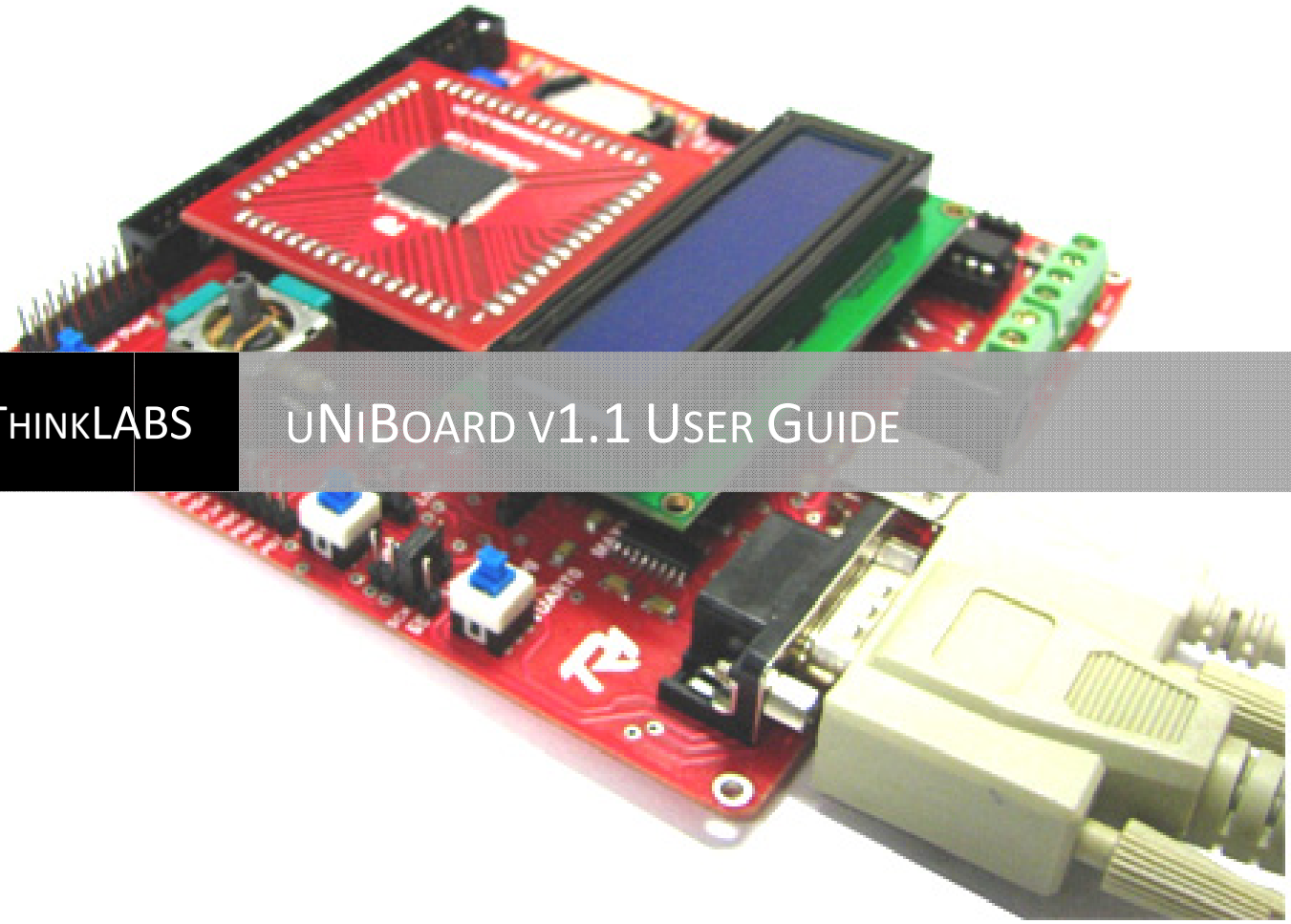
1/10/2009



THINKLABS

UNIBOARD V1.1 USER GUIDE

Open platform for Embedded and Real Time Systems Programming |
TRI Technosolutions Pvt. Ltd.

Table of Contents

Introduction

The **uNiBoard version 1.1** is an ideal **open source development platform** for **Embedded and Real Time Systems Programming.** Powered by a RISC machine (ATMega128) that provides a throughput of 16 MIPS and up to 128KB of internal storage (flash), the board would be suitable for any sort of embedded application development.

On-board peripherals like Joystick along with the communication ports (RS232) and the Gtkterm driver (hyper-terminal for Windows) make the board apt for basic game development in an Embedded (non-OS) as well as OS based environment. RT Kernels with small footprint **(uC/OS-II, FreeRTOS, nut OS**) can be ported on the board to gain hands-on experience of Real time application design.

The board is powered by the USB port. The board is also programmable through USB port thereby making it complete stand-alone lab equipment needing nothing apart from a basic PC/laptop to get started with the development process. The open interface (open LED interface, open ports) extend the platform's role for prototyping applications like external device/sensor interfacing. The controller by itself supports protocols like SPI, I2C (on-board I2C based RTC), UART (dual programmable UART) which can be used for multi-board communication.

Additionally, the board also features an on-board motor driver which allows to control up to two DC motors bidirectionally. External supply, if required for the motors can be provided with appropriate hardware configuration settings.

The board along with its **content-rich user manual** is a perfect companion to have for hobbyists/aspirants seeking a career in Embedded software design, since it can accommodate preliminary **applications** like port control or sensor data processing built **on Embedded C** to **complex real time applications** built **on RTOS** like DAS (Data Acquisition Systems), embedded web-server, FAT FS for embedded systems and more.

We plan to adhere to the Creative Commons license (Refer licensing terms), which has the philosophy of **"Share, Remix, Reuse - Legally".** It means that anyone is allowed to produce copies of the board, to redesign it, or even to sell boards that copy the design. You don't need to pay a license fee to the **ThinkLABS team** or even ask permission. However, if you republish the reference design, you have to credit the original group. And if you tweak or change the board, your new design must use the same or a similar Creative Commons license to ensure that new versions of the board will be equally free and open.

Package Contents

The uNiBoard development platform with the peripherals listed below

- uNiBoard development board – 1 unit
- 16x2 LCD – 1 unit
- Serial cable (DB9) – 1 unit
- USB cable – 1 unit
- FRC connector cables – 2 units
- CD consisting of uNiBoard user manual (based on Linux (Ubuntu) /Windows), essential software packages and sample codes (RTOS/non-OS applications) – 1 unit
- Pouch for storage – 1 unit

Testing the Board

This section lists the procedure to test the board. To enable you to quickly test the Board, it comes with a test program loaded. So, make sure you first test the board before any programming.

If you have programmed the controller thereby erasing the test program from the flash or in case you find that the sample program has not been loaded (which would be a rare case) then you need to load the test program onto the board (refer to the section Programming the Board). The test program is provided on CD uNiBoard Contents directory (inside sample codes). This code tests the board's peripherals (LCD, UART, LED's, Buzzer, and Joystick).

Testing the Board on Windows OS

Follow the steps to confirm if the uNiBoard is OK.

STEP 1: Connect the LCD on the Board (refer to section Hardware connections).

STEP 2: Use FRC cable to connect open LED port and Port C.

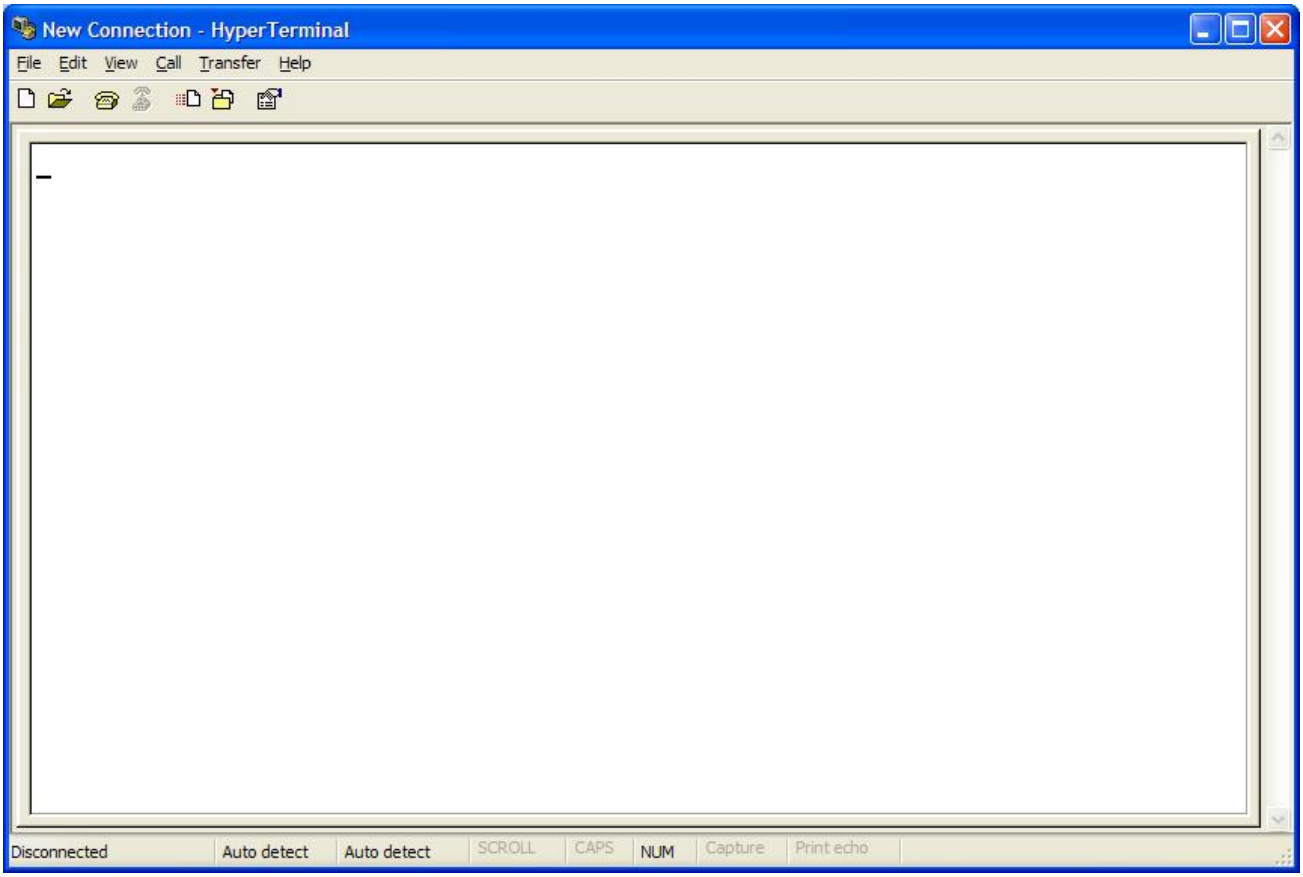
STEP 3: Using the serial cable connect uNiBoard to the PC.

STEP 4: If you are using Windows OS then you need install the USB drivers for the uNiBoard once (refer to the section Software Installation >> Software Installation for Windows OS >> Installing the USB drivers for uNiBoard).

STEP 5: Using the USB cable connect the uNiBoard to the PC.

STEP 6: Turn ON the Board once the drivers are installed correctly.

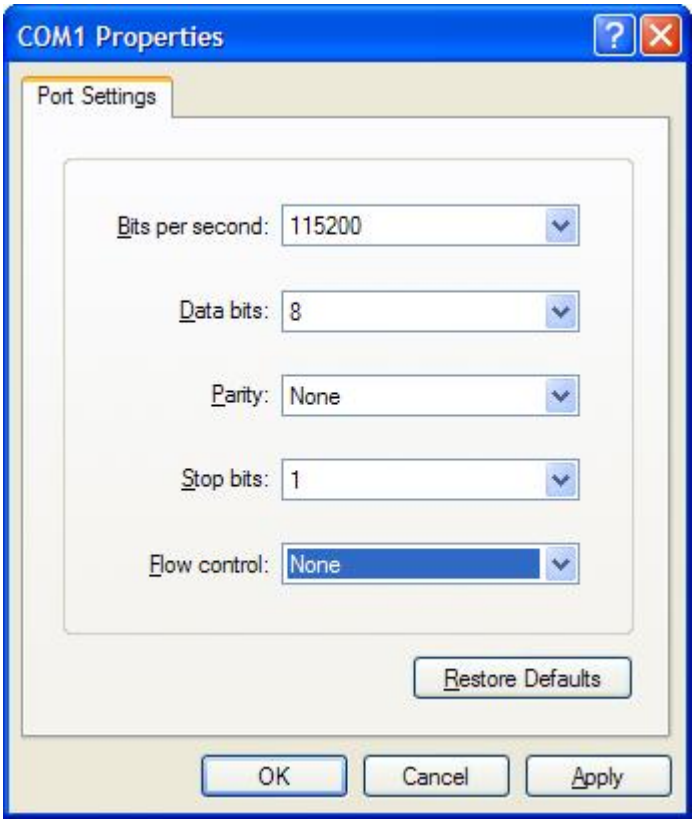
STEP 7: Open the Serial terminal and set the baud rate to 115200, parity to none, and stop bits to 1. So click on Start >> programs >> Programs >> Accessories >> Communications >> hyper terminal as shown below.



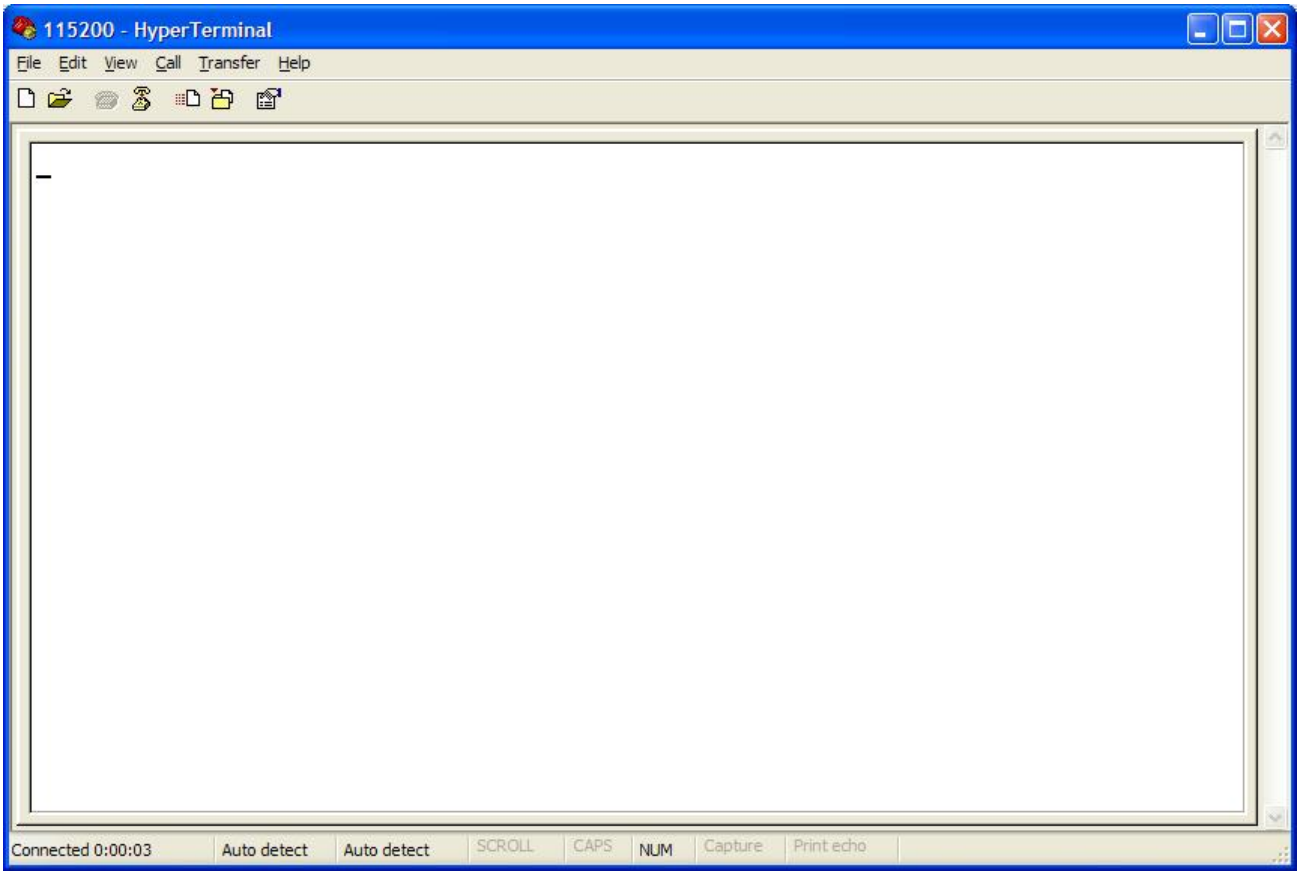
STEP 8: Select File menu >> New Connection then Enter Name for connection and click OK.

STEP 9: Select the COM Port where you have connected the serial cable.

STEP 10: Select the settings as given below for this test program.



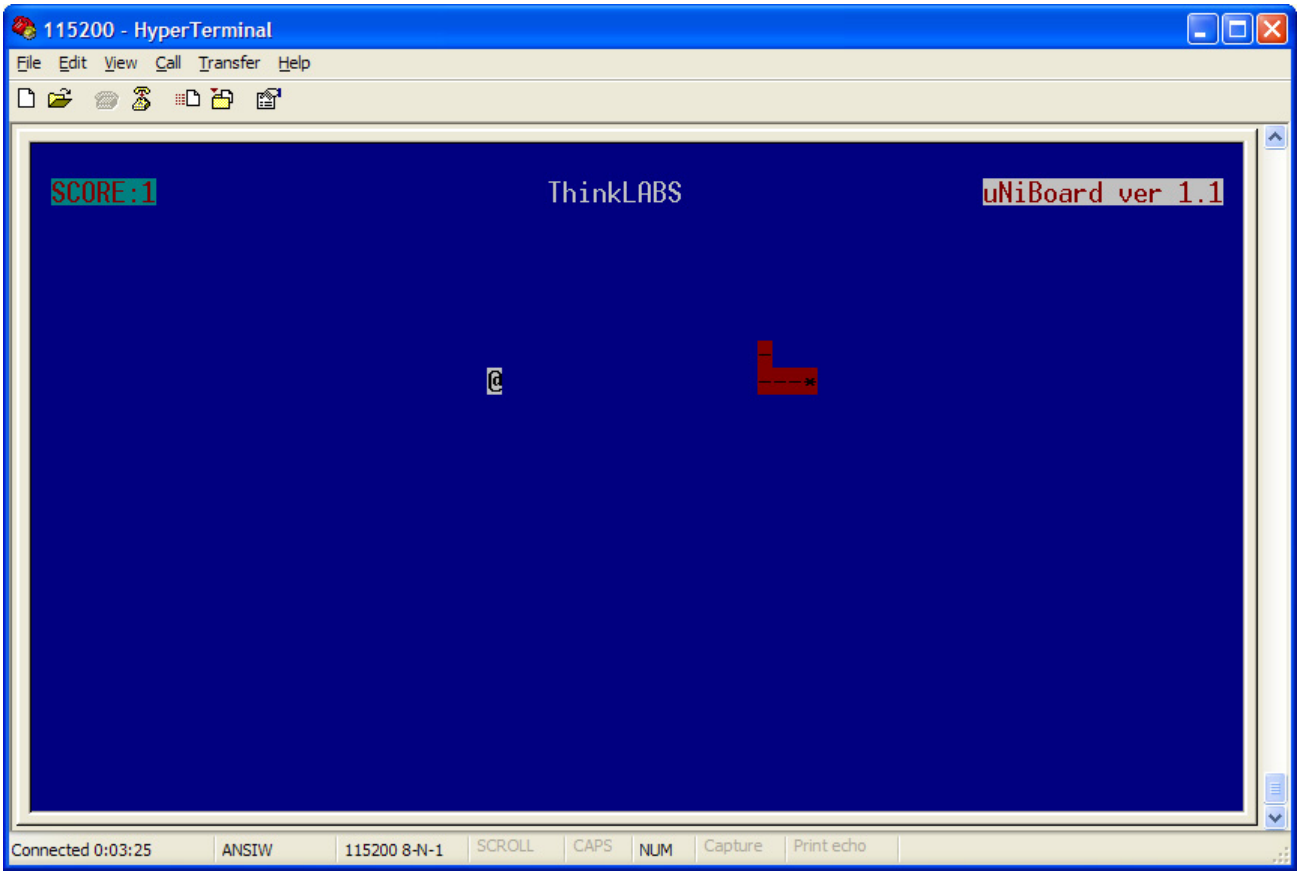
STEP 11: Select Apply and then OK.



STEP 12: Turn ON the Board.

STEP 13: The test program that we have loaded demonstrates SNAKE game on any serial terminal which is VT102 compatible. If everything is correctly installed you should be able to see a game of SNAKE (modified) as well as play using the Joystick.

While the program is running you will also be able to see the SCORE being displayed on the LCD along with the version of the uNiBoard that you are currently using. While the SNAKE eats up food you will be able to hear the buzzer. Every time a new food is generated you will see the LEDs glowing thereby utilizing all uNiBoard peripherals.



Testing the Board on Linux OS

Follow the steps to check whether or not the uNiBoard is working properly on **Linux OS**

STEP 1: Step 1 to Step 3 is same as given in the Windows OS procedure.

STEP 2: Turn ON the Board. If you are wondering how to install the USB drivers, you don't need to do that at all since Linux kernel has those drivers inbuilt.

STEP 3: Open the Gtkterm (**refer to the sub-section Installing Software for Linux OS**).

STEP 4: Configure the Gtkterm to baud rate/speed to 115200, parity to none, and stop bits to 1(refer to the sub-section Gtkterm Configurations for setting the Baud rate, Parity, Stop bits).

STEP 5: Turn ON the Board.

STEP 6: Out here as well you should be able to do things (like playing SNAKE on Gtkterm) that have been listed in STEP 13.

What you need for programming the Board

Hardware requirements

- uNiBoard version 1.1 along with USB cable
- Serial cable (optional)
- FRC Cable (optional)

Software requirements

- Text Editor (Gedit on Linux/ Programmers Notepad on Windows)
- Tool chain (avr-gcc, avr-binutils, avrdude): Installation procedure is explained in the upcoming sections
- Serial terminal (Gtkterm on Linux/ hyper terminal on windows)
- Driver for usbasp (Windows requires explicit installation)

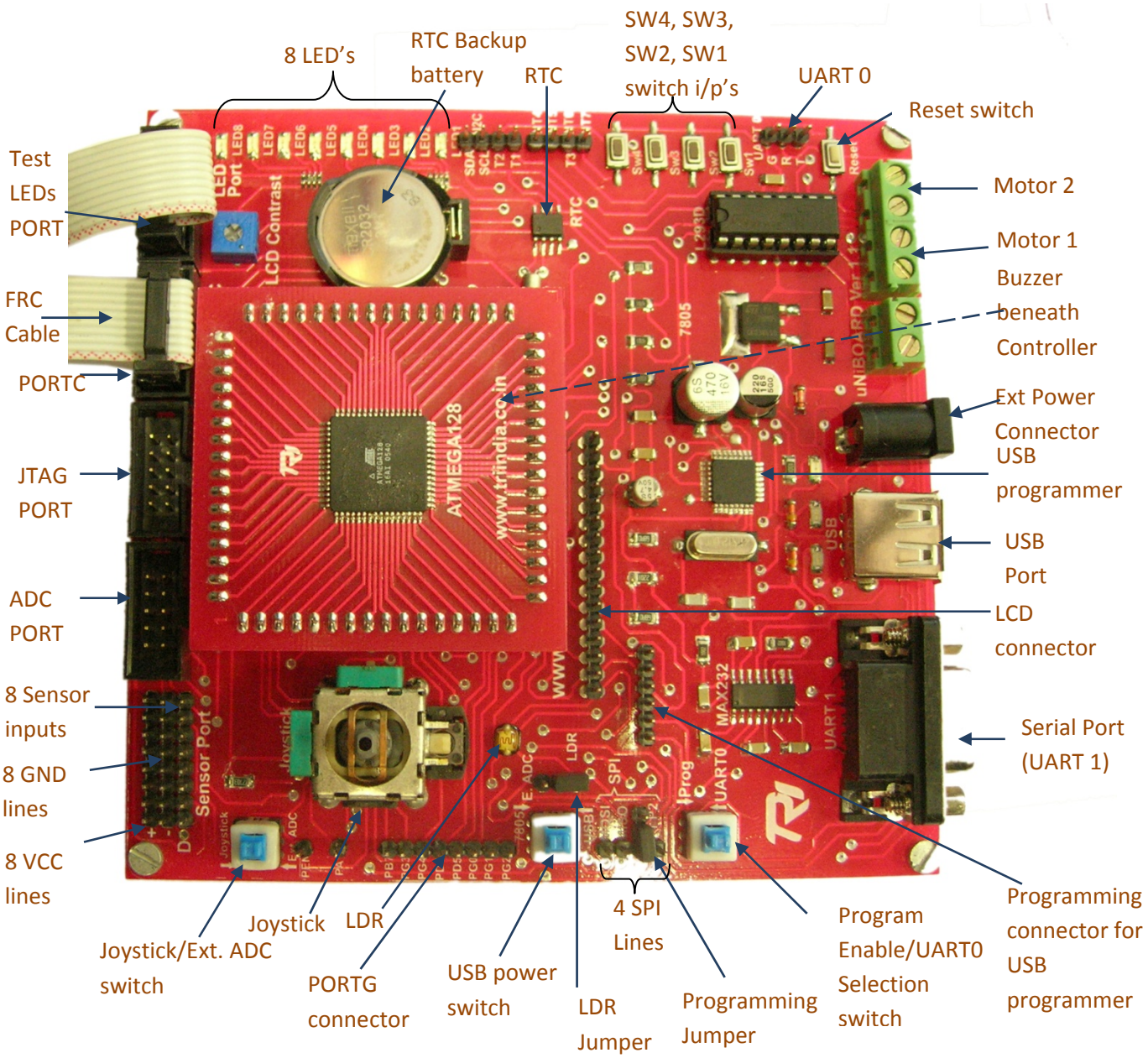
Safety and precautions to be taken

- Do not use the external power supply more than 15V.
- Board and SMD switches should be handled with care.
- Fuse settings of the target (micro-controller) should be modified only with appropriate knowledge, since locking the fuse section might render the board useless.
- **Do not** program the Board while you are using SPI lines for communication which might also render the board useless.

Pre-requisites

- Before you start, you are expected to have some knowledge of AVR microcontrollers and its software tools up to a preliminary level. We have not discussed the AVR architecture (Datasheet is the best help on the same. it has been included in the CD contents)
- The board guide does not give any conceptual knowledge with regards to RTOS. For best results, you are expected to have a reference material on the RTOS that you are going to use on the board. In case the **target RTOS** is **uC/OS-II**, you need not look beyond **MicroC/OS-II, The Real Time Kernel** by **Mr. Jean Labrosse.**
- All the samples examples based on RTOS including the SNAKE application that you must have just seen while testing the board utilize the same RTOS kernel (i.e.: uC/OS-II)

Hardware Connections



The uNiBoard version 1.1 connects to the PC using USB port. As shown in the figure 1.1 on the board there is a serial port and USB port connectors. The serial port is used for debugging purpose while USB port is used for power and burning the hex files into ATmega128 microcontroller.

The USB cable (for power/programming purposes) and serial cable can be interfaced with the PC. The serial cable is optional as we are not using it for programming the hex file into the Atmega128 chip. For any UART related activities having the serial cable is a must.

Features

Board Features

- ATmega128 controller with external crystal of 16MHZ
- usbasp: ATmega8 based USB programmer for ATmega128 (open source firmware running on hardware licensed under **GNU GPLv2** from http://www.fischl.de/usbasp)
- I2C communication lines
- SPI communication lines
- 16x2 alphanumeric LCD with contrast adjustment
- RTC with Backup Battery
- Analog Joystick with centre click
- LDR sensor
- Buzzer (beneath the processor board)
- Onboard Motor Driver (L293D)
- LEDs for USB programmer ready indicator, Programming status indicator, and Power ON indicator
- Test LEDs (open for interface with any PORT)
- Open interface Ports such as PORTF/ADC, PORTC
- JTAG interface(External JTAG hardware required)
- USB Port for programming the Board
- Dual programmable UARTs
- Push Buttons for External Interrupts, Reset
- Configuration Switches for USB Power/External power, External ADC /Joystick and Program Enable/UART0
- Modular design to permit replacement of processor board

Controller Features

The features of Atmega128 are as follows:

- Advanced RISC Architecture
- 128K Bytes of In-System Self-programmable Flash program memory
- 4K Bytes EEPROM
- 4K Bytes Internal SRAM
- JTAG (IEEE std. 1149.1 Compliant) Interface
- Two 8-bit Timer/Counters with Separate pre-scaler and Compare Modes
- Two Expanded 16-bit Timer/Counters with Separate pre-scaler, Compare Mode and
- Capture Mode
- Real Time Counter with Separate Oscillator
- Two 8-bit PWM Channels
- 6 PWM Channels with Programmable Resolution from 2 to 16 Bits
- 8-channel, 10-bit ADC
- Byte-oriented Two-wire Serial Interface
- Dual Programmable Serial USARTs
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with On-chip Oscillator
- Power-on Reset and Programmable Brown-out Detection
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- Software Selectable Clock Frequency (using Fuse bits)
- 53 Programmable I/O Lines
- 4.5V - 5.5V for Atmega128 Operating Voltages
- 64-lead TQFP

Setting up the Board configuration

Board Description

Atmega128 controller

The ATmega128 is a low-power 8-bit microcontroller based on the AVR enhanced RISC architecture. The Atmel ATmega128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

Atmega8 controller with the firmware for programming through USB

USBasp is a USB in-circuit programmer for Atmel AVR controllers. It simply consists of an ATMega8 and a couple of passive components. The programmer uses a firmware (USB driver) to program Atmega128 microcontroller. For more information on building USBasp refer http://www.fischl.de/usbasp. It is an open source firmware along with hardware licensed under GNU GPLv2.

I2C and RTC (DS1307) with Backup Battery

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information (Compensation Valid up to 2100). No special hardware configuration is required as it is mounted on our board and internally connected to the processors pins.

Analog sensors (Joystick and LDR)

The Joystick (Analog joystick used in PS2 consoles) is connected to the ADC, X axis on channel 1 (PF1) and Y axis on channel 2 (PF2) of the Atmega128 microcontroller. The LDR sensor is connected to channel 0(PF0) of the Atmega128 microcontroller.

Buzzer

The Buzzer is connected to **PA3** of the Atmega128 microcontroller, and lies beneath the processor board.

Onboard Motor Driver

The Motor driver chip (L293D) is used to drive the motors which can be connected to the PTR connectors as shown in the above figure. Through software you need to configure as follows:

PB6 and PB5 **(MOTOR1)**

PE2 and PE3 **(MOTOR2)**

PB4 (CHIP ENABLE)

There are motors which might need higher than 5V (up to 12V) operating voltage. In such cases an external supply can be given to the board and the USB power switch can be toggled to make the external supply available instead of a USB powered connection.

External power

The external power connector in the above figure can also be used for connecting rechargeable batteries and making the board operate on battery power in case of robotic or other such mobile applications.

Test LEDs

Test LED's are **pulled-up** so to **glow** the LEDs we need to make the particular port pin **Low**. You can connect the General Purpose PORT to test LED Port using FRC Cable.

Two UART's

UART1 is used to connect PC through the MAX232 voltage converter chip since pc uses RS-232 standard for serial port. UART1 can be used for debugging the code or for any sort of interaction with Gtkterm.

UART0 is not connected to MAX232 as it is left open for communication between two Boards.

LCD

LCD for uNiBoard is using 4 data lines, 2 control lines and WR of LCD is connected to GND.

DATA LINES (PA4, PA5, PA6, PA7)

CONTROL LINES (PA0 for RS, PA2 for LCD EN)

Push Buttons

External Interrupts

SW3 (INT6)

SW4 (INT7)

General Purpose switch

SW1 (PD6) Active Low

SW2 (PD7) Active Low

Reset SW

Selection Switches

USB Power (**Pressed**) / External power (**Depressed**)

Joystick (**Pressed**) / External ADC (**Depressed**)

Program Enable (**Pressed**) / UART0 (**Depressed**)

The most important part of board configuration is the configuration switches. More often than not, programming errors or abnormal program response and execution are due to faulty configuration of the switches. So, while programming, or using ADC, or external power supply or battery or while using UART0 make sure you have first set the configuration switches in the correct position before concluding that your program is not working or the board is faulty.

USB Power (**Pressed**) / External power (**Depressed**)

Refer to the uNiBoard figure where the components have been labeled to understand where these switches are physically placed on the board. While you have connected the uNiBoard cable for programming purposes or for powering the board this switch should be in the pressed position.

While connecting any external power supply or battery, this switch should be in depressed position. This makes the 7805 voltage regulator come into picture to provide 5V power supply for the ICs.

Joystick (**Pressed**) / External ADC (**Depressed**)

While you are using the analog joystick you are using the internal ADC of the controller which is available at PORTF. The LDR which is another analog sensor is also connected at PORTF. Make sure that this switch is in pressed position while the joystick is being used.

In case you want to use the open interface pins of the PORTF, in order to connect external analog sensors or digital sensors (refer labeled figure of uNiBoard), change the switch position to depressed. There is another jumper setting which you would need to do in order to take the LDR out of the circuit. This configuration will be discussed while discussing jumper settings.

Program Enable (**Pressed**) / UART0 (**Depressed**)

While programming the board this is one switch that you should never forget to press, failing which avrdude (programming software will flash an error saying “target not found”.

Once programming has been completed in order to gain access to UART0 you will need to keep the switch in the depressed position, since the UART0 pins are multiplexed with the programming pins.

General Purpose PORTS

PORTC

PORTC is an open interface port which can be used to connect any devices on FRC connector.

PORTF (Joystick/LDR/External sensors)

PORTF can be used to connect external analog or digital sensors or it can also be used as Joystick or LDR ADC channels. There are two connectors FRC connector and Berg sensor port connector either of which can be used to connect the external sensors. The Berg sensor port connector is compatible with our TRI sensors. You can visit to website at http://www.thinklabs.in/resources/ to checkout TRI sensors and you can buy at http://www.thinklabs.in/shop/.

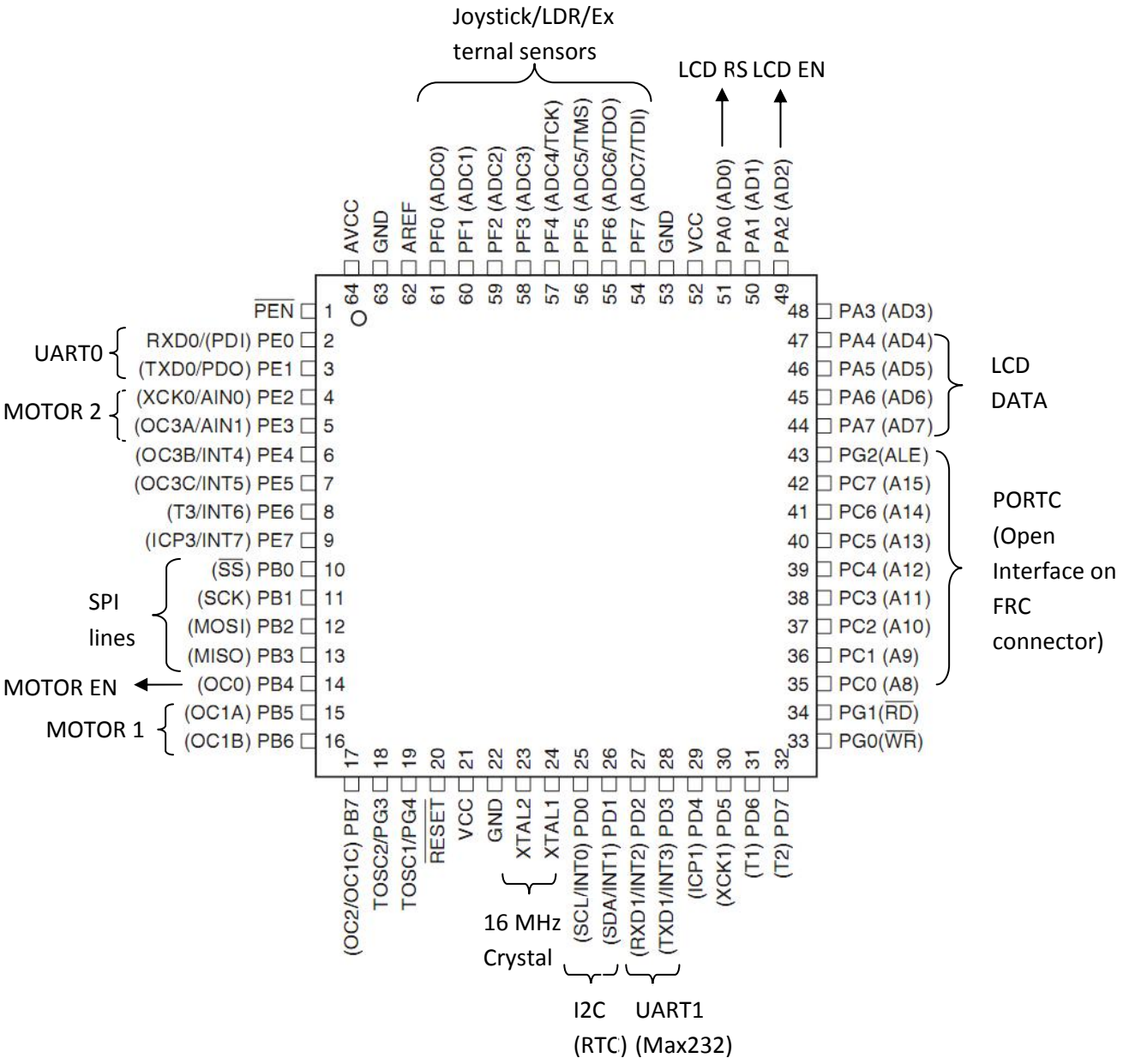
PORTG

PORTG is an open interface port which can be used to connect any devices on berg connector.

PORTC and PORT F

The PORTC and PORTF (ADC) are open ports and can be connected to test LEDs through FRC cable. These can be accessed using 10-pin the FRC connector.

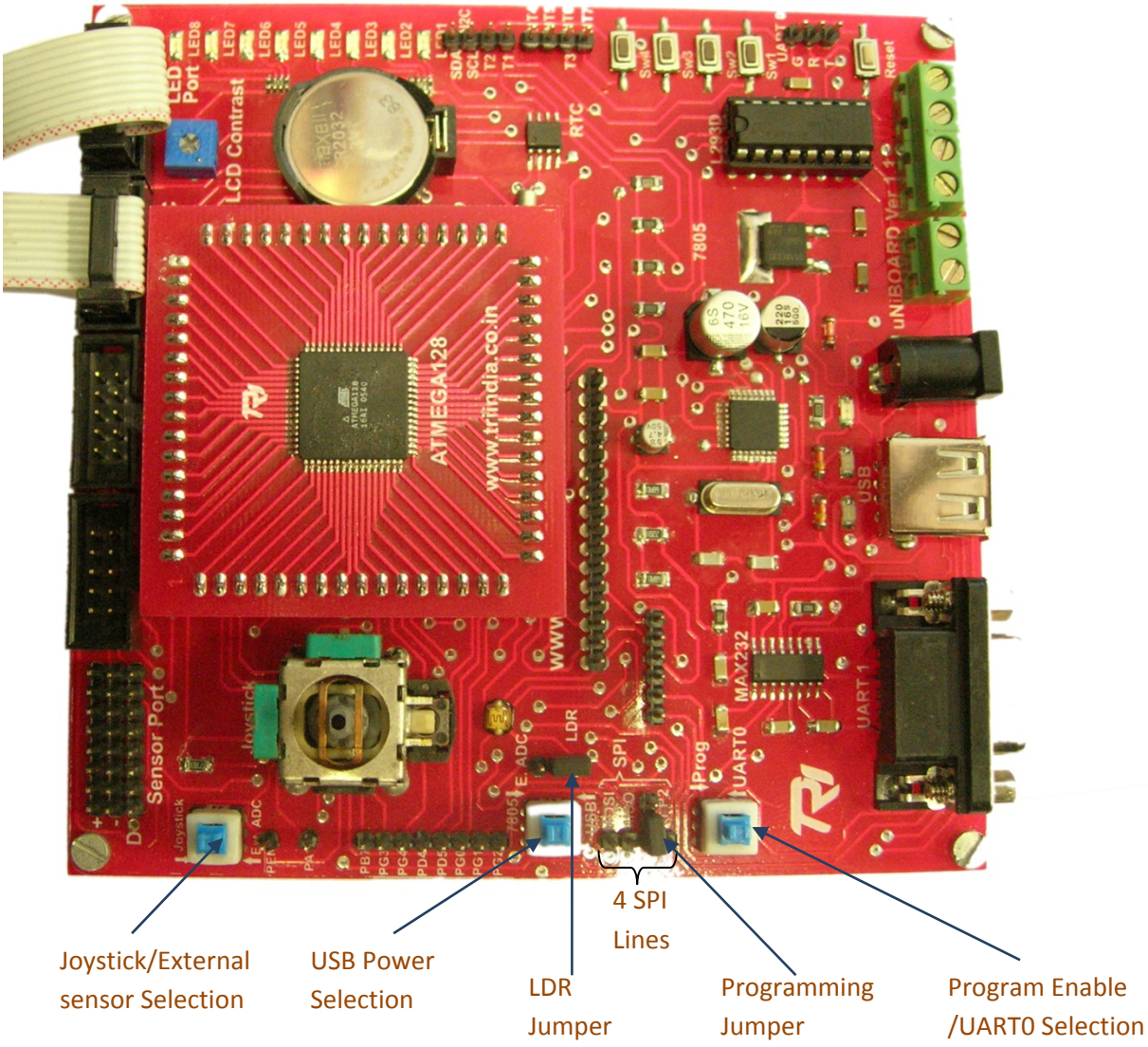
The pin diagram of Atmega128 is given below along with the manner in which we have utilized the port pins in case of uNiBoard:



Jumper Settings

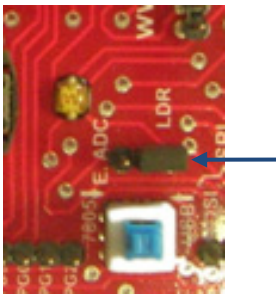
Apart from the configuration switches discussed above the jumper settings is another source of arriving to wrong conclusions provided that they have not been set correctly.

There are two jumpers on the board one for programming or SPI jumper and other one for LDR or External ADC selection. As shown in the figure below:

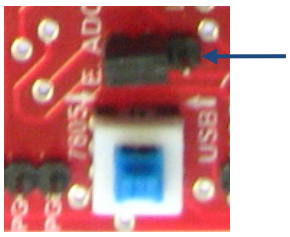


LDR jumper

You can use them as per your application requirements. In case you are connecting external sensors at the sensor port, you will need to change the position of the LDR jumper so as to disassociate it from the circuit as shown below:



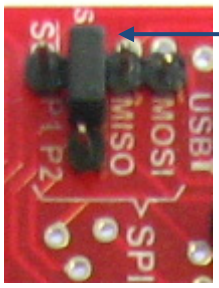
While connecting external sensors at the open interface of PORTF, this jumper position should be changed to the alternate position as shown below:



Position of jumper using external analog sensors

Programming jumper:

The programming jumper has to be retained in the position shown below under all cases except when using the SPI communication lines.



Position of jumper while using LDR sensor

While using the SPI communication lines the jumper can be either removed or placed in the adjoining pins provided which is NC (No Connection), as shown below. The latter is a safer option to ensure that you do not misplace the jumper.

Software Installations

Software Installations for Linux OS

The procedure that we are discussing for Ubuntu Linux involves downloading packages through the synaptic manager. Hence you need to have an internet connection in place before you begin with this. If not get hold of *.deb or *.rpm packages of the listed software and install it on your system.

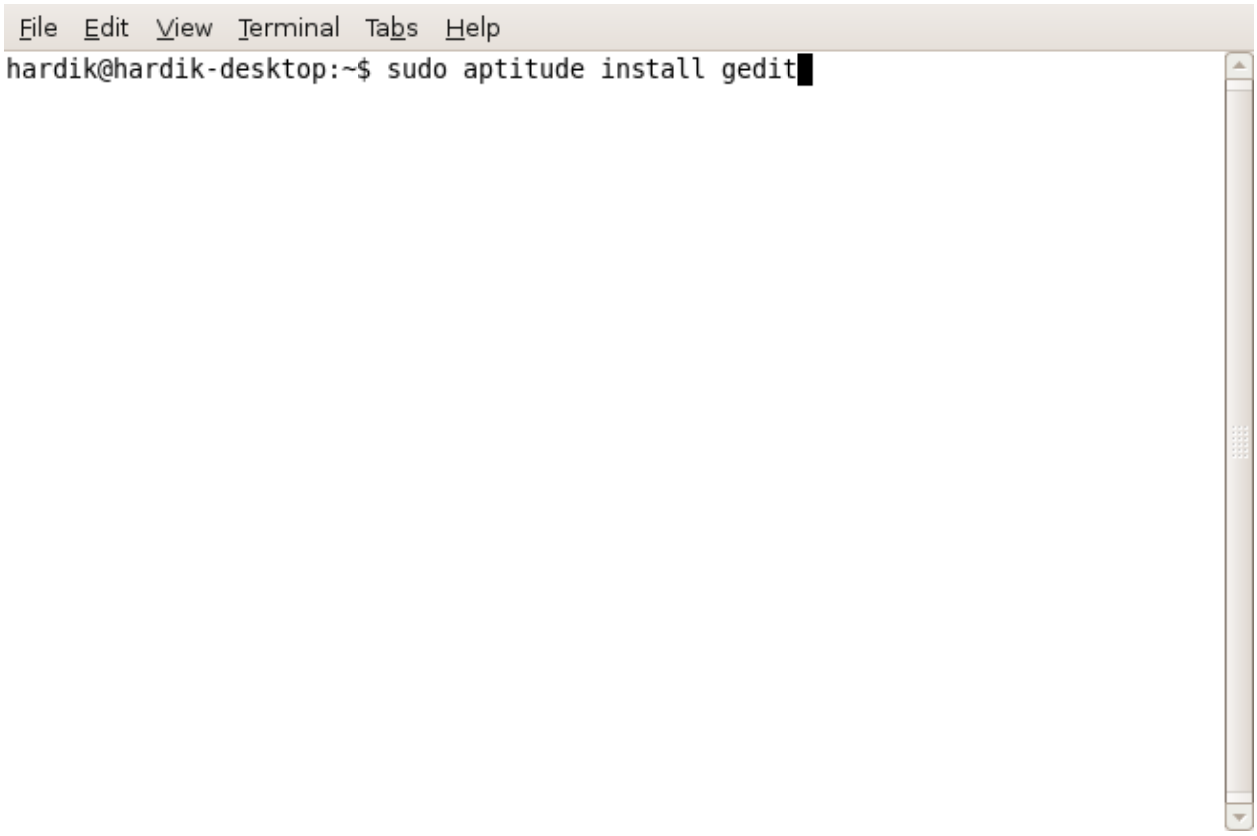
The following section describes the installation of the software packages on Ubuntu Linux.

Text Editor (Gedit Editor)

The Gedit (text editor) might be installed already on your system, since it's the default text editor provided by the distribution. If you find that it is missing you can follow the below mentioned steps.

Installing Gedit Editor:

STEP 1: Open terminal and type: "sudo aptitude install gedit" and press enter



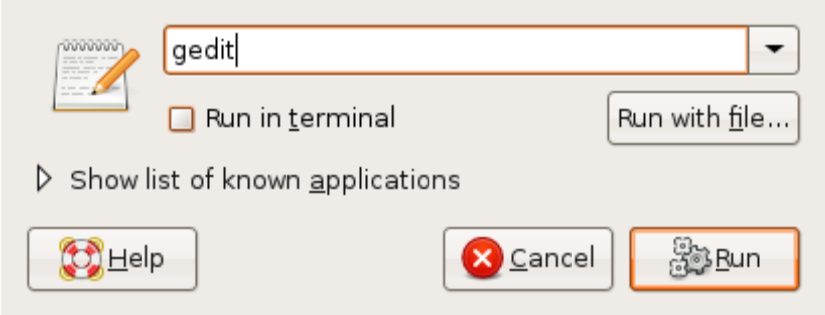
STEP 2: Enter root password.

```
File  Edit  View  Terminal  Tabs  Help
hardik@hardik-desktop:~$ sudo aptitude install gedit
[sudo] password for hardik:
```

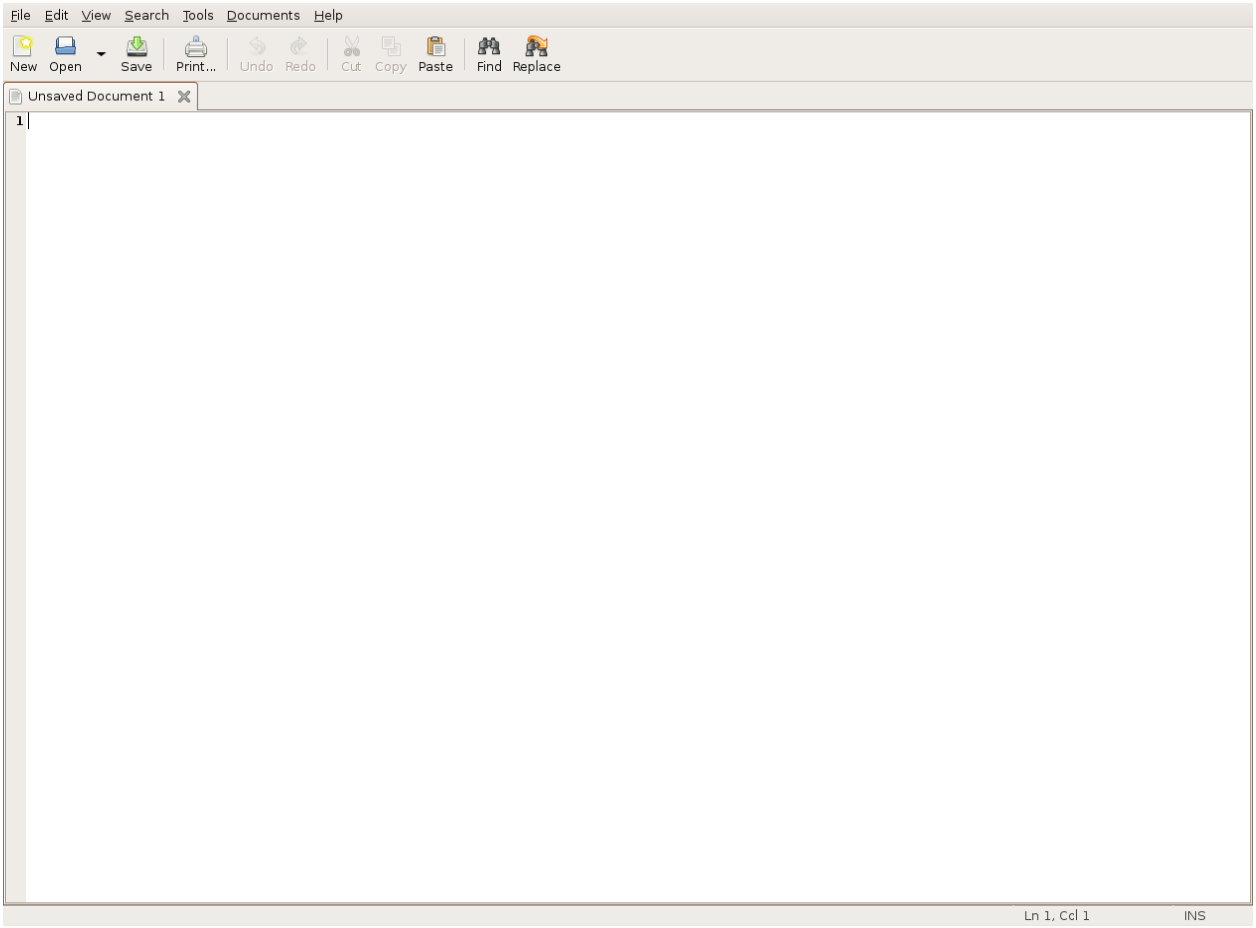
STEP 3: It will show a list of all dependencies installed as shown below

```
File  Edit  View  Terminal  Tabs  Help
The following packages have been kept back:
  app-install-data base-files bind9-host compiz compiz-core compiz-gnome
  compiz-plugins cpp deskbar-applet desktop-file-utils dnsutils dpkg eog
  evince evolution evolution-common evolution-data-server
  evolution-data-server-common evolution-exchange evolution-plugins file
  file-roller firefox firefox-3.0 firefox-3.0-gnome-support
  firefox-gnome-support gcalctool gcc gedit-common gnome-about
  gnome-app-install gnome-applets gnome-applets-data gnome-cards-data
  gnome-desktop-data gnome-games gnome-games-data gnome-keyring gnome-menus
  gnome-panel gnome-panel-data gtk2-engines-murrine
  gtk2-engines-ubuntulooks gtkhtml3.14 guidance-backends gvfs gvfs-backends
  gvfs-fuse hal-info initramfs-tools klibc-utils language-pack-en
  language-pack-en-base language-pack-gnome-en language-pack-gnome-en-base
  language-support-writing-en libbind9-30 libcairo2 libcamel1.2-11
  libdecoration0 libdeskbar-tracker libebook1.2-9 libecal1.2-7
  libedata-book1.2-2 libedata-cal1.2-6 libedataserver1.2-9
  libedataserverui1.2-8 libeel2-2 libeel2-data libegroupwise1.2-13
  libexchange-storage1.2-3 libgdata-google1.2-1 libgdata1.2-1 libgksu2-0
  libglib2.0-0 libgnome-desktop-2 libgnome-keyring0 libgnome-menu2
  libgnomeui-0 libgnomeui-common libgtk-vnc-1.0-0 libgtkhtml3.14-19
  libgvfscommon0 libisc32 libisccc30 libisccfg30 libklibc libldap-2.4-2
  liblircclient0 liblwres30 libmagic1 libnautilus-extension1 libnspr4-0d
  libnss3-1d libntfs-3g23 libpam-gnome-keyring libpanel-applet2-0
  libpango1.0-0 libpango1.0-common libparted1.7-1 libpcre3 libpolkit-gnome0
  libpoppler-glib2 libpoppler2 libpurple0 libsmbclient libssl0.9.8
  libtracker-gtk0 libtrackerclient0 linux-generic linux-headers-generic
  linux-image-generic linux-restricted-modules-generic nautilus
  nautilus-data notification-daemon ntfs-3g nvidia-glx-new
  openoffice.org-base-core openoffice.org-calc openoffice.org-common
  openoffice.org-core openoffice.org-draw openoffice.org-gnome
  openoffice.org-gtk openoffice.org-help-en-gb openoffice.org-help-en-us
  openoffice.org-impress openoffice.org-l10n-common
  openoffice.org-l10n-en-gb openoffice.org-l10n-en-za
  openoffice.org-style-human openoffice.org-writer openssl
  openssl-blacklist parted pciutils pidgin pidgin-data pm-utils
  policykit-gnome poppler-utils python-central python-gmenu
  python-launchpad-bugs python-uno rhythmbox samba-common seahorse
  smbclient tomboy tracker tracker-search-tool transmission-common
  transmission-gtk ttf-opensymbol tzdata ufw vino x11-common xbase-clients
  xorg xserver-xorg xserver-xorg-core xserver-xorg-input-all
  xserver-xorg-video-all xserver-xorg-video-amd xserver-xorg-video-cirrus
  xserver-xorg-video-geode xserver-xorg-video-intel xserver-xorg-video-nsc
  xulrunner-1.9 xulrunner-1.9-gnome-support xutils yelp
0 packages upgraded, 0 newly installed, 0 to remove and 177 not upgraded.
Need to get 0B of archives. After unpacking 0B will be used.
Writing extended state information... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Building tag database... Done
hardik@hardik-desktop:~$
```

STEP 4: Press ALT + F2 and type gedit and press enter to open the Gedit application



STEP 5: The Gedit application will open



Gedit Editor Plug-in (Embedded Terminal):

The Embedded Terminal is a gedit plug-in which fits a terminal in the gedit window through which we can execute Linux commands, compile program using the MAKE utility and download it on to our chip

NOTE: Close all gedit application.

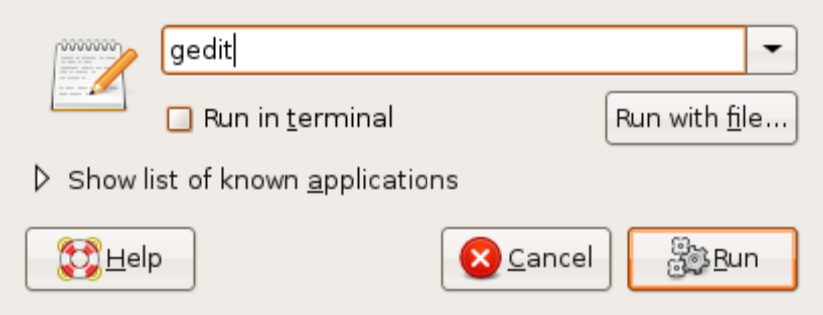
STEP 1: Open terminal and type: “sudo aptitude install gedit-plugins” and press enter

STEP 2: Enter root password.

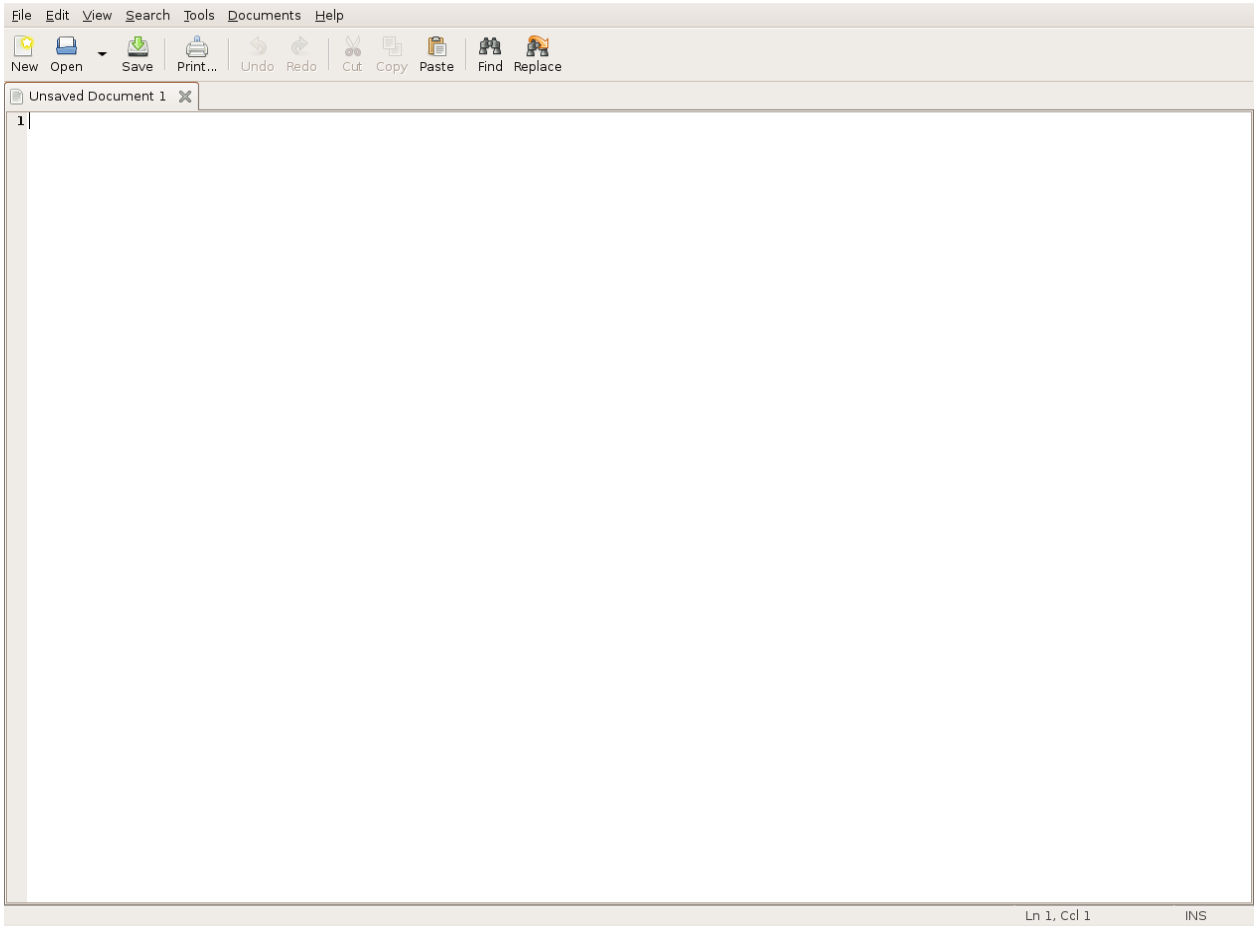
STEP 3: It will show a list of all dependencies installed as shown below

```
File  Edit  View  Terminal  Tabs  Help
Reading state information... Done
Reading extended state information
Initializing package states... Done
Building tag database... Done
hardik@hardik-desktop:~$
hardik@hardik-desktop:~$ sudo aptitude install gedit-plugins
[sudo] password for hardik:
Sorry, try again.
[sudo] password for hardik:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Building tag database... Done
The following packages have been automatically kept back:
  linux-libc-dev linux-restricted-modules-common
The following packages have been kept back:
  app-install-data base-files bind9-host compiz compiz-core compiz-gnome compiz-plugins cpp deskbar-applet desktop-file-utils dnsutils dpkg eog evince
  evolution evolution-common evolution-data-server evolution-data-server-common evolution-exchange evolution-plugins file file-roller firefox
  firefox-3.0 firefox-3.0-gnome-support firefox-gnome-support gcalctool gcc gedit-common gnome-about gnome-app-install gnome-applets gnome-applets-data
  gnome-cards-data gnome-desktop-data gnome-games gnome-games-data gnome-keyring gnome-menus gnome-panel gnome-panel-data gtk2-engines-murrine
  gtk2-engines-ubuntulooks gtkhtml3.14 guidance-backends gvfs gvfs-backends gvfs-fuse hal-info initramfs-tools klibc-utils language-pack-en
  language-pack-en-base language-pack-gnome-en language-pack-gnome-en-base language-support-writing-en libbind9-30 libcairo2 libcamel1.2-11
  libdecoration0 libdeskbar-tracker libebook1.2-9 libecal1.2-7 libedata-book1.2-2 libedata-cal1.2-6 libedataserver1.2-9 libedataserverui1.2-8 libeel2-2
  libeel2-data libegroupwise1.2-13 libexchange-storage1.2-3 libgdata-google1.2-1 libgdata1.2-1 libgksu2-0 libglib2.0-0 libgnome-desktop-2
  libgnome-keyring0 libgnome-menu2 libgnomeui-0 libgnomeui-common libgtk-vnc-1.0-0 libgtkhtml3.14-19 libgvfscommon0 libisc32 libisccc30 libisccfg30
  libklibc libldap-2.4-2 liblircclient0 liblwres30 libmagic1 libnautilus-extension1 libnspr4-0d libnss3-1d libntfs-3g23 libpam-gnome-keyring
  libpanel-applet2-0 libpango1.0-0 libpango1.0-common libparted1.7-1 libpcre3 libpolkit-gnome0 libpoppler-glib2 libpoppler2 libpurple0 libsmbclient
  libssl0.9.8 libtracker-gtk0 libtrackerclient0 linux-generic linux-headers-generic linux-image-generic linux-restricted-modules-generic nautilus
  nautilus-data notification-daemon ntfs-3g nvidia-glx-new openoffice.org-base-core openoffice.org-calc openoffice.org-common openoffice.org-core
  openoffice.org-draw openoffice.org-gnome openoffice.org-gtk openoffice.org-help-en-gb openoffice.org-help-en-us openoffice.org-impress
  openoffice.org-l10n-common openoffice.org-l10n-en-gb openoffice.org-l10n-en-za openoffice.org-style-human openoffice.org-writer openssl
  openssl-blacklist parted pciutils pidgin pidgin-data pm-utils policykit-gnome poppler-utils python-central python-gmenu python-launchpad-bugs
  python-uno rhythmbox samba-common seahorse smbclient tomboy tracker tracker-search-tool transmission-common transmission-gtk ttf-opensymbol tzdata
  ufw vino x11-common xbase-clients xorg xserver-xorg xserver-xorg-core xserver-xorg-input-all xserver-xorg-video-all xserver-xorg-video-amd
  xserver-xorg-video-cirrus xserver-xorg-video-geode xserver-xorg-video-intel xserver-xorg-video-nsc xulrunner-1.9 xulrunner-1.9-gnome-support xutils
  yelp
The following NEW packages will be installed:
  gedit-plugins
0 packages upgraded, 1 newly installed, 0 to remove and 177 not upgraded.
Need to get 300kB of archives. After unpacking 1810kB will be used.
Writing extended state information... Done
Err ftp://ftp.iitb.ac.in hardy/universe gedit-plugins 2.22.2-0ubuntu1
  Connection timeout
E: Failed to fetch ftp://ftp.iitb.ac.in/os/ubuntu/archives/pool/universe/g/gedit-plugins/gedit-plugins_2.22.2-0ubuntu1_i386.deb: Connection timeout
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Building tag database... Done
hardik@hardik-desktop:~$
```

STEP 4: Press ALT + F2 and type gedit and press enter to open the Gedit application

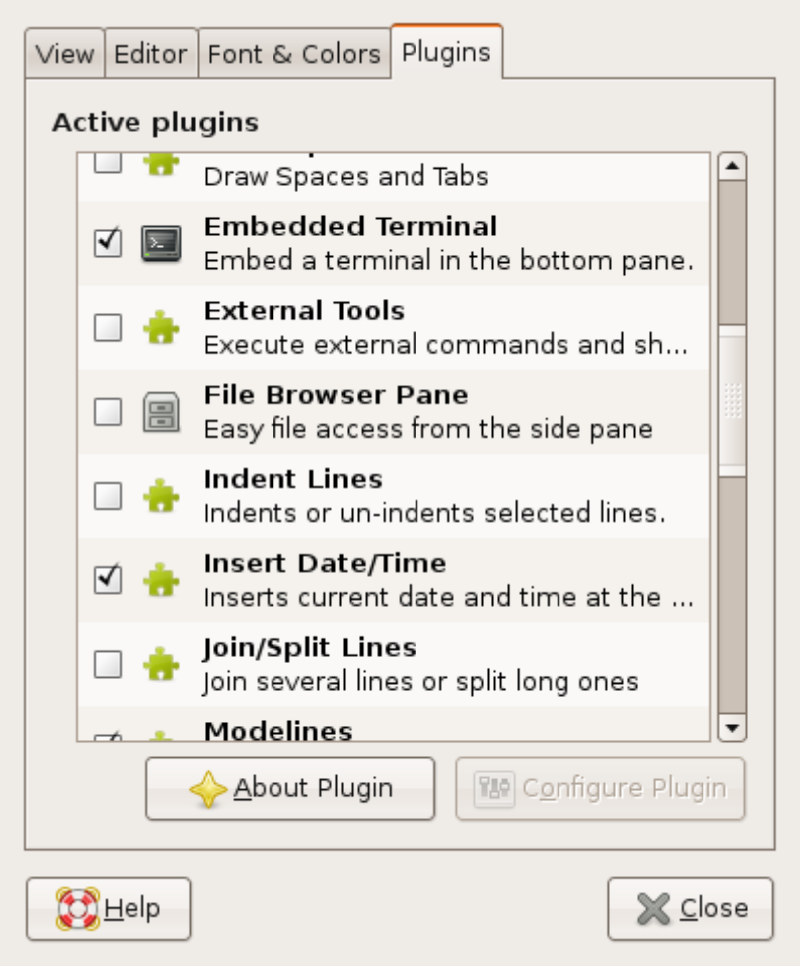


STEP 5: The Gedit application will open.

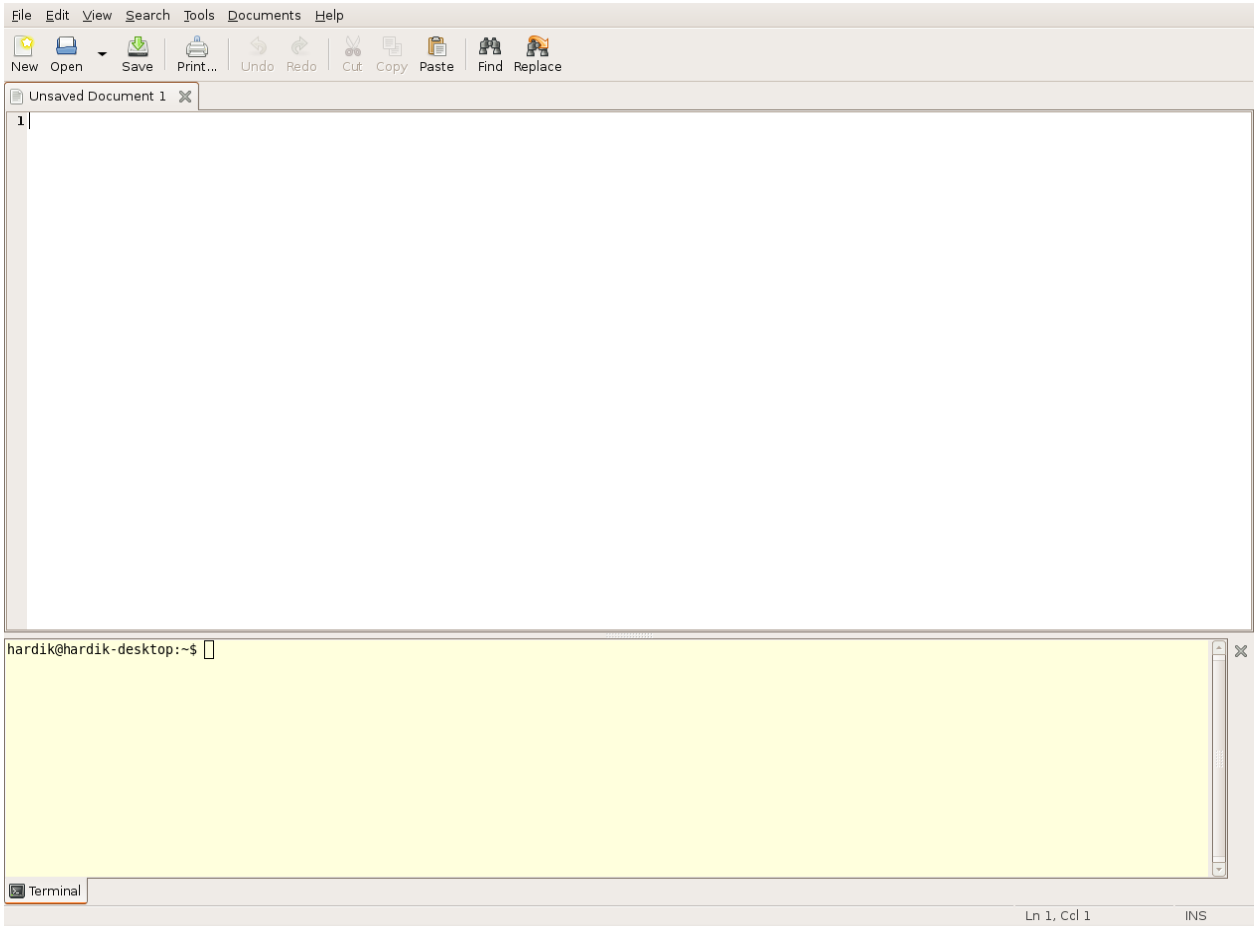


STEP 6: Then in gedit open Edit menu > preferences and goto plugins tab

STEP 7: Select Embedded Terminal and press close button.

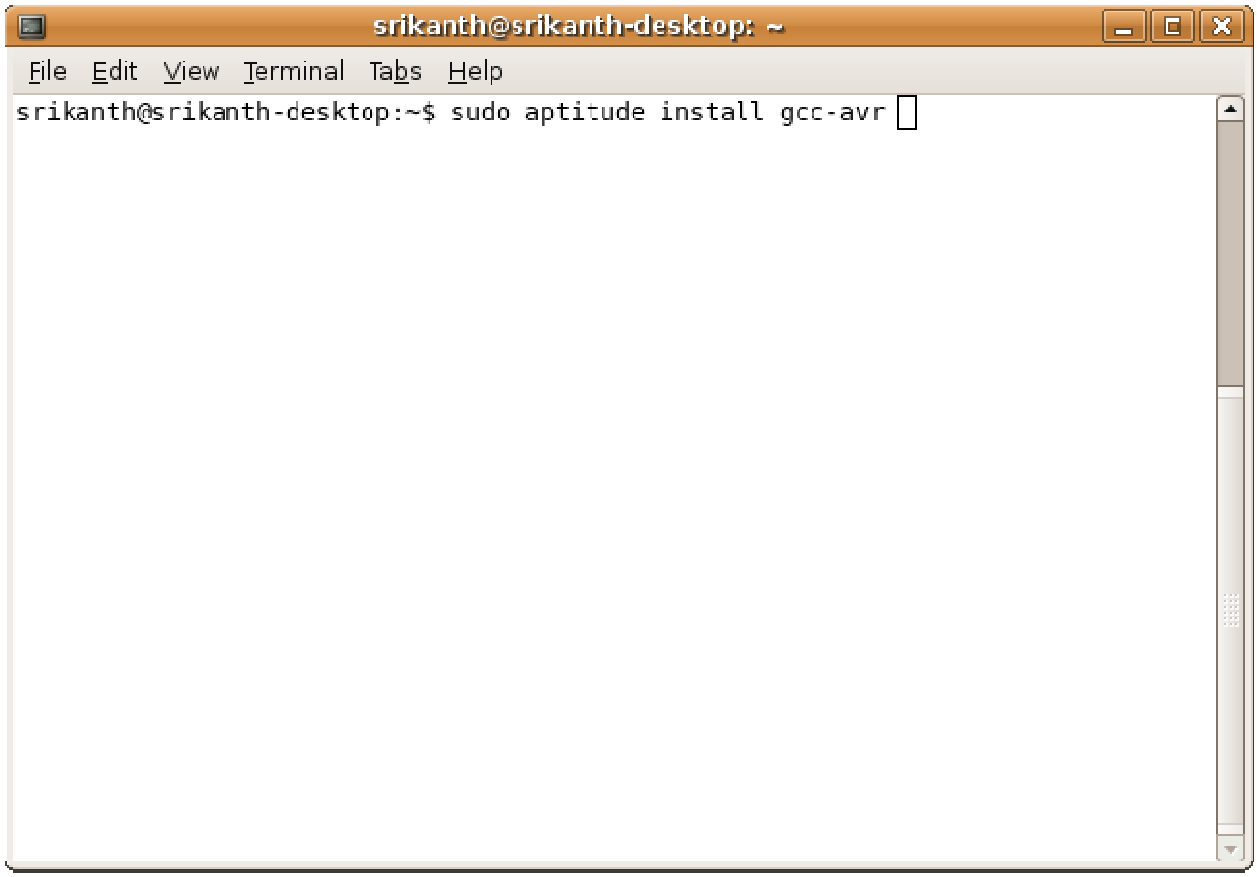


STEP 8: Go to the View menu and select Bottom pane



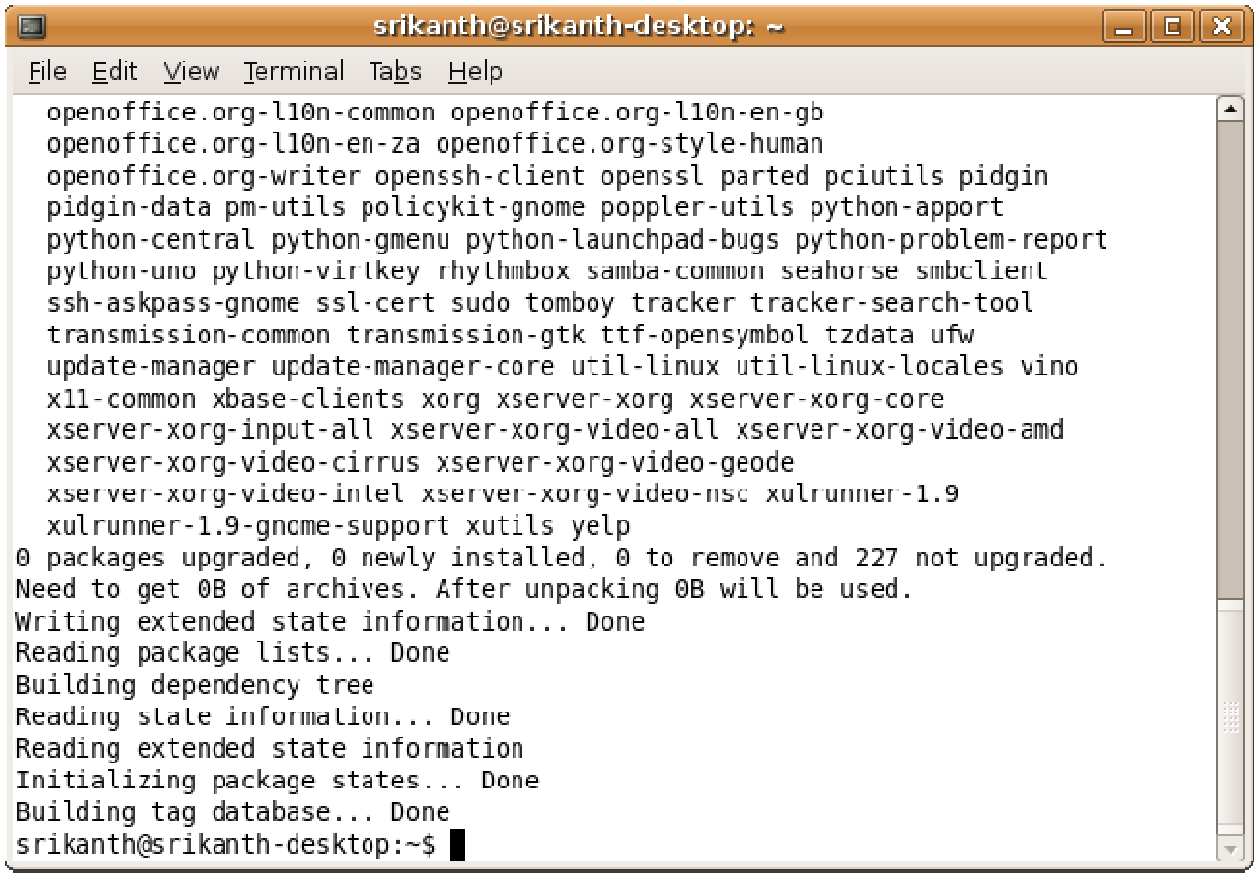
Compiler (avr-gcc)

STEP 1: Open terminal and type: “sudo aptitude install gcc-avr” and press enter



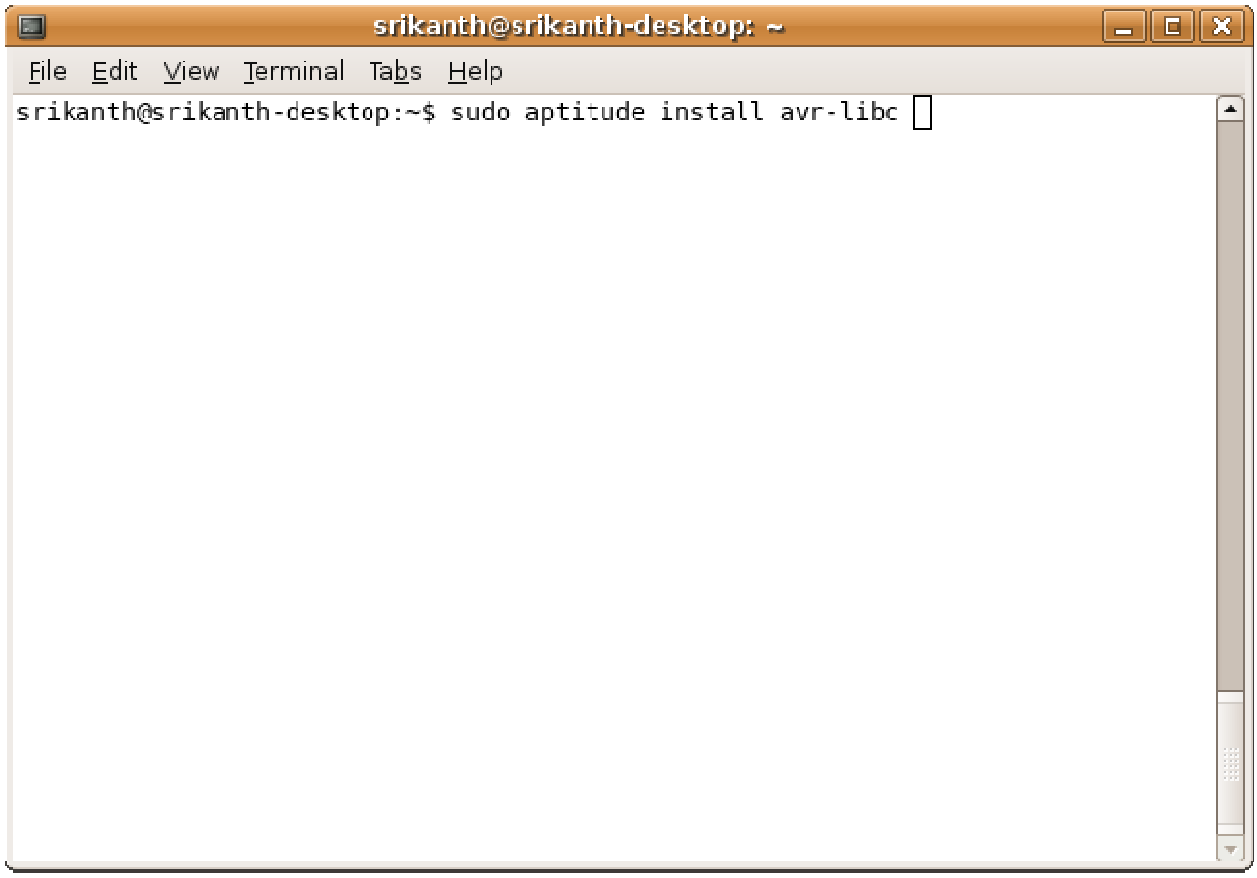
STEP 2: Enter root password.

STEP 3: It will show a list of all dependencies install as shown below.



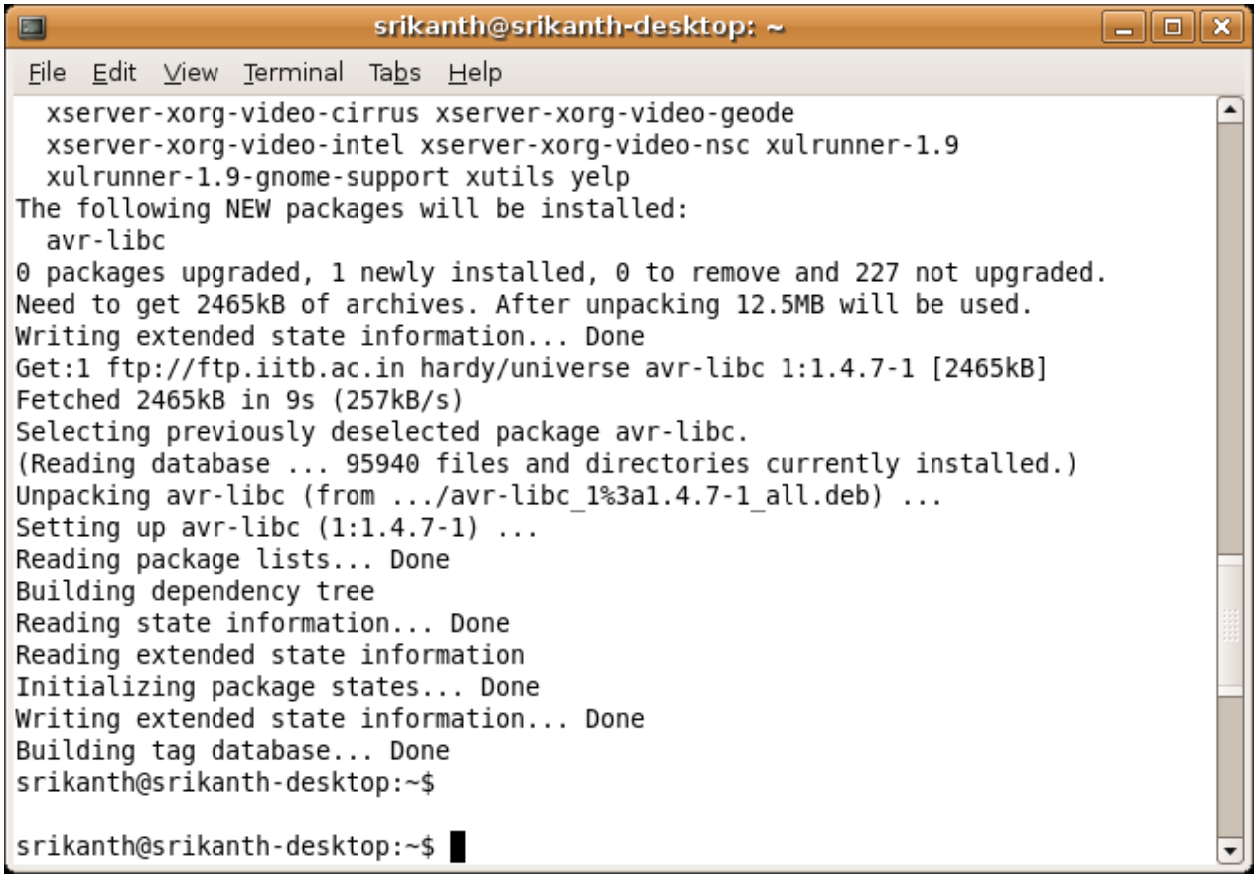
avr-libc (Standard C library for Atmel AVR development) :

STEP 1: Open terminal and type: “sudo aptitude install avr-libc” and press enter



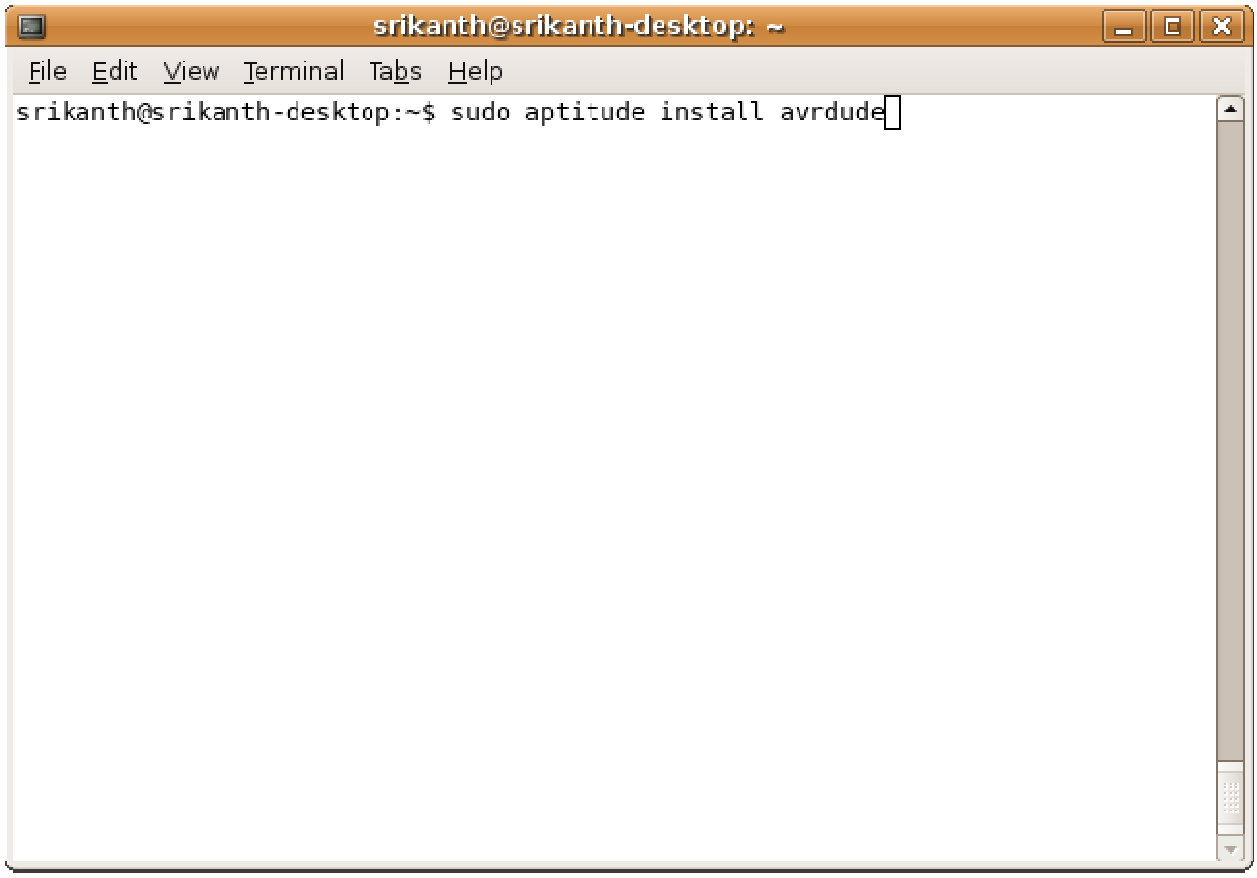
STEP 2: Enter root password

STEP 3: It will show a list of all dependencies install as shown below



avrdude (software for programming Atmel AVR microcontrollers):

STEP 1: Open terminal and type: “sudo aptitude install avrdude” and press enter



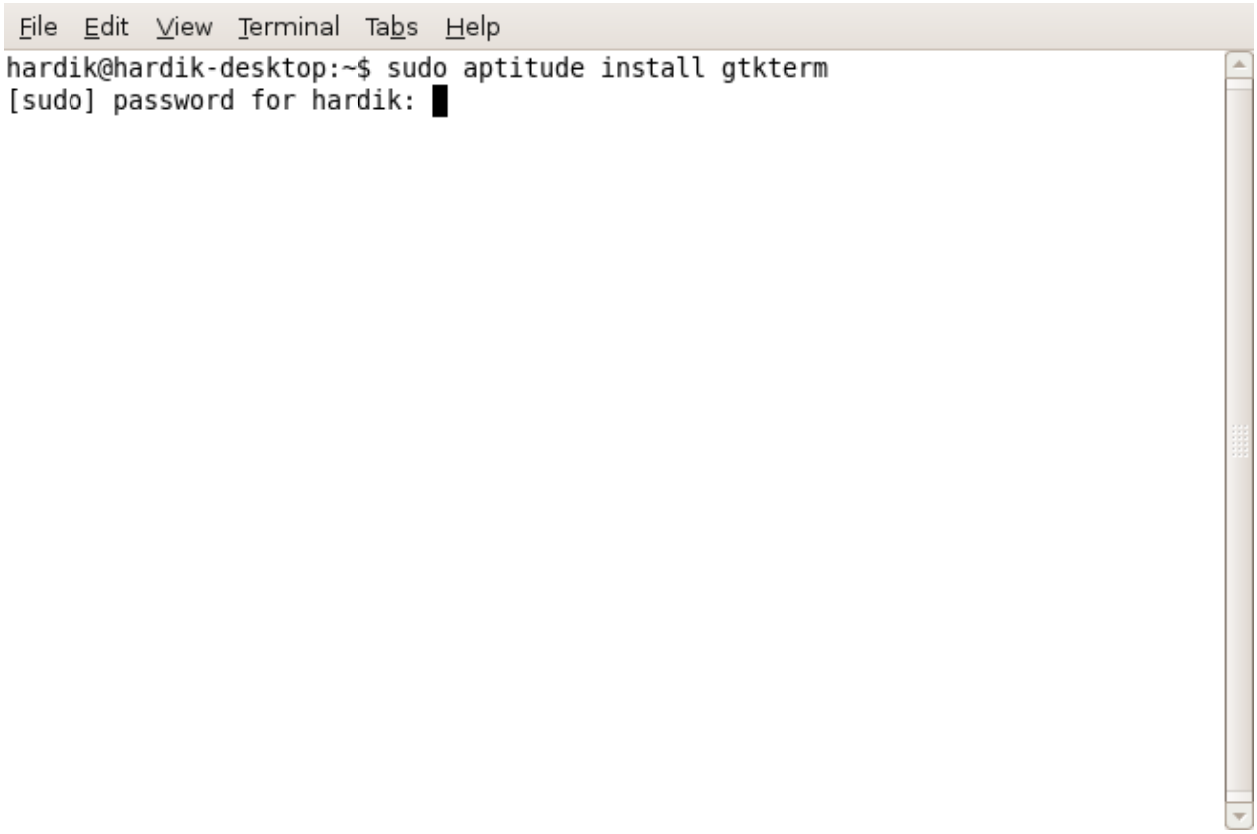
STEP 2: Enter root password

STEP 3: It will show a list of all dependencies install as shown below

Serial port terminal (Gtkterm)

The Gtkterm is analogous to Windows HyperTerminal which is useful for serial commutations between PC and uNiBoard. It is a VT102 compatible serial terminal.

STEP 1: Open terminal and type: “sudo aptitude install gtkterm” and press enter



STEP 2: Enter root password.

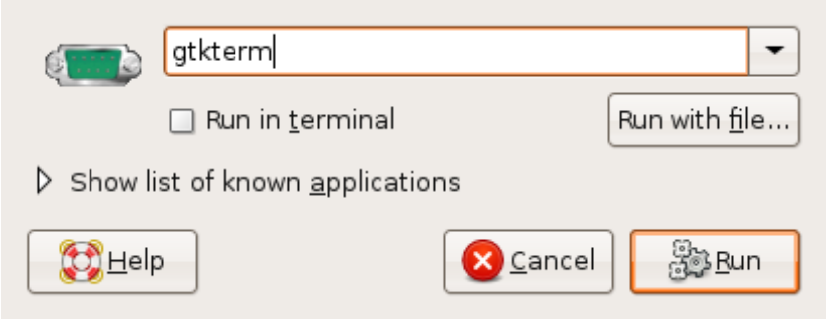
STEP 3: It will show a list of all dependencies installed as shown below.

```
File  Edit  View  Terminal  Tabs  Help
  xorg xserver-xorg xserver-xorg-core xserver-xorg-input-all
  xserver-xorg-video-all xserver-xorg-video-amd xserver-xorg-video-cirrus
  xserver-xorg-video-geode xserver-xorg-video-intel xserver-xorg-video-nsc
  xulrunner-1.9 xulrunner-1.9-gnome-support xutils yelp
The following NEW packages will be installed:
  gtkterm
0 packages upgraded, 1 newly installed, 0 to remove and 177 not upgraded.
Need to get 54.2kB of archives. After unpacking 258kB will be used.
Writing extended state information... Done
Get:1 ftp://ftp.iitb.ac.in hardy/universe gtkterm 0.99.5-1ubuntu2 [54.2kB]
Fetched 54.2kB in 5s (10.2kB/s)
Selecting previously deselected package gtkterm.
(Reading database ... 121820 files and directories currently installed.)
Unpacking gtkterm (from .../gtkterm_0.99.5-1ubuntu2_i386.deb) ...
Setting up gtkterm (0.99.5-1ubuntu2) ...

Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Writing extended state information... Done
Building tag database... Done
hardik@hardik-desktop:~$
```

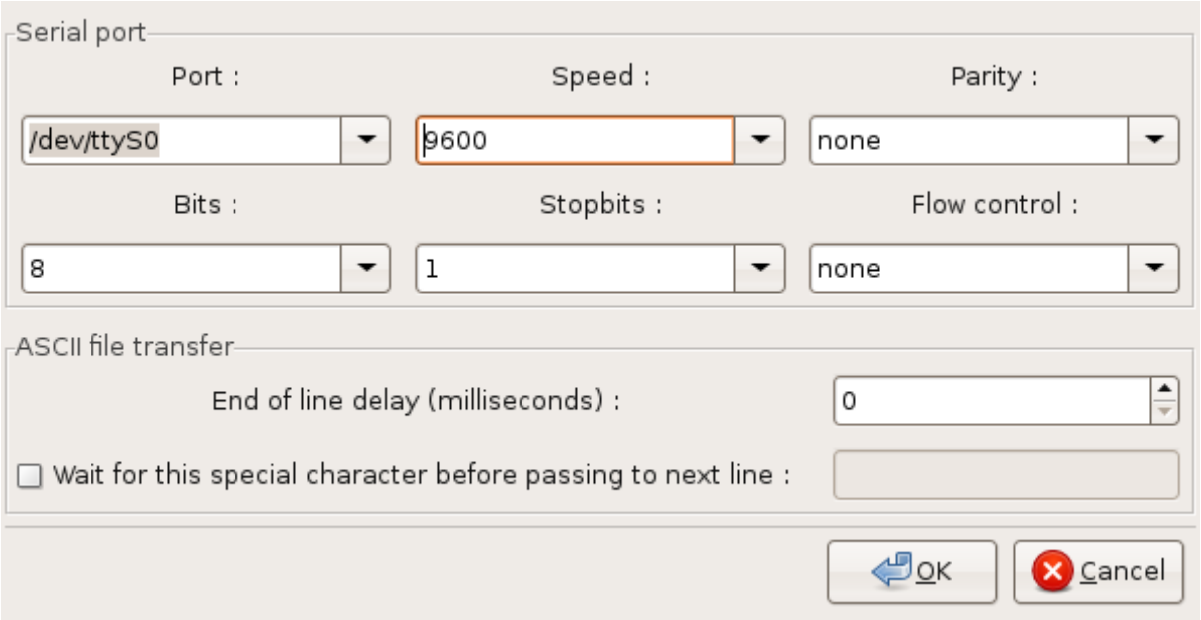
STEP 4: Press ALT + F2 and type gtkterm and press enter to open the Gtkterm application

Gtkterm Configurations for setting the Baud rate, Parity, Stop bits

STEP 1: Press ALT + F2 and type gtkterm and press enter to open the Gtkterm application and click on Run



STEP 2: Go to Configuration Menu and choose Port



STEP 3: Select the Port: /dev/ttyS0 if you have connected to the serial communication port, if not change the Port.

STEP 4: Select the Speed to the Baud rate as per in your code, you have configured for the UART.

STEP 5: Select the Parity as per in your code, you have configured for the UART.

STEP 6: Select the Stop bits as per in your code, you have configured for the UART.

STEP 7: Select the Bits as per in your code, you have configured for the UART.

STEP 8: Select the Flow control to none.

STEP 9: Click on OK

Software Installations for Windows OS

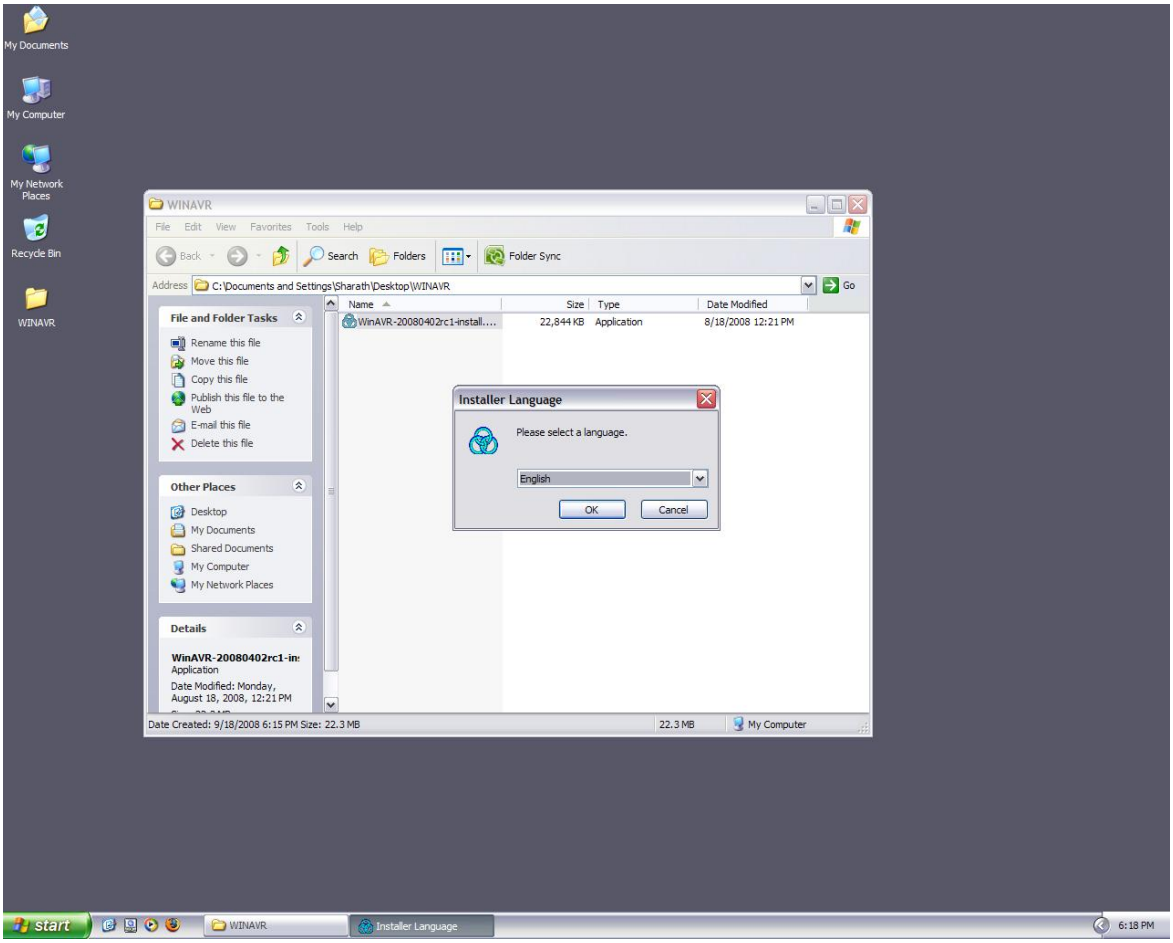
WinAVR (Includes avr-gcc, avr-binutils, avrdude)

Copy the software packages from the uNiBoard contents on CD or download latest version of WinAVR from website given below:

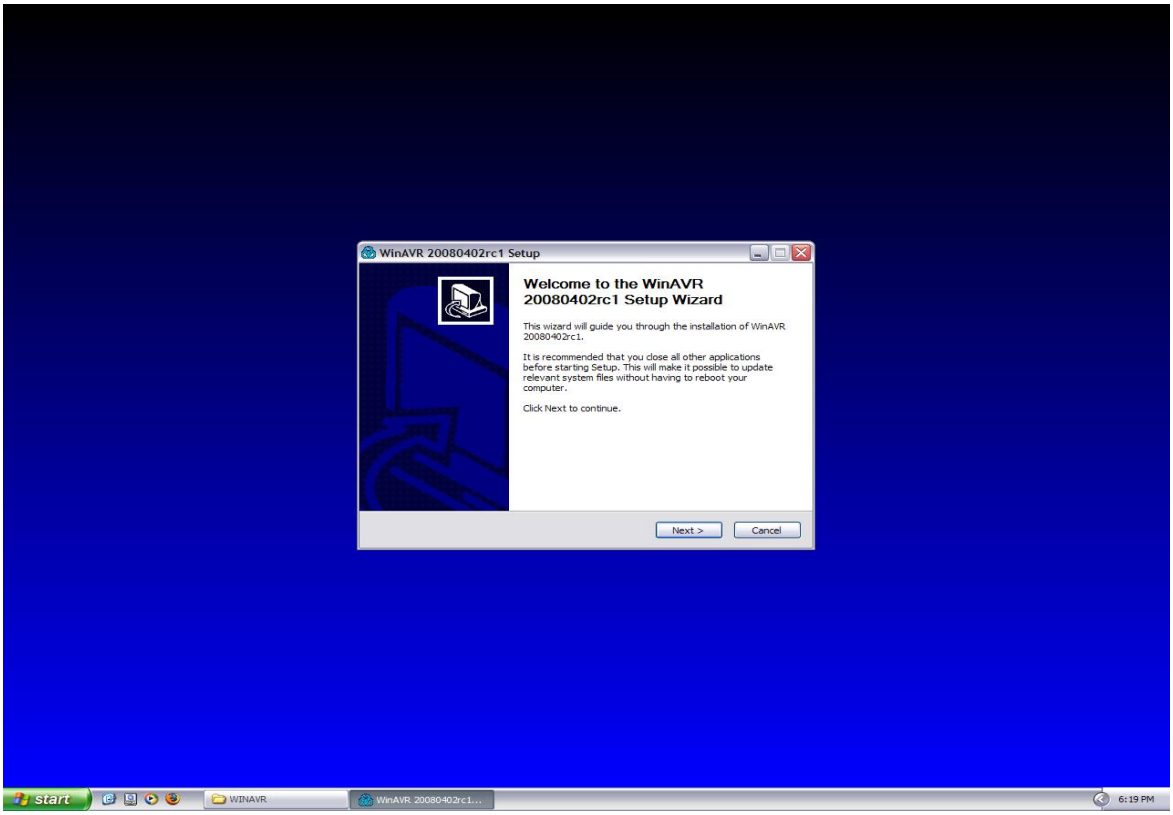
http://winavr.sourceforge.net/

After you have this package follow the steps given below:

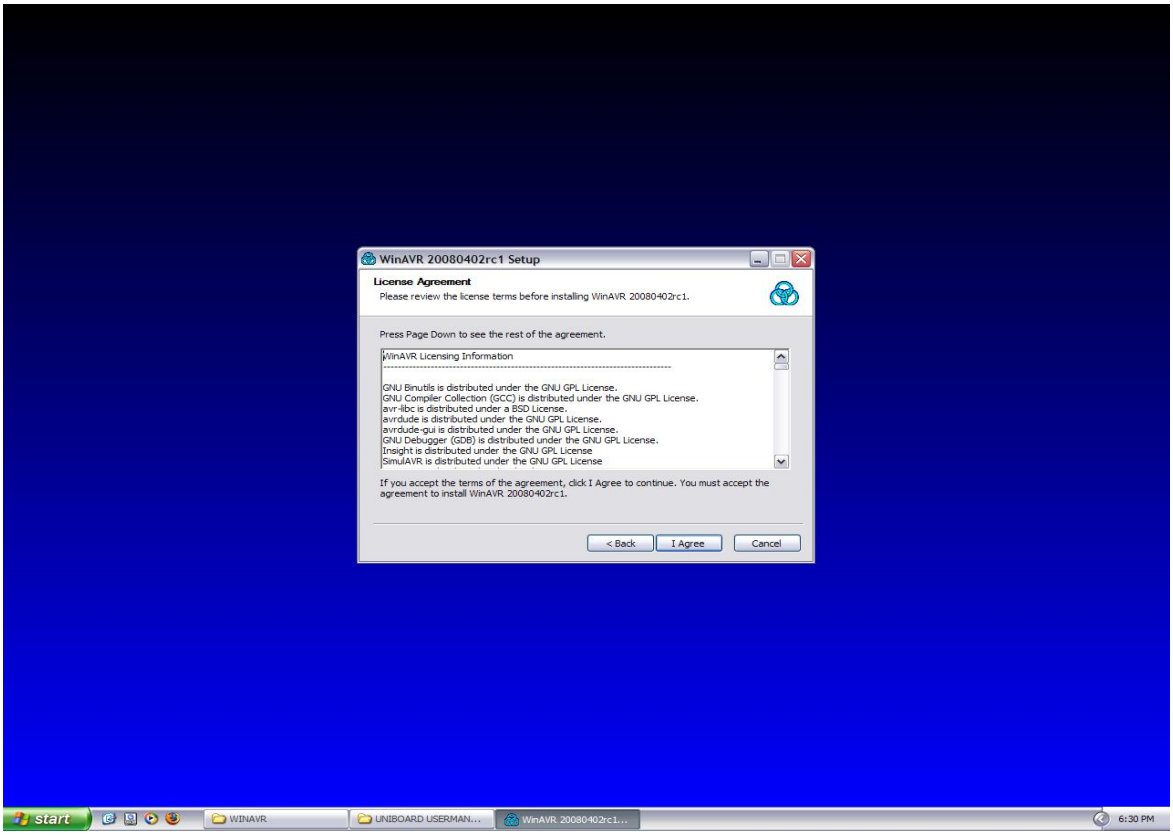
Step 1: Run the file to install the package.



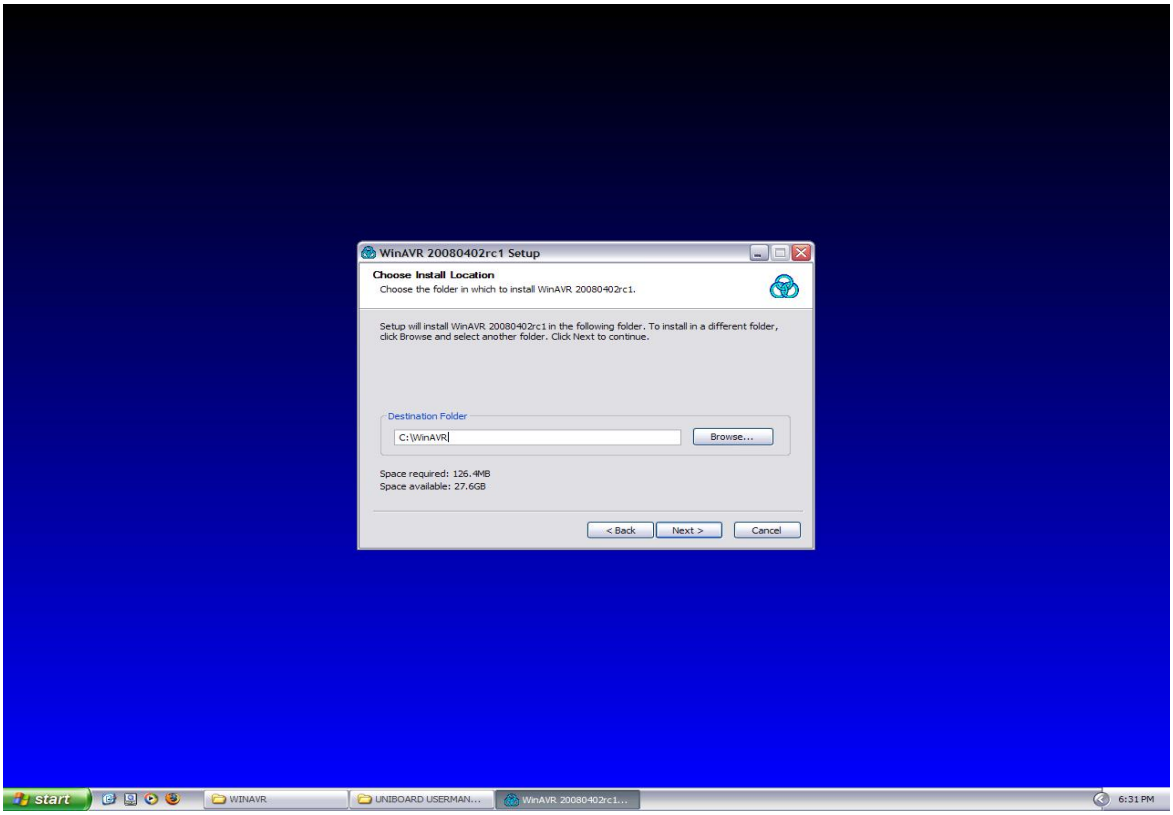
Step 2: Select the language and press OK.



Step 3: click on Next



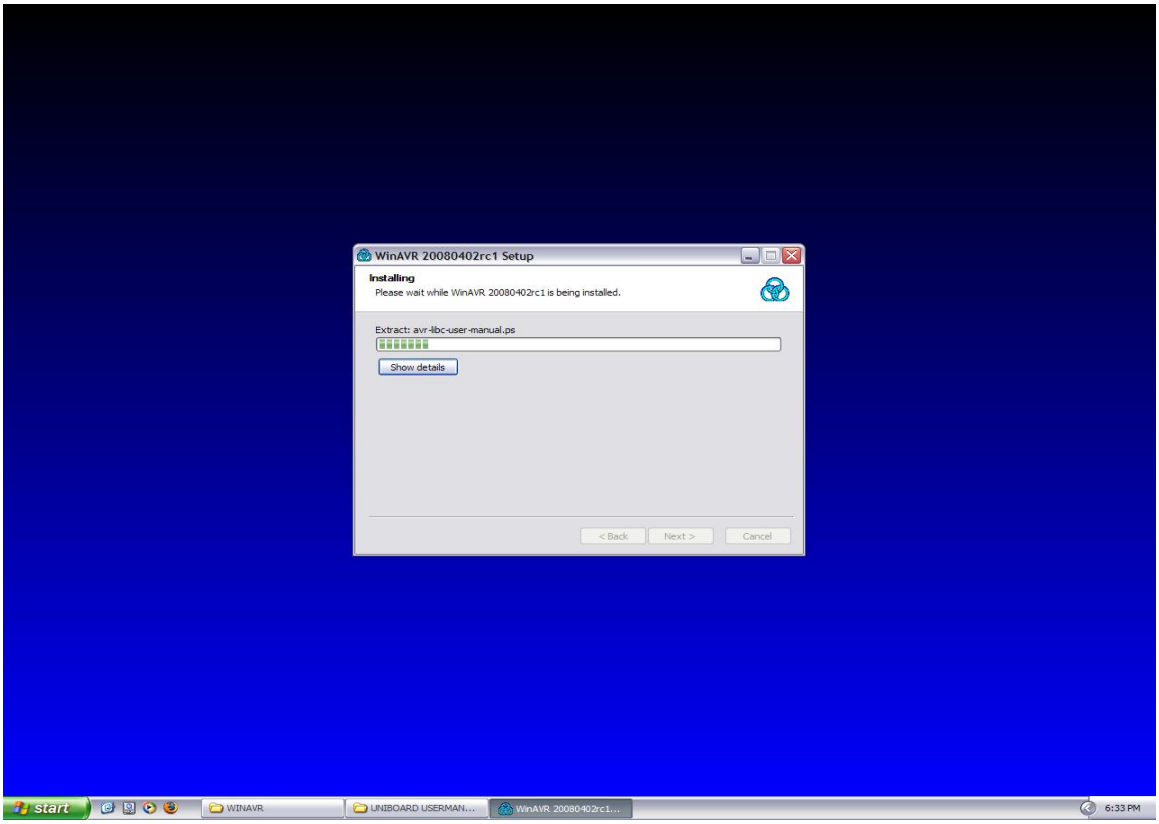
Step 4: Click on I Agree

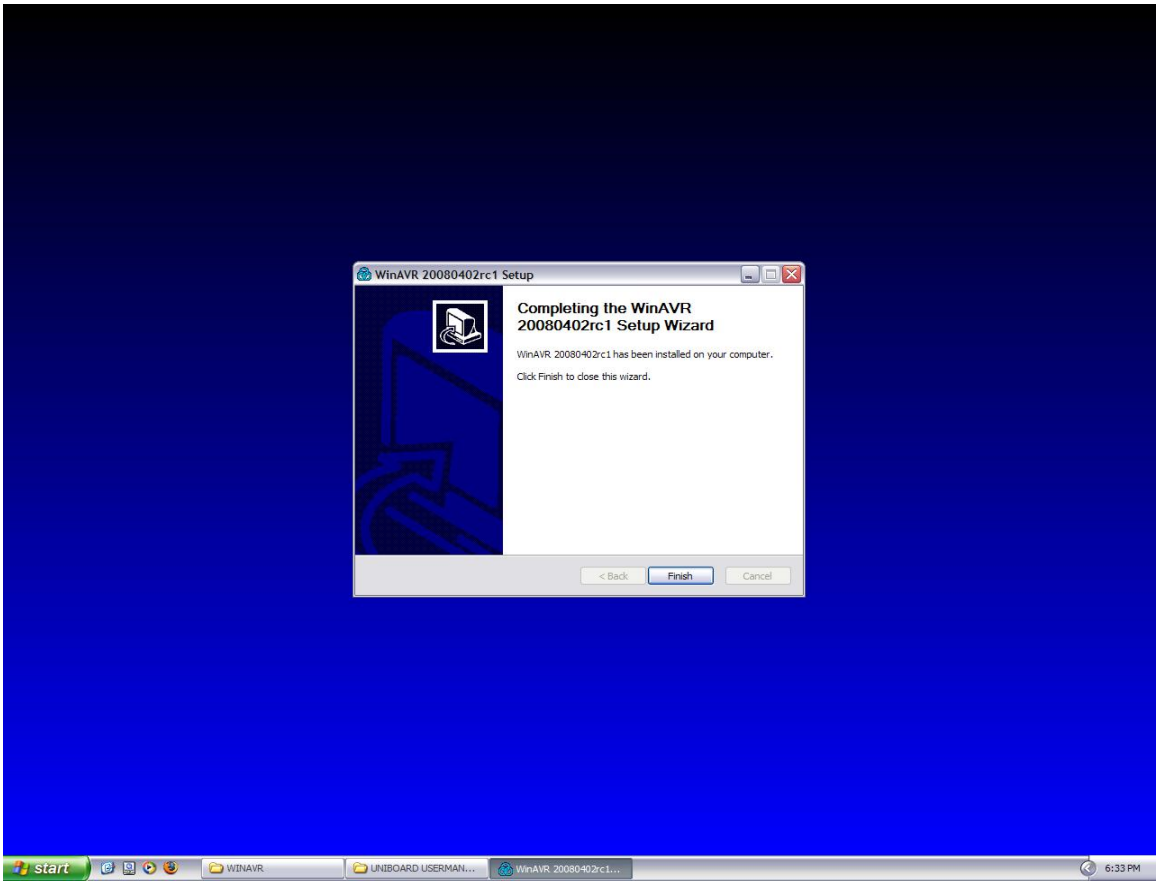


Step 5: Type the destination Folder where you want to install WinAVR and click on Next.

Step 6: Check on all i.e. Install files, Add Directories to path, and Programmer's Notepad and click on Install.

Step 7: It will display as shown below and then click on finish


WinAVR 20080402rc1 Setup
Installing
Please wait while WinAVR 20080402rc1 is being installed.
Extract: avr-libc-user-manual.ps
Show details
< Back
Next >
Cancel
start
WINAVR
UNIBOARD USERMAN...
WinAVR 20080402rc1...
6:33 PM

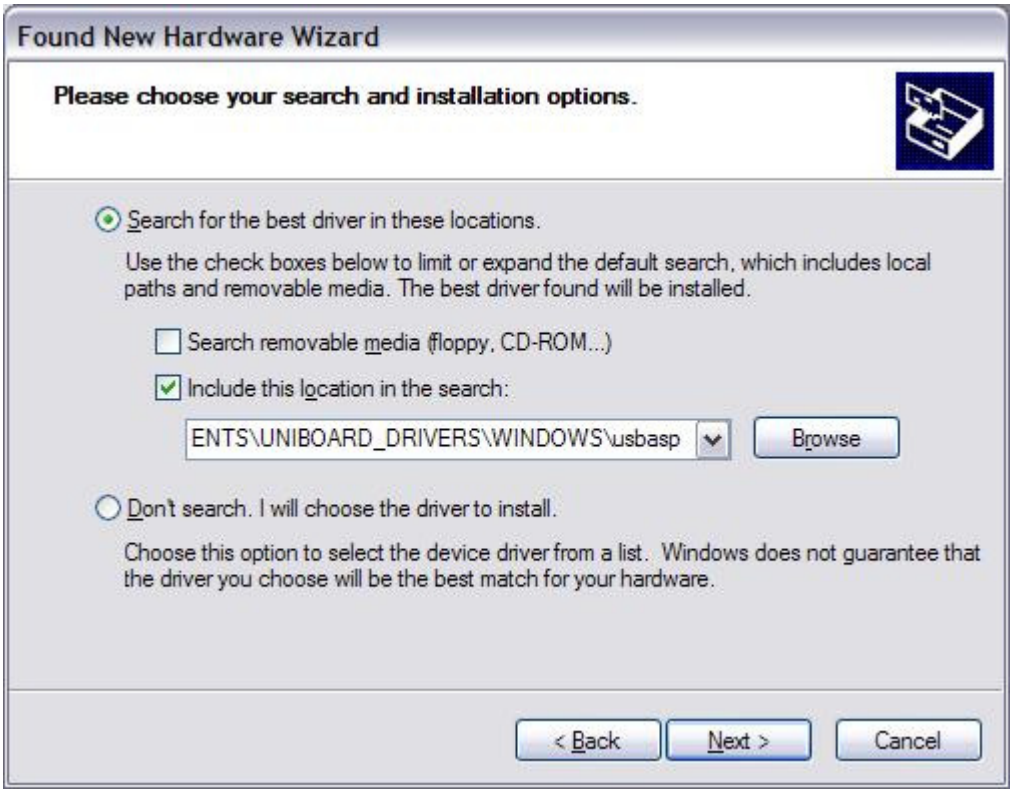

WinAVR 20080402rc1 Setup
Completing the WinAVR 20080402rc1 Setup Wizard
WinAVR 20080402rc1 has been installed on your computer.
Click Finish to close this wizard.
< Back
Finish
Cancel
start
WINAVR
UNIBOARD USERMAN...
WinAVR 20080402rc1...
6:33 PM

Installing USB drivers for uNiBoard

STEP 1: Insert the uNiBoard USB cable to PC.

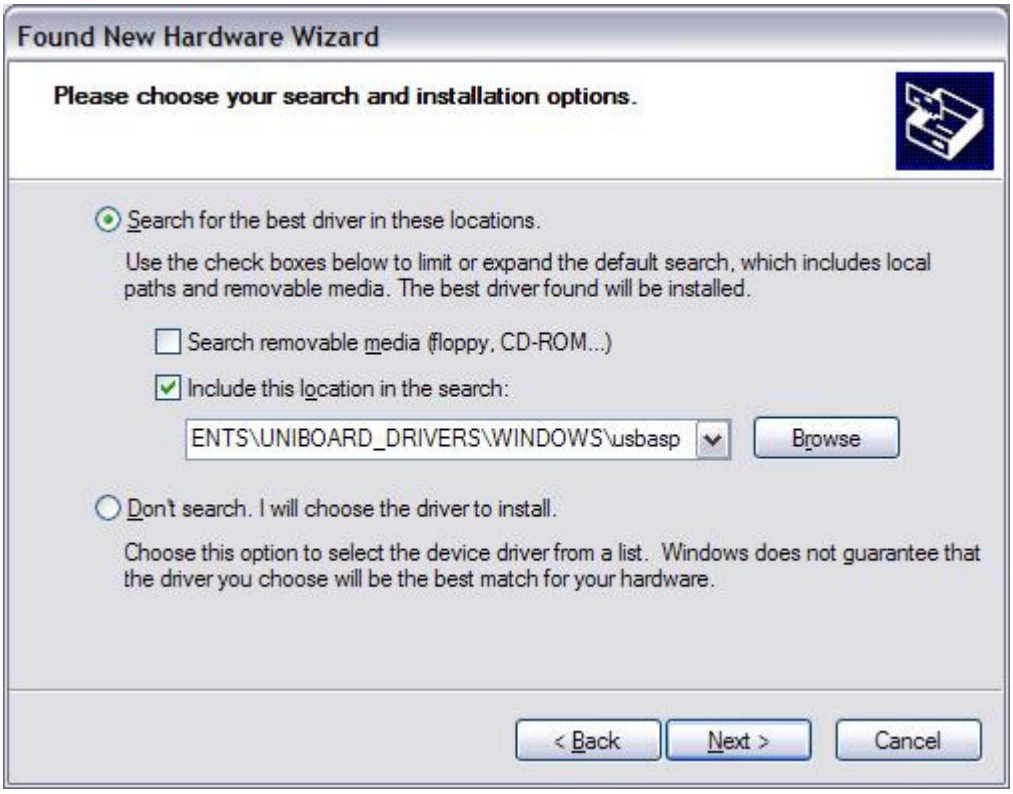
STEP 2: If the drivers are not installed then it will display the installation wizard as shown below.

STEP 3: Select the options as install from a list or specific location (Advanced) and click on next. It will display options as shown below

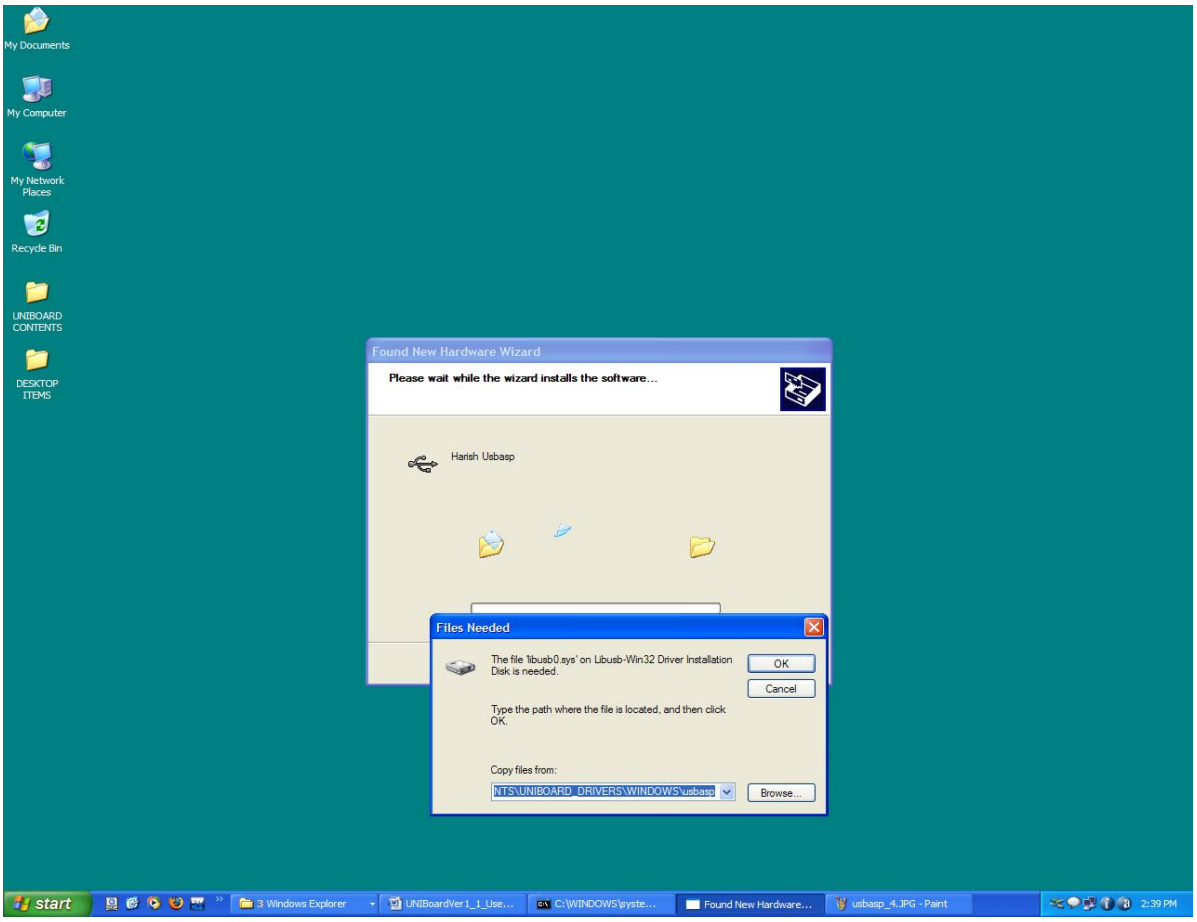


STEP 4: Click on Browse and select the drivers from the CD UNIBOARD CONTENTS directory path as \UNIBOARD CONTENTS\UNIBOARD_DRIVERS\WINDOWS\usbasp and click on OK

STEP 5: Click on Next.



STEP 6: Give the same path of drivers and click on OK



STEP 7: Click on Finish.

After the drivers are installed you are ready to program the Board.

Programming the Board

NOTE: Refer to the above sections to check that you are having all the required software tools.

Getting Started on Linux

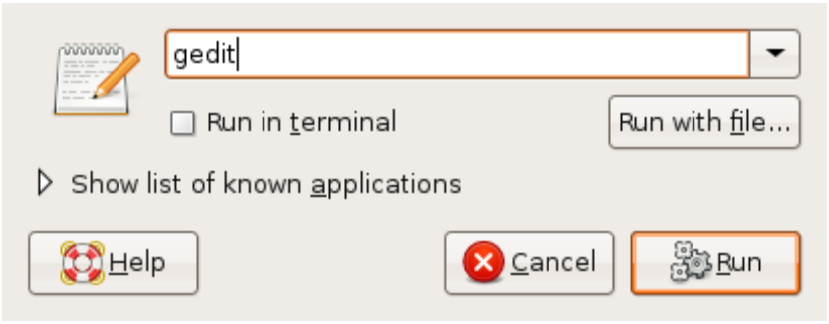
For programming the Board you must first have the hex file to be loaded into the flash section of the microcontroller. For generating the hex file you have to write an error free code, compile your code and compiler will generate hex file. Then using USBasp firmware on Atmega8 microcontroller you can program the Atmega128 microcontroller.

The following section will guide you on how to program using uNiBoard.

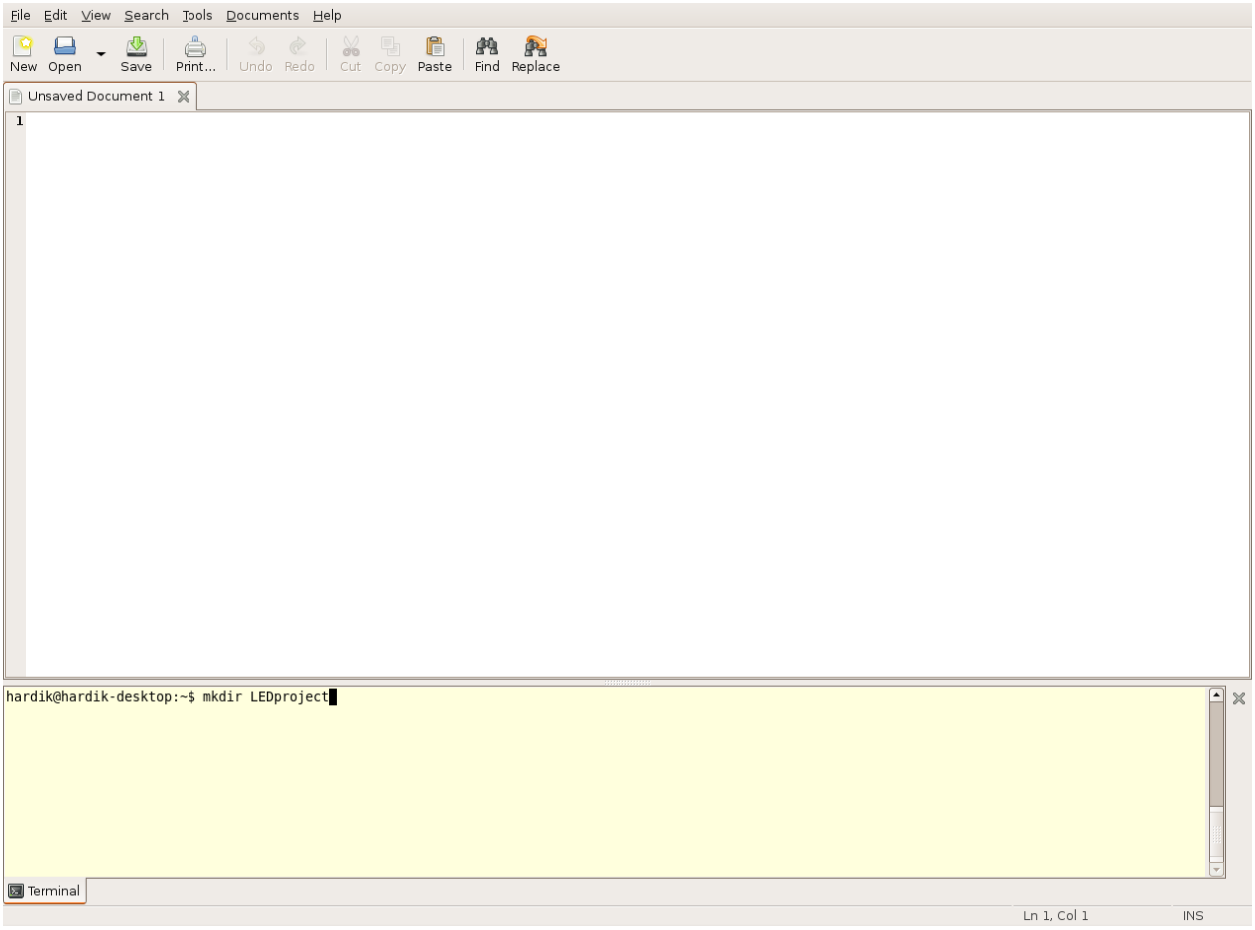
1. We will write a first program to test LEDs.

   As you know LEDs can be connected various test PORTs on UNiBoard using FRC cable (refer to chapter 5), we will use the General purpose **PORTC** as output and connect the FRC cable between PORTC and the LED PORT.

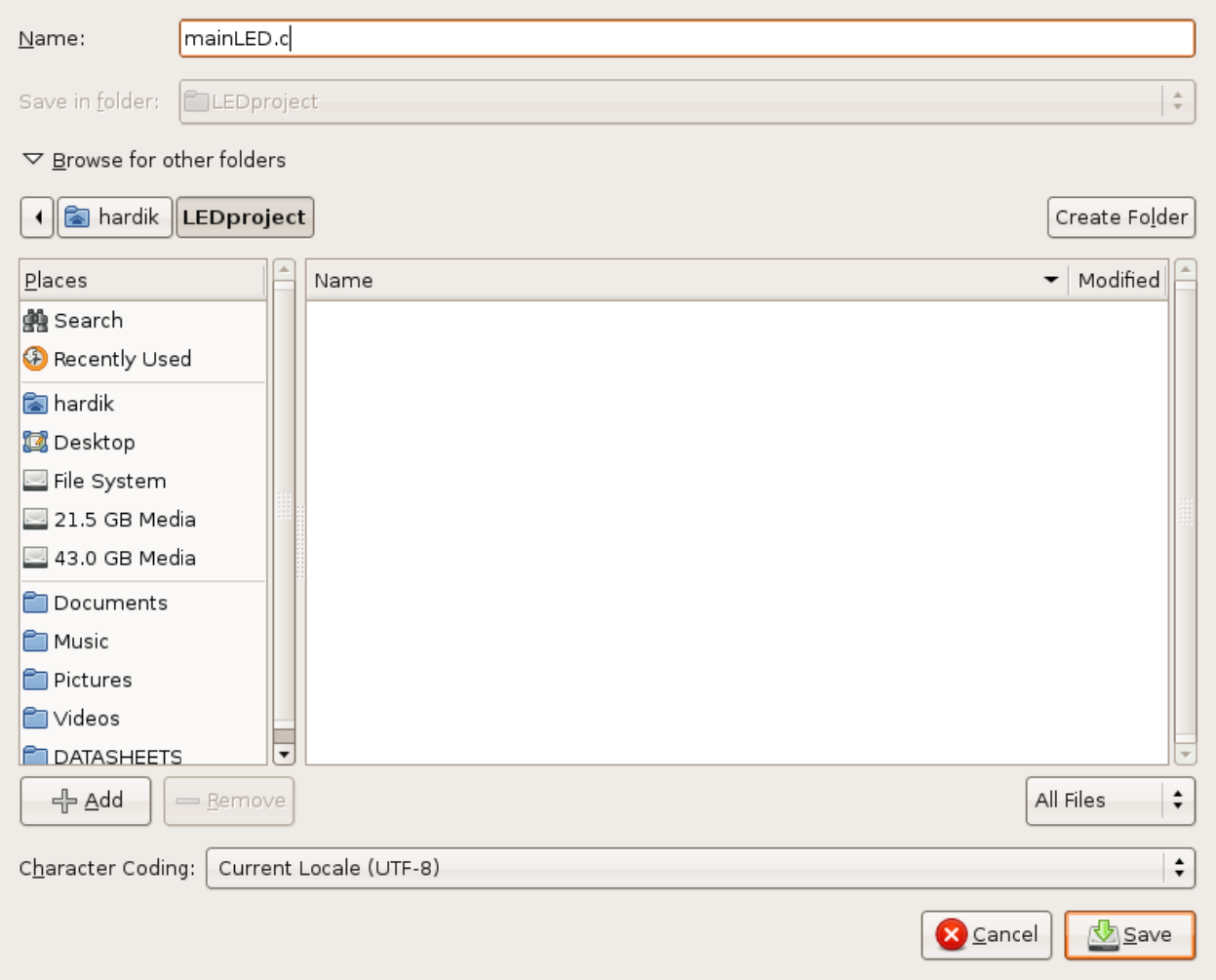
   STEP 1: Open Gedit (refer to chapter 3)



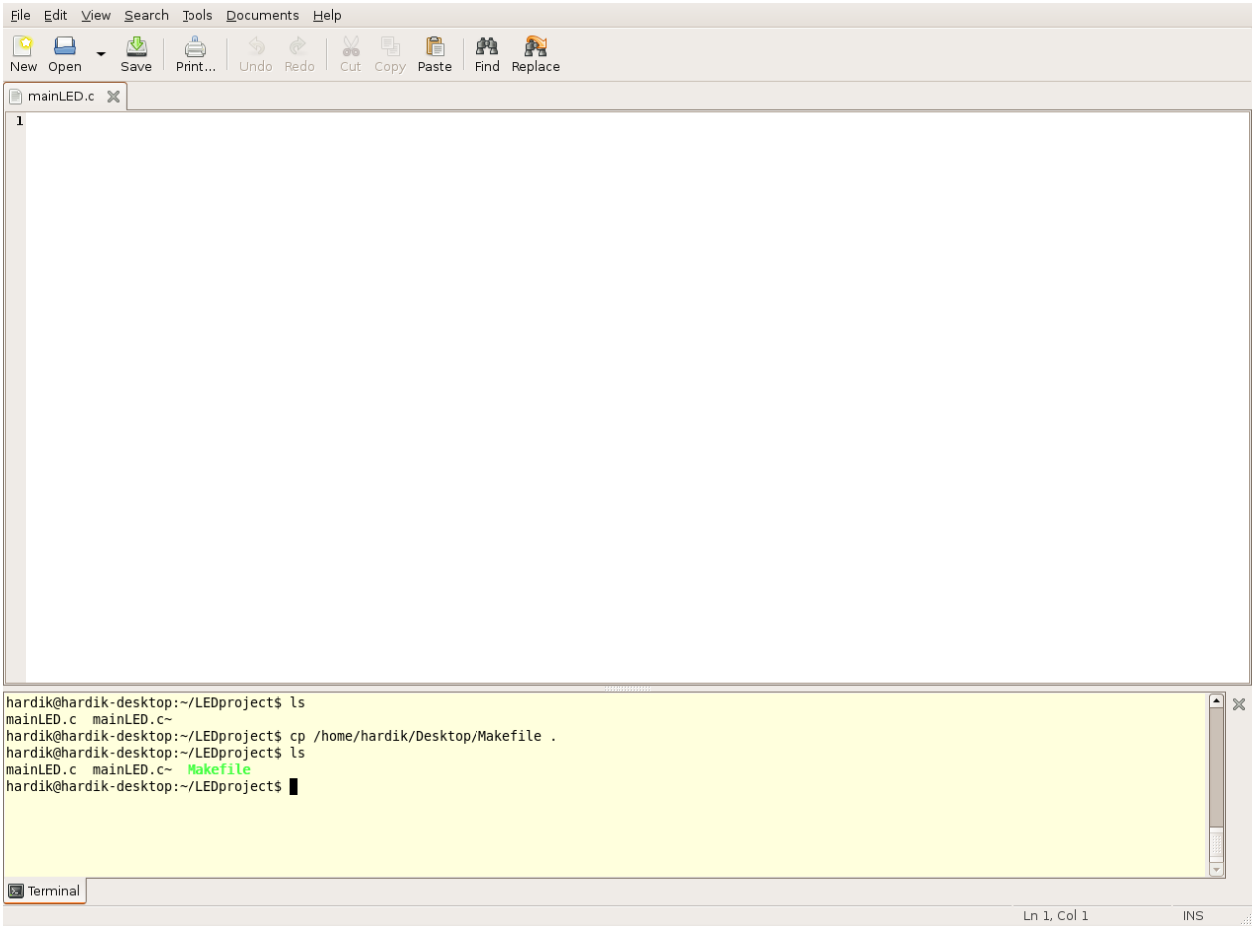
   STEP 2: Create a Project directory



STEP 3: Save the File as Xfilename.c



STEP 4: Copy the Makefile (NOTE: Make is the Utility which calls all the compiler related options which are in script file called as Makefile) in your working project directory



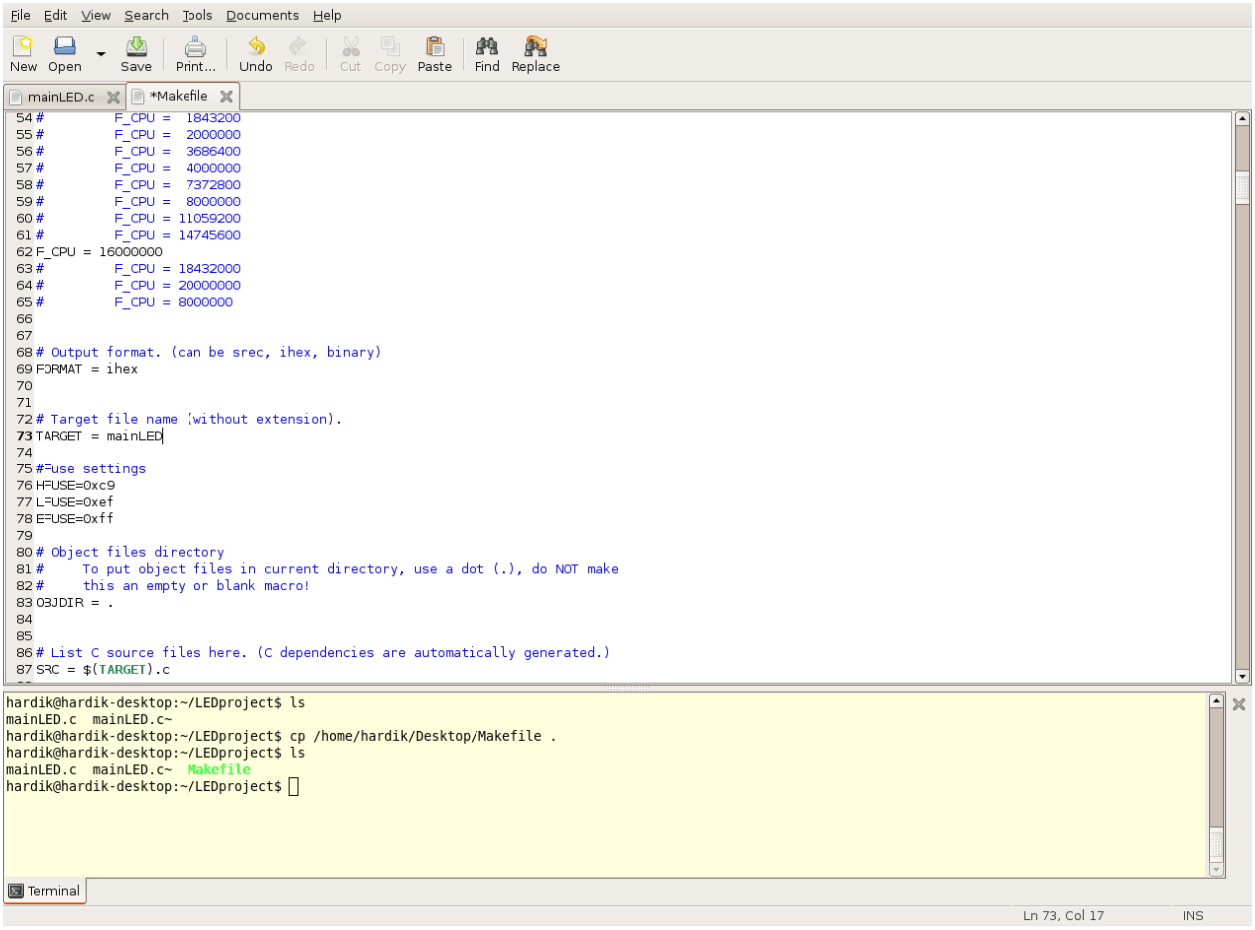
STEP 5: Write your code.

```
/*********************************************************************************************
FileName        :   mainLED.c
ProjectName     :   LED TEST
Target          :   ATmega128
CPU Clock       :   16Mhz
Dated           :   23rd July 2008
Author          :   TRI TEAM
Description     :   This is a program to test the LED's using PORTC.
*********************************************************************************************/

#include<avr/io.h>

#define LEDPORTDIR  DDRC
#define LEDPORT     PORTC

void LED_Init(void)
{
    LEDPORTDIR = 0xFF;  /* SET PORTC as Output */
    LEDPORT = 0xFF;     /* Turn OFF all the LED */
}

int main(void)
{
    volatile unsigned char idelay;
    volatile unsigned int jdelay;

    LED_Init();

    while(1)
    {
        for(idelay=0;idelay<10;idelay++)    /* Software Delay */
            for(jdelay=0;jdelay<60000;jdelay++);

        LEDPORT = LEDPORT ^ 0xFF;   /* Toggle LED */
    }
}
```

STEP 6: Save the file

STEP 7: Edit the Makefile in this file change TARGET = Xfilename (**with no extensions and no quotes (“)**). If you put any extensions it won’t work.

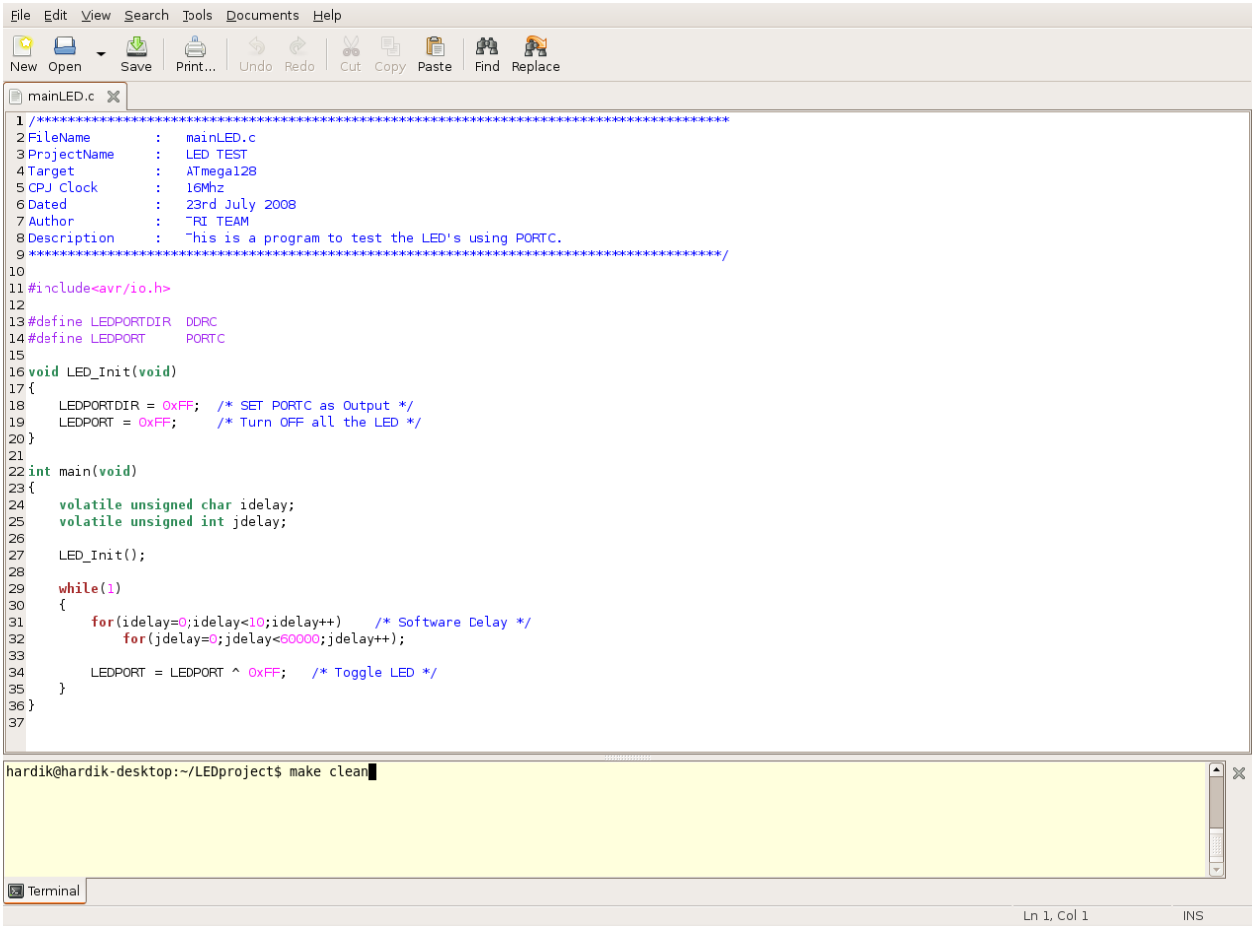
NOTE: Xfilename indicates any filename that a user will use.



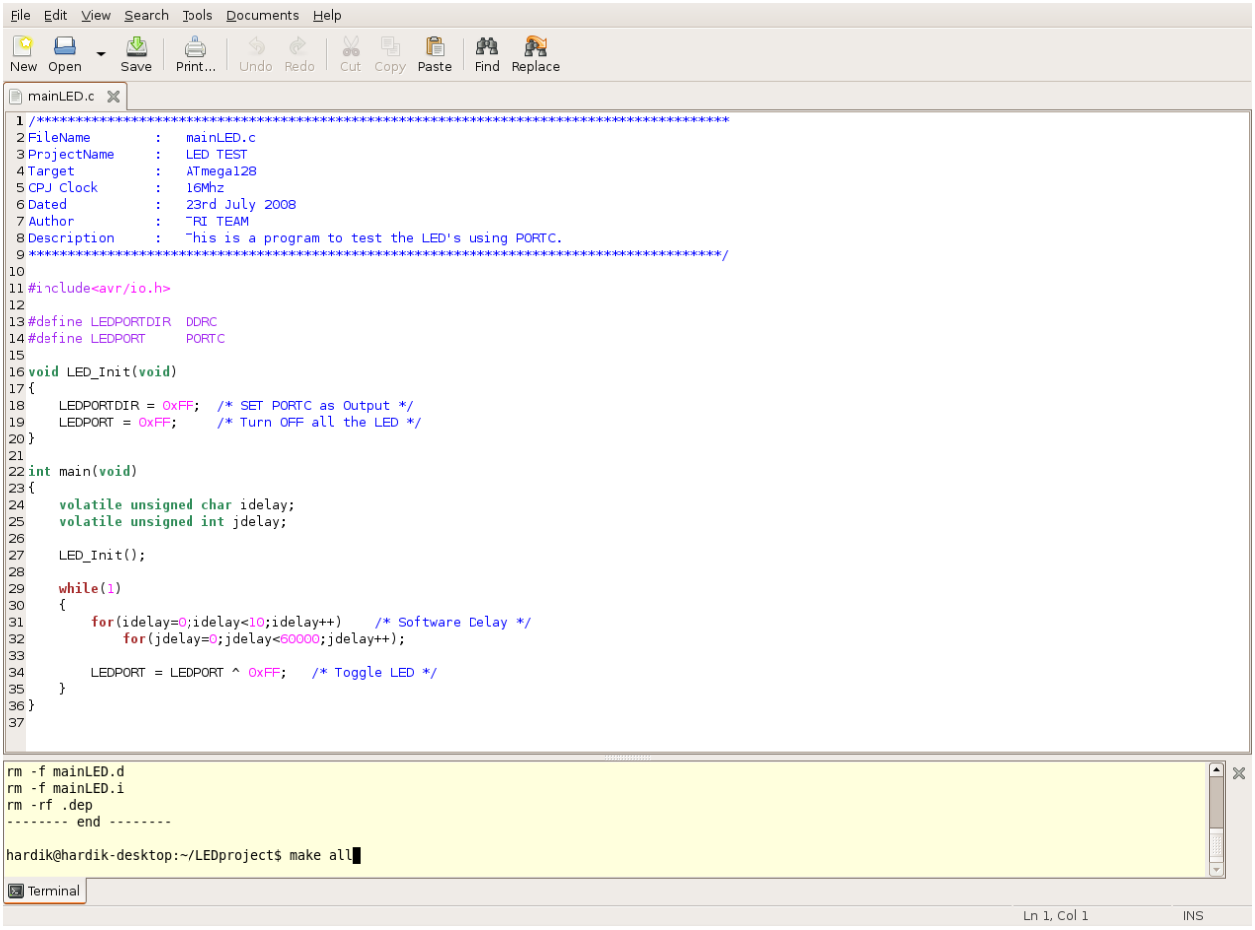
STEP 8: Save the Makefile.

STEP 9: On Linux based systems, you can use one of the makefiles of the sample codes marked as Makefile_LINUX by renaming it to Makefile, failing which you will get an error saying “No rule to make target. Stop.”

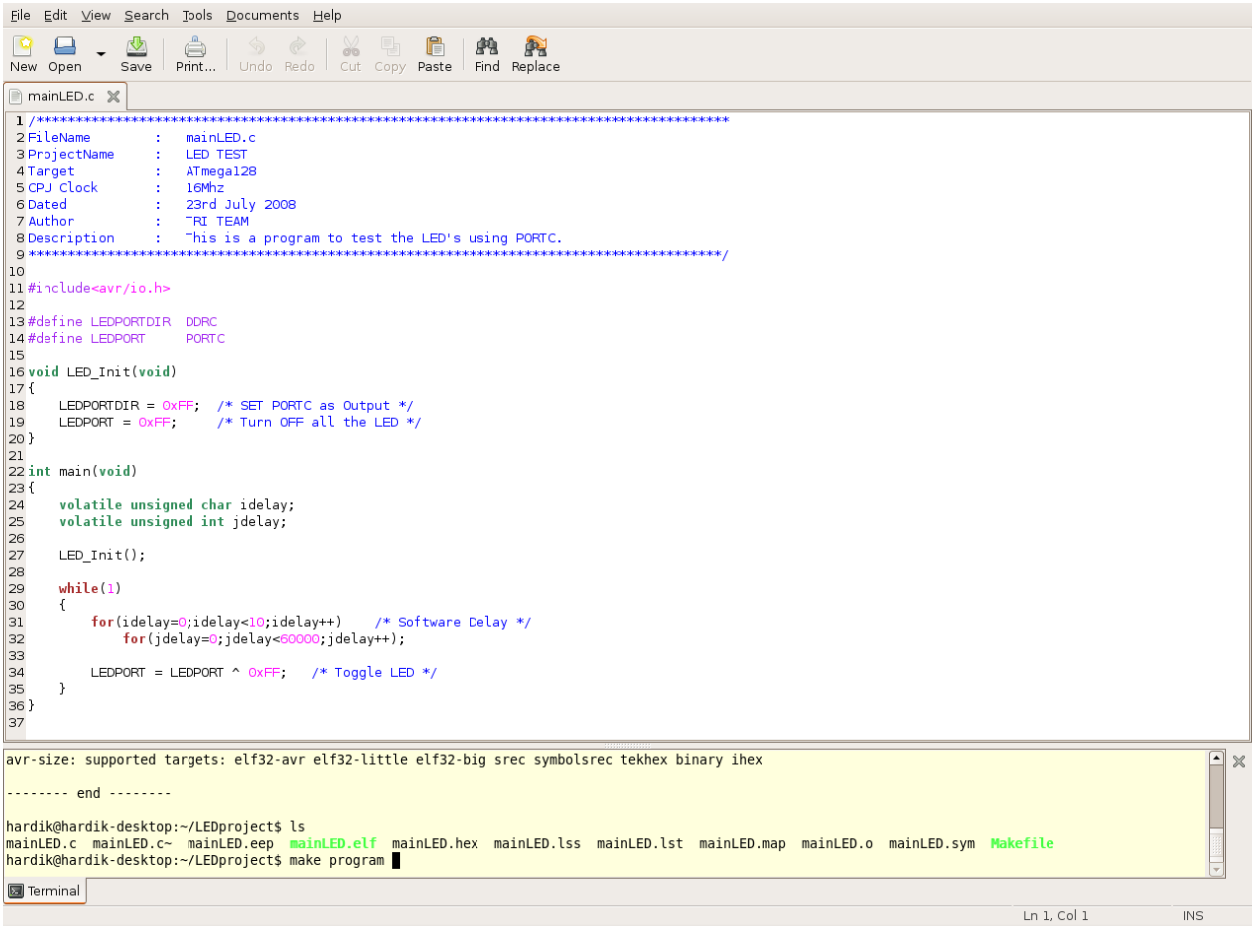
STEP 10: On the Embedded terminal goto the Project directory and type: make clean (this will remove all the previously compiled files in the project directory).



STEP 10: Type: make all (this will compile the source file and generate hex file if there are no errors in code).



STEP 11: type: make program (this will burn the hex into the microcontroller)



1.1. Changing the Fuse settings

NOTE: The fuse setting must be changed only if required.

The ATmega128 has three fuse bytes referred as High Fuse Byte, Low Fuse Byte and Extended Fuse Byte. Refer the ATmega128 datasheet for information on what the fuse bits are. To cut things short, Fuse bits configure various options like EEPROM SAVE, Programming Enable, Clock Speed, Oscillator startup time, clock source etc.

The following Table represents Extended Fuse Byte:

Extended Fuse Byte	BIT No.	Description	Default Value
-	7	-	1
-	6	-	1
-	5	-	1
-	4	-	1

-	3	-	1
-	2	-	1
M103C	1	Atmega103 compatibility mode	0 (programmed)
WDTON	0	Watchdog timer is always ON	1 (unprogrammed)

The following Table represents High Fuse Byte:

High Fuse Byte	BIT No.	Description	Default Value
OCDEN	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN	5	Enable serial programming and data downloading	0 (programmed SPI prog. enabled)
CKOPT	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through chip erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size	0 (programmed)
BOOTSZ0	1	Select Boot Size	0 (programmed)
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

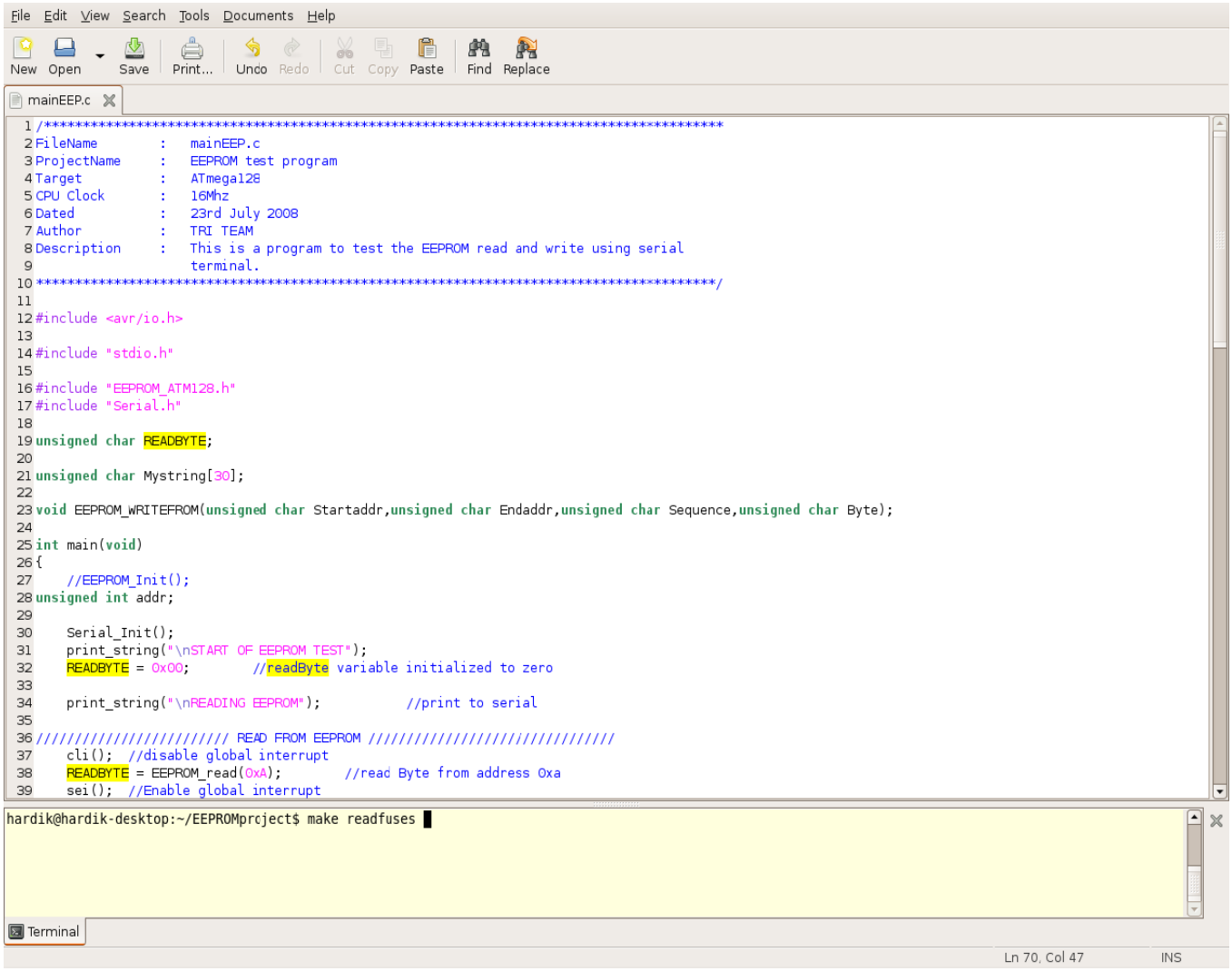
The following Table represents Low Fuse Byte:

Low Fuse Byte	BIT No.	Description	Default Value
BODLEVEL	7	Brown out detector trigger level	1 (programmed)
BODEN	6	Brown out detector enable	1 (programmed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)
SUT0	4	Select start-up time	0 (programmed)
CKSEL3	3	Select Clock source	0 (programmed)
CKSEL2	2	Select Clock source	0 (programmed)

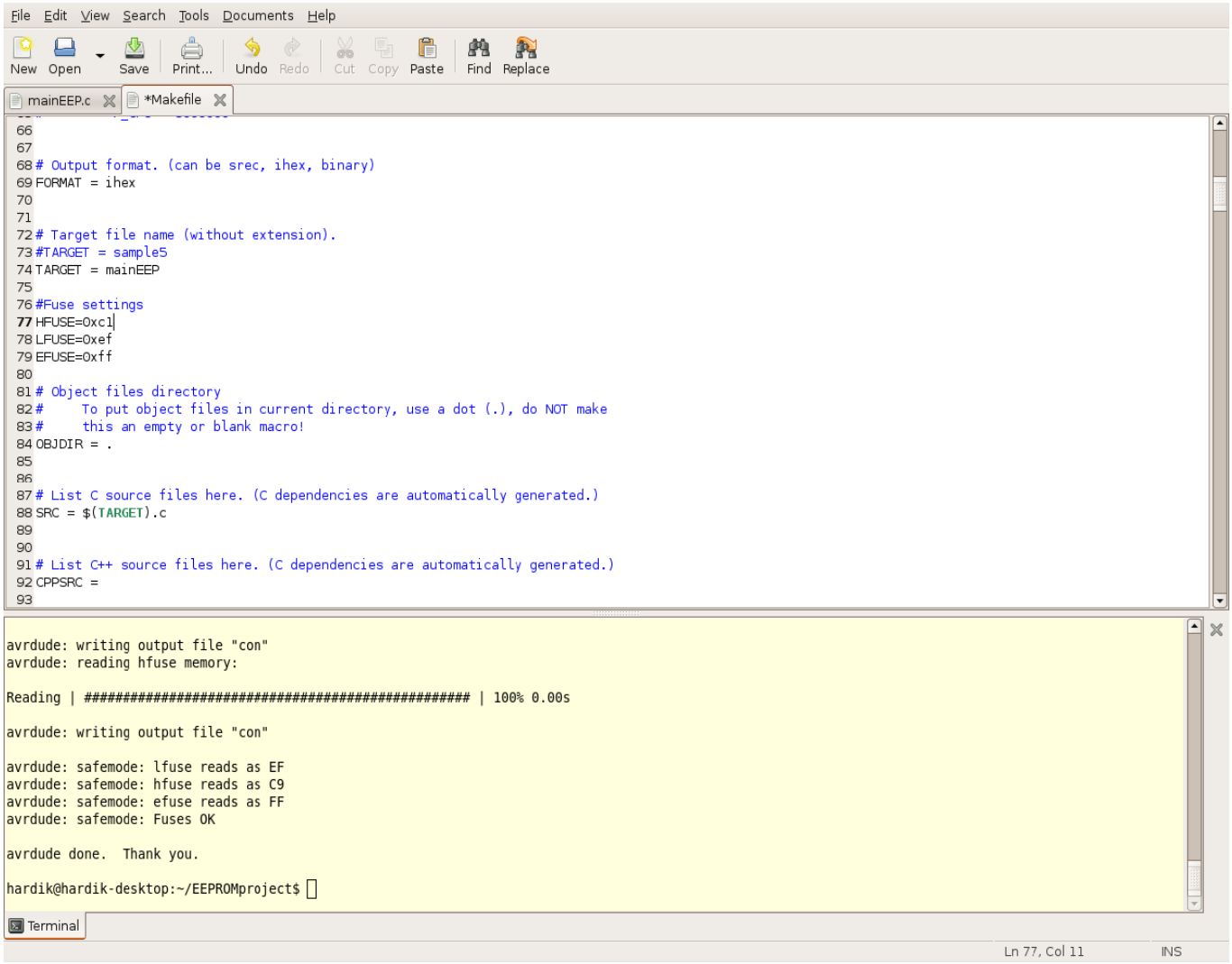
CKSEL1	1	Select Clock source	0 (programmed)
CKSEL0	0	Select Clock source	1 (unprogrammed)

The Make utility allows programming the fuse bits. The following procedure will help you configuring the fuse bits.

STEP 1: On Embedded terminal type make readfuses (This will read the fuse setting currently present on the board) as shown below.



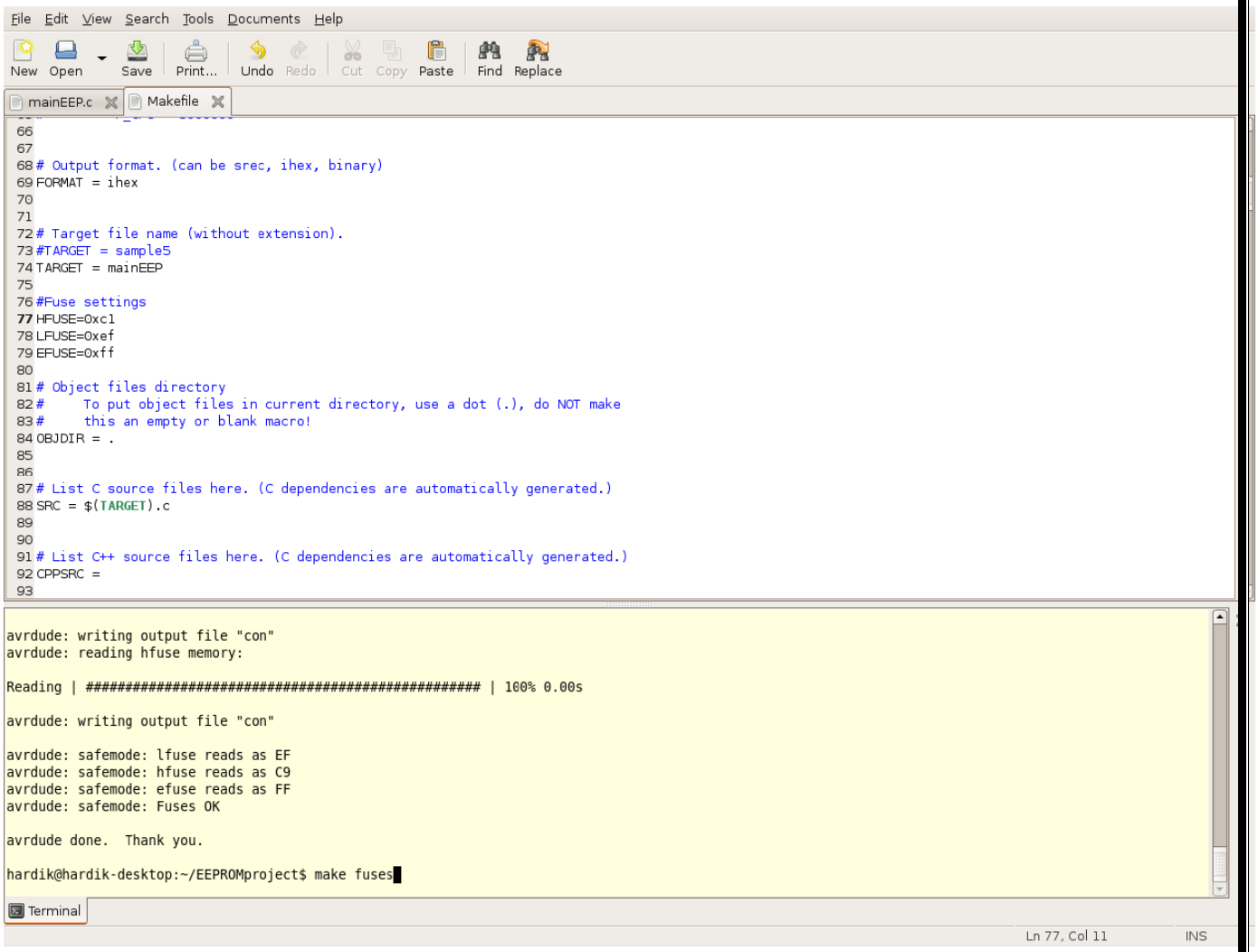
The output will be displayed as shown below:



STEP 2: Edit the Makefile as shown in the above picture as per the fuse setting (HFUSE, LFUSE and EFUSE are the labels to be edited) that is required.

STEP 3: On Embedded terminal type make fuses (This will read the fuse setting currently present on the board) as shown below.

NOTE: DO NOT MESS WITH THE FUSE BITS UNLESS YOU ARE VERY SURE OF WHAT EFFECT IT WOULD HAVE, FAILING WHICH THE PROGRAM MEMORY MIGHT BE LOCKED FOR WRITING.



STEP 4: Again on the embedded terminal type make readfuses to verify that fuse bit have been saved.

File Edit View Search Tools Documents Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

mainEEP.c

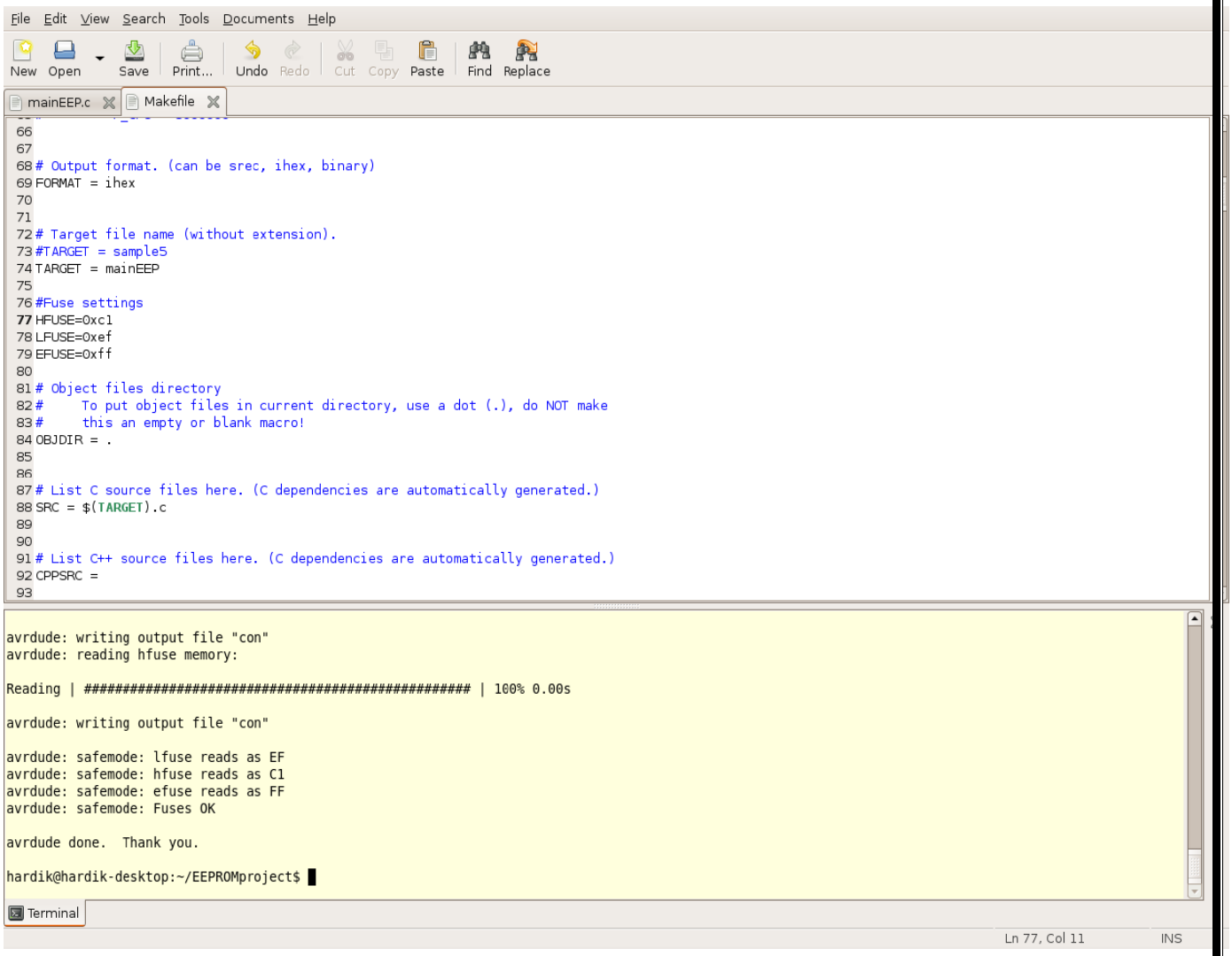
```
/*****************************************************************************************
FileName        :   mainEEP.c
ProjectName     :   EEPROM test program
Target          :   ATmega128
CPU Clock       :   16Mhz
Dated           :   23rd July 2008
Author          :   TRI TEAM
Description     :   This is a program to test the EEPROM read and write using serial
                    terminal.
*****************************************************************************************/

#include <avr/io.h>

#include "stdio.h"

#include "EEPROM_ATM128.h"
#include "Serial.h"

unsigned char READBYTE;

unsigned char Mystring[30];

void EEPROM_WRITEFROM(unsigned char Startaddr,unsigned char Endaddr,unsigned char Sequence,unsigned char Byte);

int main(void)
{
    //EEPROM_Init();
unsigned int addr;

    Serial_Init();
    print_string("\nSTART OF EEPROM TEST");
    READBYTE = 0x00;        //readByte variable initialized to zero

    print_string("\nREADING EEPROM");          //print to serial

////////////////////////// READ FROM EEPROM ////////////////////////////////
    cli();  //disable global interrupt
    READBYTE = EEPROM_read(0xA);        //read Byte from address 0xa
    sei();  //Enable global interrupt
```

```
hardik@hardik-desktop:~/EEPROMproject$ make readfuses
```

Terminal

Ln 70, Col 47 INS

As shown below the fuse setting have been modified successfully.



Getting Started on Windows

The sample programs have been given in the CD folder named

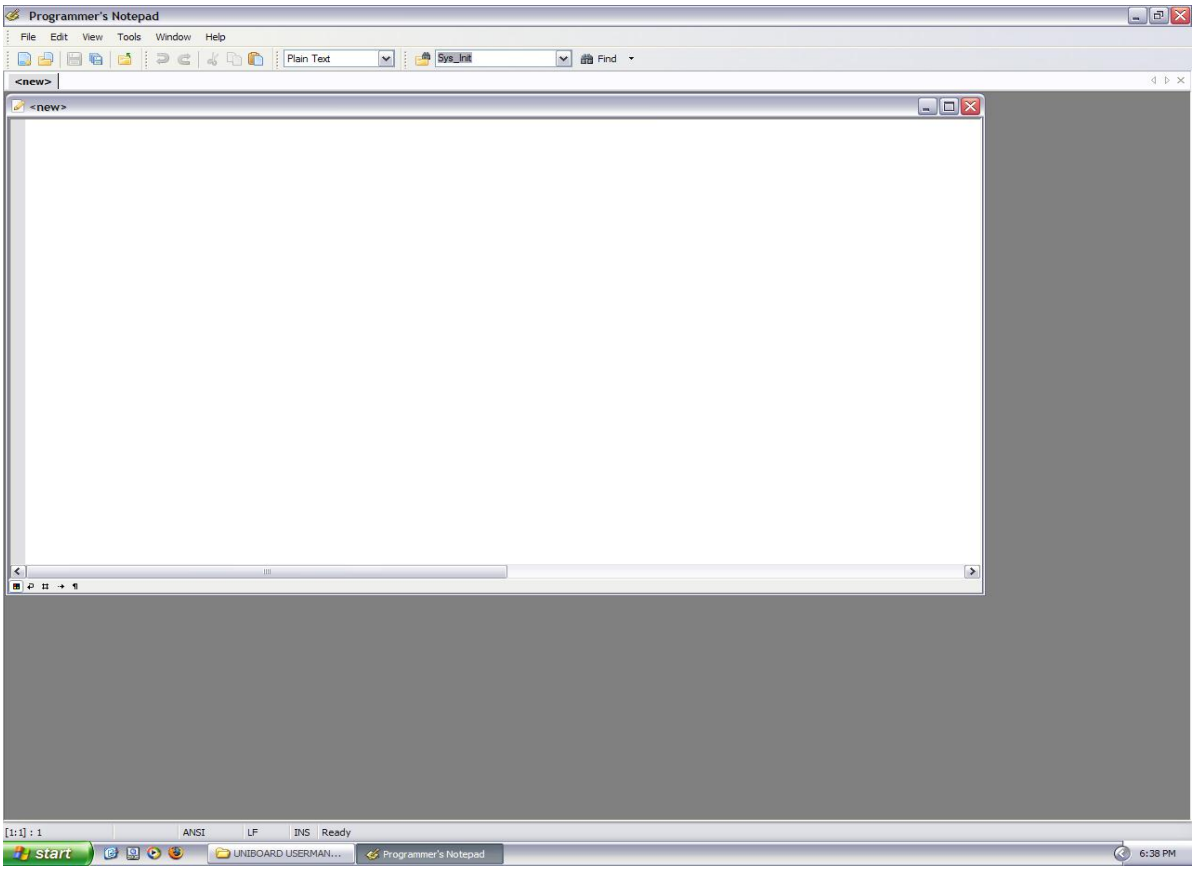
"\UNIBOARD CONTENTS\UNIBOARD_SAMPLE_CODES\NON_RTOS\" and

"\UNIBOARD CONTENTS\UNIBOARD_SAMPLE_CODES\ RTOS"

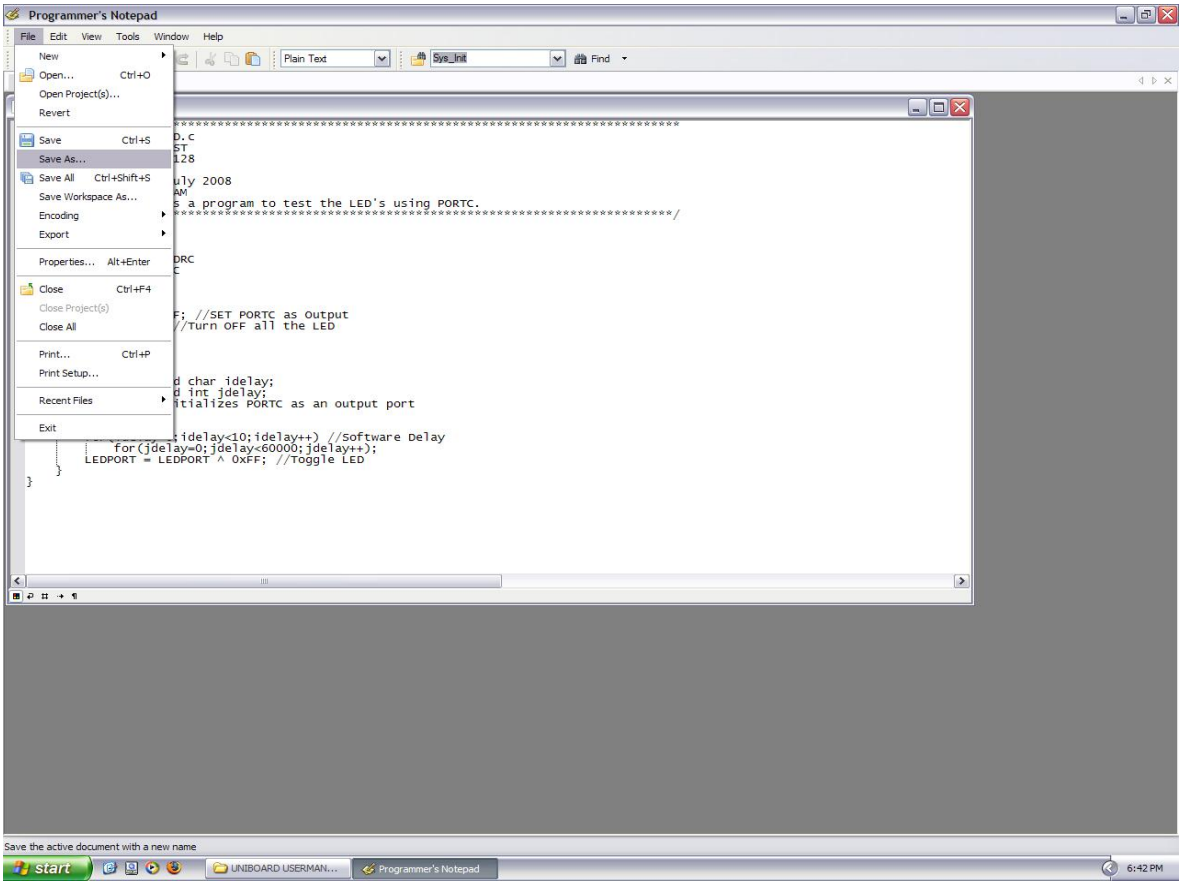
Or you can download it from our website at link http://thinklabs.in/download/

Text Editor (programmer's Notepad)

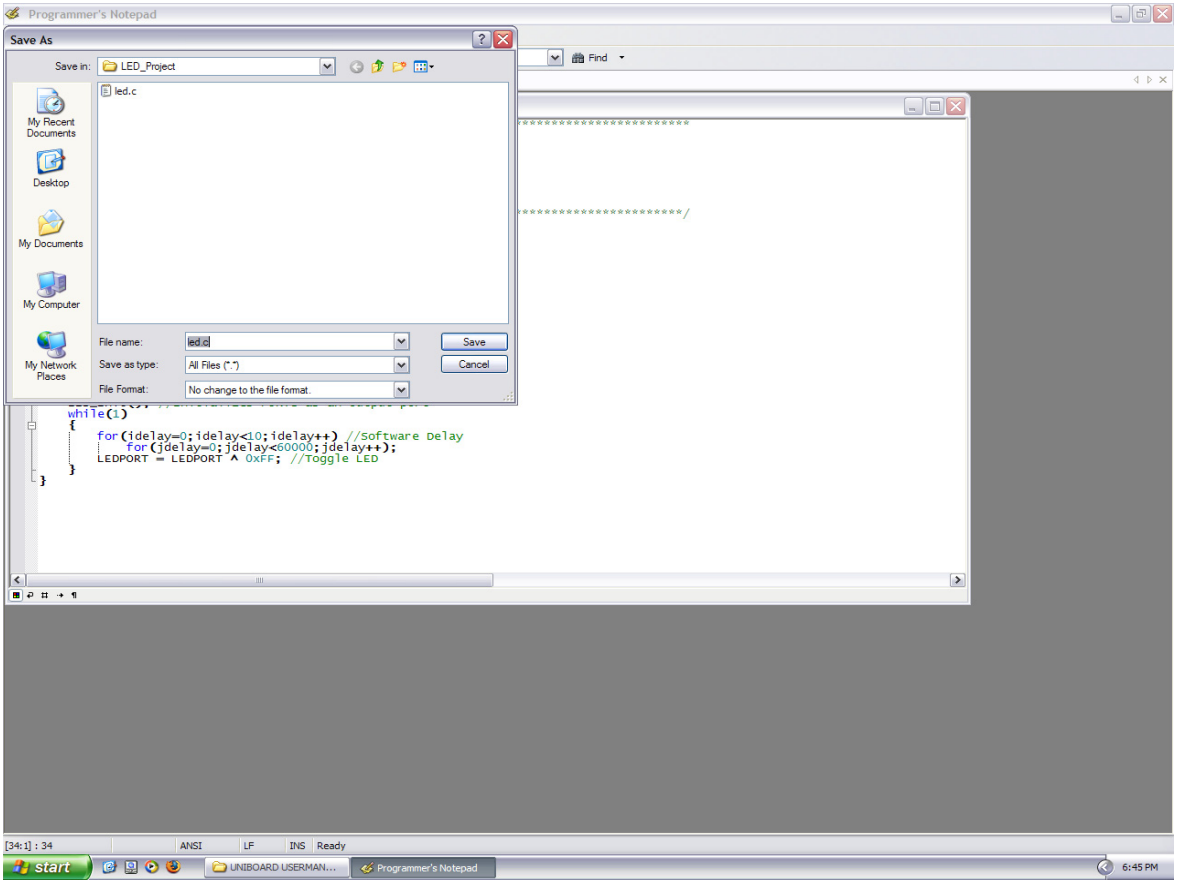
Step 1: Select Programmer's Notepad from Start menu->Programs->Win-avr->Programmers Notepad.



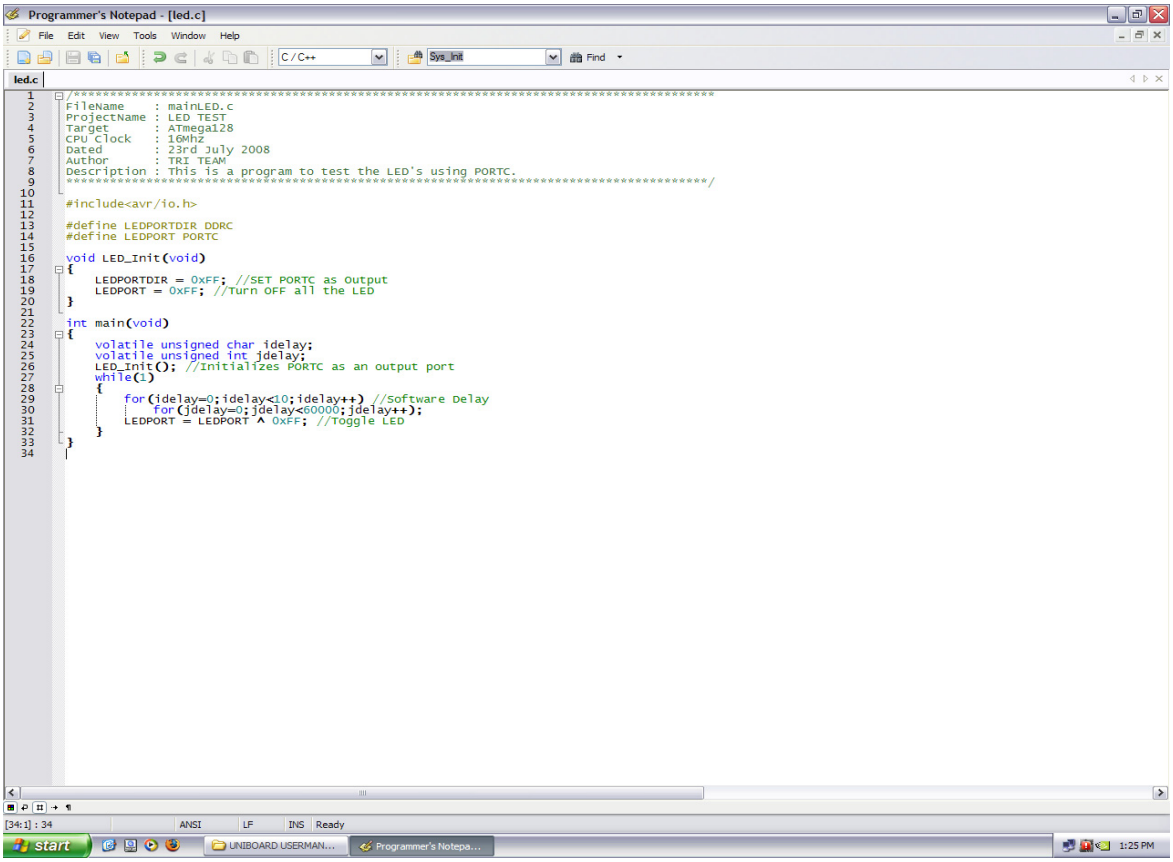
Step 2: Type your code here and save the file as shown below:



STEP 3: Save the file as anyfilename.c

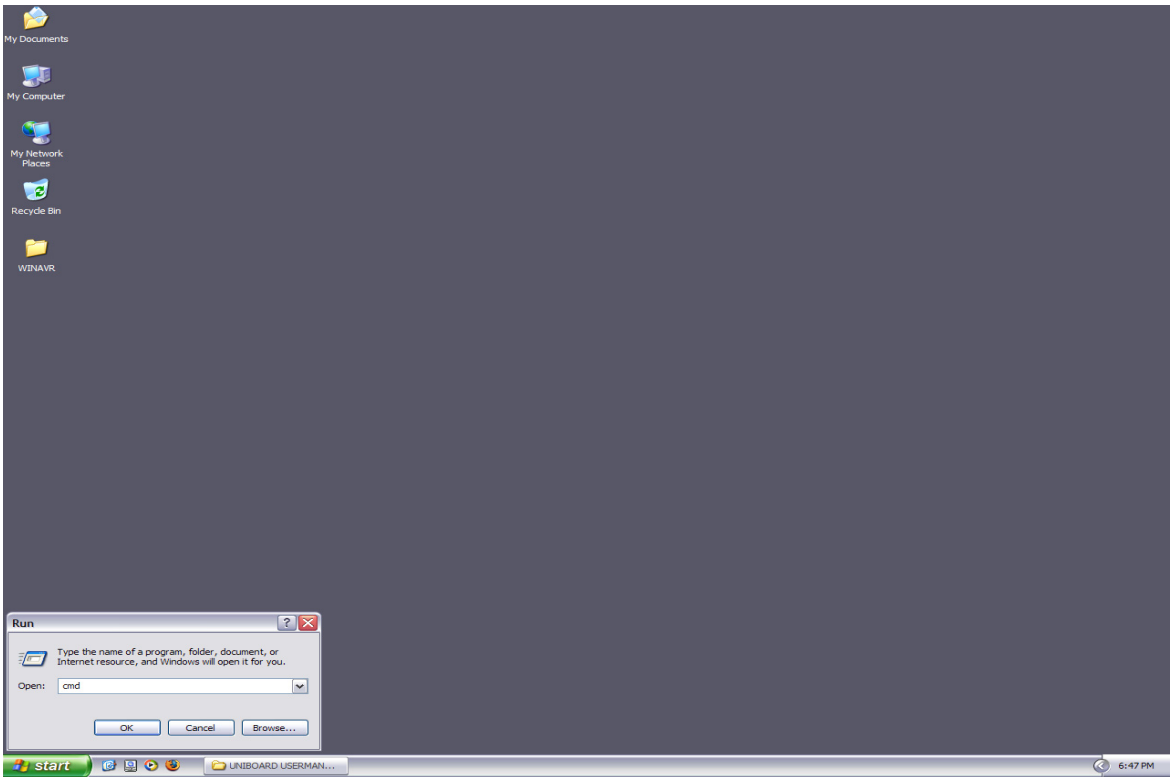


STEP 4: You can view the line numbers by selecting view menu ->Line Numbers.

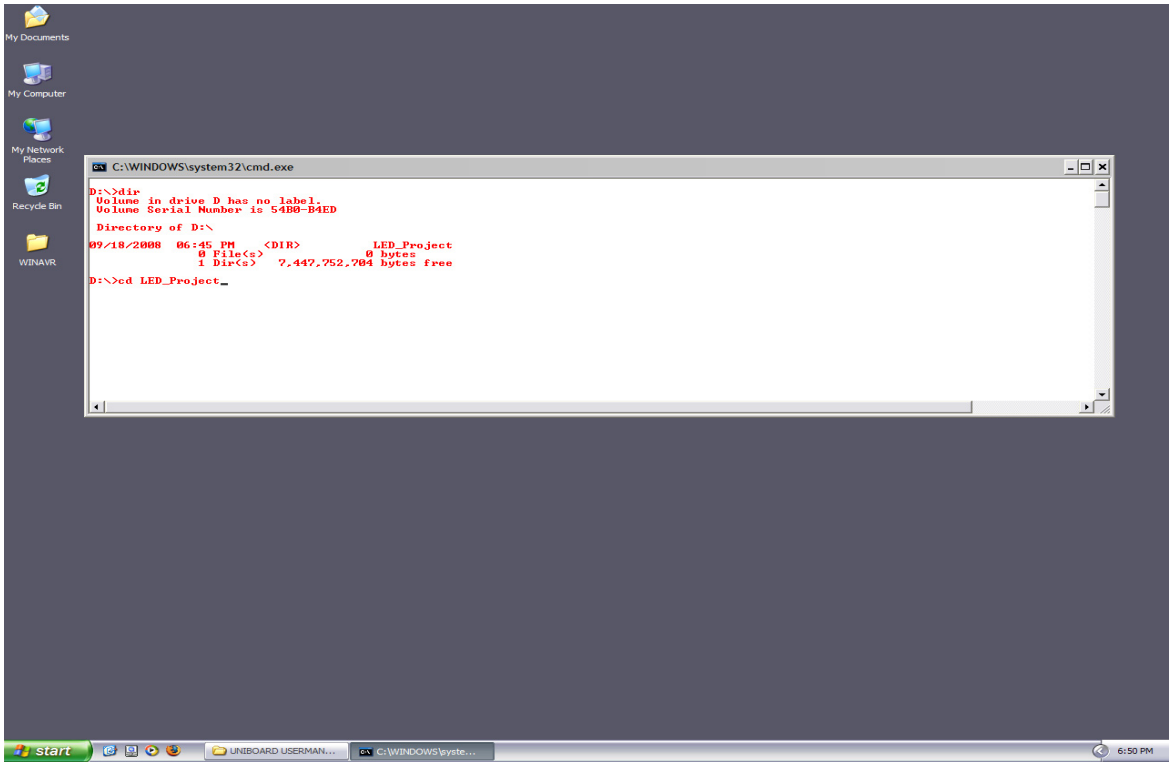

Programmer's Notepad - [led.c]
File Edit View Tools Window Help
led.c
/***
FileName : mainLED.c
ProjectName : LED TEST
Target : ATmega128
CPU Clock : 16Mhz
Dated : 23rd July 2008
Author : TRI TEAM
Description : This is a program to test the LED's using PORTC.
***/
#include<avr/io.h>
#define LEDPORTDIR DDRC
#define LEDPORT PORTC
void LED_Init(void)
{
LEDPORTDIR = 0xFF; //SET PORTC as Output
LEDPORT = 0xFF; //Turn OFF all the LED
}
int main(void)
{
volatile unsigned char idelay;
volatile unsigned int jdelay;
LED_Init(); //Initializes PORTC as an output port
while(1)
{
for(idelay=0;idelay<10;idelay++) //Software Delay
for(jdelay=0;jdelay<60000;jdelay++);
LEDPORT = LEDPORT ^ 0xFF; //Toggle LED
}
}
[34:1] : 34
ANSI
LF
INS
Ready
start
UNIBOARD USERMAN...
Programmer's Notepa...
1:25 PM

Compile the code and program the board on Windows (Command Prompt)

STEP 1: Open the command prompt by selecting Start->Run and type cmd

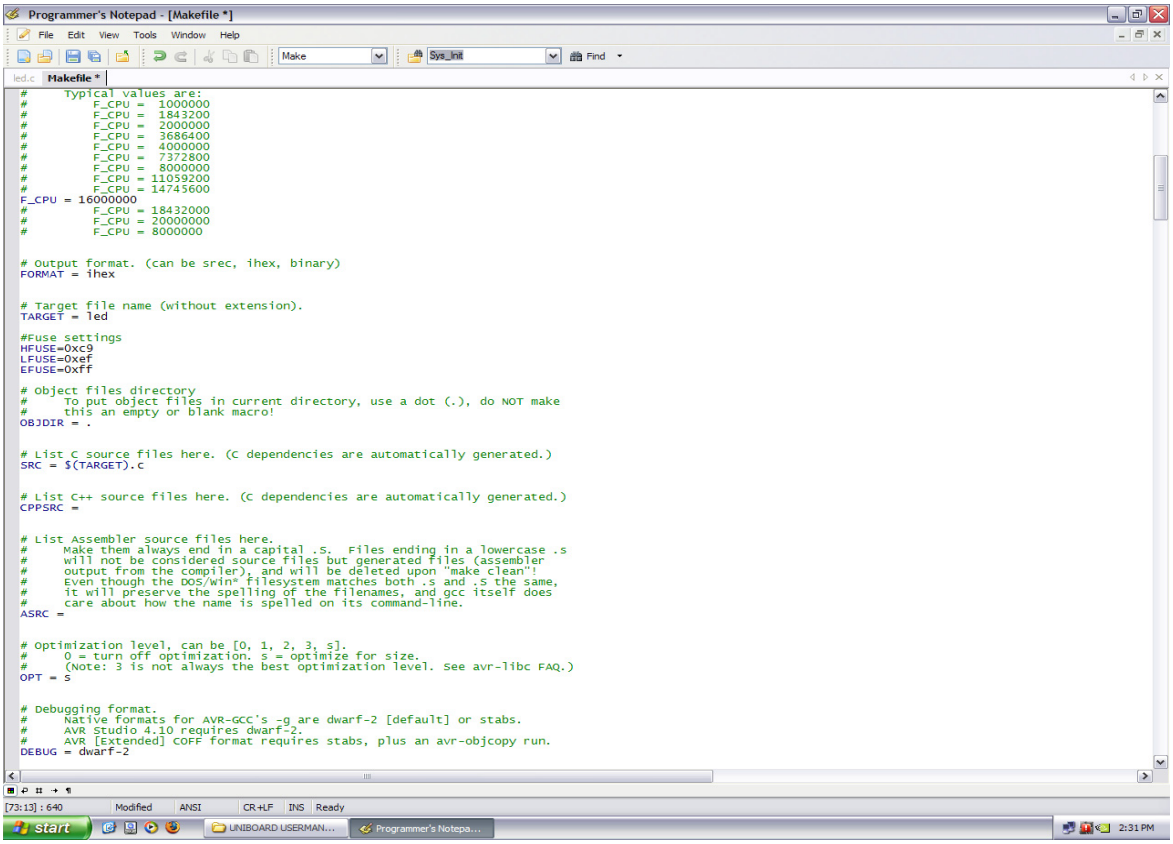


STEP 2: It will display the Command prompt and goto your project directory.

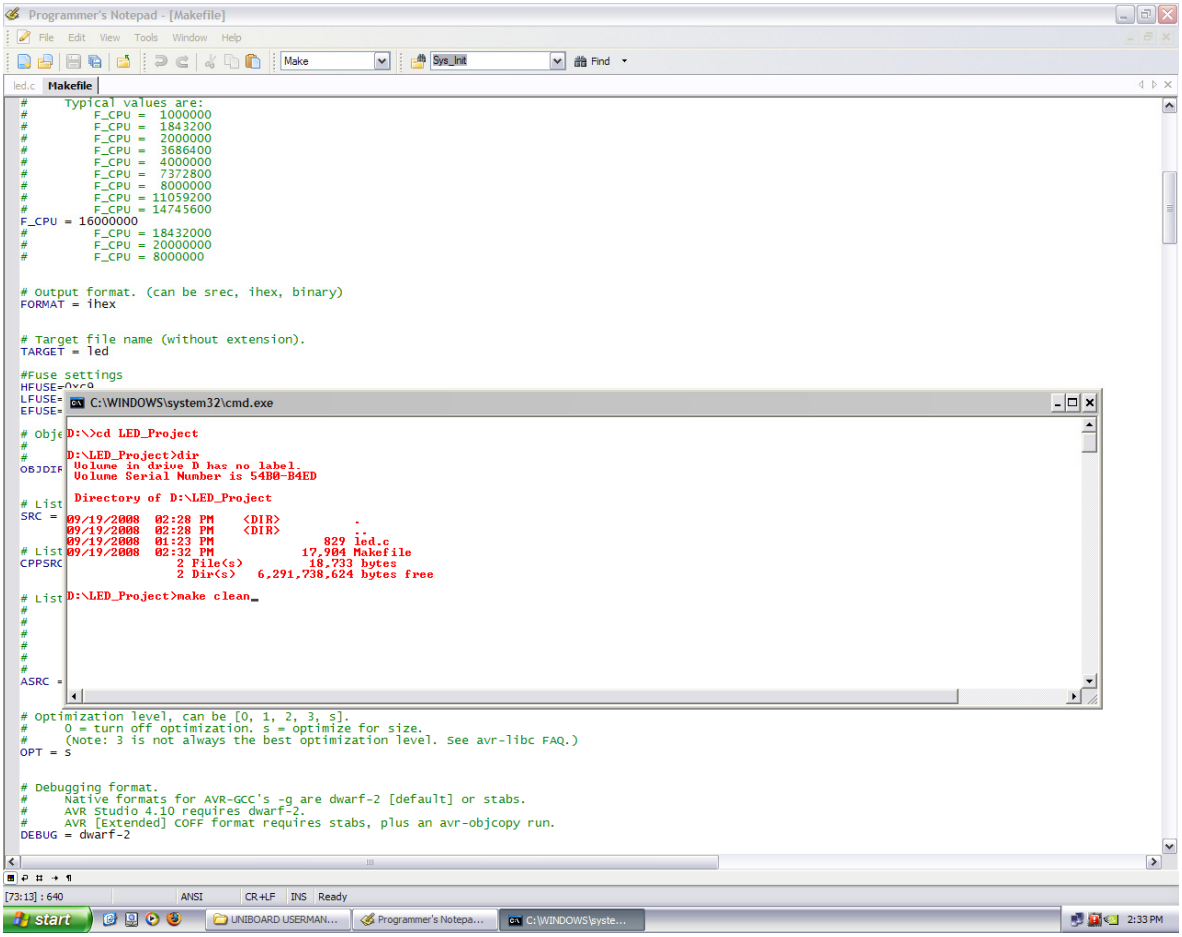


STEP 3: You must have the Makefile in your project directory. In case of windows you would find a file named Makefile_WINDOWS, which you can rename as Makefile. Make sure that you have only a single Makefile in your working directory.

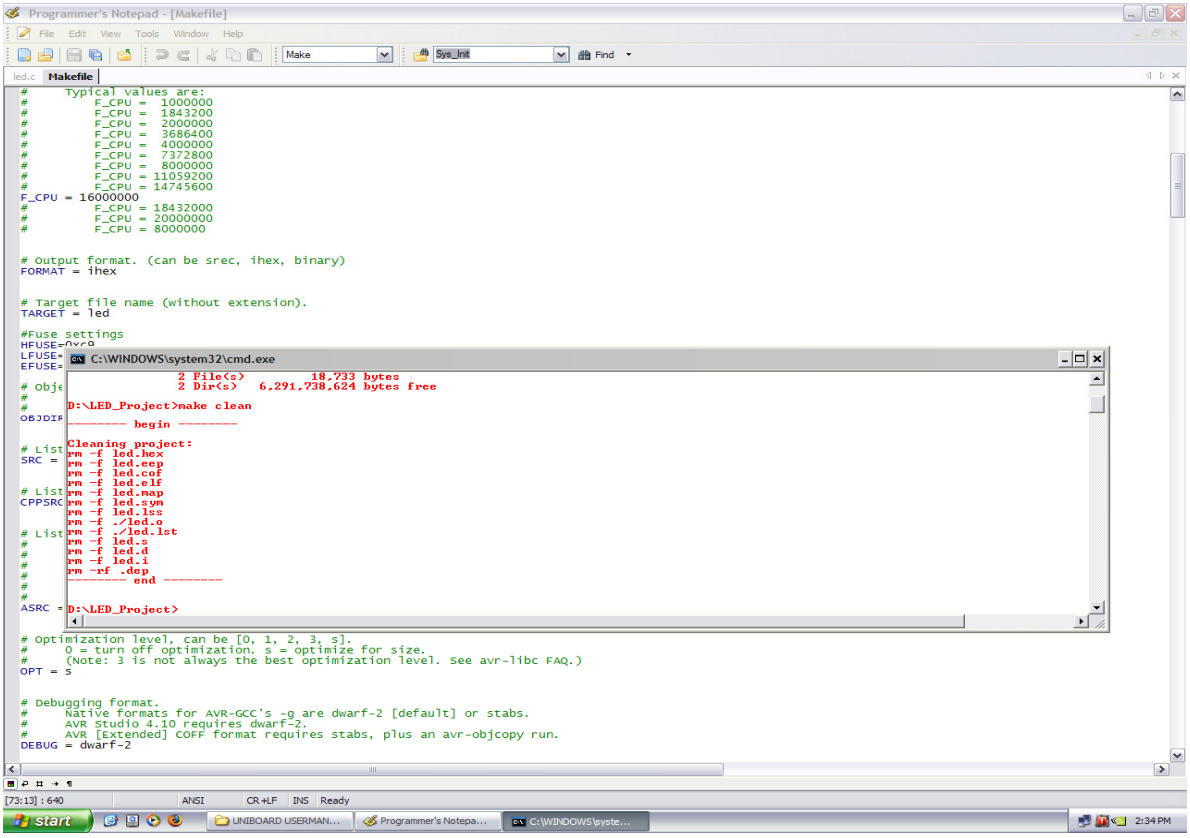
STEP 4: Edit the Makefile and set Target = filename of your C file and save the file.



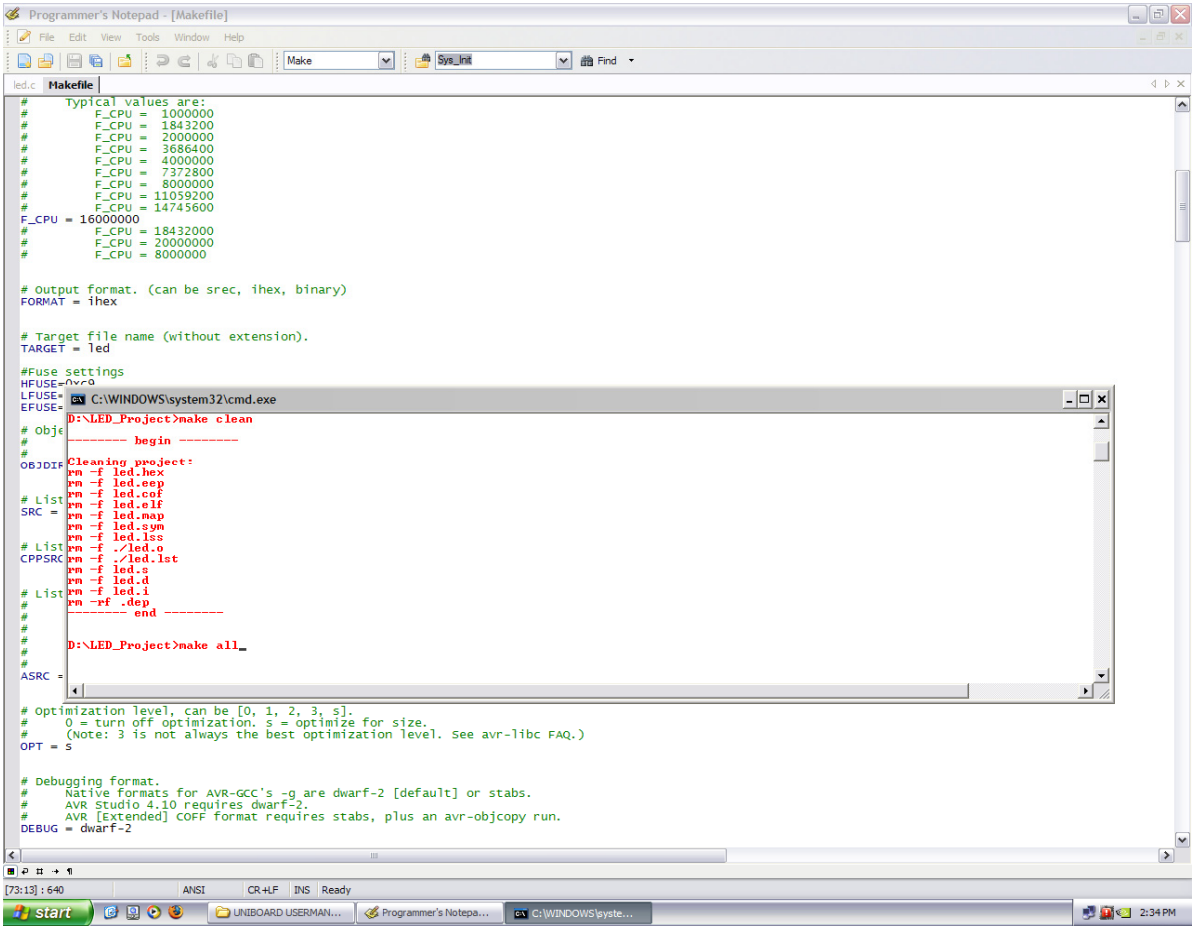
STEP 5: In command prompt window, type command as make clean and press enter



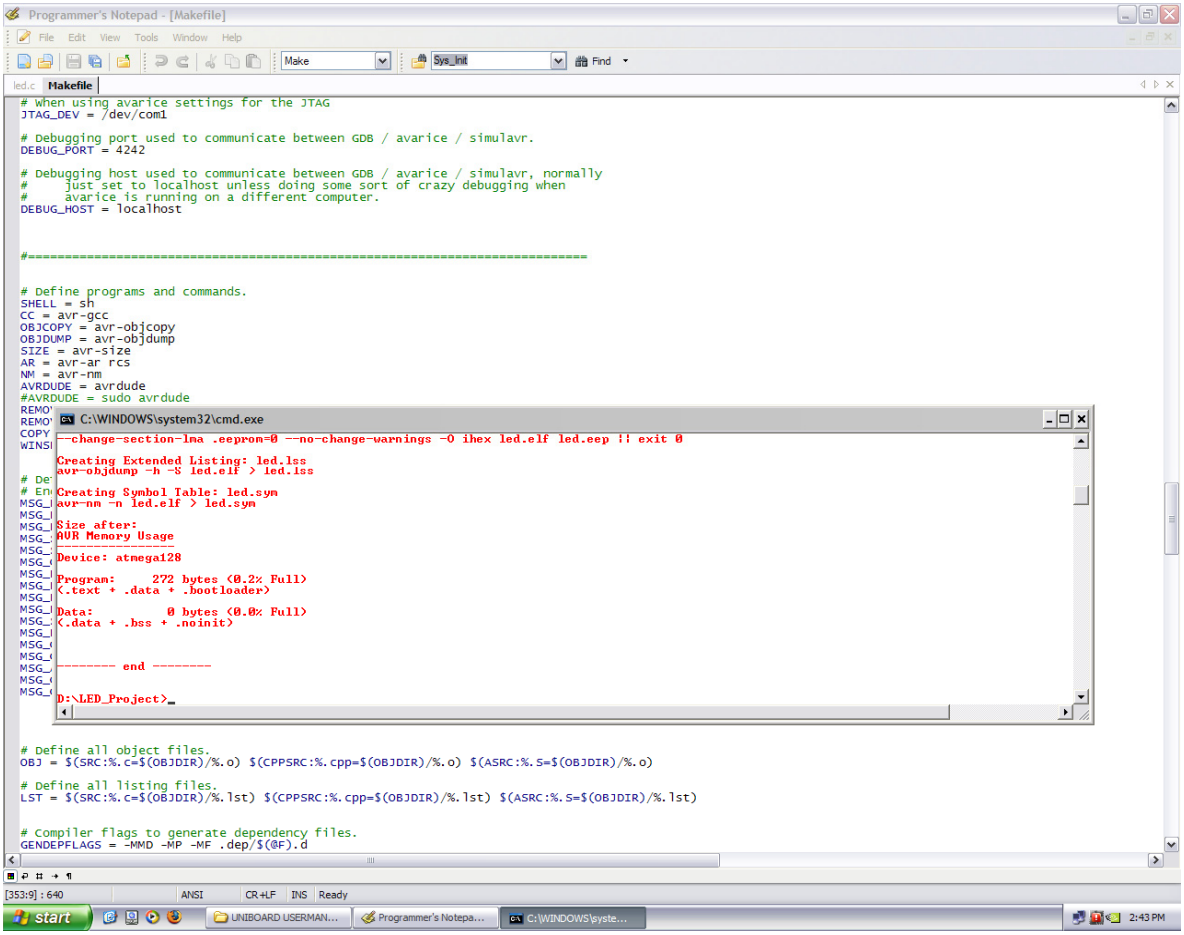
STEP 6: This will remove all previously files created due to compilation as shown below:



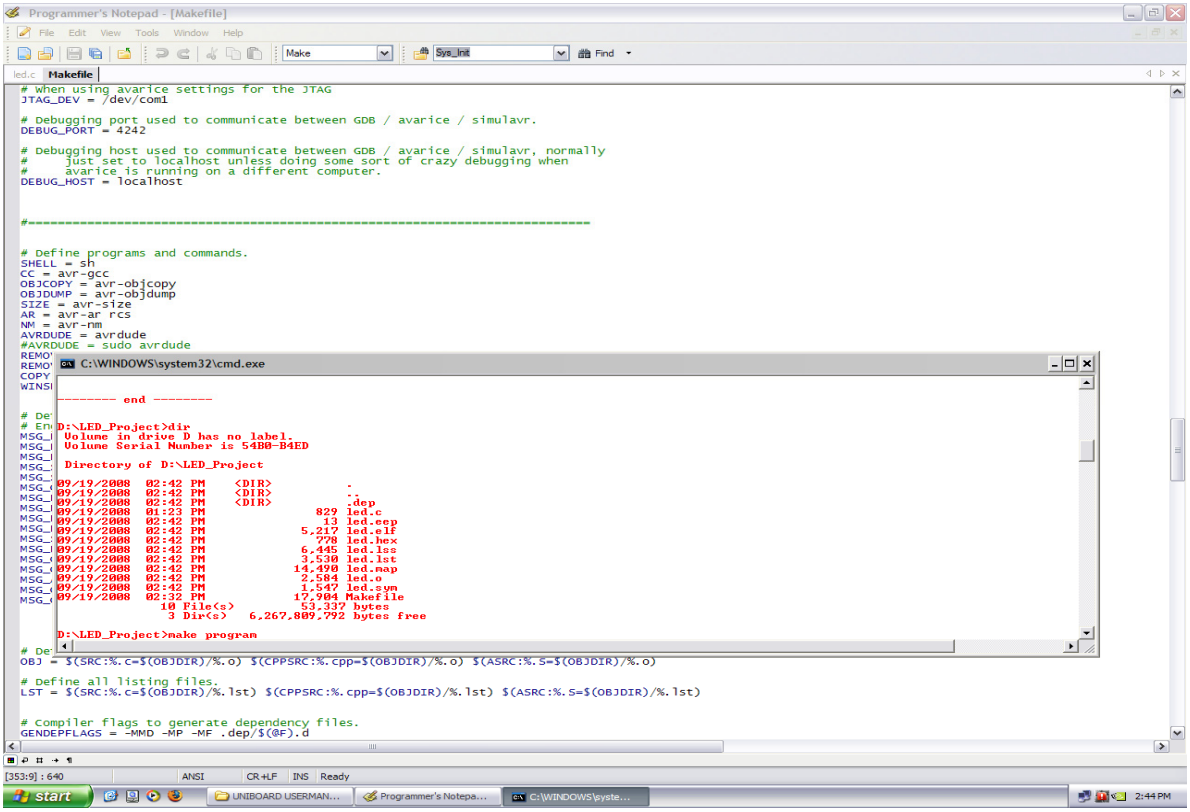
STEP 7: type make all and press enter



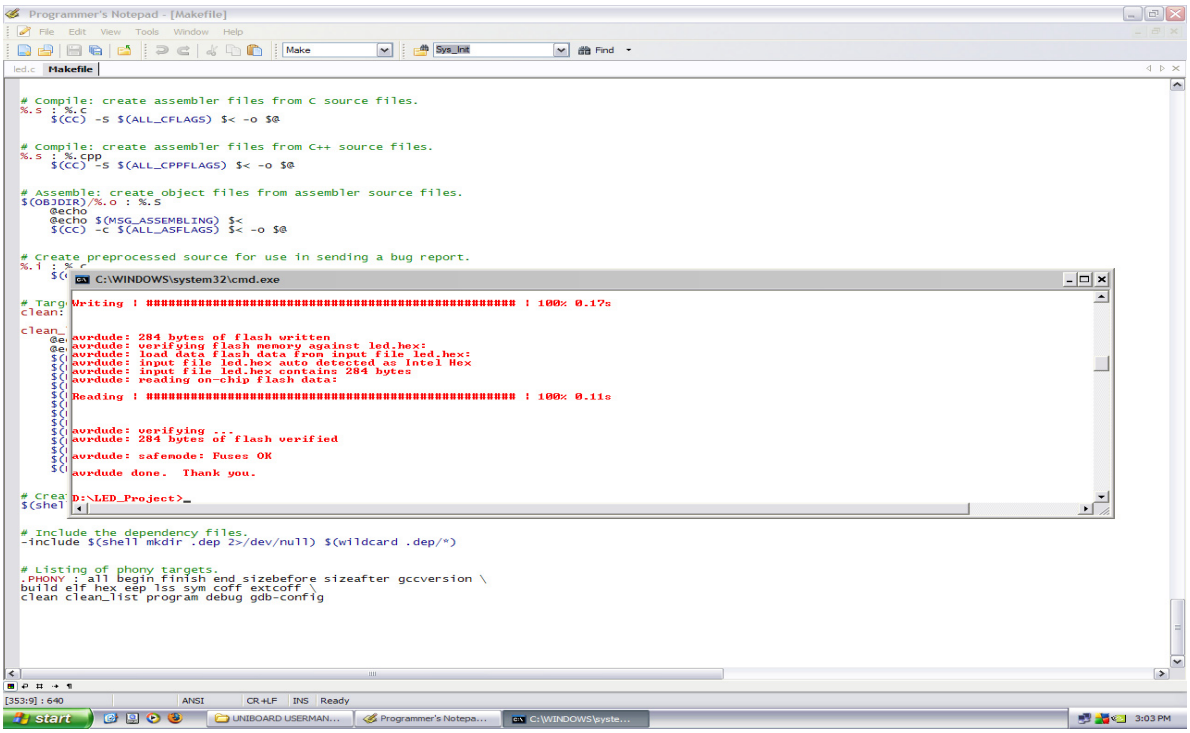
STEP 8: This command will compile the file and generate a Hex file. Type dir and press enter and check if there hex file generated as shown below.



STEP 9: For loading the compiled output to the board the board i.e. burning the hex file into the Atmega128 type command as make program.



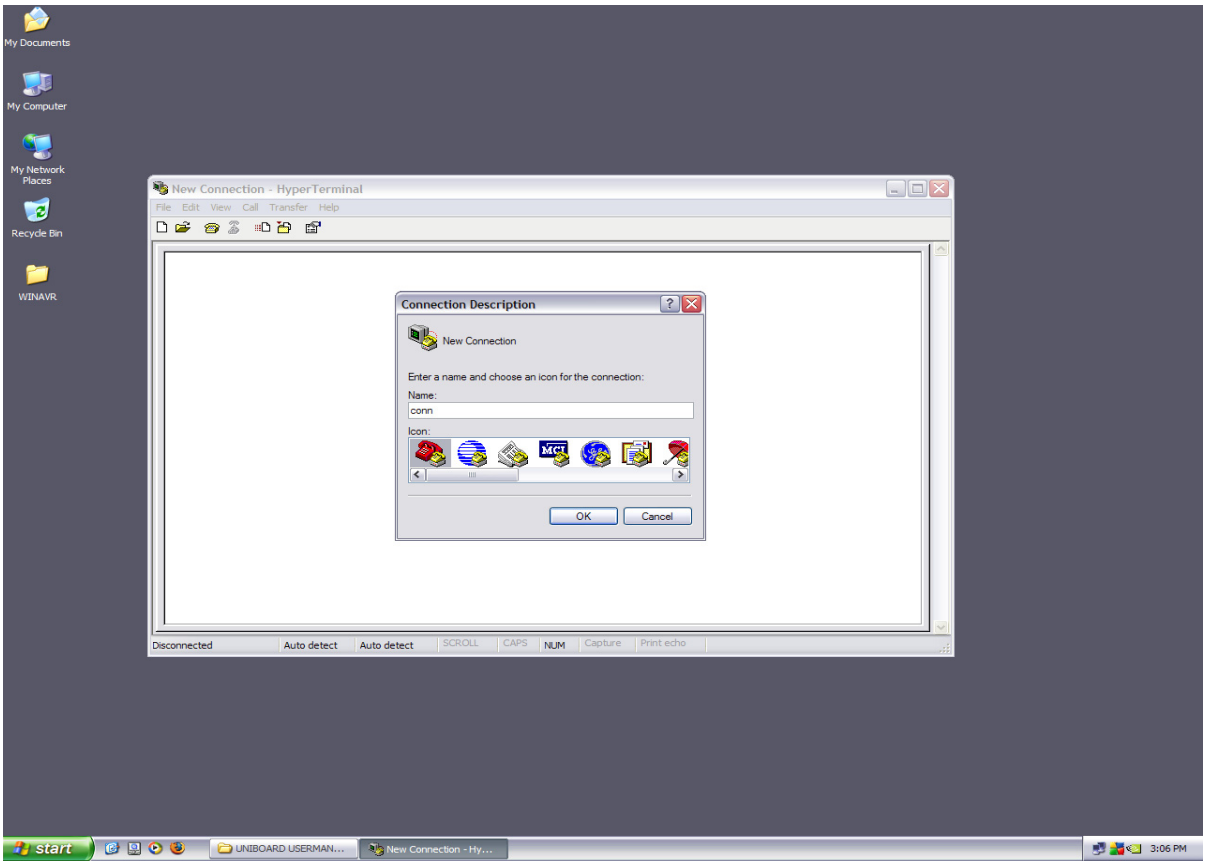
STEP 10: press enter



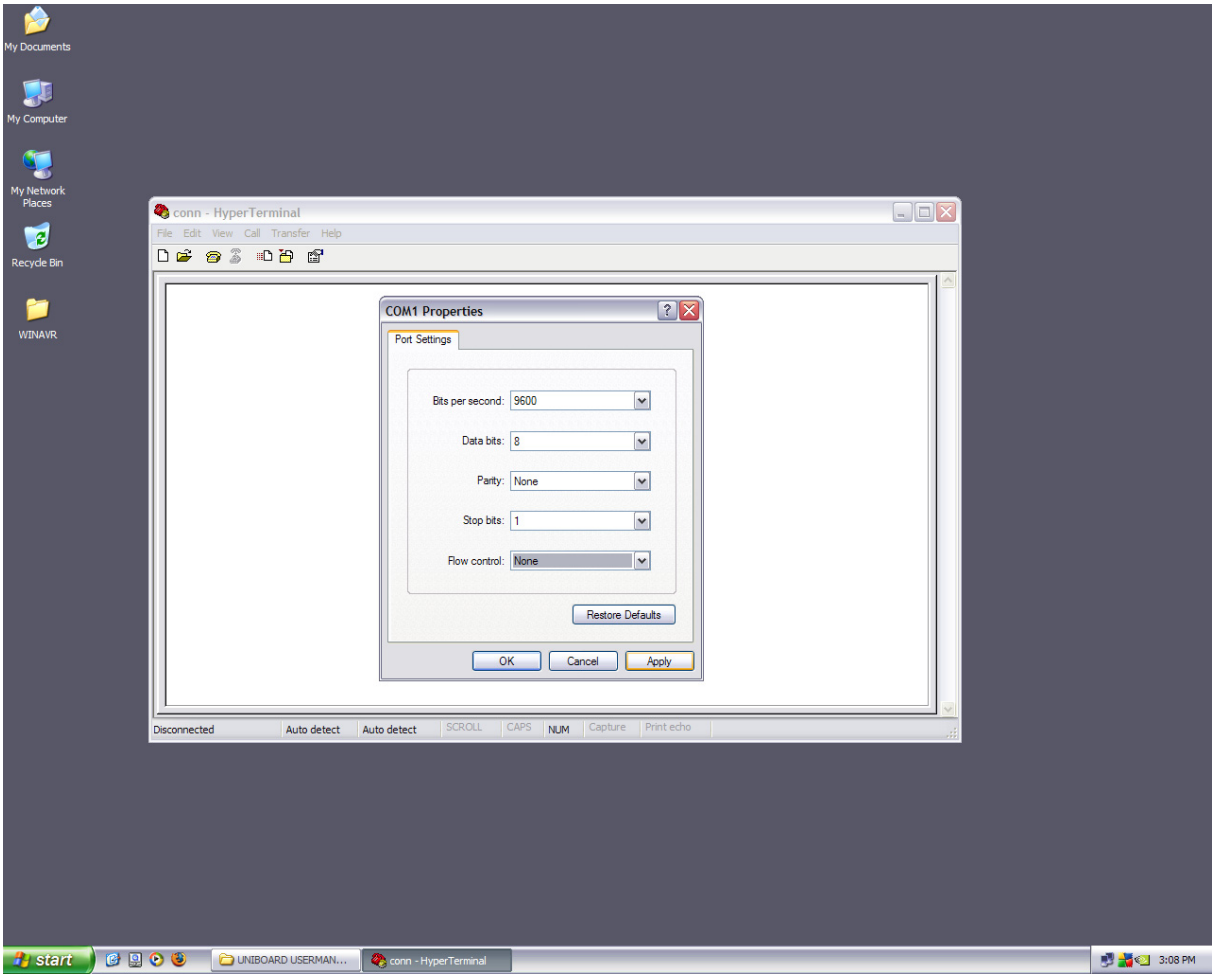
Using hyper terminal of windows

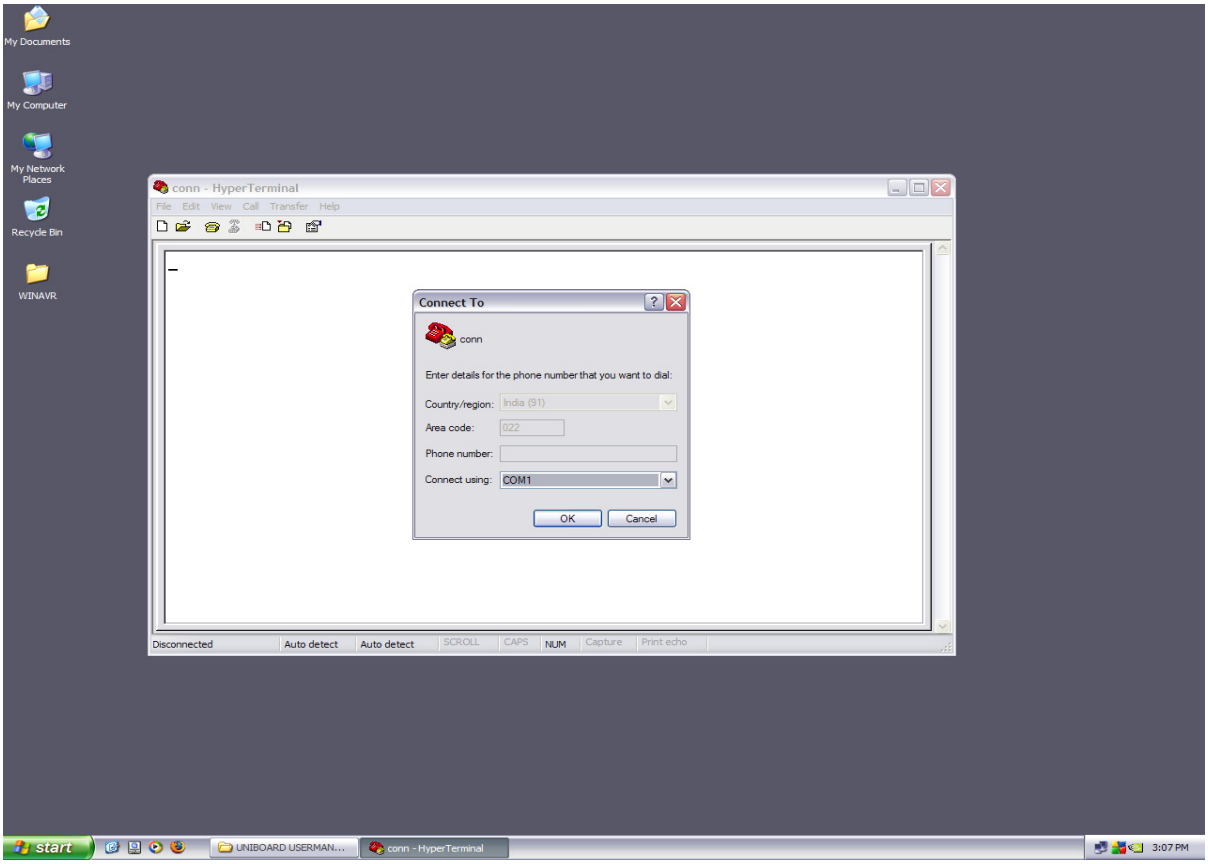
STEP 1: Run Hyper Terminal start menu->all programs->accessories->Communications->HyperTerminal.

STEP 2: Type name for make new connection

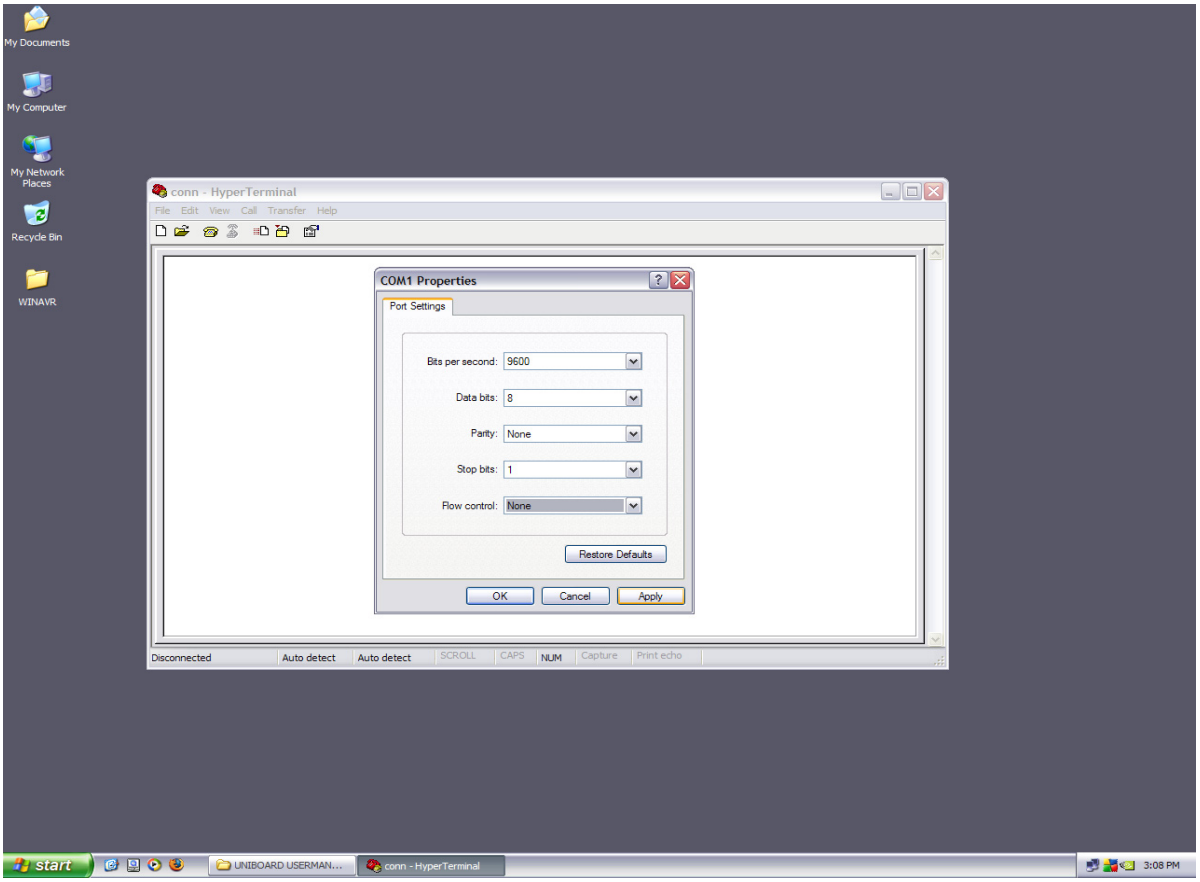


STEP 3: Select the serial port (usually it is COM1)


My Documents
My Computer
My Network Places
Recycle Bin
WINAVR
conn - HyperTerminal
File Edit View Call Transfer Help
COM1 Properties
Port Settings
Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
Restore Defaults
OK
Cancel
Apply
Disconnected
Auto detect
Auto detect
SCROLL
CAPS
NUM
Capture
Print echo
start
UNIBOARD USERMAN...
conn - HyperTerminal
3:08 PM



STEP 4: Select the Baud rate (9600), Data bits (8 bits), parity (none), stop bits (1), and flow control (none) as shown below.

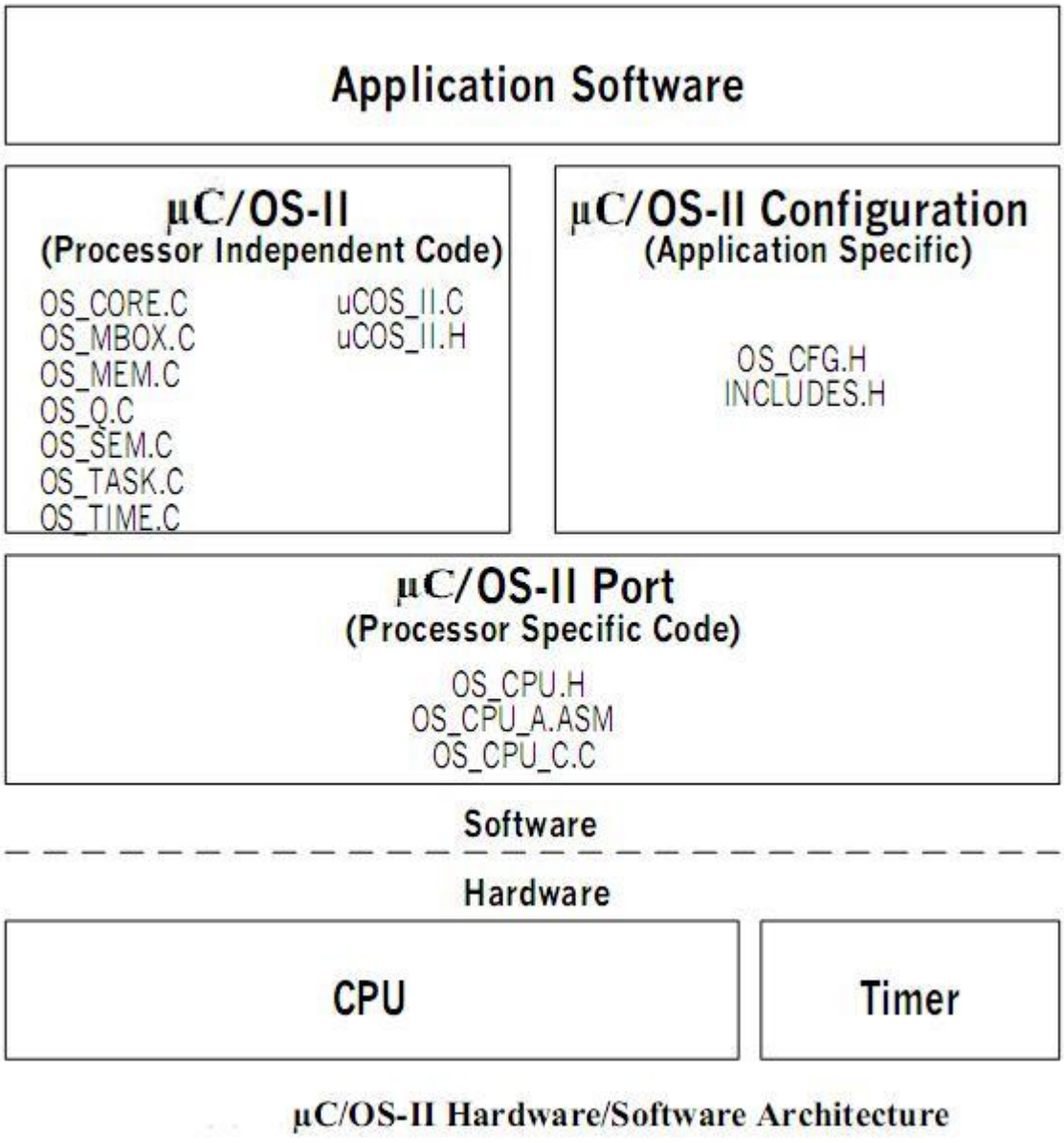


STEP 5: You can even save this connection from File->save so that you need not set all the settings again.

Getting Started with RTOS (uC/OS-II) on Windows OS

uC/OS-II Hardware and Software Architecture

Before starting of with the uC/OS-II programming you need to know its file structure i.e. how the uC/OS-II source code and its ports are structured. The diagram below gives an idea of the way in which these files have been maintained. For further information on this, refer to MicroC/OS-II, The Real Time Kernel by Jean Labrosse.



µC/OS-II Hardware/Software Architecture

Every RTOS project/source code consists of (as shown in the above figure):

- **Portable kernel code:** This is generally a code written in ANSI C which can be ported to any platform fulfilling certain requirements like memory footprint, computational power, number of registers etc.
- **BSP or port files:** These are specific to an architecture, so for every architecture / board there is a unique BSP / port files.
- **Configuration files:** These are files which can be used to optimize the kernel and reduce the memory footprint of the kernel image based on what features of the kernel would be used by any application program.
- **Application program:** This is generally written on top of the kernel code and utilizes kernel API's and functions exported by the OS for the users of the RTOS.

Code, Compile and program the RTOS (uC/OS-II) programs on Linux OS

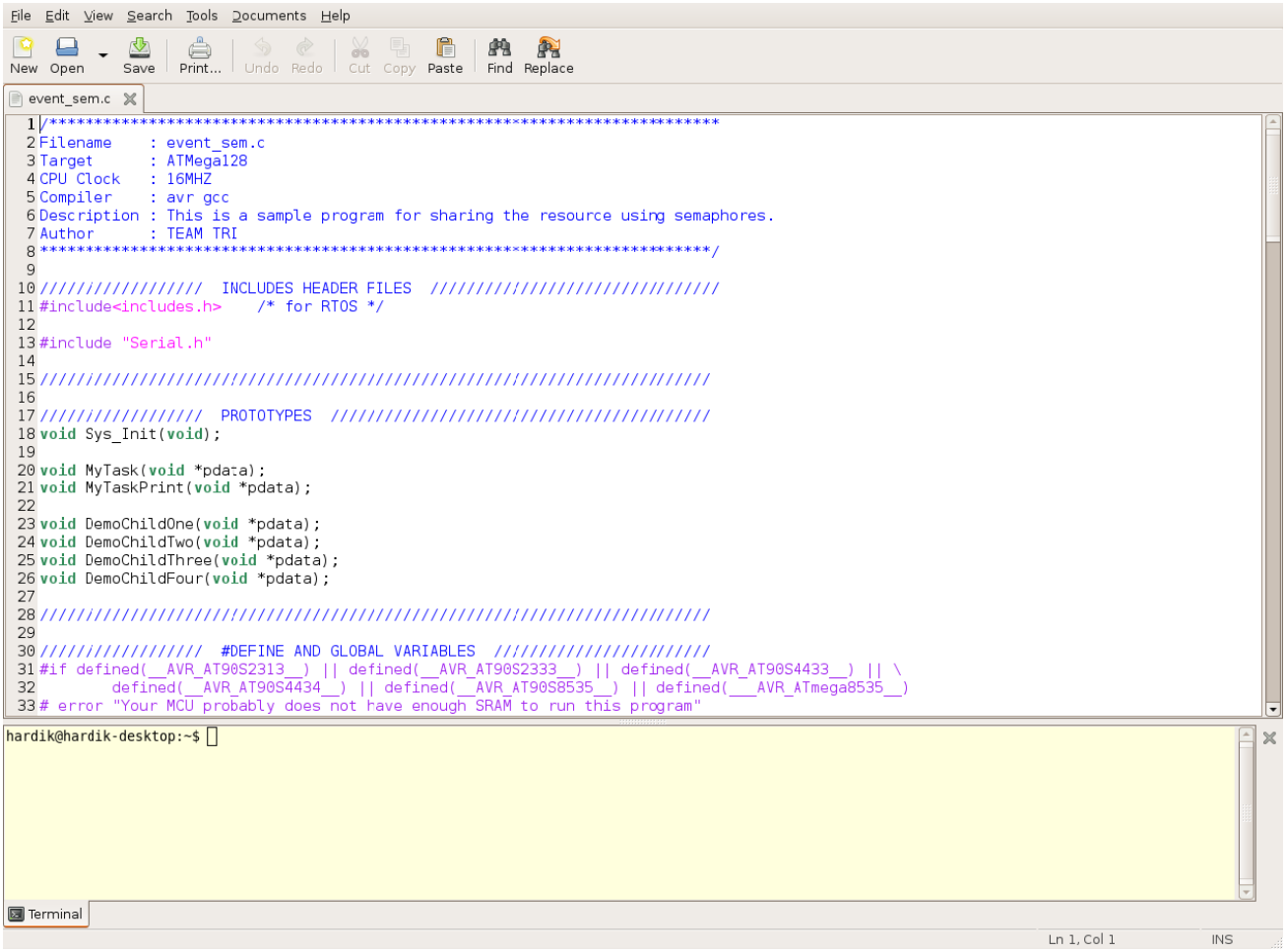
NOTE : You Should have the required software tools i.e. text editor, compiler, programmer and serial terminal as shown in the section sub-section **Software Installations for Linux OS**. Also refer to the sub-section **Getting started on Linux**.

STEP 1: Copy the "**Micrium**" directory from the UNIBOARD CONTENTS from the path " \UNIBOARD CONTENTS\UNIBOARD_UCOS_DOCS" to the **user's home directory**.

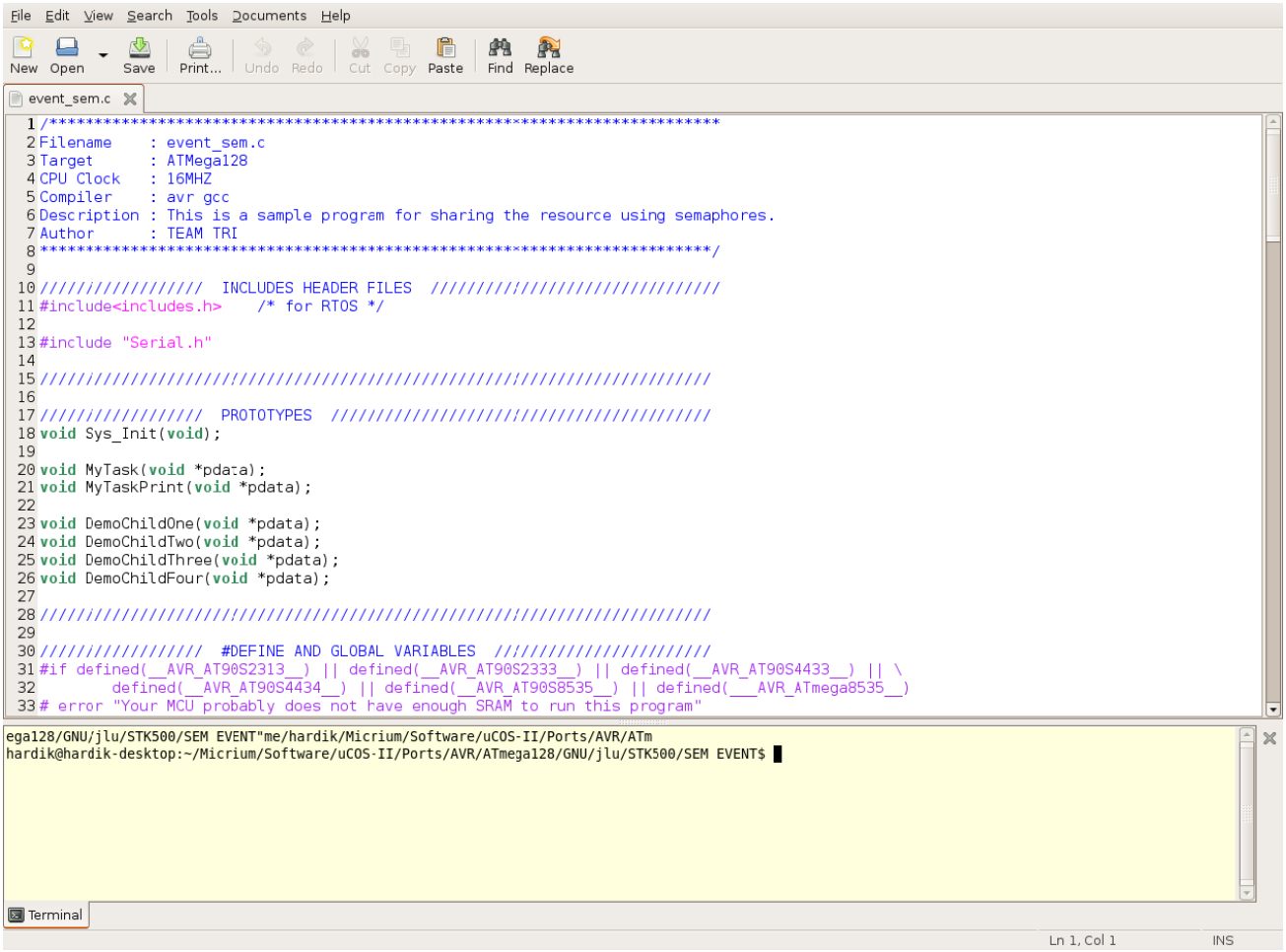
NOTE: You can go to the user's home directory from menu Places >> Home Folder.

STEP 2: Copy the sample codes given in the UNIBOARD CONTENTS from the path "\UNIBOARD CONTENTS\UNIBOARD_SAMPLE_CODES\ RTOS" onto your disk. In the project / sample code that you want to execute, **RENAME Makefile_LINUX** to **Makefile.**

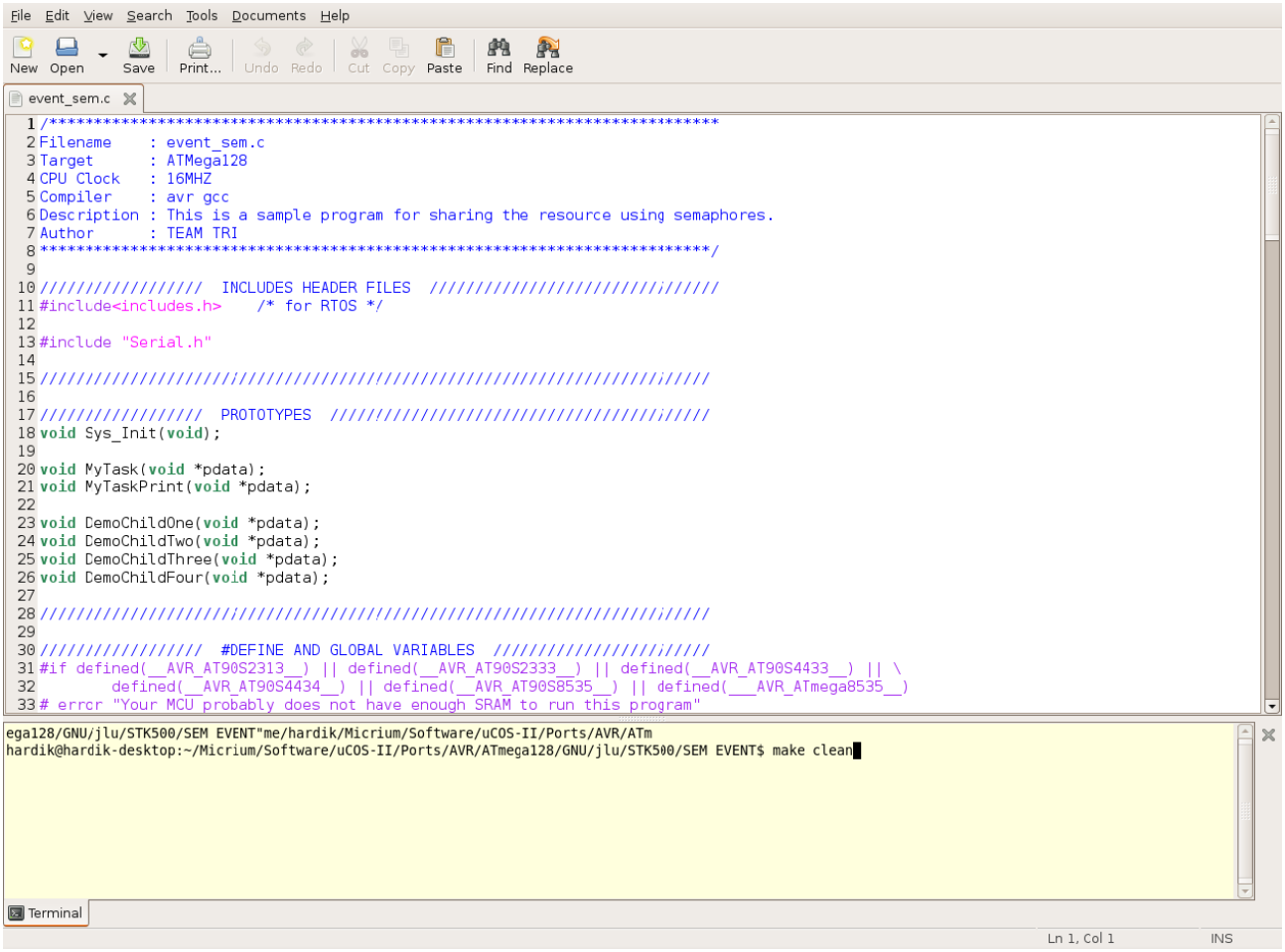
STEP 3: Open the event_sem.c file using Gedit Editor (**refer to the sub-section getting started on Linux**).



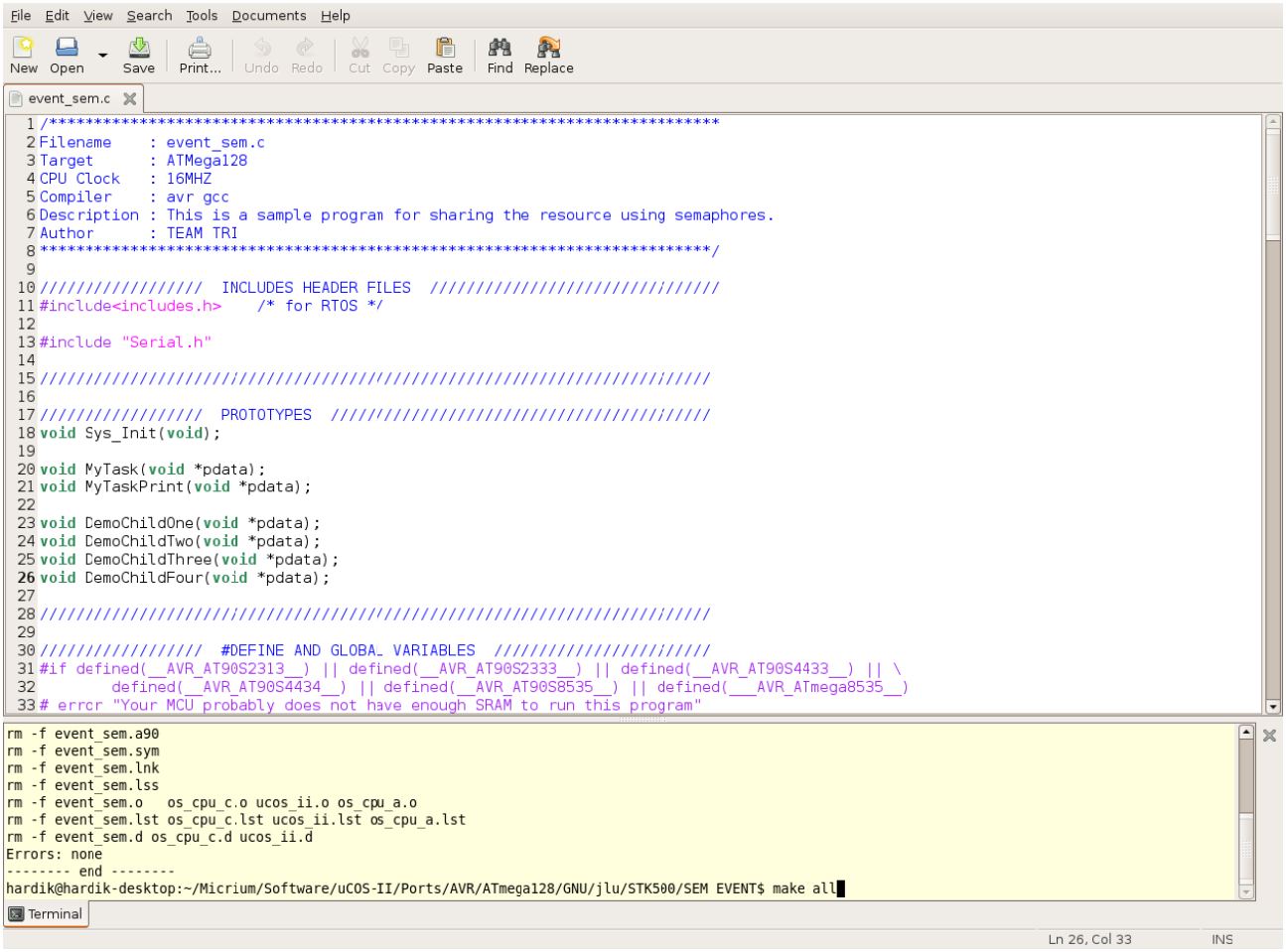
STEP 4: Go to the current program directory from Gedit Embedded terminal using cd command or right click on Bottom terminal and select change directory (**refer to the sub-section getting started on Linux**).



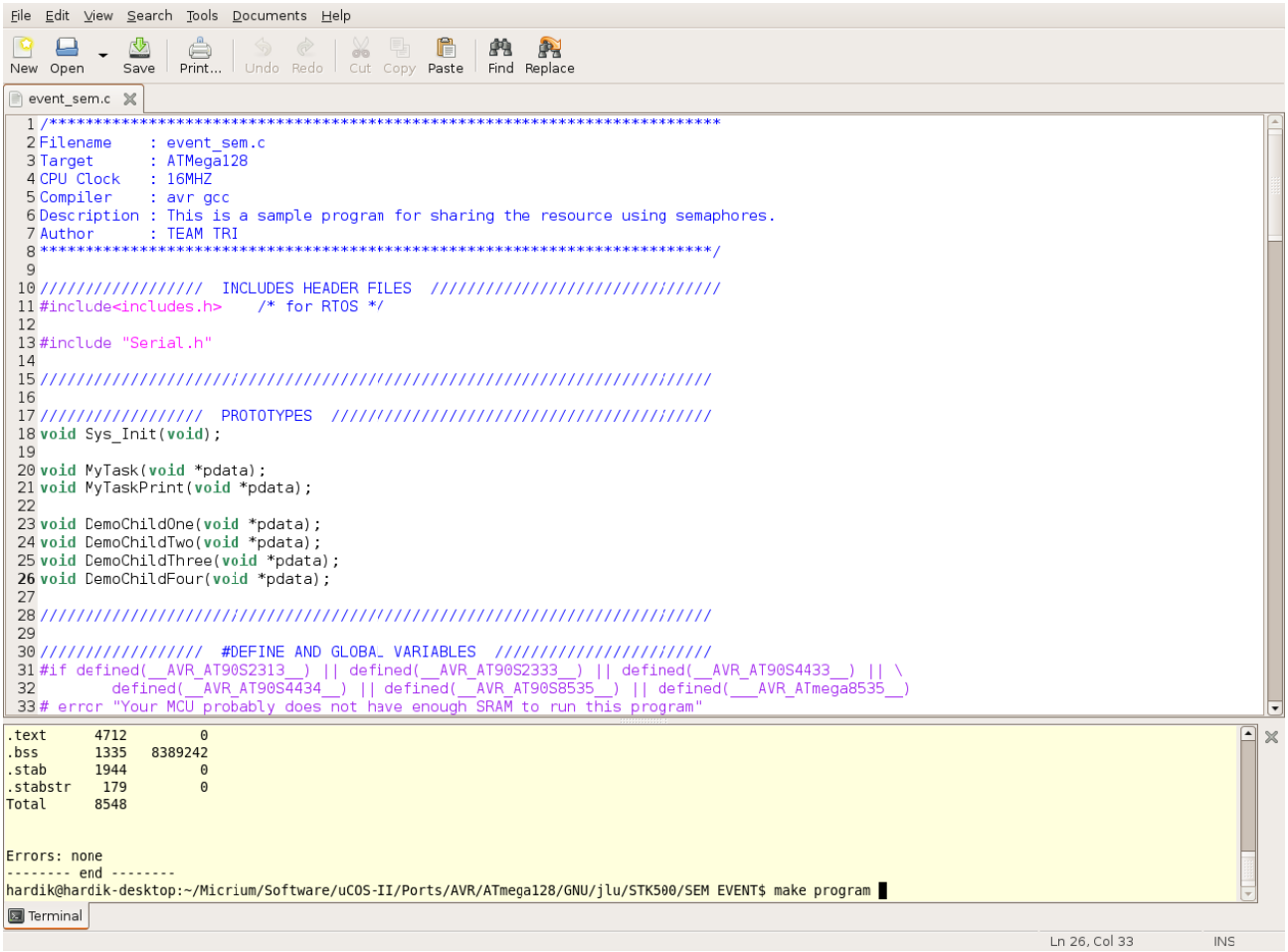
STEP 5: Type make clean on Gedit Embedded terminal.



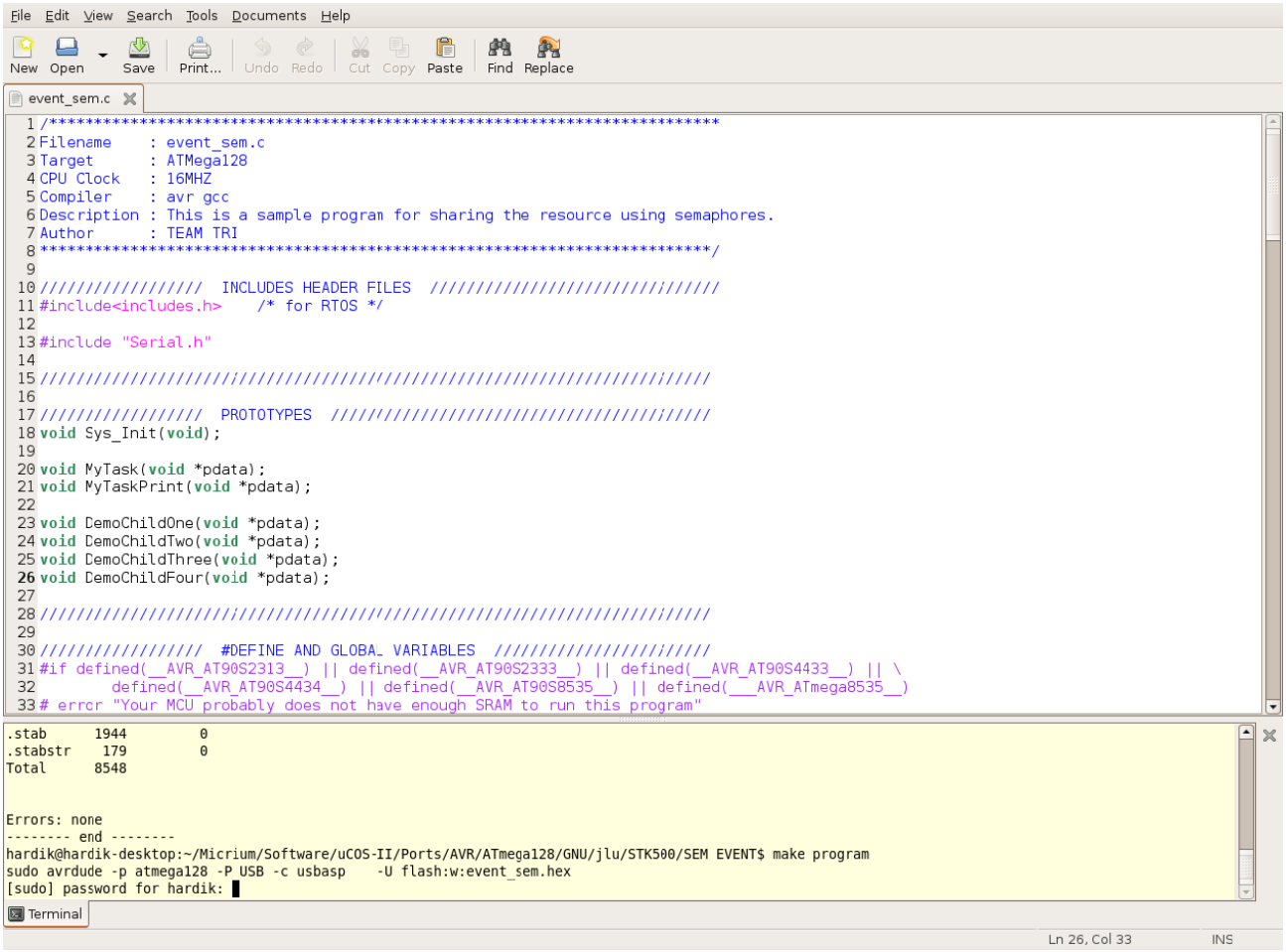
STEP 6: Type make all on Gedit Embedded terminal.



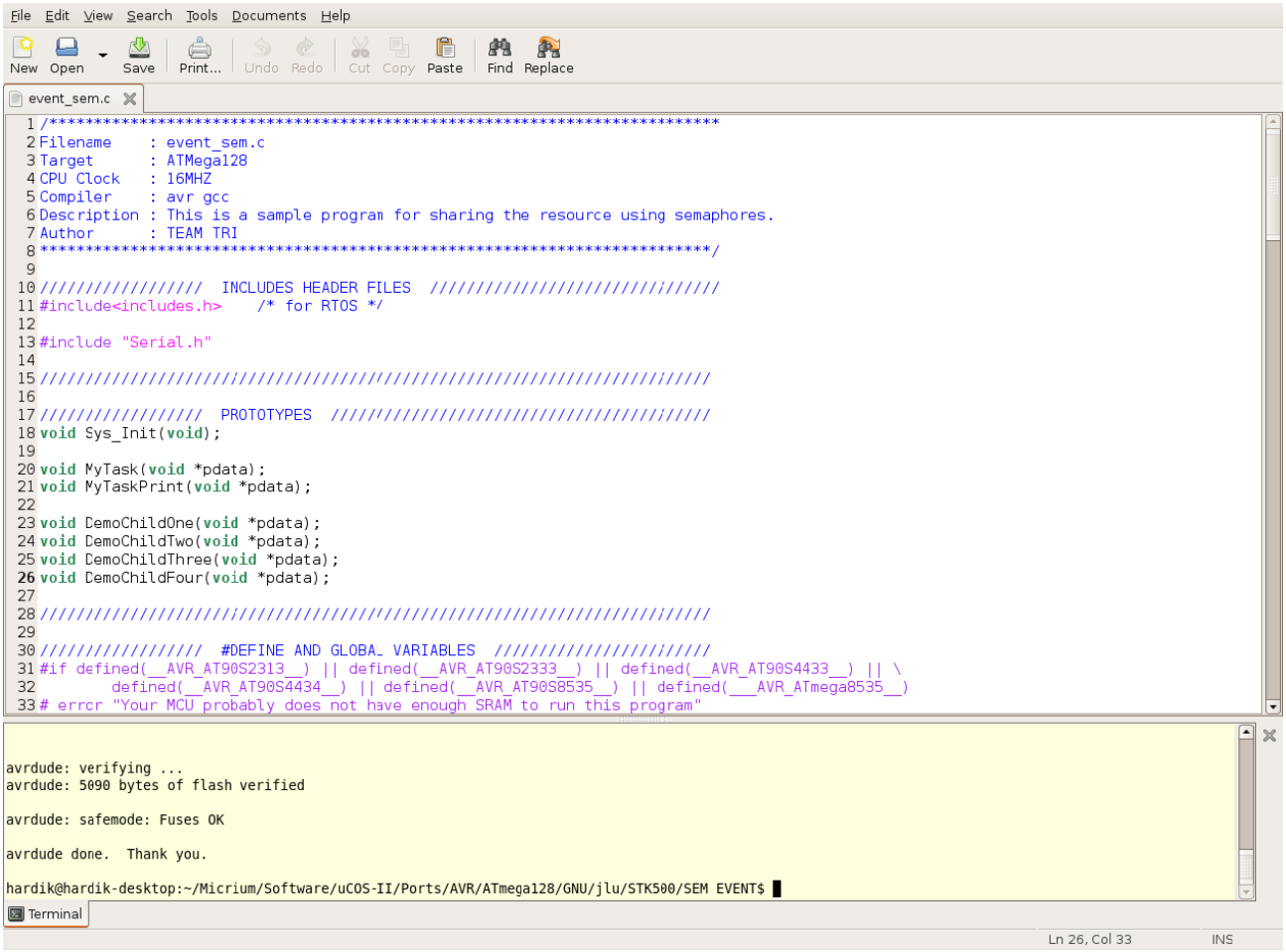
STEP 7: Type make program on Gedit Embedded terminal.



STEP 8: Enter the root password to program the Board and press enter (if prompted).

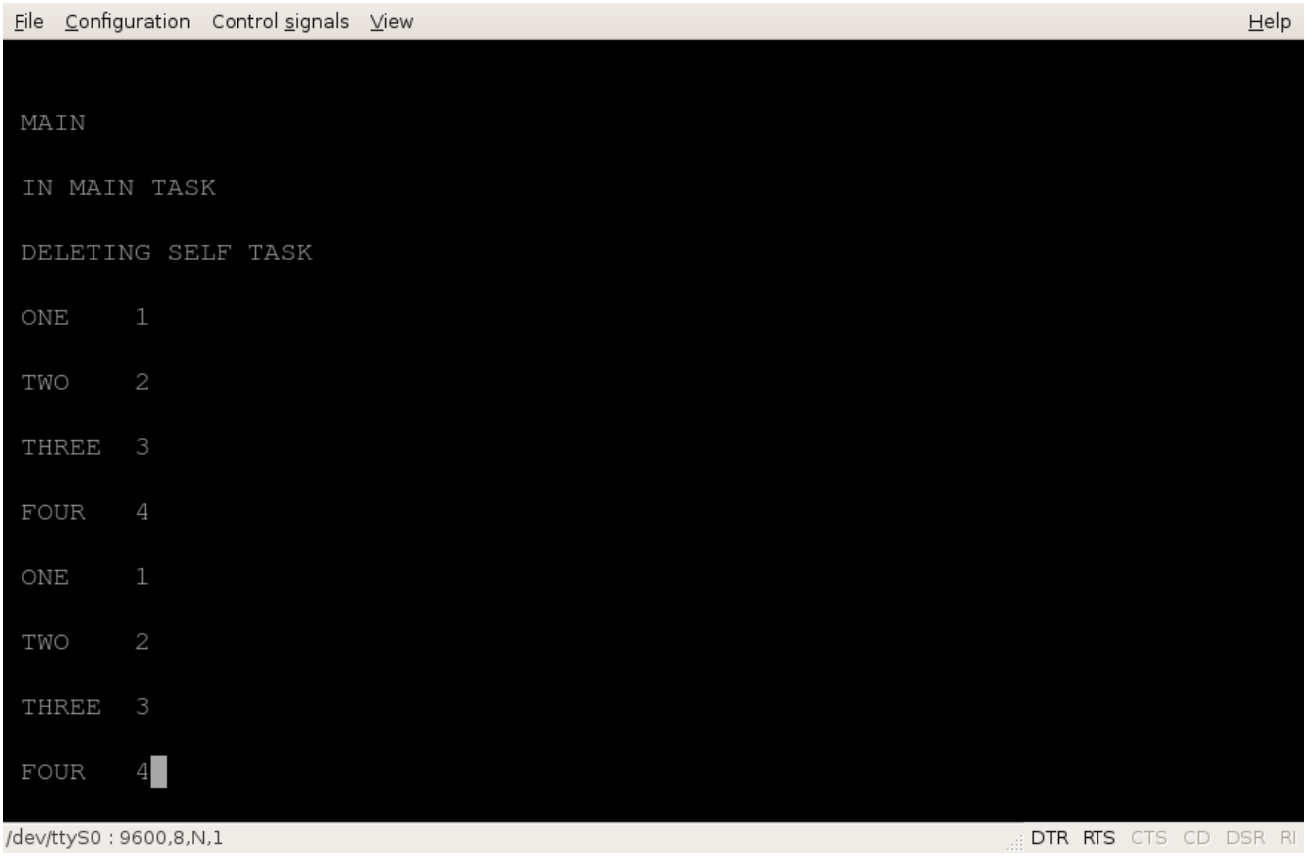


STEP 9: Turn OFF the uNiBoard after the program has been loaded successfully as shown below.



STEP 10: Open Gtkterm (**refer to the sub-section Gtkterm Configurations for setting the Baud rate, Parity, Stop bits**).

STEP 11: Turn ON the uNiBoard and Output will be displayed as shown below on Gtkterm.



Code, Compile and program the RTOS (uC/OS-II) programs on Windows OS

NOTE: You Should have the required software tools i.e. text editor, compiler, programmer, serial terminal and required drivers for programmer as shown in the sub-section Software Installations for Windows OS.

STEP 1: Copy the Micrium directory from CD Contents directory path as "\UNIBOARD CONTENTS\UNIBOARD_UCOS_DOCS\" to "C:\".

STEP 2: Go to the UNIBOARD CONTENT on CD in the directory \UNIBOARD CONTENTS\UNIBOARD_SAMPLE_CODES\ RTOS \" and copy it your disk.

STEP 3: Go to the RTOS directory that you have just loaded on your disk. Select one of the sample code e.g: 3_SEMAPHORES. Rename the Makefile_WINDOWS to Makefile in the current example directory.

STEP 4: Open the Makefile using Text Editor and search for the TARGET label. In this example it is event_sem.c so assign TARGET = event_sem as shown below at line 47.

NOTE 1: In Makefile Target = Filename should be without any extensions as shown below at line 47.

NOTE 2: Check the Path setting for uCOS-II Makefile as shown below (for Windows OS only) at line 62 and 63. See that it is complying with the path shown below in the screenshot.

NOTE 3: Check the programmer's Avrdude options as shown below (for Windows OS only) at line 208.

Programmer's Notepad - makefile

```
1   # WinAVR Sample makefile written by Eric B. Weddington, Jörg
    Wunsch, et al.
2   # Released to the Public Domain
3   # Please read the make user manual!
4   #
5   # Modified to be used with uC/OS-II by Julius Luukko
    (Julius.Luukko@lut.fi)
6   # 2003-06-27.
7   #
8   # Some more changes in order to this be usable with uC/OS-II
    version 2.52
9   # and 2.70 (Julius Luukko 2003-07-21).
10  #
11  # Moved erasing the MCU from separate command line to the same
    command
12  # line as programming (Julius Luukko 2003-10-14)
13  #
14  # Change UCOSDIR and PORTDIR to correspond to your installation.
    INCDIR1
15  # and INCDIR2 might also need to be changed. avrdude settings must
    also be
16  # checked!
17  #
18  # On command line:
19  #
20  # make all = Make software.
21  #
22  # make clean = Clean out built project files.
23  #
24  # make coff = Convert ELF to AVR COFF (for use with AVR Studio 3.x
    or VMLAB).
25  #
26  # make extcoff = Convert ELF to AVR Extended COFF (for use with AVR
    Studio
27  #                4.07 or greater).
28  #
29  # make program = Download the hex file to the device, using
    avrdude.  Please
30  #                customize the avrdude settings below first!
31  #
32  # make filename.s = Just compile filename.c into the assembler code
    only
33  #
34  # make tar = Create a tar.gz of the project files
35  #
36  # To rebuild project do "make clean" then "make all".
37  #
38
39
40  # MCU name
41  MCU = atmega128
42
43  # Output format. (can be srec, ihex, binary)
44  FORMAT = ihex
45
46  # Target file name (without extension).
47  TARGET = event_sem
48
49  # Optimization level, can be [0, 1, 2, 3, s]. 0 turns off
    optimization.
50  # (Note: 3 is not always the best optimization level. See avr-libc
```

Page 1, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
        FAQ.)
51      OPT = 2
52
53      # Where uC/OS-II and the port are
54      # For *nix systems you can use something like the next two lines if
        you
55      # have uC/OS-II v2.70
56      # UCOSDIR = $(wildcard ~/Micrium/Software/uCOS-II/Source)
57      # PORTDIR = $(wildcard
        ~/Micrium/Software/uCOS-II/Ports/AVR/ATmega128/GNU/jlu)
58      # If you have uC/OS-II v2.52, use
59      #UCOSDIR = $(wildcard ~/ucos-II/software/uCOS-II/Source)
60      #PORTDIR = $(wildcard
        ~/Micrium/Software/uCOS-II/Ports/AVR/ATmega128/GNU/jlu)
61      # With Windows, use something like the next lines
62      PORTDIR = c:/Micrium/Software/uCOS-II/Ports/AVR/ATmega128/GNU/jlu
63      UCOSDIR = c:/Micrium/SOFTWARE/uCOS-II/Source
64
65      INCDIR1 = $(UCOSDIR)
66      INCDIR2 = $(PORTDIR)
67      # INCDIR1   = $(UCOSDIR)/SOURCE
68      # INCDIR2   = $(PORTDIR)/SOURCE
69
70      # List C source files here. (C dependencies are automatically
        generated.)#
71      SRC = $(TARGET).c
72
73      # If there is more than one source file, append them above, or
        adjust and
74      # uncomment the following:
75      #SRC += foo.c bar.c
76
77      # You can also wrap lines by appending a backslash to the end of
        the line:
78      #SRC += baz.c \
79      #xyzzy.c
80
81      # uC/OS-II AVR port c source and the main uC/OS-II source
82      # If you have uC/OS-II v2.52, use the second line!
83      # UCOSSRC = $(PORTDIR)/os_cpu_c.c $(PORTDIR)/os_dbg.c
        $(UCOSDIR)/ucos_ii.c
84      UCOSSRC = $(PORTDIR)/os_cpu_c.c $(UCOSDIR)/ucos_ii.c
85
86      # List Assembler source files here.
87      # Make them end in .asm. This is different from the WinAVR example
        makefile
88      ASRC =
89
90      # uC/OS-II AVR port assembler source
91      UCOSASRC = $(PORTDIR)/os_cpu_a.asm
92
93      # List all header files, which are part of this project (only used
        for tar)
94      HDRS = os_cfg.h includes.h
95
96      # Other files you want to be included in the tar
97      OTHER = README
98
99      # Optional compiler flags.
100     #  -g:        generate debugging information (for GDB, or for COFF
        conversion)
```

Page 2, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
101  #  -O*:       optimization level
102  #  -f...:     tuning, see gcc manual and avr-libc documentation
103  #  -Wall...:  warning level
104  #  -Wa,...:   tell GCC to pass this to the assembler.
105  #    -ahlms:  create assembler listing
106  CFLAGS = -O$(OPT) \
107      -funsigned-char -funsigned-bitfields -fpack-struct
     -fshort-enums \
108      -Wall -Wstrict-prototypes \
109      -Wa,-ahlms=$(notdir $(<:.c=.lst))
110
111
112
113  # Optional assembler flags.
114  #  -Wa,...:   tell GCC to pass this to the assembler.
115  #  -ahlms:    create listing
116  #  -gstabs:   have the assembler create line number information;
     note that
117  #             for use in COFF files, additional information about
     filenames
118  #             and function names needs to be present in the
     assembler source
119  #             files -- see avr-libc docs [FIXME: not yet described
     there]
120  ASFLAGS = -Wa,-ahlms=$(notdir $(<:.asm=.lst)),-gstabs
121
122
123
124  # Optional linker flags.
125  #  -Wl,...:   tell GCC to pass this to linker.
126  #  -Map:      create map file
127  #  --cref:    add cross reference to  map file
128  LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
129
130
131
132  # Additional libraries
133  #
134  # Minimalistic printf version
135  #LDFLAGS += -Wl,-u,vfprintf -lprintf_min
136  #
137  # Floating point printf version (requires -lm below)
138  #LDFLAGS +=  -Wl,-u,vfprintf -lprintf_flt
139  #
140  # -lm = math library
141  # LDFLAGS += -lm
142
143
144
145
146  # Programming support using avrdude. Settings and variables.
147
148  # Programming hardware: avrisp stk500 avr910 pavr stk200 pony-stk200
149  #                       dt006 bascom alf
150  # Type: avrdude -c ?
151  # to get a full listing.
152  #
153  AVRDUDE_PROGRAMMER = usbasp
154
155  #amarender commented next line
156  #AVRDUDE_PORT = com1       # programmer connected to serial device
```

Page 3, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
157  #AVRDUDE_PORT = lpt1    # programmer connected to parallel port
158  AVRDUDE_PORT = USB
159  #AVRDUDE_PORT = /dev/1-6       # serial device on *nix
160  # AVRDUDE_PORT = /dev/ttyS0    # serial device on *nix
161  #amarender
162  AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
163  AVRDUDE_ERASE = -e
164  AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
165
166
167  # Uncomment the following if you want avrdude's erase cycle counter.
168  # Note that this counter needs to be initialized first using -Yn,
169  # see avrdude manual.
170  #AVRDUDE_ERASE += -y
171  #
172  # Uncomment the following if you do /not/ wish a verification to be
173  # performed after programming the device.
174  #AVRDUDE_FLAGS += -V
175  #
176  # Increase verbosity level.  Please use this when submitting bug
177  # reports about avrdude. See
     <http://savannah.nongnu.org/projects/avrdude>
178  # to submit bug reports.
179  #AVRDUDE_FLAGS += -v -v
180
181  #amarender
182  AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
183  AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
184  AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
185  AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)
186
187  #
     ------------------------------------------------------------------
     ------
188
189  # Define directories, if needed.
190  DIRAVR = c:/winavr
191  DIRAVRBIN = $(DIRAVR)/bin
192  DIRAVRUTILS = $(DIRAVR)/utils/bin
193  DIRINC = .
194  DIRLIB = $(DIRAVR)/avr/lib
195
196
197  # Define programs and commands.
198  SHELL = sh
199
200  CC = avr-gcc
201
202  OBJCOPY = avr-objcopy
203  OBJDUMP = avr-objdump
204  SIZE = avr-size
205
206
207  # Programming support using avrdude.
208  AVRDUDE = avrdude
209
210
211  REMOVE = rm -f
212  COPY = cp
213
214  HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
```

Page 4, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
215  ELFSIZE = $(SIZE) -A $(TARGET).elf
216
217  FINISH = echo Errors: none
218  BEGIN = echo -------- begin --------
219  END = echo --------  end  --------
220
221
222
223
224  # Define all object files.
225  OBJ = $(SRC:.c=.o) $(ASRC:.asm=.o)
226  OBJ += $(notdir $(UCOSSRC:.c=.o)) $(notdir $(UCOSASRC:.asm=.o))
227
228  # Define all listing files.
229  LST = $(OBJ:.o=.lst)
230
231  # Combine all necessary flags and optional flags.
232  # Add target processor to flags.
233  ALL_CFLAGS = -mmcu=$(MCU) -I. -I$(INCDIR1) -I$(INCDIR2) $(CFLAGS)
234  ALL_ASFLAGS = -mmcu=$(MCU) -I. -I$(INCDIR1) -I$(INCDIR2) -x
     assembler-with-cpp $(ASFLAGS)
235
236
237
238  # Default target.
239  all: begin gccversion sizebefore $(TARGET).elf $(TARGET).hex
     $(TARGET).eep \
240      $(TARGET).lss sizeafter finished end
241
242
243  # Eye candy.
244  # AVR Studio 3.x does not check make's exit code but relies on
245  # the following magic strings to be generated by the compile job.
246  begin:
247      @$(BEGIN)
248
249  finished:
250      @$(FINISH)
251
252  end:
253      @$(END)
254
255
256  # Display size of file.
257  sizebefore:
258      @if [ -f $(TARGET).elf ]; then echo Size before:; $(ELFSIZE);fi
259
260  sizeafter:
261      @if [ -f $(TARGET).elf ]; then echo Size after:; $(ELFSIZE);fi
262
263
264
265  # Display compiler version information.
266  gccversion :
267      $(CC) --version
268
269
270
271
272  # Convert ELF to COFF for use in debugging / simulating in
273  # AVR Studio or VMLAB.
```

Page 5, 1/7/2009 • 8:38:22 PM

Programmer's Notepad - makefile

```
274  COFFCONVERT=$(OBJCOPY) --debugging \
275      --change-section-address .data-0x800000 \
276      --change-section-address .bss-0x800000 \
277      --change-section-address .noinit-0x800000 \
278      --change-section-address .eeprom-0x810000
279
280
281  coff: $(TARGET).elf
282      $(COFFCONVERT) -O coff-avr $< $(TARGET).cof
283
284
285  extcoff: $(TARGET).elf
286      $(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof
287
288
289
290
291  # Program the device.  The magic trickery below examines the .eep hex
292  # file whether the size is > 0, and if so, reprograms the EEPROM as
293  # well.  Just delete these lines if you don't want this feature (like
294  # on the ATmegas with the EESAVE fuse bit set).
295  program: $(TARGET).hex $(TARGET).eep
296  #   $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_ERASE)
297  #amarender
298      $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH)
     $(AVRDUDE_WRITE_EEPROM)
299  #amarender commented next line
300  #   $(AVRDUDE) $(AVRDUDE_FLAGS) -i $(TARGET).hex $(AVRDUDE_ERASE)
301  #   @$(SIZE) --target=$(FORMAT) $(TARGET).eep | while read line; \
302  #   do \
303  #       set -- $$line; \
304  #       if [ "x$$1" != "x0" ] ; then continue; fi; \
305  #       if [ "$$2" -ne 0 ] ; then \
306  #           echo $(AVRDUDE) $(AVRDUDE_FLAGS) -m eeprom -i
     $(TARGET).eep; \
307  #           $(AVRDUDE) $(AVRDUDE_FLAGS) -m eeprom -i $(TARGET).eep; \
308  #           break; \
309  #       fi; \
310  #   done
311
312  tar:
313      tar czf $(TARGET)source.tgz $(SRC) $(ASRC) $(HDRS) $(OTHER)
     makefile
314
315
316  # Create final output files (.hex, .eep) from ELF output file.
317  %.hex: %.elf
318      $(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@
319
320  %.eep: %.elf
321      -$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
322      --change-section-lma .eeprom=0 -O $(FORMAT) $< $@
323
324  # Create extended listing file from ELF output file.
325  %.lss: %.elf
326      $(OBJDUMP) -h -S $< > $@
327
328
329
330  # Link: create ELF output file from object files.
331  .SECONDARY : $(TARGET).elf
```

Page 6, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
332  .PRECIOUS : $(OBJ)
333  %.elf: $(OBJ)
334      $(CC) $(ALL_CFLAGS) $(OBJ) --output $@ $(LDFLAGS)
335
336
337  # Compile: create object files from C source files.
338  %.o : %.c
339      $(CC) -c $(ALL_CFLAGS) $< -o $@
340
341
342  # Assemble: create object files from assembler source files.
343  %.o : %.asm
344      $(CC) -c $(ALL_ASFLAGS) $< -o $@
345
346
347
348
349  # Target: clean project.
350  clean: begin clean_list finished end
351
352  clean_list :
353      $(REMOVE) $(TARGET).hex
354      $(REMOVE) $(TARGET).eep
355      $(REMOVE) $(TARGET).obj
356      $(REMOVE) $(TARGET).cof
357      $(REMOVE) $(TARGET).elf
358      $(REMOVE) $(TARGET).map
359      $(REMOVE) $(TARGET).obj
360      $(REMOVE) $(TARGET).a90
361      $(REMOVE) $(TARGET).sym
362      $(REMOVE) $(TARGET).lnk
363      $(REMOVE) $(TARGET).lss
364      $(REMOVE) $(OBJ)
365      $(REMOVE) $(LST)
366      $(REMOVE) $(SRC:.c=.d) $(notdir $(UCOSSRC:.c=.d))
367
368
369
370  # Automatically generate C source code dependencies.
371  # (Code originally taken from the GNU make user manual and modified
372  # (See README.txt Credits).)
373  #
374  # Note that this will work with sh (bash) and sed that is shipped
     with WinAVR
375  # (see the SHELL variable defined above).
376  # This may not work with other shells or other seds.
377  VPATH=.:$(INCDIR1):$(INCDIR2)
378  %.d: %.c
379      set -e; $(CC) -MM $(ALL_CFLAGS) $< \
380      | sed 's/\($(*F)\)\.o[ :]*/\1.o \1.d : /g' > $(@F); \
381      [ -s $@ ] || rm -f $@
382
383
384  # Remove the '-' if you want to see the dependency files generated.
385  -include $(SRC:.c=.d) $(notdir $(UCOSSRC:.c=.d))
386
387
388
389  # Listing of phony targets.
390  .PHONY : all begin finish end sizebefore sizeafter gccversion coff
     extcoff \
```

Page 7, 1/7/2009 • 6:38:22 PM

Programmer's Notepad - makefile

```
391         clean clean_list program
392
393
394
```

Page 8, 1/7/2009 8:38:22 PM

STEP 5: Open event_sem.c from the current directory and the code will be displayed as shown below.

Programmer's Notepad - event_sem.c

```
1   /*************************************************************
    ********************
2   Filename    : event_sem.c
3   Target      : ATMega128
4   CPU Clock   : 16MHZ
5   Compiler    : avr gcc
6   Description : This is a sample program for sharing the
    resource using semaphores.
7   Author      : TEAM TRI
8   *************************************************************
    ******************/
9
10  ////////////////////  INCLUDES HEADER FILES
    ////////////////////////////////
11  #include<includes.h>    /* for RTOS */
12
13  #include "Serial.h"
14
15  ////////////////////////////////////////////////////////////////
    ////////////////////
16
17  ////////////////////  PROTOTYPES
    ////////////////////////////////////////
18  void Sys_Init(void);
19
20  void MyTask(void *pdata);
21  void MyTaskPrint(void *pdata);
22
23  void DemoChildOne(void *pdata);
24  void DemoChildTwo(void *pdata);
25  void DemoChildThree(void *pdata);
26  void DemoChildFour(void *pdata);
27
28  ////////////////////////////////////////////////////////////////
    ////////////////////
29
30  ////////////////////  #DEFINE AND GLOBAL VARIABLES
    ////////////////////////
31  #if defined(__AVR_AT90S2313__) ||
    defined(__AVR_AT90S2333__) ||
    defined(__AVR_AT90S4433__) || \
32          defined(__AVR_AT90S4434__) ||
    defined(__AVR_AT90S8535__) ||
    defined(__AVR_ATmega8535__)
33  # error "Your MCU probably does not have enough SRAM
    to run this program"
34  #elif defined(__AVR_AT90S4414__)
35  # error "Your MCU probably does not have enough
    program memory to run this program"
36  #elif defined(__AVR_AT90S8515__) || \
37        defined(__AVR_ATmega161__) ||
```

Page 1, 1/7/2009 • 5:08:18 PM

Programmer's Notepad - event_sem.c

```
        defined(__AVR_ATmega162__) ||
        defined(__AVR_ATmega163__) || \
38              defined(___AVR_ATmega8515__) ||
        defined(__AVR_ATmega16__) || defined(__AVR_ATmega32__)
        || \
39              defined(__AVR_ATmega8__)
40      #   define TCCR0VAL (_BV(CS02) | _BV(CS00))
41      #elif defined(__AVR_ATmega64__) ||
        defined(__AVR_ATmega128__) || defined(__AVR_ATmega103__)
42      #   define TCCR0VAL (_BV(CS02) | _BV(CS01) | _BV(CS00))
43      #else
44      #  error "Don't know what kind of MCU you are
        compiling for"
45      #endif
46
47      #define MYTASK_STACK 128
48      #define CHILD_TASK_STACK 128
49
50      /****************** STACKS FOR TASK
        ************************/
51      OS_STK MyTaskStack[MYTASK_STACK];
52      OS_STK MyTaskPrintStack[ MYTASK_STACK ];
53
54      OS_STK ChildOneStack[ MYTASK_STACK ];
55      OS_STK ChildTwoStack[ MYTASK_STACK ];
56      OS_STK ChildThreeStack[ MYTASK_STACK ];
57      OS_STK ChildFourStack[ MYTASK_STACK ];
58      /*****************************************************
        ******/
59      OS_EVENT *MySem_RESOURCE;   /* semaphore to guard
        shared resource */
60
61      /* SHARED VARIABLE */
62      char G_S_var = '7';
63
64      int main(void)
65      {
66
67          Sys_Init();         /* INITIALIZE THE SYSTEM */
68
69          print_string("\n\r MAIN ");
70
71          OSInit();           /* INTIALIZE THE RTOS */
72
73          MySem_RESOURCE  = OSSemCreate(0);
74
75                          /* CREATE THE TASKS */
76          OSTaskCreate(MyTask, (void *)0, &MyTaskStack[
        MYTASK_STACK - 1 ], 0 );
77          OSTaskCreate(MyTaskPrint, (void *)0, &
        MyTaskPrintStack[ MYTASK_STACK - 1 ], 1 );
```

Page 2, 1/7/2009 • 5:08:18 PM

Programmer's Notepad - event_sem.c

```
 78
 79      OSStart();          /* START Scheduler */
 80
 81      while(1);           /* WAIT FORVER */
 82  }
 83
 84
 85  void Sys_Init(void)
 86  {
 87      /* INITIALIZE THE USART */
 88      Serial_Init();
 89  }
 90
 91  void MyTaskPrint(void *pdata)
 92  {
 93      INT8U  err;
 94
 95      for(;;)
 96      {
 97          OSSemPend(MySem_RESOURCE, 0 ,&err ); /* wait
     till a tasks updates the shared variable */
 98          USART_Transmit('\t');
 99          USART_Transmit(G_S_var);
100      }
101  }
102
103
104  void MyTask(void *pdata)
105  {
106      INT8U  err;
107
108      OS_ENTER_CRITICAL();
109          TCCR0=TCCR0VAL;
                                        /* Set TIMER0
     prescaler to CLK/1024 */
110          TIMSK=_BV(TOIE0);
                                      /* Enable TIMER0
     overflow interrupt */
111          TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024);
      /* Set the counter initial value    */
112      OS_EXIT_CRITICAL();
113
114      /* create 4 processes for updating the shared
     variable */
115
116      OSTaskCreate( DemoChildOne, ( void *)0,&
     ChildOneStack[ CHILD_TASK_STACK - 1 ], 2 );
117      OSTaskCreate( DemoChildTwo, ( void *)0,&
     ChildTwoStack[ CHILD_TASK_STACK - 1 ], 3 );
118      OSTaskCreate( DemoChildThree, ( void *)0,&
     ChildThreeStack[ CHILD_TASK_STACK - 1 ], 4 );
```

Page 3, 1/7/2009 • 5:08:18 PM

Programmer's Notepad - event_sem.c

```
119         OSTaskCreate( DemoChildFour, ( void *)0,&
        ChildFourStack[ CHILD_TASK_STACK - 1 ], 5 );
120
121         print_string("\n\r IN MAIN TASK ");
122         print_string("\n\r DELETING SELF TASK");
123         err = OSTaskDel(OS_PRIO_SELF);
124
125         for (;;) {
126             print_string("\n\r THIS WILL NEVER BE PRINTED");
127         }
128     }
129
130     void DemoChildOne(void *Onedata)
131     {
132         for(;;) {
133             print_string("\n\r ONE ");
134             G_S_var = '1';  /* Updating shared resource */
135             OSSemPost(MySem_RESOURCE);  /* SIGNAL TO PRINT
        TASK */
136             OSTimeDlyHMSM(0, 0, 1,0) ; /* make the task to
        wait for 1 second */
137         }
138     }
139
140     void DemoChildTwo(void *Twodata)
141     {
142         for(;;) {
143             print_string("\n\r TWO ");
144             G_S_var = '2';  /* Updating shared resource */
145             OSSemPost(MySem_RESOURCE);  /* SIGNAL TO PRINT
        TASK */
146             OSTimeDlyHMSM(0, 0, 1,0); /* make the task to
        wait for 1 second */
147         }
148     }
149     void DemoChildThree(void *Threedata)
150     {
151         for(;;) {
152                 print_string("\n\r THREE ");
153                 G_S_var = '3';  /* Updating shared
        resource */
154                 OSSemPost(MySem_RESOURCE);  /* SIGNAL TO
        PRINT TASK */
155                 OSTimeDlyHMSM(0, 0, 1, 0); /* make the
        task to wait for 1 second */
156             }
157     }
158
159     void DemoChildFour(void *Fourdata)
160     {
161         for(;;) {
```

Page 4, 1/7/2009 • 6:08:18 PM

Programmer's Notepad - event_sem.c

```
162                 print_string("\n\r FOUR ");
163                 G_S_var = '4';  /* Updating shared
    resource */
164                 OSSemPost(MySem_RESOURCE);  /* SIGNAL TO
    PRINT TASK */
165                 OSTimeDlyHMSM(0, 0, 1, 0); /* make the
    task to wait for 1 second */
166             }
167     }
168
```

Page 6, 1/7/2009 • 5:08:18 PM

NOTE: Check the code at line 108 to 112 in function named as void MyTask(void *pdata). This code is initializing the timer0 i.e. required by scheduler for correct time delays and this macro named as CPU_CLOCK_HZ and OS_TICKS_PER_SEC are defined in the file os_cfg.h as shown in next step.

STEP 6: Open os_cfg.h from the current directory and the code will be displayed as shown below.

Programmer's Notepad - os_cfg.h

```
1   /*
2   *********************************************************
    ***********************************************
3   *
    uC/OS-II
4   *                                          The
    Real-Time Kernel
5   *
6   *                          (c) Copyright 1992-2003,
    Jean J. Labrosse, Weston, FL
7   *                                           All Rights
    Reserved
8   *
9   *                                  uC/OS-II
    Configuration File for V2.7x
10  *
11  * File : OS_CFG.H
12  * By   : Jean J. Labrosse
13  *
14  * Modifications by Julius Luukko 2003-07-14
    (Julius.Luukko@lut.fi) for avr-gcc test program.
15  *
16  * - CPU_CLOCK_HZ must be defined here (used in
    os_cpu_a.asm)
17  * - defines a default stack size, which is used in all
    stack size definitions
18  *
19  * Your applications must define CPU_CLOCK_HZ!
20  *
21  *********************************************************
    ***********************************************
22  */
23
24  #ifndef OS_CFG_H
25  #define OS_CFG_H
26
27  #define CPU_CLOCK_HZ       16000000
28
29  #define OS_TASK_DEF_STK_SIZE    128    /* Default
    stack size                                          */
30
31                                         /*
    ---------------------- MISCELLANEOUS
    ----------------------- */
32  #define OS_ARG_CHK_EN             0    /* Enable (1)
    or Disable (0) argument checking                    */
33  #define OS_CPU_HOOKS_EN           1    /* uC/OS-II
    hooks are found in the processor port files         */
34
35  #define OS_DEBUG_EN               1    /* Enable(1)
    debug variables                                     */
```

Page 1, 1/7/2009 • 5:15:29 PM

Programmer's Notepad - os_cfg.h

```
36
37    #define OS_EVENT_NAME_SIZE        32    /* Determine
      the size of the name of a Sem, Mutex, Mbox or Q    */
38
39    #define OS_LOWEST_PRIO            12    /* Defines the
      lowest priority that can be assigned ...          */
40                                            /* ... MUST
      NEVER be higher than 63!                          */
41
42    #define OS_MAX_EVENTS              3    /* Max. number
      of event control blocks in your application       */
43    #define OS_MAX_FLAGS               1    /* Max. number
      of Event Flag Groups    in your application       */
44    #define OS_MAX_MEM_PART            1    /* Max. number
      of memory partitions                              */
45    #define OS_MAX_QS                  2    /* Max. number
      of queue control blocks in your application       */
46    #define OS_MAX_TASKS              11    /* Max. number
      of tasks in your application, MUST be >= 2        */
47
48    #define OS_SCHED_LOCK_EN           1    /*     Include
      code for OSSchedLock() and OSSchedUnlock()        */
49
50    #define OS_TASK_IDLE_STK_SIZE   OS_TASK_DEF_STK_SIZE
51                                            /* Idle task
      stack size (# of OS_STK wide entries)             */
52
53    #define OS_TASK_STAT_EN            1    /* Enable (1)
      or Disable(0) the statistics task                 */
54    #define OS_TASK_STAT_STK_SIZE   OS_TASK_DEF_STK_SIZE
55                                            /* Statistics
      task stack size (# of OS_STK wide entries)        */
56    #define OS_TASK_STAT_STK_CHK_EN    1    /* Check task
      stacks from statistic task                        */
57
58    #define OS_TICK_STEP_EN            1    /* Enable tick
      stepping feature for uC/OS-View                   */
59    #define OS_TICKS_PER_SEC          61    /* Set the
      number of ticks in one second                          */
60
61
62                                            /*
      ----------------------- EVENT FLAGS
      ----------------------- */
63    #define OS_FLAG_EN                 0    /* Enable (1)
      or Disable (0) code generation for EVENT FLAGS    */
64    #define OS_FLAG_WAIT_CLR_EN        0    /* Include code
      for Wait on Clear EVENT FLAGS                     */
65    #define OS_FLAG_ACCEPT_EN          0    /*     Include
      code for OSFlagAccept()                           */
66    #define OS_FLAG_DEL_EN             0    /*     Include
```

Page 2, 1/7/2009 • 5:15:29 PM

Programmer's Notepad - os_cfg.h

```
        code for OSFlagDel()                                    */
67      #define OS_FLAG_NAME_SIZE        32    /*
        Determine the size of the name of an event flag
        group    */
68      #define OS_FLAG_QUERY_EN          0    /*     Include
        code for OSFlagQuery()                                  */
69
70
71                                               /*
        -------------------- MESSAGE MAILBOXES
        --------------------- */
72      #define OS_MBOX_EN                0    /* Enable (1)
        or Disable (0) code generation for MAILBOXES      */
73      #define OS_MBOX_ACCEPT_EN         0    /*     Include
        code for OSMboxAccept()                                 */
74      #define OS_MBOX_DEL_EN            0    /*     Include
        code for OSMboxDel()                                    */
75      #define OS_MBOX_POST_EN           0    /*     Include
        code for OSMboxPost()                                   */
76      #define OS_MBOX_POST_OPT_EN       0    /*     Include
        code for OSMboxPostOpt()                                */
77      #define OS_MBOX_QUERY_EN          0    /*     Include
        code for OSMboxQuery()                                  */
78
79
80                                               /*
        ---------------------- MEMORY MANAGEMENT
        --------------------- */
81      #define OS_MEM_EN                 0    /* Enable (1)
        or Disable (0) code generation for MEMORY MANAGER */
82      #define OS_MEM_QUERY_EN           0    /*     Include
        code for OSMemQuery()                                   */
83      #define OS_MEM_NAME_SIZE         32    /*
        Determine the size of a memory partition
        name             */
84
85
86                                               /*
        ---------------- MUTUAL EXCLUSION SEMAPHORES
        --------------- */
87      #define OS_MUTEX_EN               0    /* Enable (1)
        or Disable (0) code generation for MUTEX          */
88      #define OS_MUTEX_ACCEPT_EN        0    /*     Include
        code for OSMutexAccept()                                */
89      #define OS_MUTEX_DEL_EN           0    /*     Include
        code for OSMutexDel()                                   */
90      #define OS_MUTEX_QUERY_EN         0    /*     Include
        code for OSMutexQuery()                                 */
91
92
93                                               /*
```

Page 3, 1/7/2009 • 5:15:29 PM

Programmer's Notepad - os_cfg.h

```
                        ---------------------- MESSAGE QUEUES
                        ---------------------- */
  94    #define OS_Q_EN                    0    /* Enable (1)
        or Disable (0) code generation for QUEUES                */
  95    #define OS_Q_ACCEPT_EN             0    /*     Include
        code for OSQAccept()                                     */
  96    #define OS_Q_DEL_EN                0    /*     Include
        code for OSQDel()                                        */
  97    #define OS_Q_FLUSH_EN              0    /*     Include
        code for OSQFlush()                                      */
  98    #define OS_Q_POST_EN               0    /*     Include
        code for OSQPost()                                       */
  99    #define OS_Q_POST_FRONT_EN         0    /*     Include
        code for OSQPostFront()                                  */
 100    #define OS_Q_POST_OPT_EN           0    /*     Include
        code for OSQPostOpt()                                    */
 101    #define OS_Q_QUERY_EN              0    /*     Include
        code for OSQQuery()                                      */
 102
 103
 104                                            /*
                        ----------------------- SEMAPHORES
                        ----------------------- */
 105    #define OS_SEM_EN                  1    /* Enable (1)
        or Disable (0) code generation for SEMAPHORES            */
 106    #define OS_SEM_ACCEPT_EN           0    /*    Include
        code for OSSemAccept()                                   */
 107    #define OS_SEM_DEL_EN              0    /*    Include
        code for OSSemDel()                                      */
 108    #define OS_SEM_QUERY_EN            0    /*    Include
        code for OSSemQuery()                                    */
 109
 110
 111                                            /*
                        --------------------- TASK MANAGEMENT
                        --------------------- */
 112    #define OS_TASK_CHANGE_PRIO_EN     0    /*     Include
        code for OSTaskChangePrio()                              */
 113    #define OS_TASK_CREATE_EN          1    /*     Include
        code for OSTaskCreate()                                  */
 114    #define OS_TASK_CREATE_EXT_EN      0    /*     Include
        code for OSTaskCreateExt()                               */
 115    //change over here in OS_TASK_DEL_EN
 116    #define OS_TASK_DEL_EN             1    /*     Include
        code for OSTaskDel()                                     */
 117    #define OS_TASK_NAME_SIZE         32    /*
        Determine the size of a task
        name                          */
 118    #define OS_TASK_PROFILE_EN         0    /*     Include
        variables in OS_TCB for profiling                        */
 119    #define OS_TASK_QUERY_EN           0    /*     Include
```

Page 4, 1/7/2009 • 5:15:29 PM

Programmer's Notepad - os_cfg.h

```
        code for OSTaskQuery()                                  */
120     #define OS_TASK_SUSPEND_EN          0    /*     Include
        code for OSTaskSuspend() and OSTaskResume()             */
121     #define OS_TASK_SW_HOOK_EN          1    /*     Include
        code for OSTaskSwHook()                                 */
122
123
124                                              /*
        ---------------------- TIME MANAGEMENT
        ---------------------- */
125     #define OS_TIME_DLY_HMSM_EN         1    /*     Include
        code for OSTimeDlyHMSM()                                */
126     #define OS_TIME_DLY_RESUME_EN       0    /*     Include
        code for OSTimeDlyResume()                              */
127     #define OS_TIME_GET_SET_EN          0    /*     Include
        code for OSTimeGet() and OSTimeSet()                    */
128     #define OS_TIME_TICK_HOOK_EN        1    /*     Include
        code for OSTimeTickHook()                               */
129
130
131     typedef INT8U             OS_FLAGS;     /* Data type
        for event flag bits (8, 16 or 32 bits)                  */
132
133     #endif
134
```

Page 5, 1/7/2009 • 5:15:29 PM

NOTE 1: Here in this file os_cfg.h check at the line 27 for the macro CPU_CLOCK_HZ which should be assigned as 16MHZ i.e. the CPU frequency and at line 67 the macro OS_TICKS_PER_SEC which should be assigned as 61.

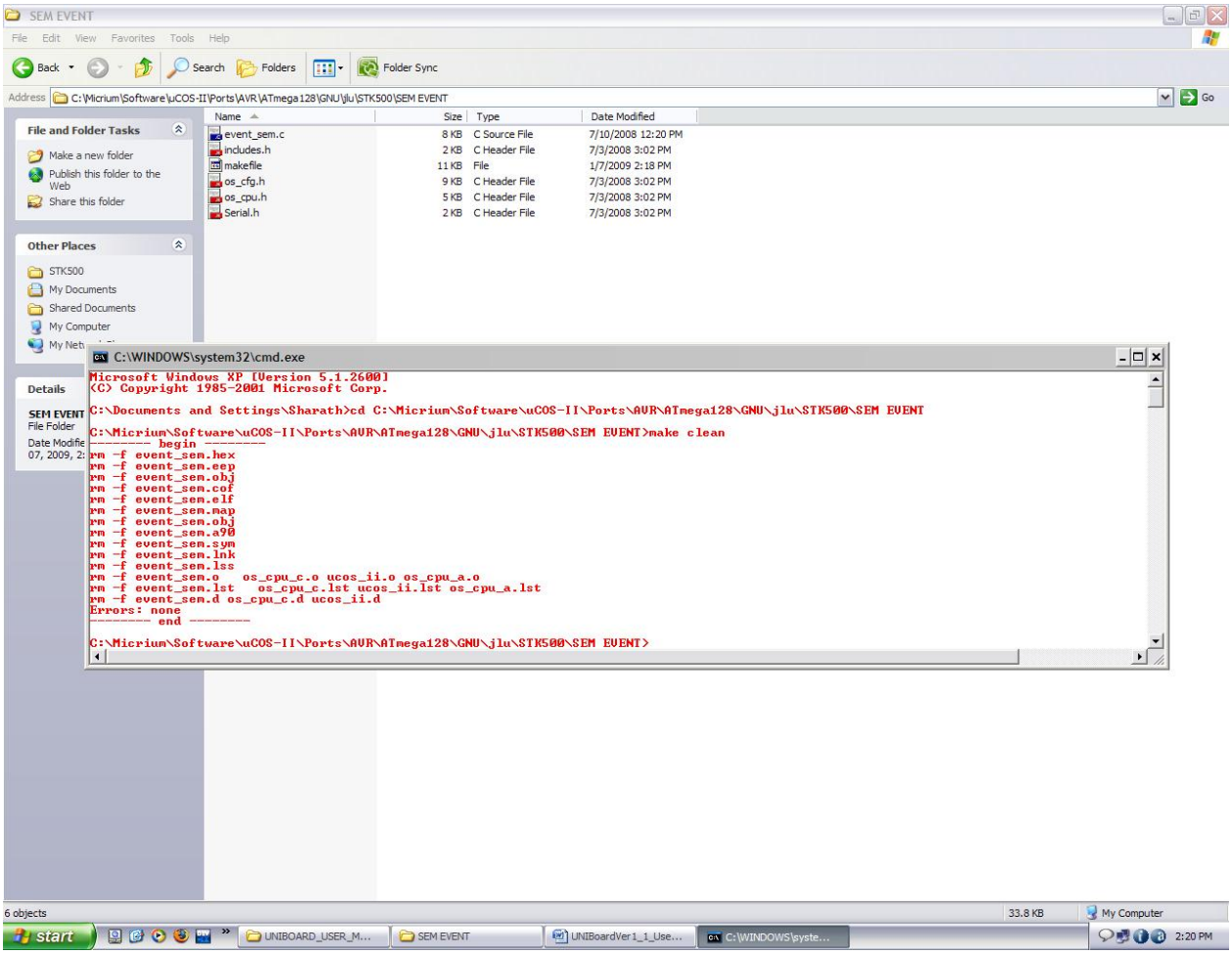
NOTE 2: If you want to use any features/API's of the uC/OS-II enable it i.e. for example in this code we have used semaphore so we need to enable the API in os_cfg.h file as shown at line 105 macro named OS_SEM_EN should be set to 1. If you don't do this then compiler will prompt for undefined reference for that API call.

STEP 7: Go to prompt and type - make clean

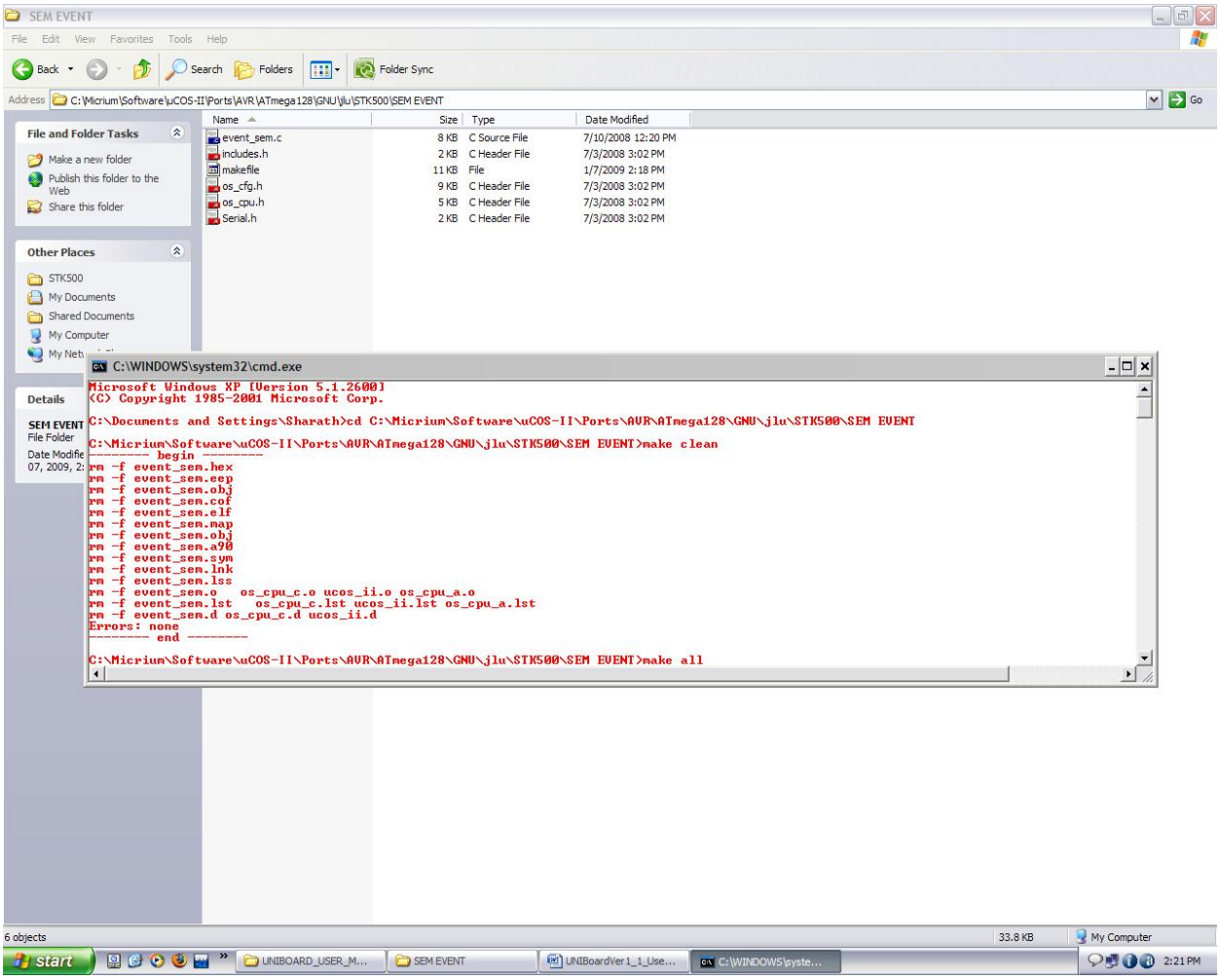
NOTE: If there **is Error on make clean** as shown below

make: * No rule to make target `/home/XYZUSER/ucos-II/software/uCOS-II/Source/ucos_ii.c', needed by `ucos_ii.d'. Stop**

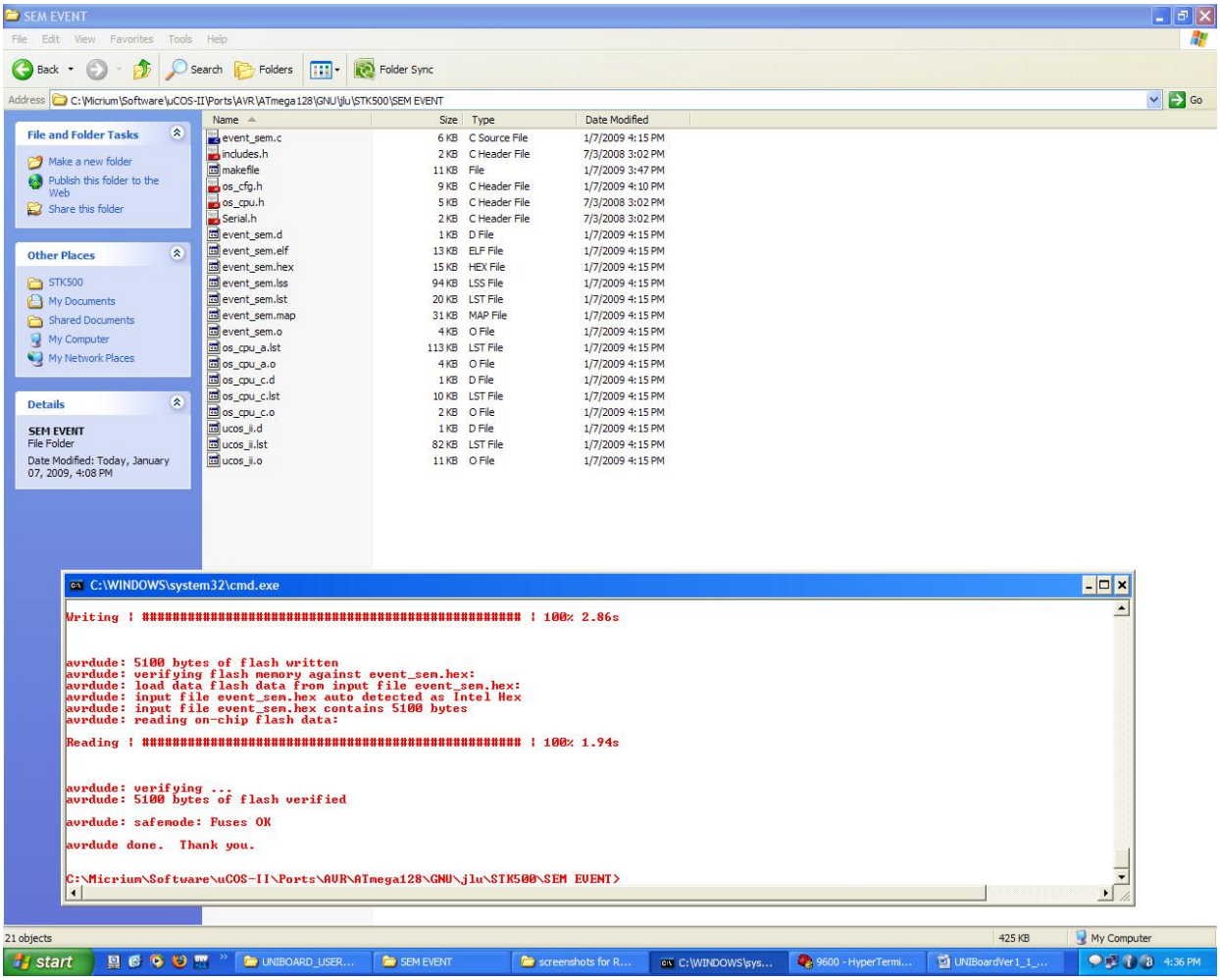
Then you need to remove *.d files from current program directory. Type **rm *.d** and then try to use make clean command again.



STEP 8: Go to prompt and type make all



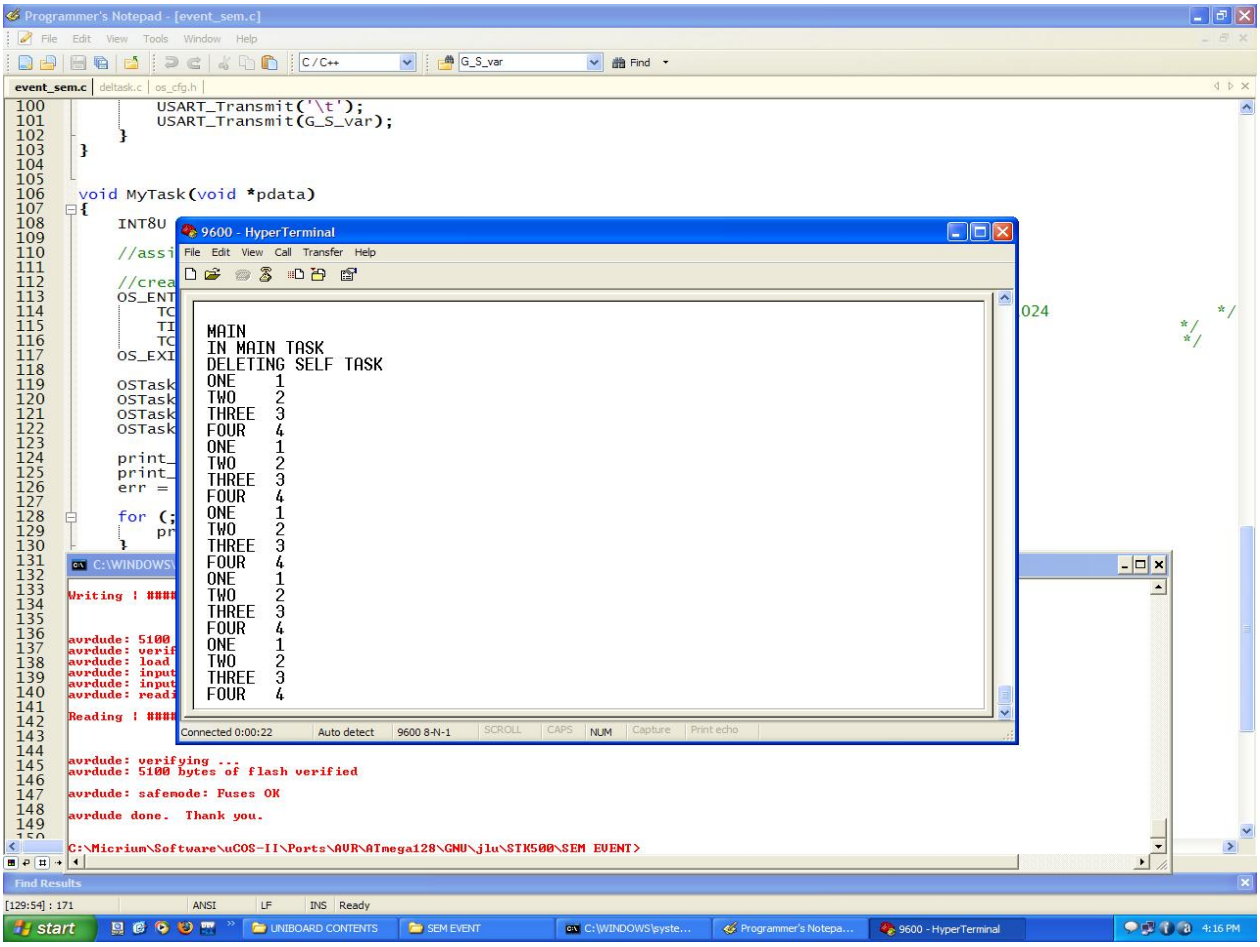
STEP 9: Go to prompt and type make program



STEP 10: Turn OFF the Board's Power Switch

STEP 11: Open the HyperTerminal utility (refer to section Using Hyper terminal of Windows).

STEP 12: Now turn it ON. The output will be displayed as shown below



You can also download the latest uC/OS-II source code from Micrium website link as http://www.micrium.com/products/rtos/ucos-ii_download.html and ports for Atmel's Mega series from http://www.micrium.com/atmel/ATmega.html

Troubleshooting

Make utility

- Make sure that you are in current project working directory.
- Make sure that you have Makefile in your current working directory. If not then rename the Makefile_WINDOWS or Makefile_LINUX to Makefile from uNiBoard Contents.
- Make sure that you have powered on the Board.
- Make sure that program selection switch is pressed.
- Make sure that you do not have multiple files named Makefile in the current working directory.

LEDs

- Make sure that you have configured Port as output.
- Make sure that you have connected the FRC cable correctly to the port where you are planning to interface the LEDs.

UART

- Make sure that you are configuring the correct UART port. Generally for serial communication you would be using UART 1 since DB9 connector is available only on UART 1.
- Make sure that baud rate that you are using is the same at both the ends.
- Make sure that you have not opened multiple Gtkterm application or HyperTerminal.
- Make sure that you have connected Serial Cable correctly and there is no loose contact.

LCD

- Make sure that you have not forgotten to call the Initialization code.
- Make sure that there is no loose contact on the H/W.
- Power up the Board again if some random string comes on LCD.

Joystick

- Make sure that you are using correct ADC channel.
- Make sure that Joystick selection switch is pressed.

LDR

- Make sure that you are using correct ADC channel.
- Make sure that LDR Jumper is placed on the Board and in the correct position.

External Interrupts

- Make sure that you have enabled pull-ups in your code.
- Make sure that you have enabled Global Interrupt.

RTC

- Make sure that you are enabled CH bit in RTC configuration 0x00 RTC's memory area.
- Make sure that you have working backup battery for correct time even if board is power-off.

SPI

- Make sure that your connections are proper; you need to remove the program jumper and use the berg connector to connect the wire to another SPI device.
- Make sure that the GND points of the master and slave have been connected.

Motor drivers

- Make sure that you have connected external power supply.
- Make sure that your ADC chip is there on the Board.
- Make sure that power selection switch is depressed

What you can do with the uNiBoard

This section describes the purpose of every sample Code given in the CD's uNiBoard Contents directory which contains the sample codes for RTOS based and Non-RTOS.

NOTE: The sample code given contains the Makefile. You need to rename the Makefile_LINUX to Makefile for Linux OS and similarly Makefile_WINDOWS to Makefile for Windows OS while using the code.

In **Non-RTOS based sample codes** we will describe in short what the code is meant for and what sort of output it will demonstrate:

1. **PORTS**

   The LEDS are connected to one of the general purpose port (PORTC) available of the ATmega128. These LEDs are connected using the FRC cable. The open FRC connectors are given on the uNiBoard so that their status can be monitored by interfacing them to LEDs using the FRC cable.

2. **Timer0 using Interrupt**

   The Software clock is implemented using hardware timer, Timer0 in Normal mode such that it generated and Interrupt after every 1ms. The Board's Crystal is 16MHZ and PRESCALAR is set to 1024 therefore 16MHZ/1024 = 0.000064 sec (resolution of timer). So for timer to overflow for every 1ms we need to load the count ((256 – 15) =241) as 0.000064 * 15 = 0.96 msec ~ 1msec. The clock is displayed on LCD and UART.

3. **Switches**

   The Switches on the uNiBoard are connected to general purpose port pins of the controller. So you can use this switches for reading the inputs from the user. The SW1 is connected to PORTDS's PIN7 and SW2 is connected to PORTD's PIN6 of the controller.

4. **EEPROM**

   The ATmega128 has an on-chip EEPROM (4KB) so that you can use it to store the non-volatile data. The code contains the some library functions so that you can store the contents in a particular pattern or same data at locations by specifying start address and end address. The output is debugged on serial window. EESAVE fuse bit has to be enabled or else on every programming cycle the EEPROM will also be erased along with flash.

5. **Joystick and VT library**

   The Joystick (analog joystick of PS2 fame) is used as a mouse on the serial window in this sample example. The Joystick X axis is connected to the on-chip ADC channel0 and Y axis to channel1. The ADC is configured such that it gives an interrupt after the conversion is over. Since only one ADC channel conversion is done at a time so we need to reconfigure it for other channel. The ADC conversion value is stored in ADC's Data register.

The VT 100 is video terminal made by DEC. It became a de facto standard used by terminal emulators. Digital's first ANSI-compliant terminal, introduced in August 1978. The VT100 was more of architecture than a simple terminal. There are two display formats: 80 columns by 24 lines and 132 columns by 14 lines. A separate advanced video option was required to display 24 lines in 132-column mode. This was standard on the VT102 and VT131.

We will use the same VT102 commands for serial terminal. The header file contains API's which uses VT102 standard sequence characters for changing the configurations of the serial terminal.

The cursor on the hyper terminal / Gtkterm can be controlled using the joystick and the joystick sensitivity/cursor speed can also be changed by pressing SW4. There are 3 different speeds programmed.

6. LCD

The LCD can be used as an output device and can be placed on the uNiBoard LCD connector. The hardware connections are as follows:

- LCD control lines Enable (PD2)
- R/~W (Hardwired GND)
- RS (PD0)
- LCD data lines (PD4 – PD7). Therefore LCD is initialized using 4 line mode.

The given LCD library contains the API's for printing character, string, values (decimal) on LCD, changing the LCD positions, clearing the LCD screen.

7. UART

There are two UART's on controller. On the uNiBoard UART1 is used to connect to the RS232 port and open UART0 connector given on the board for connecting the devices with TTL compatible device.

The UART library given configures the Baud rate, data bits, and parity and stop bits. Refer to the section "Using hyper terminal for Windows" for Windows OS and "Gtkterm Configurations for setting the Baud rate, Parity, Stop bits" for Linux OS.

8. SPI

There are two programs given for SPI, one for Master and another for Slave. You can communicate between two uNiBoard by connecting wires on SPI lines and making GND common of both the boards. The Output is debugged using the serial terminal whatever character typed on master's serial terminal is transferred to the slave's serial terminal using SPI. Make sure you have connected the SPI lines and the GND points of the master and slave correctly as follows:

MASTER	SLAVE
MISO	MISO
MOSI	MOSI
SCK	SCK
SS	SS
GND	GND

9. I2C (RTC)

The DS1307 is the RTC (Real Time Clock) chip interfaced on the I2C lines of the controller. The API's can be used for reading the date and time from RTC. Before reading you need to call this API Update_RTC_variables() which actually updates local variables that data and time API's returns. You can also set date and time for RTC using Write_RTC() API and date and time values for writing to RTC registers are taken from RTC_def_cfg.h.

The output can be seen on the UART.

10. Buzzer

The Buzzer is connected on PA3. The sample code configures the pin as output and Buzzer can be ON and OFF by setting this PIN High and LOW respectively.

In **RTOS based sample codes** we will describe in short what the code is meant for and what sort of output you should expect to be demonstrated while you are running a particular sample code:

1. Multitasking random

This is a sample program that displays random numbers on the screen. The number is placed on the random positions on the screen. Also displays the CPU utilization and number of tasks running on the Serial terminal.

2. Snake

This is a game for playing snake. There are four tasks used in the game. The task1 prints the score on the screen and also prints the start message on the LCD and UART. Score is updated by the task4 and is sent to task1 for printing. This is done using Mailbox. Task2 is used for generating food on the screen after particular time interval. This interval is set by Task3. This task basically blocks the task2 and signals after a particular interval. This is done by using semaphores as event flags. Task4 is used printing the snake and also updates the score if the snake eats the food and sends the score to task 1 for printing on serial.

The score is printed by this task on LCD whenever snake eats food. Also every time food is generated LEDs would blink. Every time the SNAKE eats food the buzzer will give a beep sound.

3. **Semaphores**

   This is a sample code which displays the shared resource as a global variable which is shared among the various tasks. There are four tasks which write to the same global variable and one print task which wait for the signal and once get the signal prints the value of the global variable on UART and again wait for the event to occur.

4. **Mutex**

   This is a simple example which shows how the priority inversion problem can be reduced using Mutex. In this example there are 3 tasks which illustrate how a low priority task owns a resource and uses a Mutex to inherit the priority of the waiting task of higher priority and this can be seen on serial terminal.

5. **Mail Box**

   In this sample code the message is been updated by many tasks using MailBox while print task waits till the Message has been written and after that Message is printed by the print task. So this shows how the message is not allowed to be modified by the other task once task owns the resource (Message). The higher priority task is the print task which will print the Message when one of the task releases the resource (Message mailbox).

6. **Message Queues**

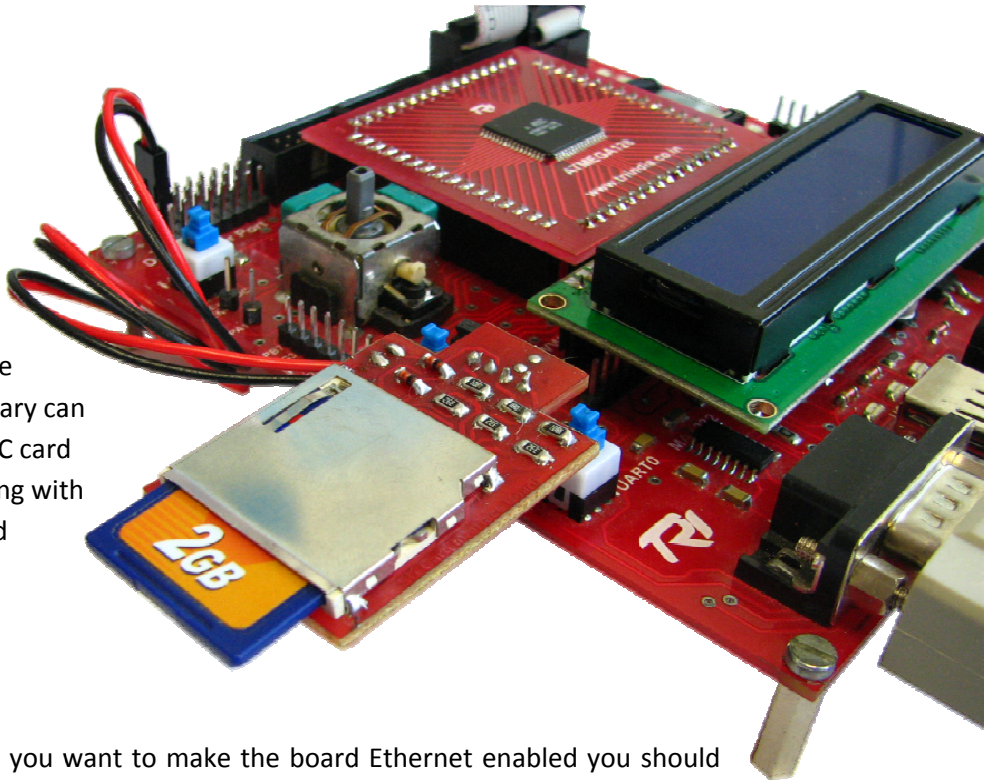
   This sample code illustrates the use of Message queue by two tasks posting the message and one task which print this message on UART.

7. **ISR and Delete Task**

   This is sample program in which the semaphores are used to update the shared variable among the various tasks. Main task (higher priority) waiting for the event to occur using semaphore as en event flag. Whenever the interrupt occurs the scheduler executes its corresponding ISR. The ISR signals the main task by posting the semaphore and signaling an event, then the main task deletes the task one by one and also deletes the task itself.

Add-ons

What we have discussed till now are some completely standalone activities. We assume that you wont just limit yourselves to these activities and explore the board's power further using accessories like SD/MMC card interface (SPI based) or Ethernet interface (SPI based). You can interface these add-ons that will be available on our website http://thinklabs.in/shop in a short while.



SD/MMC card Interface

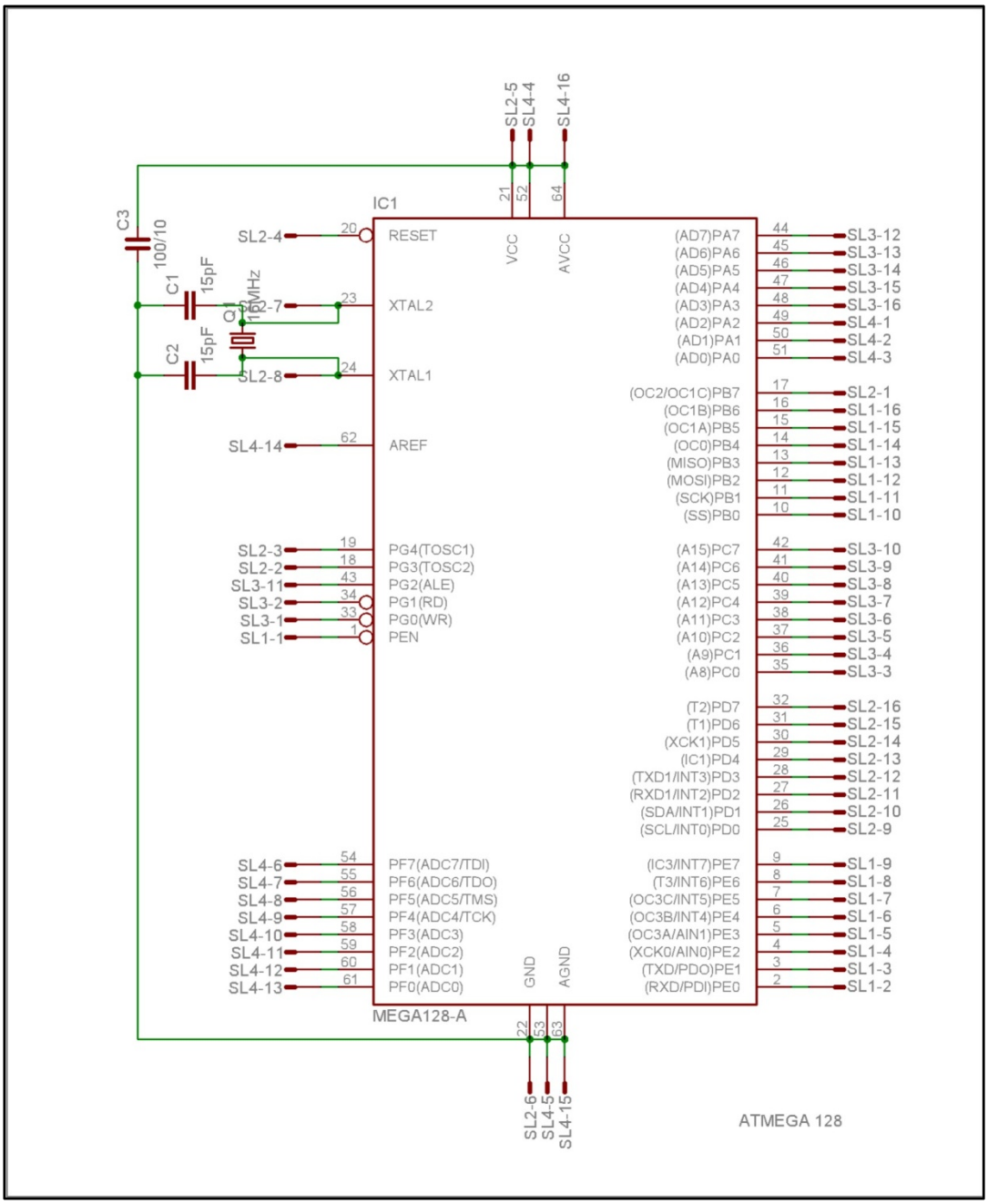
Activities related to these products like reading a FAT partitioned/formatted SD card, file handling using the Embedded File System library can be done using the SD/MMC card interface which comes along with a detailed user manual and application notes.

Ethernet Interface

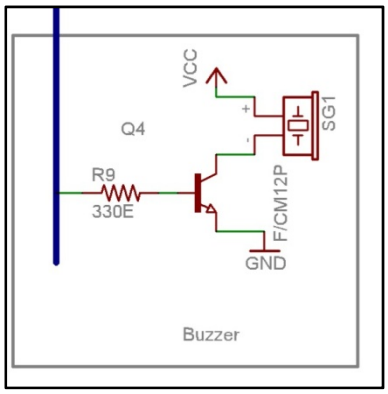
Moreover, if you feel that you want to make the board Ethernet enabled you should definitely check out the Ethernet controller board that we are planning to come up with. Activities related to using this board with the uNiBoard will be uploaded shortly on our website. Using the Ethernet controller you can also port the open source uIP stack by Adam Dunkels and enhance your embedded projects by making them Ethernet enabled. You can also build stuff like an embedded web server or control your board/devices connected to the board over the Ethernet.

uNiBoard v1.1 Schematic

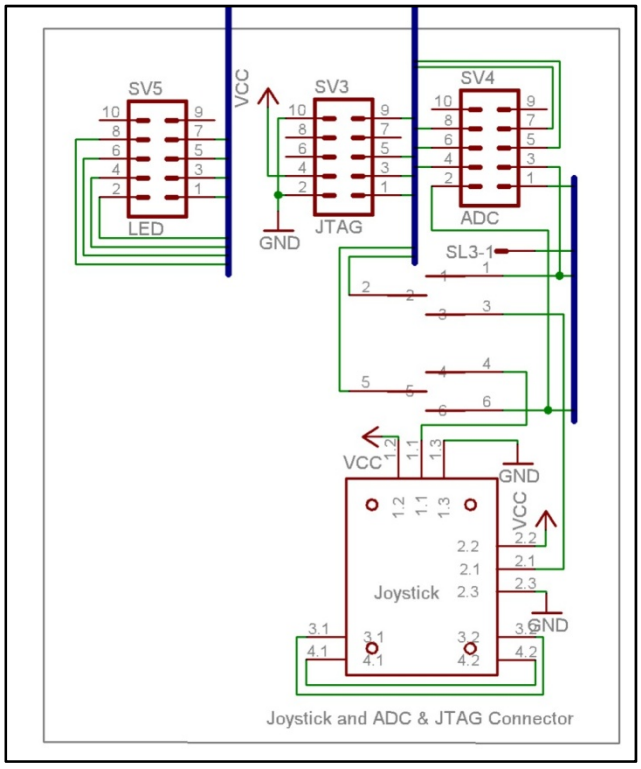
Controller section: ATmega128 pin connections



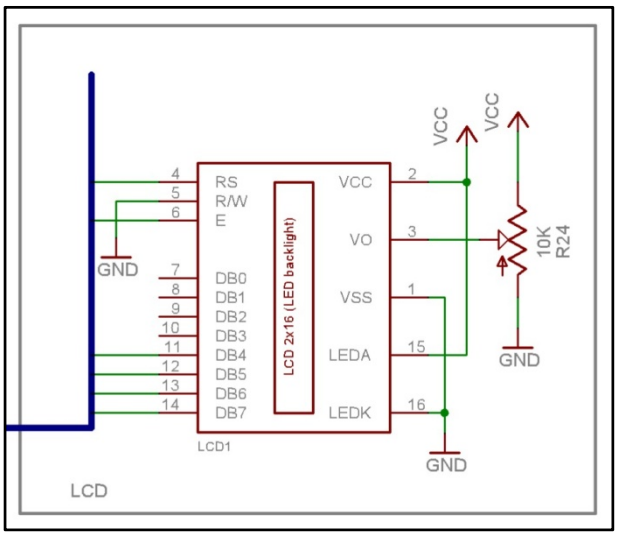
Buzzer section:



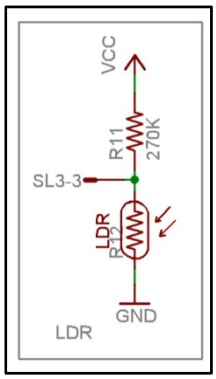
Joystick and connectors section:



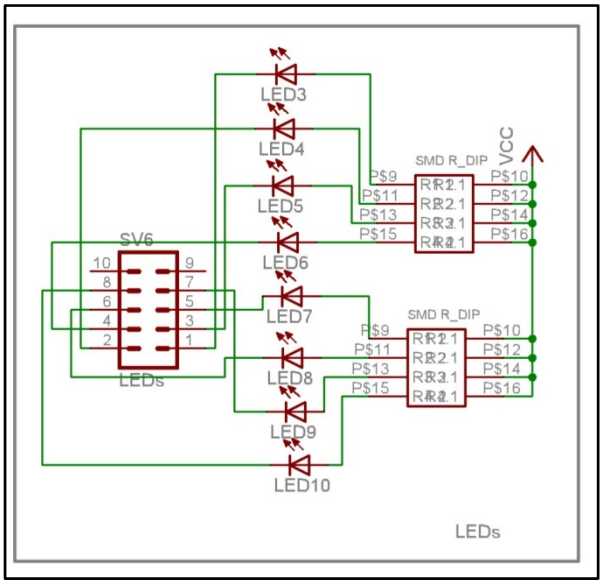
LCD section:



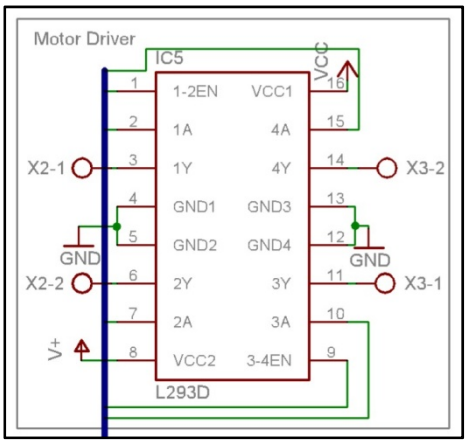
LDR section:



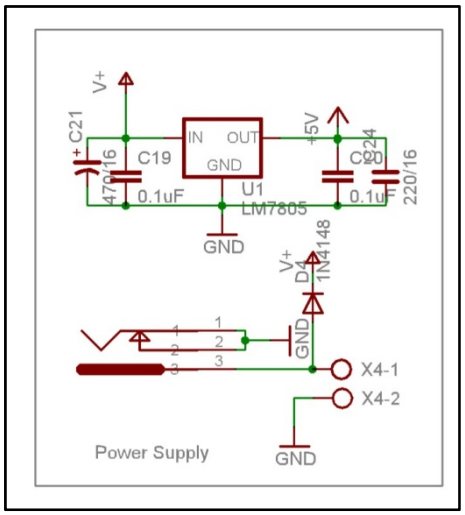
LEDs section:



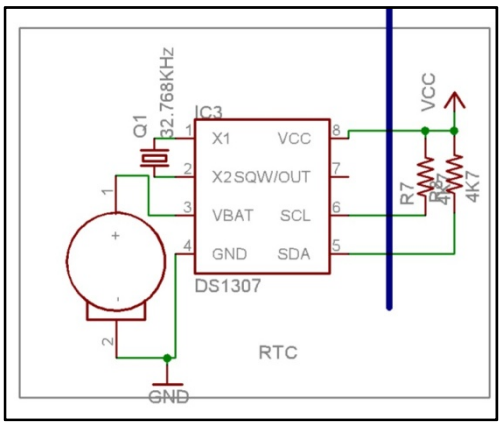
Motor Driver section:



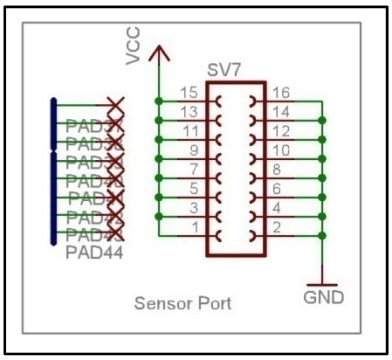
Power supply section:



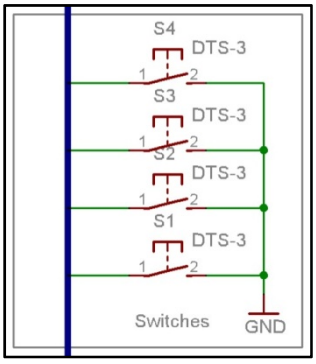
RTC section:



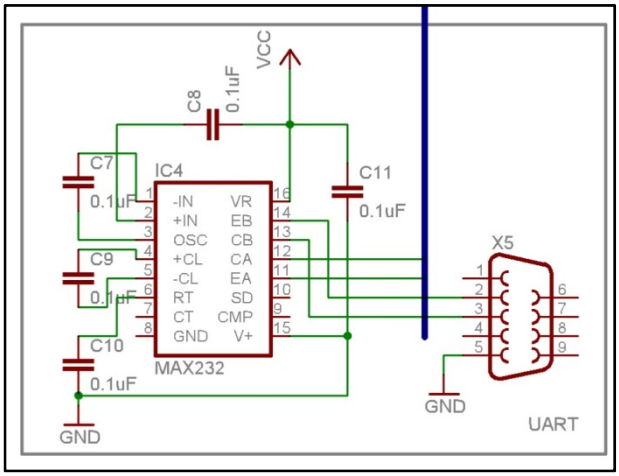
Sensor port section:



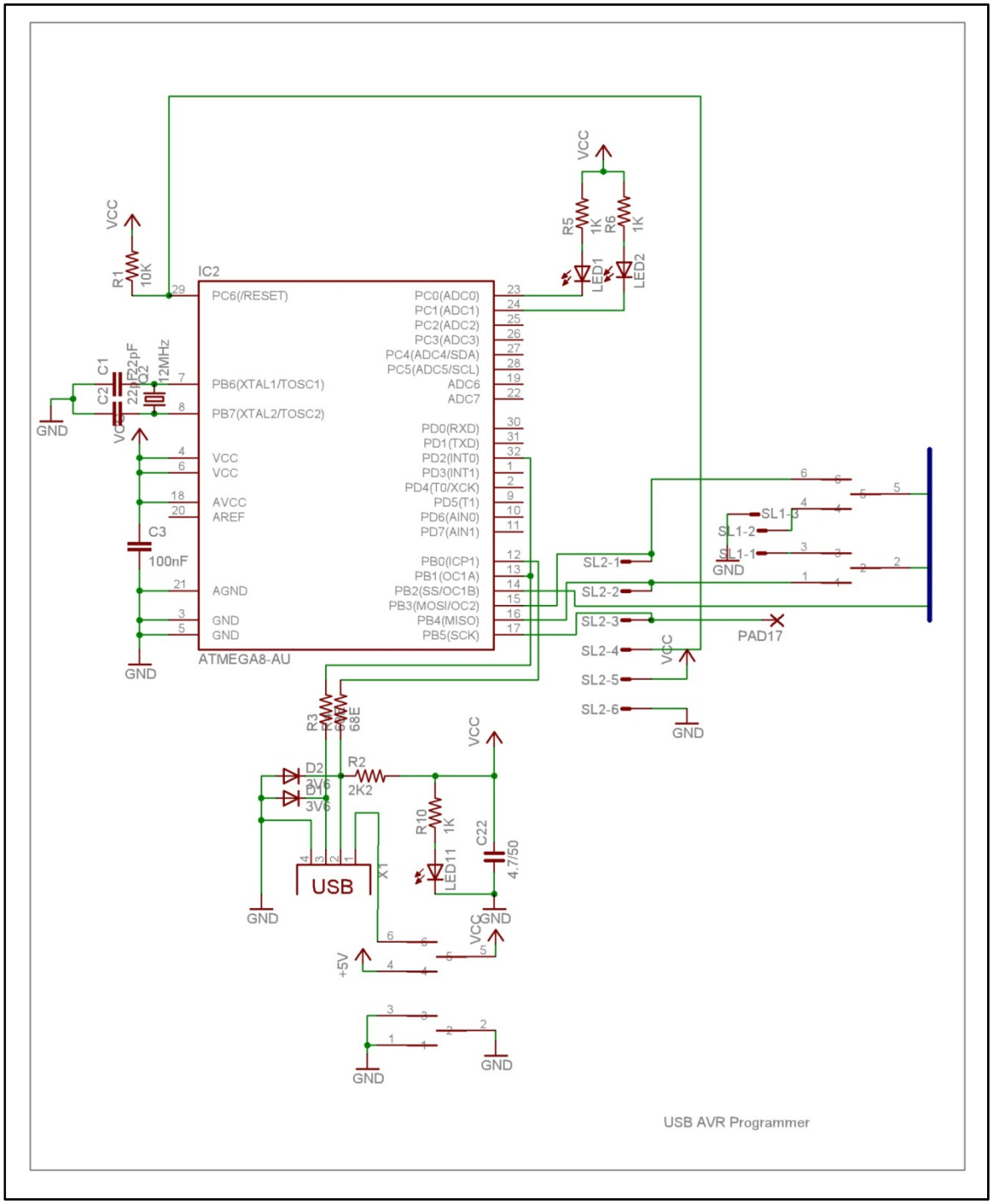
Switches section:



UART section:



USB programmer section (Courtesy: http://www.fischl.de/usbasp/):



Licensing terms

The uNiBoard is licensed by the Creative Commons license (for more details refer License.txt in CD content), which has the philosophy of **"Share, Remix, Reuse - Legally".** It means that anyone is allowed to produce copies of the board, to redesign it, or even to sell boards that copy the design. You don't need to pay a license fee to the **ThinkLABS team** or even ask permission. However, if you republish the reference design, you have to credit the original group. And if you tweak or change the board, your new design must use the same or a similar Creative Commons license to ensure that new versions of the board will be equally free and open.

Bill of Materials (BOM):

uNiBoard ver. 1.1 Bill of Materials		
Sr. no.	**Components**	**Quantity**
1	**Micro-controllers**	
	Atmega128 (TQFP)	1
	Atmega8 (TQFP)	1
2	**Other chips**	
	L293D Motor driver	1
	7805 voltage regulator	1
	MAX232 level translator	1
	DS1307 RTC	1
3	**Battery back-up**	
	Battery 3V (button cell)	1
4	**Crystal**	
	16 MHz (for Atmega128)	1
	12 MHz (for usbasp)	1
	32.768 KHz (for RTC)	
5	**Sensors**	
	LDR	1
	PS2 Analog Joystick	1
6	**Sockets**	
	16 pin IC base	1
	Battery cell holder	1
7	**LCD**	
	16x2 Alphanumeric LCD	1
8	**Cables**	
	Serial cable (DB9F to DB9M)	1
	USB Cable	1
	FRC Cables	2
9	**LED**	
	Red LED SMD	11
10	**Buzzer**	
	5V Buzzer	1
11	**Connectors**	
	DC Jack (PCB mountable)	1
	PTR connector	3
	DB9 Right angle female	1
	USB connector	1
	10 pin FRC connector	4
	Berg header (male)	1 set
	Berg header (female)	1 set

12	**Diodes**	
	3.6V Zener	2
	4148 Diode	1
13	**Resistors**	
	1K	3
	10K	2
	270K	2
	330 Ohms	2
	2K2	2
	4K7	2
	68 Ohm	2
	1K SMD SIP	2
	10K POT	1
14	**Capacitors**	
	100uF/10V	1
	4.7uF/50V	1
	470uF/16V	1
	220uF/16V	1
	15pF	2
	22pF	2
	0.1uF	15
	0.22pF	2
15	**Transistors**	
	2N2222	1
16	**Switches**	
	Power switches (Push to ON)	3
	SMD Micro key	5
	Two pin jumpers	2
17	**Metal spacers (3x15)**	4

Appendix A – References

Websites:

www.atmel.com

www.avrfreaks.net

www.fishl.de/usbasp/

www.maxim-ic.com

http://en.wikibooks.org/wiki/Embedded_Systems/Atmel_AVR

Books:

1. MicroC/OS-II by JEAN J. LABROSSE
2. Programming & Customizing The AVR Microcontroller By Dhananjay V. Gadre
3. Embedded C Programming and the Atmel AVR By Barnett and Cox
4. AVR RISC Microcontroller Handbook By Atmel

Appendix B - Services and Support

Our website support:

For any technical queries related to uNiBoard:

http://www.thinklabs.in/forums/

For articles, activities and updates related to uNiBoard:

http://www.thinklabs.in/resources/

For purchasing uNiBoard accessories or other related components:

http://www.thinklabs.in/shop/