1 Introduction

Automation in industries is the need of the hour. Everything is done electrically and not manually, right from a really unnoticed process like moving the conveyors to something as power intensive as running pumps. Industrial automation is primarily necessary for high end production and also product quality controls. Mainly, in industries, such automations are carried out by intelligent systems called Programmable Logic Controllers with the help of a host computer. A PLC, is in an actual sense, an electrical version of a microcontroller unit. It's more generic and also controls high power tools which generally work at really high voltages, unlike the microcontroller that merely works at considerably small voltage values. Basically, it intelligently controls whatever is programmed onto it by the host computer. This controller, being so versatile, is highly expensive, due to it's high rising demand and also it's operability credits.

A PLC generally has a set of input and output ports to with the external peripherals are connected, which are to be electrically controlled by the PLC. Looking into the flow of working of the PLC, the processor tries to make decisions in accordance to the program, generally written by the user using some logic used to program these units. Some of these logics include `Function Block Diagram, Ladder Diagram, Structured Text, Sequential Function Chart` and `Instruction List`.

The objective is to create an Open Source PLC, which has the capabilities of the industrial PLC, but the catch here is, it's much cheaper and simpler to work with. It incorporates a development board, that works on AVR's `ATmega16A` IC, giving upto 40 pins, most of which can be used and programmed as the Input-Output pins as it's done on a PLC. Moreover, the logic used here is `Ladder Logic` which is one of the easiest and also, robust in terms of its efficiency. Main point here is, this device is a standalone device, which can work without the host system being there, just being powered by a source. `HEX` files to be generated according to the controller are generated by a software called `LDMicro`, freely available for `Windows` and `Linux` users alike. The programming on the board is done via a `RS-232` cable by serial communication. All the processor wants now is a code to be dumped on it to start automating your small needs.

This prototype will be helpful for students to get the feel of working with the PLCs, to control small automated systems. It'll also be easy as there's no coding involved and it's using a GUI that is user friendly and also efficient. Along with the development board, which can be used as microcontroller also, there are various hardware modules to start off with and later the students can cone up with their own hardware setups and test their modules, with the board. It's really versatile as it can suit your microcontroller needs and also work as a PLC. All you need is the HEX code for your desired usage with hardware.

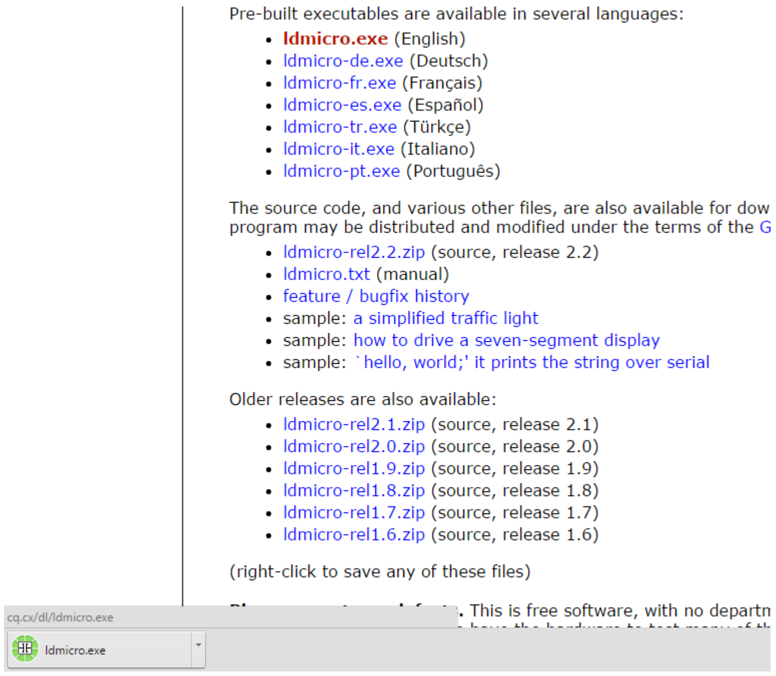
The manual covers the basics of using the associated software and also explains the interfacing with the hardware. Certain simple experiments are given so as to get accustomed to the software usage. Later, some modules that can be made using the various sensors discussed in the preceding sections are covered. Then the imagination of the student and the need can drive the making of various other modules according to the students need.

2 Setting up the LD Micro executable

Many softwares support PLC simulation in various logics.

The software that has been used in the subsequent sections is `LDMicro` that incorporates `Ladder Logic`, supporting many microcontrollers, along with the AVR's `ATmega16A`. This software helps produce the HEX files according to the microcontroller and the pin selected which can be directly dumped onto the core.

To download the software on your `Windows` system, go to `http://cq.cx/ladder.pl`. The download is a direct executable, there's no need to install it.



Pre-built executables are available in several languages:

- **ldmicro.exe** (English)
- ldmicro-de.exe (Deutsch)
- ldmicro-fr.exe (Français)
- ldmicro-es.exe (Español)
- ldmicro-tr.exe (Türkçe)
- ldmicro-it.exe (Italiano)
- ldmicro-pt.exe (Português)

The source code, and various other files, are also available for dow
program may be distributed and modified under the terms of the G

- ldmicro-rel2.2.zip (source, release 2.2)
- ldmicro.txt (manual)
- feature / bugfix history
- sample: a simplified traffic light
- sample: how to drive a seven-segment display
- sample: `hello, world;' it prints the string over serial

Older releases are also available:

- ldmicro-rel2.1.zip (source, release 2.1)
- ldmicro-rel2.0.zip (source, release 2.0)
- ldmicro-rel1.9.zip (source, release 1.9)
- ldmicro-rel1.8.zip (source, release 1.8)
- ldmicro-rel1.7.zip (source, release 1.7)
- ldmicro-rel1.6.zip (source, release 1.6)

(right-click to save any of these files)

. This is free software, with no departn

cq.cx/dl/ldmicro.exe

ldmicro.exe

Figure 1: Downloading LD Micro

3 Getting started with LDMicro

3.1 Understanding the instructions

A PLC is largely programmed using `Ladder Logic`, which is used in this software. It allows us to select the microcontroller that we wish to program via its HEX codes generated, `ATmega16A` in this case. The naming convention is very intuitive and is easy to get accustomed to. Some of the naming conventions followed are:

1. `Yvar` implies the component is connected to an input pin on the microcontroller, something like a push button that is user dependent. This generally works for the digital inputs, `HIGH(+5V) or LOW(GND)`.

2. `Xvar` implies the component is connected to an output pin on the microcontroller. This can be something like an LED or a Buzzer that is used to show the outputs. It's generally used to display digital outputs, `HIGH(+5V) or LOW(GND)`.

3. `Tvar` implies, a timer. It can be a turn on, turn off or a retentive timer, just like the ones used in the actual PLCs.

4. `Cvar` implies a counter. The arguments in this are simple logical operands to decide the upper bound upto which the counter shall work. There are circular counter too, which count circularly, without any bound.

5. `Avar` implies the values read from the Analog pins of the controller. This can be used to take the intermediate readings say from components like Potentiometers, IR sensors etc.

Things to remember are :

- Variable names can incorporate alphabets, numbers and underscores. It doesn't support the special characters.

- Do not start the name of the variable with a number.

- The variable names are case sensitive, the variable names `Relay1` and `relay1` signify two different variables.
- The instructions such as the arithmetic ones can manipulate the variables associated with the timers, counters or input, output pins.
- The variables are 16 bit signed decimals, so the variables can also be containing values that are negative pertaining to that range.
- As counters, timers in physical sense are internal in the microcontroller, we can only assign the pins to the `Xname, Yname and Aname` objects and not others.

Now that the basic ideas are clear, the use of certain instructions can be discussed.

We shall understand normally open and normally closed connections. In normally open case, the contacts are open generally, which means, giving a high input closes the contact, hereby passing the signal over the rung. Whereas, a normally closed connection would imply a high signal to the connection would break the circuit hence giving low as the output.

1. **Contacts** are something like a prototype of a switch, which implies, if the signal going into the contact is true, only then it'll be reflected as output, if false, then output is false for normally open case. These can be used as internal relays too.
2. **Coils** are basically corresponding to the output devices. They can set the output true, if the signal going into them is true for the normally open case, else it's the other way. They can be used with internal relays too. There is an option of Set only (Reset-only) which are set(reset) when the input goes from `Low to High`, and retain their states. Hence they are used with Reset-Only(Set-Only) coils to change the states as and when wanted.
3. **Internal Relays** are the ones that are never assigned pin numbers. Basically, they are the ones that are used for counters or triggers, which are not given inputs or outputs, hence no pin on the controller.
4. **Turn on Delay** simply means delaying turning on of any coil. This mean the sensor, if gives a high at the input, it delays the turning on of the next part of the rung by those many units of time

5. **Turn off Delay** literally means turning off of any coil with some delay. This mean the sensor, if gives a false or a low at the input, it delays the turning off of the next part of the rung by those many units of time

6. **Retentive timer** is used to keep track of the how much time the module under consideration has been true, it cumulates the total time, if the input has been on for atleast that much time, the output of the timer is true. It will always reamin true after this, hence it should be reset by using **Reset** instruction manually

7. **Counter** is used to count upto the given value threshold.The count is changed as the input to the counter is made high. This is used as an up counter, down counter or a circular counter. The variables can be manipulated and be suited for the application

8. **Mov** is used to move any value, be it character or numeric into the variable named under destination.

9. **Arithmetic operands** are used to manipulate the variables values to suit the logic, they cn be used on any kind of variables like the counters variables.

   These are the basic ones that are used. A detailed help for the more instructions and general guidelines is provided on the `Manual` of the software under the `Help` menu.

3.1.1 Configuration of ATmega16A

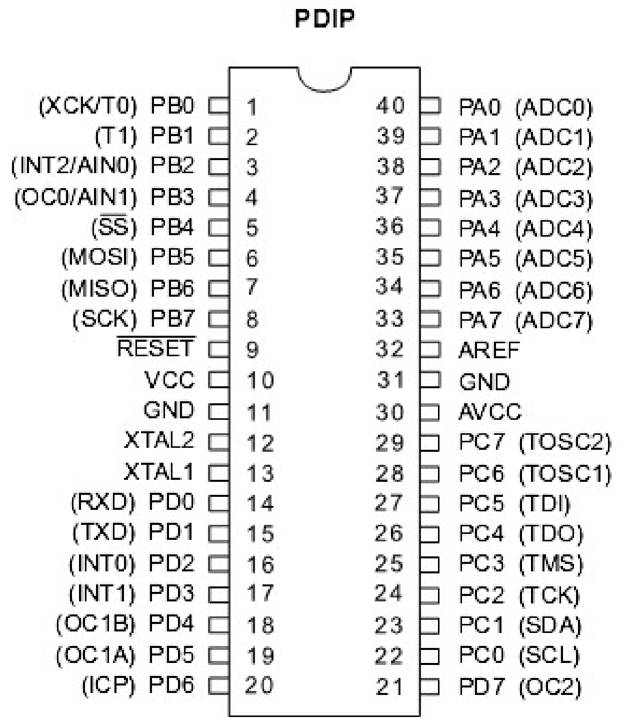


Figure 2: ATmega16A Pin Configuration

3.2 Creating ladders using LD Micro

LD Micro's simple user interface is really easy to use and is self explanatory.

To get started working with LD Micro, just double click the LD Micro executable present in your system. Once opened, you shall get such a GUI.

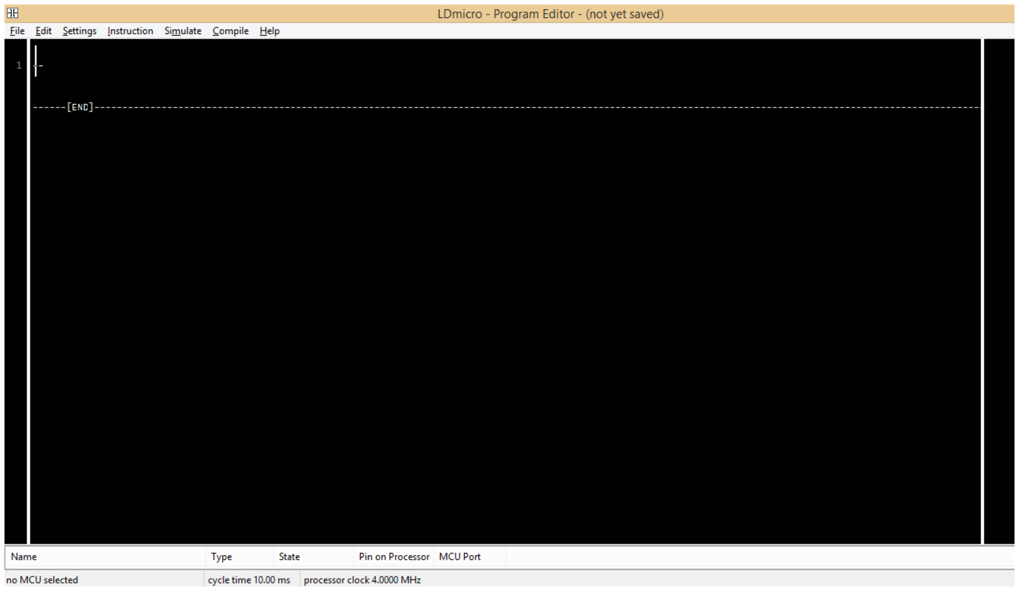
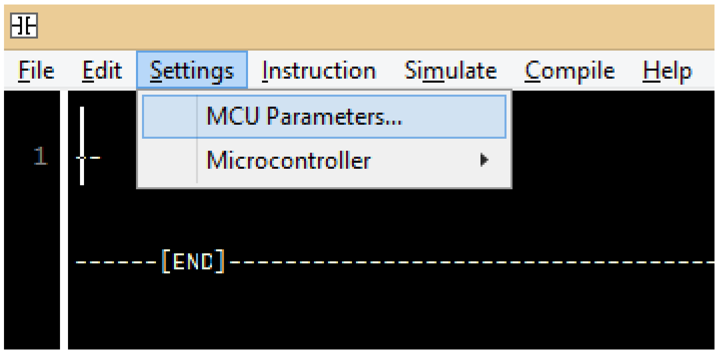

LDmicro - Program Editor - (not yet saved)
File Edit Settings Instruction Simulate Compile Help
[END]
Name Type State Pin on Processor MCU Port
no MCU selected
cycle time 10.00 ms
processor clock 4.0000 MHz

Figure 3: Startup screen of LD Micro

Before starting to make the ladder, first make the necessary changes in the default settings, so as to generate the HEX code for your microcontroller.

Under `Settings` tab, select the `MCU Parameters` Option



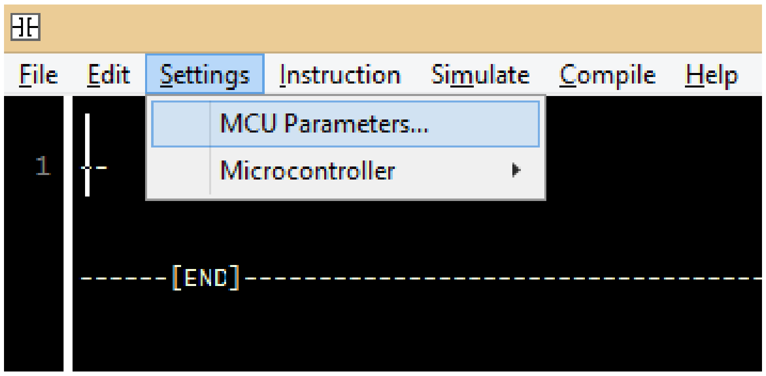


Figure 4: Changing the controller parameters

Then after selecting, change the clock frequency to 16MHz and if using UART change the baud rate to 9600.

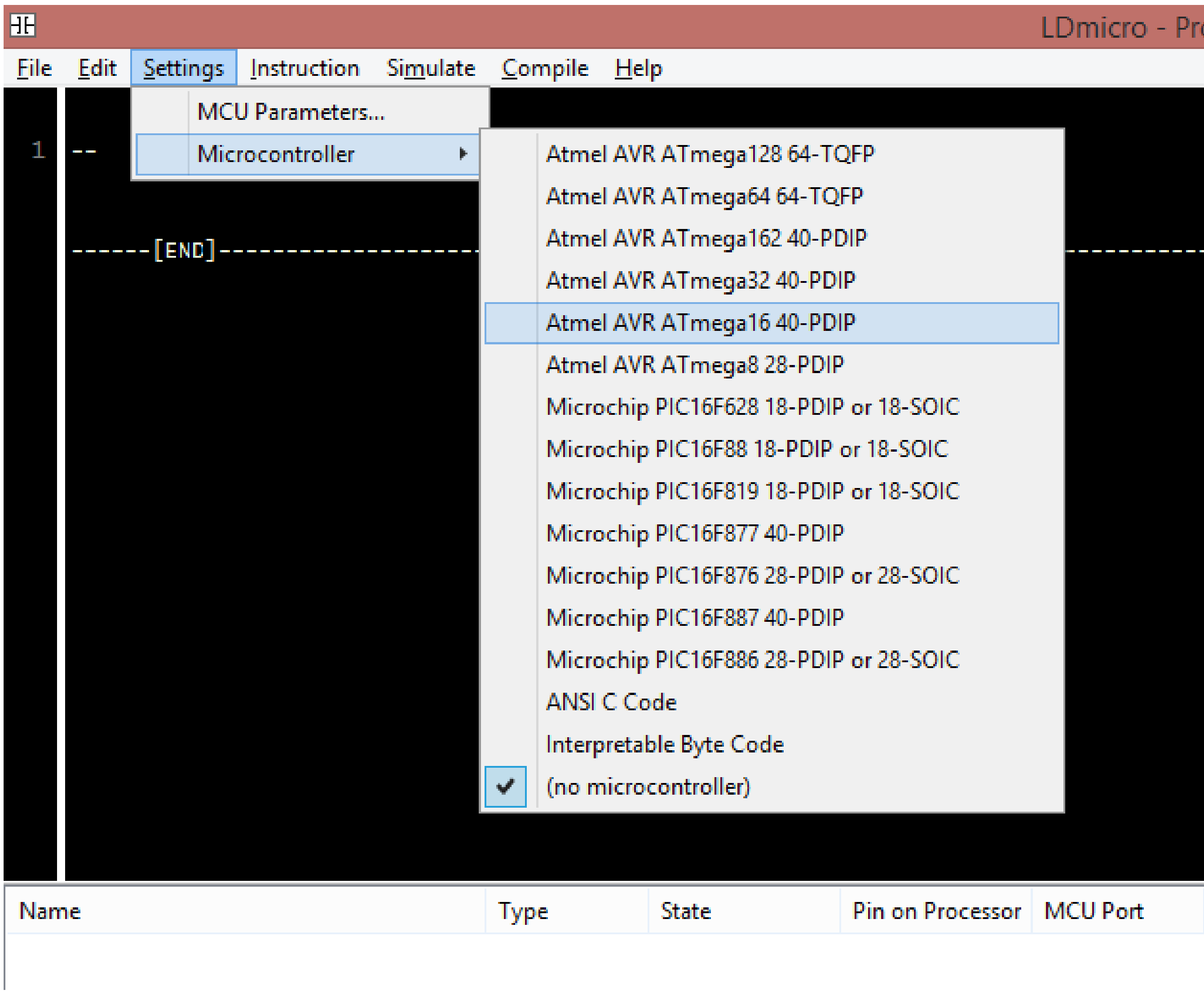

LDmicro - Pr
File
Edit
Settings
Instruction
Simulate
Compile
Help
MCU Parameters...
Microcontroller
Atmel AVR ATmega128 64-TQFP
Atmel AVR ATmega64 64-TQFP
Atmel AVR ATmega162 40-PDIP
Atmel AVR ATmega32 40-PDIP
Atmel AVR ATmega16 40-PDIP
Atmel AVR ATmega8 28-PDIP
Microchip PIC16F628 18-PDIP or 18-SOIC
Microchip PIC16F88 18-PDIP or 18-SOIC
Microchip PIC16F819 18-PDIP or 18-SOIC
Microchip PIC16F877 40-PDIP
Microchip PIC16F876 28-PDIP or 28-SOIC
Microchip PIC16F887 40-PDIP
Microchip PIC16F886 28-PDIP or 28-SOIC
ANSI C Code
Interpretable Byte Code
✓ (no microcontroller)
[END]
Name
Type
State
Pin on Processor
MCU Port

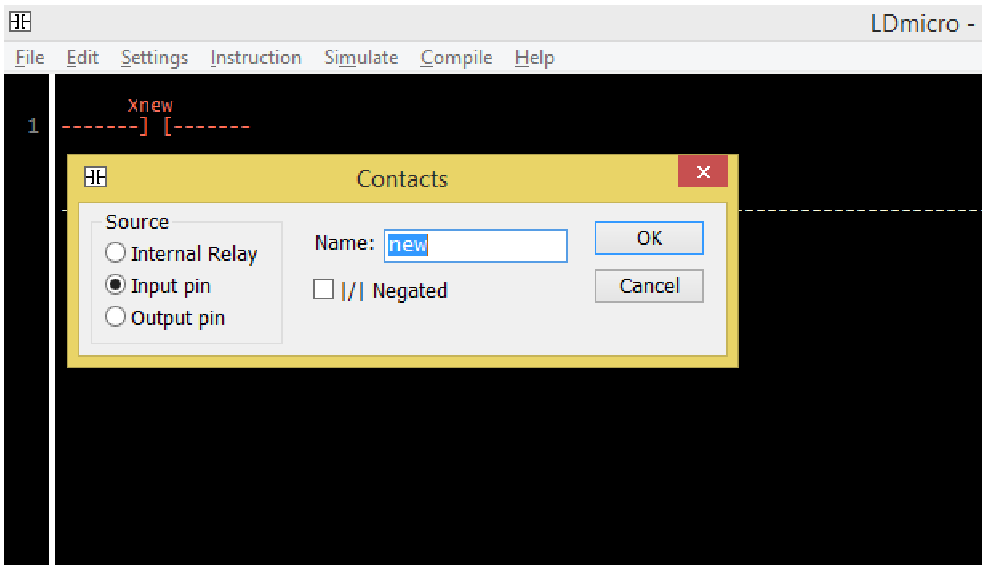
Figure 5: Selecting the appropriate microcontroller

Now that the configurations are set, the next step can be followed to design the ladder with the appropriate logic.

It's really easy to build up the Ladder, just a few clicks and you're good to go.

Start off with having a look at the `Instruction` tab. It has a lot of instructions that are in accordance with the Ladder logic followed for the PLCs and work in the same way and are similar to the ones described above under the 'Understanding the Instructions' section.

After selecting the appropriate instruction, just double click the instruction to edit the name of the variable and other parameters related to the particular instruction.



```
       Xnew
1 -------] [-------
```

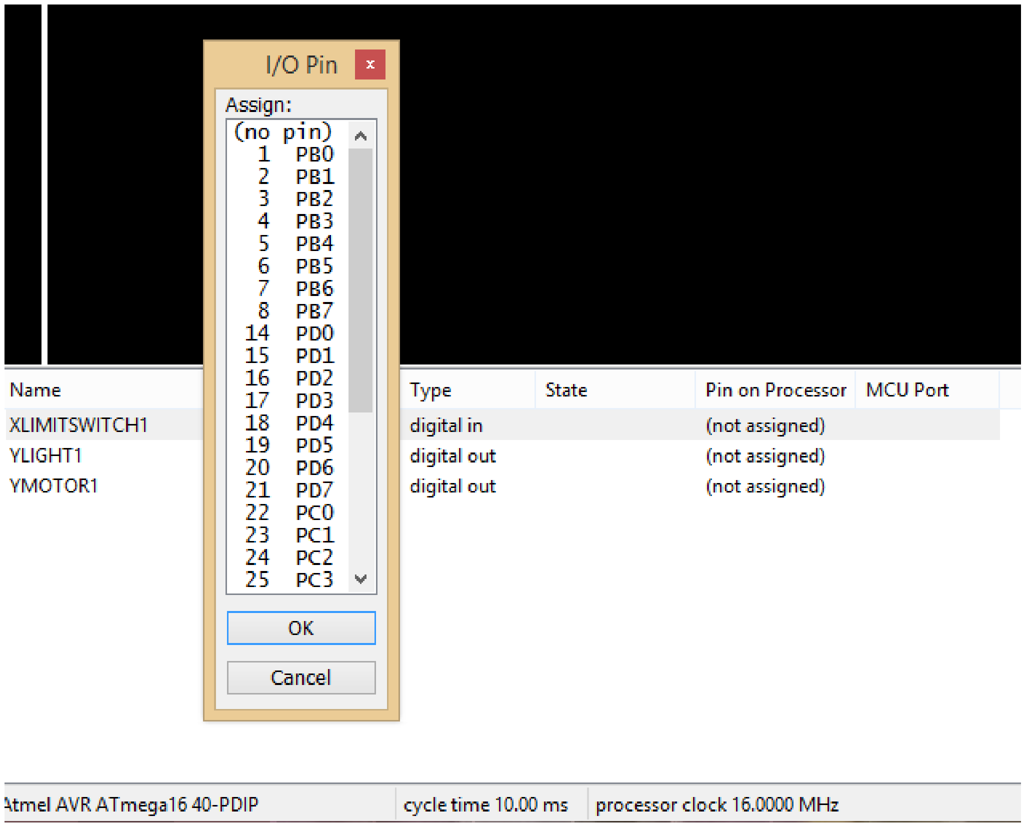
Source
- ( ) Internal Relay
- (●) Input pin
- ( ) Output pin

Name: new

☐ |/| Negated

Figure 6: Editing the parameters of an instruction

After making the appropriate ladder for the application, the digital input and outputs should be given an appropriate pin number of the controller and the connections should be done accordingly. This can be done by double clicking the appropriate pin to be configured in the consolidated table just below the ladder made.

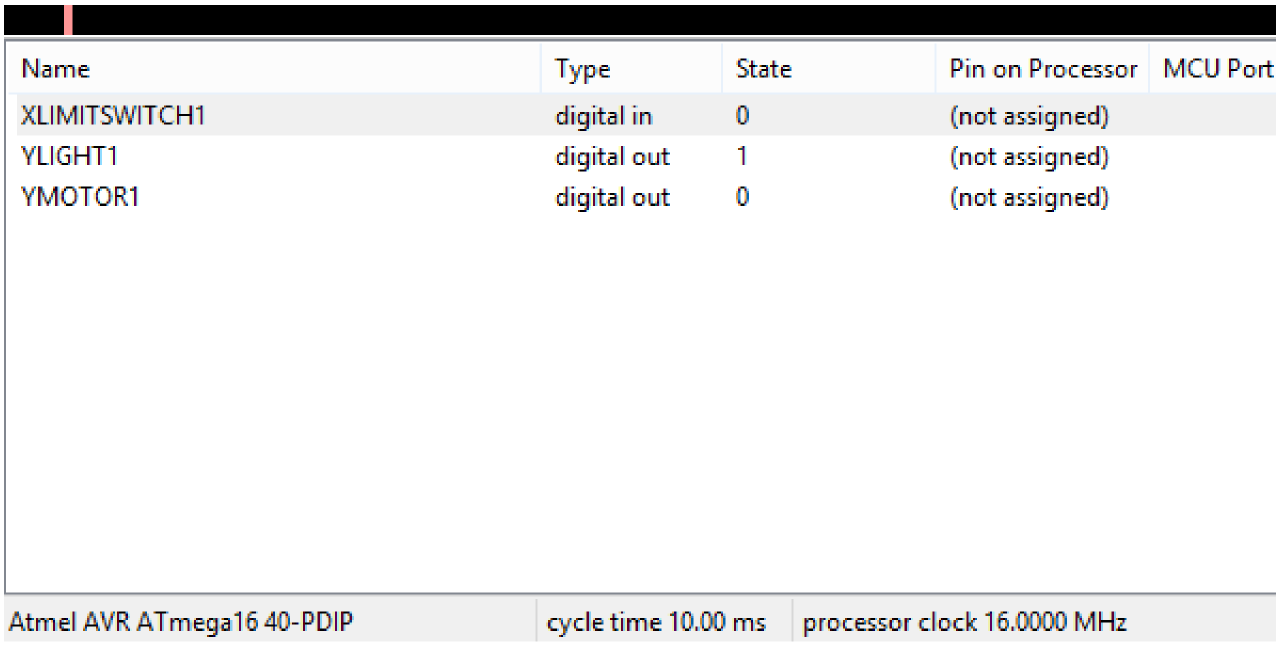


Name	Type	State	Pin on Processor	MCU Port
XLIMITSWITCH1	digital in		(not assigned)	
YLIGHT1	digital out		(not assigned)	
YMOTOR1	digital out		(not assigned)	

Atmel AVR ATmega16 40-PDIP | cycle time 10.00 ms | processor clock 16.0000 MHz

Figure 7: Assigning the appropriate pin number

Now that the logic is set, the ladder can be simulated to verify it's working. This is done by clicking the `Simulate` tab and `Simulation Mode` option under it. This is then followed by a similar screen where `Blue` indicates `LOW` and `Pink` indicates `HIGH`. To start the simulation, under `Simulate` tab click `Start Real-Time Simulation` option. After this, to change the state, just double click the instruction. The appropriate changes and values pertaining to different variables can be seen in the consolidated instruction table under the `State` column. To come out of simulation mode, click the same `Simulation Mode` option.



Name	Type	State	Pin on Processor	MCU Port
XLIMITSWITCH1	digital in	0	(not assigned)	
YLIGHT1	digital out	1	(not assigned)	
YMOTOR1	digital out	0	(not assigned)	

Atmel AVR ATmega16 40-PDIP | cycle time 10.00 ms | processor clock 16.0000 MHz

Figure 8: Tracking the pin states from the table

After the ladder along with assigning the appropriate pin numbers, the HEX code can be generated. This can be done by first saving the ladder under the `File` tab with the `Save As` option. Save the file in the appropriate destination folder with a suitable name. Remember the final destination of the same for future. Now, to generate the HEX file, select the `Compile` tab and then the option `Compile As`. Then save this text file in the desired destination folder. The location shall be noted as it'd be needed to burn the code onto the controller using the command line arguments later.

4 Programming the IC

4.1 Installing WINAVR

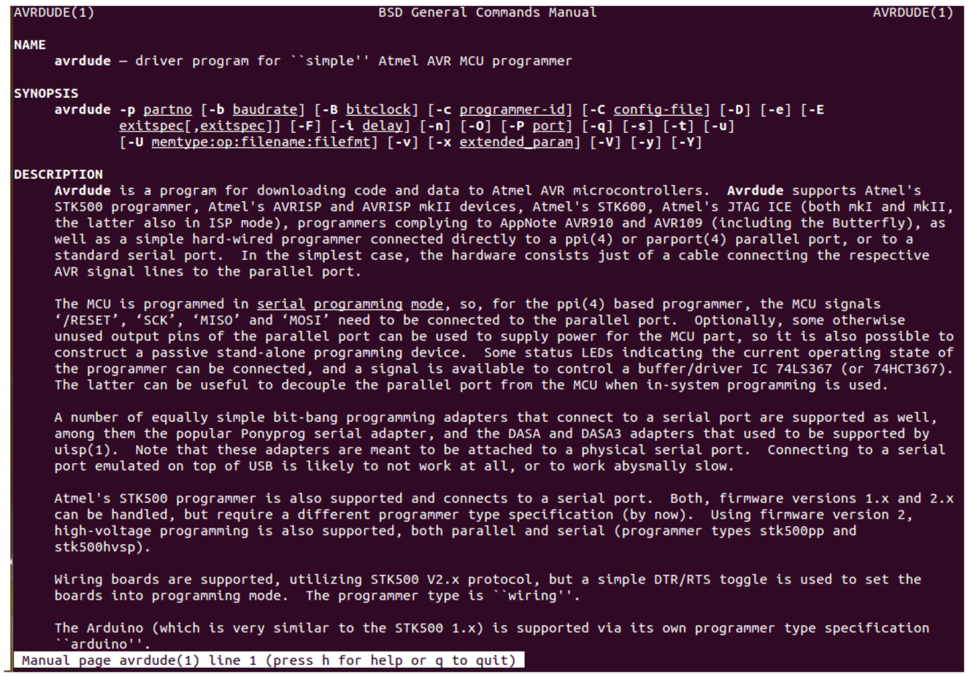
The IC is a standalone one without the Bootloader. One of the ways of dumping the code, is using an In System Programmer or an ISP. So, for burning the same we use the softwares: AVRDUDE[1] - AVR Downloader Uploader - is a program for downloading and uploading the on-chip memories of Atmels AVR microcontrollers. It can program the Flash and EEPROM, and where supported by the serial programming protocol, it can program fuse and lock bits. It's available as `WINAVR` for `Windows` and can be downloaded from here. `http://softlayer-sng.dl.sourceforge.net/project/winavr/WinAVR/20100110/WinAVR-20100110-install.exe`

Instead if you have `Linux` system, it's already pre-installed under the name `AVRDUDE`. Just enter the below command line on your `terminal`

```
man avrdude
```

You would see a screen full of text, something like the one below. To exit press `q`. If you don't get a screen like that, then open `Ubuntu Software Center` and on the search bar enter `avrdude`, then install it. Enter the same command into the terminal again and you shall see the manual.

[1]AVRDUDE has once been started by Brian S. Dean as a private project of an in-system programmer for the Atmel AVR microcontroller series, as part of the Opensource and free software tools collection available for these controllers.



```
AVRDUDE(1)                       BSD General Commands Manual                       AVRDUDE(1)

NAME
     avrdude — driver program for ``simple'' Atmel AVR MCU programmer

SYNOPSIS
     avrdude -p partno [-b baudrate] [-B bitclock] [-c programmer-id] [-C config-file] [-D] [-e] [-E
             exitspec[,exitspec]] [-F] [-i delay] [-n] [-O] [-P port] [-q] [-s] [-t] [-u]
             [-U memtype:op:filename:filefmt] [-v] [-x extended_param] [-V] [-y] [-Y]

DESCRIPTION
     Avrdude is a program for downloading code and data to Atmel AVR microcontrollers.  Avrdude supports Atmel's
     STK500 programmer, Atmel's AVRISP and AVRISP mkII devices, Atmel's STK600, Atmel's JTAG ICE (both mkI and mkII,
     the latter also in ISP mode), programmers complying to AppNote AVR910 and AVR109 (including the Butterfly), as
     well as a simple hard-wired programmer connected directly to a ppi(4) or parport(4) parallel port, or to a
     standard serial port.  In the simplest case, the hardware consists just of a cable connecting the respective
     AVR signal lines to the parallel port.

     The MCU is programmed in serial programming mode, so, for the ppi(4) based programmer, the MCU signals
     '/RESET', 'SCK', 'MISO' and 'MOSI' need to be connected to the parallel port.  Optionally, some otherwise
     unused output pins of the parallel port can be used to supply power for the MCU part, so it is also possible to
     construct a passive stand-alone programming device.  Some status LEDs indicating the current operating state of
     the programmer can be connected, and a signal is available to control a buffer/driver IC 74LS367 (or 74HCT367).
     The latter can be useful to decouple the parallel port from the MCU when in-system programming is used.

     A number of equally simple bit-bang programming adapters that connect to a serial port are supported as well,
     among them the popular Ponyprog serial adapter, and the DASA and DASA3 adapters that used to be supported by
     uisp(1).  Note that these adapters are meant to be attached to a physical serial port.  Connecting to a serial
     port emulated on top of USB is likely to not work at all, or to work abysmally slow.

     Atmel's STK500 programmer is also supported and connects to a serial port.  Both, firmware versions 1.x and 2.x
     can be handled, but require a different programmer type specification (by now).  Using firmware version 2,
     high-voltage programming is also supported, both parallel and serial (programmer types stk500pp and
     stk500hvsp).

     Wiring boards are supported, utilizing STK500 V2.x protocol, but a simple DTR/RTS toggle is used to set the
     boards into programming mode.  The programmer type is ``wiring''.

     The Arduino (which is very similar to the STK500 1.x) is supported via its own programmer type specification
     ``arduino''.
 Manual page avrdude(1) line 1 (press h for help or q to quit)
```

Figure 9: AVRDUDE manual on Linux

4.2 Setting up WINAVR on WINDOWS

Step 1: First, open up the start screen and then search : `Command Prompt`

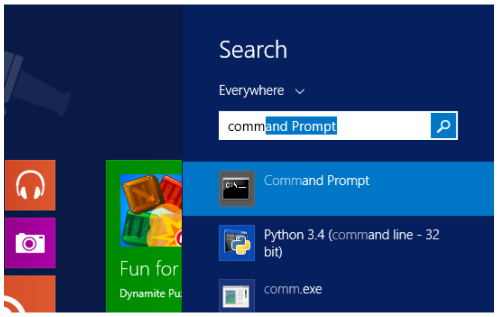
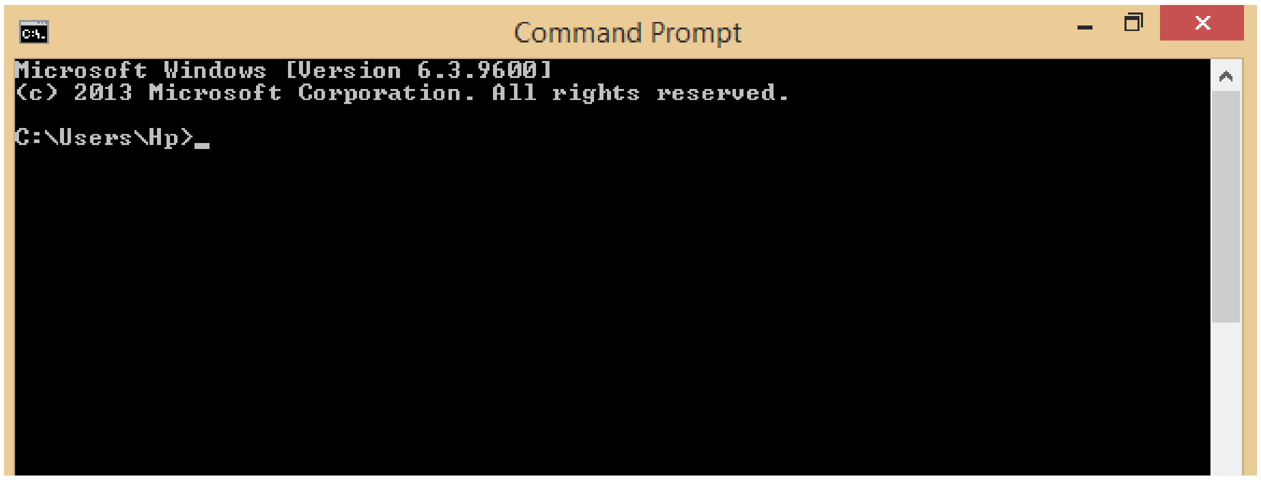


Figure 10: Terminal in Search screen

Step 2: Then, a screen like this shall pop up.

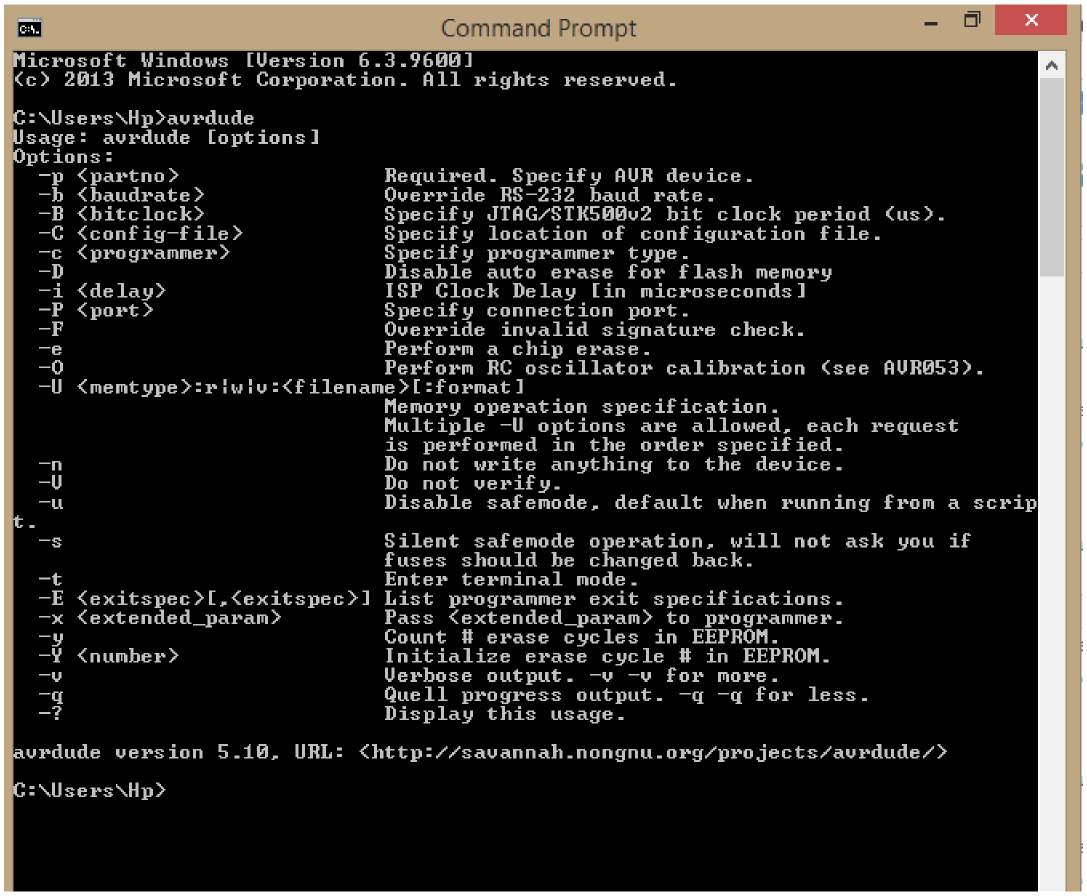


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Hp>_
```

Figure 11: Terminal in use

Step 3: Enter into the command prompt for windows: `avrdude`

You should get prompts and flags mentioned like this, which implies that you have downloaded the software properly.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Hp>avrdude
Usage: avrdude [options]
Options:
  -p <partno>                Required. Specify AVR device.
  -b <baudrate>              Override RS-232 baud rate.
  -B <bitclock>              Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>           Specify location of configuration file.
  -c <programmer>            Specify programmer type.
  -D                         Disable auto erase for flash memory
  -i <delay>                 ISP Clock Delay [in microseconds]
  -P <port>                  Specify connection port.
  -F                         Override invalid signature check.
  -e                         Perform a chip erase.
  -O                         Perform RC oscillator calibration (see AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                             Memory operation specification.
                             Multiple -U options are allowed, each request
                             is performed in the order specified.
  -n                         Do not write anything to the device.
  -V                         Do not verify.
  -u                         Disable safemode, default when running from a scrip
t.
  -s                         Silent safemode operation, will not ask you if
                             fuses should be changed back.
  -t                         Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>        Pass <extended_param> to programmer.
  -y                         Count # erase cycles in EEPROM.
  -Y <number>                Initialize erase cycle # in EEPROM.
  -v                         Verbose output. -v -v for more.
  -q                         Quell progress output. -q -q for less.
  -?                         Display this usage.

avrdude version 5.10, URL: <http://savannah.nongnu.org/projects/avrdude/>

C:\Users\Hp>
```

Figure 12: Avrdude access via WINAVR and Terminal

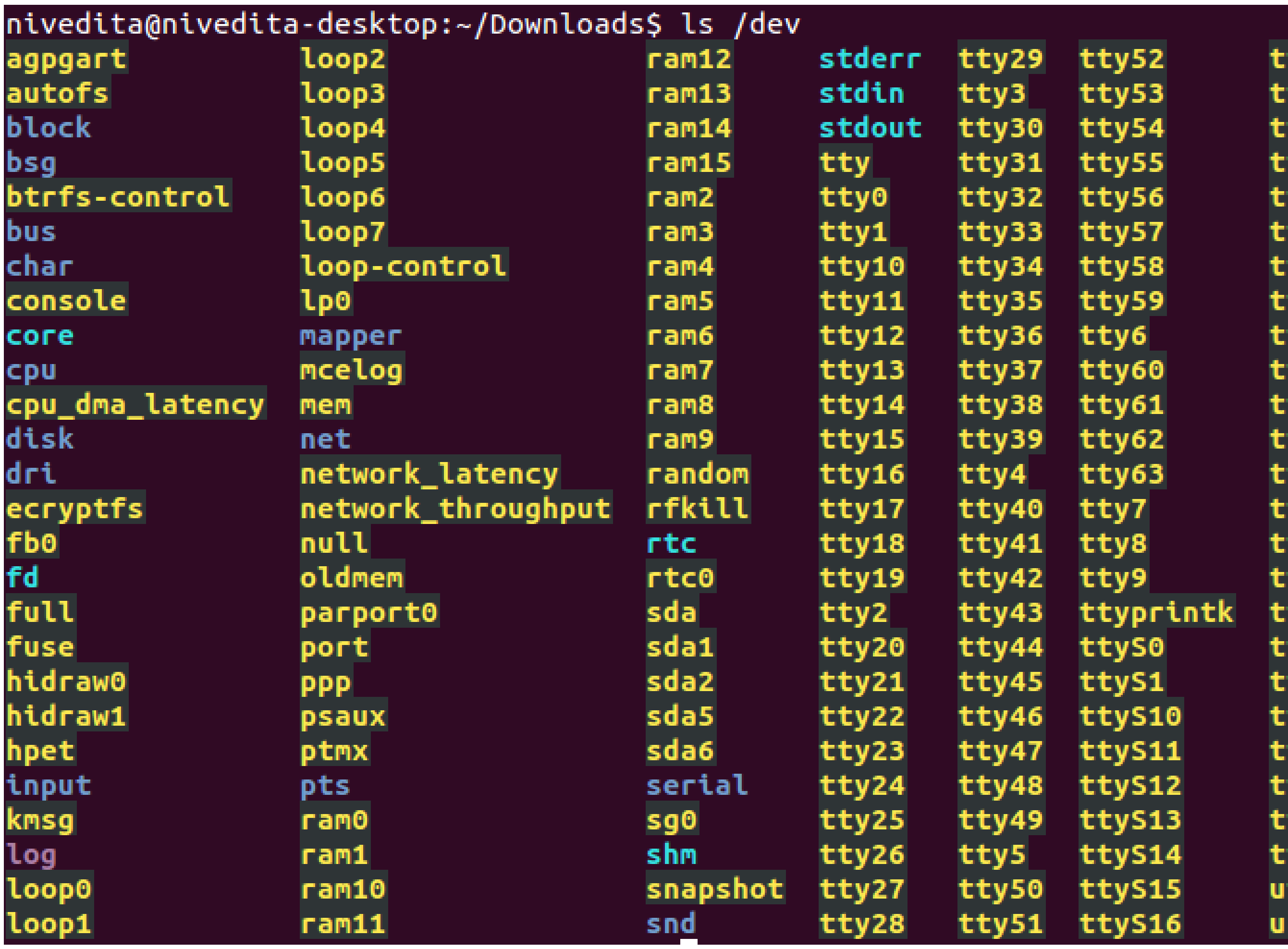
`IMP: If you do not get this, please download the software and install it again.`

After this, we can proceed with dumping the HEX code onto the IC.

4.3 Identifying the Port of the programmer

There are several programmers available to program any controller IC. They are called as ISPs. USBASP is one of them. To identify the port at which its connected in `Linux`.

Type into the `Terminal: ls /dev`



```
nivedita@nivedita-desktop:~/Downloads$ ls /dev
agpgart          loop2               ram12     stderr  tty29  tty52      t
autofs           loop3               ram13     stdin   tty3   tty53      t
block            loop4               ram14     stdout  tty30  tty54      t
bsg              loop5               ram15     tty     tty31  tty55      t
btrfs-control    loop6               ram2      tty0    tty32  tty56      t
bus              loop7               ram3      tty1    tty33  tty57      t
char             loop-control        ram4      tty10   tty34  tty58      t
console          lp0                 ram5      tty11   tty35  tty59      t
core             mapper              ram6      tty12   tty36  tty6       t
cpu              mcelog              ram7      tty13   tty37  tty60      t
cpu_dma_latency  mem                 ram8      tty14   tty38  tty61      t
disk             net                 ram9      tty15   tty39  tty62      t
dri              network_latency     random    tty16   tty4   tty63      t
ecryptfs         network_throughput  rfkill    tty17   tty40  tty7       t
fb0              null                rtc       tty18   tty41  tty8       t
fd               oldmem              rtc0      tty19   tty42  tty9       t
full             parport0            sda       tty2    tty43  ttyprintk  t
fuse             port                sda1      tty20   tty44  ttyS0      t
hidraw0          ppp                 sda2      tty21   tty45  ttyS1      t
hidraw1          psaux               sda5      tty22   tty46  ttyS10     t
hpet             ptmx                sda6      tty23   tty47  ttyS11     t
input            pts                 serial    tty24   tty48  ttyS12     t
kmsg             ram0                sg0       tty25   tty49  ttyS13     t
log              ram1                shm       tty26   tty5   ttyS14     t
loop0            ram10               snapshot  tty27   tty50  ttyS15     u
loop1            ram11               snd       tty28   tty51  ttyS16     u
```

Figure 13: Terminal in Search screen

This gives the list of connected devices on the system. After this, connect the ISP to the system and then again enter the same command on the terminal and then find out the new entries in the list. That shall be the port at which the ISP is connected. To use it in the command line with the -P flag just mention `/dev/ttyUSBx` where the argument trailing the `/dev/` one is the port at which the device is configured. This helps in recognising the device when programming on the terminal. For example use it in the command line with the appropriate flag like this.

```
avrdude -p atmega16 -P /dev/ttyUSB0 -c ponyser -v -U flash:w:test1.hex
```

Note: It can also be used with the correct defined By-id path in the serial programming ISP(RS232) by mentioning

```
/dev/serial/by-id/usb-FTDI_FT232R_USB_UART_XXXXX-port0
```

which is same as the `/dev/ttyUSBx` for identifying the same programmer. So both arguments are valid in case of RS232.

4.4 USBASP as ISP

Step 1: We require a precompiled hex file to be burnt onto our microcontroller. In our case, we have generated it by doing simple ladder programming in LDMicro and then compiling it. Before that, ensure that the proper target controller from the drop-down menu is chosen. For input and output part, assign a certain pin no. of the controller to that of the desired application. Now, a certain name is given to the file(say, blink.hex) at the destination folder.

Step 2: After being done with all the software part, one shall supply the power from 12V SMPS to the development board. Now check whether the controller is powered up properly with 5 Volt or not from the appropriate `Vcc` and `GND` pins of the controller.

Step 3: Now, just connect the FRC cable from USBASP to the 10 pin shrouded header on the board.

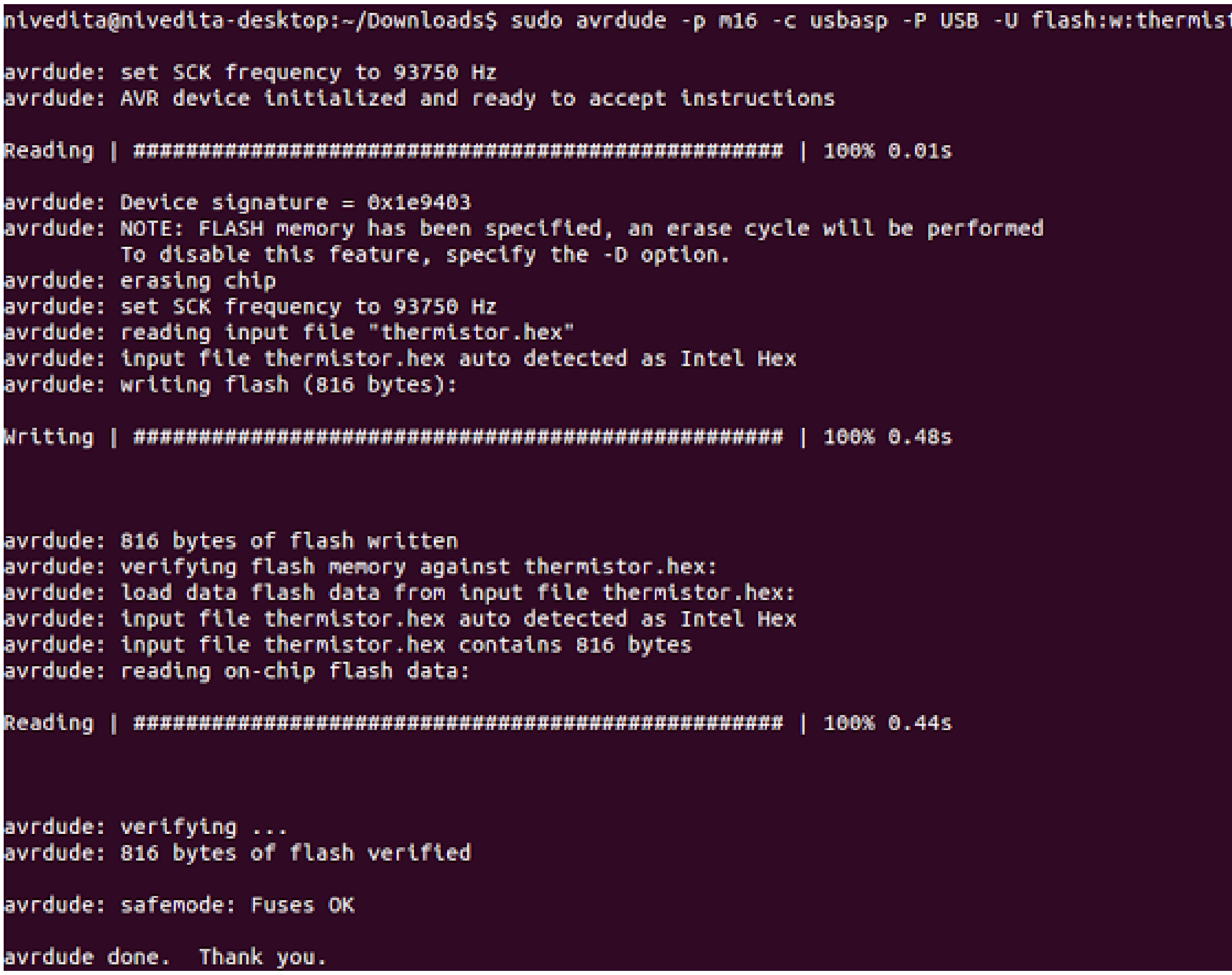
Step 4: Now, turn on `Terminal` and then change the directory to the one in which you have the saved HEX file to be dumped onto the processor.

Step 5: Now, in the terminal, enter the following command.

```
sudo avrdude -p atmega16 -P usb -c usbasp -B10 -U ash:w:Blink.hex
```

You shall see such a process going on in the terminal like the one shown below, which means that the process of erasing and writing the internal memory of the ATmega IC is in the process. On correct execution you shall get such a result. Whilst it's burning the code, the red light on the USBASP will be lit showing that it is communicating with the controller appropriately.

Remember, after the fuses are set (covered in the following sections in detail), then the -B10 flag is necessary and can be deleted.



```
nivedita@nivedita-desktop:~/Downloads$ sudo avrdude -p m16 -c usbasp -P USB -U flash:w:thermist

avrdude: set SCK frequency to 93750 Hz
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9403
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: set SCK frequency to 93750 Hz
avrdude: reading input file "thermistor.hex"
avrdude: input file thermistor.hex auto detected as Intel Hex
avrdude: writing flash (816 bytes):

Writing | ################################################## | 100% 0.48s



avrdude: 816 bytes of flash written
avrdude: verifying flash memory against thermistor.hex:
avrdude: load data flash data from input file thermistor.hex:
avrdude: input file thermistor.hex auto detected as Intel Hex
avrdude: input file thermistor.hex contains 816 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.44s



avrdude: verifying ...
avrdude: 816 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Figure 14: Using USBASP as an ISP

4.4.1 Troubleshooting

✓Check if the board is powered from the SMPS, if not, power it and try it once powered.

✓Check if all the essential flags in the command line are incorporated, if not make the appropriate changes.

✓Check if the FRC cable of the USBASP is not stranded in the midway, if so, replace it.

✓If all these fail, check if the appropriate MOSI, MISO, SCK, RST and GND pins of the header are appropriately shorted with the ones of the IC and also that the controller firmly sits on the base.

4.5 RS232 as a Programmer

To program the IC, we can also use DB9 connector via RS232 cable. The DB9 connector looks like the one below.

Figure 15: DB9 connector

For programming the IC using DB9 connector via SPI protocol, we need yo build up some external circuitry first.

The circuit connections are done as the one shown below. This involves connecting some electrical components, which are in the end connected to the same 10 pin shrouded headers that are used for the USBASP as a programmer. In the programmer board the connections made are like the one shown below.

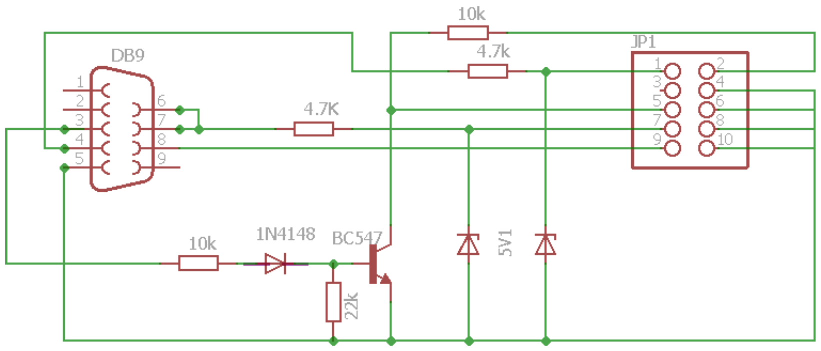


Figure 16: DB9 connector

Using DB-9 for the communication as per the RS-232 standard is chosen as the protocol is the oldest, easiest and doesn't require many hardware resources to be incorporated on the board for the IC to be programmed. As most of the present day systems like laptops may not have these ports, we can use a USB to RS-232 cable, which can be plugged into the USB port present on the system.

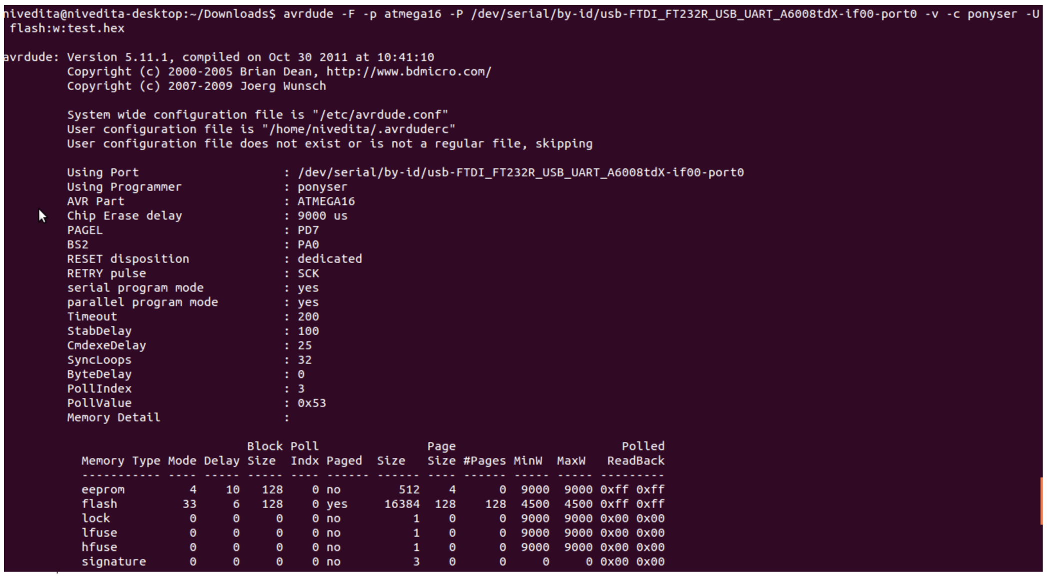
4.6 Programming using RS-232

Step 1: Connect the given RS-232 cable to the DB-9 port on the programmer board.

Step 2: You should open command line and then enter into the directory in which you have saved the HEX file compiled using LDMicro.

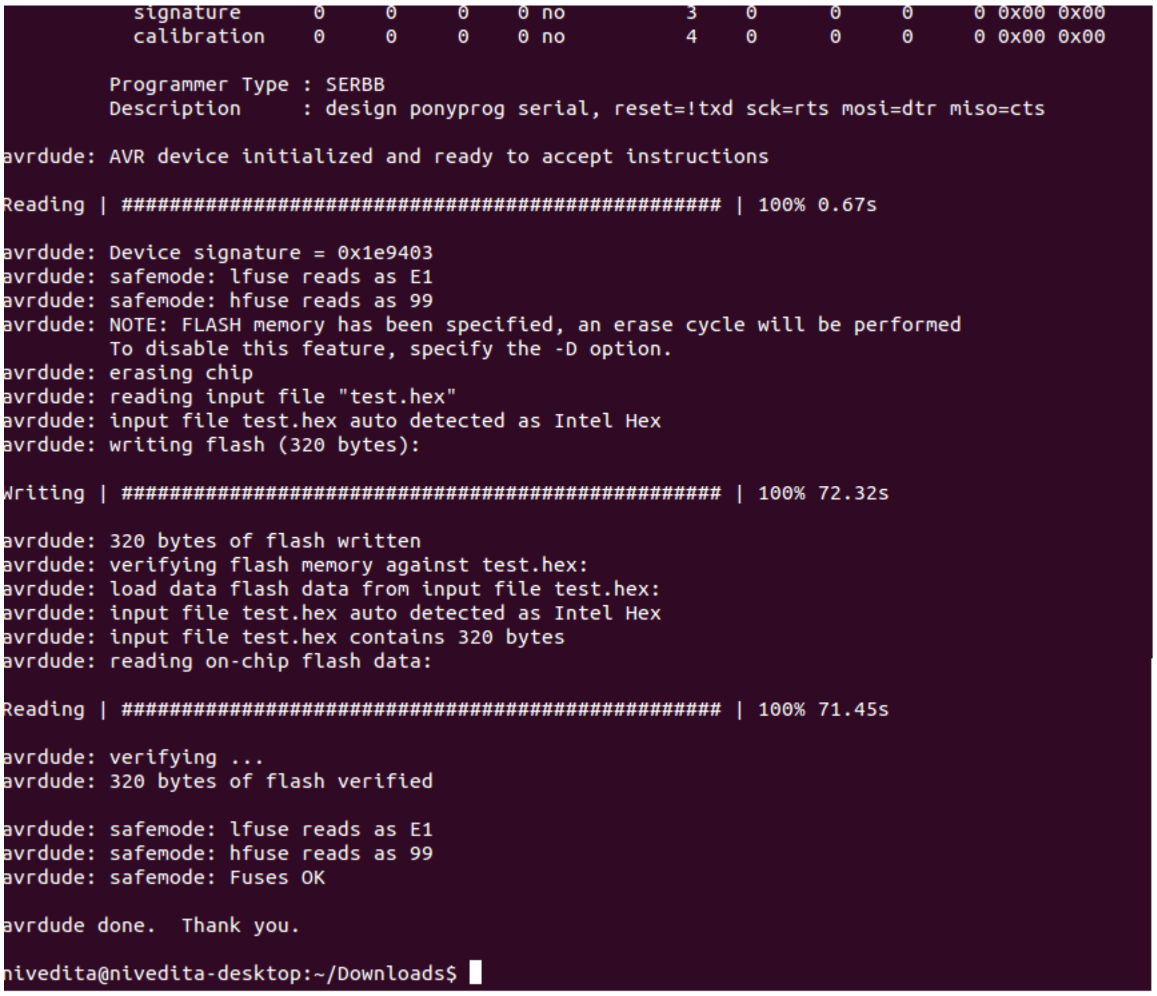
Step 3: Now that all the things are ready, finally enter the following command on `Terminal`.

```
avrdude -p atmega16 -P /dev/ttyUSB0 -c ponyser -v -U flash:w:test1.hex
```



```
nivedita@nivedita-desktop:~/Downloads$ avrdude -F -p atmega16 -P /dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A6008tdX-if00-port0 -v -c ponyser -U
 flash:w:test.hex

avrdude: Version 5.11.1, compiled on Oct 30 2011 at 10:41:10
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2009 Joerg Wunsch

         System wide configuration file is "/etc/avrdude.conf"
         User configuration file is "/home/nivedita/.avrduderc"
         User configuration file does not exist or is not a regular file, skipping

         Using Port                    : /dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A6008tdX-if00-port0
         Using Programmer              : ponyser
         AVR Part                      : ATMEGA16
         Chip Erase delay              : 9000 us
         PAGEL                         : PD7
         BS2                           : PA0
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 200
         StabDelay                     : 100
         CmdexeDelay                   : 25
         SyncLoops                     : 32
         ByteDelay                     : 0
         PollIndex                     : 3
         PollValue                     : 0x53
         Memory Detail                 :

                                  Block Poll               Page                       Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
           eeprom         4    10   128    0 no        512    4      0  9000  9000 0xff 0xff
           flash         33     6   128    0 yes     16384  128    128  4500  4500 0xff 0xff
           lock           0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           lfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00
```

This result is obtained if all the connections are fine. If only you obtain such a result, it means that the HEX code has been burnt onto the IC. If you obtain any other result, please look at the troubleshooting section.



```
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00
           calibration    0     0     0    0 no          4    0      0     0     0 0x00 0x00

         Programmer Type : SERBB
         Description     : design ponyprog serial, reset=!txd sck=rts mosi=dtr miso=cts

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.67s

avrdude: Device signature = 0x1e9403
avrdude: safemode: lfuse reads as E1
avrdude: safemode: hfuse reads as 99
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "test.hex"
avrdude: input file test.hex auto detected as Intel Hex
avrdude: writing flash (320 bytes):

Writing | ################################################## | 100% 72.32s

avrdude: 320 bytes of flash written
avrdude: verifying flash memory against test.hex:
avrdude: load data flash data from input file test.hex:
avrdude: input file test.hex auto detected as Intel Hex
avrdude: input file test.hex contains 320 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 71.45s

avrdude: verifying ...
avrdude: 320 bytes of flash verified

avrdude: safemode: lfuse reads as E1
avrdude: safemode: hfuse reads as 99
avrdude: safemode: Fuses OK

avrdude done.  Thank you.

nivedita@nivedita-desktop:~/Downloads$ 
```

Figure 17: Programming via RS-232

4.6.1 Troubleshooting

✓Check if the board is powered from the SMPS, if not, power it and try it once powered.

✓Check if all the essential flags in the command line are incorporated, if not make the appropriate changes.

✓Check if the RS-232 cable of the is connected securely to the DB9 connector.

✓If all of this fails, then check if the reset pin of the microcontroller is at `+5 Volts` if not then externally connect it to `+5 Volts`.

4.7 Setting up the fuse bits

Now that the programmer is installed on the system, we need to adjust the properties of the controller IC such that it meets the necessary requirements along with the peripherals installed on the board. Essentially, fuse bits are the ones that decide how the controller responds, like which clock frequency it responds to or its programming availabilities. Setting up the fuse is a crucial task as the controller may not respond later if the fuse bits that are set are not in accordance with the attached peripherals. Before setting the fuses, the factory settings on the controller make sure it works on the internal oscillator of clock speed 1MHz. It's mostly dependent on 2 fuses - lfuse and hfuse. Both of these have hex 8 bit values. When working on communication with the device, like UART, these fuse bits play a key role, if not defined, the controller works on the internal clock that's much slower than the external crystal. For the controller, 16MHz external crystal was selected. This meant the controller shall configure this external crystal and work on it.

`WARNING: Fuse bits control the way the controller responds. If any mistake is made in setting up the fuse bits, then the controller becomes isolated from the external circuit. This means that the controller doesn't respond and renders useless. Hence, always triple check before entering the fuse bits. This is the best practice, or else you may end up wasting one IC.`

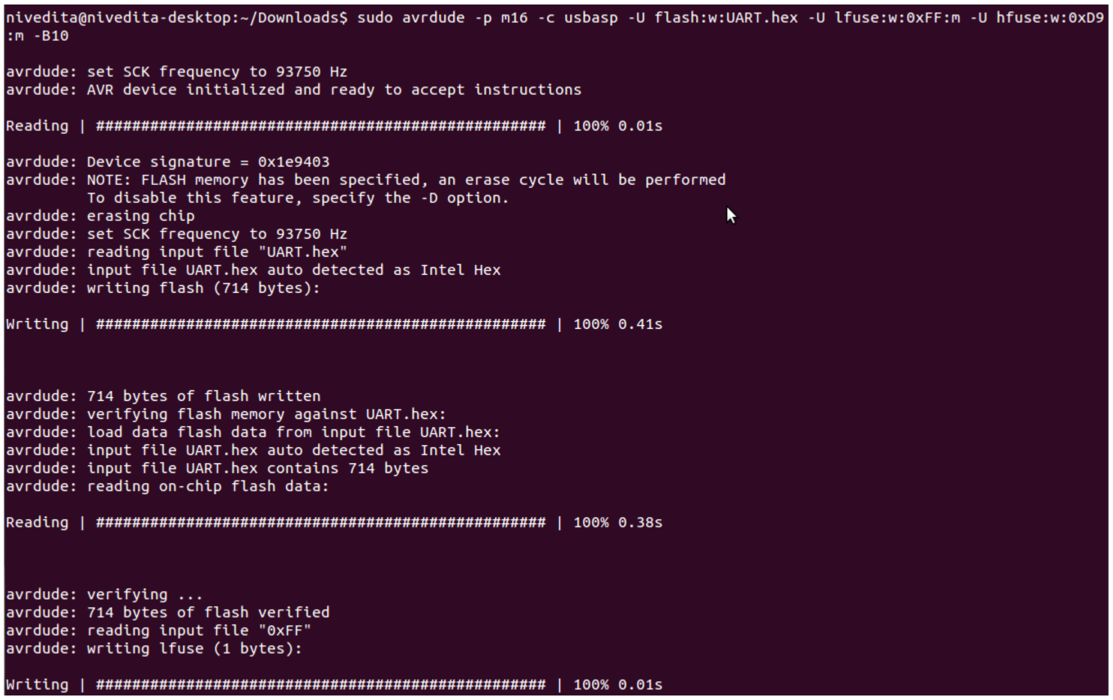
Type the following command into the `Terminal`:

For USBASP:

```
sudo avrdude -p m16 -c usbasp -U lfuse:w:0xFF:m -U hfuse:w:0xD9:m
-B10
```

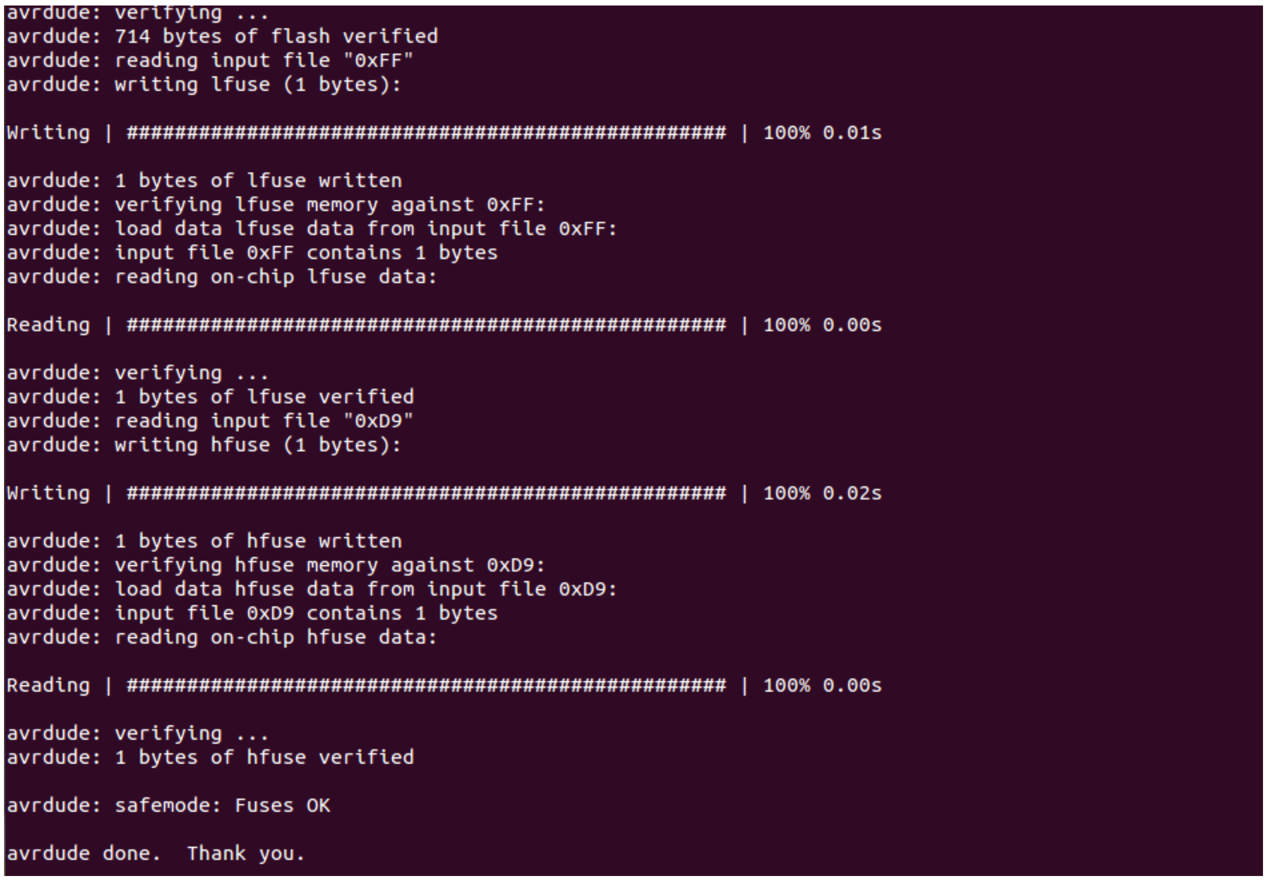
For RS232:

```
avrdude -p m16 -c ponyser -P /dev/ttyUSB0 -U lfuse:w:0xFF:m -U
hfuse:w:0xD9:m
```



```
nivedita@nivedita-desktop:~/Downloads$ sudo avrdude -p m16 -c usbasp -U flash:w:UART.hex -U lfuse:w:0xFF:m -U hfuse:w:0xD9
:m -B10

avrdude: set SCK frequency to 93750 Hz
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9403
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: set SCK frequency to 93750 Hz
avrdude: reading input file "UART.hex"
avrdude: input file UART.hex auto detected as Intel Hex
avrdude: writing flash (714 bytes):

Writing | ################################################## | 100% 0.41s



avrdude: 714 bytes of flash written
avrdude: verifying flash memory against UART.hex:
avrdude: load data flash data from input file UART.hex:
avrdude: input file UART.hex auto detected as Intel Hex
avrdude: input file UART.hex contains 714 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.38s



avrdude: verifying ...
avrdude: 714 bytes of flash verified
avrdude: reading input file "0xFF"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.01s
```

Figure 18: Setting the fuse bits



```
avrdude: verifying ...
avrdude: 714 bytes of flash verified
avrdude: reading input file "0xFF"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0xFF:
avrdude: load data lfuse data from input file 0xFF:
avrdude: input file 0xFF contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified
avrdude: reading input file "0xD9"
avrdude: writing hfuse (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0xD9:
avrdude: load data hfuse data from input file 0xD9:
avrdude: input file 0xD9 contains 1 bytes
avrdude: reading on-chip hfuse data:

Reading | ################################################## | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Figure 19: Setting the fuse bits

Try to read the fuse bits after setting it. This will help you recognize if the IC has configured the external 16MHz crystal.

If using USBASP enter :

```
avrdude -p m16 -C usb -P usbasp
```

If using RS232 programmer enter :

```
avrdude -p m16 -C /dev/ttyUSB0 -P ponyser
```

You will get a screen like this :
You can see that the fuse bits are read off. This implies that the external

crystal is working in sync with the controller. You can verify that the fuse bits are the ones that you set earlier.

If such a reading doesn't come up, follow these steps:

✓Check that the command line is proper and you have written the correct command line with all the necessary flags like the controller mentioned is correct.

✓Make sure that the power supply is connected and the programmer i.e. USBASP or RS232 is configured and the correct programmer is mentioned in the command line

✓Make sure that the controller is firmly connected to the IC base. If not press it so that it sits firmly.

✓If these do not work, then remove the controller IC from the base and then remove the crystal and solder a new 16MHz crystal. Make sure that you do not heat the crystal extensively. Due to this there's usually a mismatch in the clock speeds.

5 Using UART function

UART basically is the most basic protocol for serially transmitting and receiving the data from the sensor. UART stands for Universal Asynchronous Receiver Transmitter. The ladder logic allows us to program the UART facility easily in LD Micro. The values monitored in the ladder can be used to be displayed on the serial monitor. Say for example you have values that change with time and are varying, that is the value is analog, which means the values change over time and aren't discrete. Hence, monitoring the specific values become easy over the serial monitor.

Setting up the UART is so easy. Just connect the TX near the MAX232 IC to pin 14 of the IC and connect RX near the MAX232 IC to pin 15 of the IC. Now in the ladder use the UART send feature. The variable associated with UART send part will be the value that you would want to monitor. Mostly these values being analog will be connected to analog pins i.e. pins of Port A. So, the variables associated with the analog values have Avar kind of a structure, when using that for UART, append the 'A' in front of the variable name. So the variable linked in the UART send part will be 'Avar' and not 'var'.

For looking at the associated values, install a serial monitor on the system. This is helpful to the the Raw data associated. When monitored, the values seen are hex in nature, they start with the '0x' part, which implies the code is hexadecimal. This is helpful for suggesting thresholds and creating a level for a particular application. The hex values are assigned like 0x00 to 0x3FF which means the values range from 0 to 1023. This means the values 0V is mapped to 0x00 and value 5V (maximum from the controller) will be assigned 0x3FF value. This is how the mapping will be done.