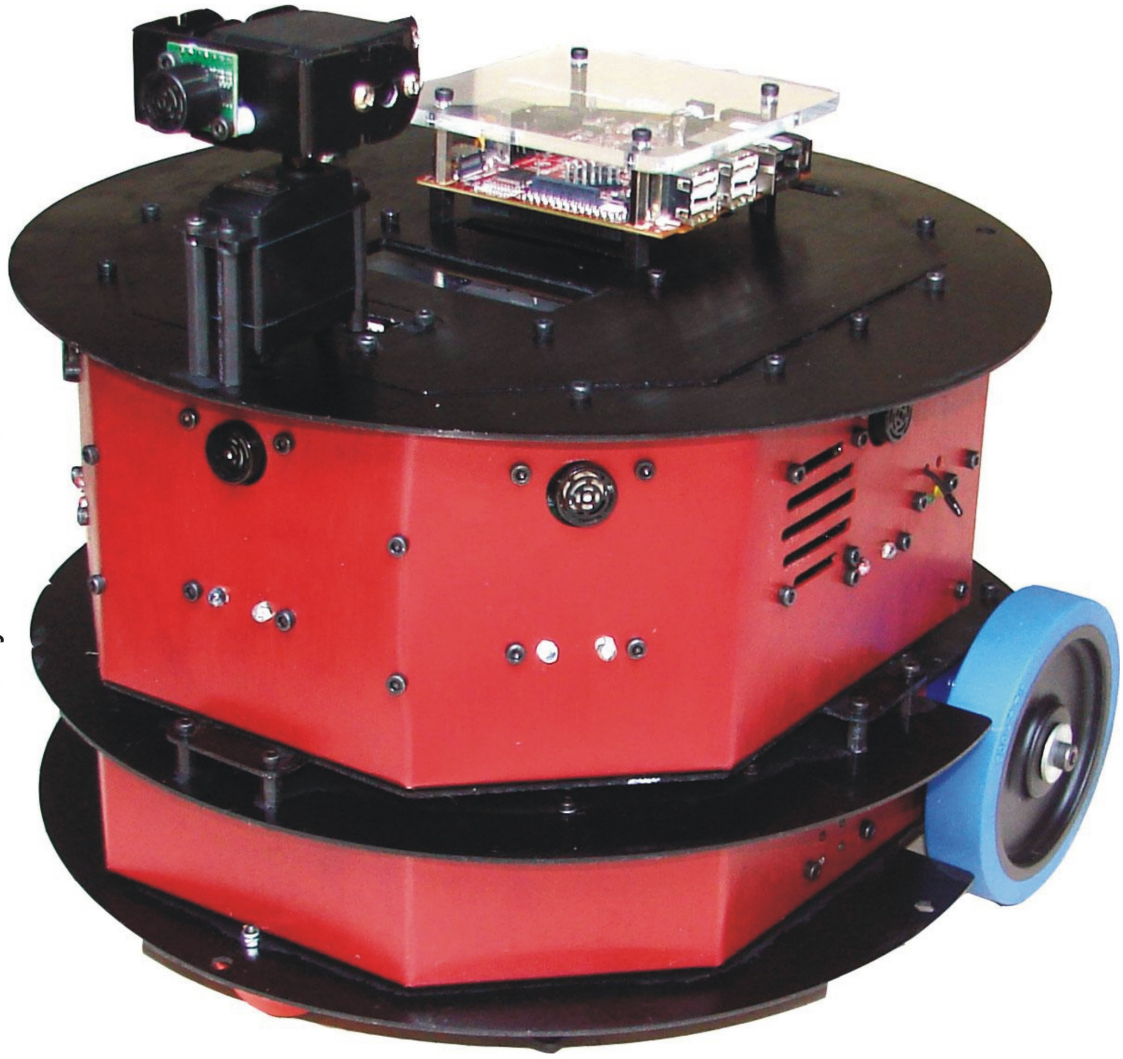
FIRE BIRD VI

LPC1769 ARM M3 CORTEX
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 VI

SOFTWARE MANUAL

Version 7
March 16, 2013

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

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 Robot's 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:

User must go through hardware and software manuals before using robot.

Table of Contents

1 Introduction

Fire Bird VI is a reliable, versatile and rugged robot for the advanced research in robotics. It's easy to use and intuitive interface gets you started on development very quickly. It's unique architecture allows it to be used in many areas of applications such as Mapping and autonomous navigation, Collaborative robotics, tele-presence and many more. Fire Bird VI supports many optional accessories such as on-board computer for vision processing, laser range finder based navigation, vision based stereo range finders, integrated inertial correction to compensate for slippage, digital compass, GPS/DGPS receiver, support for Manipulators and Grippers.

Safety precautions:

- Robot's electronics is static sensitive. Use robot in static free environment.
- Do not access any part of the robot unless robot is in the antistatic environment and user is wearing antistatic strap.
- Read the assembling and operating instructions before working with the robot.
- If robot's battery low buzzer starts beeping, immediately charge the batteries.
- To prevent fire hazard, do not expose the equipment to rain or moisture.
- Refrain from dismantling the unit or any of its accessories once robot is assembled.
- Charge the Lithium Polymer battery only with the charger provided with the robot.
- Charge the Lithium Polymer battery in the open area and on the concrete or ceramic flooring.
- Never allow Lithium Polymer battery to deep discharge. It its deep discharged, charger will refuse to charge the battery because of safety concerns.
- Mount all the components with correct polarity.
- Keep wheels away from long hair or fur.
- Keep your hands away from the wheels. Do not wear loose clothes while operating the robot. Loose cloth may get entangled in robot's wheels and can cause serious injury.
- Keep the robot away from the wet areas. Contact with water will damage the robot.
- To avoid risks of fall, keep your robot in a stable position.
- Do not attach any connectors while robot is powered ON.
- Never leave the robot powered ON when it is not in use.
- Before operating the robot, make sure that you have access to at least "Class A/B" type fire extinguisher.
- Read carefully paragraphs marked with ⚠ caution symbol.

Inappropriate Operation:

Inappropriate operation can damage your robot. Inappropriate operation includes, but is not limited to:

- Dropping the robot, running it off an edge, or otherwise operating it in an irresponsible manner.
- Interfacing new hardware without considering compatibility
- Overloading the robot above its payload capacity.
- Exposing the robot to wet environments.
- Continuing to run the robot after hair, yarn, string, or any other item has become entangled in the robot's axles or wheels.
- All other forms of inappropriate operation.
- Using robot in areas prone to static electricity.

2 Fire Bird VI Robotic Research Platform

Fire Bird VI is reliable, versatile and rugged robot for the advanced research in robotics. It's a fully programmable robot with onboard PC. Fire Bird VI is designed in collaboration with IIT Bombay and manufactured by NEX Robotics. Fire Bird VI robot comes fully assembled and ready to use. It has high quality gear motors with the resolution 360 ticks per revolution position encoder. Motors are driven by smart motion controller with velocity and acceleration control. By selecting correct locomotion drive, robot can be used in indoor and outdoor environment. Robot has 8 ultrasonic range sensors with range of 6 meters covering robot from all sides, 8 IR proximity sensors and 8 line sensors, etc. Robot is powered by high discharge, 11.1V, 5000mAH Lithium Polymer battery packs.

2.1 Block Diagram

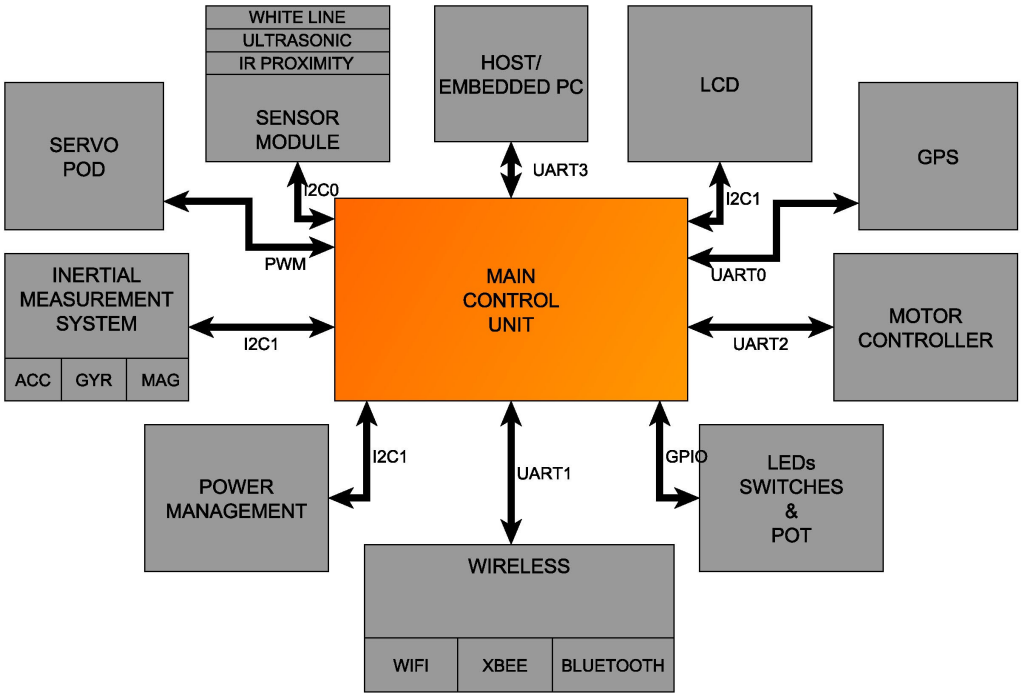


Figure 2.1: Block Diagram of Fire Bird VI

Fire Bird VI uses ARM cortex-M3 based LPC1769 microcontroller running at 120 MHz as main microcontroller. Almost all the peripherals of the robot have their own microcontroller for doing mundane but processor intensive tasks like sensor data acquisition, closed loop motion control etc. These peripherals are interfaced with the main microcontroller LPC1769 ARM cortex-M3 over UART, I2C and SPI bus. Because of this architecture, main microcontroller is free to execute complex algorithms while slave peripherals do most of the sensing and motion control tasks. Since all the peripherals are interfaced over UART, I2C and SPI bus, you can integrate your own modules seamlessly with the robot.

Following sub-sections briefly describe each module of the Fire Bird VI robot

2.1.1 Sensor Module

The sensor module consists of 8 sets of Ultrasonic and IR Proximity sensors and 8 channel White line sensors. The sensor module is interfaced to main control unit over I2C0 interface. The main task of sensor module is to gather all the sensor data and deliver it to the main control unit whenever demanded. The sensor module is also capable in controlling the power delivered to the IR Proximity and white line sensors through software. For more details on Sensor module, refer hardware manual.

2.1.2 Power Management Module

The Power management module monitors voltage, current and temperature of the battery. It is interfaced to the main control unit over I2C1 channel. It also provides alerts about the status of the battery voltage. For more details on Power Management module, refer hardware manual.

2.1.3 Inertial Measurement System

The Inertial Measurement System consists of 3 axis digital Accelerometer, 3 axis digital magnetometer and 3 axis digital gyroscope. These inertial sensors are interfaced to the main control unit over I2C1 interface.

2.1.4 Motor Controller

The motor controller is responsible for controlling the two DC motors of the robot. It is interfaced to the main control unit over UART2 interface.

2.1.5 Wireless Communication Module

The Wireless communication module features XBEE, WiFi and Bluetooth modules. At a time only one of these modules is interfaced to the main control unit over UART1 interface.

2.1.6 LCD Module

The LCD module is interfaced to the main control unit over I2C interface. It can be used to display sensor data or motor parameters.

2.1.7 Host /Embedded PC

The Host/Embedded PC communicates with main control unit through UART3-RS232 interface. The host/Embedded PC uses a command protocol in order to communicate with the robot. The protocol is thoroughly explained in chapter 3. A GUI is also developed using this protocol to test the ability of the robot. For more details on GUI please refer chapter 4.

2.1.8 GPS

Fire Bird VI robot also has the provision for interfacing a GPS module. The GPS module can be interfaced to UART0 port available on the rear panel of the robot. A compatible GPS module can be purchased separately by contacting Nex Robotics.

3 Overview of LPCXpresso IDE

LPCXpresso's IDE is an integrated software development environment for NXP's LPC Microcontrollers. It includes all the tools necessary to develop high quality software solutions in a timely and cost effective manner. LPCXpresso is based on Eclipse with many LPC specific enhancements. It also features the latest version of the industry standard GNU toolchain with a proprietary optimized C library providing professional quality tools at low cost. The LPCXpresso IDE can build an executable of any size with full code optimization and it supports a download limit of 128 kB after registration. The following figure shows single perspective (develop) view of the IDE.

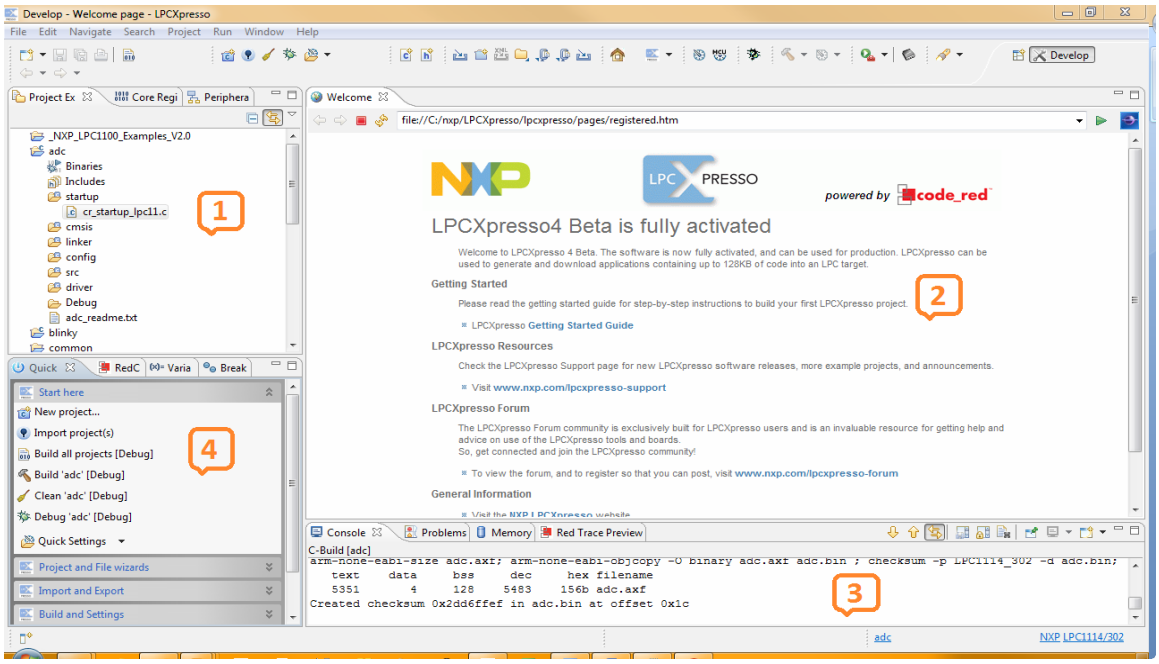


Figure 3.1: Single Perspective Develop View of LPCXpresso IDE

1. Project Explorer View: The 'Project Explorer' gives you a view of all the projects in your current 'Workspace'. A 'Workspace' is a collection of projects that are stored in a single Workspace Directory on your computer.

2. Editor: On the upper right is the editor which allows modification and saving of source code as well as setting breakpoints in debug mode.

3. Console and Problems Views: On the lower right are the Console and Problems Views. The Console View displays status information on compiling and debugging, as well as program output. The Problems View (available by changing tabs) shows all compiler errors and will navigate to the Editor View to the error location.

4. Quick Start View: The 'Quick Start' view has fast links to commonly used features. This is the best place to go to find options such as Build, Debug, and Import.

The following figure shows single perspective debug view of LPCXpresso IDE

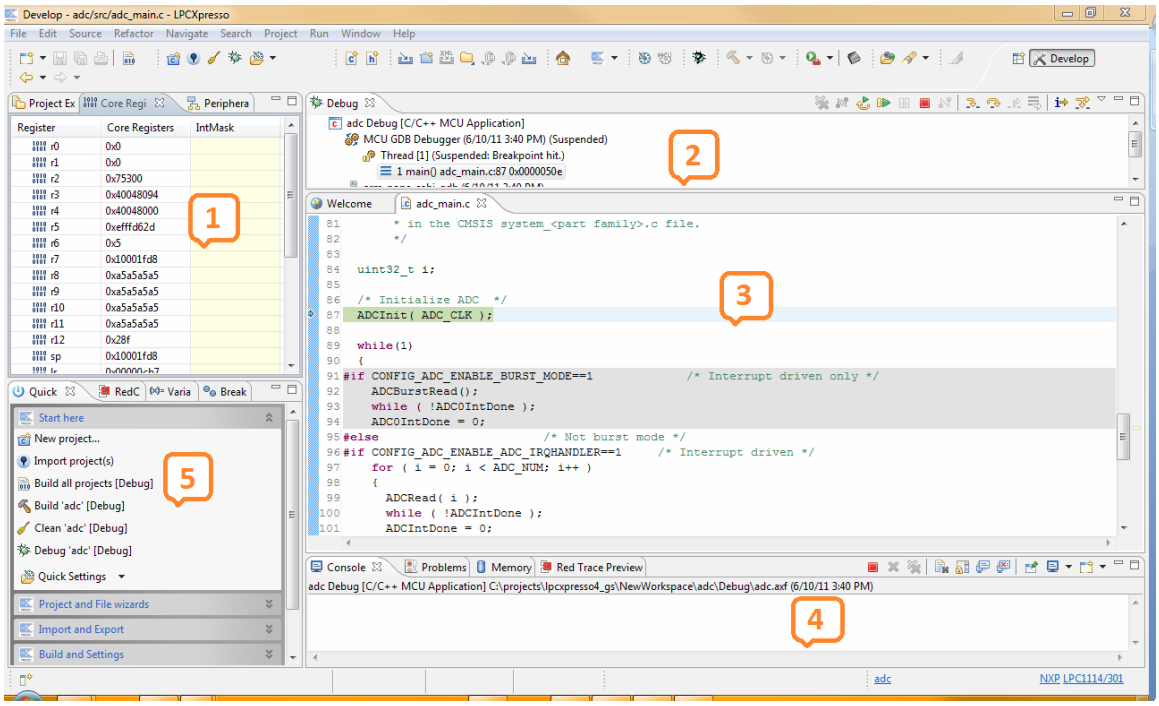


Figure 3.2: Single Perspective Debug View

1. Core Register View: This shows all of the registers in the processor core. Registers that have changed from previous step are highlighted in yellow.

2. Debug View: This shows you the stack trace and the debug toolbar. Using the icons at the top of the view, you can step through code or execute at full speed. In the 'stopped' state, you can click on any particular function and inspect its local variables in the right hand panel on the Variables tab.

3. Editor: In here you will see the code you are executing and can step from line to line. By clicking the 'i' icon at the top of the Debug view, you can switch to stepping by assembly instruction. Clicking in the left margin will set and delete breakpoints.

4. Console View: On the lower right is the Console View. The Console View displays status information on compiling and debugging, as well as program output.

5. Quick Start View: The 'Quick Start' view has fast links to commonly used features. This is the best place to go to find options such as Build, Debug, and Import.

3.1 Importing Existing Projects

LPCXpresso IDE has an option to import existing unpacked projects and archived projects into the workspace. This feature allows user to run existing sample projects or demo codes from the common workspace. The FireBird VI software package includes demo codes for GPS, Wifi Communication, Bluetooth Communication, Line follower, etc and CMSISv2 library. These codes are packed into "**FireBird VI Package.zip"** file. Use the following steps to unpack these codes in to a workspace.

1. Start LPCXpresso IDE from start menu or desktop shortcut.

2. Select the workspace folder to store your projects.

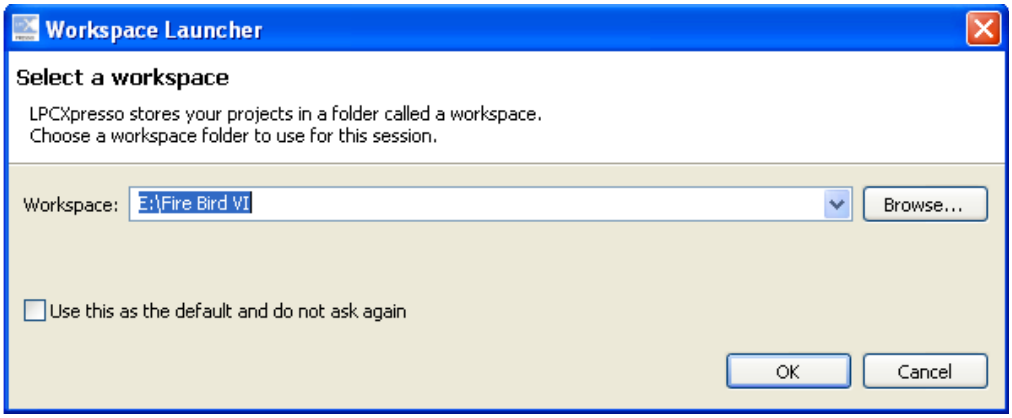


Figure 3.3: Selecting Workspace Folder

3. Copy "FireBird VI Package.zip" file from the documentation CD to your PC.

4. Select **Import projects** from quick start panel.

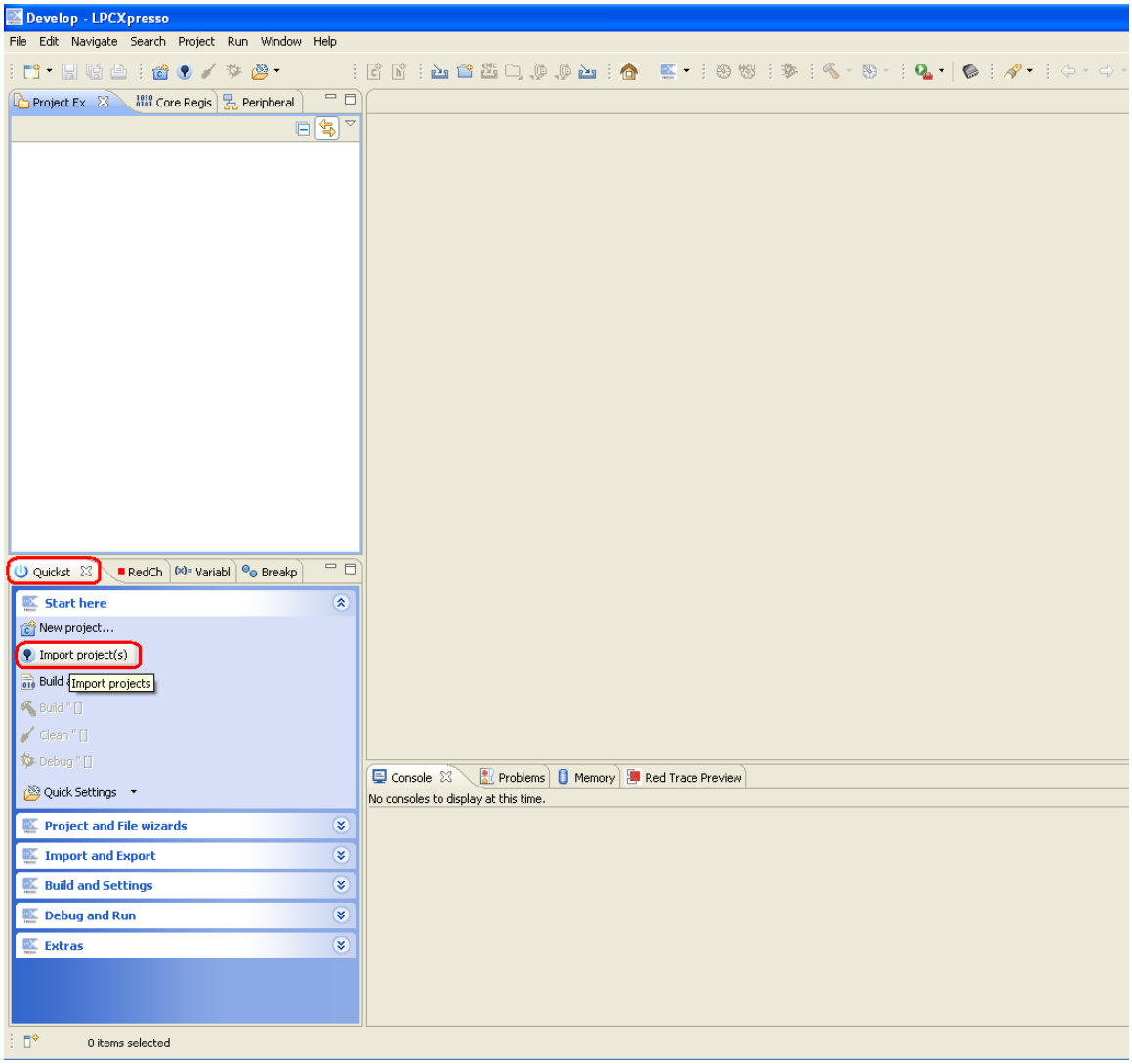


Figure 3.4: Importing Projects

5. Click browse to select the “FireBird VI Package.zip” file from its destination folder on your PC.

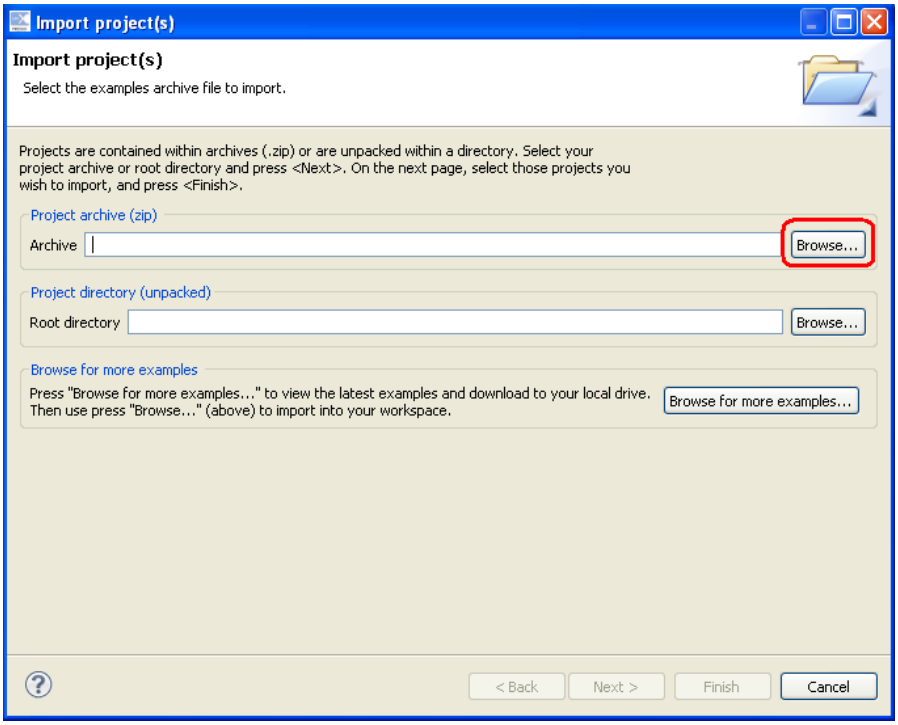


Figure 3.5: Selecting Zip File

6. After selecting the zip file, click **Next** to continue.

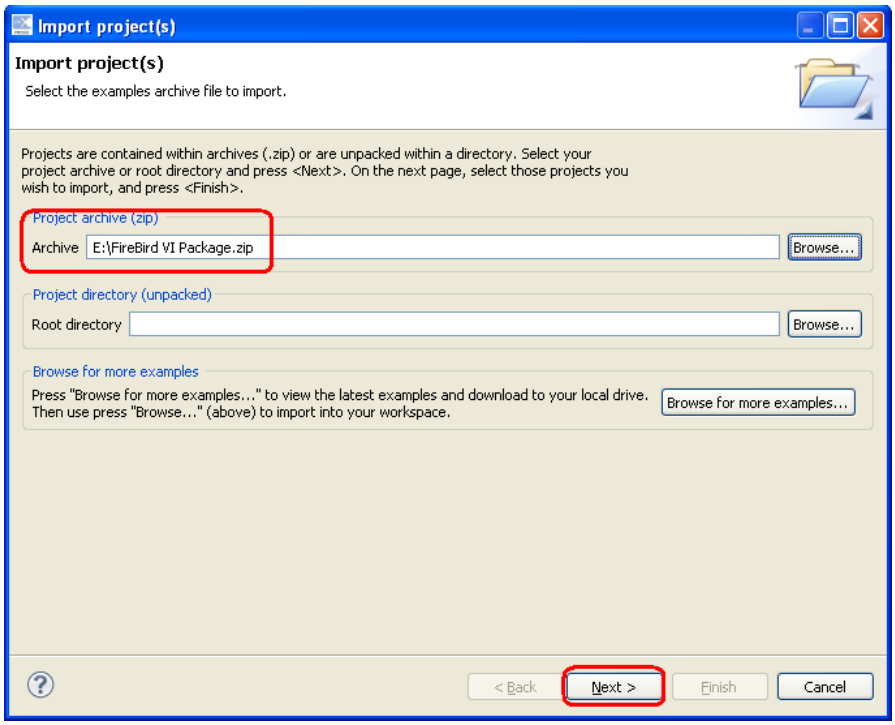


Figure 3.6: Selecting Zip File

7. In the next window, ensure that all the projects are selected and click finish to import projects.

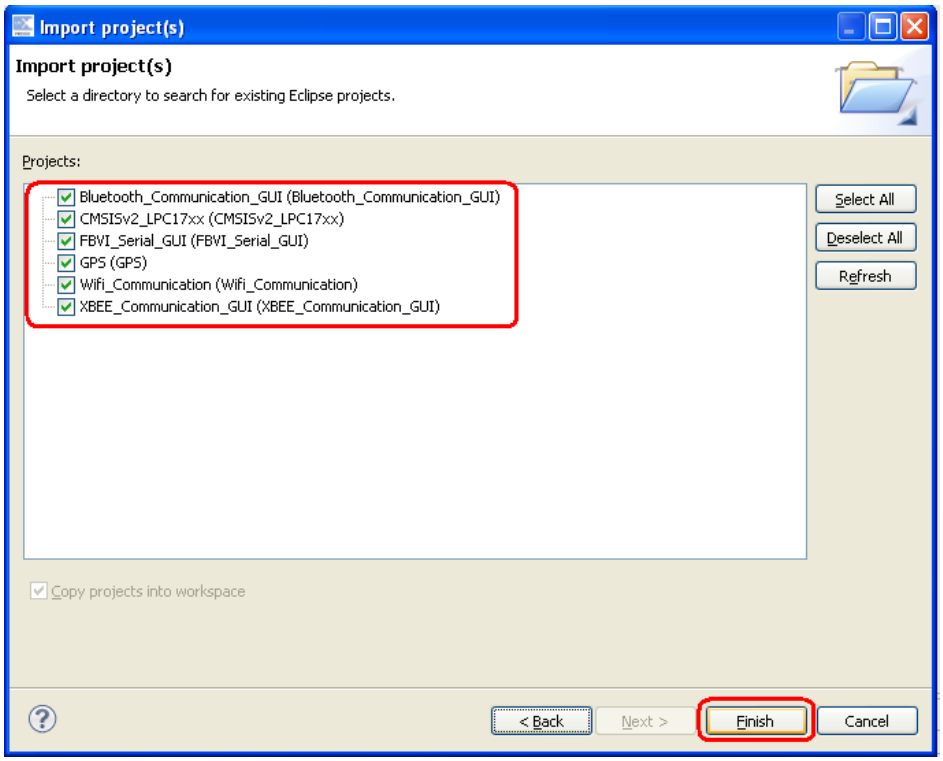


Figure 3.7: Selecting Projects

8. The imported projects should now be visible in the Project Explorer panel.

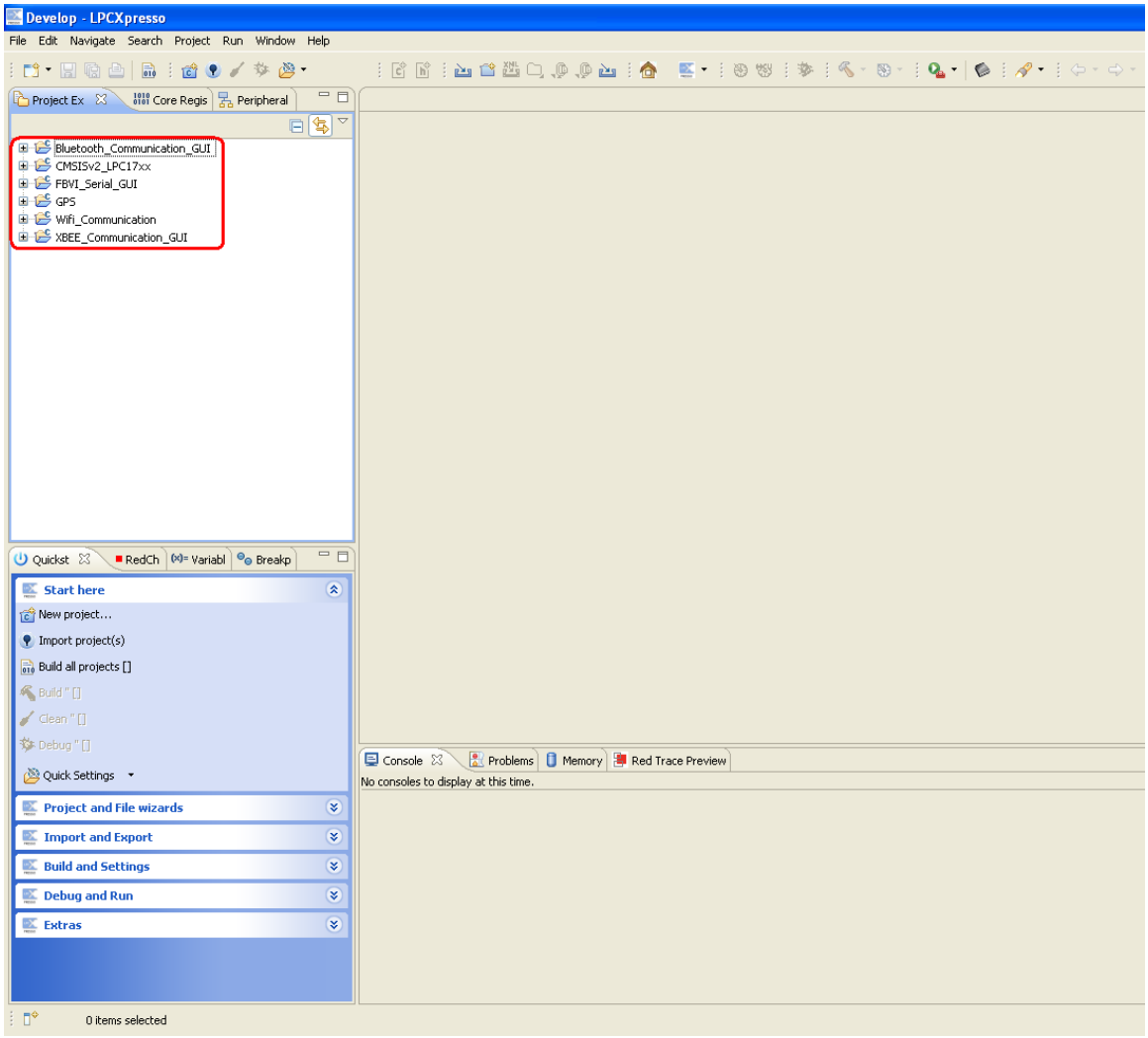


Figure 3.8

3.2 Setting Build Configurations

Like any other IDE, LPCXpresso IDE also allows to build **debug** and **release** configurations separately and any one configuration can be active at any time. By default the projects are set active in debug configuration. The configurations can be switched between debug and release mode by clicking on the tool bar icon as shown in fig. below.

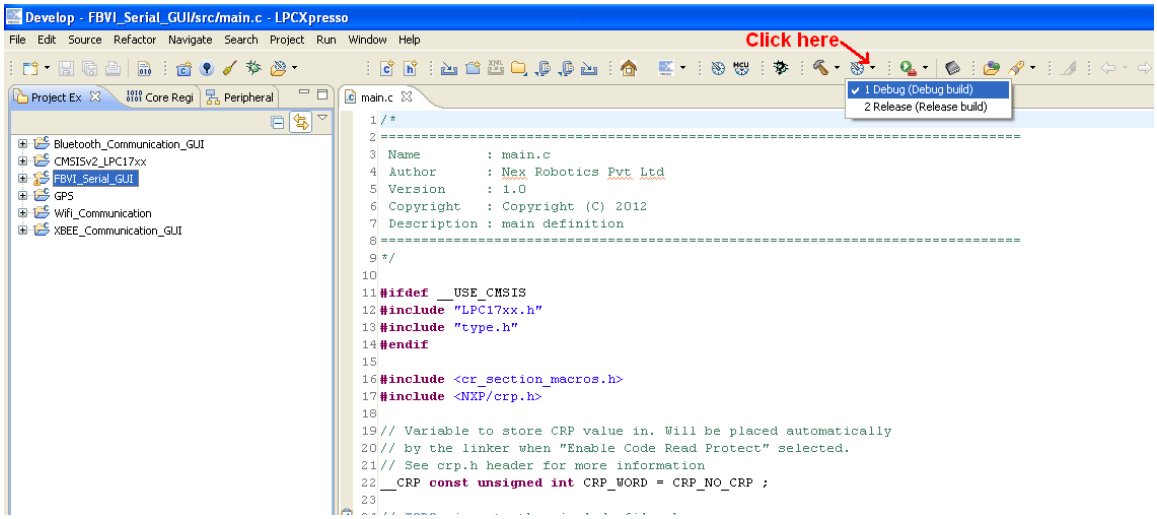


Figure 3.9: Setting Build Configuration

The Fire Bird VI related projects imported into the workspace folder are referenced to CMSISv2_LPC17xx library project in the workspace folder. **It is recommended to set same build configuration for the demo project and CMSIS project.**

3.3 Debugging a Project

Debugging a project is a one click process. The configuration settings for LPCXpresso JTAG debugger are already done at the time of JTAG installation. The debugging steps are explained below.

1. Connect JTAG USB cable to PC and 16 pin FRC connector to the robot's JTAG port.

Figure 3.10: Robot Side JTAG connection

2. Expand FBVI_Serial_GUI from the project exploer window and double click on main.c file in the src folder. This will open main.c file in the editor window at the same time quick start panel will also be indexed to FBVI_Serial_GUI project.

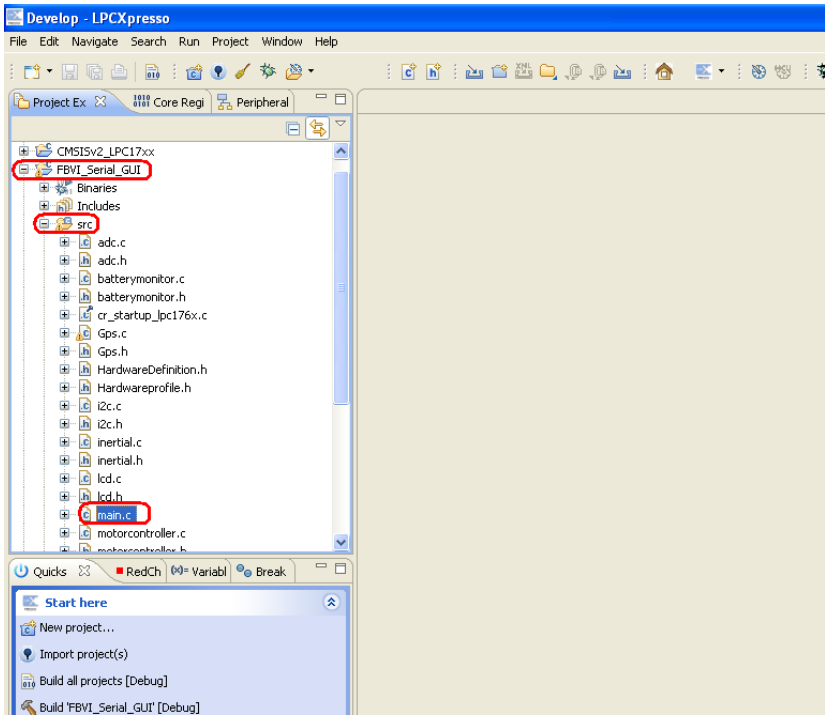


Figure 3.11

3. Turn ON the robot and click on Debug from quick start panel. This will build FBVI_Serial_GUI project and its reference project i.e. CMSISv2_LPC17xx project. After building the projects, LPCXpresso IDE will load the executable on the microcontroller.

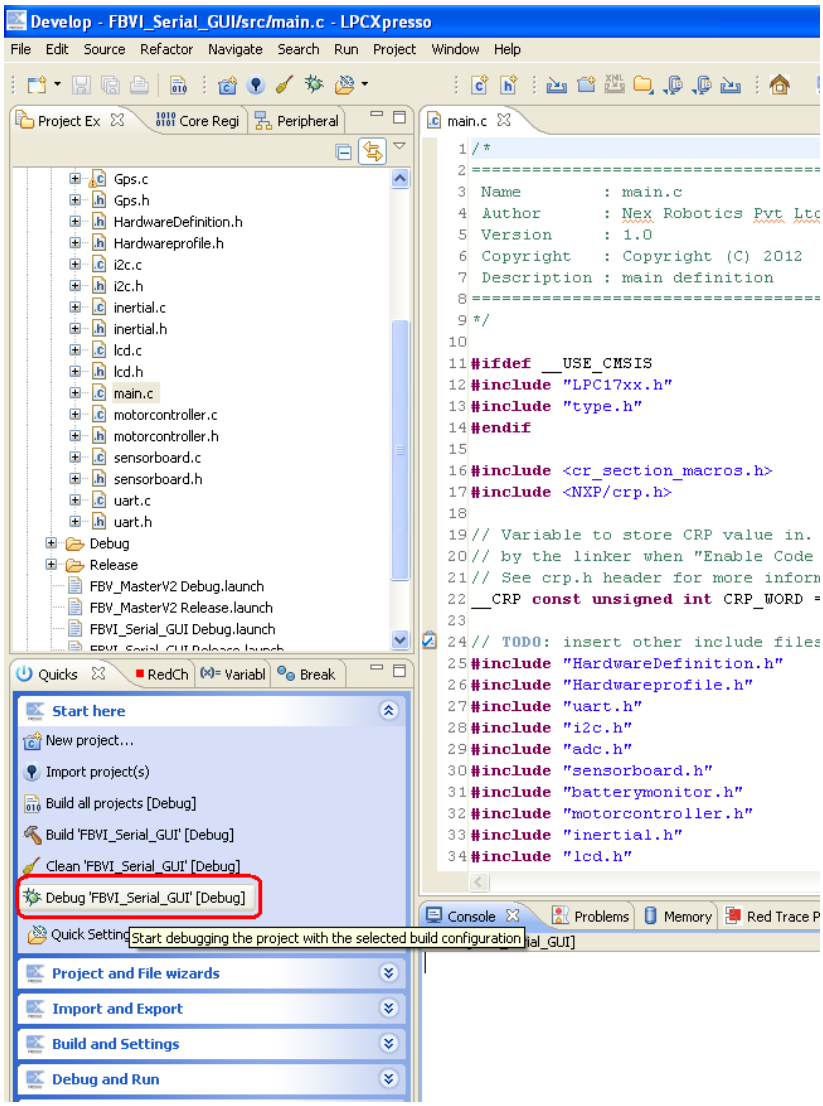


Figure 3.12: Debugging Current Project

4. On successfully loading the executable, LPCXpresso IDE will switch itself into single perspective debug mode.

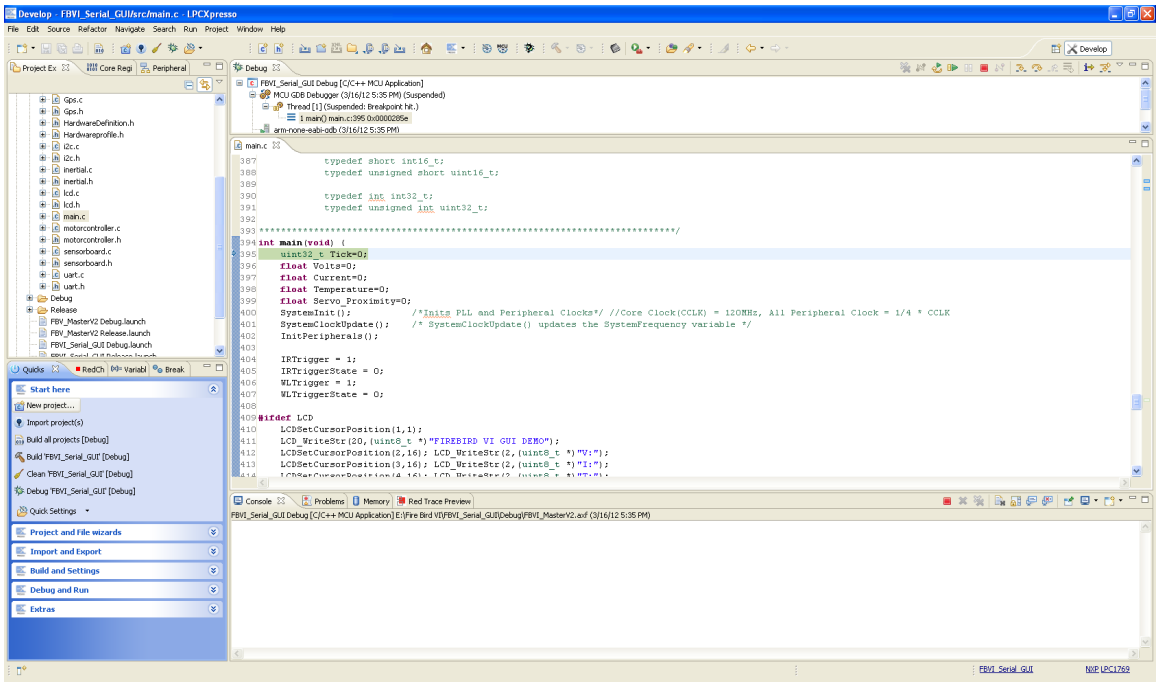


Figure 3.13: Single Perspective Debug View

5. To execute the code, click resume button in the debug panel. Alternatively you can also press F8.

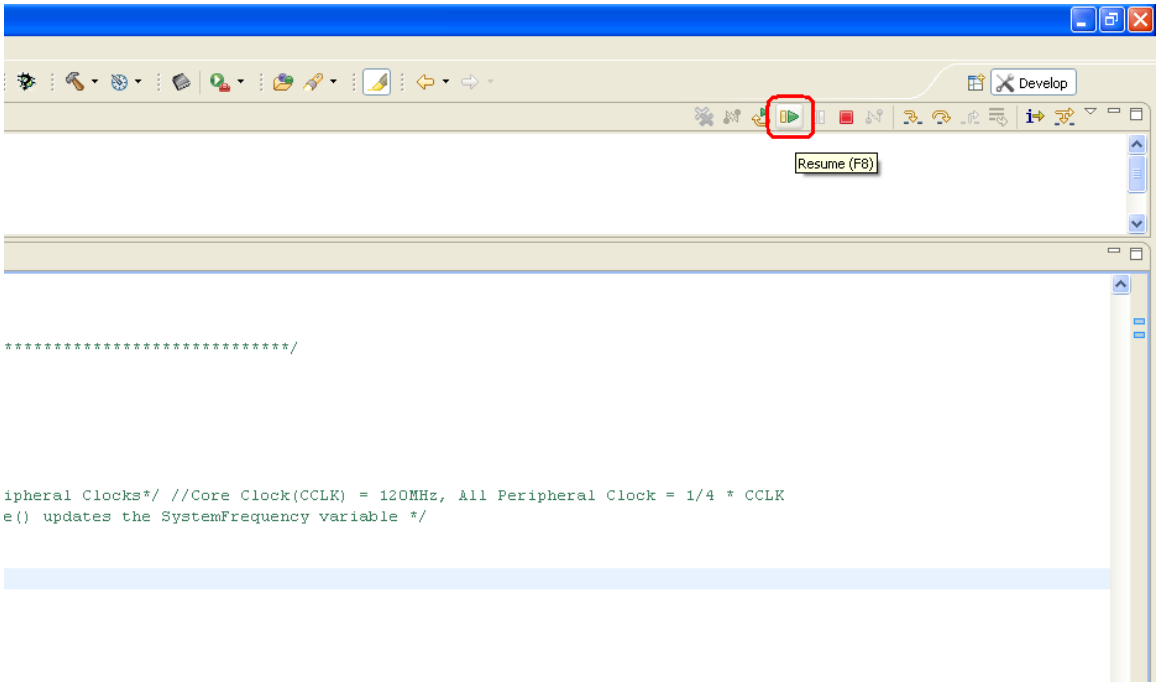


Figure 3.14: Executing Debug Configuration

6. To set a break point, double click in the blue region to the left of the line. To resume execution from the current break point, press F8 or click resume from the debug panel.

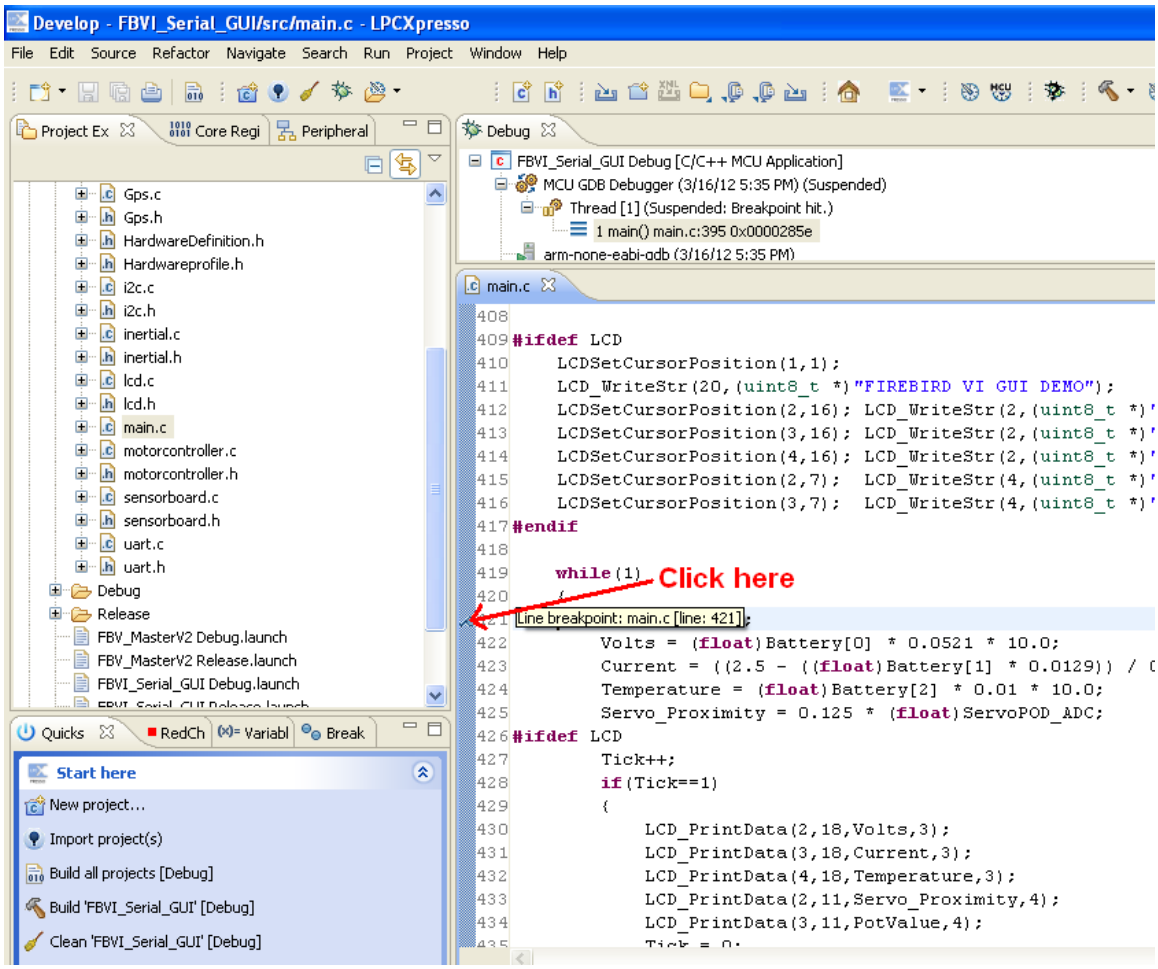


Figure 3.15: Setting Breakpoints

3.4 Loading Executable File

LPCXpresso allows loading the executable file directly without debugging the code. The steps are as follows.

1. Turn ON the robot and connect JTAG's USB cable to PC and 16 pin FRC connector to the robot's JTAG port.

2. Click build from the quick start panel.

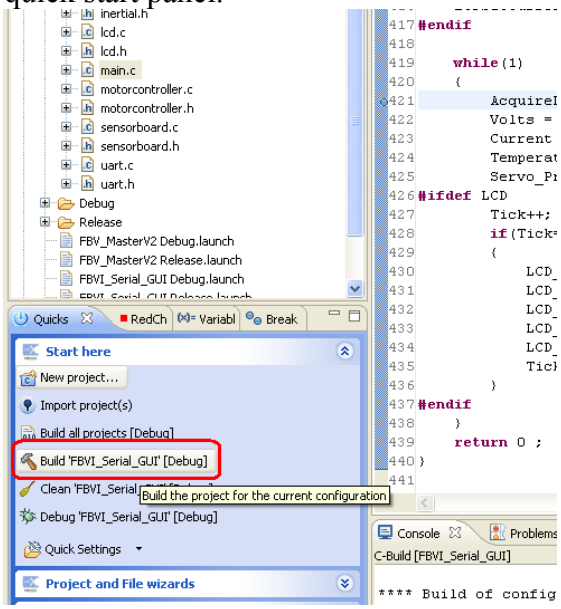


Figure 3.16: Building a Project

3. On successful build, click Program Flash button on the main tool bar.

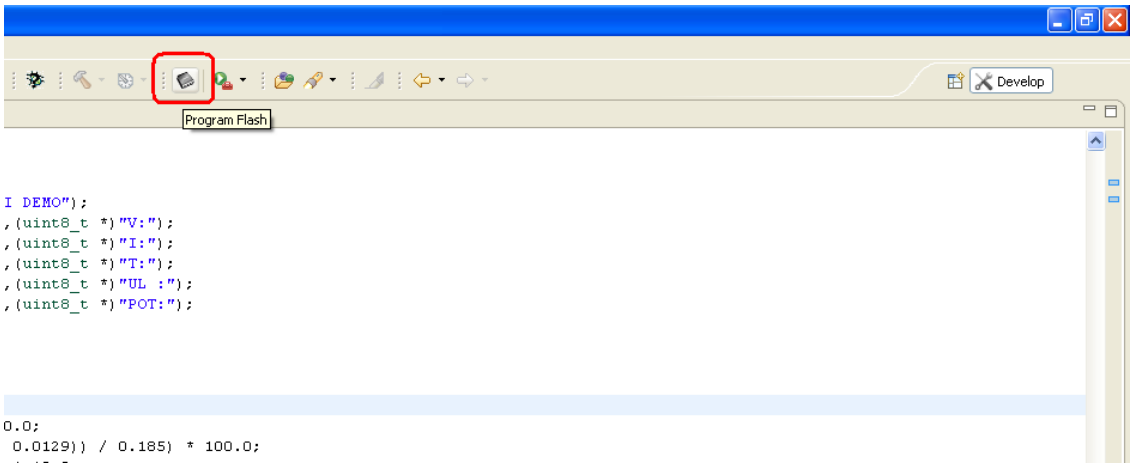


Figure 3.17: Programming Flash

4. In the following window, select **Reset target on completion** and select the appropriate .axf file from debug or release folder depending on the current build configuration. Also select **Mass erase** option and click OK.

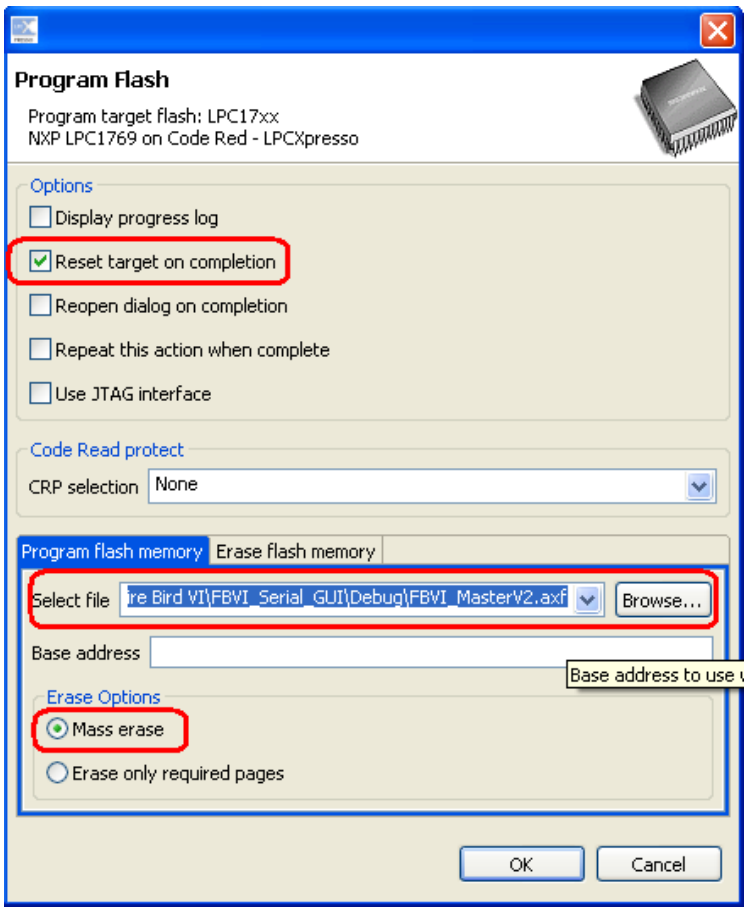


Figure 3.18: Selecting Flash Options

5. This will load the executable in to the microcontroller and reset the target to execute the code.

4 Fire Bird VI Communication Protocol

The robot communicates with host PC or embedded kit using the RS232 interface. The RS232 settings are as follows:

Baud Rate: 115200
Data Bits: 8
Parity: None
Stop Bits: None

The communication between the host computer and Fire Bird VI robot is performed by using designated commands. The host computer acts as a master and Fire Bird VI robot acts as a slave. The communication is always initiated by host device. The host device should introduce a delay of at least 3ms between two consecutive commands, so that the microcontroller gets sufficient time to process the earlier command. The communication protocol has a simple request and response structure. The Host PC requests a particular operation to which the target module responds accordingly, if required. The length of each request may vary depending on the task as shown in the table below.

Table 4.1 briefly explains the command packet sent from host to Fire Bird VI. First three byte of each command are always “NEX” which are then followed by the command type and the appropriate command data bytes

Command: Host to Fire Bird VI

Header			Command	Sub-Command/Data
‘N’	‘E’	‘X’	1byte	1byte

Table 4.1: Fire Bird VI Command Packet

In response to the command, the robot will start processing a task, and if necessary it will send a reply. The first three bytes of each reply are “FBD” followed by the command type and response data as shown in the table below. Table 4.2 explains the response packet received by host from Fire Bird VI.

Response: Fire Bird VI to Host

Header			Command	Sub-Command	Data
‘F’	‘B’	‘D’	1byte	1byte	1to16 bytes

Table 4.2: Fire Bird VI Response Packet

4.1 Sensor Module Commands

1. Get 8 bytes of Ultrasonic Sensor data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x01

Response: FBVI to Host
Length: 13 bytes

Header			Command	Sub-Command	Data							
'F'	'B'	'D'	0x05	0x01	UL0	UL1	UL2	UL3	UL4	UL7	UL6	UL7

2. Get Ultrasonic Sensor n Data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x01	n=0 to 7

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x01	n=0 to 7	1 byte

Here n=0 represents Ultrasonic sensor 0 data and so on

3. Get 8 bytes of Line Sensor data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x04

Response: FBVI to Host
Length: 13 bytes

Header			Command	Sub-Command	Data							
'F'	'B'	'D'	0x05	0x04	WL0	WL1	WL2	WL3	WL4	WL5	WL6	WL7

4. Get White Line Sensor n Data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x04	n=0 to 7

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x04	n=0 to 7	1 byte

Here n=0 represents White line sensor 0 data and so on

5. Get 8 bytes of IR Proximity data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x02

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data							
'F'	'B'	'D'	0x05	0x02	IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7

6. Get IR Proximity Sensor n data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x02	n=0 to 7

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x02	n=0 to 7	1 byte

Here n=0 represents IR Proximity sensor 0 data and so on.

7. Turn ON IR Proximity Sensor

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x01

Response: None

8. Turn OFF IR Proximity Sensor

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x02

Response: None

9. Turn ON White Line Sensor

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x03

Response: None

10. Turn OFF White Line Sensor

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x04

Response: None

11. Turn ON Ultrasonic Sensor Ranging

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x05

Response: None

12. Turn OFF Ultrasonic Sensor Ranging

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x06

Response: None

13. Turn ON Servo Pod Ultrasonic Sensor Ranging

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x07

Response: None

14. Turn OFF Servo Pod Ultrasonic Sensor Ranging

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x09	0x08

Response: None

15. Get IR Proximity Sensor ON/OFF status

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x0A	0x01

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x01	0x00 or 0x01

0x00 is OFF
0x01 is ON

16. Get White Line Sensor ON/OFF status

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x0A	0x02

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x02	0x00 or 0x01

0x00 is OFF
0x01 is ON

17. Get Ultrasonic Sensor Ranging status

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x0A	0x03

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x03	0x00 or 0x01

0x00 is OFF
0x01 is ON

18. Get Servo Pod Ultrasonic Sensor Ranging status

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x0A	0x04

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x04	0x00 or 0x01

0x00 is OFF
0x01 is ON

4.2 Sensor Module APIs

File Name:
sensorboard.c

Function:

```
uint32_t Set_WL_Trigger(uint8_t Data);
```

Description:

This function can be used for switching the power source of line sensors ON or OFF.

Input Parameters:

Parameter	Description
uint8_t Data	0x00=OFF 0x01=ON

Returns:

If command is successfully sent, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
Status =(Set_WL_Trigger(0x01));
return Status ;
```

Function:

```
uint32_t Set_IR_Trigger(uint8_t Data);
```

Description:

This function can be used for switching the power source of IR proximity sensors ON or OFF.

Input Parameters:

Parameter	Description
uint8_t Data	0x00=OFF 0x01=ON

Returns:

If command is successfully sent, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
Status =(Set_IR_Trigger(0x01));
return Status ;
```

Function:

```
uint32_t Set_UL_Trigger(uint8_t Data);
```

Description:

This function can be used for switching the power source of ultrasonic sensors ON or OFF.

Input Parameters:

Parameter	Description
uint8_t Data	0x00=OFF 0x01=ON

Returns:

If command is successfully sent, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
Status =(Set_UL_Trigger(0x01));
return Status ;
```

Function:

```
uint32_t Get_Ultrasonic_Data(uint8_t *Data);
```

Description:

This function gets 8 bytes of ultrasonic sensor data from the sensor module.

Input Parameters:

Parameter	Description
uint8_t *Data	Pointer to store the data

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
uint32_t Status;
uint8_t Ultrasonic[8];
Status = Get_Ultrasonic_Data(Ultrasonic);
return Status ;
```

Function:

```
uint32_t Get_Whiteline_Data(uint8_t *Data);
```

Description:

This function gets 8 bytes of line sensor data from the sensor module.

Input Parameters:

Parameter	Description
uint8_t *Data	Pointer to store the data

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
uint32_t Status;
uint8_t Whiteline[8];
Status = Get_Whiteline_Data(Whiteline);
return Status ;
```

Function:

```
uint32_t Get_IR_Proximity_Data(uint8_t *Data);
```

Description:

This function gets 8 bytes of IR Proximity sensor data from the sensor module.

Input Parameters:

Parameter	Description
uint8_t *Data	Pointer to store the data

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

I2C0Init() should be called before calling this function.
Note: I2C0Init is automatically called if ULTRASONIC or WHITELINE or IR_PROXIMITY is defined in Harwareprofile.h

Example:

```
uint32_t Status;
uint8_t IR_Proximity[8];
Status = Get_IR_Proximity_Data (IR_Proximity);
return Status ;
```

4.3 Power Management Module Commands

1. Read Battery Voltage

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x20	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x20	0x00	Battery Voltage

2. Read Battery Current

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x21	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x21	0x00	Battery Current

3. Read Battery Temperature

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x22	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x22	0x00	Battery Temperature

4. Read Battery Voltage, Current & Temperature

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x23	0x00

Response: FBVI to Host
Length: 8 bytes

Header			Command	Sub-Command	Data0	Data1	Data2
'F'	'B'	'D'	0x23	0x00	Voltage	Current	Temperature

Battery Voltage calculation:

Battery voltage = Data * 0.0521

For e.g. if Data = 225 dec or 0xE1,
Then Battery Voltage = 225 * 0.0521 = 11.72 Volts

Battery current calculation:

Battery Current = ((2.5 - (Data * 0.0129))) / 0.185

For e.g. if Data = 170 dec or 0xAA,
Then Battery Current = ((2.5 - (170 * 0.0129))) /0.185 = 1.6 Amps

Battery Temperature calculation:

Battery Temperature = (Data * 1.29)

For e.g. if Data = 20,
Then Battery Temperature = (20 * 1.29) = 25.8 deg. C

4.4 Power Management Module APIs

File Name:
batterymonitor.c

Function:

```
uint32_t Get_Battery_Status(uint8_t *Data);
```

Description:

This function gets battery status data from power management module. It reads three bytes of data where byte[0] is battery voltage, byte[1] is battery current and byte[2] is battery temperature.

Input Parameters:

Parameter	Description
uint8_t *Data	Points to the address where data will be stored

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

BATTERYMONITOR macro in HardwareProfile.h should be uncommented before calling this function.

Example:

```
uint32_t Status;
uint8_t Battery[3];
Status = Get_Battery_Status (Battery);
return Status ;
```

4.5 Motion Control Commands

1. Set Robot Direction

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x94	Direction

Direction:
0x01 is Forward
0x02 is Reverse
0x03 is Left
0x04 is Right
0x06 is Stop

Response: None

2. Get Left Motor Encoder Counts

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x92	0x00

Response: FBVI to Host
Length: 9 bytes

Header			Command	Sub-Command	Data0	Data1	Data2	Data3
'F'	'B'	'D'	0x92	0x00	MSB	Byte2	Byte3	LSB

3. Get Right Motor Encoder Counts

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x93	0x00

Response: FBVI to Host
Length: 9 bytes

Header			Command	Sub-Command	Data0	Data1	Data2	Data3
'F'	'B'	'D'	0x93	0x00	MSB	Byte2	Byte3	LSB

4. Clear Encoder Counts

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x8C	0x00

Response: None

5. Set Left Motor Velocity

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x95	Velocity (1 byte)

Note: Velocity = 0-Full forward,
= 128-Stop,
= 255-Full reverse

Response: None

6. Set Right Motor Velocity

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x96	Velocity (1byte)

Note: Velocity = 0-Full forward,
= 128-Stop,
= 255-Full reverse

Response: None

7. Get Left Motor Current

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x97	0x01

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x97	0x01	I_Left

I_Left provides 10 times number of amps i.e. value 25 decimal indicates 2.5 Amps

8. Get Right Motor Current

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x97	0x02

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x97	0x02	I_Right

I_Right provides 10 times number of amps i.e. value 25 decimal indicates 2.5 Amps

9. Set Acceleration

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x98	Acceleration

Response: None

10. Get Acceleration

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x99	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x99	0x00	Acceleration

11. Get Left Motor Velocity

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x9A	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x9A	0x00	Left Velocity

12. Get Right Motor Velocity

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x9B	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x9B	0x00	Right Velocity

13. Set Mode

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x90	Mode

Response: None

Refer Chapter 5 for detailed description of Mode.

14. Get Mode

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x91	0x00

Response: FBVI to Host
Length: 5 bytes

Header			Command	Sub-Command/Data
'F'	'B'	'D'	0x91	Mode

15. Set Safety ON / OFF

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x89	Safety

Response: None

Safety = 1 → Safety ON
Safety = 0 → Safety OFF

16. Set Position

Command: Host to FB VI
Length: 14 bytes

Header			Command	Position 1	Velocity 1	Position 2	Velocity 2
'N'	'E'	'X'	0x9C	0x00 0x00 0x10 0x17	0x50	0x00 0x00 0x10 0x17	0x50

Response: None

4.6 Motion Control APIs

File Name:
motorcontroller.c

Function:

```
void InitMotorController(void);
```

Description:

This function sets initial startup parameters for the motor controller.

Input Parameters: None

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
InitMotorController();
```

Function:

```
void SetAcceleration(uint8_t Acc);
```

Description:

Sets the acceleration parameter of the motion controller.

Input Parameters:

Parameter	Description
uint8_t Acc	1 byte acceleration value in the range of 1-10.

Returns:

None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
SetAcceleration(5);
```

Function:

```
uint8_t Get_Motor_Voltage(void);
```

Description:

Gets Motor controller voltage status. It reads 10 times the motor voltage. i.e. value 120 means 12.0V

Input Parameters: None

Returns:

1 byte data containing motor voltage

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t MotorVoltage=0;
MotorVoltage = Get_Motor_Voltage();
```

Function:

```
uint8_t Get_LeftMotor_Current(void);
```

Description:

Gets Left motor current value. It reads 10 times number of amps i.e. value 025 indicates 2.5 Amps

Input Parameters: None

Returns:

1 byte data containing left motor current.

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t LeftMotorCurrent=0;
LeftMotorCurrent = Get_LeftMotor_Current ();
```

Function:

```
uint8_t Get_RightMotor_Current(void);
```

Description:

Gets Right motor current value. It reads 10 times number of amps i.e. value 025 indicates 2.5 Amps.

Input Parameters: None

Returns:

1 byte data containing left motor current.

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t RightMotorCurrent=0;
RightMotorCurrent = Get_RightMotor_Current();
```

Function:

```
uint32_t GetLeftMotorCount(void);
```

Description:

This function gets 32 bit encoder counts of left motor.

Input Parameters: None

Returns:

32 bit value containing left motor encoder counts.

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t Count=0;
Count = GetLeftMotorCount ();
```

Function:

```
uint32_t GetRightMotorCount(void);
```

Description:

This function reads right motor encoder counts from motion control module.

Input Parameters: None

Returns:

32-bit unsigned encoder count value

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t Count=0;
Count = GetRightMotorCount ();
```

Function:

```
void ClearEncoderCounts(void);
```

Description:

This function clears left and right motor encoder counts.

Input Parameters: None

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
ClearEncoderCounts();
```

Function:

```
void Move(uint8_t LeftSpeed, uint8_t RightSpeed);
```

Description:

This function controls the locomotion of robot. It takes speed parameters as arguments.

Input Parameters:

Parameter	Description
`uint8_t LeftSpeed`	0=Full Forward, 128=Stop, 255=Full Reverse
`uint8_t RightSpeed`	0=Full Forward, 128=Stop, 255=Full Reverse

Returns:

None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
Move(63,63);       //Moves forward at half speed
```

Function:

```
void Stop(void);
```

Description:

This function stops motion of the robot.

Input Parameters: None

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
Move(63,63);       //Move Forward
Delay();           //Move Forward for 5 Seconds
Stop();            //Stops motion of  robot
```

Function:

```
void SetMode(uint8_t Mode);
```

Description:

This function sets the motion controller mode.

Input Parameters:

Parameter	Description
`uint8_t Mode`	0=Open Loop, 1=Closed loop, 2=Position control

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
SetMode(2);        //Set mode to position control
```

Function:

```
uint8_t GetMode(void);
```

Description:

This function gets the motion controller mode.

Input Parameters: None

Returns: Mode

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint8_t Mode;
Mode = GetMode();        //Get motion control mode
```

Function:

```
void EnableSafety(void);
```

Description:

This function enables safety mode and restricts maximum velocity.

Input Parameters: None

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
EnableSafety();   //Enable safety mode
```

Function:

```
void DisableSafety(void);
```

Description:

This function disables safety mode and removes restriction on maximum velocity.

Input Parameters: None

Returns: None

Precondition:

MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
DisableSafety();  //Disable safety mode
```

Function:

```
void SetPosition(int32_t positionLeft, int8_t velocityLeft, int32_t
positionRight, int8_t velocityRight)
```

Description:

This function is used to move robot by specified number of counts. The direction of motion for each motor is decided by velocity. The number of encoder counts to travel is always taken as absolute value of signed target count specified. On executing this command, the encoder count registers for both the motors are reset and the position counting always starts from zero.

Input Parameters:
positionLeft -> target position count of left motor encoder
velocityLeft -> velocity of left motor
positionRight -> target position count of right motor encoder
velocityRight -> velocity of right motor

Returns: None

Precondition:

1. MOTORCONTROLLER macro in Hardwareprofile.h should be uncommented before calling this function.
2. Set Mode value to 2

Example:

```
SetMode(2);
SetPosition(4119, 50, 4119, 50);
```

4.7 Inertial Sensors Commands

1. Get all 3 axis data of Accelerometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x05

Response: FBVI to Host
Length: 11 bytes

Header			Command	Sub-Command	Data0	Data1	Data2	Data3	Data4	Data5
'F'	'B'	'D'	0x05	0x05	x-axis Lsb	x-axis Msb	y-axis Lsb	y-axis Msb	z-axis Lsb	z-axis Msb

Convert Accelerometer data into g:

Acceleration (g) = (16 bit Raw data) * 0.004 / 16

2. Get all 3 axis data of Gyroscope

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x06

Response: FBVI to Host
Length: 11 bytes

Header			Command	Sub-Command	Data0	Data1	Data2	Data3	Data4	Data5
'F'	'B'	'D'	0x05	0x06	x-axis Lsb	x-axis Msb	y-axis Lsb	y-axis Msb	z-axis Lsb	z-axis Msb

Convert Gyroscope data into deg/sec:

Rate (deg/sec) = ((16 bit Raw Data) / 57.0) *0.01

3. Get all 3 axis data of Magnetometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x07

Response: FBVI to Host
Length: 11 bytes

Header			Command	Sub-Command	Data0	Data1	Data2	Data3	Data4	Data5
'F'	'B'	'D'	0x05	0x07	x-axis Msb	x-axis Lsb	y-axis Msb	y-axis Lsb	z-axis Msb	z-axis Lsb

Convert Magnetometer data into gauss:

For X and Y axis
Magnetic Field (gauss) = (16 bit Raw Data) / 1100.0)

For Z axis
Magnetic Field (gauss) = (16 bit Raw Data) / 980.0)

4. Get X axis data of Accelerometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x10	0x01

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x10	0x01	LSB	MSB

Convert Accelerometer data into g:

Acceleration (g) = (16 bit Raw data) * 0.004 / 16

5. Get Y axis data of Accelerometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x10	0x02

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x10	0x02	LSB	MSB

Convert Accelerometer data into g:

Acceleration (g) = (16 bit Raw data) * 0.004 / 16

6. Get Z axis data of Accelerometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x10	0x03

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x10	0x03	LSB	MSB

Convert Accelerometer data into g:

Acceleration (g) = (16 bit Raw data) * 0.004 / 16

7. Get X axis data of Gyroscope

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x11	0x01

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x11	0x01	LSB	MSB

Convert Gyroscope data into deg/sec:

Rate (deg/sec) = ((16 bit Raw Data) / 57.0) *0.01

8. Get Y axis data of Gyroscope

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x11	0x02

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x11	0x02	LSB	MSB

Convert Gyroscope data into deg/sec:

Rate (deg/sec) = ((16 bit Raw Data) / 57.0) *0.01

9. Get Z axis data of Gyroscope

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x11	0x03

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x11	0x03	LSB	MSB

Convert Gyroscope data into deg/sec:

Rate (deg/sec) = ((16 bit Raw Data) / 57.0) *0.01

10. Get X axis data of Magnetometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x12	0x01

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x12	0x01	MSB	LSB

Convert Magnetometer data into gauss:

Magnetic Field (gauss) = (16 bit Raw Data) / 1100.0)

11. Get Y axis data of Magnetometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x12	0x02

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x12	0x02	MSB	LSB

Convert Magnetometer data into gauss:

Magnetic Field (gauss) = (16 bit Raw Data) / 1100.0)

12. Get Z axis data of Magnetometer

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x12	0x03

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data0	Data1
'F'	'B'	'D'	0x12	0x03	MSB	LSB

Convert Magnetometer data into gauss:

Magnetic Field (gauss) = (16 bit Raw Data) / 980.0)

4.8 Inertial Sensors APIs

File Name:
inertial.c

Function:

```
void Init_L3G4200D(void);
```

Description:

This function sets initial startup parameters of L3G4200D 3 axis digital gyroscope. It is automatically called in InitPeripherals() function in main.c, if GYROSCOPE is defined in Hardwareprofile.h

Input Parameters: None

Returns: None

Precondition:

uncomment GYROSCOPE definition in Hardwareprofile.h

Example:

```
Init_L3G4200D();
```

Function:

```
void Init_LSM303DLHC_Accelerometer(void);
```

Description:

This function sets initial startup parameters of LSM303DLHC 3 axis digital accelerometer. It is automatically called in InitPeripherals() function in main.c, if ACCELEROMETER is defined in Hardwareprofile.h

Input Parameters: None

Returns: None

Precondition:

uncomment ACCELEROMETER definition in Hardwareprofile.h

Example:

```
Init_LSM303DLHC_Accelerometer();
```

Function:

```
void Init_LSM303DLHC_Magnetometer(void);
```

Description:

This function sets initial startup parameters of LSM303DLHC 3 axis digital magnetometer. It is automatically called in InitPeripherals() function in main.c, if GYROSCOPE is defined in Hardwareprofile.h

Input Parameters: None

Returns: None

Precondition:

uncomment MAGNETOMETER definition in Hardwareprofile.h

Example:

```
Init_LSM303DLHC_Magnetometer();
```

Function:

```
uint32_t Get_XYZ_Rate(uint8_t *Data);
```

Description:

This function gets all 3 axis data from L3D4200D digital gyroscope. The 6 byte data is stored in the buffer indicated by pointer that is passed to the function. The sequence of data is X_{lsb}, X_{msb}, Y_{lsb}, Y_{msb}, Z_{lsb}, Z_{msb}.

Input Parameters:

Parameter	Description
`uint8_t *Data`	Pointer to the buffer where data will be stored

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

uncomment GYROSCOPE definition in Hardwareprofile.h

Example:

```
uint32_t Status;
uint8_t Gyroscope[8];
Status = Get_XYZ_Rate (Gyroscope);
return Status ;
```

Function:

```
uint32_t Get_XYZ_Acceleration(uint8_t *Data);
```

Description:

This function gets all 3 axis data from LSM303DLHC digital accelerometer. The 6 byte data is stored in the buffer indicated by pointer that is passed to the function. The sequence of data is X_{lsb}, X_{msb}, Y_{lsb}, Y_{msb}, Z_{lsb}, Z_{msb}.

Input Parameters:

Parameter	Description
`uint8_t *Data`	Pointer to the buffer where data will be stored

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

uncomment ACCELEROMETER definition in Hardwareprofile.h

Example:

```
uint32_t Status;
uint8_t Accelerometer[8];
Status = Get_XYZ_Acceleration(Accelerometer);
return Status ;
```

Function:

```
uint32_t Get_XYZ_Magnetometer(uint8_t *Data);
```

Description:

This function gets all 3 axis data from LSM303DLHC digital magnetometer. The 6 byte data is stored in the buffer indicated by pointer that is passed to the function. The sequence of data is X_{msb}, X_{lsb}, Y_{msb}, Y_{lsb}, Z_{msb}, Z_{lsb}.

Input Parameters:

Parameter	Description
`uint8_t *Data`	Pointer to the buffer where data will be stored

Returns:

If data is successfully fetched, returns SUCCESS
Else returns FAILURE

Precondition:

uncomment MAGNETOMETER definition in Hardwareprofile.h

Example:

```
uint32_t Status;
uint8_t Magnetometer[8];
Status = Get_XYZ_Magnetometer(Magnetometer);
return Status ;
```

4.9 LCD Module APIs

Filename:
lcd.c

```
void InitLCD(void);
```

Description:

Initializes LCD module. Clears any data on the screen, turns ON backlight_and hides cursor. It is automatically called in InitPeripherals() function in main.c, if LCD is defined in Hardwareprofile.h

Input Parameters: None

Returns: None

Precondition:

uncomment LCD definition in Hardwareprofile.h

Example:

```
InitLCD();
```

Function:

```
uint32_t LCD_WriteStr(uint8_t Len,uint8_t *Data);
```

Description:

This function sends string of characters to LCD. The LCD accepts ASCII characters.

Input Parameters:

Parameter	Description
`uint8_t Len`	Length of string to be written
`uint8_t *Data`	Points to the string to be written

Returns:

If data is successfully written, returns SUCCESS
Else returns FAILURE

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function. The cursor position should be set using **`LCDSetCursorPosition`** before calling this function.

Example:

```
LCD_WriteStr(12,(uint8_t*)"Nex Robotics\0");
```

Function:

```
uint32_t LCDSetCursorPosition(uint8_t Row, uint8_t Col);
```

Description:

This function sets the cursor position on LCD.

Input Parameters:

Parameter	Description
`uint8_t Row`	Sets cursor row (1 to 4)
`uint8_t Col`	Sets cursor column(1 to 20)

Returns:

Returns I2C bus status. Refer i2c.h for status codes

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
if(LCDSetCursorPosition(1,1)==I2C_OK)
{
      LCD_WriteStr(12,(uint8_t*)"Nex Robotics\0");
}
```

Function:

```
uint32_t LCDSetCursorHome(void);
```

Description:

This function sets the cursor position back to home position i.e. row=1 & col=1.

Input Parameters:
None

Returns:

Returns I2C bus status. Refer i2c.h for status codes

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:
None

Function:

```
uint32_t LCDSetBackLight(uint8_t Data);
```

Description:

This function controls the back light of LCD display.

Input Parameters:

Parameter	Description
`uint8_t Data`	1=ON 0=OFF

Returns:

Returns I2C bus status. Refer i2c.h for status codes

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
LCDSetBackLight(1);  //Turn ON LCD back light
```

Function:

```
uint32_t LCDClearScreen(void);
```

Description:

This function clears entire LCD screen and sets cursor to home position.

Input Parameters:
None

Returns:

Returns I2C bus status. Refer i2c.h for status codes

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
uint32_t LCDClearScreen();
```

Function:

```
uint32_t LCDSetBackLight(uint8_t Data);
```

Description:

This function controls the back light of LCD display.

Input Parameters:

Parameter	Description
`uint8_t Data`	1=ON 0=OFF

Returns:

Returns I2C bus status. Refer i2c.h for status codes

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
LCDSetBackLight(1);  //Turn ON LCD back light
```

Function:

```
uint32_t LCD_PrintData(uint8_t Row,uint8_t Col,uint32_t Number,uint8_t
Digits);
```

Description:

This function prints decimal data on the LCD screen. It formats decimal number upto 5 digits in to a string. The number of digits can also be specified.

Input Parameters:

Parameter	Description
`uint8_t Row`	Specifies row number
`uint8_t Col`	Specifies column number
`uint32_t Number`	The decimal number to be converted.
`uint8_t Digits`	Specifies number of digits to be displayed (1 to 5)

Returns:

If successfully written, returns SUCCESS
Else FAILURE

Precondition:

LCD macro in Hardwareprofile.h should be uncommented before calling this function.

Example:

```
LCD_PrintData (1,1,12345,5);
```

4.10 Servo Pod Commands

1. Set Servo Pod Pan angle

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x06	Pan angle(0 - 180)

Response: None

2. Set Servo Pod Tilt angle

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x07	Tilt angle(0 - 180)

Response: None

3. Get Servo Pod Pan Angle

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0xAA	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0xAA	0x00	Pan Angle

4. Get Servo Pod Tilt Angle

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0xAB	0x00

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0xAB	0x00	Tilt Angle

5. Get Servo Pod Ultrasonic Sensor Status

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x0A	0x04

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x04	0=OFF, 1=ON

6. Get Servo Pod Ultrasonic Sensor Data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x0A

Response: FBVI to Host
Length: 6 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x0A	0x04	1 byte data

4.11 Servo Pod APIs

Filename:
sensorboard.c

Function:

```
void UpdateServoPos(uint8_t Angle, uint8_t Servo)
```

Description:

This function controls the pan and tilt angle of servo motor.

Input Parameters:

Parameter	Description
`uint8_t Angle`	Specifies angle value in terms of degrees (0 -180).
`uint8_t Servo`	Specifies Pan servo motor ot Tilt servo motor. 1 = PAN 2 = TILT 3 = AUX

Returns: None

Precondition:

uncomment SERVOPOD definition in Hardwareprofile.h

Example:

```
UpdateServoPos(60, 2);  // Update servo pod tilt angle
```

4.12 Miscellaneous Commands

1. Get Robot ID

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0xFA	0x00

Response: FBVI to Host
Length: 5 bytes

Header			Command	Data
'F'	'B'	'D'	0xFA	ID

2. Get Hardware Version

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0xFB	0x00

Response: FBVI to Host
Length: 5 bytes

Header			Command	Data
'F'	'B'	'D'	0xFB	HW_VER

3 Get Software Version ID

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0xFC	0x00

Response: FBVI to Host
Length: 5 bytes

Header			Command	Data
'F'	'B'	'D'	0xFC	SW_VER

4 Get Potentiometer Data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x09

Response: FBVI to Host
Length: 7 bytes

Header			Command	Sub-Command	Data	
'F'	'B'	'D'	0x05	0x09	LSB	MSB

Note: Potentiometer Data is 12-bit and it is split into two bytes. The upper 4 bits of MSB are always zero.

5 Get AD7998 Data

Command: Host to FBVI
Length: 5 bytes

Header			Command	Sub-Command/Data
'N'	'E'	'X'	0x05	0x08

Response: FBVI to Host
Length: 21 bytes

Header			Command	Sub-Command	Data
'F'	'B'	'D'	0x05	0x08	16 bytes of data Byte0 = CH0_LSB & Byte 1 = CH0_MSB and so on.

Note: Refer AD7998 datasheet for more details on data format

5 Motor controller

5.1 Introduction

Motor controller communicates with LPC1769 main controller over UART2 at 115200 bps baud rate. It uses a simple protocol for communication with robot. The robot is the master and motor controller acts as slave and executes the commands sent to it by the robot. The velocity of each motor is restricted by safety mode. In order to use the motors at full velocity, the safety mode needs to be disabled. The safety mode is always on by default.

5.2 Protocol for communication with motor controller

Every command is sent with 3 header bytes, viz. "NEX" (0x4E 0x45 0x58) and then the command followed by data bytes if any. The motor controller will then respond to the command if required. The following table explains various commands and their description.

Command	**Name**	**Bytes sent**	**Bytes returned**	**Description**
0x21	GET VELOCITY 1	4	1	Returns the current requested speed of motor 1
0x22	GET VELOCITY 2	4	1	Returns the current requested speed of motor 2
0x23	GET ENCODER 1	4	4	Returns motor 1 encoder count, 4 bytes returned high byte first (signed)
0x24	GET ENCODER 2	4	4	Returns motor 2 encoder count with MSB first (signed)
0x25	GET ENCODERS	4	8	Returns 8 bytes - encoder1 count, encoder2 count
0x26	GET VOLTS	4	1	Returns the input battery voltage level
0x27	GET CURRENT 1	4	1	Returns the current drawn by motor 1
0x28	GET CURRENT 2	4	1	Returns the current drawn by motor 2
0x29	GET VERSION	4	1	Returns the MD25 software version
0x2A	GET ACCELERATION	4	1	Returns the current acceleration level
0x2B	GET MODE	4	1	Returns the currently selected mode
0x2C	GET VI	4	3	Returns battery volts, motor1 current and then motor2 current
0x31	SET SPEED 1	5	0	Sets new speed for motor 1

0x32	SET SPEED 2	5	0	Sets new speed for motor 2
0x33	SET ACCELERATION	5	0	Sets new acceleration
0x34	SET MODE	5	0	Sets the mode (Mode: 0 = open loop control, 1 = closed loop velocity control, 2 = position control)
0x35	RESET ENCODERS	4	0	Sets both encoder counts to zero
0x40	SET POSITION	13	0	Sets target position for motor 1 and motor 2. Header (3 bytes) + Command (1 byte) + position 1 (4 byte high byte first, signed) + velocity 1 (1 byte) + position 2 (4 byte high byte first, signed) + velocity 2 (1 byte)
0x44	SAFETY MODE	4	0	Set safety mode. (0x01: ON, 0x00: OFF)

Encoder count: The motor controller provides separate encoder counts for two motors with 32 bit signed representation. The command for reading encoder count returns the 4 byte count with high byte first.
Example:
To get encoder count of motor 1, send

Header	Command
0x4E 0x45 0x58	0x23

The motor controller would respond with 4 bytes,
BYTE1 = 0x00; BYTE2 = 0x11; BYTE3 = 0x45; BYTE4 = 0x32;
This can be stored in a 32 bit signed variable as,

```
int32 count = 0;
count += BYTE1<<24;  //(0x00 shifted by 24 bits)
count += BYTE2<<16;  //(0x11 shifted by 16 bits)
count += BYTE3<<8;   //(0x45 shifted by 8 bits)
count += BYTE4;      //(0x32)
```

The received count now equals 0x00114532 hex.

The GET ENCODERS command works exactly in same manner with encoder 1 count followed by encoder 2 count.
The encoder count registers can be reset to zero by sending 0x35 (RESET ENCODERS).

Velocity: The direction of motion is controlled by specifying the velocity of the motor. The velocity is 8 bit value with 0 = Maximum clockwise, 128 = stop, 255 = Maximum anti-clockwise.
Example:

To get speed of motor 1, send

Header	Command
0x4E 0x45 0x58	0x26

The motor controller would respond with, 0x60 (96 decimal)

Acceleration: The acceleration value determines the time taken by motors to reach the specified speed. A value of 0 means slowest and 9 means fastest acceleration.

Position: The motor controller uses 32 bit position encoder. The position values are signed with positive value representing forward motion and negative value representing backward motion. The communication protocol provides commands to read and reset the encoder count of each motor individually. In position control mode, a single command is available to set the target position of each motor. The motor controller uses the absolute value of position specified. The direction of motion is always decided by the velocity.

Mode: Motor controller supports following modes

Mode 0: Open loop velocity control

This is the simplest mode of operation. In this mode, amount of power to be given to the motor can be specified. This is open loop control mode. In this mode, motor has the quickest response but there is no closed loop velocity control and the actual velocity may vary depending on output torque required for movement.

Mode 1: Closed loop velocity control

This is closed loop velocity control mode. In this mode, motor controller will try to increase the power to the motor until motor achieves the specified velocity or it reaches its maximum output power. The motor will maintain its velocity even while climbing and going down the slope.

Mode 2: Position control mode

In this mode, the motor can be controlled to move by a specified number of counts. The distance is specified in terms of encoder counts for each motor. The maximum velocity and acceleration for reaching the desired position can be set. The direction of motion is always decided by the velocity. The number of encoder counts to travel is always taken as absolute value of signed target count specified. The encoder count registers for both the motors are reset and the position counting always starts from zero.

Example:

To move both motors by a count of 0x1017 with velocity 0x50, send

Header	Command	Position 1	Velocity 1	Position 2	Velocity 2
0x4E 0x45 0x58	0x40	0x00 0x00 0x10 0x17	0x50	0x00 0x00 0x10 0x17	0x50

This will reset the 32 bit encoder count registers and motors will start moving till the target encoder count is achieved. The direction of motion is decided by the velocity specified.

Note: In position control mode, every new position command will reset the encoder count registers of both the motors and the target encoder count will be achieved by starting from zero.

Battery Voltage: The voltage of the motor power supply can be read by using this command. The value returned by motor controller is 10 times the actual voltage (119 for 11.9 volts).
Example:
To read the motor voltage, send

Header	Command
0x4E 0x45 0x58	0x26

The motor controller will respond with, 0x77 (119 decimal) -> 11.9v

Current: The current consumed by each motor can be read by GET CURRENT 1 and GET CURRENT 2 commands. The value returned by motor controller is 10 times the actual current (16 for 1.6 A).
Example:
To read the motor 1 current, send

Header	Command
0x4E 0x45 0x58	0x26

The motor controller will respond with, 0x10 (16 decimal) -> 1.6A

Safety Mode: The safety mode restricts the speed of both the motors. The safety mode is always on by default. A command is provided to enable or disable the safety mode.
Example:
To disable safety mode, send

Header	Command	Data
0x4E 0x45 0x58	0x26	0x00

6 Serial LCD

6.1 Introduction

The I2C and serial 20x4 LCD module from Nex robotics can be controlled over I2C or a standard serial port with TTL 5V level signals. This dramatically reduces the number of pins consumed by LCD while interfacing with any microcontroller. It uses only two data lines for communication with any device. A well defined and easy to use protocol ensures smooth and error free operation. 560 bytes of FIFO buffer ensures reliable and fast writing operation on display.
The LCD module is connected to I2C1 port of main microcontroller of the robot. The default I2C address of LCD module is 0xC6. Section 4.9 explains the functions provided in the library for accessing serial LCD on the robot.

6.2 Protocol for communication with LCD

The following table explains various commands and their description.

Command	Name	Description
0x00	null (ignored)	Ignored as a no operation
0x01	Cursor Home	Sets the cursor to the home position (top left)
0x02	Set cursor (1-80 or 32)	Cursor to a position specified by the next byte, where 1 is the top left and 80/32 is the bottom right
0x03	set cursor (line, column)	Sets cursor using two bytes, where first byte is the line and the second byte is the column
0x04	Hide cursor	stops the position cursor from appearing on the display
0x05	Show underline cursor	Changes the cursor to the underline type
0x06	Show blinking cursor	Changes the cursor to the blinking type
0x08	Backspace	deletes the preceding character from the current position on the display
0x09	Horizontal tab (by tab set)	Moves the current position across by the tab space set by command 18 (default tab space 4)
0x0A	Smart line feed	Moves the cursor down one line to the position beneath in the same column
0x0B	Vertical tab	Moves the cursor up one line to the position above in the same column
0x0C	Clear screen	Clears the screen and sets cursor to the home position
0x0D	Carriage Return	Moves the cursor to the start of the next line
0x0F	Software version	Module returns a single byte software version
0x11	Clear Column	Clears the contents of the current column and moves

		cursor right by one column
0x12	Tab set	Sets the required tab size, the following byte can be a size of between 1 and 10
0x13	Backlight on	Turns the backlight of the LCD03 on
0x14	Backlight off (default)	Turns the backlight of the LCD03 off
0x15	Disable startup message	Disables the display of setup information at power up
0x16	Enable startup message	Enables the display of setup information at power up
0x19	Change Address	Change the I2C bus address of the LCD
0x20-0xFF	ASCII characters	Writes ASCII characters straight to the display

Example:
To switch the backlight ON, send 0x13.
To set cursor at line 2 and column 4, send 0x03 0x02 0x04.
To display 'A' character at present cursor location, send 0x41 (hex representation for 65).
To clear the display, send 0x0C.

7 Introduction to FireBird VI GUI

Fire Bird VI robot can be controlled by GUI via UART3 RS232 port. To control the robot using GUI via appropriate mode of communication, load appropriate hex file on the robot. The baud rate used by GUI for RS232 serial communication is 115200 bps.

7.1 Installing FireBird VI GUI

Step1: Copy “FIRE BIRD VI setup” folder which is located inside the folder “GUI and Related Firmware” from the documentation CD on the PC.

Click on “setup.exe” which is located in the “FIRE BIRD VI setup” folder.

Step 2: Click Next Button to continue.

Step 3: Browse the location where set up will install or set the default location and click Next Button to start the installation.

Step 4: When installation is successfully completed, click Close to exit.

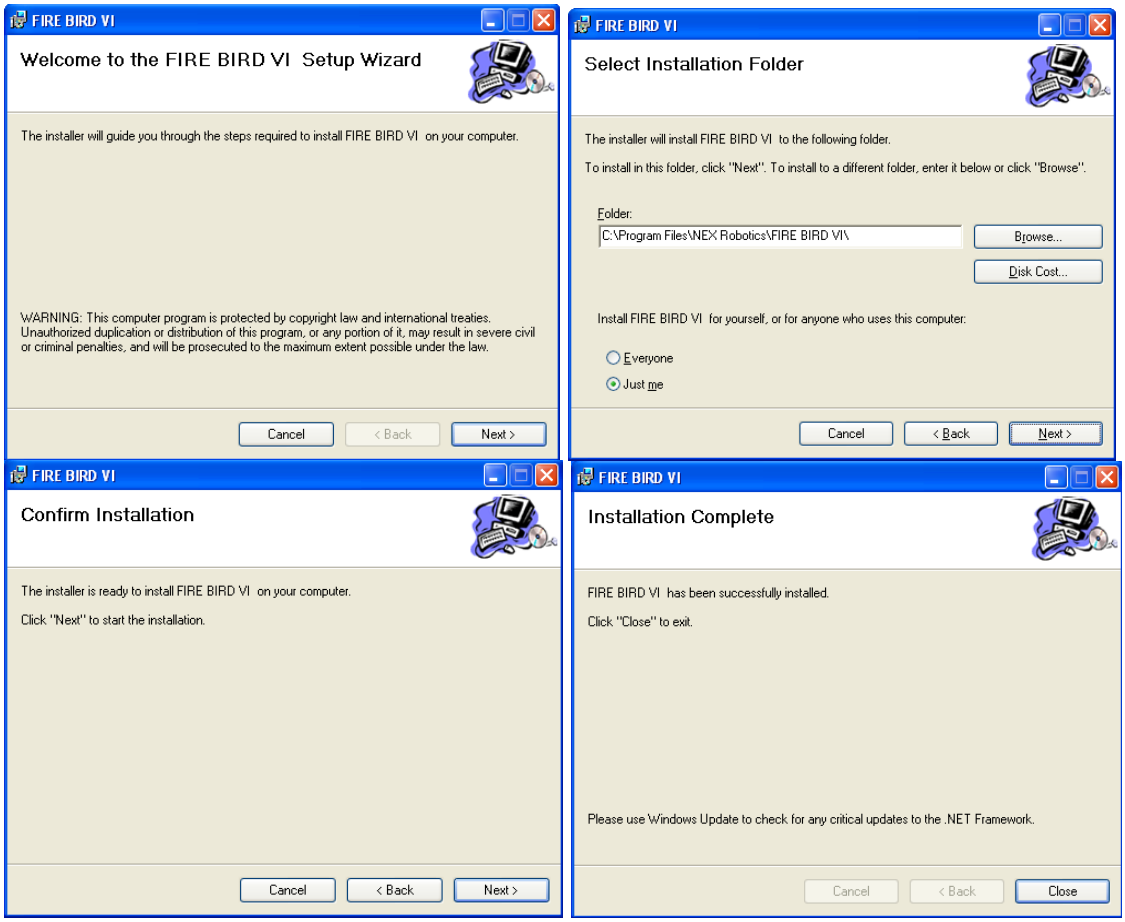


Figure 5.1 Installation procedure for GUI

7.2 Installing USB to RS232 Drivers

If the PC or laptop does feature RS232 port, you can still communicate with robot using USB to RS232 serial converter. The RS232 to serial converter can be obtained from NEX Robotics website.

To install the drivers, refer USB to serial converter product manual. The drivers and product manual and drivers can be obtained from NEX Robotics website.

Figure 5.2 USB to Serial Converter

7.3 Using GUI

Step 1: After successful installation go to Start -> All Programs -> FIRE BIRD VI -> FIRE BIRD VI or click on Fire Bird VI icon on your desktop to open the GUI.

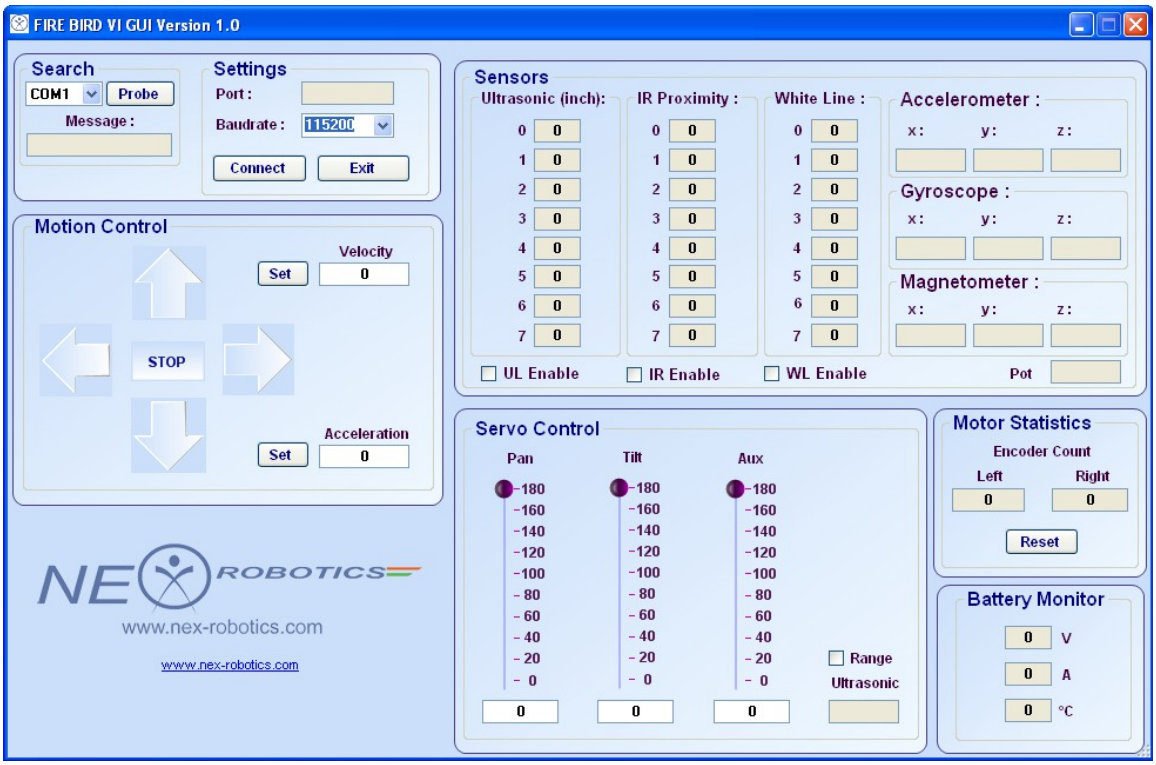


Figure 5.3: Selecting correct COM port

Step 2: Connect Robot with the PC using appropriate cable.

Step 3: Select baud rate as 115200. The GUI will prepare a list of available COM ports. Select the appropriate comport from dropdown box and click probe. If communication link is successfully established between robot and PC, the GUI should display Fire Bird VI in the message box in search panel. If link is not established recheck com port and probe again

Step 4: After finding connected modules on COM port, click on "Connect" button to connect to the Fire Bird VI robot. This will show the data from each sensor in GUI. The robots movements can be controlled using "Motion Control" panel on the GUI.

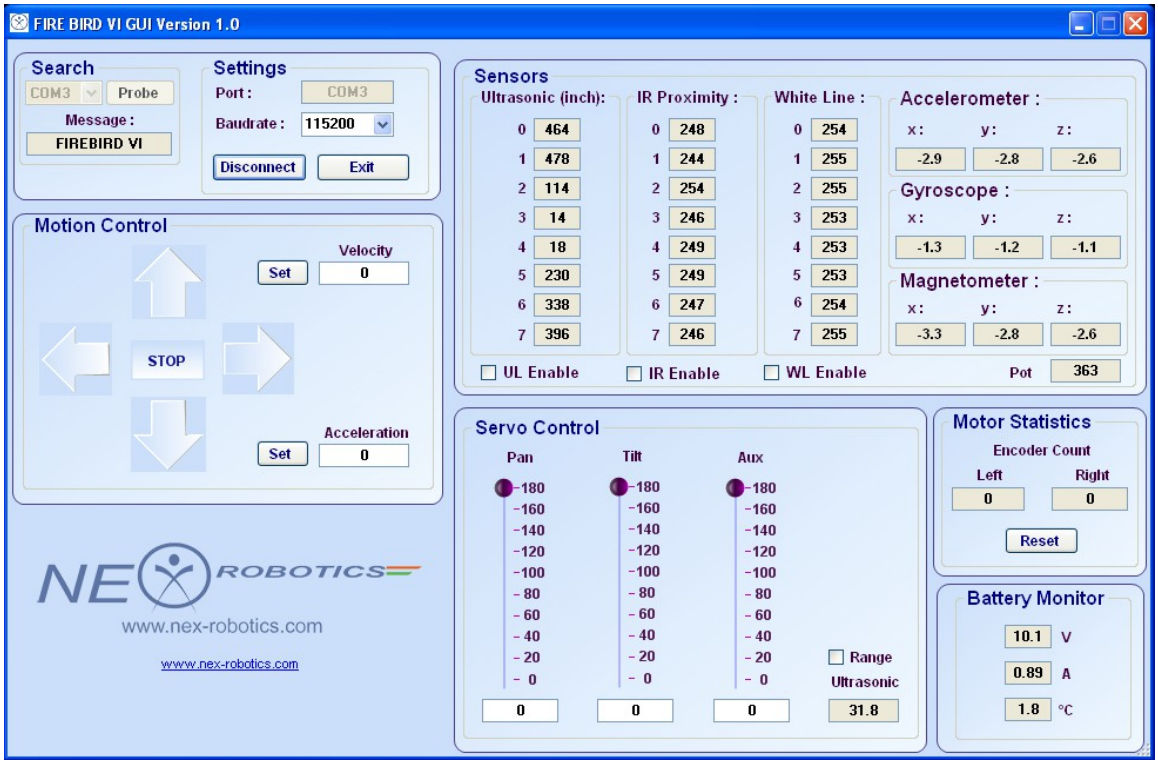


Figure 5.4: GUI showing robot's data

Panels on GUI:

1) **Search and Settings**: Search panel allows user to find available COM ports on the PC. Select any COM port and press "Probe" button to identify if Fire bird VI is connected to that port. For example, on probing COM5, if robot is connected to that particular comport, then the textbox in the Settings panel will show COM5 and textbox in search panel will show "Fire Bird VI". The "Connect" and "Disconnect" buttons can be used for connecting and disconnecting from Fire Bird VI.

2) **Sensors**: This panel shows the data received from Ultrasonic, IR proximity, White line sensor, accelerometer, gyroscope, magnetometer and potentiometer.

3) **Motion Control:** This panel provides the facility to control robots motions by setting Velocity and Acceleration.

4) **Battery Monitor:** It shows the battery parameters such as battery Voltage, Current and Temperature.

5) **Motor Statistics:** This panel shows motors encoder count for each motor. This panel also provides the facility to reset encoder count.

6) **Servo Control:** Using Pan, Tilt and Aux bars you can control the movements of connected servo motor.

8 Bluetooth Communication with Fire Bird VI

8.1 Introduction

Fire Bird VI features an on board Bluetooth adapter module to communicate with Bluetooth devices such as USB Bluetooth dongles connected to a PC. The Bluetooth module on Fire Bird VI is connected to the UART of the main microcontroller. The PC side Bluetooth module also uses serial port profile and creates two comports each for incoming and outgoing connection. The Bluetooth communication by default takes place at 9600 bps.

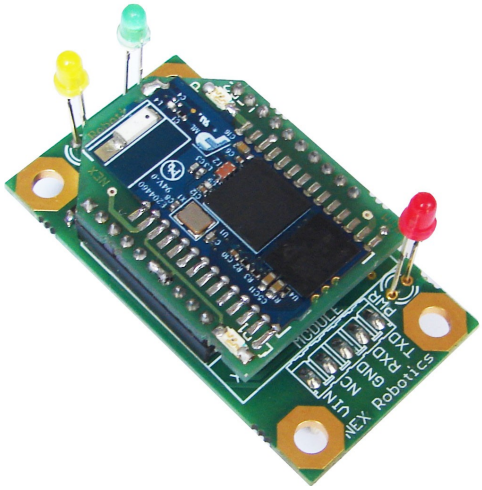


Figure 6.1: Bluetooth adapter module on Fire Bird VI

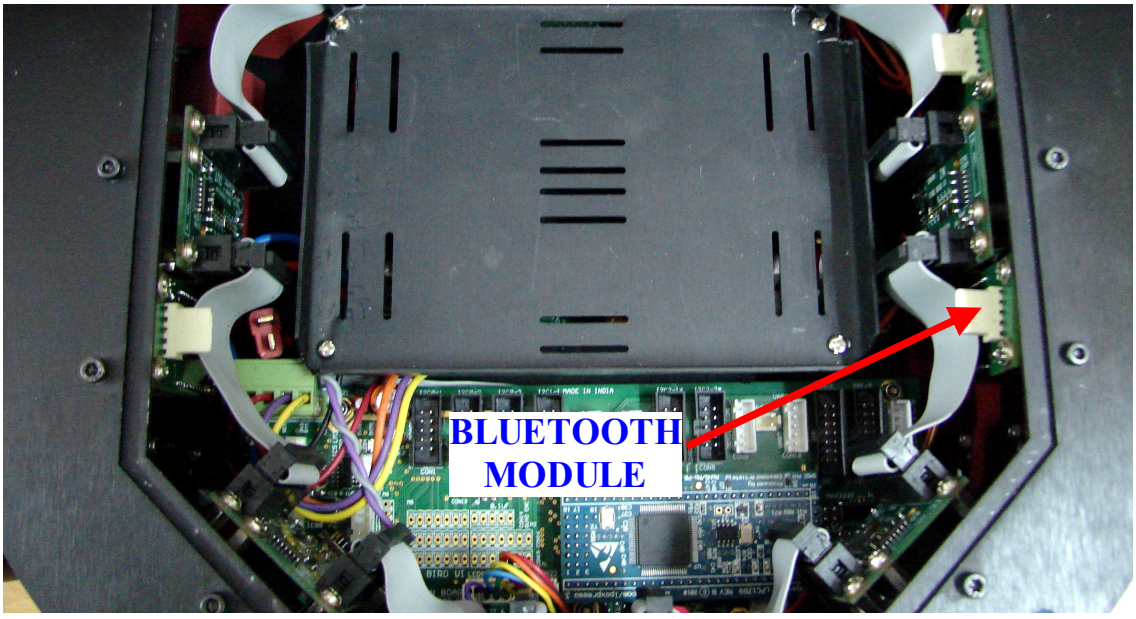


Figure 6.2: Location of Bluetooth module on Fire Bird VI

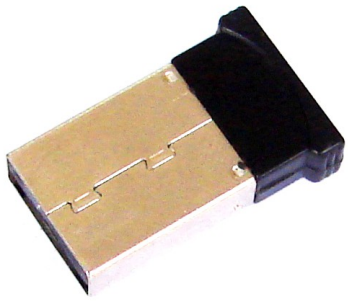


Figure 6.3: Bluetooth dongle for PC

8.2 Hardware Connections on Fire Bird VI

Connect Bluetooth adapter module connector to UART 1 connector on the main board as shown in fig. below.

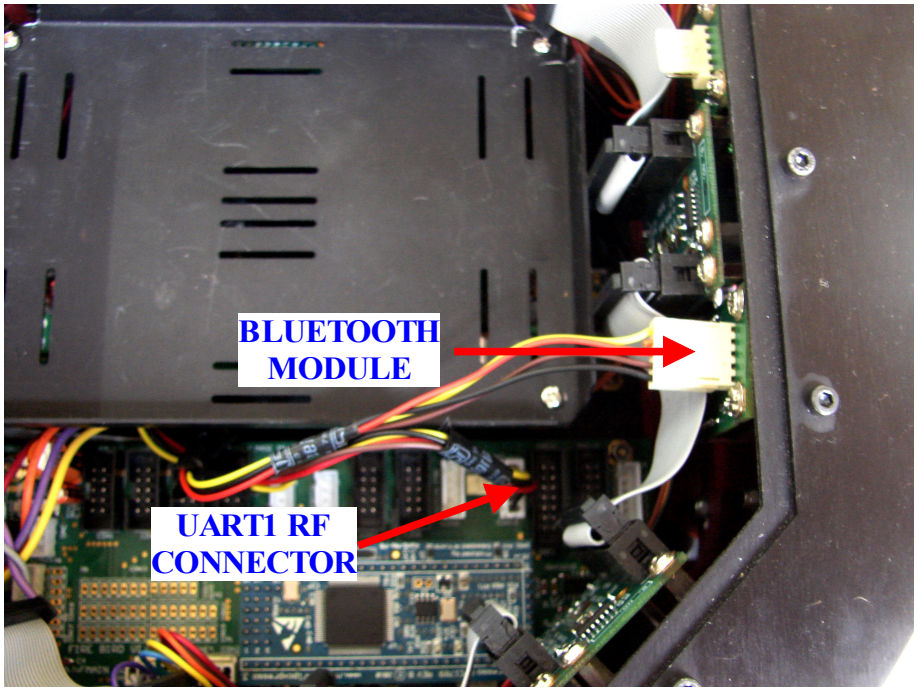


Figure 6.4: Bluetooth Module connection of Fire Bird VI

8.3 Bluetooth Module Installation on PC

1. Connect USB Bluetooth dongle to PC and let windows install the generic drivers for the Bluetooth module.

2. If driver installation is completed, go to windows control panel and click on Bluetooth icon

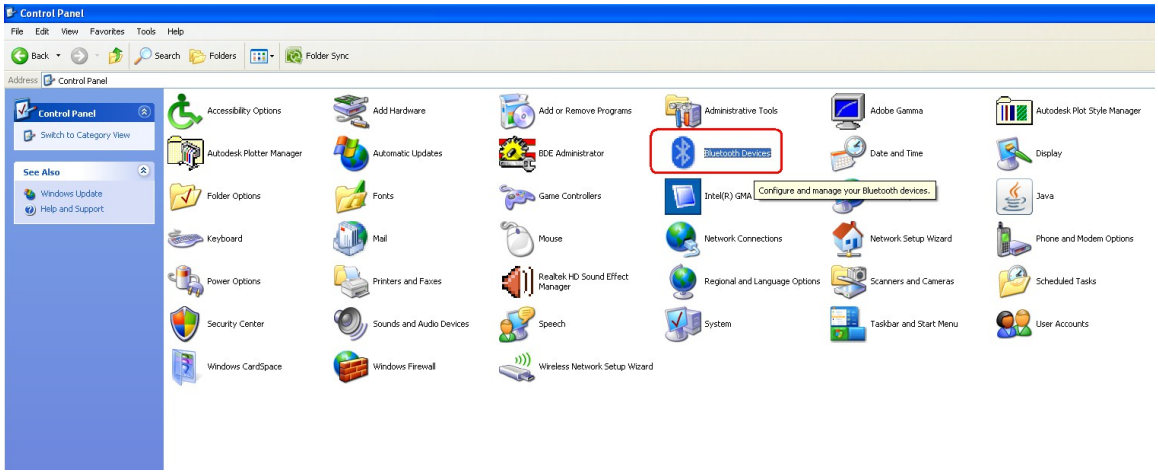


Figure 6.5: Windows Control Panel

3. In the Bluetooth devices windows click **Add** to add new devices.

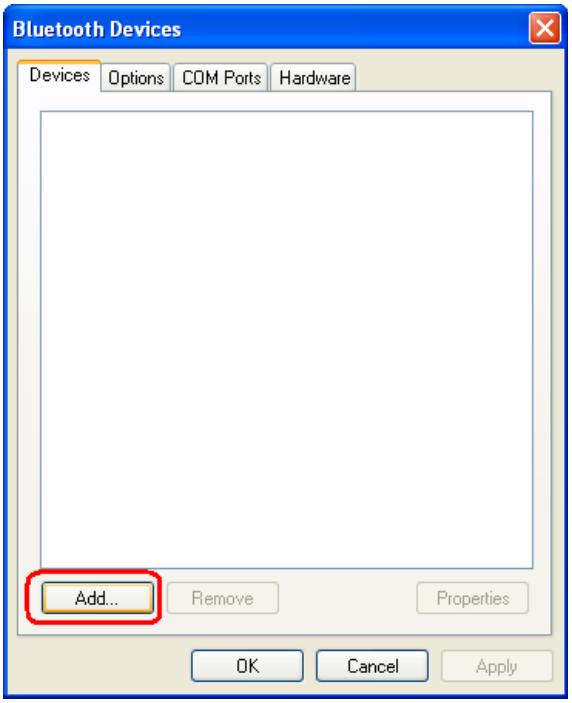


Figure 6.6: Bluetooth devices window

4. In the following window, select the check box and before clicking next ensure that Fire Bird VI is turned ON and Bluetooth module is connected the main board.

Figure 6.7: Add Bluetooth device wizard

5. On clicking next windows will start searching the Bluetooth devices. It should list **AUBTM-20** as a new device on finishing the search. If this device is not listed try searching again and ensure that robot is turned ON and Bluetooth module is connected to Fire Bird VI. If the device is listed click next to continue.

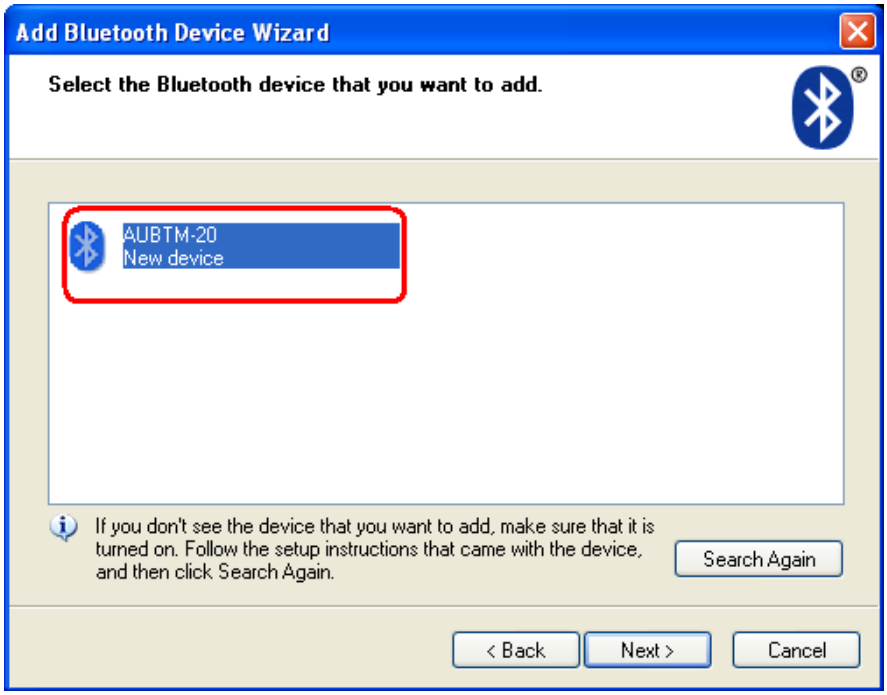


Figure 6.8: Selecting Bluetooth device

6. In the following window use pass key as **8888** and click next to continue.

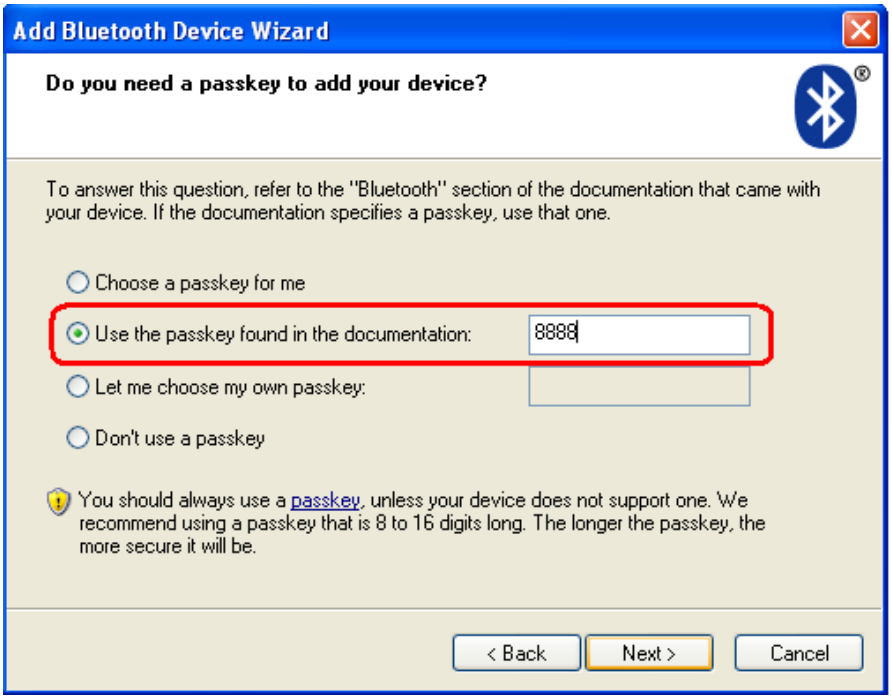


Figure 6.9: Paring with Bluetooth device

7. On click next windows will pair with device and install incoming and outgoing ports for communication.

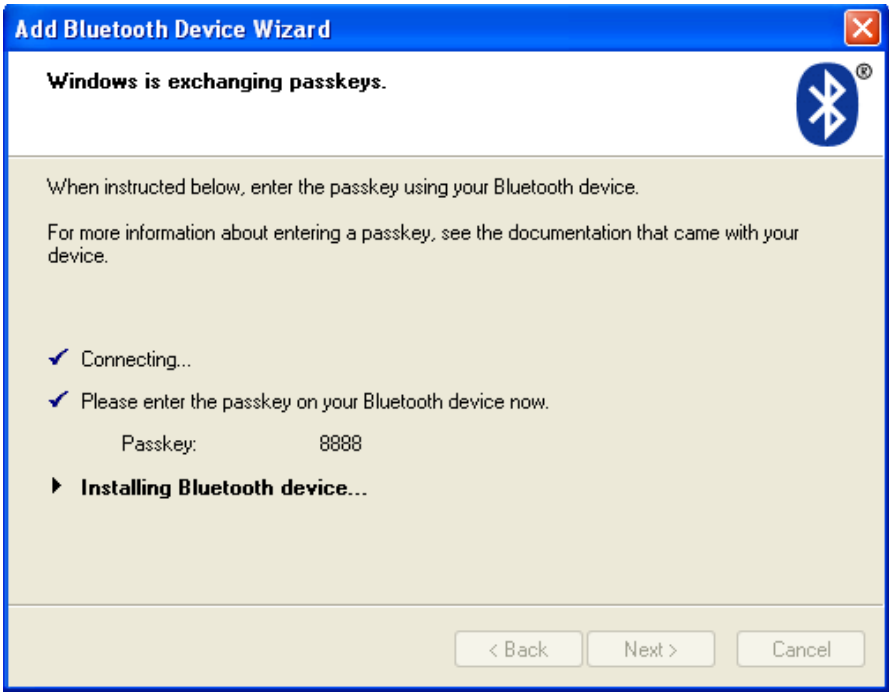


Figure 6.10: Installing Bluetooth Communication ports

8. On successful installation, windows should list the number of incoming and outgoing ports. Keep note of the outgoing port number as it will be used to make communication with the robot using the GUI software. Click Finish to continue.

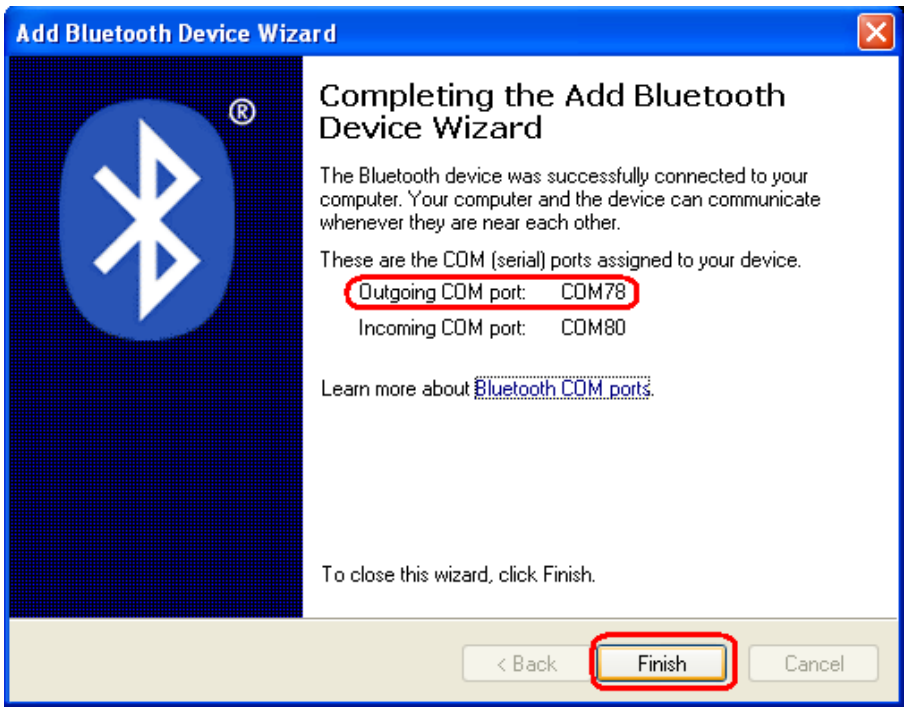


Figure 6.11: Completing Bluetooth Device installation

9. Click OK to close Bluetooth devices window.

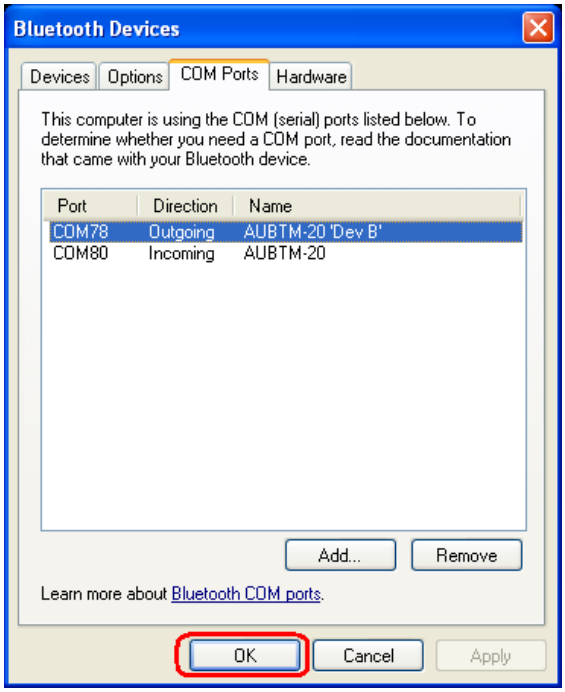


Figure 6.12: Bluetooth Devices window

8.4 Bluetooth Communication using GUI

1. Start Fire Bird VI GUI software. Ensure that Bluetooth module is connected to the UART1 connector in the main board and turn ON the robot.

2. In the GUI software select baud rate as **9600** and from the drop down box select outgoing COM port and click on Probe.

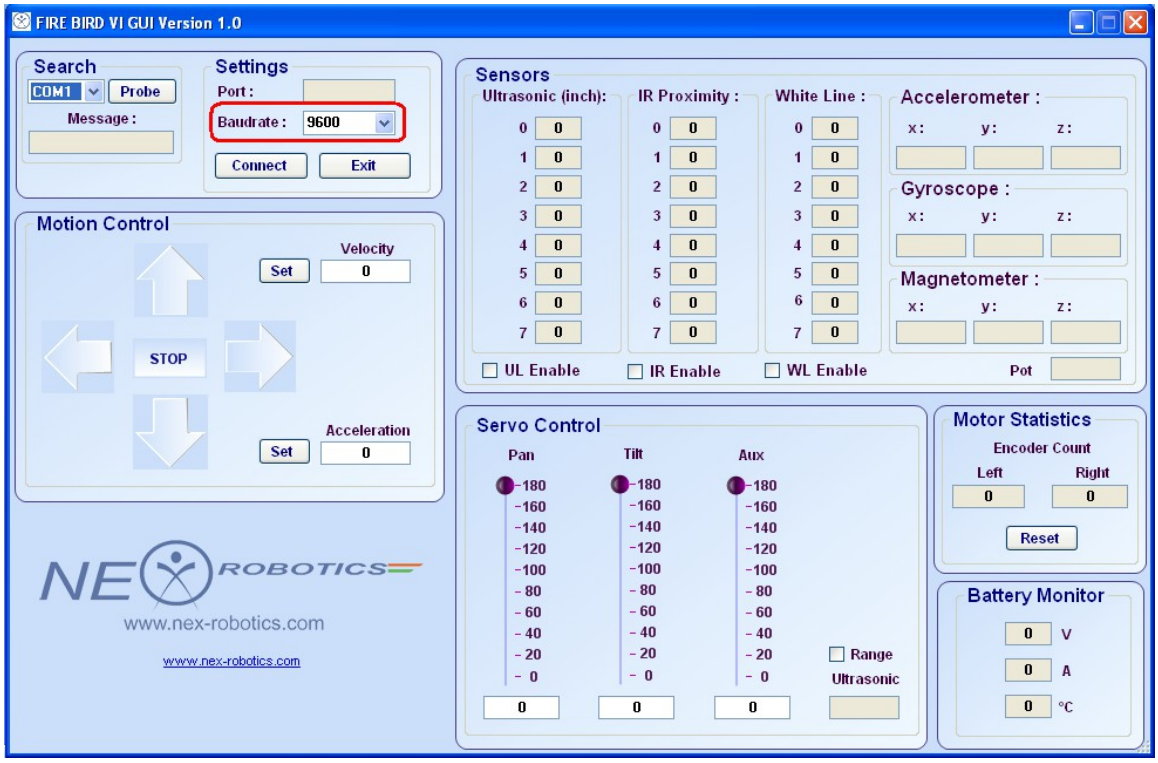


Figure 6.13: Setting baud rate for Bluetooth Communication

3. If Bluetooth link is successfully established between robot and PC, the GUI should display Fire Bird VI in the message box. Click on connect button to start the communication. The Link indicator LED on the Bluetooth adapter module on Fire Bird VI will glow continuously during communication process and it will start blinking if communication is disconnected.

9 WiFi Communication with Fire Bird VI

9.1 Introduction

Fire Bird VI features an on board Wifi adapter module to communicate with Wifi enabled devices such as PCs or mobile phones. The Wifi module on Fire Bird VI sends and receives serial data over UART. The wifi module can be configured to operate in infrastructure or adhoc mode. It supports most of the communication protocols which includes UDP, HTTP, DCP, FTP, DHCP, ICMP, ARP, etc. It also supports wifi security encryption protocols like WPA, WPA-2, and WEP. For more details refer the WiFly wifi module documentation from the CD.

For the demo application, wifi module on Fire Bird VI is configured in ADHOC mode. The configuration steps are explained in following sections.

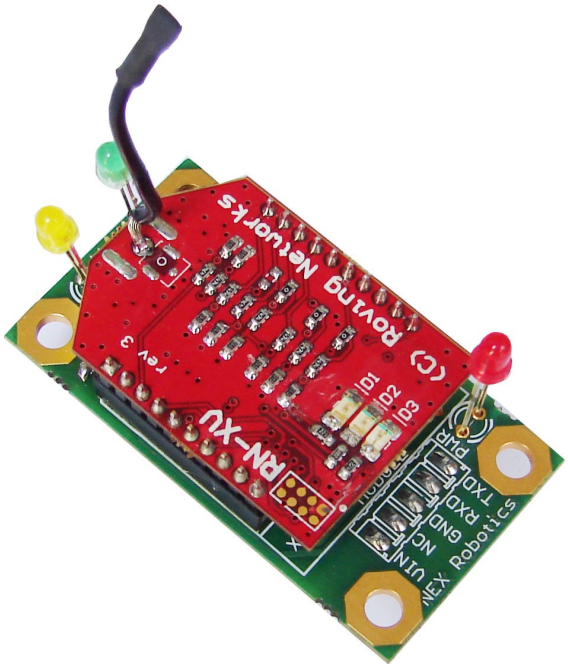


Figure 7.1: Wifi module on Fire Bird VI

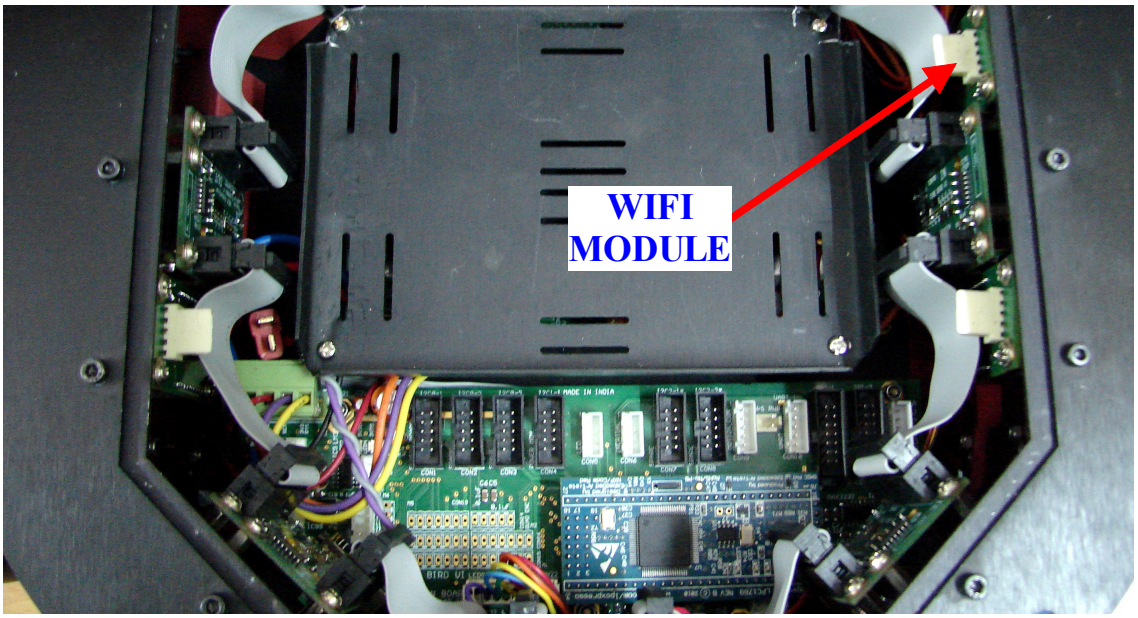


Figure 7.2: Location of Wifi module on Fire Bird VI

Figure 7.3: Wifi module for PC

9.2 Hardware Connections on Fire Bird VI

Connect wifi module cable to UART1 connector on the main board as shown in the fig. below.

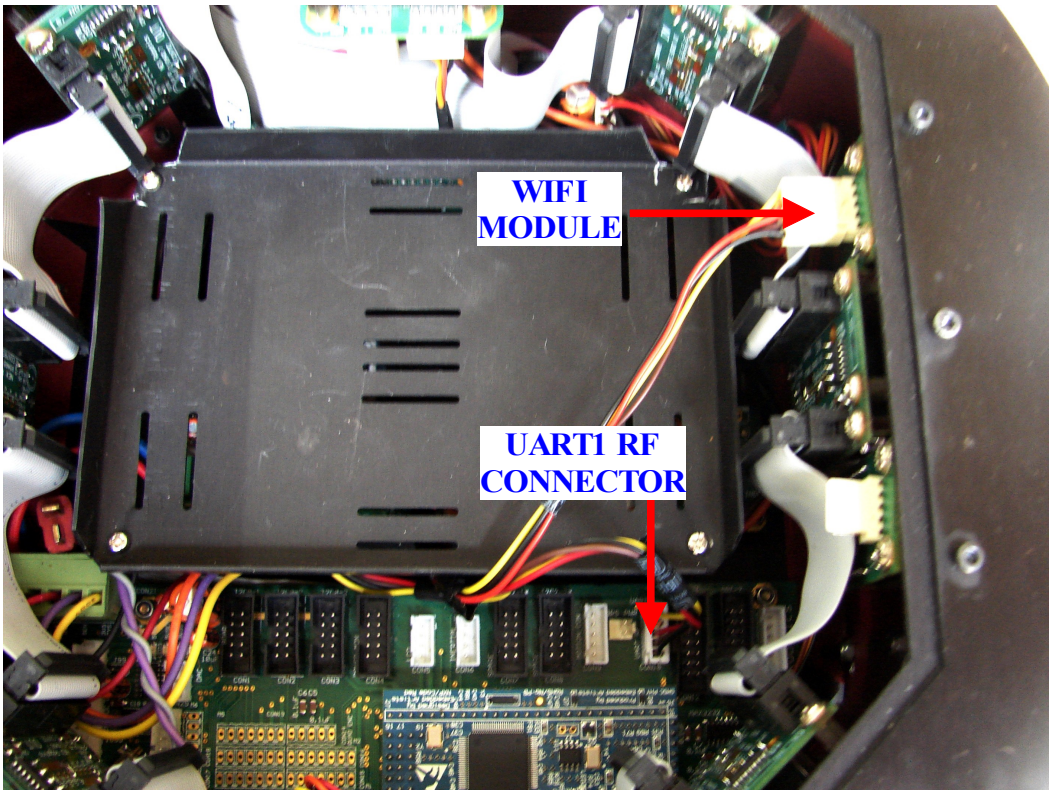


Figure 7.4: Wifi Module connection on Fire Bird VI

The wifi module on Fire Bird VI can be connected to a router or it can be configured to operate in ADHOC mode. In case if you separately purchase the wifi module or if you want to configure other parameters of the module you may use the following steps as reference. Otherwise you may directly skip to section 7.6 : Wifi Communication with robot.

9.3 Configuring Tera Term

As wifi module sends and receives data over serial port, its configuration can also be done over serial port by using any terminal software and USB to serial adapter compatible with wifi module. The USB to serial adapter can be connected to WiFi module on robot as shown below.

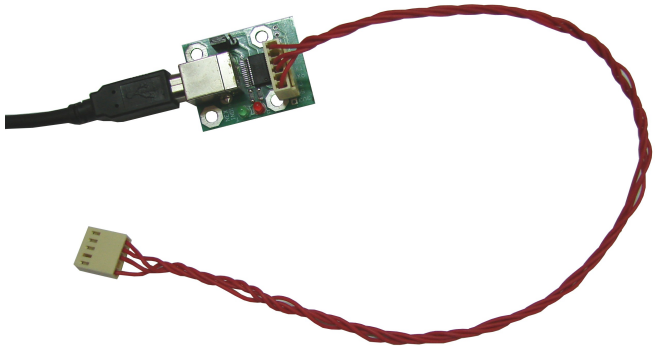


Figure 7.5: USB to Serial adapter

The Tera Term can be used for communicating and controlling the WiFi module. The WiFi module is configured to operate at 115200 bps. Steps for configuring the Tera Term are given below.

1. Click Setup->Serial Port on Tera Term.

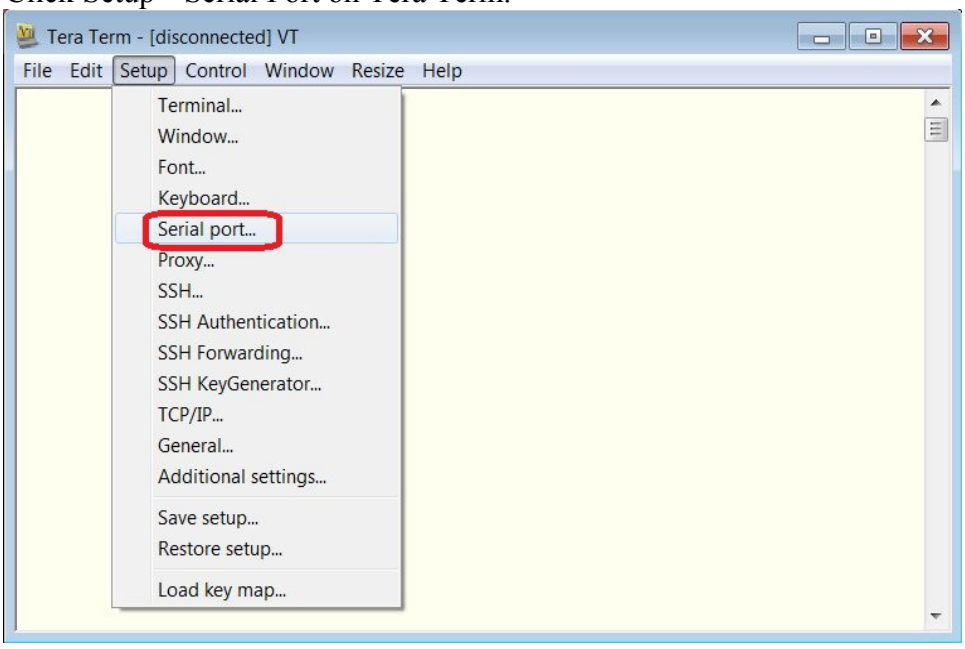


Figure 7.6: Serial Port setup

2. This will open a Serial port setup window. Select Baud Rate of “115200” and click OK.

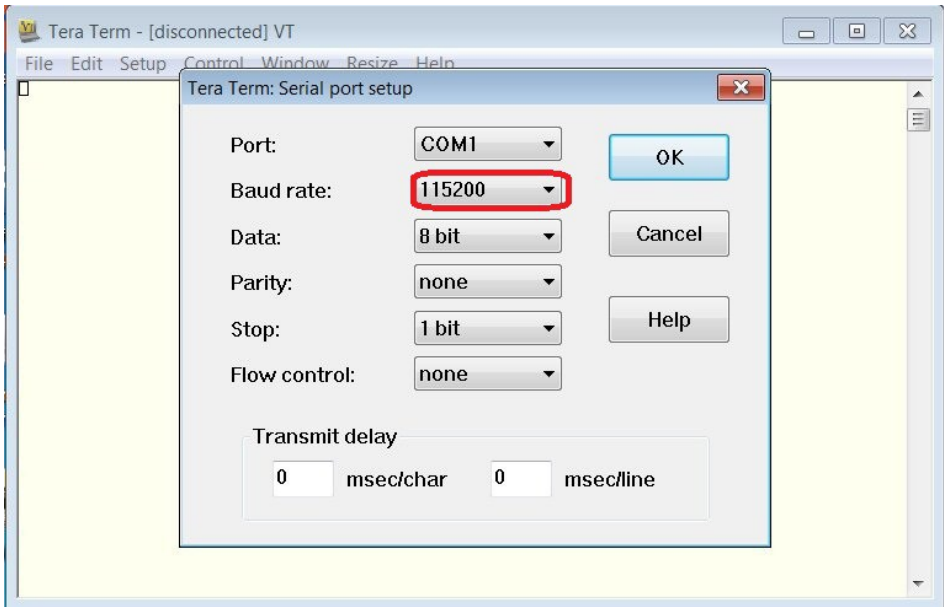


Figure 7.7: Baud Rate selection

3. This configuration must be saved. Click Setup->Save Setup to save the configuration.

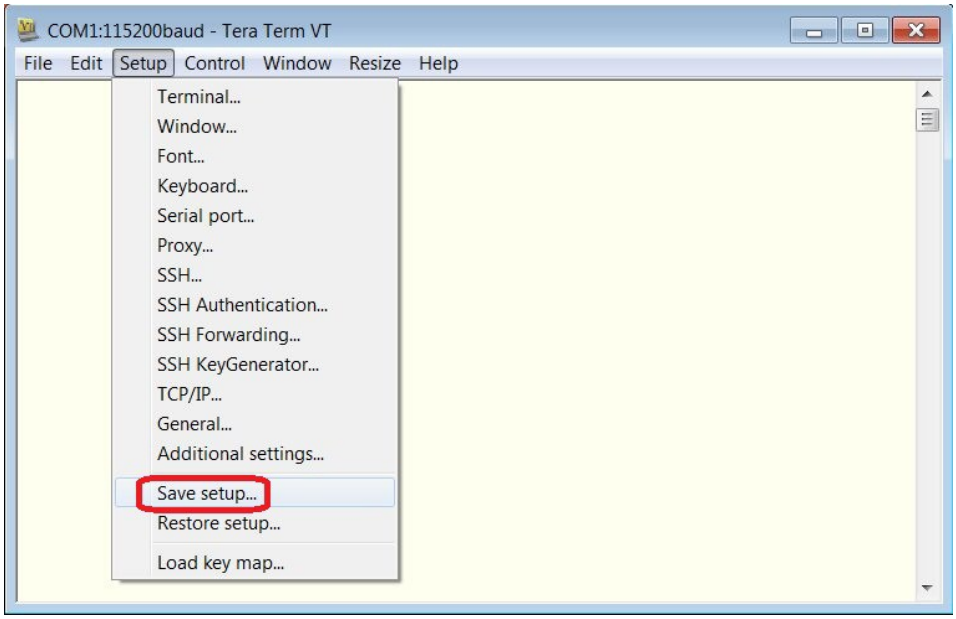


Figure 7.8: Save setup

4. The configuration must be saved to a TERATERM.INI file present in location "C:\Program Files/teraterm". This will ensure that the same configuration will be loaded at next startup.

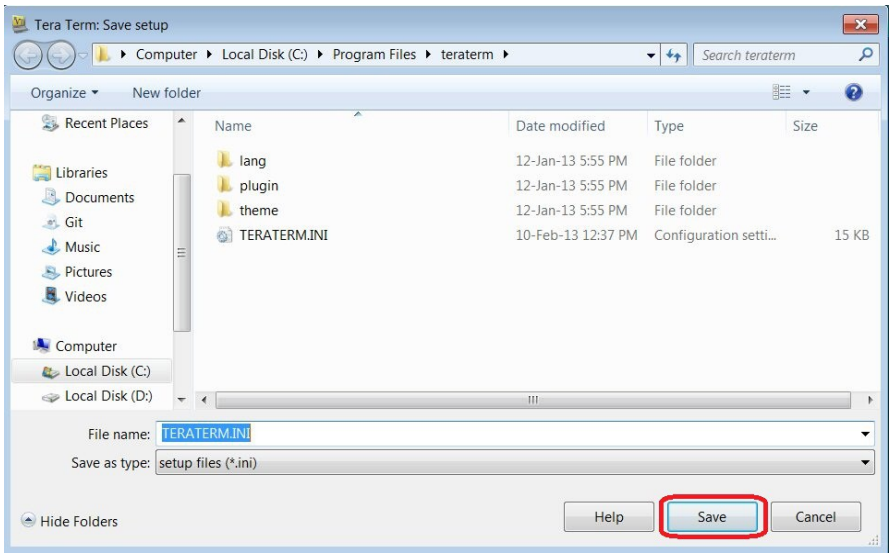


Figure 7.9: Save to default configuration file

9.4 Connecting to a router

The steps for connecting WiFi module on robot to a WiFi router are as follows:

1. Start the WiFi router. Make sure that the authentication type is set to WPA2-PSK. Note down the SSID and passphrase. This information will be used while configuring the WiFi module on robot.
2. Open Tera Term software from start menu or desktop shortcut. A new connection window will pop up.

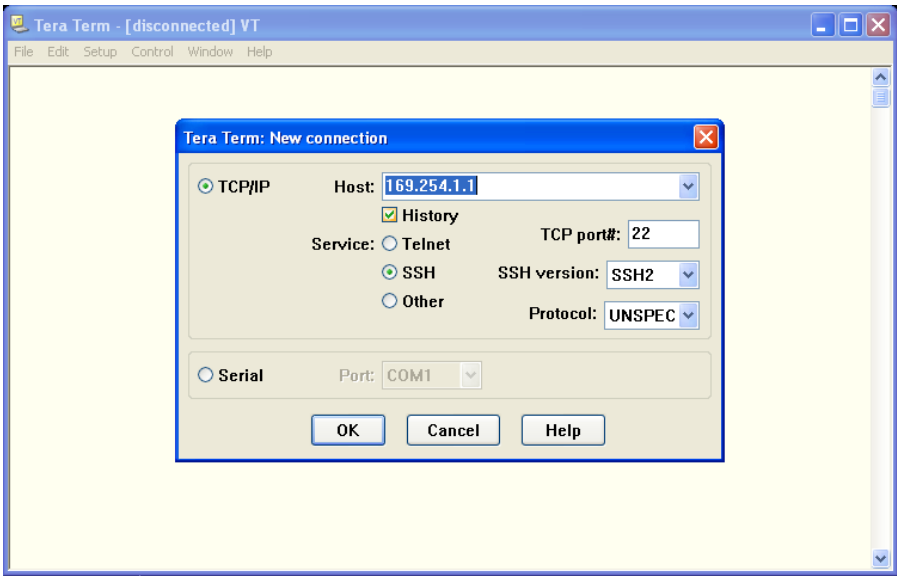


Figure 7.6: Tera Term Software

3. Click on **Serial** radio button and select the USB to serial adapter COM port from the drop down box and click **OK** to continue.

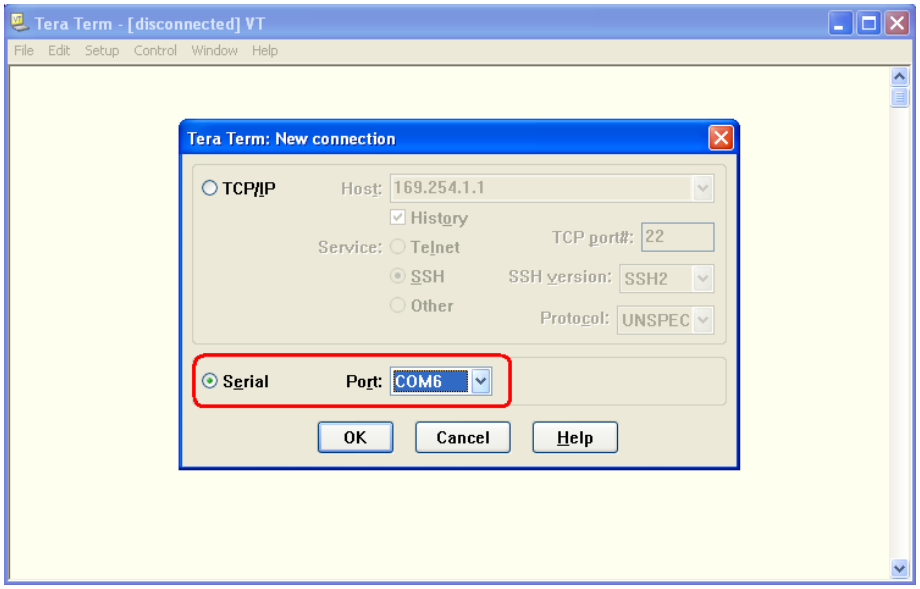


Figure 7.7: Tera Term New Connection

4. Tera term will open the selected COM port. Make sure that the baud rate is set to 115200 bps, as shown in section 7.2. The WiFi module is configured to operate at 115200 bps.

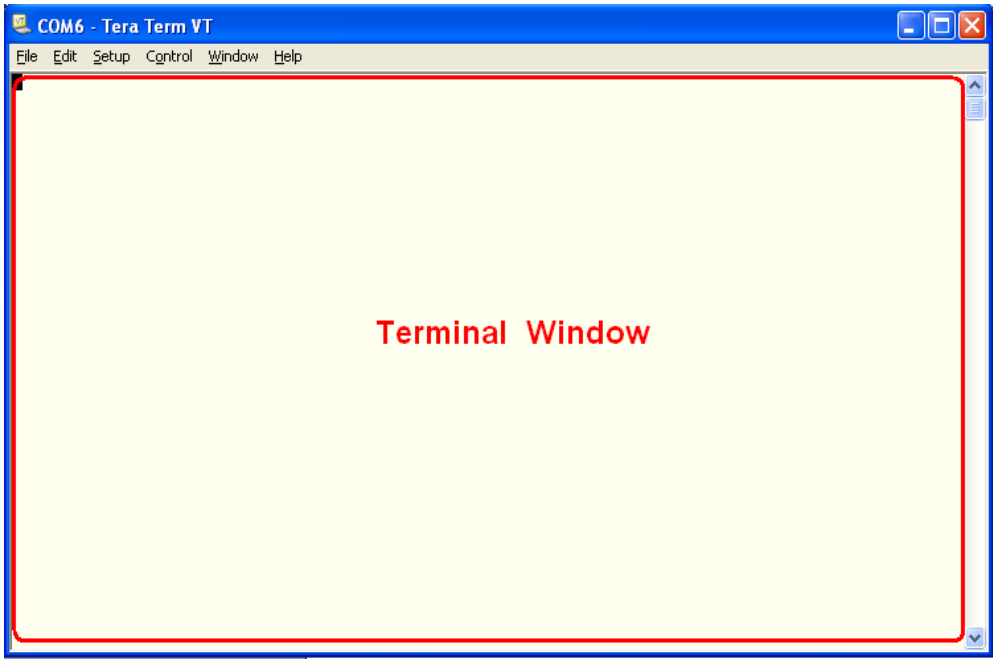


Figure 7.8: Tera Term Terminal window

5. The WiFi module can be configured only when it is in command mode. To enter the command mode, type **$$$** in terminal window. The module should respond with CMD indicating that it is in command mode.

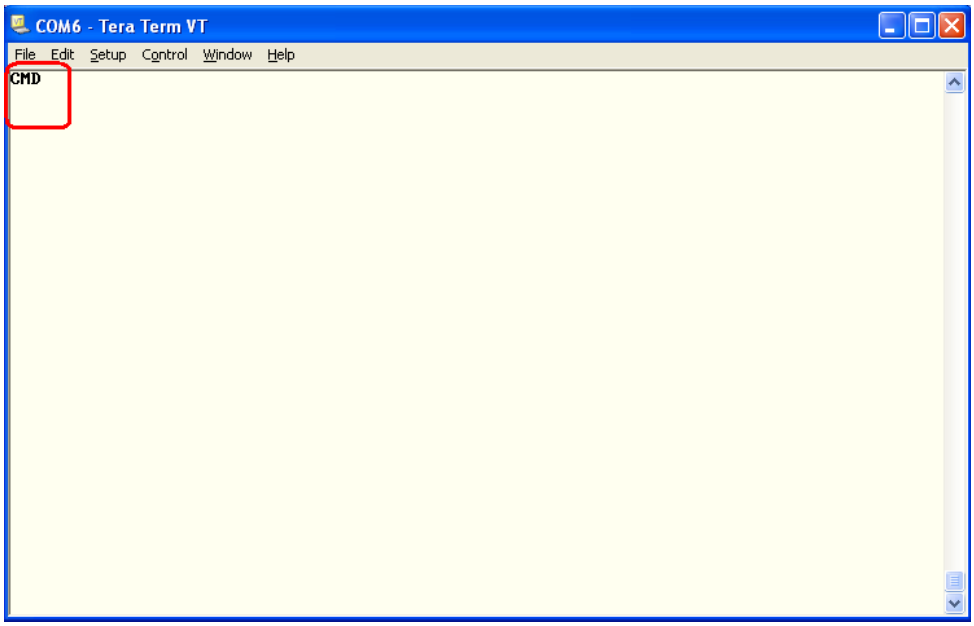


Figure 7.9: Entering Command Mode

6. To connect to WiFi router, type following commands

show net *// display existing configuration*
set w s <SSID> *// set network SSID*
set w p <pass phrase> *// set pass phrase to be used for connection*
save *// save network ssid for join on powerup*
reboot *// reboot with new configuration*

Note: The IP address asigned by router can be found by sending "*get ip*" command over serial port to the WiFi module.

9.5 WiFi Communication with Robot using router

After configuring the WiFi module to communicate with router, the WiFi module will connect to router on power-up. The firmware required to be loaded can be found on documentation DVD in XBEE_GUI_DEMO sample code present in "FireBird VI Examples.zip" file in Experiments folder. To communicate and send basic commands to the robot, we can use Tera Term MACRO feature. This is required because, some of the commands can not be given using keyboard as they involve hexadecimal values that are not mapped to ACII characters present on keyboard. A MACRO is basically a set of commands that are executed when we call it. A tera term macro file has an extension of .ttl. In this case, it mainly contains 'send' command to send the hexadecimal data required for sending a particular command to the robot. For example, a macro file with command "*send 'NEX' $94 $01*" will send the forward command when executed. Sample macro files for moving the robot are given on documentation DVD in Softwares and Drivers\teratermMacros folder. To execute a macro,

1. Connect to the robot using tera term. Start tera term. In New connection window, click on TCPIP radio button and select Telnet service. Enter Host IP same as that assigned by router and TCP port# as **2000** and click OK.

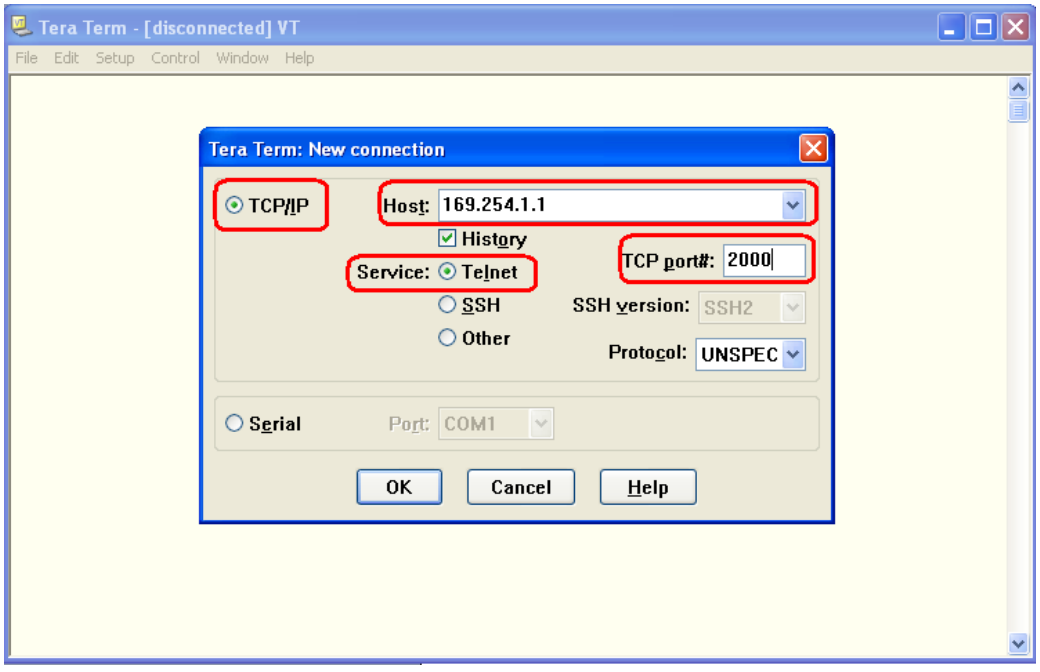


Figure 7.10: Tera Term Telnet Setup

2. Tera Term will create a telnet connection and the wireless module on Fire Bird VI should respond with ***Hello*.**

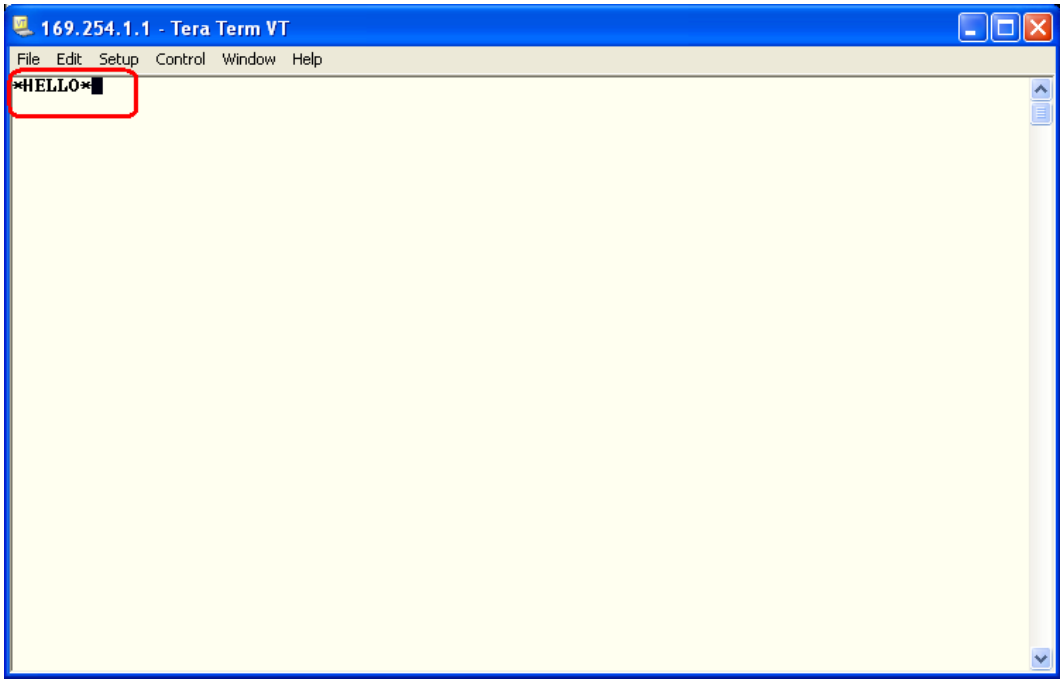


Figure 7.18: Tera Term Telnet Setup

3. Go to Setup-> Terminal from menu bar.

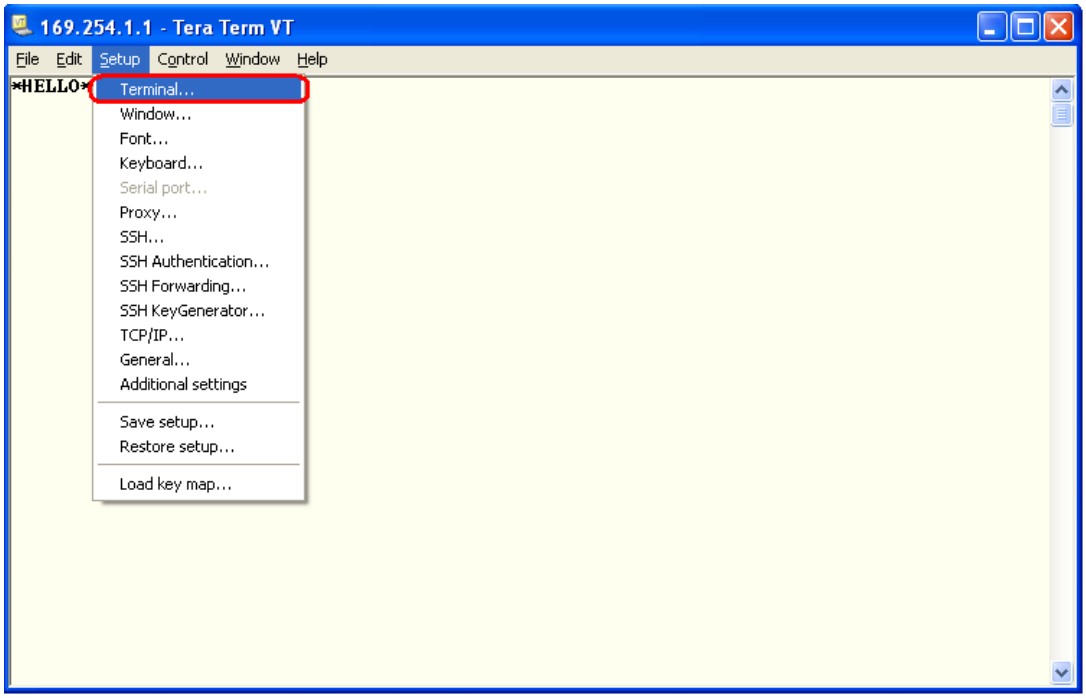


Figure 7.19: Tera Term Terminal Setup

4. In the terminal setup window, select local echo check box. This will display commands typed in the terminal widow. Click OK to close the setup window.

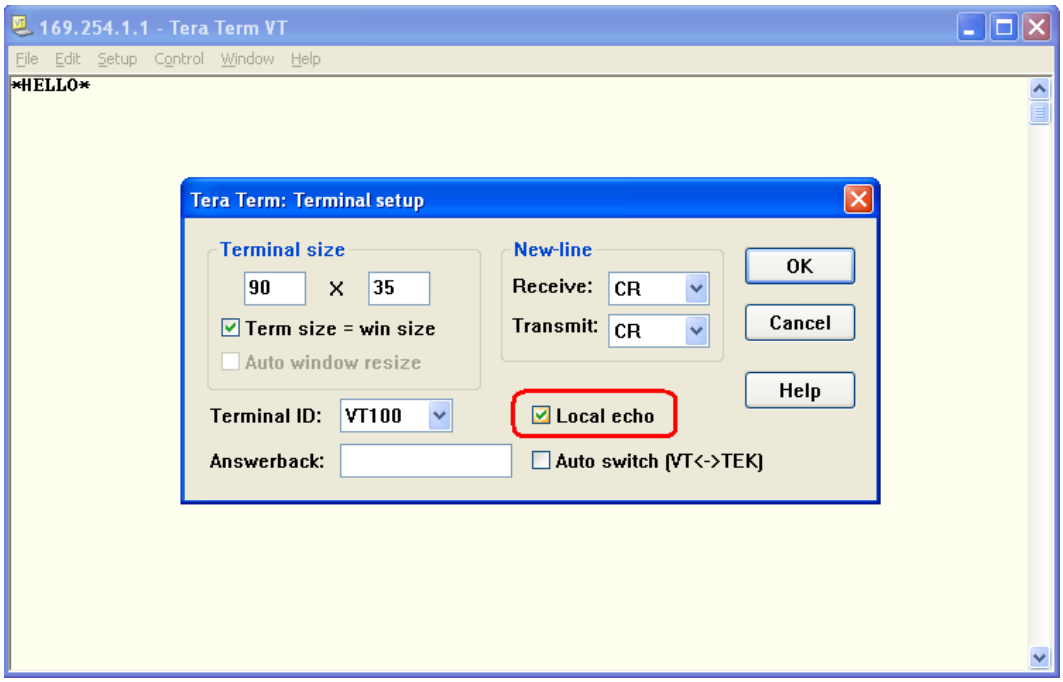


Figure 7.20: Tera Term Terminal Setup

5. **Important: Change the language to English by going to Setup → General. If the language is not selected as English, the binary data will not be sent correctly by Tera Term.**
6. To send a command using MACRO, go to Control → Macro

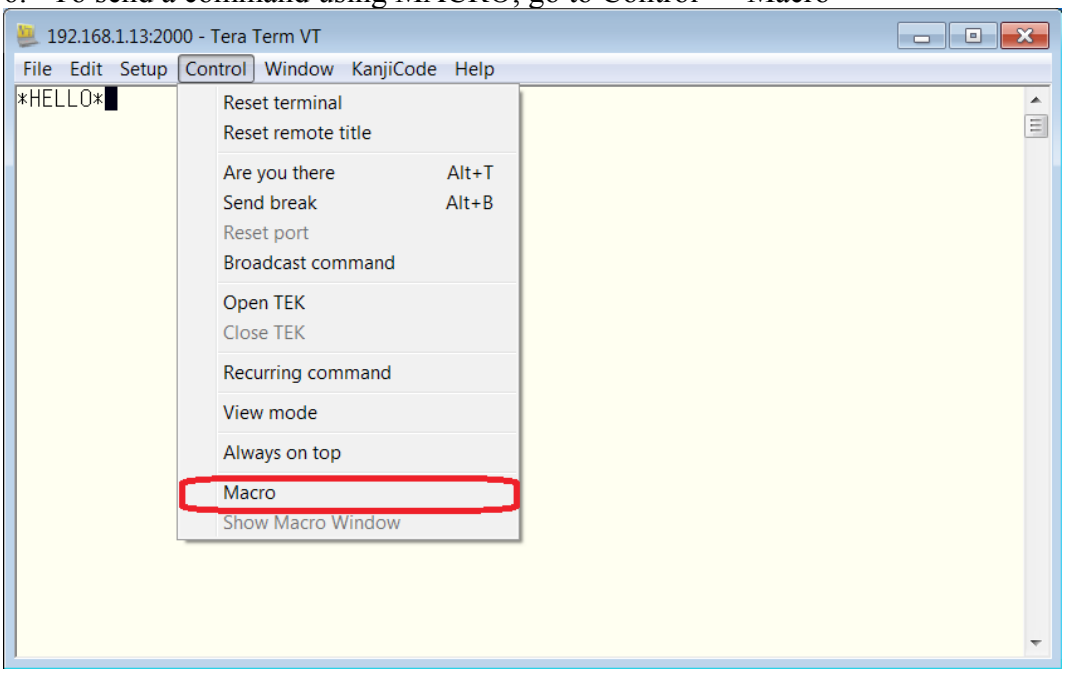


Figure : Open Macro dialog

7. Select the macro, and click Open and press Enter.

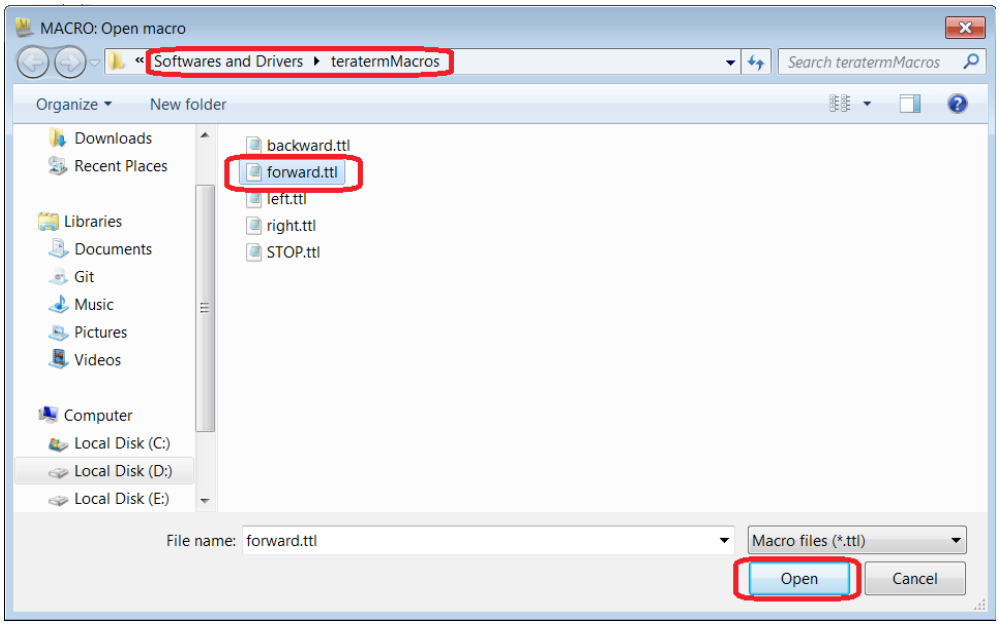


Figure : Open a Macro

8. This will send the command written in macro to the robot. You can write your own macro with set of commands defined for communication with robot.
9. To close the connection, go to File-> Disconnect.

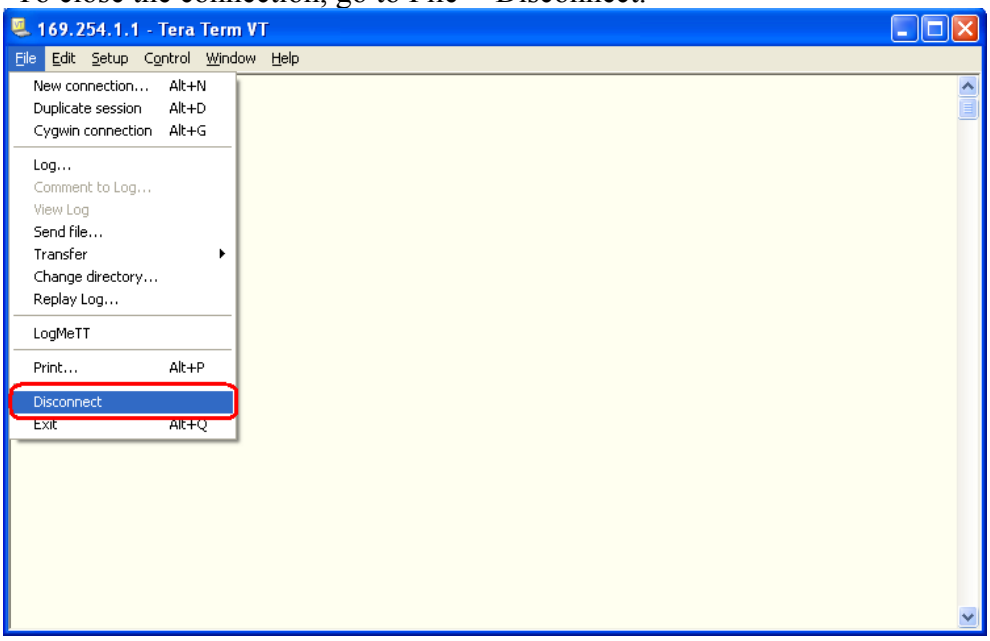


Figure 7.22: Closing Telnet connection

9.6 Configuring ADHOC mode

The ADHOC configuration steps are as follows:

1. Install drivers of USB to serial adapter by referring its manual. Disconnect USB cable from PC and mount wifi module on USB to serial adapter. Reconnect USB cable to PC.

2. Install Tera Term software from the CD.

3. Open Tera Term software from start menu or desktop shortcut. A new connection window will pop up with terminal window in the background

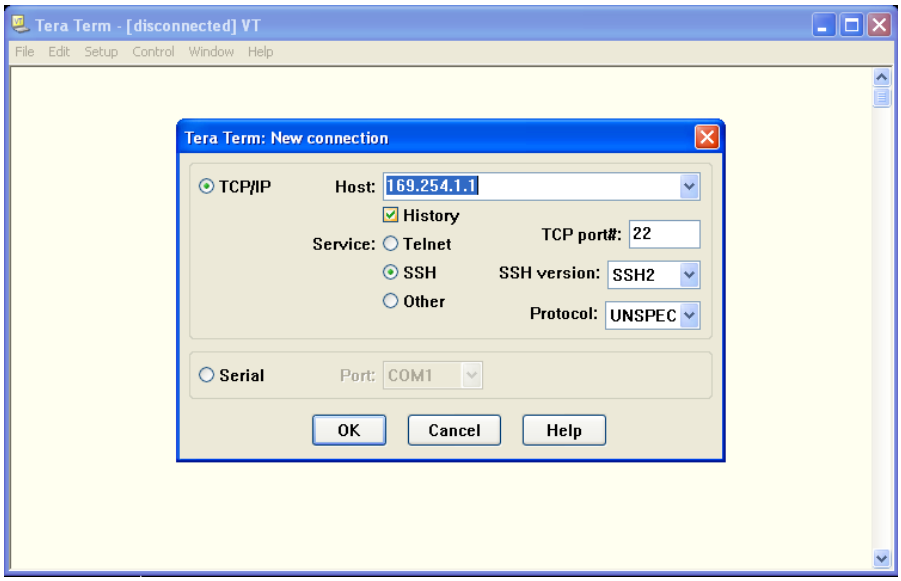


Figure 7.6: Tera Term Software

4. Click on **Serial** radio button and select the USB to serial adapter comport from the drop down box and click **OK** to continue.

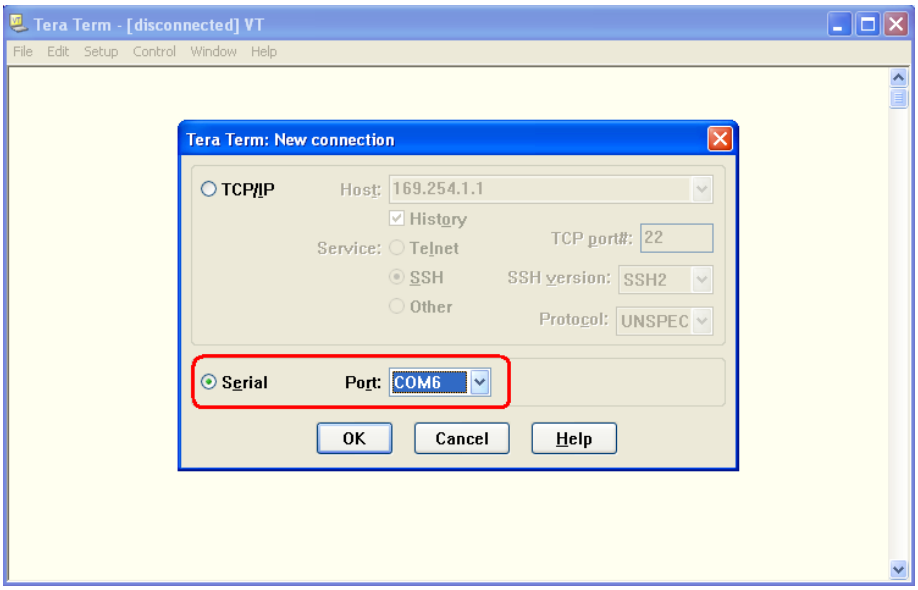


Figure 7.7: Tera Term New Connection

5. Tera term will open the selected com port. Make sure that the baud rate is set to 115200 bps, as shown in section 7.2. The wifi module is configured to operate at 115200 bps.

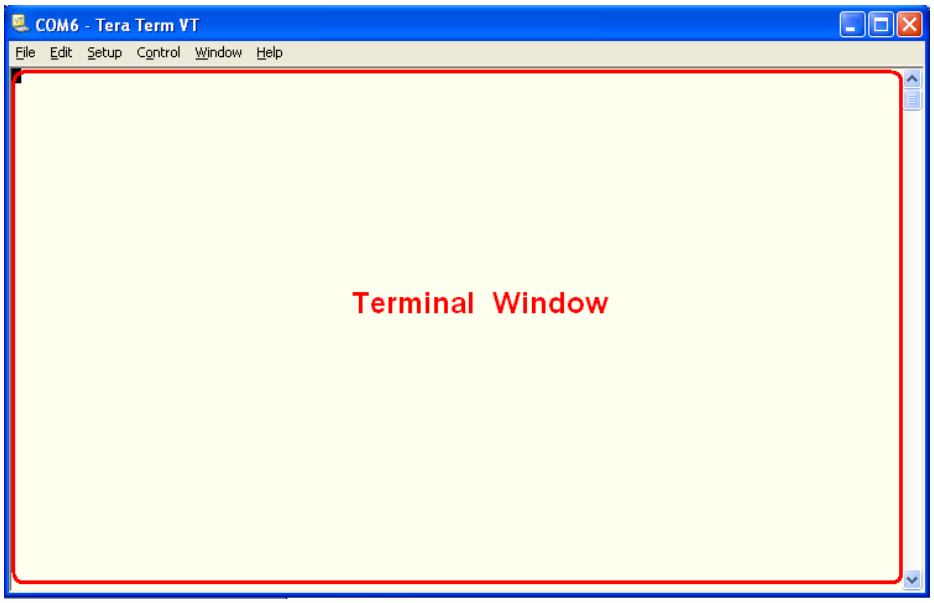


Figure 7.8: Tera Term Terminal window

6. The wifi module can be configured only when it is in command mode. To enter into the command mode, type **$$$** in the terminal window. The module should respond with CMD indicating that it is in command mode.

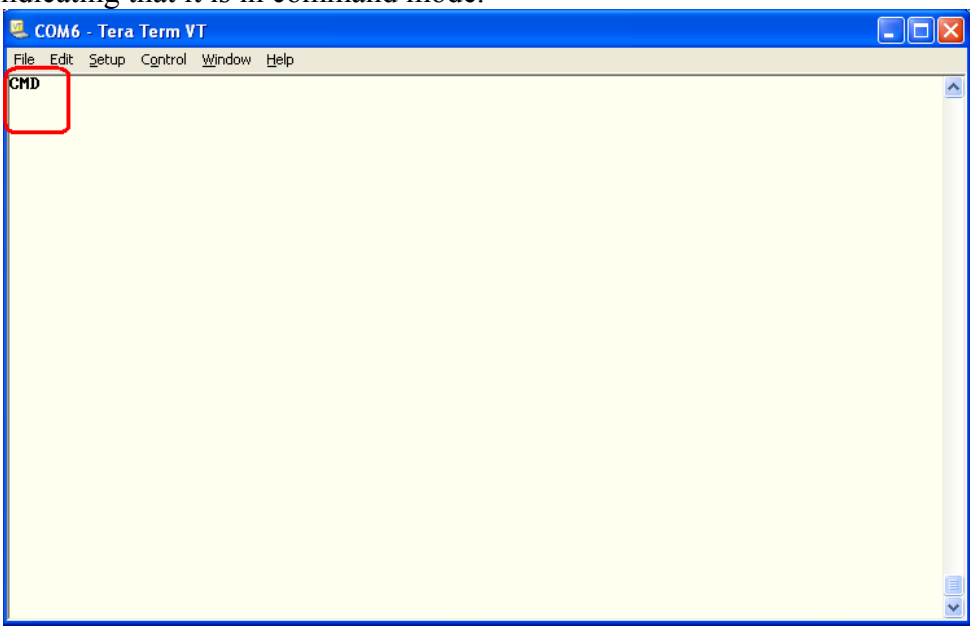


Figure 7.9: Entering Command Mode

7. The first command is to enable adhoc mode. Type ***set wlan join 4*** and press enter. The module should respond with AOK indicating positive acknowledgement of the command. The acknowledgement should be followed by the version of the module. Type commands carefully as they cannot be seen in the terminal window. If you receive error, retype the command carefully.

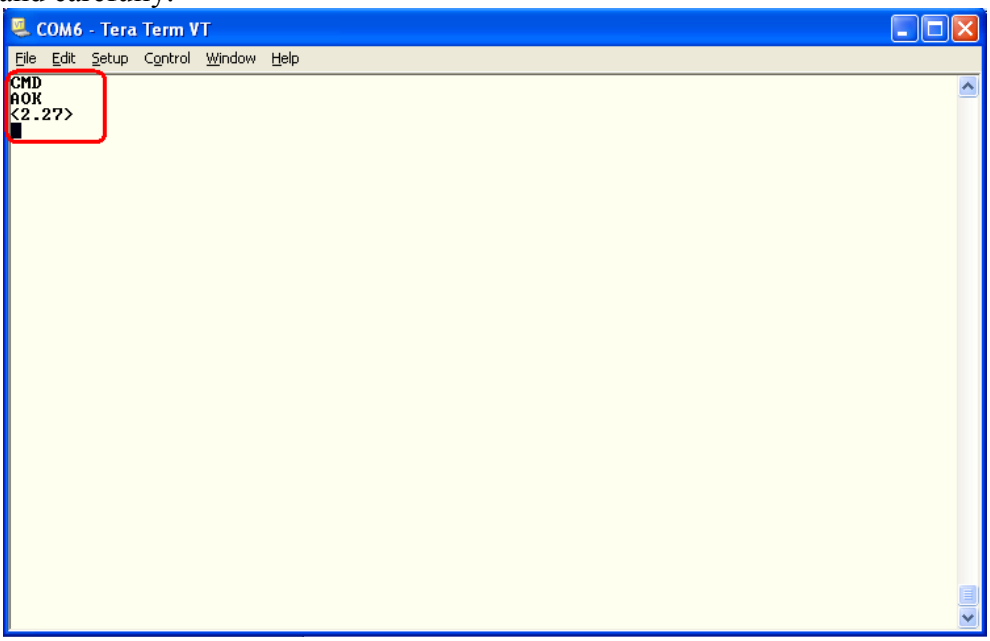


Figure 7.10: Command Acknowledgement

8. The next command will set SSID of the module. This SSID will appear in the windows wireless network connection wizard. Type ***set wlan ssid Nex_Robotics*** and press enter. The module should again respond with AOK.

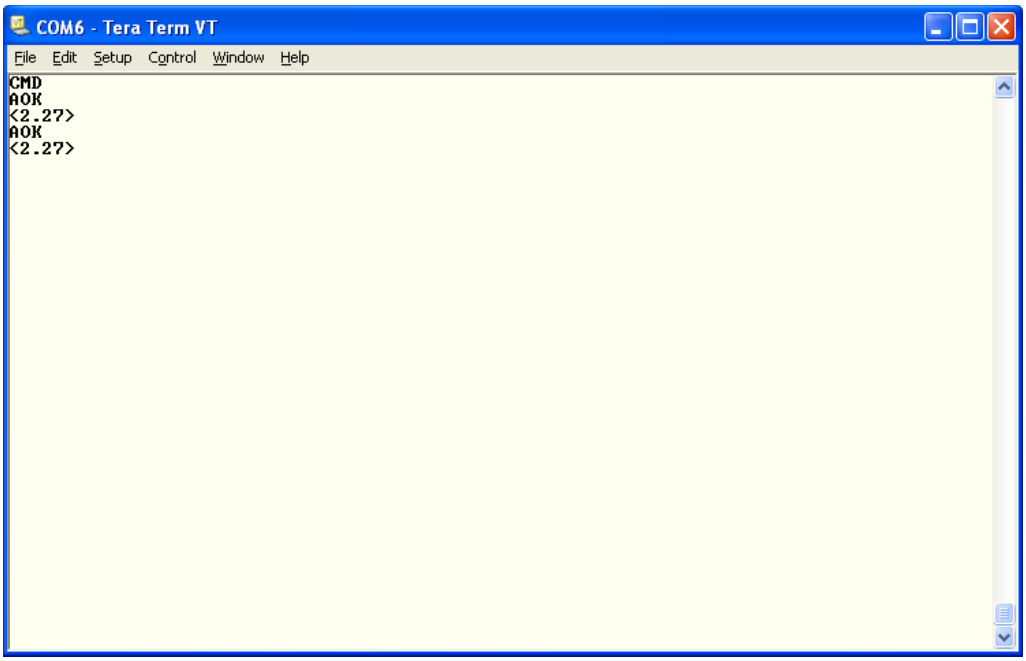


Figure 7.11: Command Acknowledgement

9. The following set commands should be issued to complete the configuration. If you receive error while entering the command, retype the command.

set wlan chan 1
set ip address 169.254.1.1
set ip netmask 255.255.0.0
set ip dhcp 0

10. The next step is to save the configuration so that it will be retained during power up. Type ***save*** and press enter followed by ***reboot*** and once again press enter.

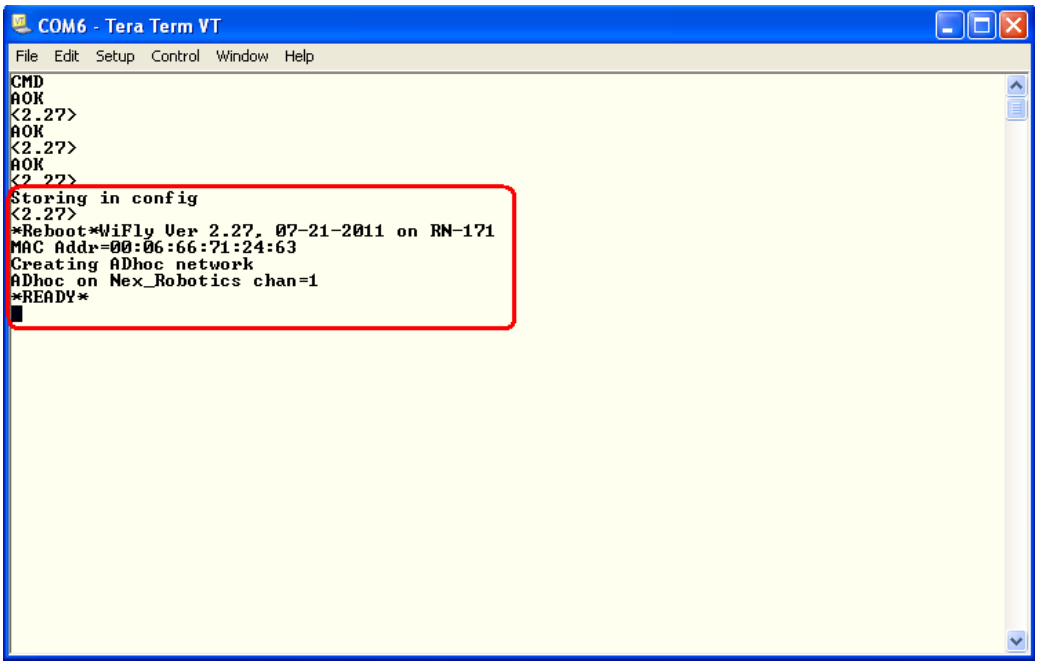


Figure 7.11: Command Acknowledgement

11. Rebooting should create an ADHOC connection with the selected channel number.

12. Now disconnect USB cable from PC and unmount WiFi module from USB to serial adapter and mount it on wi-fi adapter on Fire Bird VI. Connect wi-fi adapter cable to the UART1 connector on the main board as shown in fig. 7.4.

9.7 Installing USB WiFi dongle

Install USB wifi dongle drivers from the CD that came with the dongle. The USB dongle may have its own wireless configuration wizard. For demonstration we will use windows wireless configuration utility.

9.8 WiFi Communication with the robot

The wifi communication requires Wifi Communication firmware to be loaded and running on LPC1769 microcontroller. The firmware can be found in the documentation CD. It is also added in LPC xpresso IDE at the time of importing archived projects.
The following steps will guide in creating the wifi communication link.

1. Open Network Connections from control panel. Turn ON the robot.

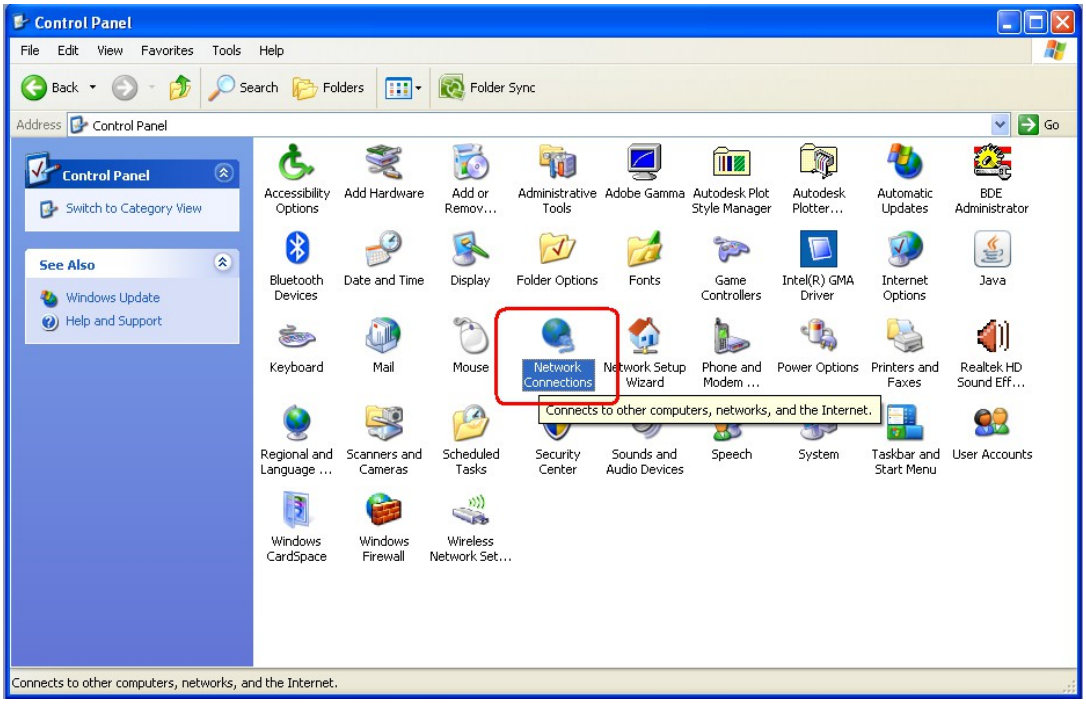


Figure 7.12: Windows Control Panel

2. Click on Wireless Network connections icon. Windows will show the SSIDs of available networks. Select the SSID that you had entered during adhoc configuration and click connect. If SSID in not shown, click on **Refresh network list** from the top left corner.

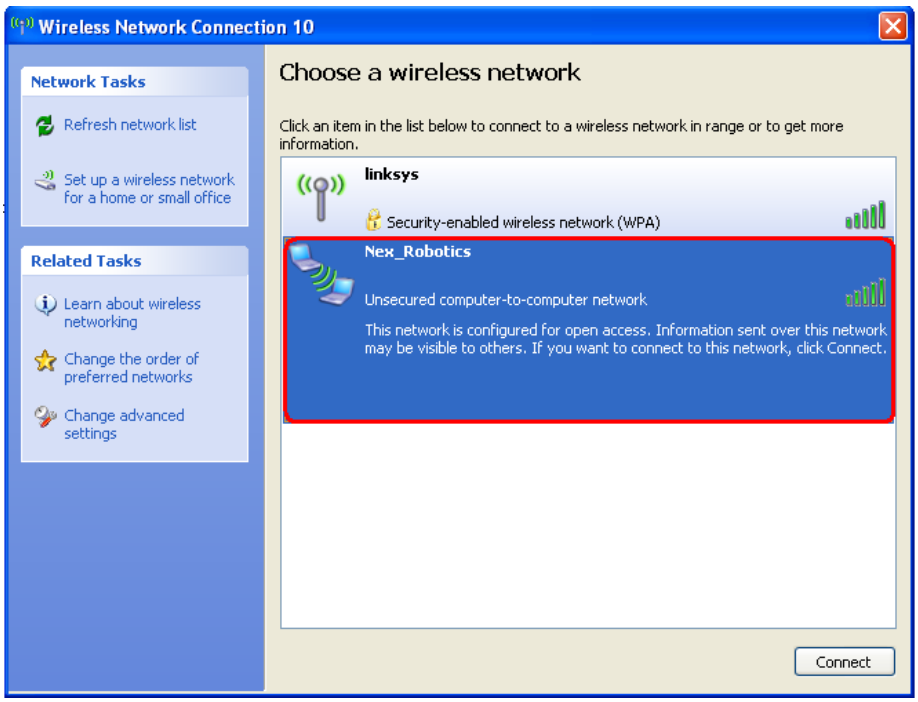


Figure 7.13: Wireless Network connection wizard

3. In the following window click connect.

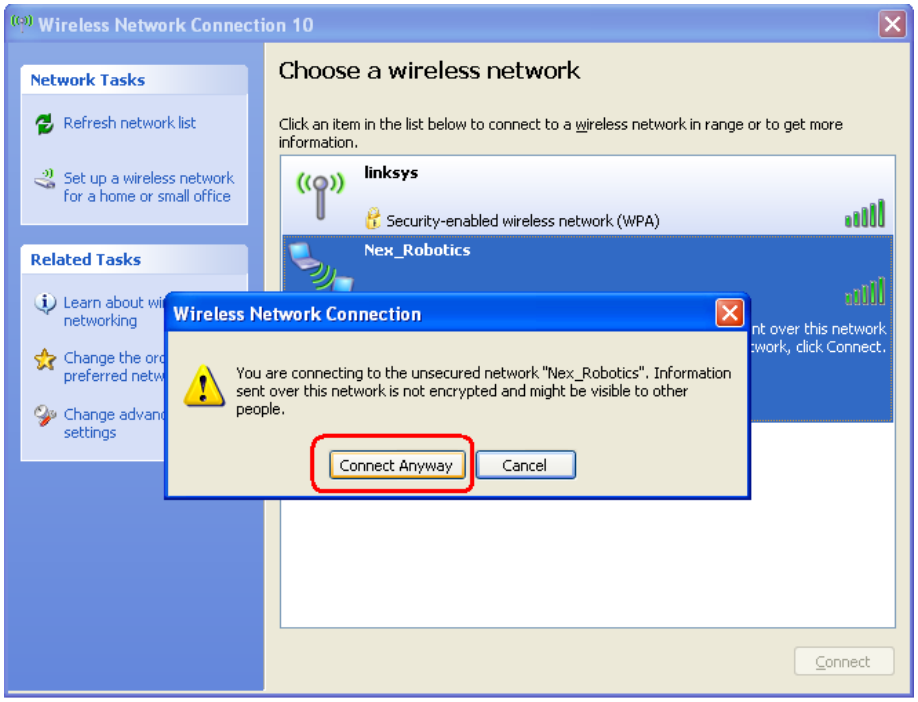


Figure 7.14: Wireless Network connection wizard

4. On clicking connect; windows will start acquiring IP address. Wait until IP address is acquired. It may take approximately 30 seconds or more.

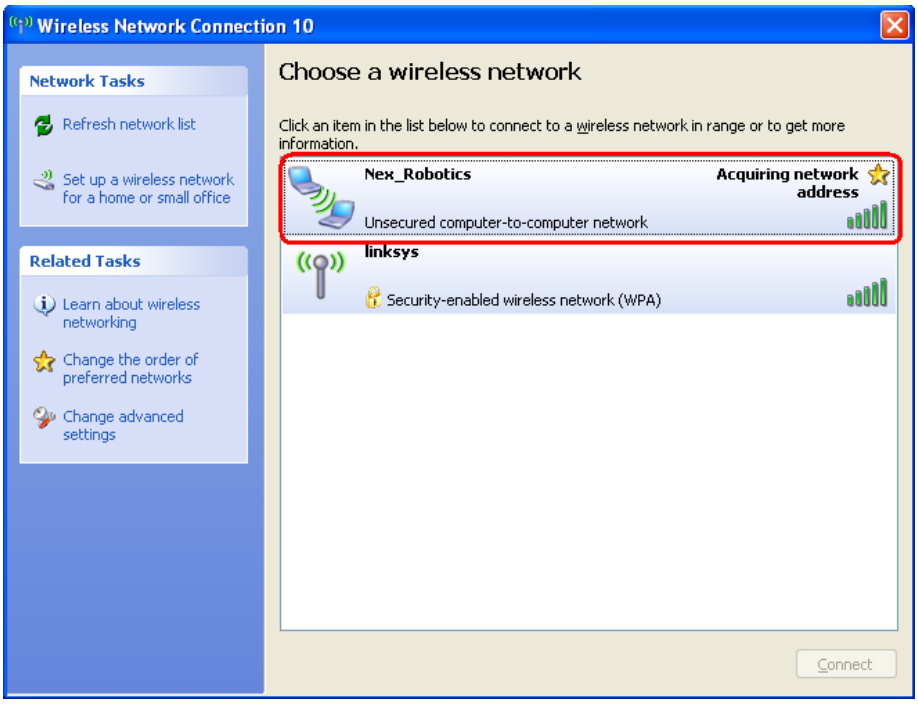


Figure 7.15: Acquiring Network Address

5. On successfully acquiring IP address the status should change to connected.

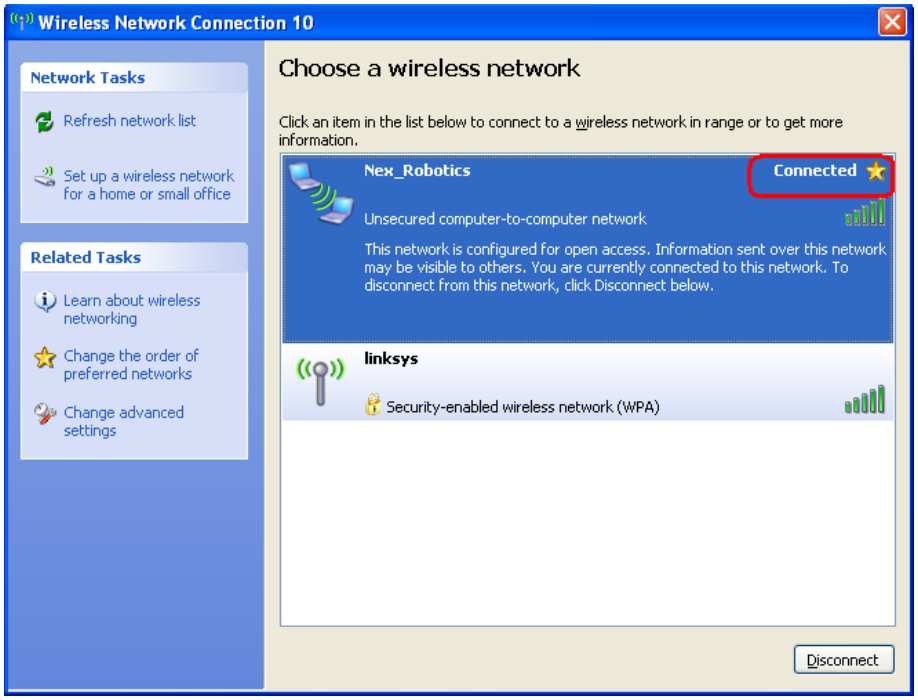


Figure 7.16: Wireless Connection Status

6. Close Wireless network connection window and open Tera Term.

7. In Tera Term: New connection window, click on TCPIP radio button and select Telnet service. Enter Host IP as **169.254.1.1** and TCP port# as **2000** and click OK.

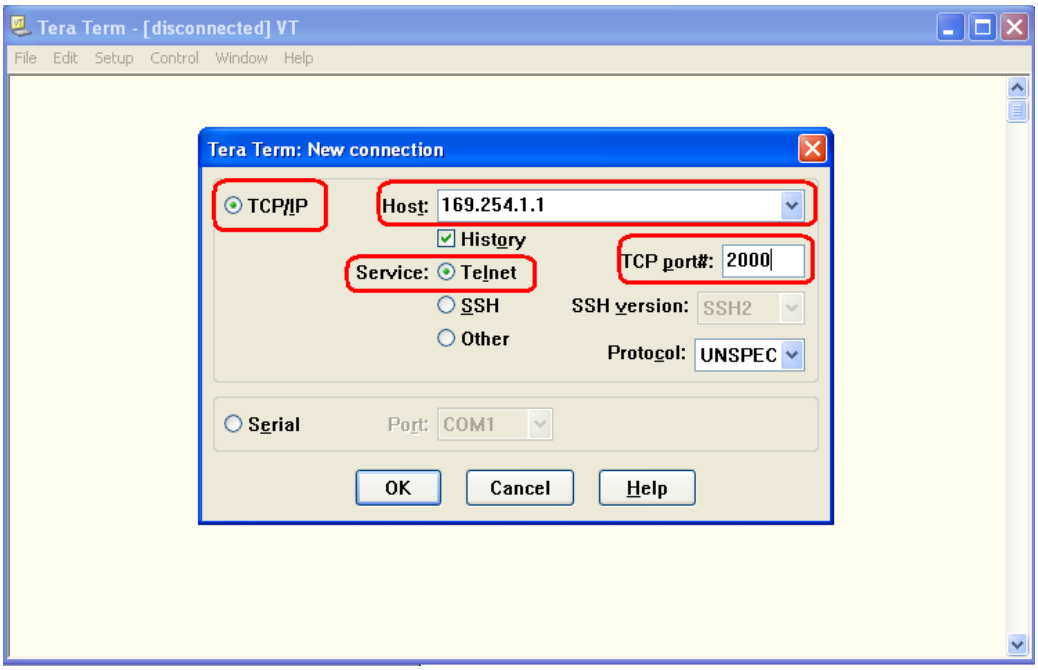


Figure 7.17: Tera Term Telnet Setup

8. Tera Term will create a telnet connection and the wireless module on Fire Bird VI should respond with ***Hello*.**

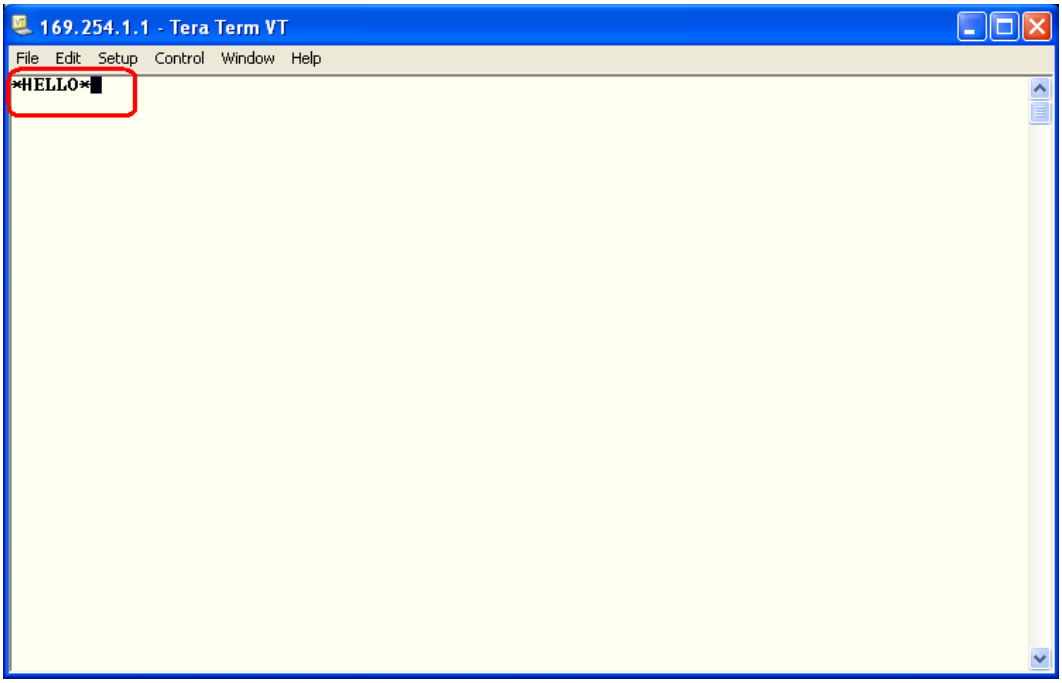


Figure 7.18: Tera Term Telnet Setup

9. Go to Setup-> Terminal from menu bar.

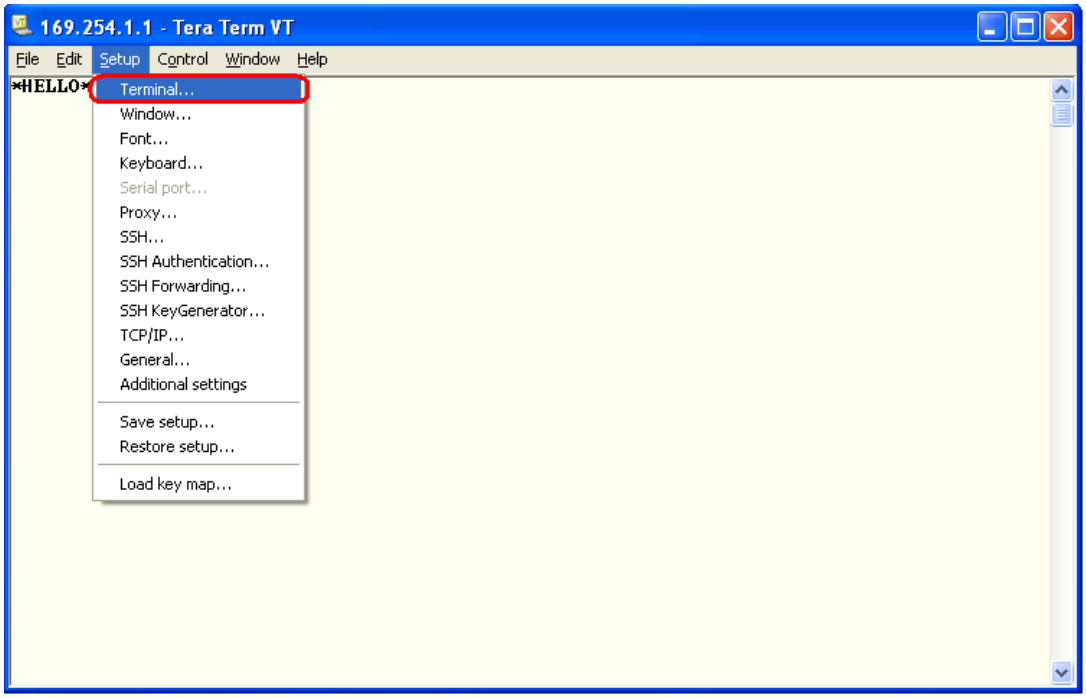


Figure 7.19: Tera Term Terminal Setup

10. In the terminal setup window, select local echo check box. This will display commands typed in the terminal widow. Click OK to close the setup window.

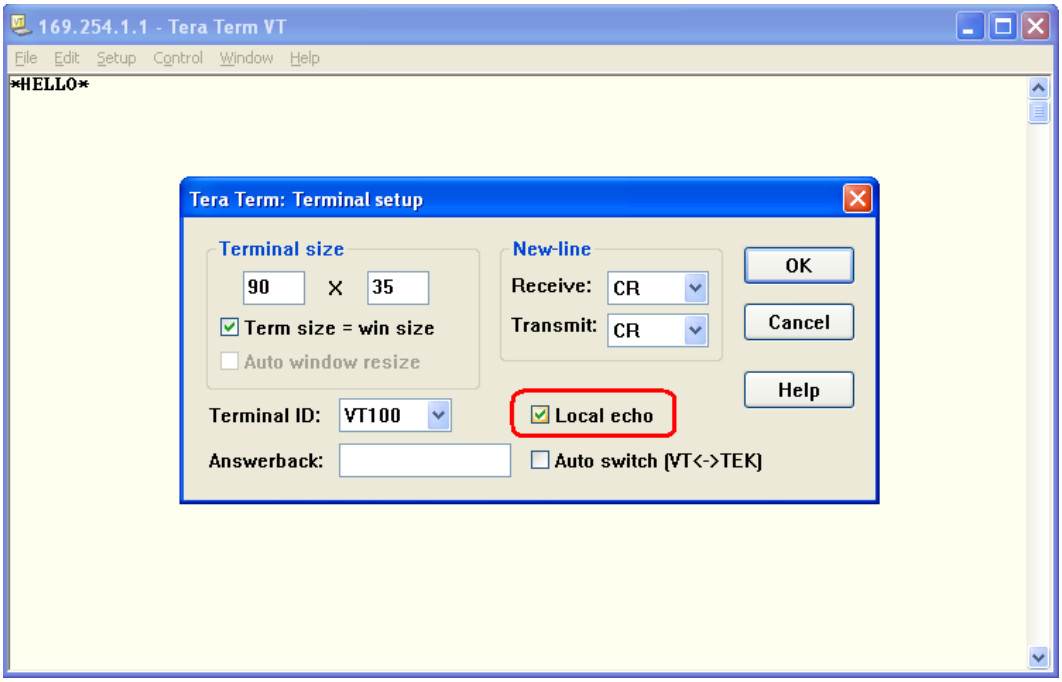


Figure 7.20: Tera Term Terminal Setup

11. In the terminal window press space bar key once. This will synchronize Fire Bird VI UART with the terminal window.

12. Type **NEX3** and press space bar key. This action will move robot in anti clockwise direction or left direction.

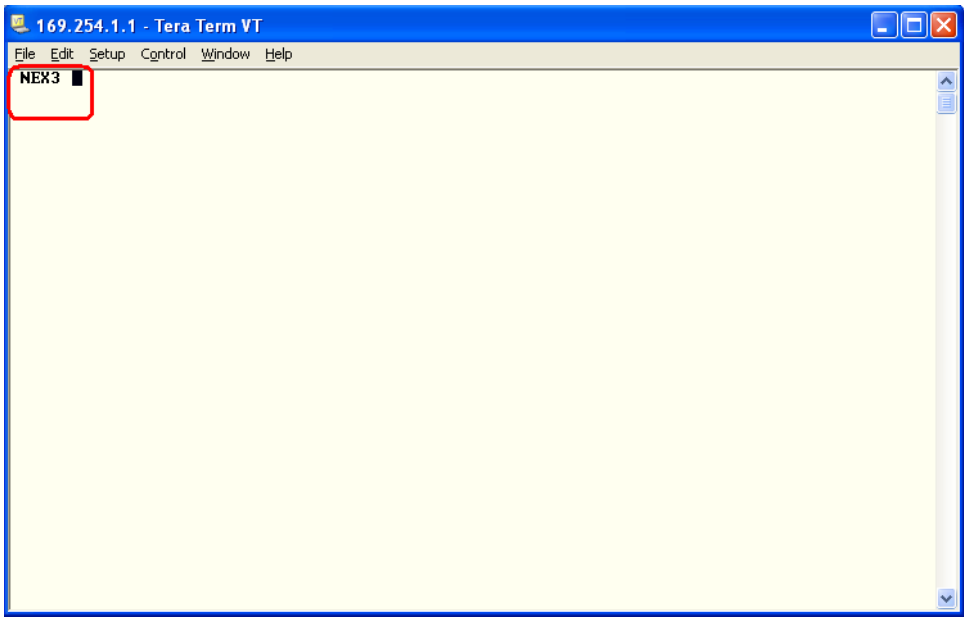


Figure 7.21: Tera Term Terminal Setup

You may also try other commands as listed below:

NEX1: Move Forward
NEX2: Move Backward
NEX3: Move Left
NEX4: Move Right
NEX0: Stop

13. To close the connection, go to File-> Disconnect.

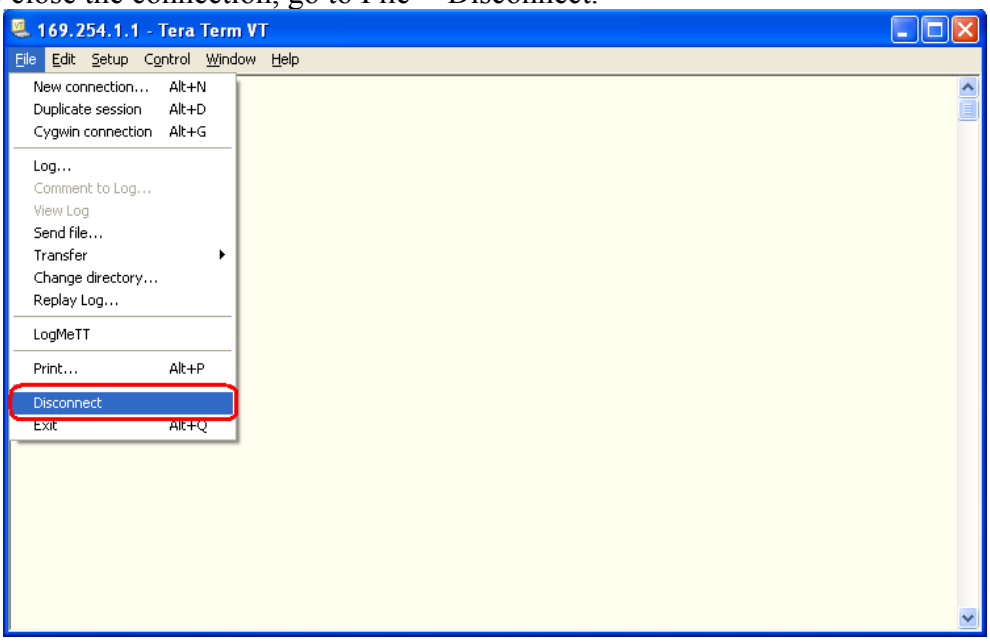


Figure 7.22: Closing Telnet connection

10 XBEE Communication with Fire Bird VI

10.1 Introduction

The XBEE communication uses tiny radio transceiver operating at 2.4 GHz and following 802.15.4 IEEE standard. The XBEE adapter board on Fire Bird VI is interfaced to UART of LPC1769 and shares same UART port as Bluetooth and Wifi. The XBEE module is configured to operate at data speed of 9600 bps. The other end of XBEE transceiver can be a similar XBEE module connected to a robot or a microcontroller development kit or a PC. For demonstration of XBEE communication we will use USB to serial adapter shown in fig. 8.3. The demonstration also uses Fire Bird VI GUI configured at baud rate of 9600.

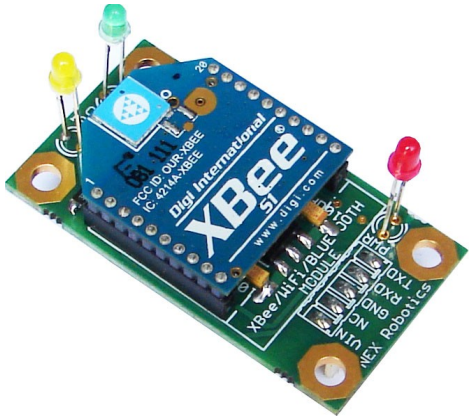


Figure 8.1: XBEE module on Fire Bird VI

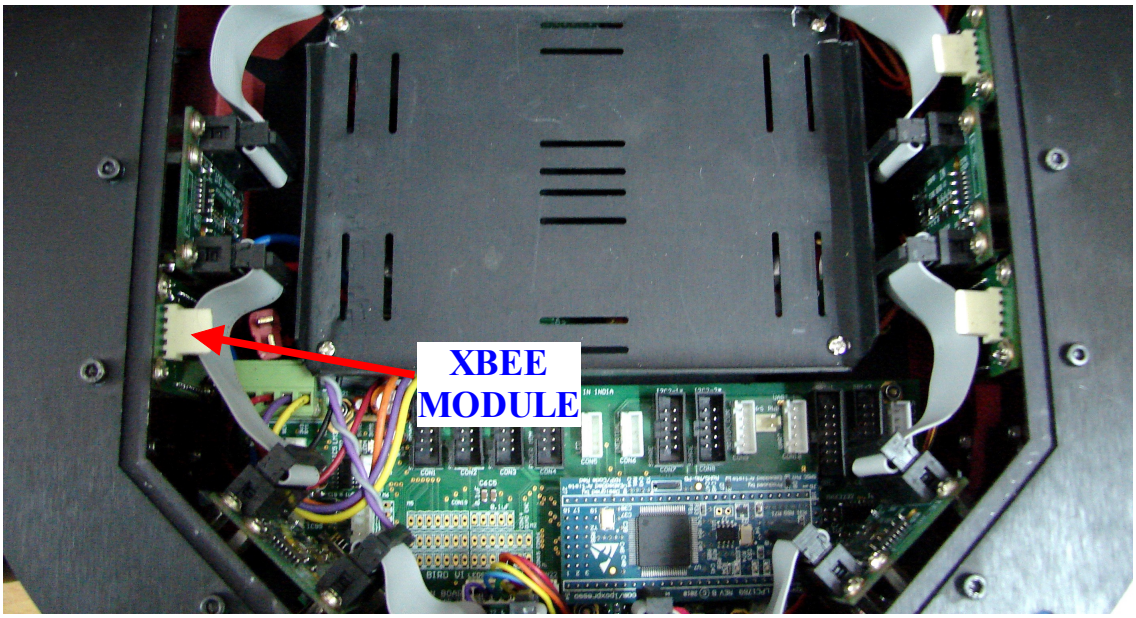


Figure 8.2: Location of XBEE adapter board on Fire Bird VI

Figure 8.3: XBEE module mounted on USB to serial adapter

10.2 Hardware Connections on Fire Bird VI

Connect XBEE adapter module cable to UART1 connector on the main board as shown in the fig. below.

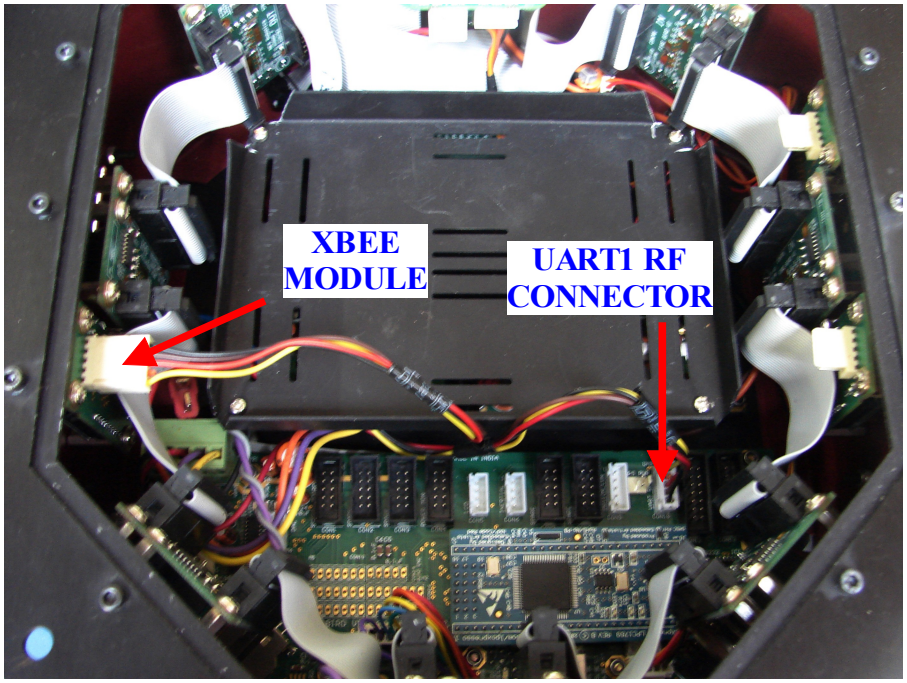


Figure 8.4: Wifi Module connection on Fire Bird VI

10.3 XBEE Communication using GUI

It is assumed that Fire Bird VI GUI is installed on your PC. If not, please refer chapter 5 for GUI installation

1. Start Fire Bird VI GUI from start menu or desktop shortcut and turn ON the robot.

2. Select baud rate as 9600 from the drop down box.

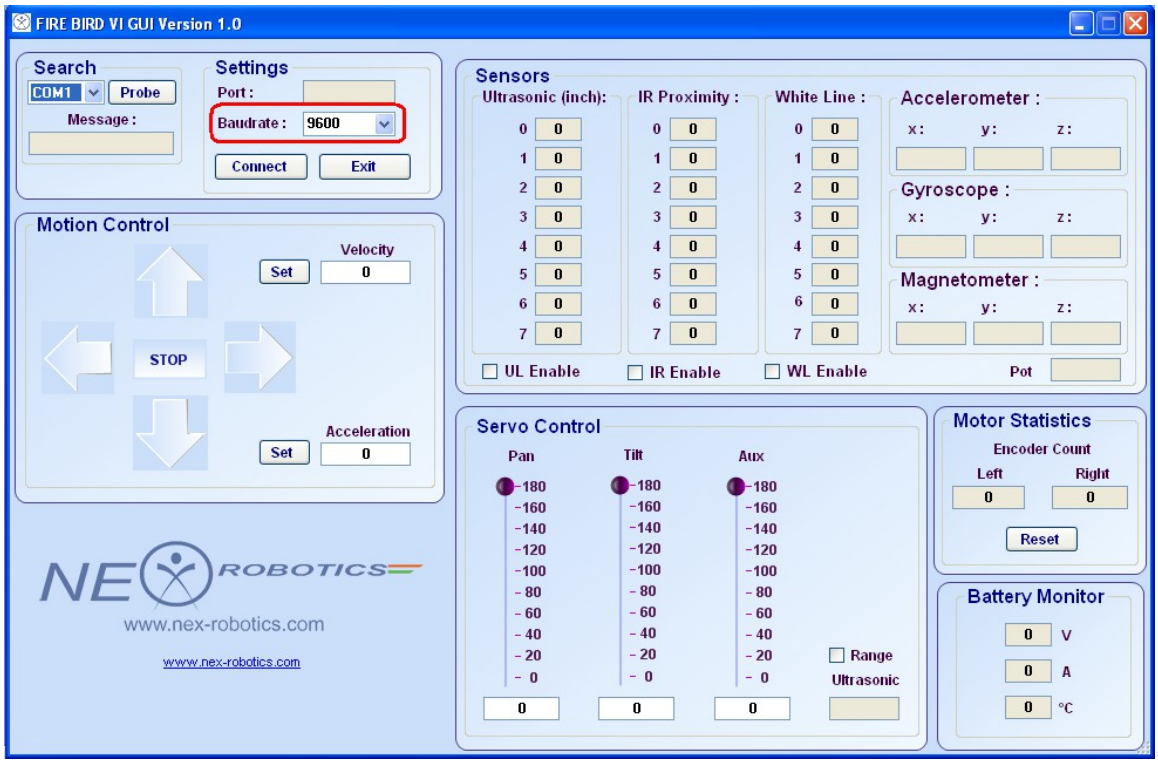


Figure 8.5: Setting baud rate for Xbee communication

3. Select the COM port of USB to serial adapter board and click probe. The message box should display Fire Bird VI. If not, ensure that robot is ON and comport is correct.

4. Click connect to start the communication.