 motion_set(0x09);
}

void left (void) //Left wheel backward, Right wheel forward
{
 motion_set(0x05);
}

void right (void) //Left wheel forward, Right wheel backward
{
 motion_set(0x0A);
}

void soft_left (void) //Left wheel stationary, Right wheel forward
{
 motion_set(0x04);
}

void soft_right (void) //Left wheel forward, Right wheel is stationary
{
 motion_set(0x02);
}

void soft_left_2 (void) //Left wheel backward, right wheel stationary
{
 motion_set(0x01);
}

void soft_right_2 (void) //Left wheel stationary, Right wheel backward
{
 motion_set(0x08);
}

void stop (void) //hard stop if PORTL 3 and PORTL 4 pins are at logic 1
{
 motion_set(0x00);
}
```

3.6.5 Functions for Robot's sensors switching on / off

Using these function robots sensors can be switched on or off. Before using these functions, make sure that Jumper 2, 3, 4 are open on the main board. For more details refer to the section 3.10, 3.11 and 3.12 of the Hardware Manual.

Connections:

PORTG 2: Power control of Sharp IR Range sensor 2, 3, 4 and red LEDs of the white line sensors
PORTH 2: Power control of Sharp IR Range sensor 1, 5
PORTH 3: Power control of IR Proximity sensors 1 to 8

Note: If Jumper 2, 3, 4 are open on the main board then setting logic 1 at the pin turns off the corresponding set of sensors.

3.6.5.1 MOSFET_switch_config()

Sets direction of PORTG 2, PORTH 2 and PORTH 3 as output with initial value set to logic 0

```
void MOSFET_switch_config (void)
{
 DDRH = DDRH | 0x0C; //make PORTH 3 and PORTH 1 pins as output
 PORTH = PORTH & 0xF3; //set PORTH 3 and PORTH 1 pins to 0

 DDRG = DDRG | 0x04; //make PORTG 2 pin as output
 PORTG = PORTG & 0xFB; //set PORTG 2 pin to 0
}
```

3.6.5.2 Functions for controlling power delivered to the sensors

```
void turn_on_sharp234_wl (void)
//turn on Sharp IR range sensors 2, 3, 4 and white line sensor's red LED
{
  PORTG = PORTG & 0xFB;
}

void turn_off_sharp234_wl (void)
//turn off Sharp IR range sensors 2, 3, 4 and white line sensor's red LED
{
 PORTG = PORTG | 0x04;
}

void turn_on_sharp15 (void) //turn on Sharp IR range sensors 1,5
{
  PORTH = PORTH & 0xFB;
}

void turn_off_sharp15 (void) //turn off Sharp IR range sensors 1,5
{
 PORTH = PORTH | 0x04;
}
```

```
void turn_on_ir_proxi_sensors (void) //turn on IR Proximity sensors
{
 PORTH = PORTH & 0xF7;
}

void turn_off_ir_proxi_sensors (void) //turn off IR Proximity sensors
{
 PORTH = PORTH | 0x08;
}

void turn_on_all_proxy_sensors (void)
// turn on Sharp 2, 3, 4, red LED of the white line sensors, Sharp 1, 5 and IR proximity sensor
{
 PORTH = PORTH & 0xF3; //set PORTH 3 and PORTH 1 pins to 0
 PORTG = PORTG & 0xFB; //set PORTG 2 pin to 0
}

void turn_off_all_proxy_sensors (void)
// turn off Sharp 2, 3, 4, red LED of the white line sensors Sharp 1, 5 and IR proximity sensor
{
 PORTH = PORTH | 0x0C; //set PORTH 3 and PORTH 1 pins to 1
 PORTG = PORTG | 0x04; //set PORTG 2 pin to 1
}
```

3.6.6 Functions for configuring Position encoder pins

3.6.6.1 left_encoder_pin_config()

```
//Function to configure INT4 (PORTE 4) pin as input for the left position encoder
void left_encoder_pin_config (void)
{
 DDRE  = DDRE & 0xEF;  //Set the direction of the PORTE 4 pin as input
 PORTE = PORTE | 0x10; //Enable internal pull-up for PORTE 4 pin
}
```

3.6.6.2 right_encoder_pin_config()

```
//Function to configure INT5 (PORTE 5) pin as input for the right position encoder
void right_encoder_pin_config (void)
{
 DDRE  = DDRE & 0xDF;  //Set the direction of the PORTE 4 pin as input
 PORTE = PORTE | 0x20; //Enable internal pull-up for PORTE 4 pin
}
```

3.6.7 Functions for configuring servo motor control pins

3.6.7.1 servo1_pin_config();

```
//Configure PORTB 5 pin for servo motor 1 operation
void servo1_pin_config (void)
{
 DDRB  = DDRB | 0x20;  //making PORTB 5 pin output
 PORTB = PORTB | 0x20; //setting PORTB 5 pin to logic 1
}
```

3.6.7.2 servo2_pin_config();

```
//Configure PORTB 6 pin for servo motor 2 operation
void servo2_pin_config (void)
{
 DDRB  = DDRB | 0x40;  //making PORTB 6 pin output
 PORTB = PORTB | 0x40; //setting PORTB 6 pin to logic 1
}
```

3.6.7.3 servo3_pin_config();

```
//Configure PORTB 7 pin for servo motor 3 operation
void servo3_pin_config (void)
{
 DDRB  = DDRB | 0x80;  //making PORTB 7 pin output
 PORTB = PORTB | 0x80; //setting PORTB 7 pin to logic 1
}
```

3.6.8 lcd_port_config()

```
void lcd_port_config (void)
{
 DDRC = DDRC | 0xF7; //all the LCD pin's direction set as output
 PORTC = PORTC & 0x80; // all the LCD pins are set to logic 0 except PORTC 7
}
```

3.6.9 adc_pin_config()

```
void adc_pin_config (void)
{
 DDRF = 0x00; //set PORTF direction as input
 PORTF = 0x00; //set PORTF pins floating
 DDRK = 0x00; //set PORTK direction as input
 PORTK = 0x00; //set PORTK pins floating
}
```

3.6.10 spi_pin_config()

```
void spi_pin_config (void)
{
 DDRB = DDRB | 0x07;
 PORTB = PORTB | 0x07;
}
```

4. Robot Position Control Using Interrupts

Fire Bird V incorporates various interrupt handling mechanisms such as timer overflow interrupts, timer compare interrupts, serial interrupts for doing specific tasks. In this chapter, we will have a brief overview of interrupt concept and will implement external hardware interrupts for position estimation of robots using position encoders.

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

Each interrupt has a vector address assigned to it low in program memory. The compiler places the starting address of the associated interrupt service routine and a relative jump instruction at the vector location for each interrupt. When the interrupt occurs, the program completes executing its current instruction and branches to the vector location associated with that interrupt. The program then executes the relative jump instruction to the interrupt service routine (ISR) and begins executing the ISR. For more information on the interrupt vectors refer to table 14.1 in the ATMEGA2560 datasheet which is located in the “datasheet” folder in the documentation CD.

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

4.1 Using Interrupts

Interrupts needs to be initialized before they become active. Initializing interrupt is a three step process. The first step is to select the trigger type for the interrupt. We are using falling edge trigger. This is selected by setting bits in EICRA (INT3 to INT0) and EICRB (INT7 to INT4) registers. Second step is to unmask the interrupt that we want to use in the EIMSK register. In the third step we globally enable all the unmasked interrupts. To enable unmasked interrupts we need to set global interrupt enable bit in the status register (SREG). This is done by instruction “sei();”.

4.1.1 Registers involved

4.1.1.1 EICRA – External Interrupt Control Register A

Bit	7	6	5	4	3	2	1	0
	ISC31	**ISC30**	**ISC21**	**ISC20**	**ISC11**	**ISC10**	**ISC01**	**ISC00**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits

The External Interrupts 3 - 0 are activated by the external pins INT3:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 4.1. Edges on INT3:0 are registered

asynchronously. Pulses on INT3:0 pins wider than 50 nanoseconds will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low. When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its Interrupt Enable bit in the EIMSK Register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its Interrupt Flag bit (INTFn) in the EIFR Register before the interrupt is re-enabled.

ISCn1	ISCn2	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Table 4.1: Interrupt Sense Control

Note:

- n = 0, 1, 2 or 3.
- When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

4.1.1.2 EICRB – External Interrupt Control Register B

Bit	7	6	5	4	3	2	1	0
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:0 – ISC71, ISC70 - ISC41, ISC40: External Interrupt 7 - 4 Sense Control Bits
The External Interrupts 7 - 4 are activated by the external pins INT7:4 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 4.2. The value on the INT7:4 pins are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. Observe that CPU clock frequency can be lower than the XTAL frequency if the XTAL divider is enabled. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.

ISCn1	ISCn2	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Table 4.2: Interrupt Sense Control

Note:

- n = 7, 6, 5 or 4.
- When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.
- Compare table 4.1 and 4.2. Interrupt 0 to 3 and Interrupt 4 to 7 are bit different in nature.

4.1.1.3 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	INT7	**INT6**	**INT5**	**INT4**	**INT3**	**INT2**	**INT1**	**INT0**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable

When an INT7:0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Registers – EICRA and EICRB – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

4.1.1.4 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	INT7	**INT6**	**INT5**	**INT4**	**INT3**	**INT2**	**INT1**	**INT0**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:0 – INTF7:0: External Interrupt Flags 7 - 0

When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set (one), the MCU will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. These flags are always cleared when INT7:0 are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See “Digital Input Enable and Sleep Modes” on page 74 of the ATMEGA2560 datasheet for more information.

4.1.2 Functions for configuring interrupt pins (called inside the "port_init()" function)

```
//Function to configure INT4 (PORTE 4) pin as input for the left position encoder
void left_encoder_pin_config (void)
{
 DDRE  = DDRE & 0xEF;  //Set the direction of the PORTE 4 pin as input
 PORTE = PORTE | 0x10; //Enable internal pull-up for PORTE 4 pin
}

//Function to configure INT5 (PORTE 5) pin as input for the right position encoder
void right_encoder_pin_config (void)
{
 DDRE  = DDRE & 0xDF;  //Set the direction of the PORTE 4 pin as input
 PORTE = PORTE | 0x20; //Enable internal pull-up for PORTE 4 pin
}
```

4.1.3 Functions for configuring external interrupts for position encoders

```
void left_position_encoder_interrupt_init (void) //Interrupt 4 enable
{
 cli(); //Clears the global interrupt
 EICRB = EICRB | 0x02; // INT4 is set to trigger with falling edge
 EIMSK = EIMSK | 0x10; // Enable Interrupt INT4 for left position encoder
 sei();   // Enables the global interrupt
}

void right_position_encoder_interrupt_init (void) //Interrupt 5 enable
{
 cli(); //Clears the global interrupt
 EICRB = EICRB | 0x08; // INT5 is set to trigger with falling edge
 EIMSK = EIMSK | 0x20; // Enable Interrupt INT5 for right position encoder
 sei();   // Enables the global interrupt
}
```

4.1.4 Function for initialization of interrupts

```
//Function to initialize all the devices
void init_devices()
{
 cli(); //Clears the global interrupt
 left_position_encoder_interrupt_init();
 right_position_encoder_interrupt_init();
 sei();   // Enables the global interrupt
}
```

4.1.5 Interrupt Service Routine (ISR)

After initializing interrupts, the next step is to define the Interrupt Service Routine (ISR). ISR in AVR Studio can be written in two different ways.

α. ISR (INT0_vect)

β. SIGNAL(SIG_INTERRUPT0)

Both of these formats are valid syntactically but we will be using *ISR (INT0_vect)*

Various syntaxes for ISR are described in datasheet of ATMEGA2560 microcontroller and also in < iomxx0_1.h> files in winavr/avr/include/avr folder.

```
//ISR for right position encoder
ISR(INT5_vect)
{
 //Your code
}

//SR for left position encoder
ISR(INT4_vect)
{
//Your code
}
```

4.2 Robot position control using interrupts

Interrupt 4 (INT4) and interrupt 5 (INT5) are connected to the robot's position encoder. Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of optical encoder and slotted disc assembly. When this slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period indicates velocity. For more details on the hardware refer to section 3.8 and 3.9 from the Hardware Manual.

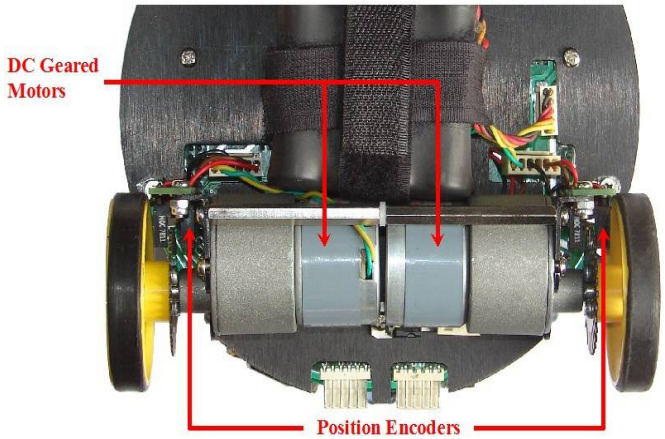


Figure 4.1: DC geared motors and position encoders

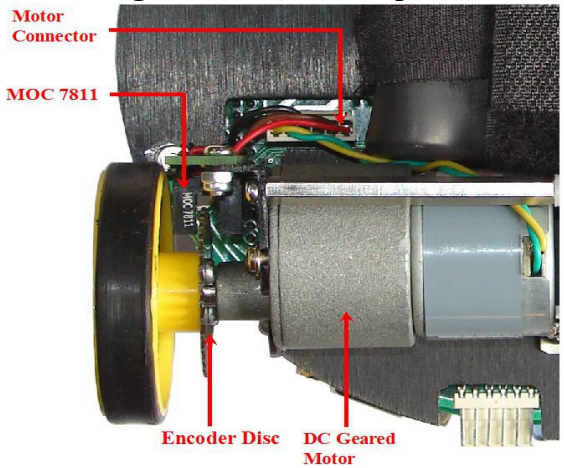


Figure 4.2: Position encoder assembly

4.2.1 Calculation of position encoder resolution:

Case 1: Robot is moving forward or backward (encoder resolution is in mm)

Wheel diameter: 5.1cm
Wheel circumference: 5.1cm * 3.14 = 16.014cm = 160.14mm
Number slots on the encoder disc: 30
Position encoder resolution: 163.2 mm / 30 = 5.44mm / pulse.

Case 2: Robot is turning with one wheel rotating clockwise while other wheel is rotating anti clockwise. Center of rotation is in the center of line passing through wheel axel and both wheels are rotating in opposite direction (encoder resolution is in degrees)

Distance between Wheels = 15cm

Radius of Circle formed in 360^0 rotation of Robot = Distance between Wheels / 2
= 7.5 cm

Distance Covered by Robot in 360^0 Rotation = Circumference of Circle traced
= 2 x 7.5 x 3.14
= 47.1 cm or 471mm

Number of wheel rotations of in 360^0 rotation of robot
= Circumference of Traced Circle / Circumference of Wheel
= 471 / 160.14
= 2.941

Total pulses in 360^0 Rotation of Robot
= Number of slots on the encoder disc / Number of wheel rotations of in 360^0 rotation of robot
= 30 x 2.941
= 88.23 (approximately 88)

Position Encoder Resolution in Degrees = 360 / 88
= 4.090 degrees per count

Case 3: Robot is turning with one wheel stationary while other wheel is rotating clockwise or anti clockwise. Center of rotation is center of the stationary wheel (encoder resolution is in degrees)

In this case only one wheel is rotating and other wheel is stationary so robot will complete its 360^0 rotation with stationary wheel as its center.
Radius of Circle formed in 360^0 rotation of Robot = Distance between Wheels
= 15 cm

Distance Covered by Robot in 360^0 Rotation = Circumference of Circle traced
= 2 x 15 x 3.14
= 94.20 cm or 942 mm

Number of wheel rotations of in 360^0 rotation of robot
= Circumference of Traced Circle / Circumference of Wheel
= 942 / 160.14
= 5.882

Total pulses in 360^0 Rotation of Robot
= Number of slots on the encoder disc / Number of wheel rotations of in 360^0 rotation of robot
= 30 x 5.882
= 176.46 (approximately 176)

Position Encoder Resolution in Degrees = 360 /176
= 2.045 degrees per count

4.2.2 Interrupt service routine for position encoder

4.2.2.1 ISR for right position encoder

```
//ISR for right position encoder
ISR(INT5_vect)
{
 ShaftCountRight++;  //increment right shaft position count
}
```

4.2.2.2 ISR for left position encoder

```
//ISR for left position encoder
ISR(INT4_vect)
{
 ShaftCountLeft++;  //increment left shaft position count
}
```

4.2.3 Functions for robot position control

4.2.3.1 Function for rotating robot by specific degrees

```
//Function used for turning robot by specified degrees
void angle_rotate(unsigned int Degrees)
{
 float ReqdShaftCount = 0;
 unsigned long int ReqdShaftCountInt = 0;

 ReqdShaftCount = (float) Degrees/ 4.090; // division by resolution to get shaft count
 ReqdShaftCountInt = (unsigned int) ReqdShaftCount;
 ShaftCountRight = 0;
 ShaftCountLeft = 0;

 while (1)
 {
  if((ShaftCountRight >= ReqdShaftCountInt) | (ShaftCountLeft >= ReqdShaftCountInt))
  break;
 }
 stop(); //Stop robot
}
```

4.2.3.2 Function for moving robot forward and back by specific distance

```
//Function used for moving robot forward by specified distance
void linear_distance_mm(unsigned int DistanceInMM)
{
 float ReqdShaftCount = 0;
 unsigned long int ReqdShaftCountInt = 0;

 ReqdShaftCount = DistanceInMM / 5.338; // division by resolution to get shaft count
 ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;

 ShaftCountRight = 0;
 while(1)
 {
  if(ShaftCountRight > ReqdShaftCountInt)
  {
      break;
  }
 }
 stop(); //Stop robot
}
```

4.2.3.3 Forward in mm

```
void forward_mm(unsigned int DistanceInMM)
{
 forward();
 linear_distance_mm(DistanceInMM);
}
```

4.2.3.4 Backward in mm

```
void back_mm(unsigned int DistanceInMM)
{
 back();
 linear_distance_mm(DistanceInMM);
}
```

4.2.3.5 left in degrees

```
void left_degrees(unsigned int Degrees)
{
// 88 pulses for 360 degrees rotation 4.090 degrees per count
 left(); //Turn left
 angle_rotate(Degrees);
}
```

4.2.3.6 right in degrees

```
void right_degrees(unsigned int Degrees)
{
// 88 pulses for 360 degrees rotation 4.090 degrees per count
 right(); //Turn right
 angle_rotate(Degrees);
}
```

4.2.3.7 soft left in degrees

```
void soft_left_degrees(unsigned int Degrees)
{
 // 176 pulses for 360 degrees rotation 2.045 degrees per count
 soft_left(); //Turn soft left
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

4.2.3.8 soft right in degrees

```
void soft_right_degrees(unsigned int Degrees)
{
 // 176 pulses for 360 degrees rotation 2.045 degrees per count
 soft_right();  //Turn soft right
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

4.2.3.9 soft left 2 in degrees

```
void soft_left_2_degrees(unsigned int Degrees)
{
 // 176 pulses for 360 degrees rotation 2.045 degrees per count
 soft_left_2(); //Turn reverse soft left
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

4.2.3.10 soft right 2 in degrees

```
void soft_right_2_degrees(unsigned int Degrees)
{
 // 176 pulses for 360 degrees rotation 2.045 degrees per count
 soft_right_2();  //Turn reverse soft right
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

4.2.4 Application example of robot position control

Located in the folder “Experiments \ Position_Control_Interrupts” folder in the documentation CD.

This experiment demonstrates use of position encoders.

Concepts covered: External Interrupts, Position control

Connections:
PORTA3 to PORTA0: Robot direction control
PL3, PL4: Robot velocity control. Currently set to 1 as PWM is not used
PE4 (INT4): External interrupt for left motor position encoder
PE5 (INT5): External interrupt for the right position encoder

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: ATMEGA2560
Frequency: 14745600
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)
It is observed that external interrupts does not work with the optimization level -Os

2. Auxiliary power can supply current up to 1 Ampere while Battery can supply current up to 2 Ampere. When both motors of the robot changes direction suddenly without stopping, it produces large current surge. When robot is powered by Auxiliary power which can supply only 1 Ampere of current, sudden direction change in both the motors will cause current surge which can reset the microcontroller because of sudden fall in voltage. It is a good practice to stop the motors for at least 0.5seconds before changing the direction. This will also increase the useable time of the fully charged battery.

5. Timer / Counter Operations on the Robot

ATMEGA2560 has 2 eight bit timers (timer 0 and timer 2) and 4 sixteen bit timers (timer 1, 3, 4 and 5). All the timers have independent Output Compare Units with PWM support. These timers can be used for accurate program execution timing (event management) and wave generation.

Fire Bird V uses these timers mainly for the following applications:

- Velocity control – Timer 5 is used to generate PWM for robot's velocity control.
- Servo motor control – Timer 1 is used in 10 bit fast PWM mode to control servo motors.
- Event scheduling – Timer with timer overflow interrupt is used for event scheduling.
- Velocity calculation – Timer can be used for robot's velocity estimation.

In the Fire Bird V ATMEGA2560 robot Timer 5 is used to generate PWM for robot velocity control. Timer 1 is used for servo motor control. All other timers are free and can be used for other purposes.

Note: Theory content of this chapter is based on the ATMEGA2560 datasheet which is located in the "datasheet" folder in the documentation CD.

General features of the 8 bit timers 0 and 2

- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

General features of the 16 bit timers 1, 3, 4 and 5

- True 16-bit Design (i.e., Allows 16-bit PWM)
- Three independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- Twenty independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1, TOV3, OCF3A, OCF3B, OCF3C, ICF3, TOV4, OCF4A, OCF4B, OCF4C, ICF4, TOV5, OCF5A, OCF5B, OCF5C and ICF5)

5.1 Important terms involved in the timers:

BOTTOM: The counter reaches the BOTTOM when it becomes 0x0000.

MAX: The counter reaches its MAX value when it becomes 0xFF (decimal 255) for 8 bit timer or 0xFFFF (decimal 65535) for 16 bit timer.

TOP: The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCRnA or ICRn Register. The assignment is dependent on the mode of operation.

Accessing 16-bit Registers
The TCNTn, OCRnA/B/C, and ICRn are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same Temporary Register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the Temporary Register for the high byte. Reading the OCRnA/B/C 16-bit registers does not involve using the Temporary Register. To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

Modes of operation in timers:

1. Normal mode
2. Clear timer on compare match (CTC) mode
3. Fast PWM mode
4. Phase correct PWM mode
5. Phase and frequency correct PWM mode

For more information on the timer operation refer to ATMEGA2560 datasheet.

5.2 16 bit Timer Registers

Note:

- In all the terms ‘n’ represents timer number which can be 1, 3, 4 or 5 and ‘X’ represents output compare channel number which can be A, B or C.
- For more detailed description refer to section 16 to section 20 of the ATMEGA2560 datasheet which is located in the “datasheet” folder in the documentation CD.

Clock source for the Timers

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select CSn2:0) bits located in the Timer/Counter control Register B (TCCRnB). Timer/Counter 0, 1, 3, 4, and 5 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to all Timer/Counters. Tn is used as a general name, n = 1, 3, 4 or 5.

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency (fCLK_I/O). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either fCLK_I/O/8, fCLK_I/O/64, fCLK_I/O/256, or fCLK_I/O/1024.

5.2.1 TCCRnA – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0
	COMnA1	**COMnA0**	**COMnB1**	**COMnB0**	**COMnC1**	**COMnC0**	**WGMn1**	**WGMn0**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A
Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B
Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C

The COMnA1:0, COMnB1:0, and COMnC1:0 control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bits are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver. When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnX1:0 bits is dependent of the WGMn3:0 bits setting. Table 5.1 shows the COMnA1:0, COMnB1:0 and COMnC1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OC5A, OCnB, OCnC disconnected
0	1	WGMn3:0 = 14 or 15: Toggle OCnA on Compare Match, OCnB and OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode).
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode).

Table 5.1: COMnX1:0 bit functionality when the WGMn3:0 bits are set to fast PWM mode.

Bit 1:0 – WGM51:0: Waveform Generation Mode
Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 5.3. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. For more information on the different modes, refer “Modes of Operation” on page 148 of the ATMEGA2560 datasheet.

5.2.2 TCCRnB – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0
	ICNCn	**ICESn**	**-**	**WGMn3**	**WGMn2**	**CSn2**	**CSn1**	**CSn0**
Read / Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ICNC5: Input Capture Noise Canceler
Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

Bit 6 – ICESn: Input Capture Edge Select
This bit selects which edge on the Input Capture Pin (ICP5) that is used to trigger a capture event. When the ICES5 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture. When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled. When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the input capture function is disabled.

Bit 5 – Reserved Bit
This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

Bit 4:3 – WGMn3:2: Waveform Generation Mode
See TCCRnA Register description and refer to table 5.3

Bit 2:0 – CSn2:0: Clock Select
The three clock select bits select the clock source to be used by the Timer/Counter.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Table 5.2: Clock select bit description

5.2.3 TCCRnC – Timer/Counter Control Register C

Bit	7	6	5	4	3	2	1	0
	FOCnA	**FOCnB**	**FOCnC**	-	-	-	-	-
Read / Write	W	W	W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – FOCnA: Force Output Compare for Channel A
Bit 6 – FOCnB: Force Output Compare for Channel B
Bit 5 – FOCnC: Force Output Compare for Channel C

The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnX1:0 bits setting. Note that the FOCnA/FOCnB/FOCnC bits are implemented as strobes. Therefore it is the value present in the COMnx1:0 bits that determine the effect of the forced compare. A FOCnA/FOCnB/FOCnC strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCRnA as TOP. The FOCnA/FOCnB/FOCnB bits are always read as zero.

Bit 4:0 – Reserved Bits
These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be written to zero when TCCRnC is written.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Table 5.3: Waveform generation mode bit description

5.2.4 TIMSKn – Timer/Counter n Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	-	-	ICIEn	-	OCIEnC	OCIEnB	OCIEnA	TOIEn
Read / Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 5 – ICIEn: Timer/Counter n, Input Capture Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter n Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 105 in the ATMEGA2560 datasheet.) is executed when the ICFn Flag, located in TIFRn, is set.

Bit 3 – OCIEnC: Timer/Counter n, Output Compare C Match Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter n Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 105 in the ATMEGA2560 datasheet.) is executed when the OCFnC Flag, located in TIFRn, is set.

Bit 2 – OCIEnB: Timer/Counter n, Output Compare B Match Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter n Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 105 in the ATMEGA2560 datasheet.) is executed when the OCFnB Flag, located in TIFRn, is set.

Bit 1 – OCIEnA: Timer/Counter n, Output Compare A Match Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter n Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (See "Interrupts" on page 105 in the ATMEGA2560 datasheet.) is executed when the OCFnA Flag, located in TIFRn, is set.

Bit 0 – TOIEn: Timer/Counter n, Overflow Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter n Overflow interrupt is enabled. The corresponding Interrupt Vector (See "Interrupts" on page 105 in the ATMEGA2560 datasheet.) is executed when the TOVn Flag, located in TIFRn, is set.

5.2.5 TIFRn – Timer/Counter n Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	-	-	**ICFn**	-	**OCFnC**	**OCFnB**	**OCFnA**	**TOVn**
Read / Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 5 – ICFn: Timer/Counter n, Input Capture Flag
This flag is set when a capture event occurs on the ICPn pin. When the Input Capture Register (ICRn) is set by the WGMn3:0 to be used as the TOP value, the ICFn Flag is set when the counter reaches the TOP value. ICFn is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICFn can be cleared by writing a logic one to its bit location.

Bit 3– OCFnC: Timer/Counter n, Output Compare C Match Flag
This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register C (OCRnC). Note that a Forced Output Compare (FOCnC) strobe will not set the OCFnC Flag. OCFnC is automatically cleared when the Output Compare Match C Interrupt Vector is executed. Alternatively, OCFnC can be cleared by writing a logic one to its bit location.

Bit 2 – OCFnB: Timer/Counter1, Output Compare B Match Flag
This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB). Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB Flag. OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB can be cleared by writing a logic one to its bit location.

Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag
This flag is set in the timer clock cycle after the counter (TCNTn value matches the Output Compare Register A (OCRnA). Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA Flag. OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

Bit 0 – TOVn: Timer/Counter n, Overflow Flag
The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to Table 5.3 for the TOVn Flag behavior when using another WGMn3:0 bit setting. TOVn is automatically cleared when the Timer/Counter n Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

5.3 Velocity control using PWM

5.3.1 Concept of PWM

Pulse width modulation is a process in which duty cycle of constant frequency square wave is modulated to control power delivered to the load i.e. motor.

Duty cycle is the ratio of **'T-ON/ T'**. Where **'T-ON'** is ON time and **'T'** is the time period of the wave. Power delivered to the motor is proportional to the **'T-ON'** time of the signal. In case of PWM the motor reacts to the time average of the signal.

PWM is used to control total amount of power delivered to the load without power losses which generally occur in resistive methods of power control.

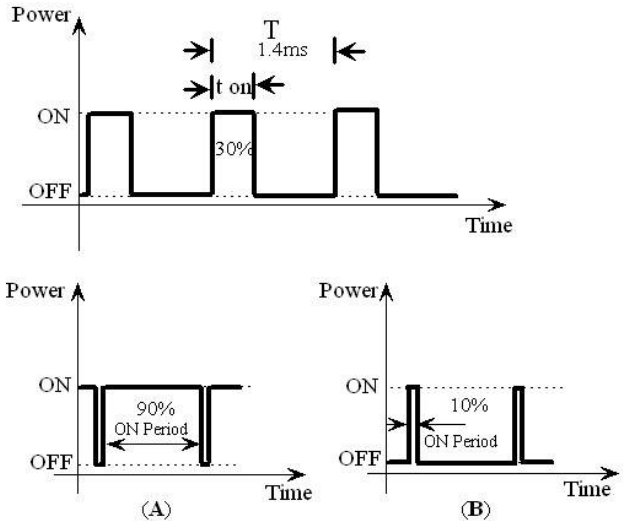


Figure 5.1: Pulse Width Modulation (PWM)

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

Microcontroller Pin	Function
PL3 (OC5A)	Pulse width modulation for the left motor (velocity control)
PL4 (OC5B)	Pulse width modulation for the right motor (velocity control)
PA0	Left motor direction control
PA1	Left motor direction control
PA2	Right motor direction control
PA3	Right motor direction control

Table 5.4: Pin functions for the motion control

5.3.2 PWM generation using Timer

PWM using Timer n

All 16 bit timers are identical in nature. We are using timer 5 for PWM as input pins of the motor driver IC L293D are connected to PL3 (OC5A) and PL4 (OC5B).

For robot velocity control Timer 5 is used in 8 bit fast PWM generation mode. In the non inverting compare output mode.

The counter counts from BOTTOM to MAX and again restarts from BOTTOM. In non-inverting compare output mode, the output compare (OC5X) is cleared on the compare match between TCNT5 and OCR5X, and set at BOTTOM. Where X is A, B or C. In inverting compare output mode output (OC5X) is set on compare match and cleared at BOTTOM.

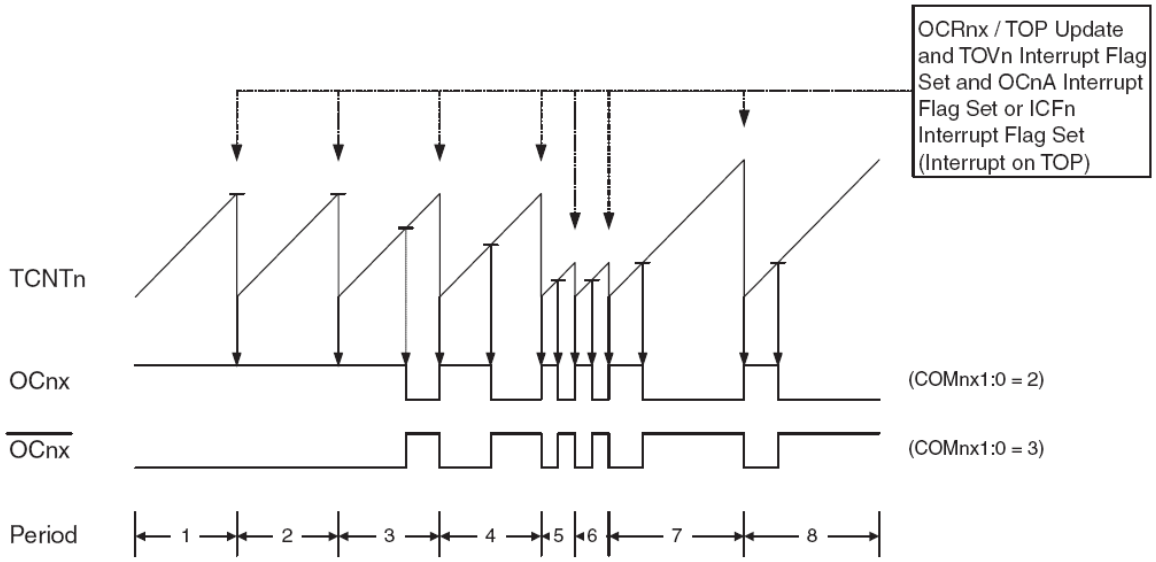


Figure 5.2: Time diagram for fast PWM mode

In 8 bit fast PWM mode the counter is incremented until the counter value matches either fixed value of 0x00FF hex and then value is rolled over again to 0. In the non-inverting PWM mode output on the output compare pins (OC5A, OC5B and OC5C in this case) is logic 0 when counter starts at 0. When counter value is matched with OCRnx (in this case OCR5AL, OCR5BL and OCR5CL output at the output at the corresponding compare pins (OC5A, OC5B and OC5C in this case) becomes logic 1. It stays at logic 1 till counter rolls over from 0xFF to 0. At the roll over value of these OCRnx pins is set to logic 0. To change the duty cycle of the PWM form 0 to 100% duty cycle in the 8 bit fast PWM generation mode value of OCRnx can be set between 0 to 255 (0x00 to 0xFF).

5.3.3 Timer 5 configuration in 8 bit fast PWM mode

Function Timer5_init() function initializes the function in 8 bit fast PWM generation mode.

```
// Timer 5 initialized in PWM mode for velocity control
// Prescale: 256
// PWM 8bit fast, TOP=0x00FF
// Timer Frequency:225.000Hz
void timer5_init()
{
        TCCR5B = 0x00;          //Stop
        TCNT5H = 0xFF;          //Counter higher 8-bit value to which OCR5xH value is compared with
        TCNT5L = 0x01;          //Counter lower 8-bit value to which OCR5xH value is compared with
        OCR5AH = 0x00;          //Output compare register high value for Left Motor
        OCR5AL = 0xFF;          //Output compare register low value for Left Motor
        OCR5BH = 0x00;          //Output compare register high value for Right Motor
        OCR5BL = 0xFF;          //Output compare register low value for Right Motor
        OCR5CH = 0x00;          //Output compare register high value for Motor C1
        OCR5CL = 0xFF;          //Output compare register low value for Motor C1
        TCCR5A = 0xA9;          //COM5A1=1, COM5A0=0; COM5B1=1, COM5B0=0; COM5C1=1
                                // COM5C0=0
//For Overriding normal port functionality to OCRnA outputs. WGM51=0, WGM50=1 Along With GM52 //in
TCCR5B for Selecting FAST PWM 8-bit Mode
        TCCR5B = 0x0B;          //WGM12=1; CS12=0, CS11=1, CS10=1 (Prescaler=64)
}
```

PWM frequency calculation:

PWM frequency = System Clock / N (1 + TOP)
= 14.7456 MHz / 256 (1 + 255)
= 225.000 Hz

Where
System clock = Crystal frequency = 14.7456MHz
Prescale = N = 256
TOP = 255 (8 bit resolution)

System Clock / Prescale	8-bit (TOP = 255)
System Clock	F_{pwm} = 57.600 KHz
System Clock / 8	F_{pwm} = 7.200 KHz
System Clock / 64	F_{pwm} = 900.000Hz
System Clock / 256	F_{pwm} = 225.000 Hz
System Clock / 1024	F_{pwm} = 56.250 Hz

Table 5.2: 8 bit PWM fast frequency for different prescale options

5.3.4 Function for timer 5 initialization

```
void init_devices (void) //use this function to initialize all devices
{
 cli(); //disable all interrupts
 timer5_init();
 sei(); //re-enable interrupts
}
```

cli(); disables all the interrupts and sei(); enables all the interrupts.

It is very important that all the devices should be configured after disabling all the interrupts. All the peripherals of the microcontroller will be configured inside init_devices() function.

5.3.5 Functions for PWM output pin configuration and robot's velocity control

5.3.5.1 Functions for PWM output pin configuration (called inside the "port_init()" function)

```
void motion_pin_config (void)
{
 DDRA = DDRA | 0x0F;
 PORTA = PORTA & 0xF0;
 DDRL = DDRL | 0x18;   //Setting PL3 and PL4 pins as output for PWM generation
 PORTL = PORTL | 0x18; //PL3 and PL4 pins are for velocity control using PWM.
}
```

5.3.5.2 Function for robot's velocity control

```
void velocity (unsigned char left_motor, unsigned char right_motor)
{
        OCR5AL = (unsigned char)left_motor;
        OCR5BL = (unsigned char)right_motor;
}
```

This function takes velocity for left motor and right motor as input parameter and assigns them to output compare register OCR5A and OCR5B. Channel A is used for left motor and channel B is used for right motor. Since we are using PWM in 8 bit resolution its ok to only loading lower byte of the OCR5A and OCR5B registers.

5.3.6 Application example for robot velocity control

Located in the folder "Experiments \ Velocity_Control_using_PWM" folder in the documentation CD.

This experiment demonstrates robot velocity control using PWM.

Concepts covered: Use of timer to generate PWM for velocity control

There are two components to the motion control:
1. Direction control using pins PORTA0 to PORTA3
2. Velocity control by PWM on pins PL3 and PL4 using OC5A and OC5B of timer 5.

Connections: Refer to table 5.1 for connection details.

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code
   Microcontroller: ATMEGA2560
   Frequency: 14745600
   Optimization: -O0
   (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. Auxiliary power can supply current up to 1 Ampere while Battery can supply current up to 2 Ampere. When both motors of the robot changes direction suddenly without stopping, it produces large current surge. When robot is powered by Auxiliary power which can supply only 1 Ampere of current, sudden direction change in both the motors will cause current surge which can reset the microcontroller because of sudden fall in voltage. It is a good practice to stop the motors for at least 0.5seconds before changing the direction. This will also increase the useable time of the fully charged battery.

5.4 Servo motor control using Timer 1

Fire Bird V robot has camera pod and sensor pod attachment. It has two servo motor for pan and tilt movement. Servo motor is an essentially geared DC motor which rotates from 0^0 to 180^0 based upon input signal in the form of pulse train.

Servo control is done by sending a PWM signal to the PWM input pin of the servo motor. The servo motor compares this signal to the actual position of the servo and adjusts the angle of the servo motor accordingly. Position of the servo motor is determined by the pulse width of waveform which has frequency between 30Hz to 60Hz. Generic servo motor gives 0 to 180 degrees rotation for pulse width of 1 to 2 milliseconds. 50Hz is considered as ideal frequency. The pulse width may change for different type of servo motor

We will be generating approximately 52.25 Hz (approx 19.13ms time period) PWM signal using timer 1. Pulse will remain high for first 1 to 2 ms depending on the angle we want to set and then pulse will remain low for the rest of the time.

Camera pod has two servo motors called as pan and tilt motors. Pan motor moves camera left and right. It has 180^0 swing. Tilt motor moves camera up and down. It can also give swing of 180^0 but after mounting camera swing gets restricted to 0 to 120^0.

We will control these servo motors using timer 1. Very high resolution PWM waveform can be generated using timer input capture or overflow interrupt along with output compare register (OCR) but it is bit complicated method and requires frequent interrupt servicing. We will be using a simpler method to generate control signal for these servo motors which gives relatively less resolution in servo angle control but does not require frequent interrupt servicing. Timer 1 will be used in 10 bit fast PWM mode which has 10 bit resolution. Servo motor can be connected to the three servo connectors (S1, S2 and S3) which are located on the ATMEGA2560 microcontroller adaptor board. For more details refer to section 3.19.11 from the Hardware Manual.

For concept of PWM and PWM generation using timer refer to section 5.3.1 and 5.3.2.

5.4.1 Timer 1 configuration in 10 bit fast PWM mode

Function Timer1_init() function initializes the function in 10 bit fast PWM generation mode.

```
//TIMER1 initialization in 10 bit fast PWM mode
//prescale:256
// WGM: 7) PWM 10bit fast, TOP=0x03FF
// actual value: 52.25Hz
void timer1_init(void)
{
 TCCR1B = 0x00;          //stop
 TCNT1H = 0xFC;          //Counter high value to which OCR1xH value is to be compared with
 TCNT1L = 0x01;          //Counter low value to which OCR1xH value is to be compared with
 OCR1AH = 0x03;          //Output compare Register high value for servo 1
 OCR1AL = 0xFF;          //Output Compare Register low Value For servo 1
 OCR1BH = 0x03;          //Output compare Register high value for servo 2
 OCR1BL = 0xFF;          //Output Compare Register low Value For servo 2
 OCR1CH = 0x03;          //Output compare Register high value for servo 3
 OCR1CL = 0xFF;          //Output Compare Register low Value For servo 3
 ICR1H  = 0x03;
 ICR1L  = 0xFF;
 TCCR1A = 0xAB;
//COM1A1=1, COM1A0=0; COM1B1=1, COM1B0=0; COM1C1=1 COM1C0=0
//For Overriding normal port functionality to OCRnA outputs. WGM11=1, WGM10=1. Along With //WGM12 in
TCCR1B for Selecting FAST PWM Mode TCCR1C = 0x00;
 TCCR1B = 0x0C; //WGM12=1; CS12=1, CS11=0, CS10=0 (Prescaler=256)
}
```

5.4.2 Timer 1 initialization

```
//Function to initialize all the peripherals
void init_devices(void)
{
 cli(); //disable all interrupts
 port_init();
 timer1_init();
 sei(); //re-enable interrupts
}
```

5.4.3 Functions to rotate servo motor by specified angle in the multiple of 1.86 degrees

```
//Function to rotate Servo 1 by a specified angle in the multiples of 1.86 degrees
void servo_1(unsigned char degrees)
{
 float PositionPanServo = 0;
 PositionPanServo = ((float)degrees / 1.86) + 35.0;
 OCR1AH = 0x00;
 OCR1AL = (unsigned char) PositionPanServo;
}


//Function to rotate Servo 2 by a specified angle in the multiples of 1.86 degrees
void servo_2(unsigned char degrees)
{
 float PositionTiltServo = 0;
```

```
 PositionTiltServo = ((float)degrees / 1.86) + 35.0;
 OCR1BH = 0x00;
 OCR1BL = (unsigned char) PositionTiltServo;
}

//Function to rotate Servo 3 by a specified angle in the multiples of 1.86 degrees
void servo_3(unsigned char degrees)
{
 float PositionServo = 0;
 PositionServo = ((float)degrees / 1.86) + 35.0;
 OCR1CH = 0x00;
 OCR1CL = (unsigned char) PositionServo;
}
```

5.4.4 Functions to make servo motor free to rotate

"servo_n_free" functions unlock the servo motors from the any angle and make them free by giving 100% duty cycle at the PWM. This function can be used to reduce the power consumption of the motor if it is holding any load against the gravity.

```
void servo_1_free (void) //makes servo 1 free rotating
{
 OCR1AH = 0x03;
 OCR1AL = 0xFF; //Servo 1 off
}

void servo_2_free (void) //makes servo 2 free rotating
{
 OCR1BH = 0x03;
 OCR1BL = 0xFF; //Servo 2 off
}

void servo_3_free (void) //makes servo 3 free rotating
{
 OCR1CH = 0x03;
 OCR1CL = 0xFF; //Servo 3 off
}
```

5.4.5 Application example for servo motor control

Located in the folder "Experiments \ Servo_Motor_Control_using_PWM" folder in the documentation CD.

This experiment demonstrates Servo motor control using 10 bit fast PWM mode.

Concepts covered: Use of timer to generate PWM for servo motor control

Connection Details:
PORTB 5 (OC1A): Servo 1(Camera pod pan servo)
PORTB 6 (OC1B): Servo 2 (Camera pod tile servo)
PORTB 7 (OC1C): Servo 3 (Reserved)

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

    Microcontroller: ATMEGA2560
    Frequency: 14745600
    Optimization: -O0
    (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. 5V supply to these motors is provided by separate low drop voltage regulator "5V Servo" which can supply maximum of 800mA current. It is a good practice to move one servo at a time to reduce power surge in the robot's supply lines. Also preferably take ADC readings while servo motor is not moving or stopped moving after giving desired position.

6. LCD Interfacing

To interface LCD with the microcontroller in default configuration requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To reduce the number of I/Os required for LCD interfacing we can use 4 bit interfacing mode which requires 3 control signals with 4 data lines. In this mode upper nibble and lower nibble of commands/data set needs to be sent separately. Figure 6.1 shows LCD interfacing in 4 bit mode. The three control lines are referred to as EN, RS, and RW.

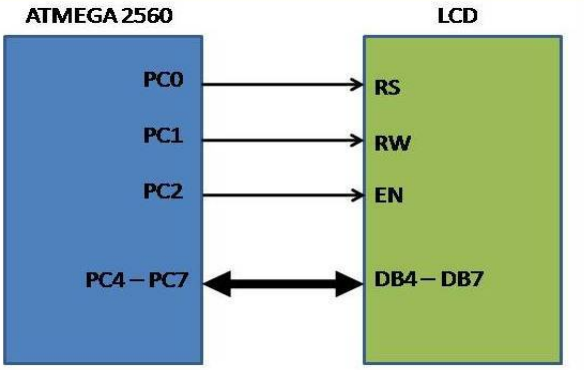


Figure 6.1: LCD interfacing in 4 bit mode

Microcontroller	LCD PINS	Description
VCC	VCC	Supply voltage (5V).
GND	GND	Ground
PC0	RS (Control line)	Register Select
PC1	R/W (Control line)	READ /WRITE
PC2	EN (Control Line)	Enable
PC4 to PC7	D4 to D7 (Data lines)	Bidirectional Data Bus
	LED+, LED-	Back light control

Table 6.1: LCD pin mapping with the microcontroller

The EN line is called "Enable" and it is connected to PC2. This control line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again.

The RS line is the "Register Select" line and it is connected to PC0. When RS is low (0), the data is treated as a command or special instruction by the LCD (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is treated as text data which should be displayed on the screen.

The RW line is the "Read/Write" control line and it is connected to PC1. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading from) the LCD.

The data bus is bidirectional, 4 bit wide and is connected to PC4 to PC7 of the microcontroller. The MSB bit (DB7) of data bus is also used as a Busy flag. When the Busy flag is 1, the LCD is

in internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/W = 1, the Busy flag is output on DB7. The next instruction must be written after ensuring that the busy flag is 0.

We are using LCD in 4-bit mode. In the 4-bit mode the data is sent in nibbles with higher nibble sent first followed by the lower nibble. Initialization of LCD in 4-bit mode is done only after setting the LCD for 4-bit mode. LCD reset sequence include following steps.

1. Wait for about 20ms.
2. Send the first value 0x30.
3. Wait for about 10ms.
4. Send the second value 0x30.
5. Wait for about 1ms.
6. Send the third value 0x30.
7. Wait for about 1ms.
8. Send 0x20 for selecting 4-bit mode.
9. Wait for 1ms.

Before we can display any data on the LCD we need to initialize the LCD for proper operation. The first instruction we send must tell the LCD that we will be communicating with it using 4-bit data bus. Remember that the RS line must be low if we are sending a command to the LCD. In the second and third instruction we clear and reset the display of the LCD. The fourth instruction sets the display and cursor ON. In fifth instruction we place the cursor at the start. Check the lcd_init( ) function to see how all this is put in code.

The function lcd_reset() and lcd_init completes the initialization of LCD in 4-bit mode. Now following steps are followed to send the command/data in 4-bit mode.

1. Mask lower 4-bits.
2. Send command/data to the LCD port.
3. Send enable signal to EN pin.
4. Mask higher 4-bits.
5. Shift bits left by 4 positions (to bring lower bits to upper bits position).
6. Send command/data to the LCD port.
7. Send enable signal to EN pin.

The function lcd_wr_command() and lcd_wr_char() are for sending the command and data respectively to the LCD.

For using the busy flag (polling method) the LCD is read in the similar way, i.e. nibble by nibble, here we are not using the polling method and instead we are providing the necessary delay between the commands.

After the initialization of LCD in 4-bit mode is complete, then for sending the data in nibbles there is no need of providing any delay between two nibbles of same byte, the most significant

nibble (higher 4-bits) is sent first, immediately followed by the least significant nibble (lower 4-bits).

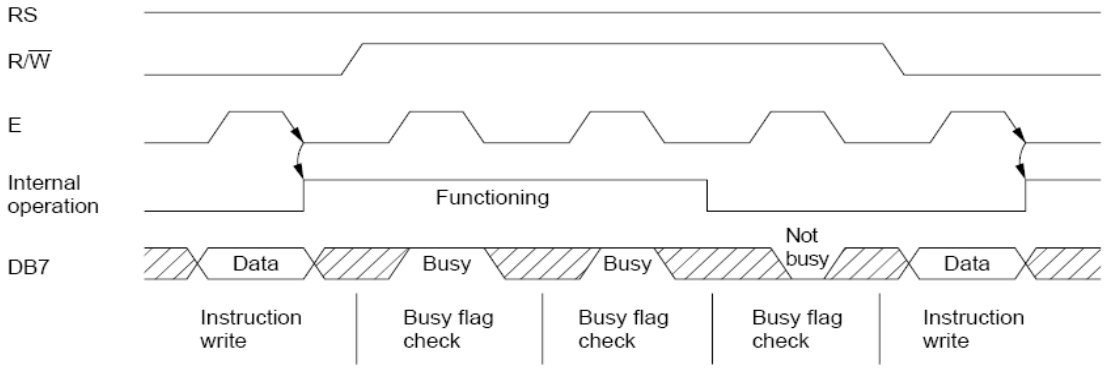


Figure 6.2: LCD interface timing diagram

For more details on the LCD, refer to "hd44780u.pdf" in the folder "datasheet" in the documentation CD.

6.1 Functions used for the LCD display

Note: All the functions are defined in the lcd.c file. It is located inside the "Experiments" folder inside the documentation CD.

6.1.1 LCD port configure (called inside the "port_init()" function)

```
void lcd_port_config (void)
{
 DDRC = DDRC | 0xF7; //all the LCD pin's direction set as output
 PORTC = PORTC & 0x80; // all the LCD pins are set to logic 0 except PORTC 7
}
```

6.1.2 Setting LCD in 4 bit mode

```
void lcd_set_4bit()
{
        _delay_ms(1);

        cbit(lcd_port,RS);      //RS=0 --- Command Input
        cbit(lcd_port,RW);      //RW=0 --- Writing to LCD
        lcd_port = 0x30;        //Sending 3 in the upper nibble
        sbit(lcd_port,EN);      //Set Enable Pin
        _delay_ms(5);           //delay
        cbit(lcd_port,EN);      //Clear Enable Pin

        _delay_ms(1);

        cbit(lcd_port,RS);      //RS=0 --- Command Input
        cbit(lcd_port,RW);      //RW=0 --- Writing to LCD
        lcd_port = 0x30;        //Sending 3 in the upper nibble
        sbit(lcd_port,EN);      //Set Enable Pin
        _delay_ms(5);           //delay
        cbit(lcd_port,EN);      //Clear Enable Pin

        _delay_ms(1);
```

```
        cbit(lcd_port,RS);          //RS=0 --- Command Input
        cbit(lcd_port,RW);          //RW=0 --- Writing to LCD
        lcd_port = 0x30;            //Sending 3 in the upper nibble
        sbit(lcd_port,EN);          //Set Enable Pin
        _delay_ms(5);               //delay
        cbit(lcd_port,EN);          //Clear Enable Pin

        _delay_ms(1);

        cbit(lcd_port,RS);          //RS=0 --- Command Input
        cbit(lcd_port,RW);          //RW=0 --- Writing to LCD
        lcd_port = 0x20;            //Sending 2 in the upper nibble to initialize LCD 4-bit mode
        sbit(lcd_port,EN);          //Set Enable Pin
        _delay_ms(5);               //delay
        cbit(lcd_port,EN);          //Clear Enable Pin
}
```

6.1.3 LCD initialization function

```
//Function to Initialize LCD
void lcd_init()
{
        _delay_ms(1);
        lcd_wr_command(0x28);  //4-bit mode and 5x8 dot character font
        lcd_wr_command(0x01);  //Clear LCD display
        lcd_wr_command(0x06);  //Auto increment cursor position
        lcd_wr_command(0x0E);  //Turn on LCD and cursor
        lcd_wr_command(0x80);  //Set cursor position
}
```

6.1.4 Function to write command on LCD

```
/Function to write command on LCD
void lcd_wr_command(unsigned char cmd)
{
        unsigned char temp;
        temp = cmd;
        temp = temp & 0xF0;
        lcd_port &= 0x0F;
        lcd_port |= temp;
        cbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
        _delay_ms(5);

        cbit(lcd_port,EN);

        cmd = cmd & 0x0F;
        cmd = cmd<<4;
        lcd_port &= 0x0F;
        lcd_port |= cmd;
        cbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
```

```
        _delay_ms(5);
        cbit(lcd_port,EN);
}
```

6.1.5 Function for LCD home

```
void lcd_home()
{
        lcd_wr_command(0x80);
}
```

6.1.6 Function to Print String on LCD

```
void lcd_string(char *str)
{
        while(*str != '\0')
        {
                lcd_wr_char(*str);
                str++;
        }
}
```

6.1.7 Position the LCD cursor at "row", "column"

```
//Position the LCD cursor at "row", "column"
void lcd_cursor (char row, char column)
{
        switch (row) {
                case 1: lcd_wr_command (0x80 + column - 1); break;
                case 2: lcd_wr_command (0xc0 + column - 1); break;
                case 3: lcd_wr_command (0x94 + column - 1); break;
                case 4: lcd_wr_command (0xd4 + column - 1); break;
                default: break;
        }
}
```

6.1.8 Function to print any input value up to the desired digit on LCD

```
/ Function to print any input value up to the desired digit on LCD
void lcd_print (char row, char column, unsigned int value, int digits)
{
        unsigned char flag=0;
        if(row==0||coloumn==0)
        {
                lcd_home();
        }
        else
        {
                lcd_cursor (row, column);
        }
        if(digits==5 || flag==1)
        {
                million=value/10000+48;
                lcd_wr_char(million);
                flag=1;
        }
        if(digits==4 || flag==1)
```

```
        {
                temp = value/1000;
                thousand = temp%10 + 48;
                lcd_wr_char(thousand);
                flag=1;
        }
        if(digits==3 || flag==1)
        {
                temp = value/100;
                hundred = temp%10 + 48;
                lcd_wr_char(hundred);
                flag=1;
        }
        if(digits==2 || flag==1)
        {
                temp = value/10;
                tens = temp%10 + 48;
                lcd_wr_char(tens);
                flag=1;
        }
        if(digits==1 || flag==1)
        {
                unit = value%10 + 48;
                lcd_wr_char(unit);
        }
        if(digits>5)
        {
                lcd_wr_char('E');
        }
}
```

6.2 Application examples

6.2.1 Application example to print string on the LCD

Located in the folder “Experiments \ LCD_interfcing” folder in the documentation CD.

This program shows how to write string on the LCD

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

   Microcontroller: ATMEGA2560
   Frequency: 14745600
   Optimization: -O0
   (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. Buzzer is connected to PC3. Hence to operate buzzer without interfering with the LCD, buzzer should be turned on or off only using buzzer function

6.2.2 Application example to print sensor data on the LCD

It also involves concept of ADC. It will be covered in chapter 7.

7. Analog to Digital Conversion

Fire Bird V has three white line sensors, one Sharp IR range sensor with four add-on sockets for additional Sharp IR range sensors, eight Analog IR proximity sensors. Robot can also measure its own battery voltage and current. All these sensors give analog output. We need to use ATMEGA2560 microcontroller's ADC (Analog to Digital Converter) to convert these analog values in to digital values.

The ATMEGA2560 features a 10-bit successive approximation Analog to Digital Converter (ADC). The ADC block is connected to an 16-channel Analog Multiplexer which allows 16 single-ended voltage inputs from the pins of PORTF and PORTK. The minimum value represents GND and the maximum value represents the voltage on the AREF pin (5 Volt in the case of Fire Bird V).

7.1 ADC Resolution

The resolution of the ADC indicates the number of discrete values it can produce over the range of analog values. The values are usually stored electronically in binary form, so the resolution is usually expressed in bits. In consequence, the number of discrete values available, or "levels", is usually a power of two. For example, an ADC with a resolution of 8 bits can encode an analog input to one in 256 different levels, since $2^8 = 256$. The values can represent the ranges from 0 to 255 (i.e. unsigned integer) or from -128 to 127 (i.e. signed integer), depending on the application.

ATMEGA2560 microcontroller has ADC with 10 bit resolution.

$V_{resolution} = V_{full\ scale} / 2^n - 1$
Where $V_{full\ scale}$ = 5V; n = 10 or 8

Case 1: n = 10 (10 bit resolution)
$V_{resolution} = 5V / 2^{10} - 1$
$V_{resolution} = 4.8875mV$

Case 2: n = 8 (8 bit resolution)
$V_{resolution} = 5V / 2^8 - 1$
$V_{resolution} = 19.6078mV$

7.2 Registers for ADC

7.2.1 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	ADEN	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ADEN: ADC Enable
Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning off the ADC while a conversion is in progress, will terminate this conversion.

Bit 6 – ADSC: ADC Start Conversion
In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC. ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Bit 5 – ADATE: ADC Auto Trigger Enable
When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

Bit 4 – ADIF: ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

Bit 3 – ADIE: ADC Interrupt Enable
When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Table 7.1 ADC prescaler selections

7.2.2 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0
	-	ACME	-	-	MUX5	ADTS2	ADTS1	ADTS0
Read / Write	R	R/W	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 3 – MUX5: Analog Channel and Gain Selection Bit
This bit is used together with MUX4:0 in ADMUX to select which combination in of analog inputs are connected to the ADC. If this bit is changed during a conversion, the change will not go in effect until this conversion is complete. For more details refer to table 26.4 in the ATMEGA2560 datasheet.

7.2.3 ADMUX– ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7:6 – REFS1:0: Reference Selection Bits
As shown in Table 7.2, these bits select the voltage reference for the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal VREF turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Voltage Reference with external capacitor at AREF pin
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Table 7.2: Voltage reference selection for ADC

Note:
If 10x or 200x gain is selected, only 2.56 V should be used as Internal Voltage Reference. For differential conversion, only 1.1V cannot be used as internal voltage reference.

Bit 5 – ADLAR: ADC Left Adjust Result
The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “ADCL and ADCH – The ADC Data Register” in the section 7.2.5.

Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits
The value of these bits selects which combination of analog inputs are connected to the ADC. For more details see Table 7.3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set)

MUX5:0	ADC pin	Pin function	Pin status
000000	PF0/ADC0	ADC input for battery voltage monitoring	Input (Floating)
000001	PF1/ADC1	ADC input for white line sensor 3	Input (Floating)
000010	PF2/ADC2	ADC input for white line sensor 2	Input (Floating)
000011	PF3/ADC3	ADC input for white line sensor 1	Input (Floating)
000100	PF4/ADC4/TCK	ADC input for IR proximity analog sensor 1*****	Input (Floating)
000101	PF5(ADC5/TMS)	ADC input for IR proximity analog sensor 2*****	Input (Floating)
000110	PF6/(ADC6/TD0)	ADC input for IR proximity analog sensor 3*****	Input (Floating)
000111	PF7(ADC7/TDI)	ADC input for IR proximity analog sensor 4*****	Input (Floating)
100000	PK0/ADC8/PCINT16	ADC input for IR proximity analog sensor 5	Input (Floating)
100001	PK1/ADC9/PCINT17	ADC input for Sharp IR range sensor 1	Input (Floating)
100010	PK2/ADC10/PCINT18	ADC input for Sharp IR range sensor 2	Input (Floating)
100011	PK3/ADC11/PCINT19	ADC input for Sharp IR range sensor 3	Input (Floating)
100100	PK4/ADC12/PCINT20	ADC input for Sharp IR range sensor 4	Input (Floating)
100101	PK5/ADC13/PCINT21	ADC input for Sharp IR range sensor 5	Input (Floating)
100110	PK6/ADC14/PCINT22	ADC Input For Servo Pod 1	Input (Floating)
100111	PK7/ADC15/PCINT23	ADC Input For Servo Pod 2	Input (Floating)

Table 7.3 Input channel selection and functions

***** For using Analog IR proximity (1, 2, 3 and 4) sensors short the jumper J2. To use JTAG via expansion slot of the microcontroller socket remove these jumpers.

Note:
Table 7.3 is a simplified version of the table 26.4 from the ATMEGA2560 datasheet customized to the Fire Bird V ATMEGA2560 robot
MUX4:1 are located inside ADMUX register. MUX5 is located in the ADCSRB register.

7.2.4 ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0
	ACD	**ACBG**	**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**
Read / Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ACD: Analog Comparator Disable
When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

Bit 6 – ACBG: Analog Comparator Band gap Select
When this bit is set, a fixed band gap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. When the band gap reference is used as input to the Analog Comparator, it will take a certain time for the voltage to stabilize. If not stabilized, the first conversion may give a wrong value. For more information see “Internal Voltage Reference” on page 62 of the ATMEGA2560 datasheet.

Bit 5 – ACO: Analog Comparator Output
The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

Bit 4 – ACI: Analog Comparator Interrupt Flag
This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

Bit 3 – ACIE: Analog Comparator Interrupt Enable
When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

Bit 2 – ACIC: Analog Comparator Input Capture Enable
When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the input capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the input capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select
These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 7.4.

ACIS1	ACIS0	Interrupt mode
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Rising Output Edge

Table 7.4: ACIS1/ACIS0 settings

7.2.5 ADCL and ADCH – The ADC Data Register

Case 1: ADLAR = 0;

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R	R	R	R
Bit	15	14	13	12	11	10	9	8
ADCH							**ADC9**	**ADC8**
ADCL	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**
Bit	7	6	5	4	3	2	1	0
Read / Write	R	R	R	R	R	R	R	R
Initial value	0	0	0	0	0	0	0	0

Case 2: ADLAR = 1; (Left adjust)

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R	R	R	R
Bit	15	14	13	12	11	10	9	8
ADCH	**ADC9**	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**
ADCL	**ADC1**	**ADC0**						
Bit	7	6	5	4	3	2	1	0
Read / Write	R	R	R	R	R	R	R	R
Initial value	0	0	0	0	0	0	0	0

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form. When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision (7 bit + sign bit for differential input channels) is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH. The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

7.3 Functions for ADC

7.3.1 Function to configure pins for ADC (called inside the "port_init()" function)

```
//ADC pin configuration
void adc_pin_config (void)
{
 DDRF = 0x00; //set PORTF direction as input
 PORTF = 0x00; //set PORTF pins floating
 DDRK = 0x00; //set PORTK direction as input
 PORTK = 0x00; //set PORTK pins floating
}
```

7.3.2 Function to configure ADC

```
//Function to Initialize ADC
void adc_init()
{
        ADCSRA = 0x00;
        ADCSRB = 0x00;          //MUX5 = 0
        ADMUX = 0x20;           //Vref=5V external --- ADLAR=1 --- MUX4:0 = 0000
        ACSR = 0x80;
        ADCSRA = 0x86;          //ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}
```

7.3.3 Function to initialize ADC

```
void init_devices (void)
{
 cli(); //Clears the global interrupts
 port_init();
 adc_init();
 sei(); //Enables the global interrupts
}
```

7.3.4 Function to get ADC value

```
//This Function accepts the Channel Number and returns the corresponding Analog Value
unsigned char ADC_Conversion(unsigned char Ch)
{
	unsigned char a;
	if(Ch>7)
	{
		ADCSRB = 0x08;
	}
	Ch = Ch & 0x07;
	ADMUX= 0x20| Ch;
	ADCSRA = ADCSRA | 0x40;		//Set start conversion bit
	while((ADCSRA&0x10)==0);	//Wait for ADC conversion to complete
	a=ADCH;
	ADCSRA = ADCSRA|0x10;		//clear ADIF (ADC Interrupt Flag) by writing 1 to it
	ADCSRB = 0x00;
	return a;
}
```

7.4 Application examples

7.4.1 Application example to display ADC sensor data on the LCD

Located in the folder "Experiments \ ADC_Sensor_Display_on_LCD" folder in the documentation CD.

7.4.2 Application example to follow white line

Located in the folder "Experiments \ White_Line_Following" folder in the documentation CD.

7.4.3 Application example to perform Adaptive Cruise Control (ACC) while following the white line

Located in the folder "Experiments \ Adaptive_Cruise_Control" folder in the documentation CD.

Note for all the application examples:

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

   Microcontroller: ATMEGA2560
   Frequency: 14745600
   Optimization: -O0
   (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. Make sure that you copy the lcd.c file in your folder

3. Distance calculation is for Sharp GP2D12 (10cm-80cm) IR Range sensor

8. Serial Communication

Serial Communication using UART

The Fire Bird V can communicate with other robots / devices serially using either wired link or wireless module. Serial communication is done in asynchronous mode. In the asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization.

ATMEGA2560 have four USART (0 to 3) ports available for serial communication.

1. RS232 Serial Communication on UART1.
2. USB Communication using FT232 USB to serial converter on UART2.
3. ZigBee Wireless Communication on UART0 (if XBee wireless module is installed).
4. TTL level serial communication pins on the expansion port on the microcontroller adaptor board by UART3.

8.1 Registers involved in the serial communication

Note: in the following registers 'n' represents the UART number which can be 0, 1, 2 or 3.

8.1.1 UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	RXCn	**TXCn**	**UDREn**	**FEn**	**DORn**	**UPEn**	**U2Xn**	**MPCMn**
Read / Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – RXCn: USART Receive Complete
This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Complete interrupt (see description of the RXCIEn bit in the section 8.1.2).

Bit 6 – TXCn: USART Transmit Complete
This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag can generate a Transmit Complete interrupt (see description of the TXCIEn bit in the section 8.1.2 in the ATMEGA2560 datasheet).

Bit 5 – UDREn: USART Data Register Empty
The UDREn Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn Flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit). UDREn is set after a reset to indicate that the Transmitter is ready.

Bit 4 – FEn: Frame Error
This bit is set if the next character in the receive buffer had a Frame Error when received. I.E. when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

Bit 3 – DORn: Data OverRun
This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

Bit 2 – UPEn: USART Parity Error
This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

Bit 1 – U2Xn: Double the USART Transmission Speed
This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation. Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

Bit 0 – MPCMn: Multi-processor Communication Mode
This bit enables the Multi-processor Communication mode. When the MPCMn bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCMn setting. For more detailed information see “Multi-processor Communication Mode” on page 222 of the ATMEGA2560 datasheet.

8.1.2 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0
	RXCIEn	**TXCIEn**	**UDRIEn**	**RXENn**	**TXENn**	**UCSZn2**	**RXB8n**	**TXB8n**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – RXCIEn: RX Complete Interrupt Enable n
Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

Bit 6 – TXCIEn: TX Complete Interrupt Enable n
Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable n
Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

Bit 4 – RXENn: Receiver Enable n
Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

Bit 3 – TXENn: Transmitter Enable n
Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

Bit 2 – UCSZn2: Character Size n
The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRnC sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Bit 1 – RXB8n: Receive Data Bit 8 n
RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

Bit 0 – TXB8n: Transmit Data Bit 8 n
TXB8n is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDRn.

8.1.3 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0
	UMSELn1	**UMSELn0**	**UPMn1**	**UPMn0**	**USBSn**	**UCSZn1**	**UCSZn0**	**UCP0Ln**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bits 7:6 – UMSELn1:0 USART Mode Select
These bits select the mode of operation of the USARTn as shown in Table 8.1.

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Table 8.1: UMSELn Bit settings

Note: See "USART in SPI Mode" on page 232 of the ATMEGA2560 datasheet for full description of the Master SPI Mode (MSPIM) operation

Bits 5:4 – UPMn1:0: Parity Mode
These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

UPMn1	UPMn0	Parity mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Table 8.2: UPMn Bits settings

Bit 3 – USBSn: Stop Bit Select
This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Table 8.3: USBSbit settings

Bit 2:1 – UCSZn1:0: Character Size
The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

UCSZn2	UCSZn1	UCSZn0	Character size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Table 8.4: UCSZn Bits Settings

Bit 0 – UCPOLn: Clock Polarity
This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCKn).

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

Table 8.5: UCPOLn Bit Settings

8.1.4 UBRRnL and UBRRnH – USART Baud Rate Registers

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R/W	R/W	R/W	R/W
Bit	15	14	13	12	11	10	9	8
ADCH	-	-	-	-	**UBRR11**	**UBRR10**	**UBRR9**	**UBRR8**
ADCL	**UBRR7**	**UBRR6**	**UBRR5**	**UBRR4**	**UBRR3**	**UBRR2**	**UBRR1**	**UBRR0**
Bit	7	6	5	4	3	2	1	0
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

Baud rate calculation:
Crystal frequency: 14.7456 MHz
Required baud rate: 9600 bits per second

UBRR = (System Clock / (16 * baud rate)) – 1
= (14.7456 MHz / (16 * 9600)) – 1
= 95
= 0x5F (hex)
UBRRH = 0x00
UBRRL = 0x5F

Baud rate	2400	4800	9600	14.4k	19.2k	28.8k	38.4k	57.6k	76.8k	115.2k

Table 8.6: Value of UBRR for different baud rate for 14.7456 MHz crystal

For 14.7456MHz crystal frequency, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings as shown in the table 8.6.

Note:
While loading values in the UBRR register load values in the UBRRH resistor first and then in UBRRL register.

8.1.5 UDRn – USART I/O Data Register n

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDRn. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDREn Flag in the UCSRnA Register is set. Data written to UDRn when the UDREn Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxDn pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

8.2 Functions used in serial communication

Note:

In all the functions below function will remain same for other UARTs. Only ‘1’ of UART1 will be replaced with the appropriate UART number

8.2.1 Function to configure UART1

```
//Function To Initialize UART1
// desired baud rate:9600
// actual baud rate:9600 (error 0.0%)
// char size: 8 bit
// parity: Disabled
void uart1_init(void)
{
 UCSR1B = 0x00; //disable while setting baud rate
 UCSR1A = 0x00;
 UCSR1C = 0x06;
 UBRR1L = 0x5F; //set baud rate lo
 UBRR1H = 0x00; //set baud rate hi
 UCSR1B = 0x98;
}
```

8.2.2 Function to initialize uart 1

```
void init_devices()
{
 cli(); //Clears the global interrupts
 port_init();  //Initializes all the ports
 uart1_init(); //Initialize UART1 for serial communication
 sei();   //Enables the global interrupts
}
```

8.2.3 Receive complete ISR

When UART receives eight data bits on receive pin of the microcontroller, RXC flag is set. If RXCIE interrupt is enabled then receive complete interrupt triggers ISR. This ISR then reads valid data from UDR1 and stores it in a separate variable before next character is received and overwritten. It is always recommended to save data read from UDR1 in a separate variable as next character received will overwrite and destroy the existing data in UDR1.

```
SIGNAL(SIG_USART1_RECV)    // ISR for receive complete interrupt
{
        data = UDR1;      //making copy of data from UDR1 in 'data' variable
        //Insert your coder here
}
```

8.2.4 Data register empty ISR

The transmitter side of the UART is double buffered containing UDRn to hold the data written from the program and transmit register to actually transmit parallel data sequentially bit-by-bit on the transmit pin. The data written to UDRn is transferred to transmit register. At this point, the UDRn is available to accept next data word from the program. This sets UDRE flag and if UDRIE interrupt is enabled then UDRn data register empty interrupt triggers ISR. This ISR then loads next data byte to be transmitted into UDRn.

```
SIGNAL (SIG_USART1_DATA)
{
        UDR1 = tx_data;
         // Insert your code here…….
}
```

8.2.5 Transmit complete ISR

In the case of packet based data communication it is necessary to know when a byte has been completely transmitted out of microcontroller. The TXC flag is provided to indicate that the transmit register is empty and no new data is waiting to be transmitted. If transmit register is empty it sets TXC flag and if TXCIE interrupt is enabled then Transmit complete interrupt triggers ISR. This ISR can be used as a confirmation of the byte that was loaded in UDR1 is successfully transmitted out of the microcontroller transmit pin. This interrupt can be used to check if all the bytes in a packet transmission are transmitted successfully.

```
SIGNAL (SIG_USART1_TRANS)
{
        //Insert your code here……..
}
```

8.3 Application example for serial communication

Note:
All the application examples are identical in nature.
Robot can be controlled using wired or wireless link using PC with these application examples.
Refer to chapter 6 from the Hardware Manual for using these application examples.

8.3.1 RS232 serial communication using UART1

Located in the folder “Experiments \ A_Serial_Communication” folder in the documentation CD.

8.3.2 USB communication using FT232 USB to serial converter using UART2

Located in the folder “Experiments \ B_Serial_Communication_USB-RS232” folder in the documentation CD.

8.3.3 Serial communication over wireless using ZigBee wireless module sing UART0

Located in the folder “Experiments \ C_Serial_Communication_ZigBee_wireless” folder in the documentation CD.

9. SPI Communication

Fire Bird V robot can be interfaced with more than 30 sensors at the same time. ATMEGA2560 does not have sufficient number of ADC available of sensor interfacing. Hence ATMEGA8 microcontroller is connected with ATMEGA2560 microcontroller over the SPI port. Jumper J4 needs to be removed before attempting to do ISP with ATMEGA2560 and ATMEGA8 as there SPI lines are connected with the jumper J4. For more details on the jumpers, refer to the section 3.19.6 in the Hardware Manual.

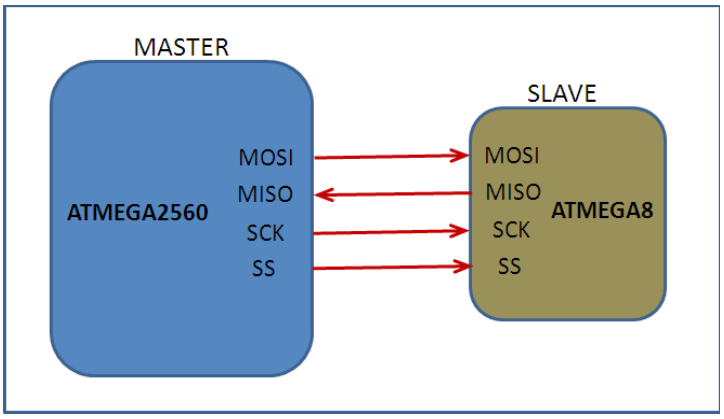


Figure 9.1: SPI Interface Block Diagram

9.1 Concept of SPI communication

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between several peripheral devices or between microcontrollers. ATMEGA2560 and ATMEGA8 both have inbuilt SPI peripheral. SPI pins consist of MOSI (Master Output Slave Input), MISO (Master Input Slave Output), SCK (Serial Clock) and SS* (Slave Select).

Features of the SPI communication:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

SPI communication process:

The basic operation of SPI involves the Master initiating the communication.

- Master sets the **SS** (Slave Select) pin low to tell the slave that communication is about to start.
- The master writes a byte onto **MOSI** (Master Output Slave Input) pin and the slave does the same on the MISO (Master Input Slave Output) pin.

- As the master ticks the clock pin SCK it will read the value of MISO pin and slave will read the value of MOSI pin.

Note:
For master any pin can act as slave select while for slave, pin SS must be used as Slave Select.

The communication through SPI interface is a simultaneous transmission and reception through Shift Registers. The data is exchanged bit by bit serially between the master and slave shift registers.

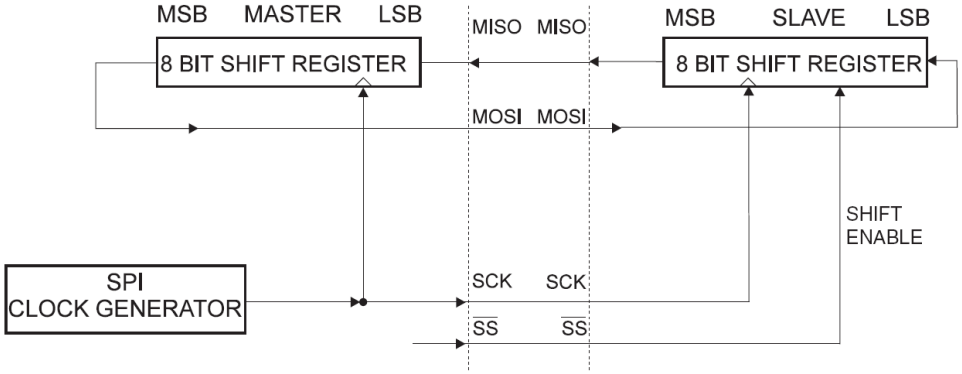


Figure 9.2: SPI MASTER-SLAVE Interconnection (Ref: ATMEGA2560 datasheet)

For example SPI master shift register have 0x8E and SPI slave shift register have 0x32 initially, now as the master initiates the communication by setting the SS pin high. One bit from either of the shift registers is transferred to other on each clock tick at the SCK pin. Figure 9.3 explains this process.

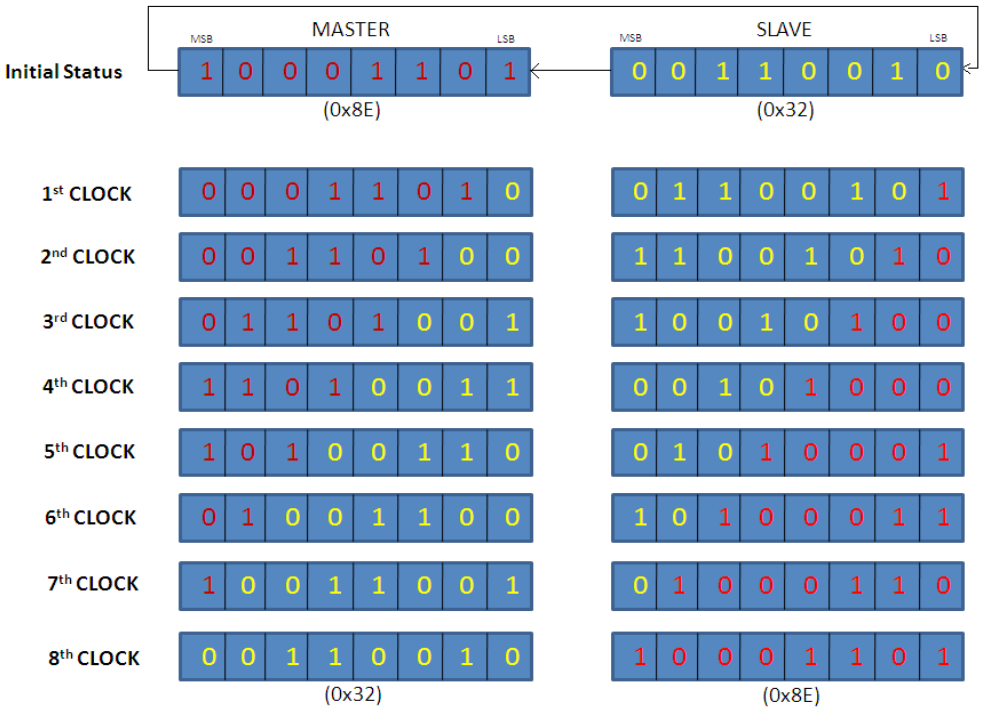


Figure 9.3: Byte transfer between master and slave device in SPI communication

Thus one byte is completely transmitted after eight clock cycles and data byte of slave and master are exchanged simultaneously. In order to receive a byte from the slave microcontroller, master microcontroller has to send a byte to the slave microcontroller.

9.2 Registers involved in the SPI communication

9.2.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0
	SPIE	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – SPIE: SPI Interrupt Enable
This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and if the Global Interrupt Enable bit in SREG is set.

Bit 6 – SPE: SPI Enable
When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

Bit 5 – DORD: Data Order
When the DORD bit is written to one, the LSB of the data word is transmitted first. When the DORD bit is written to zero, the MSB of the data word is transmitted first.

Bit 4 – MSTR: Master/Slave Select
This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If SS is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

Bit 3 – CPOL: Clock Polarity
When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to figure 9.4 and figure 9.5 for an example. The CPOL functionality is summarized in the table 9.1.

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

Table 9.1: CPOL functionality

Bit 2 – CPHA: Clock Phase
The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to figure 9.4 and figure 9.5 for an example. The CPOL functionality is summarized in the table 9.2.

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

Table 9.2: CPHA functionality

Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in the table 9.3.

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$F_{osc}/4$
0	0	1	$F_{osc}/16$
0	1	0	$F_{osc}/64$
0	1	1	$F_{osc}/128$
1	0	0	$F_{osc}/2$
1	0	1	$F_{osc}/8$
1	1	0	$F_{osc}/32$
1	1	1	$F_{osc}/64$

Table 9.3: Relationship between SCK and the oscillator frequency

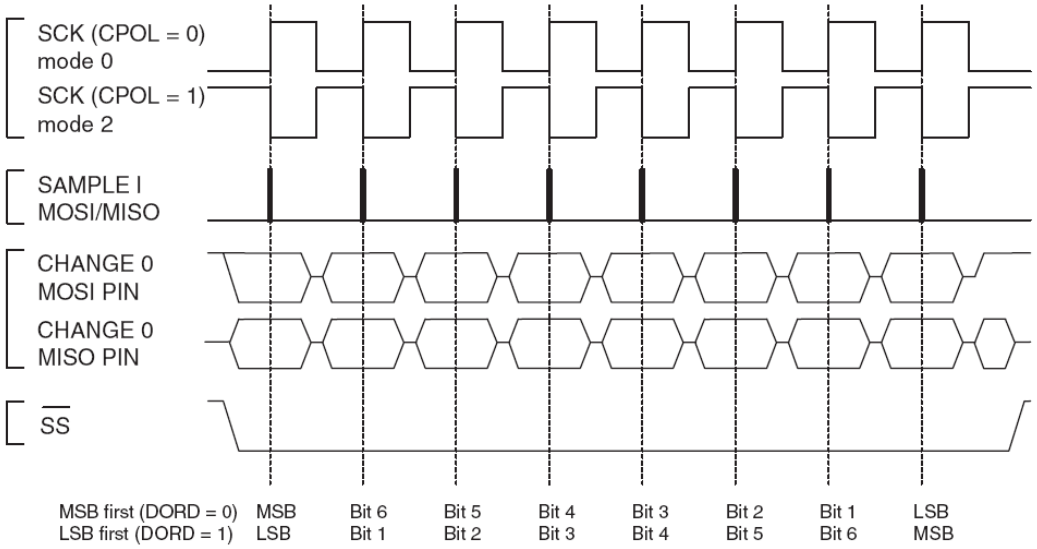


Figure 9.4: SPI transfer format with CPHA = 0

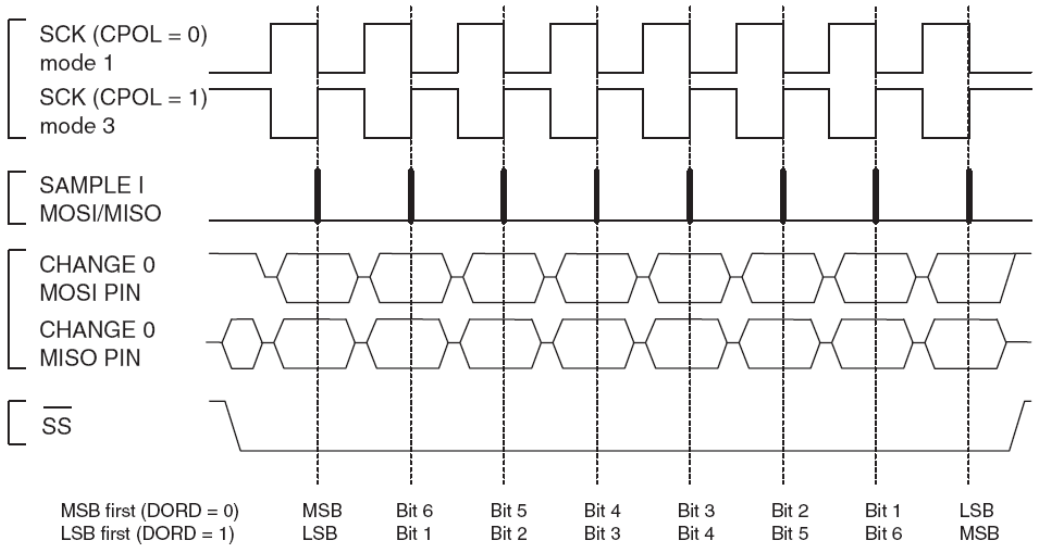


Figure 9.5: SPI transfer format with CPHA = 1

9.2.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0
	SPIF	**WCOL**	-	-	-	-	-	**SPI2X**
Read / Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – SPIF: SPI Interrupt Flag
When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If SS is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

Bit 6 – WCOL: Write Collision Flag
The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

Bit 5:1 – Res: Reserved Bits
These bits are reserved bits and will always read as zero.

Bit 0 – SPI2X: Double SPI Speed Bit
When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see table 9.5). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower. The SPI interface on the ATMEGA2560 is also used for program memory and EEPROM downloading or uploading.

9.2.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0
	MSB							**LSB**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	X	X	X	X	X	X	X	X

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

9.3 Functions for SPI communication (Master)

9.3.1 SPI Master pin configuration (called inside the “port_init()” function)

```
void spi_pin_config (void)
{
 DDRB = DDRB | 0x07;
 PORTB = PORTB | 0x07;
}
```

9.3.2 Function to configure SPI port for communication in master mode

```
//Function To Initialize SPI bus
// clock rate: 86400hz
void spi_init(void)
{
 SPCR = 0x53; //setup SPI
 SPSR = 0x00; //setup SPI
 SPDR = 0x00;
}
```

9.3.4 Function to initialize SPI port

```
void init_devices(void)
{
 cli(); //disable all interrupts
 port_init();
 spi_init();
 sei(); //re-enable interrupts
}
```

9.3.5 Function to send a byte from the master and receive a byte from the slave device

```
//Function to send byte to the slave microcontroller and get ADC channel data from the slave microcontroller
unsigned char spi_master_tx_and_rx (unsigned char data)
{
 unsigned char rx_data = 0;

 PORTB = PORTB & 0xFE; // make SS pin low
 SPDR = data;
 while(!(SPSR & (1<<SPIF))); //wait for data transmission to complete

 _delay_ms(1); //time for ADC conversion in the slave microcontroller

 SPDR = 0x50; // send dummy byte to read back data from the slave microcontroller
 while(!(SPSR & (1<<SPIF))); //wait for data reception to complete
 rx_data = SPDR;
 PORTB = PORTB | 0x01; // make SS high
 return rx_data;
}
```

9.4 Application examples

9.4.1 Application example: SPI master (ATMEGA2560)

Located in the folder “Experiments \ A_SPI_Master” folder in the documentation CD.

This program is for ATMEGA2560 (master) microcontroller. This program demonstrates SPI communication between master (ATMEGA2560) and slave (ATMEGA8) microcontroller. LCD displays analog values of the IR Proximity sensors 6, 7, 8 which are obtained from the ATMEGA8 (slave) microcontroller.

Notes:

1. Setting for ATMEGA2560 (master) microcontroller

   Make sure that in the configuration options following settings are done for proper operation of the code
   Microcontroller: ATMEGA2560
   Frequency: 14745600
   Optimization: -O0
   (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2. ATMEGA2560 (master) and ATMEGA8 (slave) microcontrollers use SPI bus for ISP as well as for the communication between them. Before doing ISP we need to disconnect the SPI bus between these two microcontrollers. Remove tree jumpers marked by J4 on the ATMEGA2560 microcontroller adaptor board before doing ISP.

3. Connect 3 jumpers marked by J4 to connect SPI bus between the microcontrollers.

4. Do not pass value more than 7 to the function "spi_master_tx_and_rx" else it will give back random value

9.4.2 Application example: SPI slave (ATMEGA8)

Located in the folder “Experiments \ B_SPI_Slave” folder in the documentation CD.

This program is for ATMEGA8 (slave) microcontroller. This program is the default firmware for the ATMEGA8 (slave) microcontroller. This program demonstrates SPI communication between master (ATMEGA2560) and slave (ATMEGA8) microcontroller. LCD displays analog values of the IR Proximity sensors 6, 7, 8.

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

   Microcontroller: atmega8
   Frequency: 8000000
   Optimization: -O0
   (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)