Society of Robotics & Automation

Workshop Manual

WALL-E 2014

an autonomous Line Follower Robot

Veermata Jijabai Technological Institute
H.R. Mahajani Marg
Mumbai 400 019

sra.vjti.info

Kit Contents

Mechanical	
Part	**Quantity**
Chassis	1
Motors	2
Wheels	2
Caster Wheel	1
12V Adapter	1
Screwdrivers	1
Electronic	
Part	**Quantity**
SRA Development Board	1
USBASP Programmer	1
Sensor Board	1
FRC Cable	1
16x2 LCD	1

INDEX

TITLE	PAGE NO.
THE MICROCONTROLLER	2
DEVELOPMENT BOARD: i. Power Supply ii. Microcontroller iii. Motor Driver iv. LCD Display	 10 12 15 18
BRUSHING UP CODING	22
USBASP PROGRAMMER	25
Making a MAKEFILE	39
ABBREVIATIONS	45
SRA LIBRARY	46

The Microcontroller

What is it?

Just like our brain ,differentiates and analyses the surroundings ,utilizing key assets like our eyes , ears, nose, hands ,legs etc. .The microcontroller does the same for our board, with assets including led arrays, LCD modules, Sharp sensors, motor drivers.

A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and

programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications

General Gyaan:

Microprocessor is an IC which has only the CPU inside them i.e. only the processing powers such as Intel's Pentium 1,2,3,4, core 2 duo, i3, i5 etc. These microprocessors don't have RAM, ROM, and other peripheral on the chip. A system designer has to add them externally to make them functional. Application of microprocessor includes Desktop PC's, Laptops, notepads etc.

But this is not the case with Microcontrollers. Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single

chip. At times it is also termed as a mini computer or a computer on a single chip. Today different manufacturers produce microcontrollers with a wide range of features available in different versions. Some manufacturers are ATMEL, Microchip, TI, Freescale, Philips, Motorola etc.

Microcontrollers are designed to perform specific tasks. Specific means applications where the relationship of input and output is defined. Depending on the input, some processing needs to be done and output is delivered. For example, keyboards, mouse, washing machine, digicam, pen drive, remote, microwave, cars, bikes, telephone, mobiles, watches, etc. Since the applications are very specific, they need small resources like RAM, ROM, I/O ports etc. and hence can be embedded on a single chip. This in turn reduces the size and the cost.

Microprocessor find applications where tasks are unspecific like developing software, games, websites, photo editing, creating documents etc. In such cases the relationship between input and output is not defined. They need high amount of resources like RAM, ROM, I/O ports etc.

The clock speed of the Microprocessor is quite high as compared to the microcontroller. Whereas the microcontrollers operate from a few MHz to 30 to 50 MHz, today's microprocessor operate above 1GHz as they perform complex tasks.

Comparing microcontroller and microprocessor :

In terms of cost is not justified. Undoubtedly a microcontroller is far cheaper than a microprocessor. However microcontroller cannot be used in place of microprocessor and using a microprocessor is not advised in place of a microcontroller as it makes the application quite costly. Microprocessor cannot be used stand alone. They need other peripherals like RAM, ROM, buffer, I/O ports etc and hence a system designed around a microprocessor is quite costly.

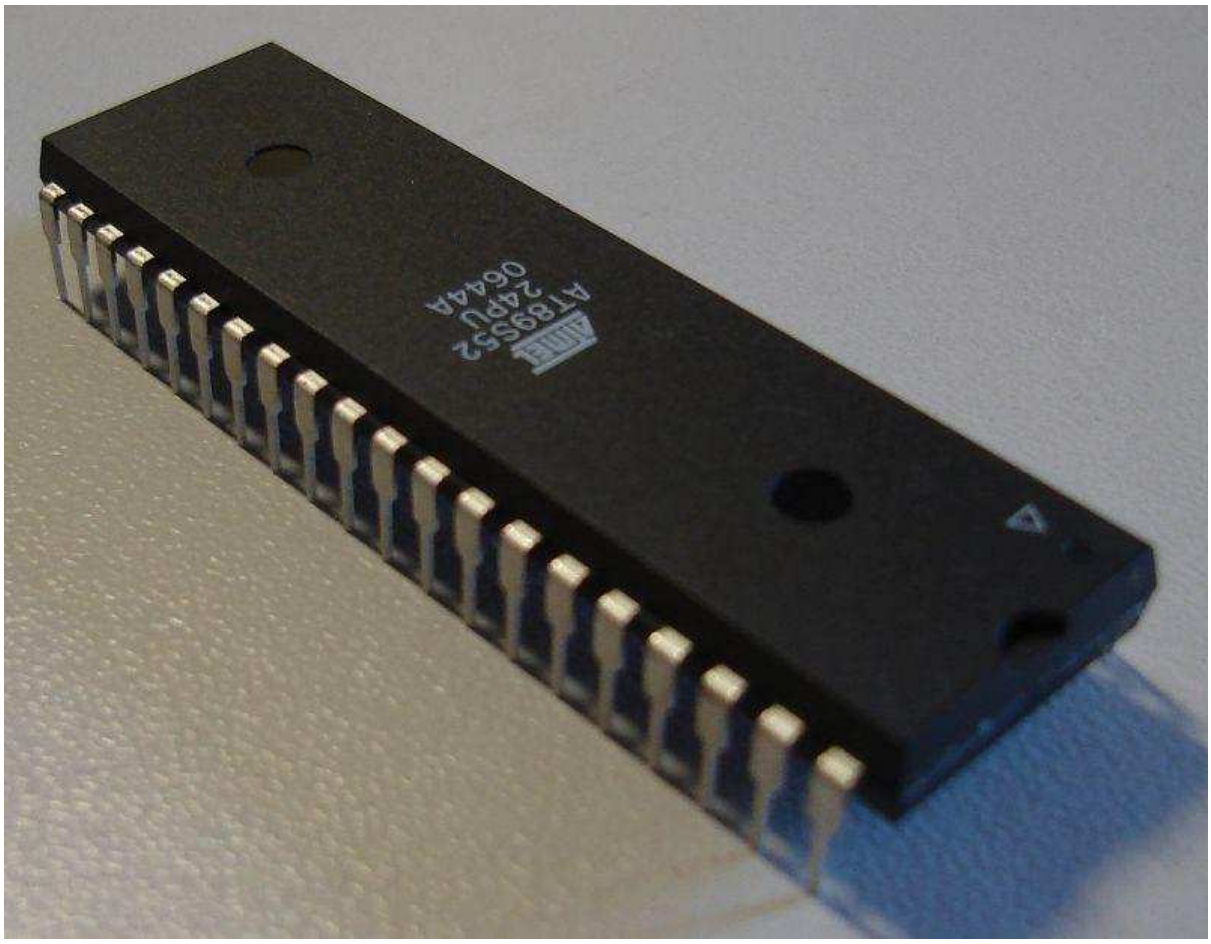


Fig.1

The ATmega 16 Microcontroller.

What is ATmega?

ATmega is basically a type of microcontroller developed by Atmel AVR.

The **AVR** is a modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed

to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

It was conceived by two students at the Norwegian Institute of Technology (NTH) Alf-EgilBogen and VegardWollan.

What types of MCUs does AVRmake?

AVRs are generally classified into six broad groups:

- **tinyAVR** — the ATtiny series
  - 0.5–16 kB program memory
  - 6–32-pin package
  - Limited peripheral set

- **megaAVR** — the ATmega series
  - 4–512 kB program memory
  - 28–100-pin package
  - Extended instruction set (multiply instructions and instructions for handling larger program memories)
  - Extensive peripheral set
- **XMEGA** — the ATxmega series
  - 16–384 kB program memory
  - 44–64–100-pin package (A4, A3, A1)
  - Extended performance features, such as DMA, "Event System", and cryptography support.
  - Extensive peripheral set with ADCs
- **Application-specific AVR**
  - megaAVRs with special features not found on the other members of the AVR family, such as LCD controller, USB controller, advanced PWM, CAN, etc.

- **FPSLIC (AVR with FPGA)**
  - FPGA 5K to 40K gates
  - SRAM for the AVR program code, unlike all other AVRs
  - AVR core can run at up to 50 MHz

- **32-bit AVRs**

In 2006 Atmel released microcontrollers based on the new, 32-bit, AVR32 architecture. They include SIMD and DSP instructions, along with other audio and video processing features. This 32-bit family of devices is intended to compete with the ARM based processors. The instruction set is similar to other RISC cores, but it is not compatible with the original AVR or any of the various ARM cores.

The Development Board

The main blocks of the development board –

1. Power supply and voltage regulation
2. Microcontroller (ATmega 16/32)(Figure 1)
3. Motor drivers
4. LCD display connections.

1. Power supply and voltage regulation

- All digital ICs work at a voltage around 5v and may get damaged at higher voltages but motors require a high voltage of about 12 volts to function. To solve this discrepancy, a voltage regulator circuit is added on to the board. The voltage regulator takes an input of 12V or higher and gives a 5V output and hence only one power supply is needed to power the entire robot.

- The voltage regulator on the board is LM7805.

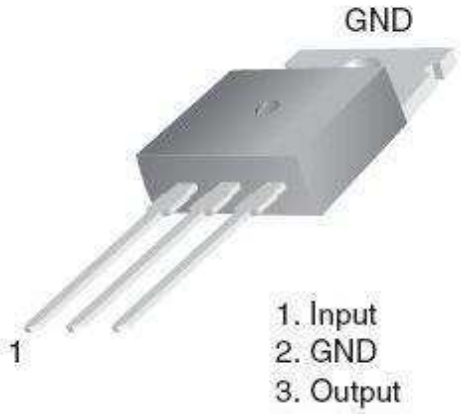


Fig.2

The board has provisions for taking 12 V input from a battery as well as from a power supply (through a power jack).

(It would look like the one below)

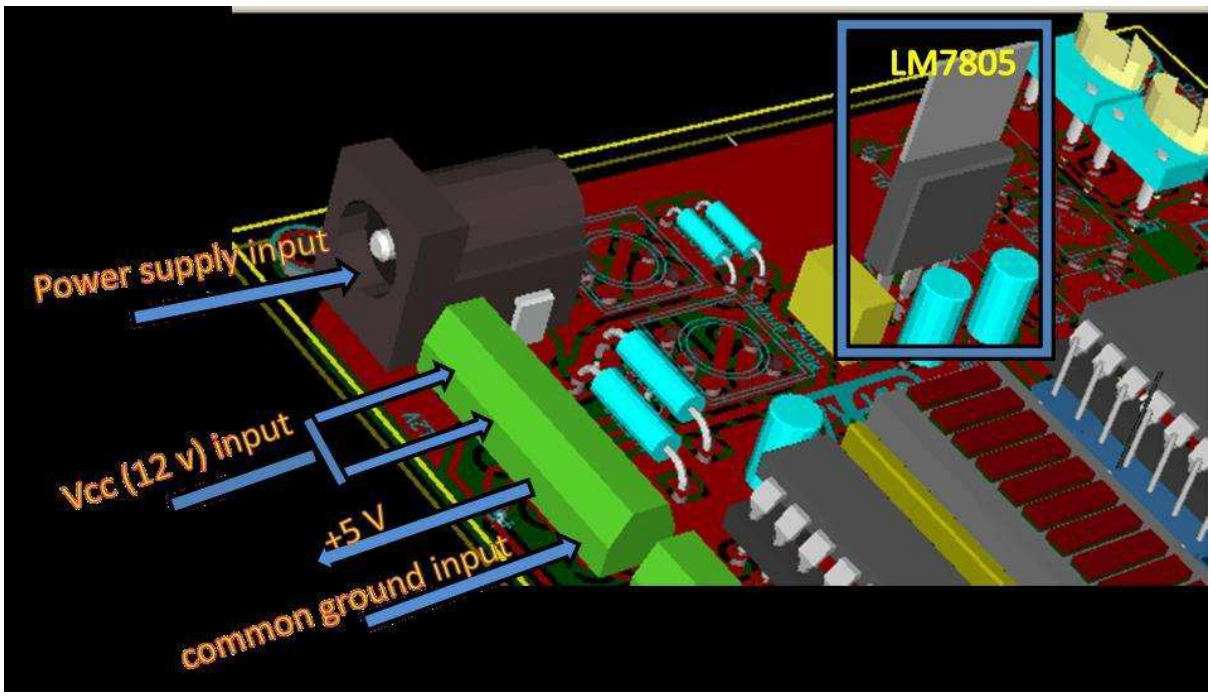


Fig.3

2. Microcontroller

The Controller we have utilized is an ATmega 16

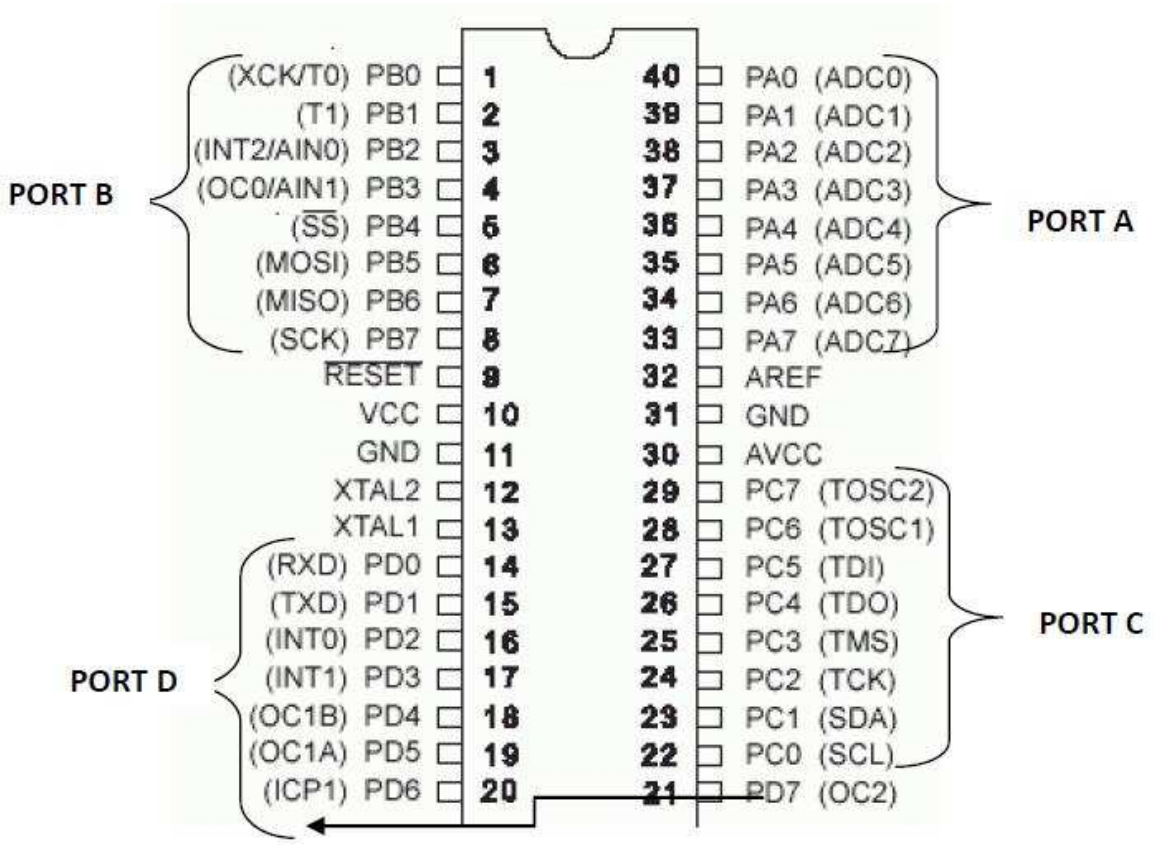


Fig.4

Pin Descriptions:

VCC: Digital supply voltage.

GND: Ground.

Port A (PA7..PA0) :

Port A serves as the analog input to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pinscan(PINx) provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with

both high sink and source capability. When pins PA0 to PA7are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0):

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Port B also serves the functions of various special features of the ATmega16.

Port C (PC7..PC0) :

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5 (TDI), PC3 (TMS) and PC2 (TCK)(Refer Fig. 4) will be activated even if a reset occurs.

Port C also serves the functions of the JTAG interface and other special features, to know more check out the ATmega 16 datasheet.

Port D (PD7..PD0):

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

RESET

Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running.

XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2

It's the Output from the inverting Oscillator amplifier.

AVCC

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.

AREF

It is the analog reference pin for A/D Convertor.

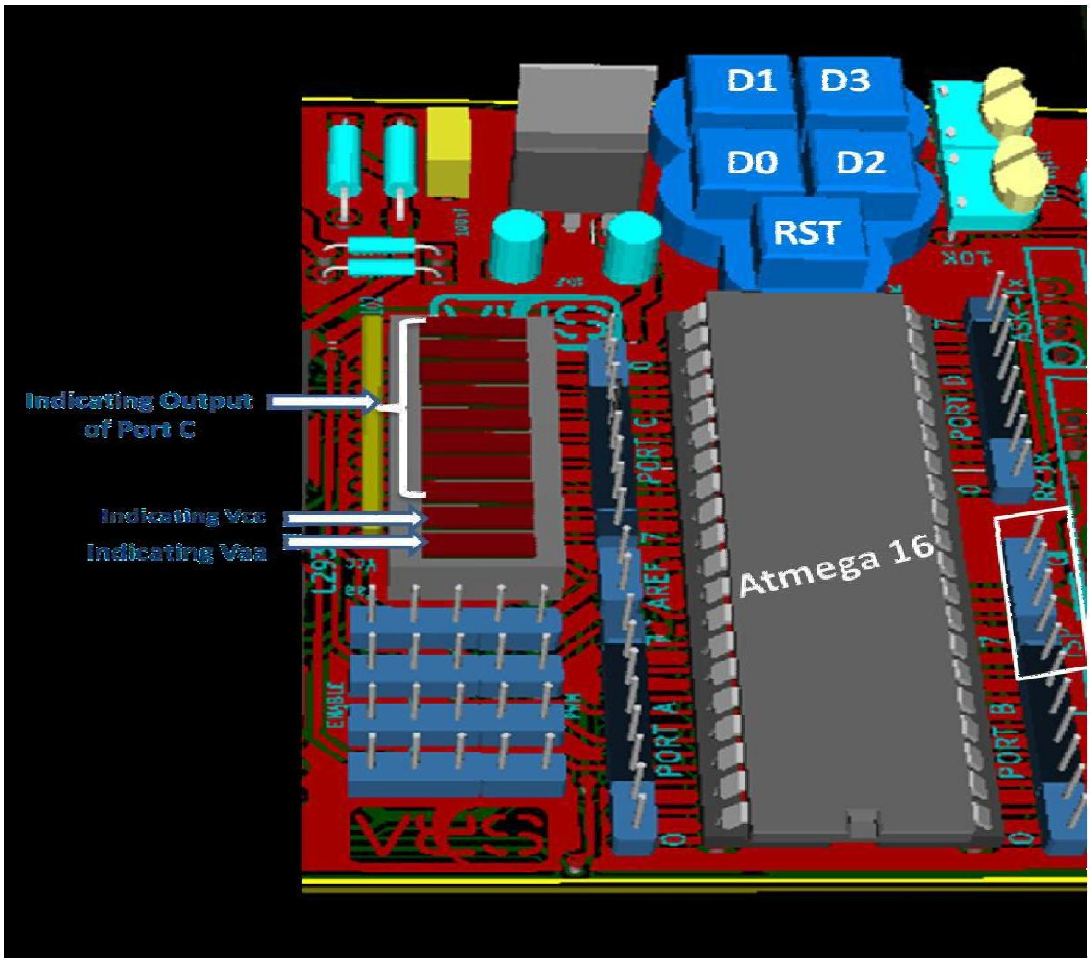


Fig.5 (For more information check out the datasheet)

3. Motor Driver

Generally, even the simplest robot requires a motor to rotate a wheel or performs particular action. Since motors require more current than the microcontroller pin can typically generate, you need some type of a switch (Transistors, MOSFET, Relay etc.,) which can accept a small current, amplify it and generate a larger current, which further drives a motor. This entire process is done by what is known as a **motor driver**

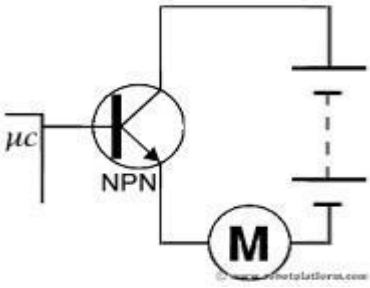


Fig.6

Introduction to L293D IC

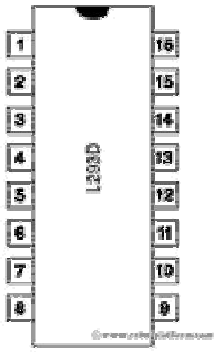


Fig.7

L293D IC generally comes as a standard 16-pin DIP (dual-in line package). This motor driver IC can simultaneously control two small motors in either direction; forward and reverse with just 4 microcontroller pins (if you do not use enable pins). Some of the features (and drawbacks) of this IC are:

1. Output current capability is limited to 600mA per channel with peak output current limited to 1.2A (non-repetitive). This means you cannot drive bigger motors with this IC. However, most small motors used in hobby robotics should work. If you are unsure whether the IC can handle a particular motor, connect the IC to its circuit and run the motor with your finger on the IC. If it gets really hot, then beware... Also note the words "non-repetitive"; if the current output repeatedly reaches 1.2A, it might destroy the drive transistors.

2. Supply voltage can be as large as 36 Volts. This means you do not have to worry much about voltage regulation.

3. L293D has an enable facility which helps you enable the IC output pins. If an enable pin (refer datasheet and figure 7) is set to logic high, then state of the inputs match the state of the outputs. If you pull this low, then the outputs will be turned off regardless of the input states

4. The datasheet also mentions an "over temperature protection" built into the IC. This means an internal

sensor senses its internal temperature and stops driving the motors if the temperature crosses a set point

5. Another major feature of **L293D** is its internal clamp diodes. This flyback diode helps protect the driver IC from voltage spikes that occur when the motor coil is turned on and off (mostly when turned off)

6. The logical low in the IC is set to 1.5V. This means the pin is set high only if the voltage across the pin crosses 1.5V which makes it suitable for use in high frequency applications like switching applications (up to 5KHz)

7. Lastly, this integrated circuit not only drives DC motors, but can also be used to drive relay solenoids, stepper motors etc.

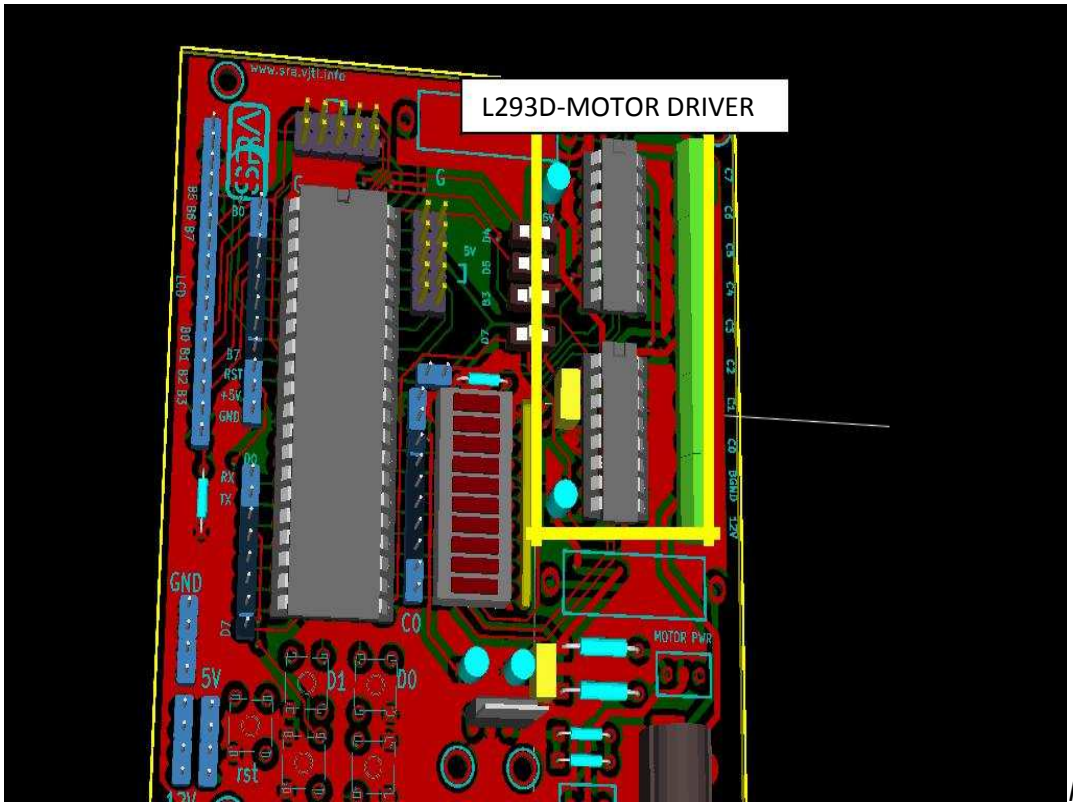


Fig.8

4. Liquid Crystal Display (LCD)

LCD is basically an output device which can be used to display various values and parameters in Robotics. LCD comes in various shapes and sizes. The LCD which SRA uses primarily is the 16x2 LCD display. Here 16 indicate the number of characters in one row and 2 represents the number of rows i.e. 2.

Following is the image of 2x16 LCD display:

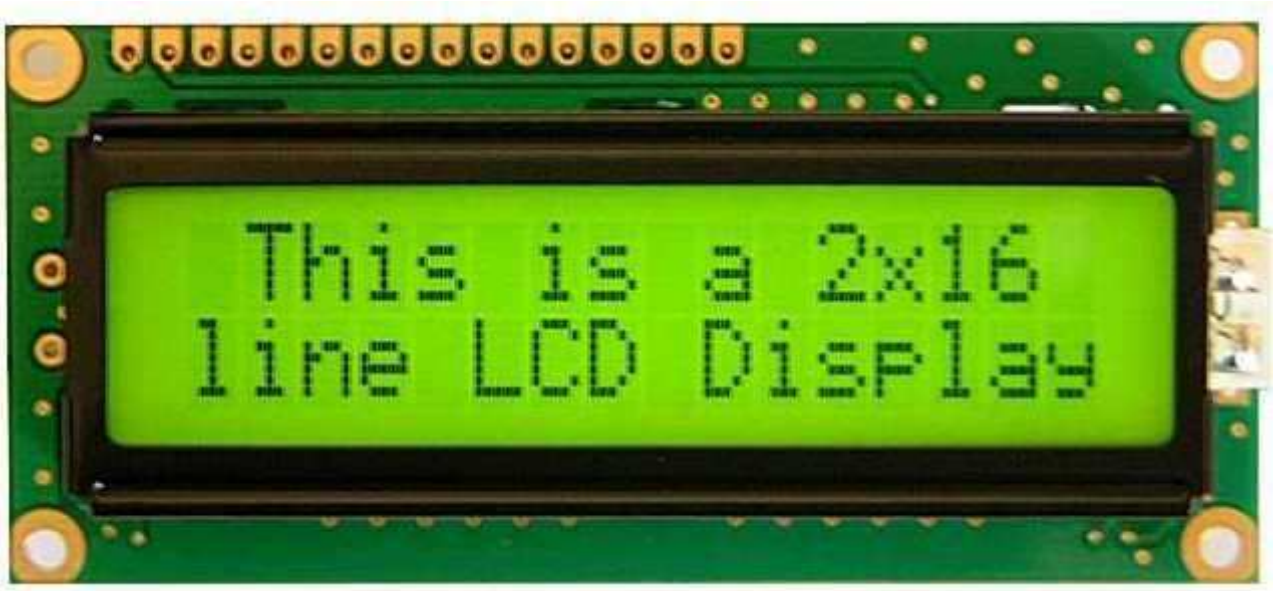


Fig. 9

Main functions:

- Displaying Sensor Values.
- Debugging.
- Creating Embedded C based games.

The basic pin diagram is:

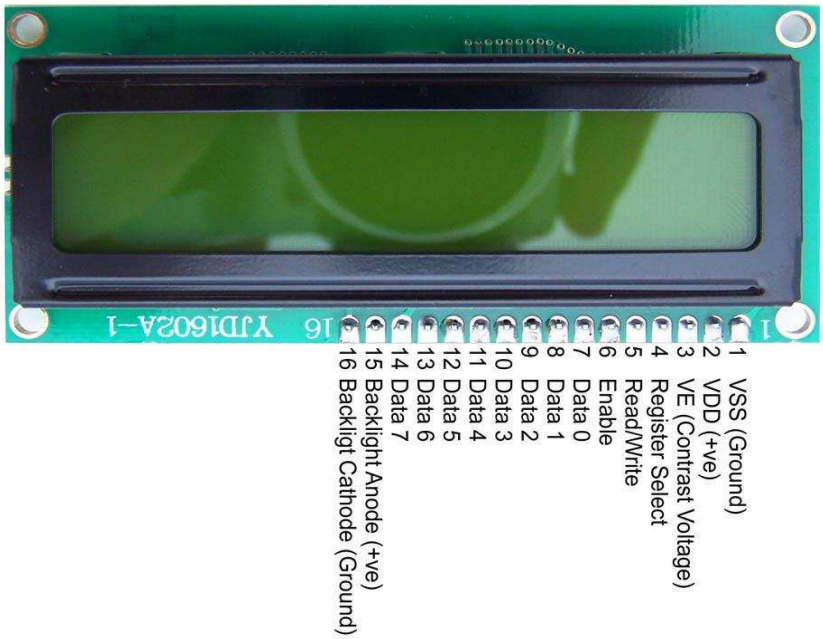


Fig. 10

All the connections of the LCD on the Wall-E board are made to Port B.

The Kicad connections:

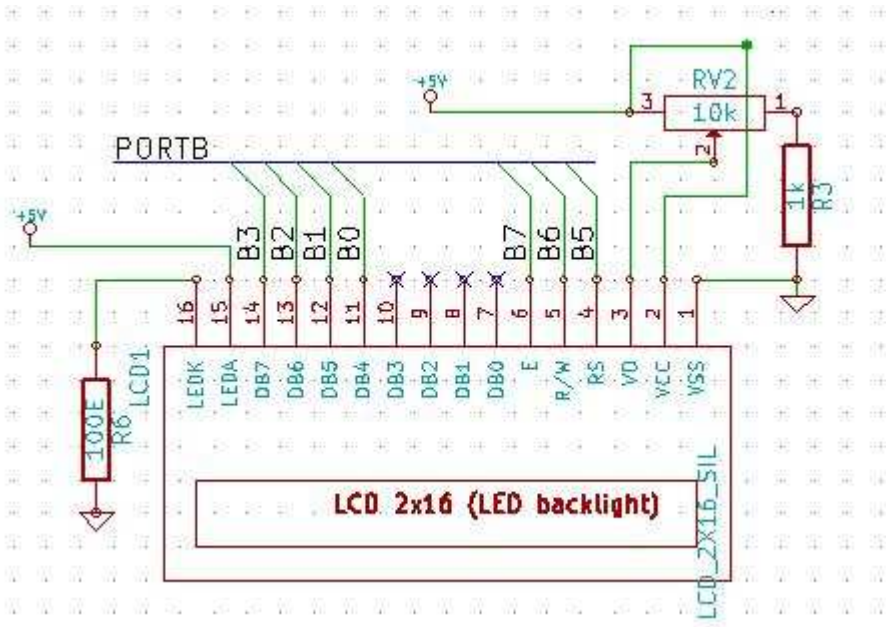


Fig.11

The above information is helpful while debugging. In order to debug, one must also have the knowledge of what each pin does. This can be easily understood with the help of fig.3 and fig.4.

Interfacing of LCD using the SRA development board:

While coding for using the LCD, we must keep in mind that LCD is an external device and has its own IC and Microcontroller. The IC and microcontroller are present below the black circle (at the back of your LCD module) for copyright reasons. Therefore, in order to use LCD there are a bunch of files which we need to include in the code. That part is explained in detail in SRA library.

The most common functions of LCD are:

- **lcd_init(style);**

Initializes the LCD pins. Needs to be called in the main() before using the LCD 'style' can be 'blink' or 'underline' depending upon the cursor you want. SRA Board has a provision for LCD on PORTB

- **lcd_write_string("string");**

Used to write a string of characters on the LCD starting from (0,0).

- **lcd_write_string_xy (intx, inty,"string");**

Used to write a string starting from (x,y).The LCD is a 16x2 character LCD. These 16x2 cells can be considered as a matrix with values from (0,0) to (15.1). (x,y) will take the cursor to that location and write the string upto (15,y)

Once it reaches 15, the remaining string is neglected

- lcd_write_int_xy(intx,inty,intnumber,intfl) Used

to write a number at (x,y) with a field length fl.

Example: 3 with a field length 2 is 03 and with field length 3 is 003 and so on

- lcd_clear();

Clears the LCD. Cursor returns to original

position Additional information:

MECHANICAL DATA		
Item	Nominal Dimensions	Unit
Module Size(W×H×T)	80.0×36.0×12.0	mm
Viewing Area(W×H)	64.5×14.0	mm
Character Size(W×H)	3.00×5.02	mm
Dot Size(W×H)	0.52×0.54	mm

ABSOLUTE MAXIMUM RATINGS					
Item	Symbol	Min	Type	Max	Unit
Operating Voltage	VDD	4.5	5.0	5.5	V
Operating Current	IDD	1.0	1.3	1.5	mA
LED Voltage	V_{LED}	1.5	5.0	5.5	V
LED Current	I_{LED}	75	140	200	mA
Operating Temp.	Topr	-20	—	+70	℃
Storage Temp.	Tsto	-30	—	+80	℃

ELECTRICAL CHARACTERISTICS					
Item	Symbol	Min	Type	Max	Unit
Input Hight Voltage	V_{IH}	2.2	—	VDD	V
Input Low Voltage	V_{IL}	0	—	0.6	V
Output Hight Voltage	V_{OH}	2.4	—	VDD	V
Output Low Voltage	V_{OL}	0	—	0.4	V

PIN CONNECTIONS			
PIN	Symbol	Level	Function
1	VSS	—	GND(0V)
2	VDD	—	Supply Voltage for Logic(+5V)
3	V0	—	Power supply for LCD
4	RS	H/L	H: Data; L: Instruction Code
5	R/W	H/L	H: Read; L: Write
6	E	H/L	Enable Signal
7	DB0	H/L	Data Bus Line
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	BL1	—	Backlight Power(+5V)
16	BL2	—	Backlight Power(0V)

Fig.12

BRUSHING UP CODING!!

WHAT IS A LIBRARY? :-

A library is collection of all the functions and all the macros which are to be used while programming.

Eg: <avr/io>. For using these libraries the particular file must be included in your program by writing them as header files.

HEADER FILES:-

#include<avr/io.h>

This header file contains appropriate IO definitions.

#include<stdlib.h>

This file declares some basic C macros and functions as defined by the ISO standard, plus some AVR-specific extensions.

#include<compat/deprecated.h>

This header file contains several items that used to be available in previous versions of this library, but have eventually been deprecated over time. These items are supplied within that header file for backward compatibility reasons only, so old source code that has been written for previous library versions could easily be maintained until its end of-life. Use of any of these items in new code is strongly discouraged.

#include<util/delay.h>

Contains convenience functions for busy/wait delay loops. The functions available allow the specification of microsecond, and millisecond delays directly.

#include<avr/eeprom.h>

This header file declares the interface to some simple library routines suitable for handling the data EEPROM contained in the AVR microcontrollers.

SOME COMMONLY USED FUNCTIOINS:

TO SET A PORT INPUT OR OUTPUT:-

DDRX=0x00;

By writing DDRX to 0x00 the PORTX becomes an input port and by writing it to 0xFF the port becomes and output port.(Here X is name of the port eg. A,B,C or D)

Eg:-DDRA=0x00;

This means all the PINS of PORTA becomes input.

Similarly if you want to set only two or three pins as input, let say PIN 2 & 3 of PORTD and rest all as output, you can do it by writing

DDRD=0b11110011;

DELAY FUNCTIONS:-

_delay_us(x);

This function gives a delay of x microsecond in the program. Requires argument x and does not return anything

_delay_ms(x);

This function gives a delay of x milliseconds in the program. Requires argument x and does not return anything.

TO MAKE PIN HIGH OR LOW:-

sbi(PORTX,y)

Where X can be either A,B,C or D and y denotes the PIN no.

-->This function sets the pin HIGH.

cbi(PORTX,y)

Where X can be either A,B,C or D and y denotes the PIN no.
-->This function sets the pin LOW.

TO CHECK WHETHER PIN IS HIGH OR LOW:-

bit_is_set(PORTX,y)

Here X can be A,B,C or D and y denotes the PIN no. This function returns 1 if the particular is HIGH and 0 if the particular pin is LOW

bit_is_clear(PORTX,y)

Here X can be A,B,C or D and y denotes the PIN no. This function returns 1 if the particular is LOW and 0 if the particular pin is HIGH.

USBASP Programmer

Introduction

USBasp is a USB in-circuit programmer for Atmel AVR controllers. It simply consists of an ATMega8 and a few passive components. The programmer uses a firmware-only USB driver; no special USB controller is needed.

Some of the key features include:

a. Works under multiple platforms. Linux, Mac OS X and Windows are tested,

b. Programming speed is up to 5kBytes/sec, and

c. SCK option to support targets with low clock speed (< 1,5MHz).

Layout

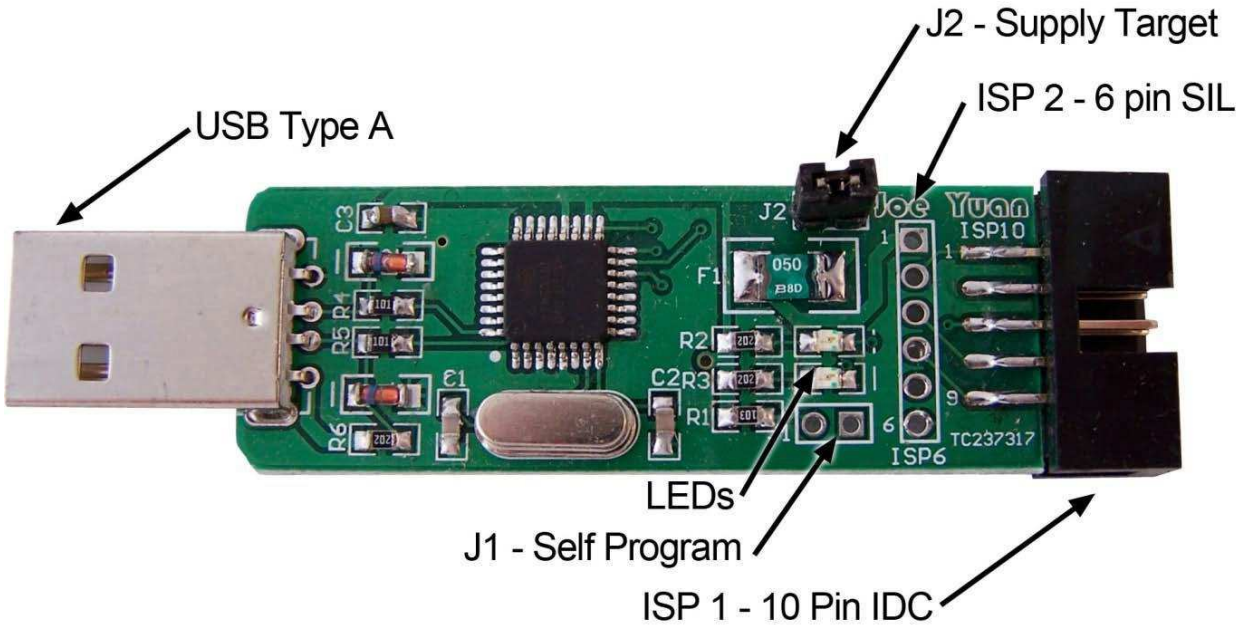


Fig. 13

USB Type A

The USB end of the programmer connects directly into your computers USB port.

2.2. ISP1 – 10 pin IDC

The 10 pin ISP connection provides an interface to the microcontroller. This interface uses a 10 pin IDC connector and the pinout is shown in Figure 2

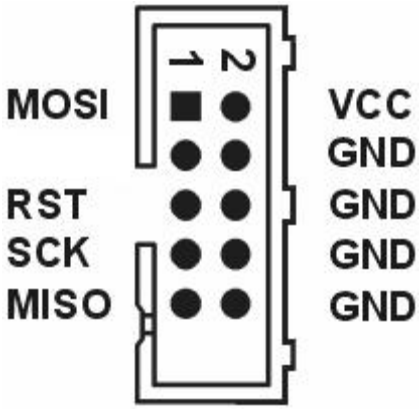


Fig.14

LEDs

The USBASP programmer has 2 LEDs near the ISP2 connection. These have the following functions: a. LED 1 – Power

b. LED 2 – Programmer communicating with target device

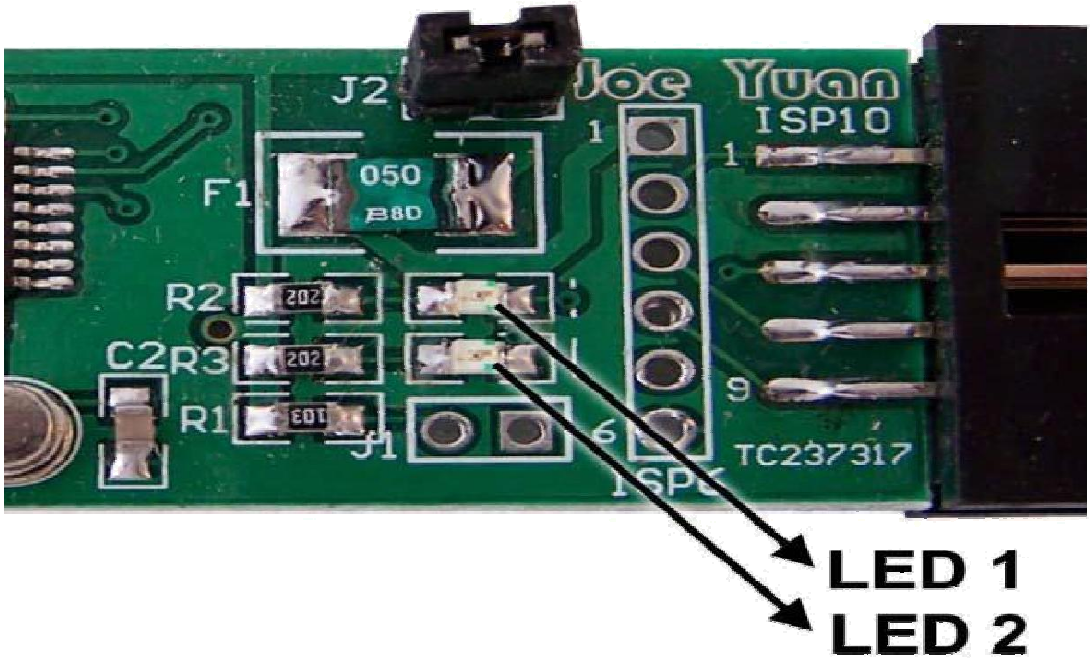


Fig. 15

Using the Programmer

3.1. Connecting the programmer to your computer

Connecting the programmer to your computer comprises of 2 steps:

a. Physically connecting the programmer to the USB port, and

b. Installing drivers in order for it to work.

Whilst the USBasp programmer will work on a wide variety of operating systems, this procedure will focus on Widows Vista 32 bit and Windows XPs

3.1.1. Windows Vista (32 Bit)

3.1.1.1. Required items

Items required to run this procedure are: a. USBasp programmer

b. Computer with USB port and Windows Vista 32 bit installed

c. USBasp drivers downloaded and unzipped from http://www.protostack.com/download/USBasp-driver-0.1.12.1.zip

3.1.1.2. Assumptions

This procedure assumes that: a. The logged in user has sufficient permissions to install unsigned device drivers

3.1.1.3. Procedure

To install the USBasp programmer: a. Insert the programmer into an available USB port

b. When the “Found New Hardware” dialog opens, select “Locate and install driver sotware (recommended)”

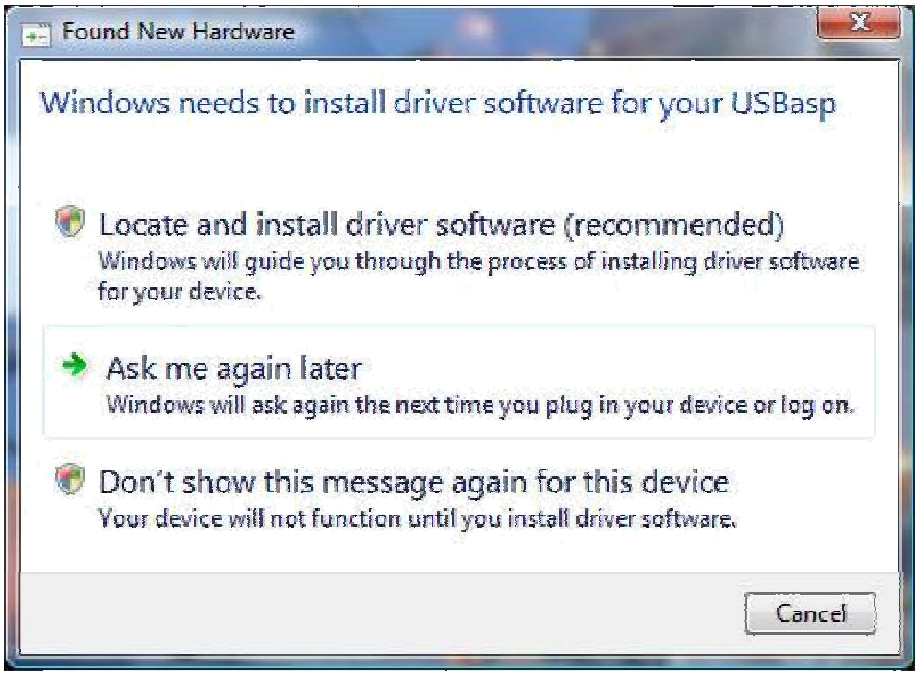


Fig. 15

Driver Installation on Vista 32 bit – Found new hardware

c. Wait while Windows Vista attempts to locate a driver

d. When the "Found New Hardware – USBasp" dialog box is displayed, select "I don't have the disc. Show me other options"

Fig.16

Driver Installation on Vista 32 bit - Found New Hardware – USBasp

e. On the next screen select “Browse my computer for driver software (advanced)”

Fig.17

Driver Installation on Vista 32 bit – Windows couldn’t find driver software for your device

f. Click Browse and select the folder where you unzipped the USBasp drivers, then click Next

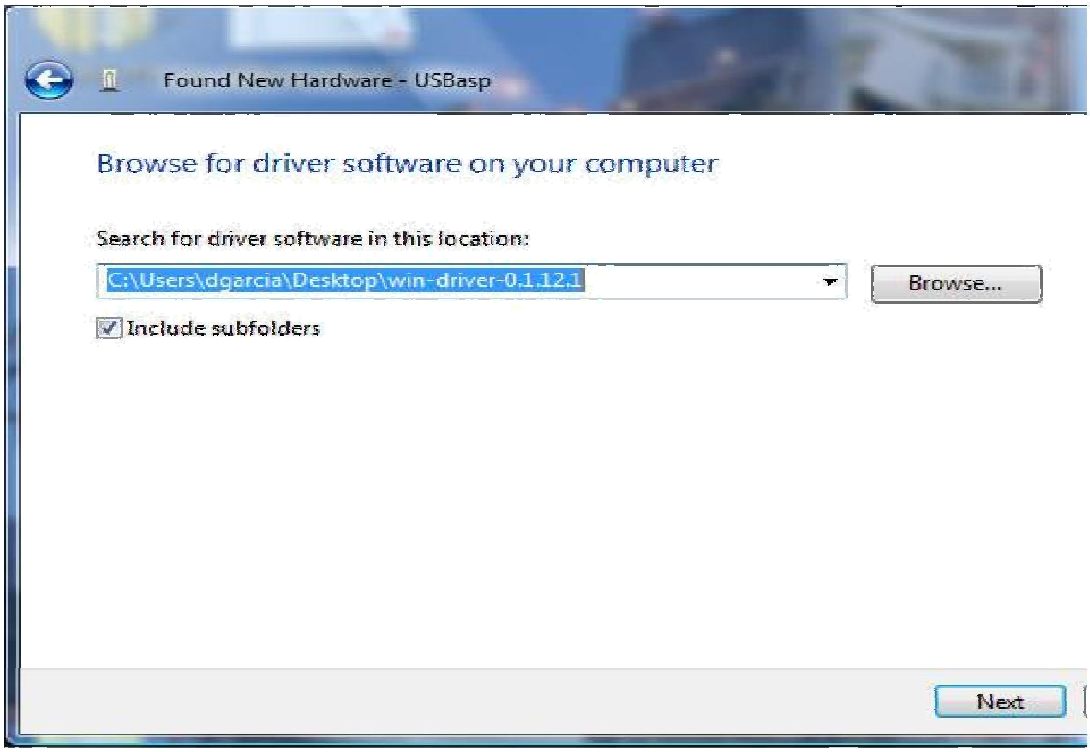


Fig. 18

Driver Installation on Vista 32 bit – Browse for driver software

g. When the windows security dialog box is opened, select "Install this driver software anyway". *Note: This security warning is raised because the device driver files are not signed with a digital certificate. This does not mean that the file will cause a security problem, but rather that windows cannot guarantee it source. Click on "see details" for more information.*

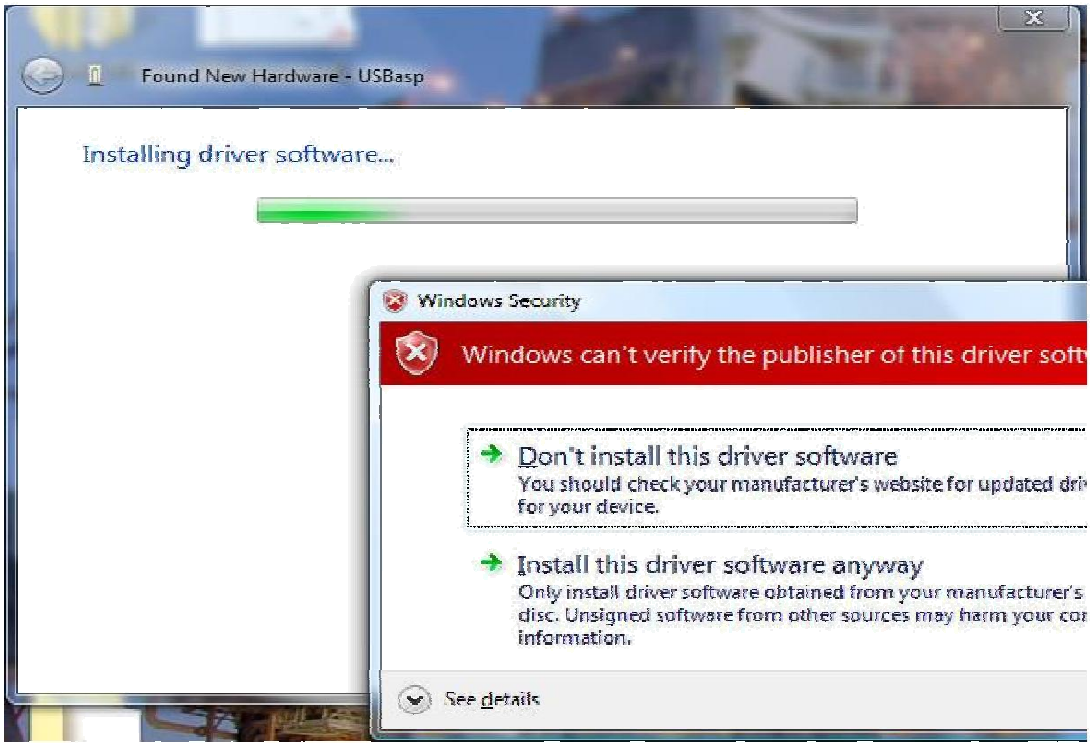


Fig. 19

Installation on Vista 32 bit – Security warning

h. When the installation is complete, a confirmation screen will be displayed. Click close to close it.

Fig.20 Installation on Vista 32 bit – Confirmation

Your programmer is now ready for use

3.1.2. Windows XP (32 bit)

3.1.2.1. Required items

Items required to run this procedure are: a. USBasp programmer

b. Computer with USB port and Windows XP 32 bit installed

c. USBasp drivers downloaded and unzipped from http://www.protostack.com/download/USBasp-driver-0.1.12.1.zip

3.1.2.2. Assumptions

This procedure assumes that: a. The logged in user has sufficient permissions to install unsigned device drivers

3.1.2.3. Procedure

To install the USBasp programmer: a. Insert the programmer into an available USB port

b. When the “New Hardware Wizard” dialog box is displayed, select “No, not this time” then click Next

Installation on Windows XP – New Hardware Wizard

c. On the next page select “Install from a list of specific location (Advanced)” then click Next

Figure 14. Installation on Windows XP – Insert CD or install from specific location

d. On the Search and Installation options page

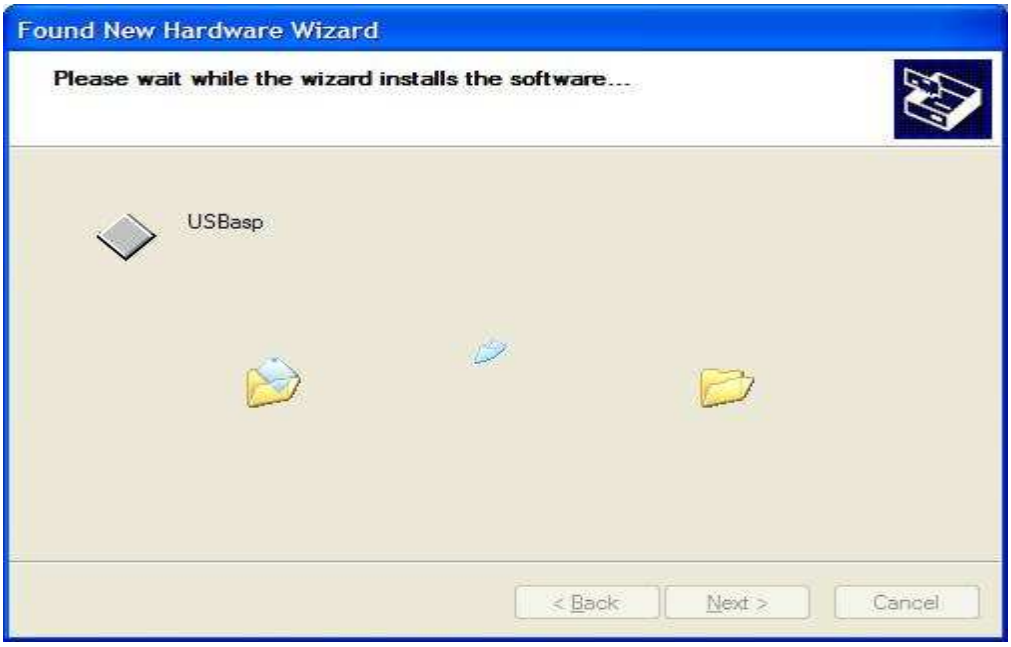
(1) Ensure that “Include this location in the search” is checked,

(2) Click Browse and select the folder where you unzipped the USBasp drivers, then

(3) Click Next

Installation on Windows XP – Specify Location

e. Wait for the driver to install



Installation on Windows XP – Driver installation

f. When the installation is complete, a confirmation screen will be displayed. Click close to close it.

Installation on Windows XP – Installation Confirmation

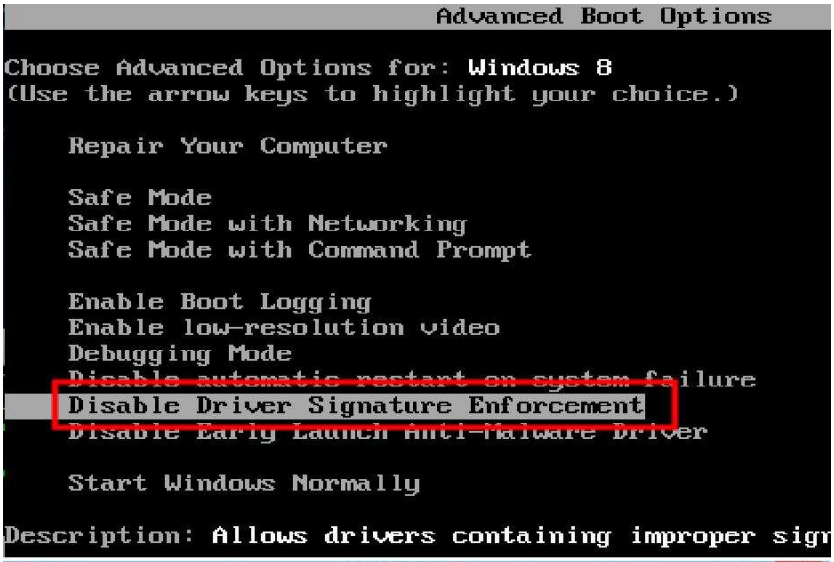
g. Your programmer is now ready for use

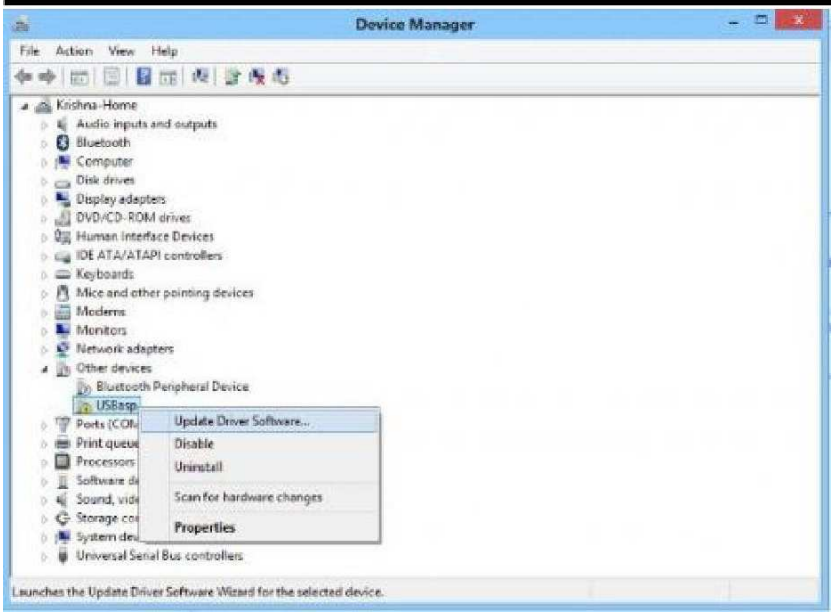
Steps to install USBASP in Windows 8

Actually the main problem is that Microsoft doesn't allow you to install drivers which are not 'digitally signed' directly (or easily) on windows vista and later versions (7 and 8). To get around this you must temporarily disable the signature verification process. So, here is a way through which you can do this-

1. Move the cursor over the top or bottom right corner or the screen and it will show extra options on the right side of the screen. Choose the Settings option (icon looks like a Gear).
2. After this chose 'Change PC Settings'.
3. Choose General in the left menu and click on the 'Restart Now' in the bottom on the right side.
4. Select troubleshoot.
5. Then select 'Advanced options' and then 'Startup Settings' and click restart to select option like, 'Disable Driver Signature Enforcement'.
6. You will see a black MS-DOS screen, choose the option 'Disable Driver Signature Enforcement' and press enter.
7. Now you have successfully disabled the driver signature verification and you can install whatever you want!
8. Now connect your USBasp programmer to the USB port of your PC/laptop. Ignore the message that 'Device driver software was not successfully installed'.
9. Download USBasp drivers and extract it somewhere on your PC/Laptop.

10. Go to Device Manager in control Panel and you will find 'LibUSB-Win32 Devices'. Click on it and select 'Update Driver Software...'





11. Select 'Browse my computer for driver software'. After this browse to the location where you have extracted the USBasp drivers.

12. Click next and ignore the security warning 'Windows can't verify the publisher of this driver software' and select 'Install this driver software anyway'

13. You will get a completion note after this and you are ready to use your programmer (Fig. 20).

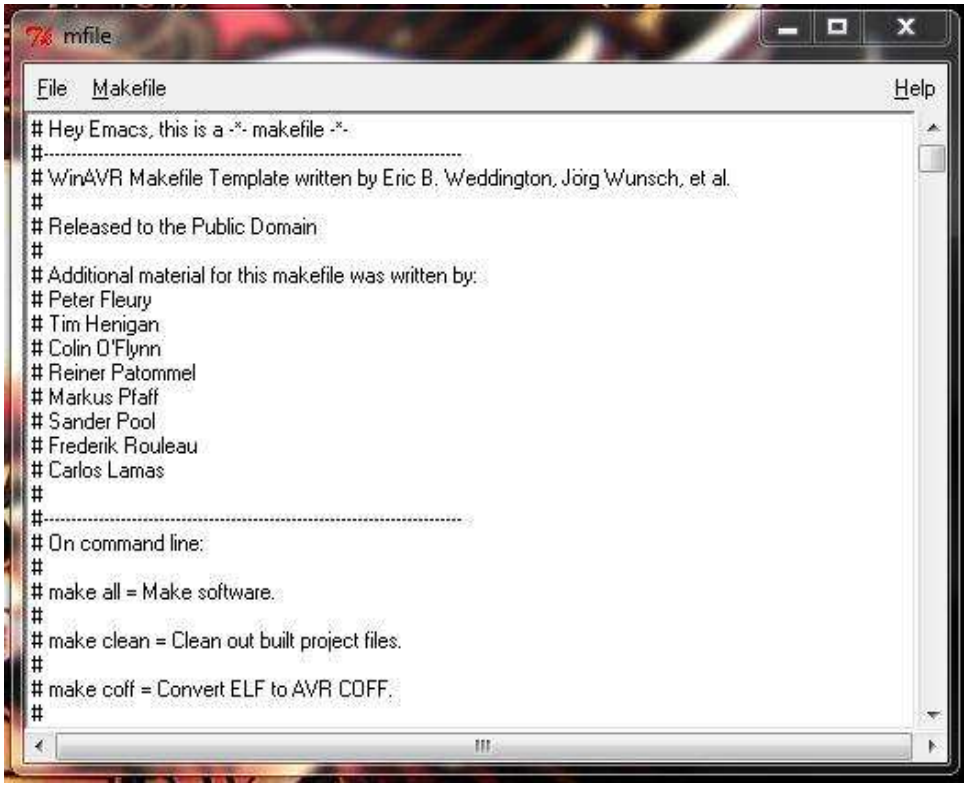
How to make a Make-File?

Make file is essential to program the micro-controller with the created code. The compiler compiles the code written in 'Embedded-C' language into low level 'Hex' format which the controller understands. However one must specify which controller is going to be used, its working frequency, and the name of the file where the code is written etc. before it is programmed into the micro-controller. All this is done by the 'make file'.

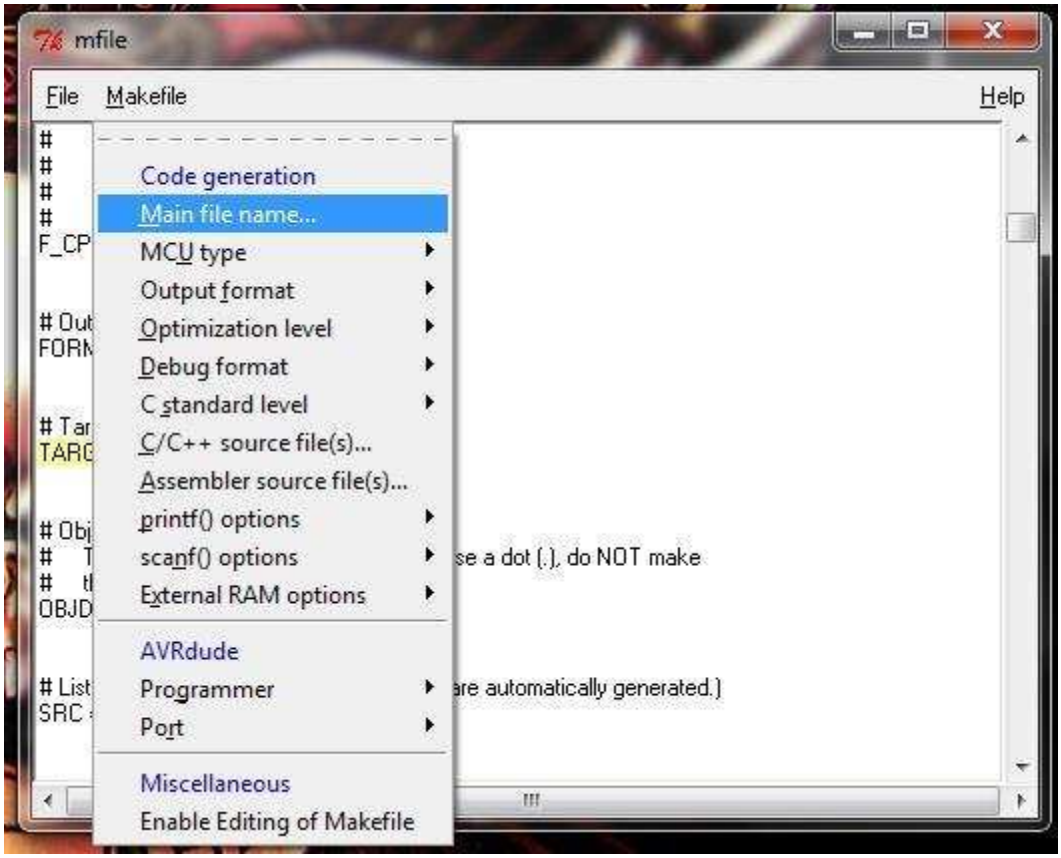
After installing 'Winavr' an option of 'Mfile' is created in the start menu.

To create the makefile follow these steps-

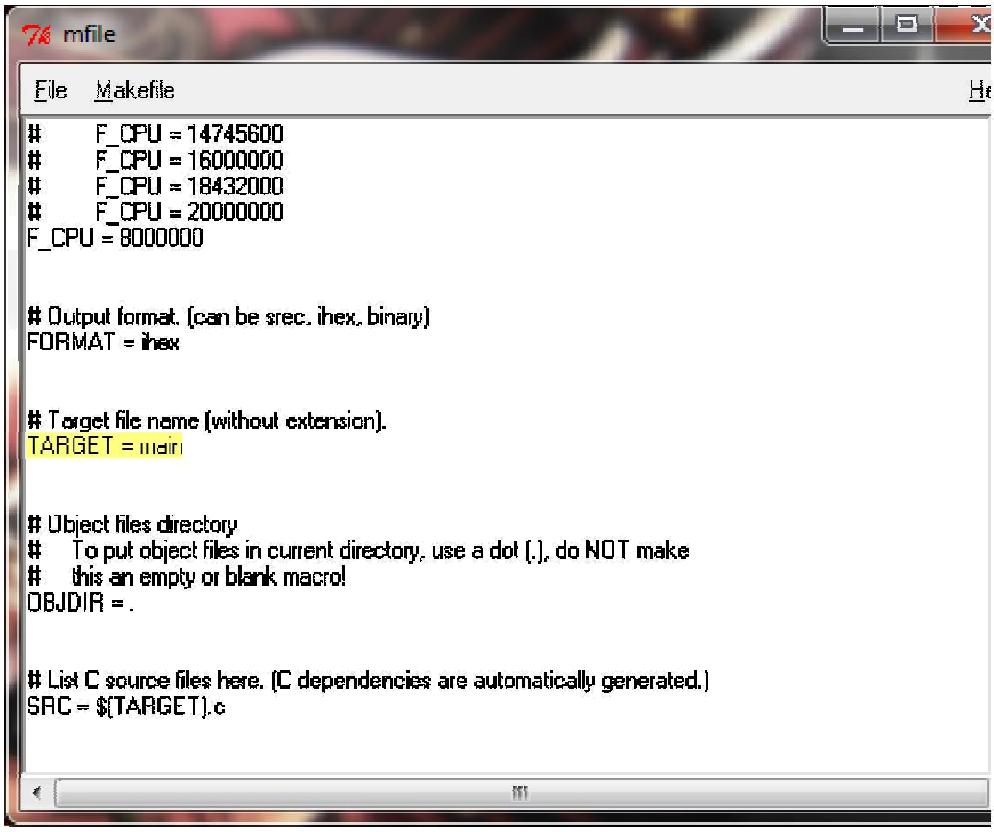
1. Double click it to open and create a makefile for your program.



2. Click on Makefile option.

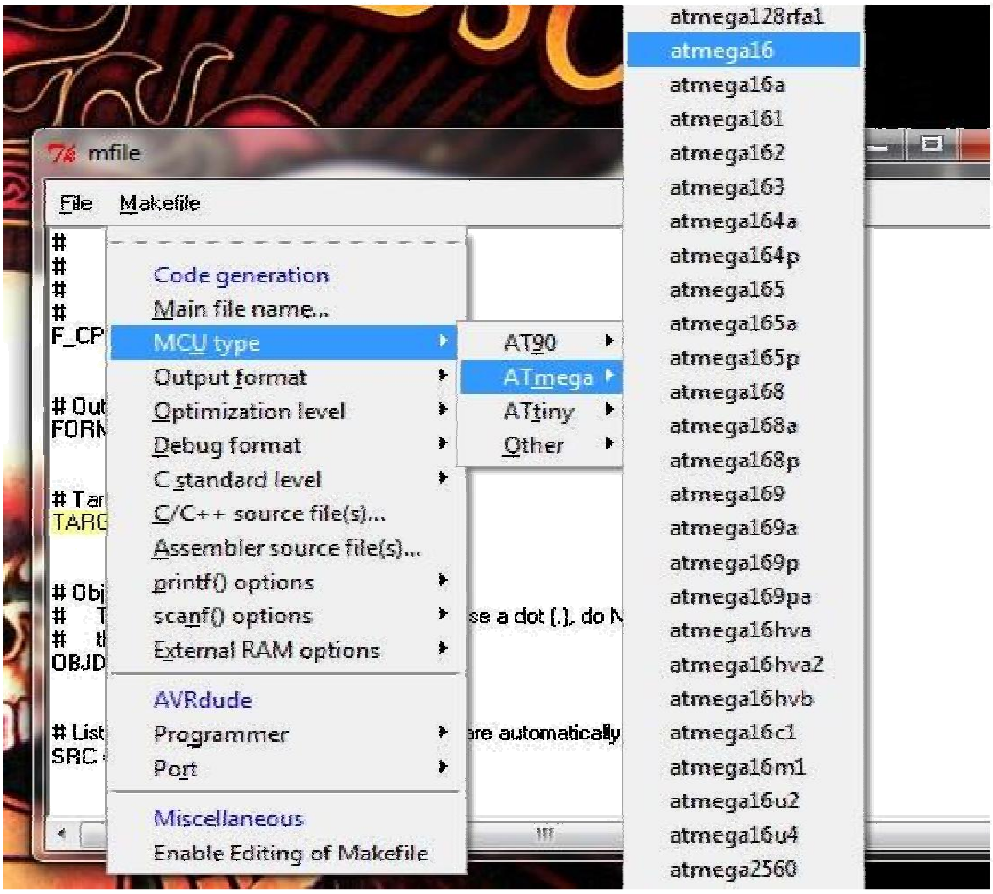


3. Click on **‘Main file name’**- It asks for the name of the file. This file name is the same as the name of the program file which you have created in the ‘Programmer’s notepad’. Write the same



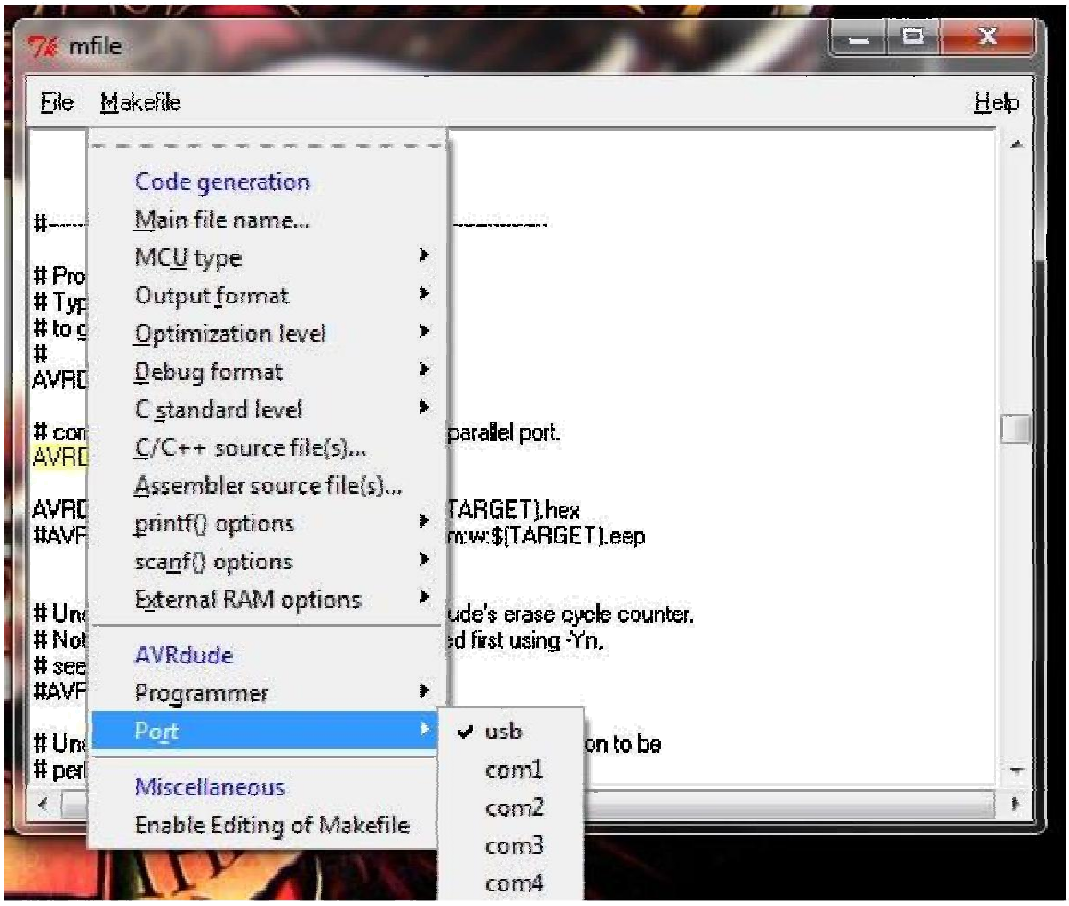
of the program file without any extension. After writing the name, the dialog box highlights **'TARGET=main'** where main is the name of the program file which is already created in programmers notepad.

4. Click on Makefile option and select the correct **'MCU type'.** You must specify the correct micro-controller in order the burn the program in it. Here select **Atmega 16**.



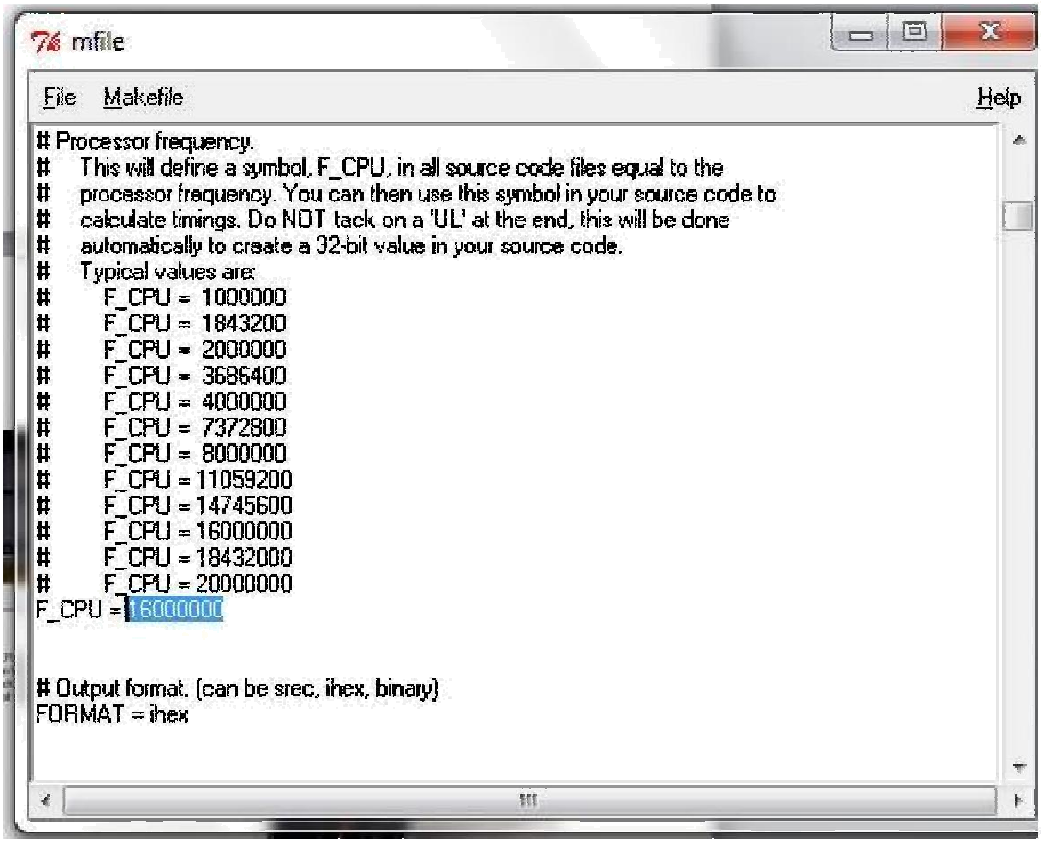
5. Click on Makefile option then go to programmer option. We must also specify the type of programmer we are using. We use ‘usbasp’ programmer to program the code into the controller. However this option is not given in the programmers list, hence select any programmer of your choice.

6. Click on Makefile option and go to **‘port’**. We must also specify the port of the computer which we are using to program the controller. Usbasp is connected to ‘usb port’ hence select ‘**usb**’ option.



NOTE- The micro-controller we are using is Atmega 16 which works on external 16MHz clock frequency. However the default frequency specified in the makefile is 8MHz. Hence we need to change this default 8MHz frequency to 16MHz.

7. For this Click on Makefileoption then select 'Enable Editing of Makefile' to edit and change the default frequency. Now in the dialog box search for 'processor frequency'. There the last option says '**F_CPU=8000000'** which means the default frequency selected is 8MHz. Change this frequency to **'F_CPU=16000000'**.



Now we are almost done making the makefile and ready to program the code in the controller.

8. Save the makefile in the same folder where you have saved your code file. A file with name ‘Makefile’ is created in that folder which will burn the code compiled in programmer’s notepad to the microcontroller.

Acronyms and Abbreviations

Acronym and Abbreviation	Description
AVR	According to Atmel, AVR stands for nothing, it's just a name. Others say it stands for Advanced Virtual RISC. However, the inventors of the AVR series chips are named Alf EgilBogen and VegardWollan, so you be the judge.
IDC	Insulation Displacement Connector
ISP	In System Programmer
LED	Light Emitting Diode
RISC	Reduced Instruction Set Computing
SCK	Slow Clock
SIL	Single in Line
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

SRA Library Files

The SRA library is made specifically for the connections available on the SRA Development Board and the features used.

The main purpose of making this library is to hide the

"SCARY" code from the person new to the world of embedded programming!!!

The library includes various functions used for ADC, PWM, LCD Operations, USART, Moving the Robot, Blinking the LEDs etc. All these functions have been included in a different header file. We will explore each of the header file and the functions included in it here.

---

sra/basic.h

This header file contains the very basic initializations for the ATMega. This file must be included in all the programs. The following functions are a part of this header file.

port_init()

This function initializes all the PORTS of ATMega 16. (Fig.4)

PORT A is declared as input-high

PORT B is declared as input-high

PORT C is declared as output-low

PORT D is declared as input-high

delay_sec(int x): Gives a time delay of 'x' seconds.

delay_millisec(int x): Gives a time delay of ‘x’ milliseconds.

delay_microsec(int x): Gives a time delay of ‘x’ microseconds.

flick(): A library function to blink the LEDs. Used mainly for testing and checking the reach of code.

---

sra/adc.h

This header contains two functions that are used in Analog to Digital conversion of data.

adc_init()

Initializes the ADC mechanism of the microcontroller**.**Needs to be called in the main() for using the ADC feature. Requires no arguments and does not return anything. ADC is mostly required when a sensor is used to obtain data from the real world

adc_start(unsigned char channel)

This function is used to calculate the ADC value on a particular pin number of the PORTA (Fig.4).The argument that we need to pass is the pin number of PORTA which is also known as channel. Returns the ADC value of the pin number passed in the argument.

sra/pwm.h

pwm1_init()

Initializes the pwm1 feature of the microcontroller.PWM is Pulse Width Modulation which is used to obtain a voltage between 0 and 5V.It can be considered as a reverse process of ADC .This function needs to be called in the main in order to use the PWM1 feature .On the SRA Board PWM pins are connected to the enable pins of motor drivers which enable us to vary the motor speeds.

set_pwm1a(int a) &set_pwm1b(int a)

These functions will set the OCR values to ‘a’. The obtained voltage is calculated by:

$$\frac{\text{_____}}{\text{_____}}$$

These functions are mainly used to change the motor speed by the technique of PWM.

sra/lcd.h

lcd_init(style)

Initializes the LCD pins (Fig. 10) . Needs to be called in the main() before using the LCD ‘style’ can be ‘blink’ or ‘underline’ depending upon the cursor you want SRA Board has a provision for LCD on PORTB.

lcd_write_string(“string”)

Used to write a string of characters on the LCD starting from (0,0).

lcd_write_string_xy(intx,inty,”string”)

Used to write a string starting from (x,y). The LCD is a 16x2 character LCD. These 16x2 cells can be considered as a matrix with values from (0,0) to (15.1). (x,y) will take the cursor to that location and write the string up to (15,y) . Once it reaches 15, the remaining string is neglected

lcd_write_int_xy(intx,inty,intnumber,intfl)

Used to write a number at (x,y) with a field length fl.Example: 3 with a field length 2 is 03 and with field length 3 is 003 and so on

lcd_clear(): Clears the LCD.Cursor returns to original position

sra/botmotion.h

bot_motion_init()

Initializes the motion. Needs to be called to make the bot move. PORTC 4,5,6,7 (Refer fig.4 & fig. 8) are used for the motion. The positive terminal of left motor should be connected to the C4 pin output and positive terminal of right motor should be connected to C6 pin output

bot_left_forward() : Makes the left wheel of the robot move forward.

bot_left_backward() : Makes the left wheel of the robot move backward.

bot_right_forward(): Makes the right wheel of the robot move forward.

bot_right_backward() : Makes the right wheel of the robot move backward

bot_forward(): Both wheels move in the forward direction.

bot_backward(): Both the wheels move in the reverse direction

bot_left(): The right wheel moves forward making the robot go left.

bot_right(): The left wheel moves forward making the robot go right.

bot_stop(): This function is used to stop the motion of the robot

bot_brake(): This function is used to brake(lock the motors) of the robot

---

sra/switch.h

---

switch_init(): Initializes the switches. Needs to be called to use the switches

pressed_switch0()

pressed_switch1()

pressed_switch2()

pressed_switch3()

These functions return 1 or 0 if the switch is pressed or not. Switches are mainly used to perform a task after an input from the user

---