INTRODUCTION

Computer numerical control is an advanced form of soft automation developed to control the motion and operation of machine tools. Numerical control machine was invented around in 19th century to reduce work load, it is a method in which the manufacturing machine uses coded format, digits and letters. Its advantages include high efficiency, flexibility and high production rate, low cost of production, less working time and less losses in production. It includes three main steps that is receiving data, interpreting data and accordingly control action. The term “CNC” is a generic term which can be used to describe many types of device, this would include plotters, vinyl cutters, 3D printers, milling machines and others. CNC stands for Computer Numerically Controlled and basically means that the physical movements of the machine are controlled by instructions, such as co-ordinate positions that are generated using a computer. The term “CNC Machine” is typically used to refer to a device which uses a rotating cutting tool which moves in 3 or more axes (X, Y and Z) to cut-out or carve parts in different types of materials.

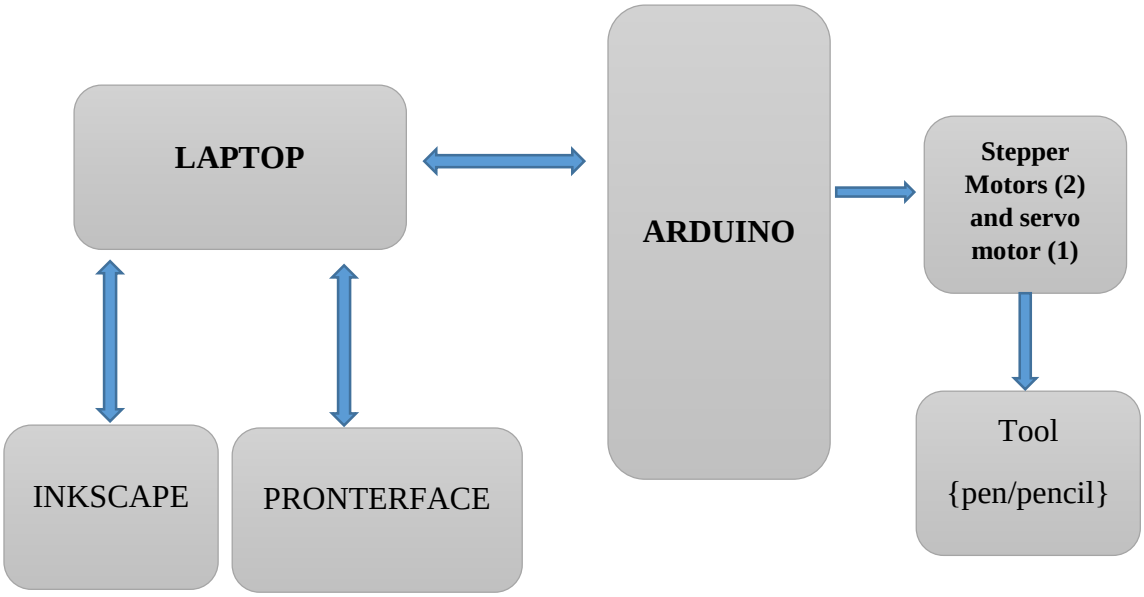
A plotter works closely with a computer's imaging software to produce a final picture or object. The first step in using a plotter is to enter the appropriate coordinates for where you want the image to appear on the paper. Modern software allows the user to accomplish this goal very easily by drawing lines and images with the imaging software. Once the schematics for the image are complete, the computer downloads the coordinates to the plotter, which interprets the code and calculates the most efficient path for the pen and paper.

A machine control unit (MCU) decides the tool depth of cut, cutting speed etc. Motion of tool is based on Right hand coordinate system. Three axis of rotation x, y, z for three dimensional motion of tool plus an axis of rotation. The z-axis is one of the three which allows the movement of router in up and down direction. This axis is very important because it controls the depth. The y-axis functions as motor mount to move z-axis in addition with slide mechanism, x-axis uses two pieces one for front and one for back which serves as height stands.

MOTIVATION

Computer Numeric Control (CNC) refers to a wide variety of machines which are controlled electronically and have many uses, including milling, drawing, extruding, cutting, and lathing. CNC machines are really expensive. They are widely used in the fabrication of both electronic and mechanical parts of large machines .So our group has decided to do a model to know about theoretical and practical knowledge about this concept [2D Robotic Plotter].

BLOCK DIAGRAM AND EXPLANATION



Stepper Motor- It is the heart of CNC plotter. The size and type of motor, speed, accuracy, CNC router precision etc. mainly two types of motors are used in CNC machines they are stepper motors and servo motors, within these are also many classifications.

Arduino Microcontroller- It controls the position of stepper motor with help of program. It is an open source prototyping platform based on easy to use hardware and software. They have digital and analog input/output pins that can interface into various expansion boards and other circuits and an Atmel 8, 16 or 32-kbit memory (flash) AVR microcontroller with complementary components that helps in programming and incorporation into other circuits. Arduino programs are written in any programming language with a compiler that produces binary machine code. Here we are using Arduino Uno for controlling process.

MODULE 1: Hardware Section

Hardware

Electronic hardware consists of interconnected electronic components which perform analog or logic operations on received and locally stored information to produce as output or store resulting new information or to provide control for output actuator mechanisms. Electronic hardware can range from individual chips/circuits to distributed information processing systems. Well-designed electronic hardware is composed of hierarchies of functional modules which inter-communicate via precisely defined interfaces. The XY-plotter consists of two axes operating orthogonally to each other. Each axis includes a CD drive system that is driven by an appropriate means. Additionally, a third axis, with limited motion capability is used to actuate the write head.

ARDUINO

Introduction

Arduino or Genuino (outside US) is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The project Arduino is based on microcontroller board designs, produced by several vendors, using various microcontrollers. The Arduino provide sets of digital and analog Input/output (I/O) pins that can interface to various expansion boards (termed *shields*) and other circuits. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

The first Arduino was introduced in 2005, aiming to provide a low cost, easy way for novices and professionals to create devices that interact with their environment

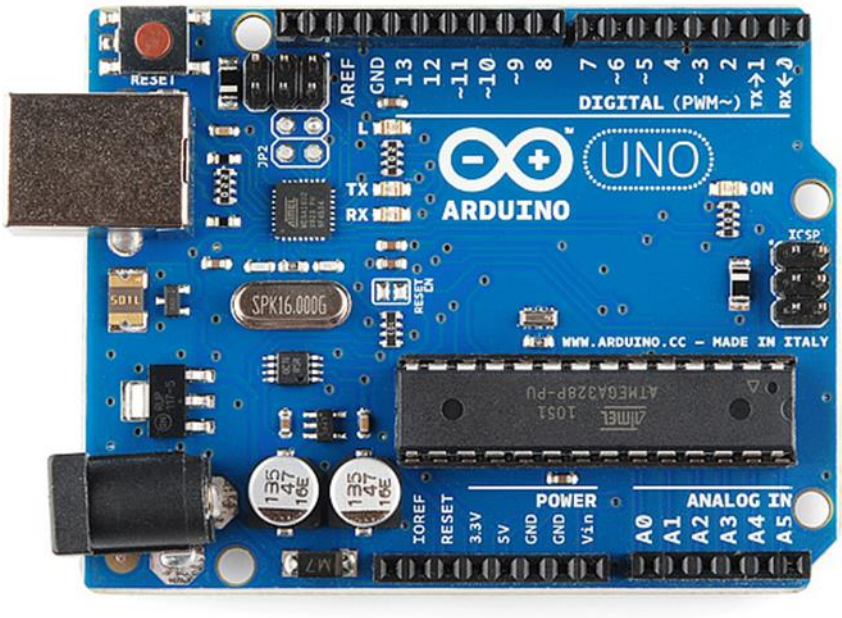
using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots, thermostats, and motion detectors.

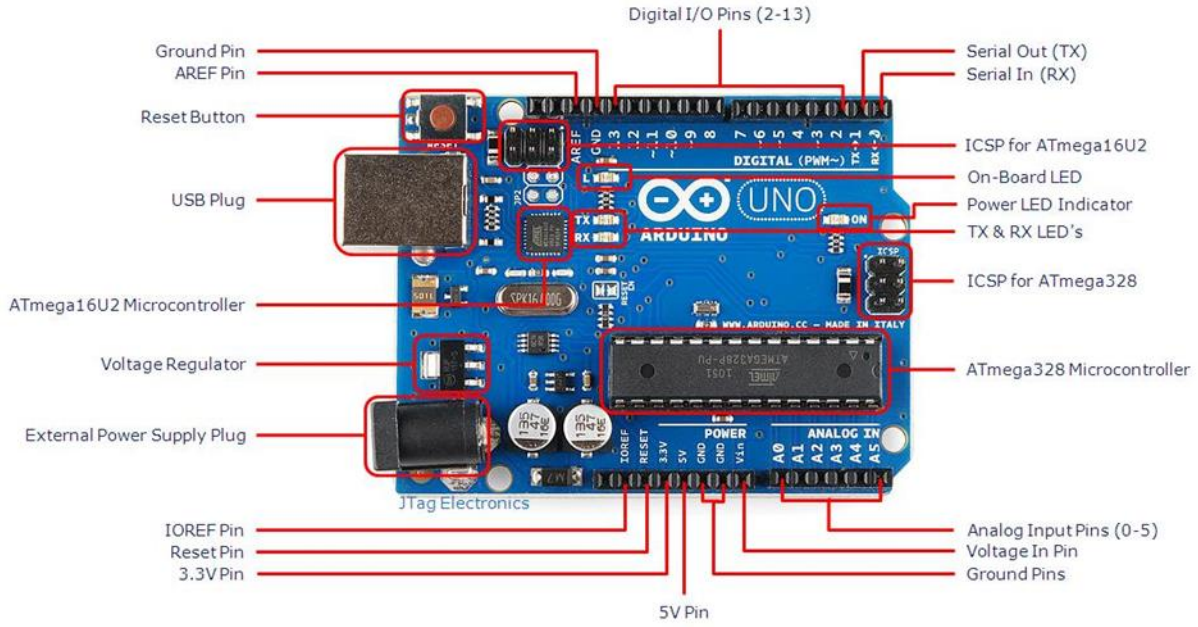
Hardware

Arduino/Genuino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform.

The microcontroller in Arduino Uno is pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external programmer. This makes using an Arduino more straightforward by allowing the use of an ordinary computer as the programmer.





Technical Specifications

Microcontroller	Atmega328P
Operating Voltage	5V
Input Voltage (Recommended)	7-12V
Digital I/O Pins	14 (of which 6 provide PWM Output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
Size and Weight	68.6 x 53.4 mm / 25 g

Power (USB / Barrel Jack)

The Arduino/Genuino Uno board can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the

board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

The board can operate on an external supply from 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **Vin**: The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V**: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board.
- **3V3**: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND**: Ground pins. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- **IOREF:** This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.

In addition, some pins have specialized functions:

- **Serial**: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts**: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

- **PWM**: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function. These pins act as normal digital pins, but can also be used for Pulse-Width Modulation (PWM).
- **SPI**: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- **LED**: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI**: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

The Uno has 6 analog inputs, labelled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference () function.

There are a couple of other pins on the board:

- **AREF**: Reference voltage for the analog inputs. Used with analogReference (). It is used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.
- **Reset:** Pushing it will temporarily connect the reset pin of microcontroller to ground and restart any code that is loaded on the Arduino.

Memory

There are three pools of memory in Arduino Uno:

- Flash memory (program space), is where the Arduino sketch is stored.
- SRAM (static random access memory) is where the sketch creates and manipulates variables when it runs.
- EEPROM is memory space that programmers can use to store long-term information.

The ATmega328 has 32 KB (with 0.5 KB occupied by the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Communication

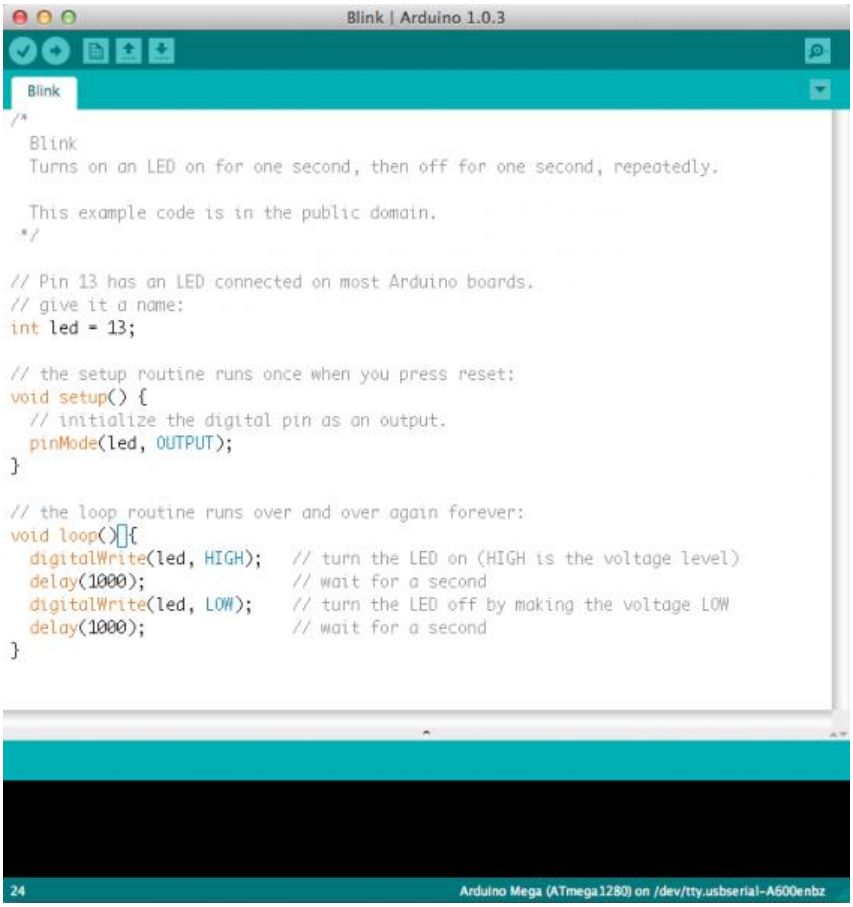
Arduino/Genuino Uno has a number of facilities for communicating with a computer. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board.

The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A Software Serial library allows serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino Software (IDE) includes a Wire library to simplify use of the I2C bus.

Software

The Arduino project provides the Arduino integrated development environment (IDE), which is a cross-platform application written in the programming language Java. It originated from the IDE for the languages Processing and Wiring.



Arduino IDE.

A program written with the IDE for Arduino is called a "sketch". The Arduino IDE supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consists of two functions that are compiled and linked with a program stub main () into an executable cyclic executive program:

setup (): a function that runs once at the start of a program and that can initialize settings.

loop (): a function called repeatedly until the board powers off.

After compiling and linking with the GNU tool chain, also included with the IDE distribution, the Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

Advantages of Arduino

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms.
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well.
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

CONCLUSION

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The project Arduino is based on microcontroller board designs, produced by several vendors, using various microcontrollers. The Arduino provide sets of digital and analog input/output (I/O) pins that can interface to various expansion boards (termed *shields*) and other circuits. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

ATMEGA328P Microcontroller

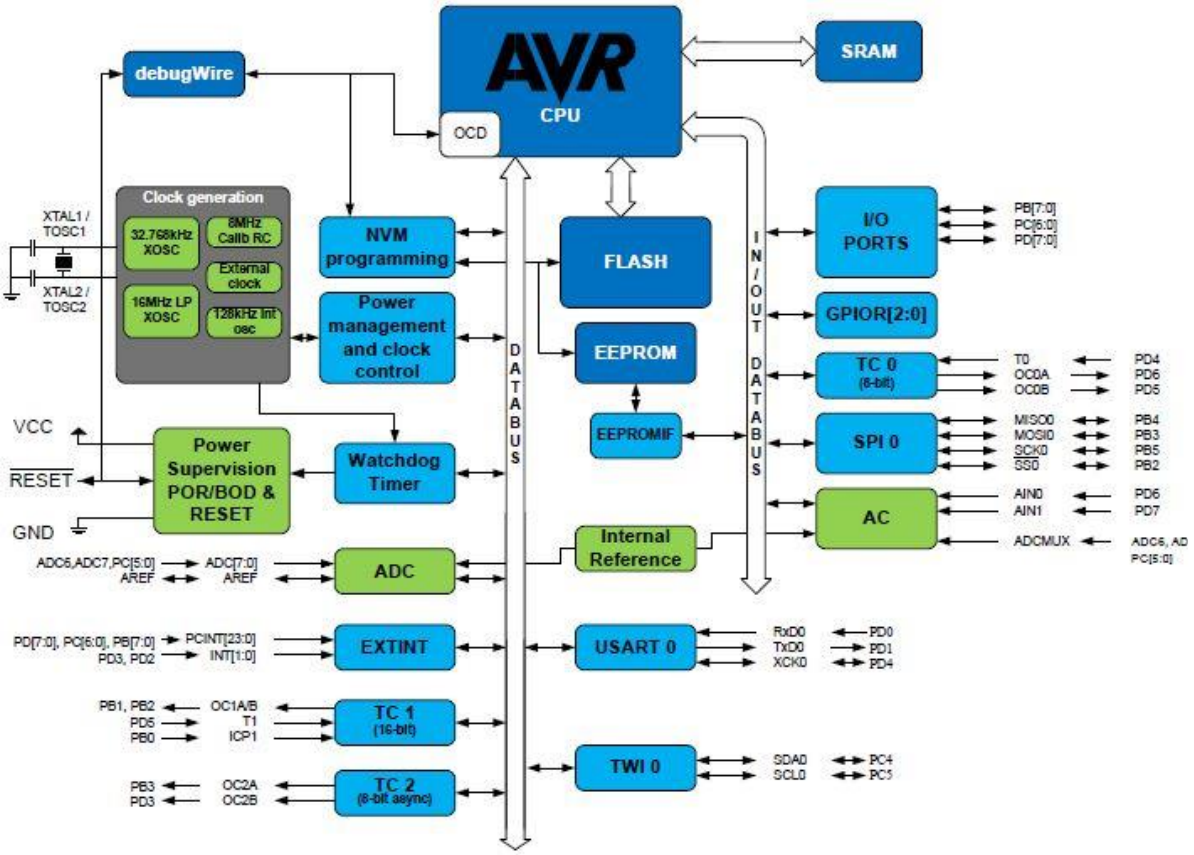
The Atmel AVR® core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers. The ATmega328/P provides the following features: 32Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 1Kbytes EEPROM, 2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 1 serial programmable USARTs , 1 byte-oriented 2-wire Serial Interface (I2C), a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages) , a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main oscillator and the asynchronous timer continue to run. Atmel offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS™) technology for unambiguous detection of key events. The easy-to-use QTouch Suite tool chain allows you to explore, develop and debug your own touch applications. The device is manufactured using Atmel’s high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core.

The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328/P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications. The ATmega328/P is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

Configuration Summary

Features	ATmega328/P
Pin Count	28/32
Flash (Bytes)	32K
SRAM (Bytes)	2K
EEPROM (Bytes)	1K
General Purpose I/O Lines	23
SPI	2
TWI (I^2C)	1
USART	1
ADC	10-bit 15kSPS
ADC Channels	8
8-bit Timer/Counters	2
16-bit Timer/Counters	1

Block Diagram



The Atmel® picoPower® ATmega328/P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328/P achieves throughputs close to 1MIPS per MHz. This empowers system designer to optimize the device for power consumption versus processing speed.

Features

High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family

• Advanced RISC Architecture

– 131 Powerful Instructions

– Most Single Clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 20 MIPS Throughput at 20MHz

– On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory Segments

– 32KBytes of In-System Self-Programmable Flash program

Memory

– 1KBytes EEPROM

– 2KBytes Internal SRAM

– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM

– Data Retention: 20 years at 85°C/100 years at 25°C(1)

– Optional Boot Code Section with Independent Lock Bits

• In-System Programming by On-chip Boot Program

• True Read-While-Write Operation

– Programming Lock for Software Security

• Atmel® QTouch® Library Support

– Capacitive Touch Buttons, Sliders and Wheels

– QTouch and QMatrix® Acquisition

– Up to 64 sense channels

• Peripheral Features

– Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode

– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

– Real Time Counter with Separate Oscillator

– Six PWM Channels

– 8-channel 10-bit ADC in TQFP and QFN/MLF package

• Temperature Measurement

– 6-channel 10-bit ADC in PDIP Package

• Temperature Measurement

– Two Master/Slave SPI Serial Interface

– One Programmable Serial USART

– One Byte-oriented 2-wire Serial Interface (Philips I2C compatible)

– Programmable Watchdog Timer with Separate On-chip Oscillator

– One On-chip Analog Comparator

– Interrupt and Wake-up on Pin Change

• Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated Oscillator

– External and Internal Interrupt Sources

– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

• I/O and Packages

– 23 Programmable I/O Lines

– 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

• Operating Voltage:

– 1.8 - 5.5V

• Temperature Range:

– -40°C to 105°C

• Speed Grade:

– 0 - 4MHz @ 1.8 - 5.5V

– 0 - 10MHz @ 2.7 - 5.5V

– 0 - 20MHz @ 4.5 - 5.5V

• Power Consumption at 1MHz, 1.8V, 25°C

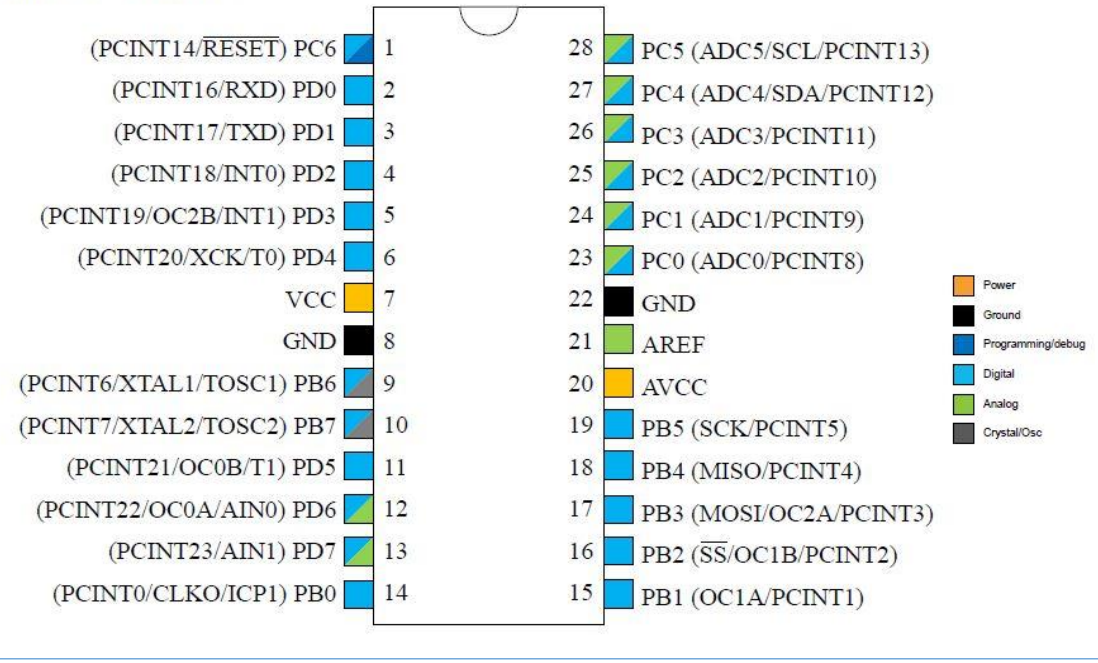
– Active Mode: 0.2mA

– Power-down Mode: 0.1µA

– Power-save Mode: 0.75µA (Including 32kHz RTC)

Pin-out

Figure 5-1. 28-pin PDIP



Pin Descriptions

1. **VCC**

   Digital supply voltage.

2. **GND**

   Ground.

3. **Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2**

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. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

4. Port C (PC[5:0])

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] 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.

5. PC6/RESET

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is un-programmed, PC6 is used as a 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. Shorter pulses are not guaranteed to generate a Reset.The various special features of Port C are elaborated in the *Alternate Functions of Port C* section.

6. Port D (PD[7:0])

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.

7. **AVCC**

AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. 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. Note that PC[6:4] use digital supply voltage, VCC.

8. **AREF**

AREF is the analog reference pin for the A/D Converter.

9. **ADC[7:6] (TQFP and VFQFN Package Only)**

In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Capacitive Touch Sensing

QTouch Library

The Atmel® QTouch® Library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR® microcontrollers. The QTouch Library includes support for the Atmel QTouch and Atmel QMatrix® acquisition methods.

Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing API's to retrieve the channel information and determine the touch sensor states.

The QTouch Library is FREE and downloadable from the Atmel website at the following location: http://www.atmel.com/technologies/touch/. For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from the Atmel website.

I/O Multiplexing

(32-pin MLF/TQFP) Pin#	(28-pin MLF) Pin#	(28-pin PIPD) Pin#	PAD	EXTINT	PCINT	ADC/AC	OSC	T/C #0	T/C #1	USART 0	I2C 0	SPI 0
1	1	5	PD[3]	INT1	PCINT19			OC2B				
2	2	6	PD[4]		PCINT20			T0		XCK0		
4	3	7	VCC									
3	4	8	GND									
6	-	-	VCC									
5	-	-	GND									
7	5	9	PB[6]		PCINT6		XTAL1/ TOSC1					
8	6	10	PB[7]		PCINT7		XTAL2/ TOSC2					
9	7	11	PD[5]		PCINT21			OC0B	T1			
10	8	12	PD[6]		PCINT22	AIN0		OC0A				
11	9	13	PD[7]		PCINT23	AIN1						
12	10	14	PB[0]		PCINT0		CLKO	ICP1				
13	11	15	PB[1]		PCINT1			OC1A				
14	12	16	PB[2]		PCINT2			OC1B				$\overline{SS}$0
15	13	17	PB[3]		PCINT3			OC2A				MOSI0
16	14	18	PB[4]		PCINT4							MISO0
17	15	19	PB[5]		PCINT5							SCK0
18	16	20	AVCC									
19	-	-	ADC6			ADC6						
20	17	21	AREF									
21	18	22	GND									
22	-	-	ADC7			ADC7						
23	19	13	PC[0]		PCINT8	ADC0						
24	20	24	PC[1]		PCINT9	ADC1						
25	21	25	PC[2]		PCINT10	ADC2						
26	22	26	PC[3]		PCINT11	ADC3						
27	23	27	PC[4]		PCINT12	ADC4					SDA0	
28	24	28	PC[5]		PCINT13	ADC5					SCL0	
29	25	1	PC[6]/ $\overline{RESET}$		PCINT14							
30	26	2	PD[0]		PCINT16					RXD0		
31	27	3	PD[1]		PCINT17					TXD0		
32	28	4	PD[2]	INT0	PCINT18							

Each pin is by default controlled by the PORT as a general purpose I/O and alternatively it can be assigned to one of the peripheral functions.

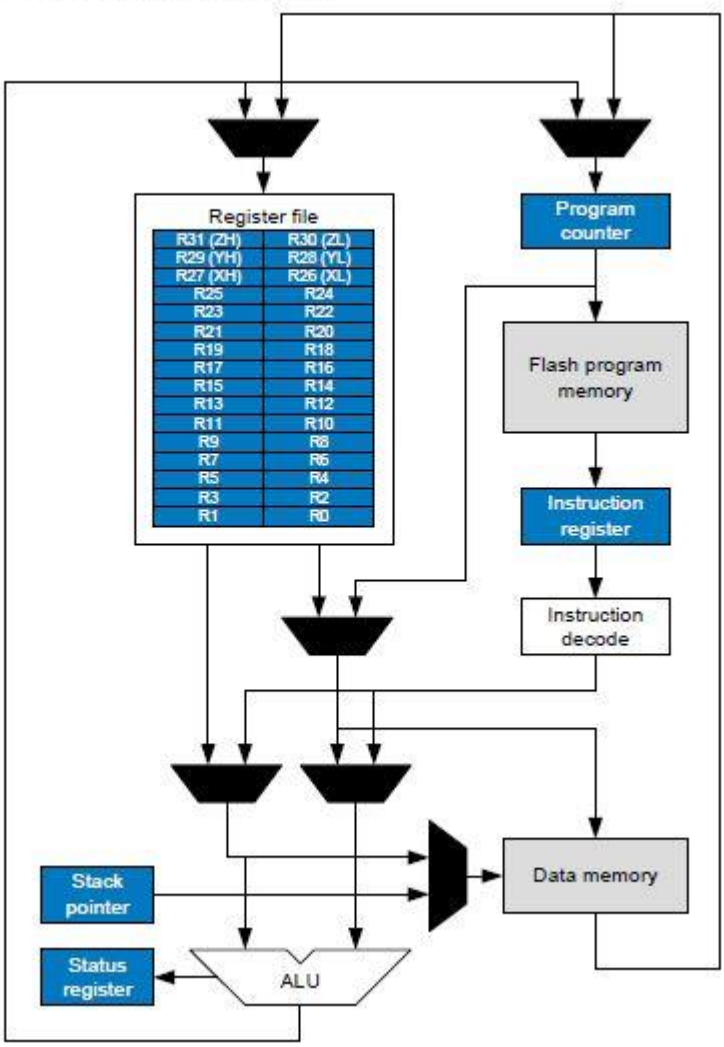
The above table describes the peripheral signals multiplexed to the PORT I/O pins.

AVR CPU Core

11.1. Overview

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access

Block Diagram of the AVR Architecture



memories, perform calculations, control peripherals, and handle interrupts.

In order to maximize performance and parallelism, the AVR uses a Harvard architecture with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File - in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing- enabling efficient address calculations. One of the address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation. Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture. The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, this device has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories - arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See *Instruction Set Summary* section for a detailed description.

Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. The Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

Status Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: SREG

Offset: 0x5F

Reset: 0x00

Property: When addressing as I/O Register: address offset is 0x3F

Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Bit 6 – T: Copy Storage

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

Bit 5 – H: Half Carry Flag

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Flag is useful in BCD arithmetic. See the *Instruction Set Description* for detailed information.

Bit 4 – S: Sign Flag, S = N ⊕ V

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the *Instruction Set Description* for detailed information.

Bit 3 – V: Two's Complement Overflow Flag

The Two's Complement Overflow Flag V supports two's complement arithmetic. See the *Instruction Set Description* for detailed information.

Bit 2 – N: Negative Flag

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 1 – Z: Zero Flag

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

Bit 0 – C: Carry Flag

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the *Instruction Set Description* for detailed information.

General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

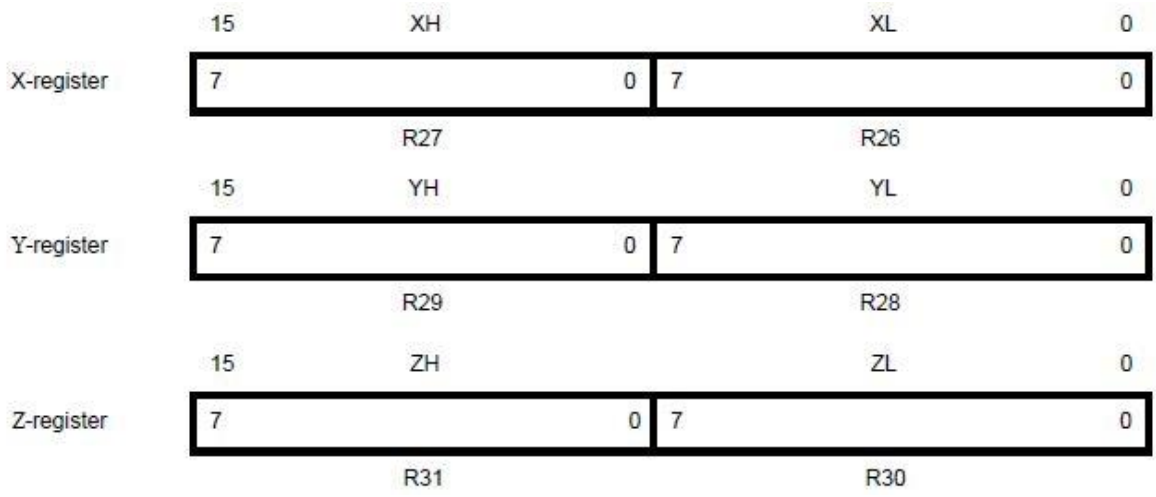
AVR CPU General Purpose Working Registers

	7 0	Addr.	
	R0	0x00	
	R1	0x01	
	R2	0x02	
	...		
	R13	0x0D	
General	R14	0x0E	
Purpose	R15	0x0F	
Working	R16	0x10	
Registers	R17	0x11	
	...		
	R26	0x1A	X-register Low Byte
	R27	0x1B	X-register High Byte
	R28	0x1C	Y-register Low Byte
	R29	0x1D	Y-register High Byte
	R30	0x1E	Z-register Low Byte
	R31	0x1F	Z-register High Byte

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown in the figure, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer registers can be set to index any register in the file.

The X-register, Y-register, and Z-register

The registers R26...R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in the figure.



CONCLUSION

The ATmega328/P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. The P stands for Pico power. By executing powerful instructions in a single clock cycle, the ATmega328/P achieves throughputs close to 1MIPS per MHz .This empowers system designer to optimize the device for power consumption versus processing speed. Due to the above mentioned features, the ATmega328/P is most suited microcontroller for this project.

OTHER COMPONENTS

STEPPER MOTORS

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied.

When to Use?

A stepper motor can be a good choice whenever controlled movement is required. They can be used to advantage in applications where you need to control rotation angle, speed, position and synchronism. Because of the inherent advantages listed previously, stepper motors have found their place in many different applications. Some of these include printers, plotters, high-end office equipment, hard disk drives, medical equipment, fax machines, automotive and many more.

Types of Stepper Motor:

There are three main types of stepper motors, they are:

1. Permanent magnet stepper
2. Hybrid synchronous stepper
3. Variable reluctance stepper

Permanent Magnet Stepper Motor: Permanent magnet motors use a permanent magnet (PM) in the rotor and operate on the attraction or repulsion between the rotor PM and the stator electromagnets.

Variable Reluctance Stepper Motor: Variable reluctance (VR) motors have a plain iron rotor and operate based on the principle that minimum reluctance occurs with minimum gap, hence the rotor points are attracted toward the stator magnet poles.

Hybrid Synchronous Stepper Motor: Hybrid stepper motors are named because they use a combination of permanent magnet (PM) and variable reluctance (VR) techniques to achieve maximum power in a small package size.

Advantages of Stepper Motor:

1. The rotation angle of the motor is proportional to the input pulse.
2. The motor has full torque at standstill.
3. Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non-cumulative from one step to the next.
4. Excellent response to starting, stopping and reversing.
5. Very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependent on the life of the bearing.
6. The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
7. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
8. A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Applications:

1. **Industrial Machines** – Stepper motors are used in automotive gauges and machine tooling automated production equipment.
2. **Security** – new surveillance products for the security industry.
3. **Medical** – Stepper motors are used inside medical scanners, samplers, and also found inside digital dental photography, fluid pumps, respirators and blood analysis machinery.
4. **Consumer Electronics** – Stepper motors in cameras for automatic digital camera focus and zoom functions.

And also have business machines applications, computer peripherals applications.

Operation of Stepper Motor:

Stepper motors operate differently from DC brush motors, which rotate when voltage is applied to their terminals. Stepper motors, on the other hand, effectively have multiple

toothed electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energized by an external control circuit, for example a microcontroller.

To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. The point when the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. So when the next electromagnet is turned ON and the first is turned OFF, the gear rotates slightly to align with the next one and from there the process is repeated. Each of those slight rotations is called a step, with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise. Stepper motor doesn't rotate continuously, they rotate in steps. There are 4 coils with 90^o angle between each other fixed on the stator. The stepper motor connections are determined by the way the coils are interconnected. In stepper motor, the coils are not connected together. The motor has 90^o rotation step with the coils being energized in a cyclic order, determining the shaft rotation direction. The working of this motor is shown by operating the switch. The coils are activated in series in 1 sec intervals. The shaft rotates 90^o each time the next coil is activated. Its low speed torque will vary directly with current.

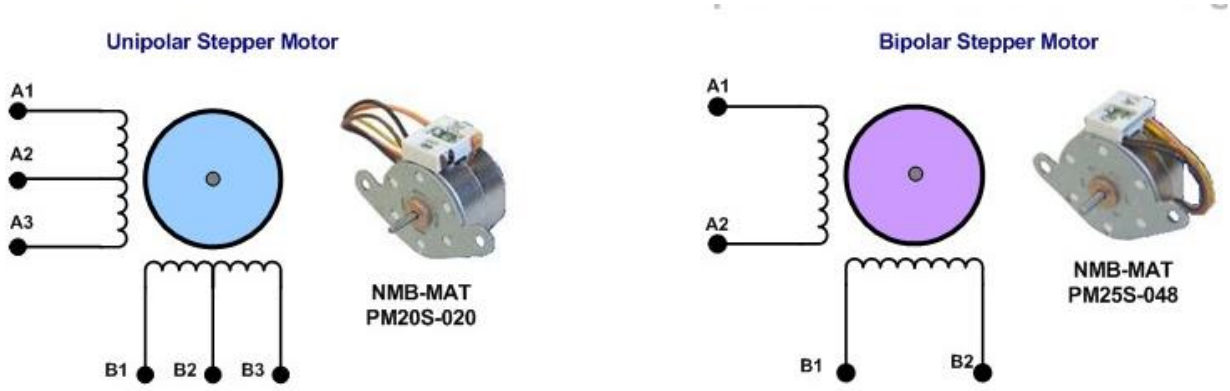
Unipolar stepper motors

A unipolar stepper motor has two windings per phase, one for each direction of magnetic field. Since in this arrangement a magnetic pole can be reversed without switching the direction of current, the commutation circuit can be made very simple (eg. a single

transistor) for each winding. Typically, given a phase, one end of each winding is made common: giving three leads per phase and six leads for a typical two phase motor. Often, these two phase commons are internally joined, so the motor has only five leads.
A microcontroller or stepper motor controller can be used to activate the drive transistors in the right order, and this ease of operation makes unipolar motors popular with hobbyists; they are probably the cheapest way to get precise angular movements.
(For the experimenter, one way to distinguish common wire from a coil-end wire is by measuring the resistance. Resistance between common wire and coil-end wire is always half of what it is between coil-end and coil-end wires. This is due to the fact that there is actually twice the length of coil between the ends and only half from center (common wire) to the end.)
A quick way to determine if the stepper motor is working is to short circuit every two pairs and try turning the shaft, whenever a higher than normal resistance is felt, it indicates that the circuit to the particular winding is closed and that the phase is working.

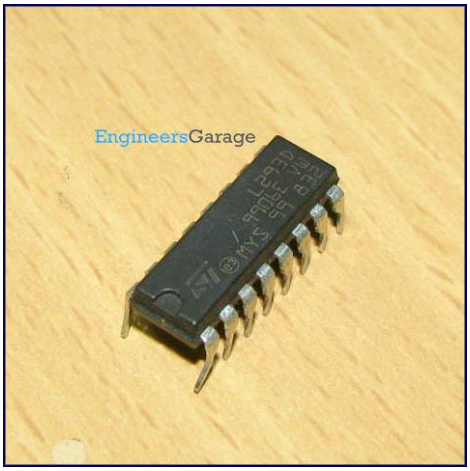
Bipolar stepper motors

Bipolar motors have a single winding per phase. The current in a winding needs to be reversed in order to reverse a magnetic pole, so the driving circuit must be more complicated, typically with an H-bridge arrangement (however there are several off the shelf driver chips available to make this a simple affair). There are two leads per phase, none are common. Because windings are better utilized, they are more powerful than a unipolar motor of the same weight. This is due to the physical space occupied by the windings. A unipolar motor has twice the amount of wire in the same space, but only half used at any point in time, hence is 50% efficient (or approximately 70% of the torque output available). Though bipolar is more complicated to drive, the abundance of driver chip means this is much less difficult to achieve.
An 8-lead stepper is wound like a unipolar stepper, but the leads are not joined to common internally to the motor. This kind of motor can be wired in several configurations.



The Unipolar and Bipolar Stepper Motor Windings

L293D motor driver IC

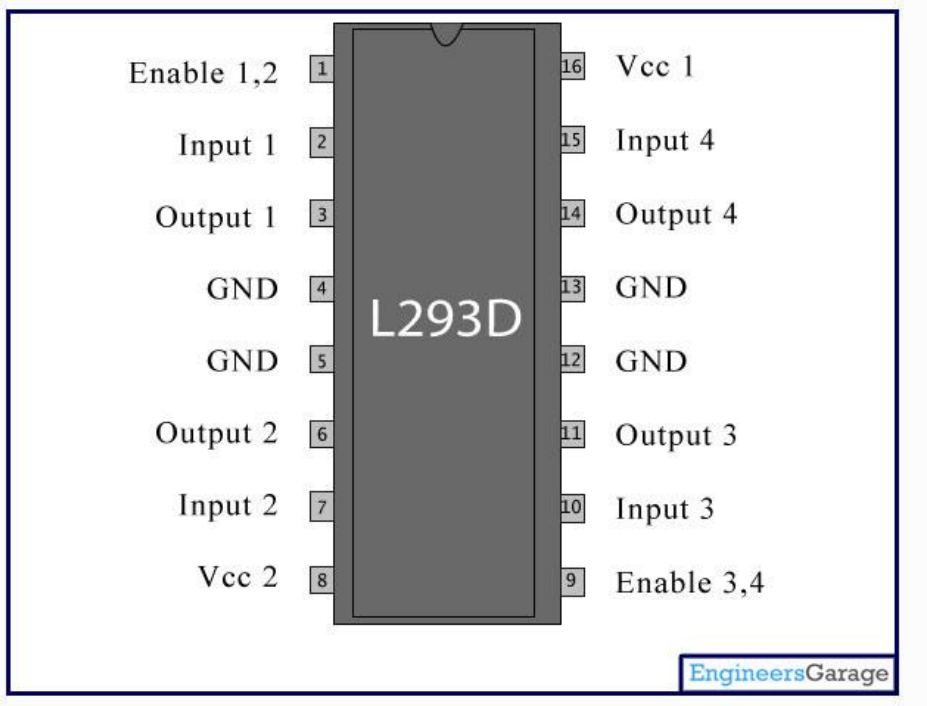


L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state.

Pin Diagram:

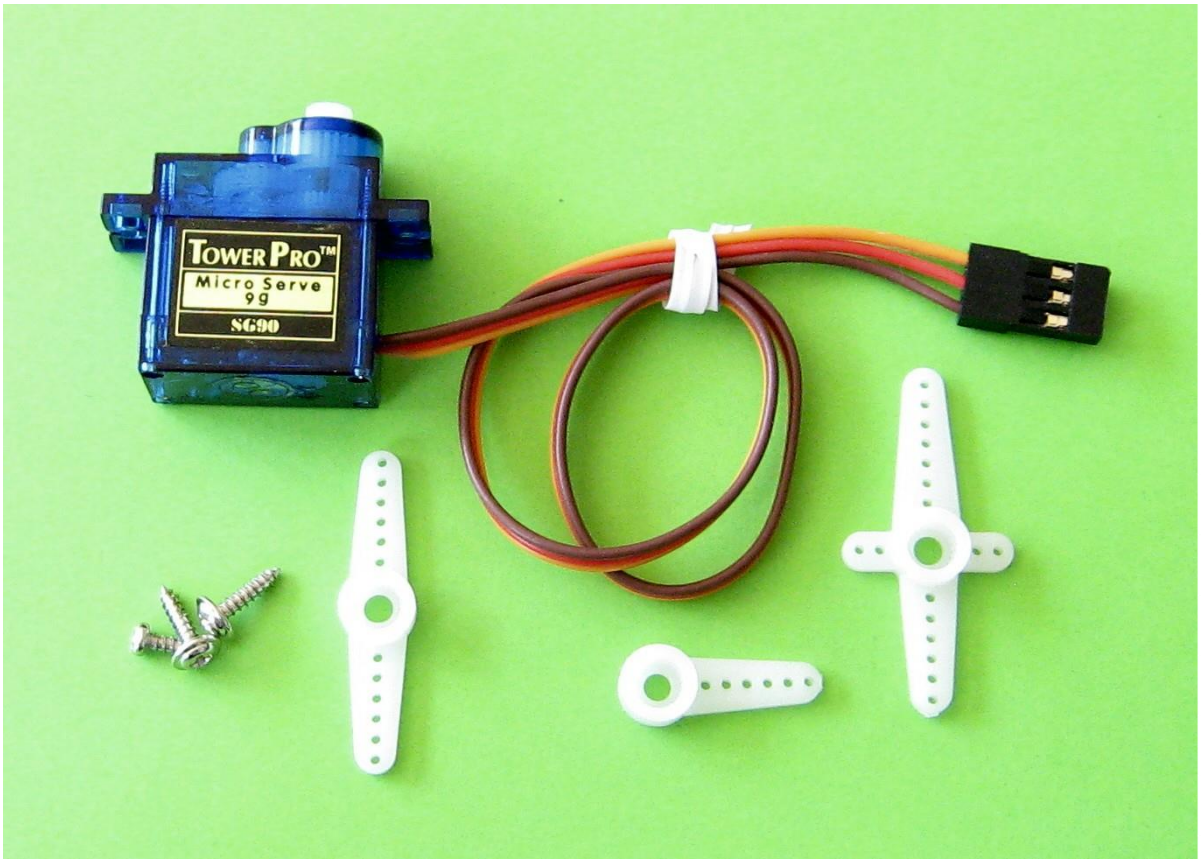


Pin Description:

Pin No	Function	Name
1	Enable pin for Motor 1; active high	Enable 1,2
2	Input 1 for Motor 1	Input 1
3	Output 1 for Motor 1	Output 1
4	Ground (0V)	Ground
5	Ground (0V)	Ground
6	Output 2 for Motor 1	Output 2
7	Input 2 for Motor 1	Input 2
8	Supply voltage for Motors; 9-12V (up to 36V)	Vcc_2
9	Enable pin for Motor 2; active high	Enable 3,4
10	Input 1 for Motor 1	Input 3
11	Output 1 for Motor 1	Output 3
12	Ground (0V)	Ground
13	Ground (0V)	Ground
14	Output 2 for Motor 1	Output 4
15	Input2 for Motor 1	Input 4
16	Supply voltage; 5V (up to 36V)	Vcc_1

Servo motor

A servo motor is an electrical device which can push or rotate an object with great precision. To rotate and object at some specific angles or distance, servo motor is used. It is just made up of simple motor which run through servo mechanism. If motor is used is DC powered then it is called DC servo motor, and if it is AC powered motor then it is called AC servo motor. We can get a very high torque servo motor in a small and light weight packages. Doe to these features they are being used in many applications like toy car, RC helicopters and planes, Robotics, CNC Machine etc. The position of a servo motor is decided by electrical pulse and its circuitry is placed beside the motor.

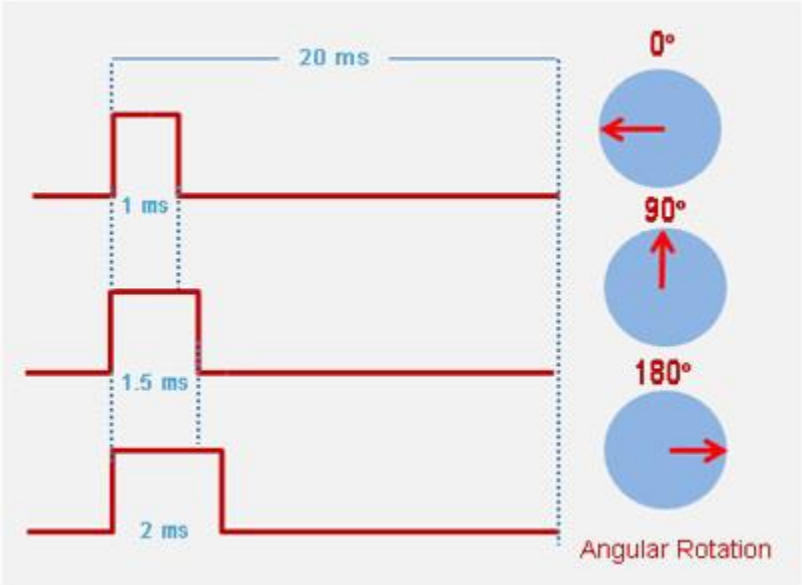


Working principle of Servo Motors.

A servo consists of a Motor (DC or AC), a potentiometer, gear assembly and a controlling circuit. First of all we use gear assembly to reduce RPM and to increase torque of motor. Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer. Now an electrical signal is given to another input terminal of the error detector amplifier. Now difference between these two signals, one comes from potentiometer and another comes from

other source, will be processed in feedback mechanism and output will be provided in term of error signal. This error signal acts as the input for motor and motor starts rotating. Now motor shaft is connected with potentiometer and as motor rotates so the potentiometer and it will generate a signal. So as the potentiometer's angular position changes, its output feedback signal changes. After sometime the position of potentiometer reaches at a position that the output of potentiometer is same as external signal provided. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the signal generated at potentiometer, and in this situation motor stops rotating.

Controlling Servo Motor



Servo motor is controlled by PWM (Pulse with Modulation) which is provided by the control wires. There is a minimum pulse, a maximum pulse and a repetition rate. Servo motor can turn 90 degree from either direction form its neutral position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 position, such as if pulse is shorter than 1.5ms shaft moves to 0 and if it is longer than 1.5ms than it will turn the servo to 180.

Servo motor works on PWM (Pulse width modulation) principle, means its angle of rotation is controlled by the duration of applied pulse to its Control PIN. Basically servo motor is made up of DC motor which is controlled by a variable resistor (potentiometer) and some gears. High speed force of DC motor is converted into torque by Gears. We know that WORK= FORCE X DISTANCE, in DC motor Force is less and distance (speed) is high and in Servo, force is High and distance is less. Potentiometer is connected to the output shaft of the Servo, to calculate the angle and stop the DC motor on required angle. Servo motor can be rotated from 0 to 180 degree, but it can go up to 210 degree, depending on the manufacturing. This

degree of rotation can be controlled by applying the Electrical Pulse of proper width, to its Control pin.

Servo checks the pulse in every 20 milliseconds. Pulse of 1 ms (1 millisecond) width can rotate servo to 0 degree, 1.5ms can rotate to 90 degree (neutral position) and 2 ms pulse can rotate it to 180 degree.

CONCLUSION

Stepper motors are very useful in controlled and high precision work such as in CNCs. Various types of Stepper motors are available. L293DNE is the best cheap motor driver IC which can we use for educational purposes. L293D is a typical Motor driver or Motor Driver IC which allows DC motor to drive on either direction. It is an H-bridge which can be used to drive two dc motors or one stepper motor. Servo Motors are easy to use motors for precision movement (angle). Stepper motors from CD/DVD drives of old computers are bipolar in nature and have 4 wires which can be driven by one L293D IC.

INDUSTRIAL DESIGN

Introduction

The complete mechanical system was designed in the scrap metallic CD drive and scrap plywood.

The designs in the project are:

- X-Y Direction.
- Pen setup.
- Stand holding the Whole.
- Final Setup

Y-axis: basic axis carries X-axis move from front to back.

X-axis: carries Z-axis move from left to right.

Z-axis: carries pen part move up and down.

❖ X-Y Direction

In computing, an optical disc drive (ODD) is a disk drive that uses laser light or electromagnetic waves within or near the visible light spectrum as part of the process of reading or writing data to or from optical discs. Some drives can only read from certain discs, but recent drives can both read and record, also called burners or writers. Compact discs, DVDs, and Blu-ray discs are common types of optical media which can be read and recorded by such drives. Optical disc drives that are no longer in production include CD-ROM drive, CD writer drive, and combo (CD-RW/DVD-ROM) drive. As of 2015, DVD writer drive is the most common for desktop PCs and laptops. There are also the DVD-ROM drive, BD-ROM drive, Blu-ray Disc combo (BD-ROM/DVDRW/CD-RW) drive, and Blu-ray Disc writer drive.

The stepper motor setup of CD drives are used in X-Y direction co-ordinate axis.

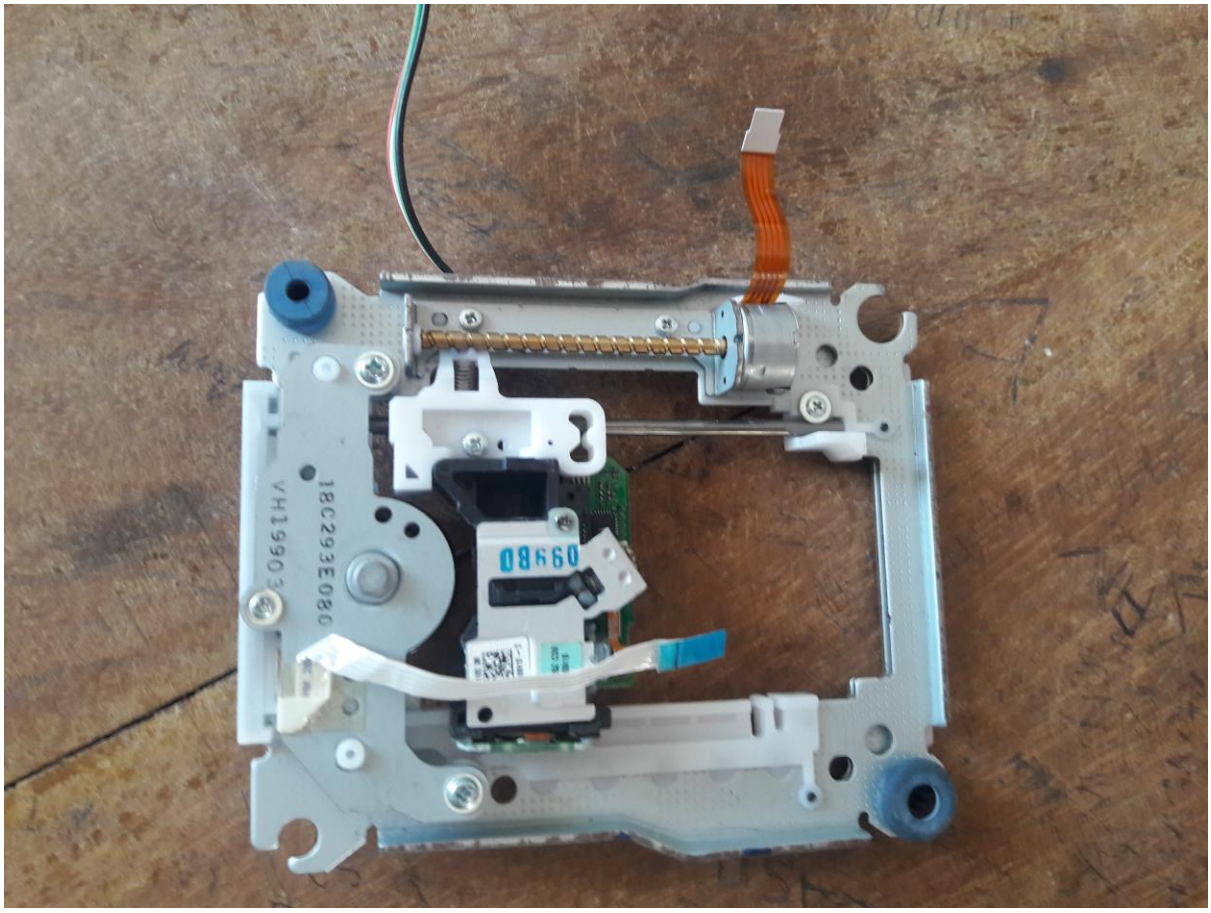


Fig: Lens Frame in CD Drive (Containing Stepper Motor)

❖ Stand holding the whole

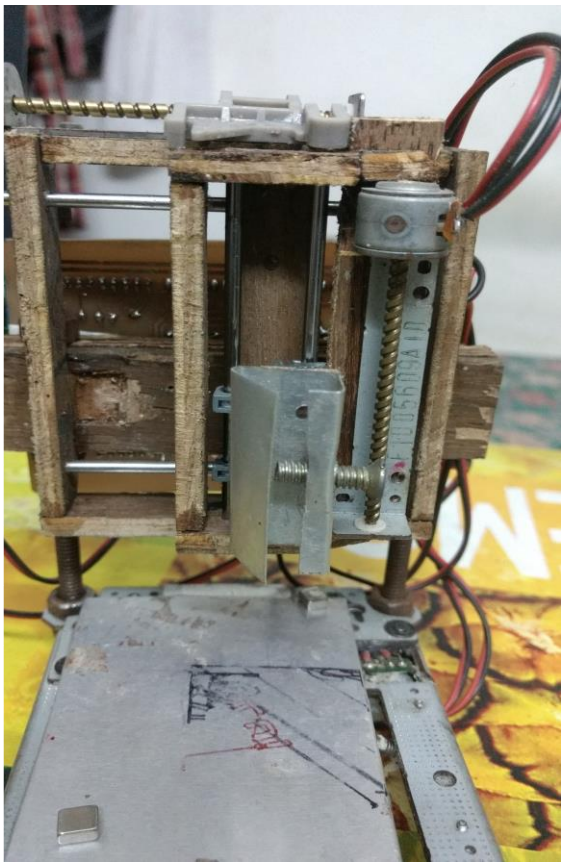
The stand holding all the parts are made by the scrap plywood from a shop constructing site. The pieces are joined together respectively for holding the x and y axis.

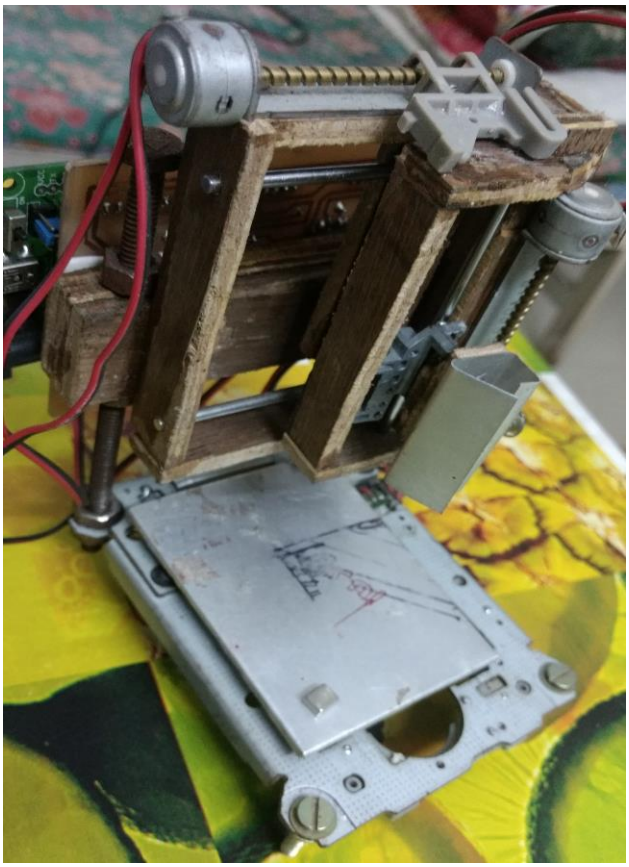
❖ Pen Setup (Z-axis)

For pen setup (z axis) another stepper motor setup and aluminum sheet is used. Servomotor is adjusted inside it to get the up and movement required to plot the object.

❖ Final Setup

All the sections are integrated together to get a good output.

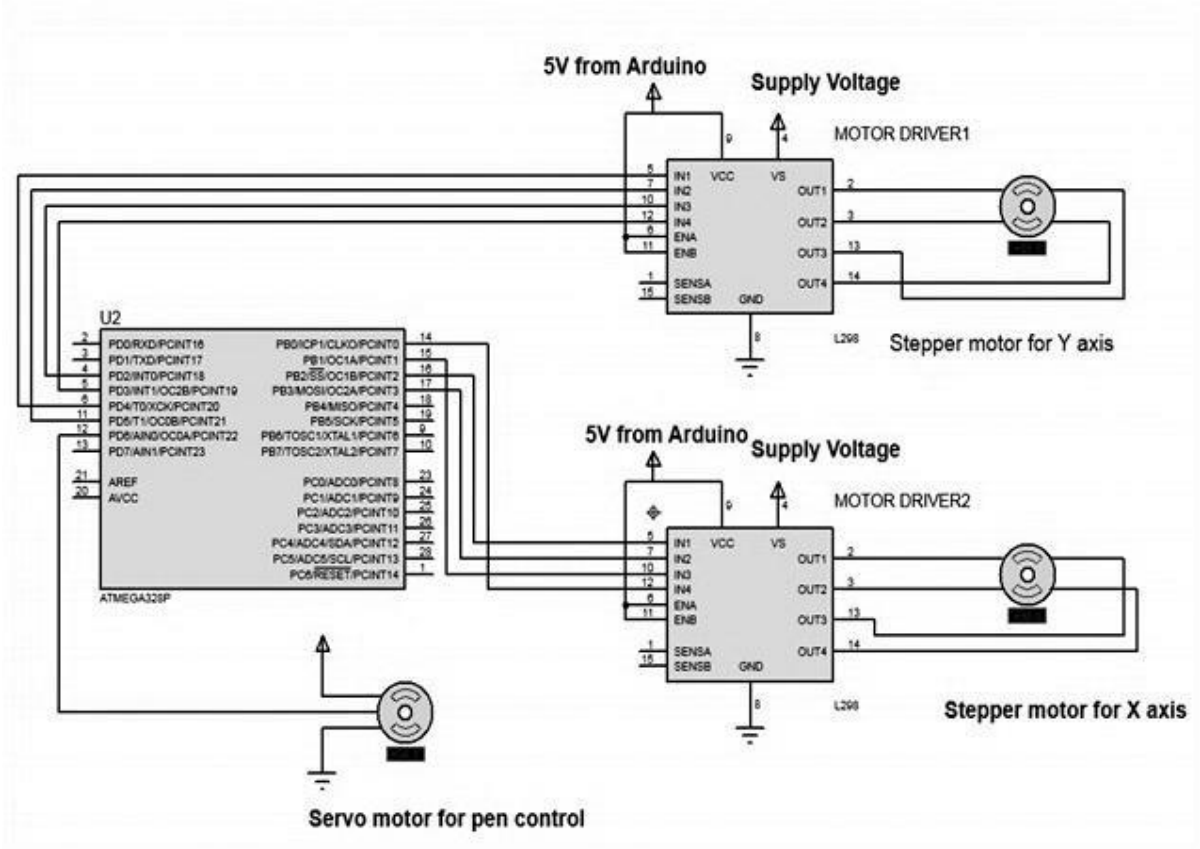




CONCLUSION

A simple model of a 3 axis CNC plotter is made using CD/DVD stepper motors and scrap CD drive covers and plywood. Each axes tested individually and a test print is plotted finally.

CIRCUIT DIAGRAM AND EXPLANATION



For the X and Y axes we will use two stepper motors and rails from dvd/cd drives and for the Z axis we will use a small servo motor that moves the pen up and down. For the mounting base we will use a small piece of aluminum sheet.

You can easily attach a pen (or pencil) - irrespective of its thickness - on it. I tried to use an extension of cutting tool (e.g.Dremel) to engrave materials with no success. So this mini CNC can only be used as a small plotter and not as an engraver machine.

The Arduino-based circuit is using the ATmega328 microcontroller, two L293DNE motor driver ICs and an USB to serial module. You can easily make it with the Arduino Uno board and a breadboard.

❖ Steps involved in the project

Step 1-Industrial Design

1. First step to start building this CNC machine is to disassemble two DVD/CD drives and take off them the stepper motors. Use the screwdriver to open them and take o_ them the rails.
2. The outer metallic cover of cd drive is welded perpendicularly to make the stand holding the x and y axis.
3. Attach the cd drive stepper motor setup as x and y axis. And make sure that the Y axis is straight to CNC base and the X axis vertically to it.
4. Z axis (pen setup) is attached to the x axis. The pen setup is made up of aluminum sheet, the servo motor is attached to it and the pen is setup inside the fiber using screw and spring.
5. A metallic base is attached to the Y axis for using as paper base. Then a paper is put above it with the help some magnets. The printing area is 4x4cm.

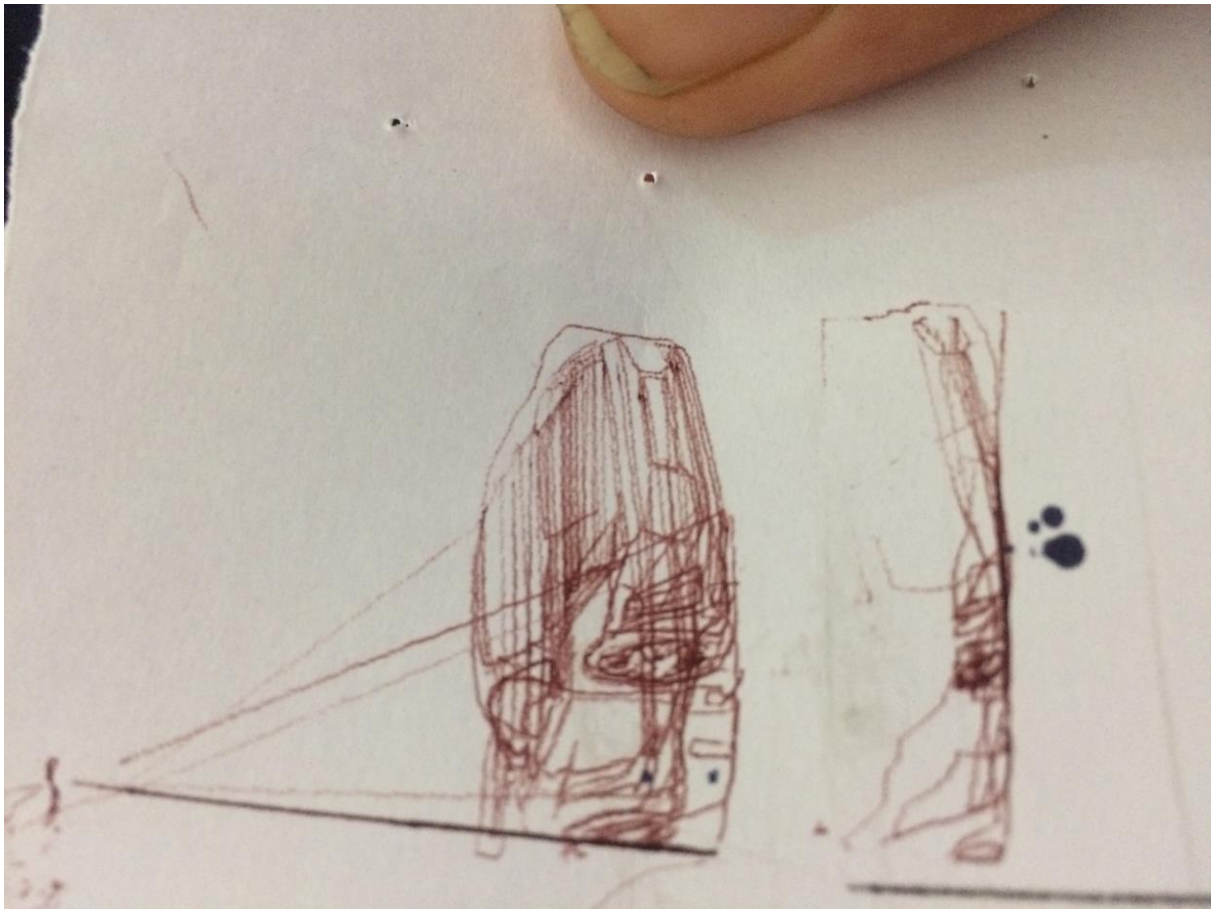
Step 2-Arduino and Stepper Motor Setup

1. The L293D motor driver with the Arduino board is mounted on it.
2. The Arduino is connected the computer port.
3. Check the stepper motors and the servo motor.
4. The stepper motors and the servo motor are connected to the motor shield.
5. The external power is connected.

Step 3-Burning of Program and Gcode take in

1. The mini CNC plotting sketch is burned to the Arduino microprocessor (ATmega 328p) by using Arduino IDE.
2. Gcode is made by Inkscape program.
3. Then use the Pronterface program. This program sends 'gcode' images to the CNC plotter.
4. Plotting of the image is done.

OUTPUT:



Module 2: Communication Section

Software

Engineering as a discipline often requires more integration than large amounts of original development. In a typical project, writing new code presents significant challenges, and the number of features shared between projects means that it is possible to create shared components which implement common features. A library or an existing module allows the use of a well developed and tested component, which saves significant resources in the implementation of the project. The drawback of components is the need to integrate various potentially conflicting interfaces, and the need to understand a complex system in order to effectively use the component. Components can be purchased, or may be freely available, as in the case of Open Source software. Open Source also provides the opportunity to contribute new features and bug fixes back in to the community. The programs and tools we chose for this project are all open source, and use international standards, which allowed to rapidly develop the features needed.

The project software system consists of:

1. Inkscape

Inkscape is a free and open-source vector graphics editor; it can be used to create or edit vector graphics such as illustrations, diagrams, line arts, charts, logos and complex paintings. Inkscape's primary vector graphics format is Scalable Vector Graphics (SVG), however many other formats can be imported and exported.

Inkscape can render primitive vector shapes (e.g. rectangles, ellipses, polygons, arcs, spirals, stars and 3D boxes) and text. These objects may be filled with solid colors, patterns, radial or linear color gradients and their borders may be stroked, both with adjustable transparency. Embedding and optional tracing of raster graphics is also supported, enabling the editor to create vector graphics from photos and other raster sources. Created shapes can be further manipulated with transformations, such as moving, rotating, scaling and skewing.

There are two basic types of graphic images: bitmap (or raster) images and vector images.

In the first case, the image is defined in terms of rows and columns of individual pixels, each with its own color. In the second case, the image is defined in terms of lines, both straight and curved. A single straight line is described in terms of its two end points.

The difference in these types of graphic images becomes readily apparent when a drawing is enlarged. The same line is shown on the left and right. On the left it is displayed as a bitmap image, while on the right it is displayed as a vector. In both cases, the line has been scaled up by a factor of four from its nominal size.

When the bitmap resolution of a drawing matches the display resolution, the objects in the drawing look smooth. The same drawing, but defined as a bitmap image on the left and a vector image on the right. If the output device has the same resolution as the bitmap image, there is little difference between the appearances of the two images.

If the bitmap resolution is significantly less than the display resolution, the display will show jagged lines. The head of the gentleman in the above drawings has been scaled up by a factor of five. Now one can see a difference in the quality of the bitmap drawing (left) and the vector drawing (right). Note that the bitmap image uses anti-aliasing, a method of using grayscale to attempt to smooth the drawing.

All output devices, with few exceptions, use a raster or bitmap image to display graphics. The real difference between drawing with bitmap graphics and vector graphics is the point at which the image is converted into a bitmap. In the case of vector graphics, this conversion is done at the very last step before display, ensuring that the final image matches exactly the resolution of the output device.

Scalar Vector Graphics (SVG)

SVG stands for Scalable Vector Graphics. Scalable refers to the notion that a drawing can be scaled to an arbitrary size without losing detail. Scalable also refers to the idea that a drawing can be composed of an unlimited number of smaller parts, parts that can be reused many times. The SVG standard is directed toward a complete description of two-dimensional graphics, including animation in an XML (eXtensible Markup Language) format. XML is an open standard for describing a document in a way that can be easily extended and is resistant

to future changes in the document specification. A drawing saved in one version of SVG by one version of a drawing program should be viewable, to the full extent possible, by any previous or future version of any drawing program that adheres to the SVG standard. If a program doesn't support something in the SVG standard, it should just skip over any part of a drawing that uses it, rendering the rest correctly.

SVG files are small and drawings described by the standard adapt well to different presentation methods. This has led to great interest in the standard. Support is included in many web browsers (Firefox, Chrome, Opera, Safari, and Internet Explorer from version 9), or is available through plug-ins (e.g., Adobe [http://www.adobe.com/svg/viewer/install/], Ssrc SVG [http://www.savarese.com/software/svgplugin/], and Google [http://www.google.com/chromeframe]). Over a dozen companies including Apple (iPhone), Blackberry, LG, Motorola, Nokia, Samsung, and Sony Ericsson produce mobile phones that utilize a subset of the full SVG standard (SVG Tiny) that has been tailored for devices with limited resources.

Inkscape is a free and open-source vector graphics editor; it can be used to create or edit vector graphics such as illustrations, diagrams, line arts, charts, logos and complex paintings. Inkscape's primary vector graphics format is Scalable Vector Graphics (SVG) version 1.1. While Inkscape can import and export several formats, all editing workflow inevitably occur within the guidelines of the SVG format.

Inkscape can render primitive vector shapes (e.g. rectangles, ellipses, polygons, arcs, spirals, stars and isometric boxes), text and regions containing raster graphics. It also supports image tracing, enabling the editor to create vector graphics from photos and other raster sources. Created shapes can be subjected to further transformations, such as moving, rotating, scaling and skewing. These objects may be filled with solid colors, patterns, radiant or linear color gradient, their borders stroked or their transparency changed.

Inkscape SVG-based vector drawing program is useful for drawing:

- Illustrations for the Web.
- Graphics for mobile phones.
- Simple line drawings.
- Cartoons.
- Complex works of art.

- Figures for articles and books.
- Organization charts.

The file format that Inkscape uses is compact and quickly transmittable over the Internet. Yet it is powerful and can describe complex drawings that are scalable to any size. Support for the format has been added to web browsers and is already included in many mobile phones.

Inkscape supports the drawing of regular shapes (rectangles, circles, etc.), arbitrary paths, and text. These objects can be given a wide variety of attributes such as color, gradient or patterned fills, alpha blending, and markers. Objects can be transformed, cloned, and grouped. Hyperlinks can be added for use in web browsers. The Inkscape program aims to be fully XML, SVG, and CSS compliant.

Inkscape is available prepackaged for the Windows, Macintosh, and Linux operating systems.

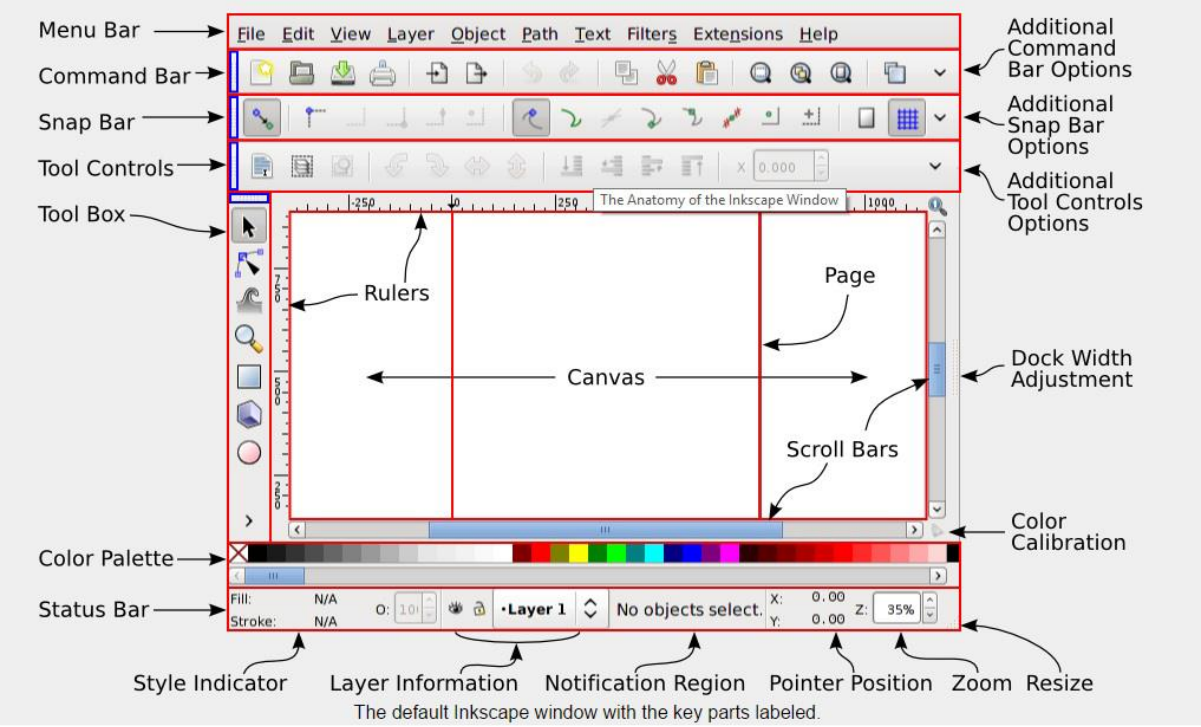
The program and its source code are freely available. They can be obtained from the Inkscape website [http://www.inkscape.org/]. Inkscape is undergoing very rapid development with new features being added and compliance to the SVG standard being constantly improved.

Inkscape Window

Start by opening Inkscape. This window contains several major areas, many containing clickable icons or pull-down menus. The following figure shows this window and labels key parts.

The Command Bar, Snap Bar, Tool Controls, and Tool Box are detachable by dragging on the handles (highlighted in blue) at the far left or top. They can be returned to their normal place by dragging them back. New in v0.48: Some of the bars change position depending on which option is selected at the bottom of the View menu. When Default is selected, the Command Bar is on the top while the Snap Bar is on the right. When Custom is selected, the Command Bar and the Snap Bar are both on the top. When Wide is selected, the Command Bar and the Snap Bar are both on the right. By default, Default is used if you are not using a \Wide Screen" display while Wide is used if you are. A width to height aspect ratio of greater than 1.65 is defined to be wide. These bars, as well as the Palette and Status Bar, can be hidden using the View Show/Hide submenu.

As Inkscape has grown more complex, the area required to include icons and entry boxes for all the various items has also grown leading to problems when Inkscape is used on small screens. The Command Bar, Snap Bar, Tool Controls, and Tool Box have variable widths or heights. If there are too many items to be shown in the width (height) of the Inkscape window, a small down arrow will appear on the right side or bottom of the bars. Clicking on this arrow will open a drop-down menu with access to the missing items.



The default Inkscape window with the key parts labeled.

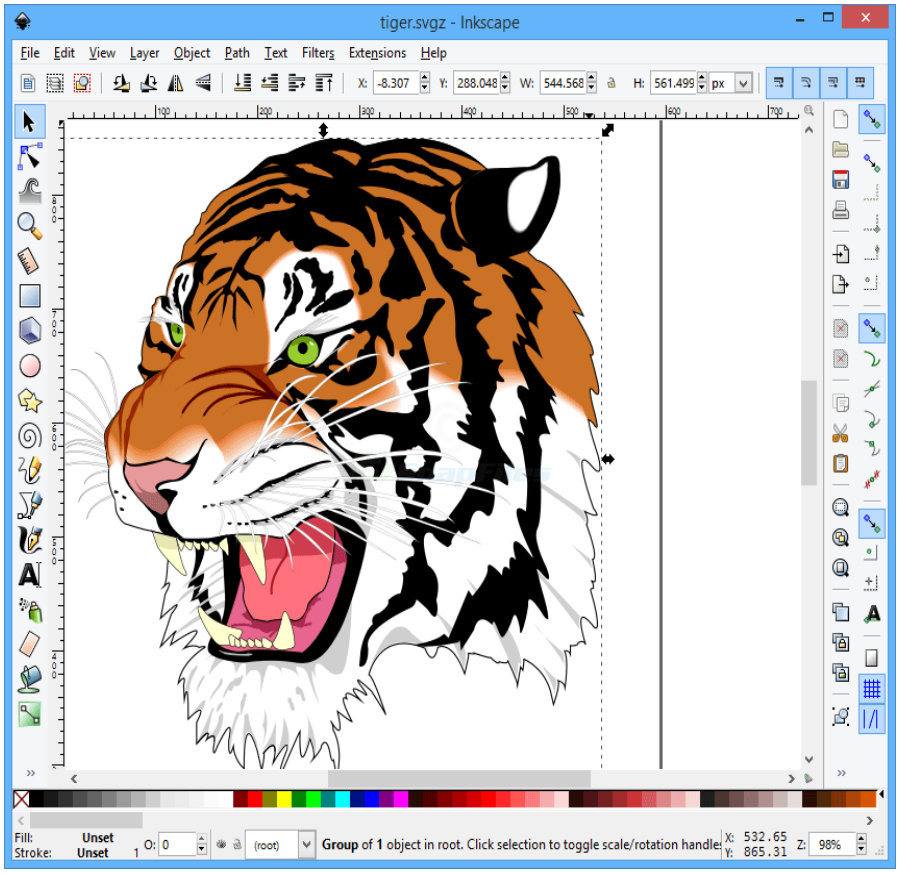
In this project the use of Inkscape is to convert any image (formats) into graphics code usually known as GCODE. .GCODE formats are generated by integrating Inkscape with necessary extension files.

Generating gcode files using Inkscape

1. Download and install Inkscape 0.48.5 version.
2. Install an Add-on that enables the export images to gcode files.
3. Open the Inkscape, go to File menu and click "Document Properties".
4. Change the custom size.
5. Now close this window.
6. Open the required image.
7. Re-size the image to fit our printing area.
8. Click Path from menu and "Trace Bitmap". Make required changes.

9. Click ok and close the window.
10. Now, move the gray scale image, and delete the color one behind it. Move the grey image to the correct place again and click from Path menu "Object to path".
11. Final, go to file menu, click save as and select .gcode. Click ok on next window.

GCode Tools: Gcode tools is an open source Inkscape extension, to export gcode for use with a CNC machine, written in the Python programming language. Inkscape extensions work in the standard Unix IO model, taking SVG on standard input, and output transformed SVG on standard output. The Gcode tools extension generates G-Code from the SVG input and writes it to a file as a side effect of the SVG transformation. This python extension can be easily downloaded as a .ZIP file from https://github.com/martymcguire/inkscape-unicorn.



2. Printrun

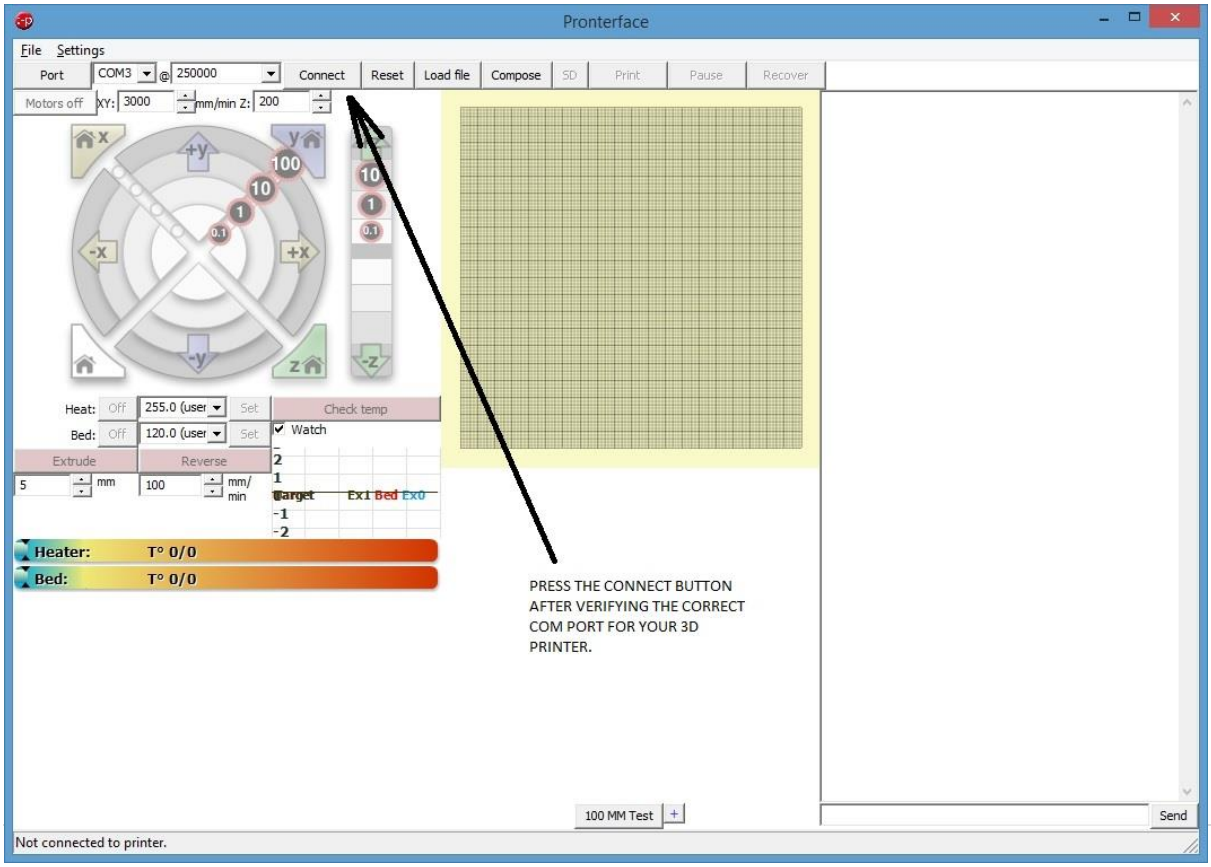
- **Pronterface**

Printrun is a set of G-code sending applications, written by Kliment. It consists of printcore (dumb G-code sender), Pronsole (featured command line G-code sender), **Pronterface** (featured G-code sender with graphical user interface), and a small collection of helpful scripts. Together with skeinforge or Slic3r they form a powerful printing toolchain.

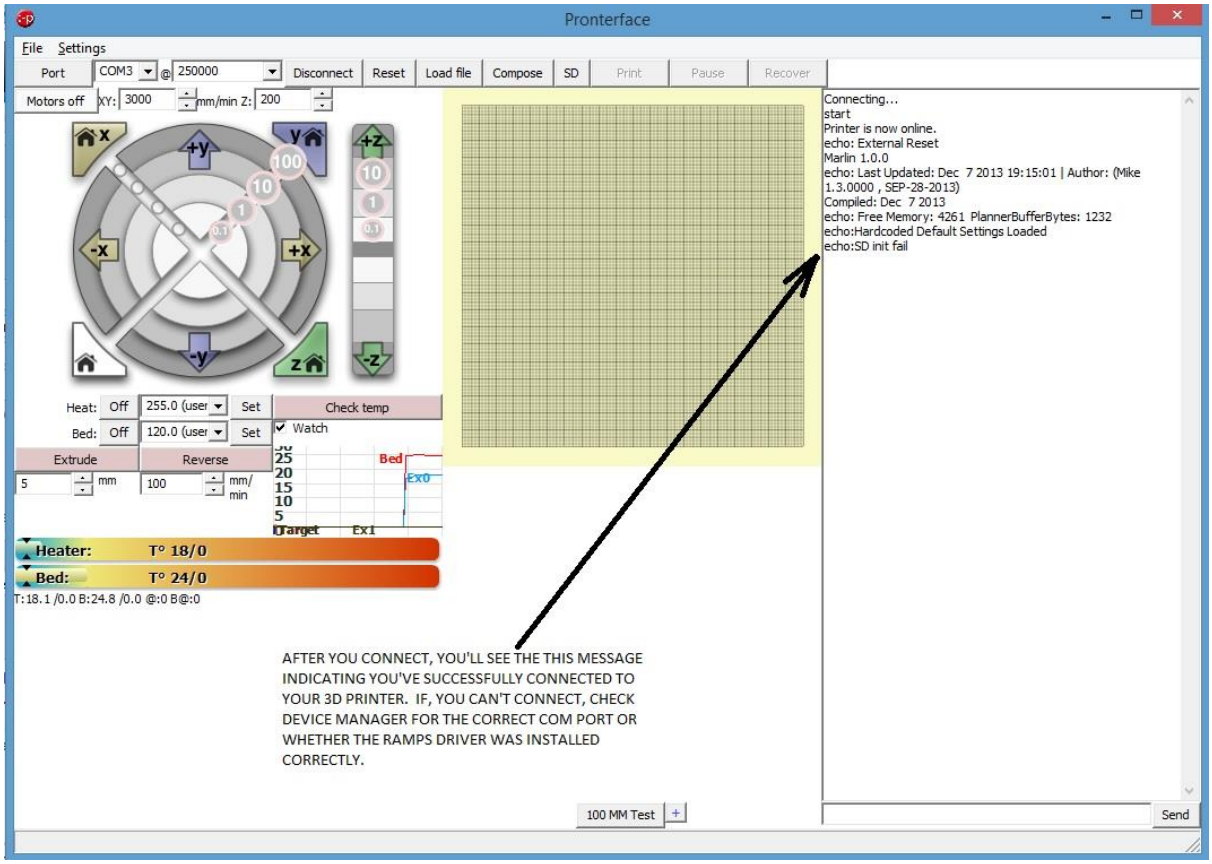
Pronterface is a host software for Reprap electronics, originally developed by Kliment. It is mostly oriented towards 3D printing, but can also be used to control laser cutters and CNC routers.

Connecting to your plotter

Start by connecting your plotter to your computer via a USB cable. Connect the provided power supply to the plotter and plug the power supply into a power outlet. Next open Pronterface by navigating to the directory you have installed Pronterface in (i.e. C:\Pronterface) and double click on the Pronterface.exe icon. After Pronterface loads select



the com **port** your plotter is connected to and then set the **baud** rate to 9600. Next click on the connect button. You will see a message in the right column of Pronterface indicating that the printer has successfully connected.

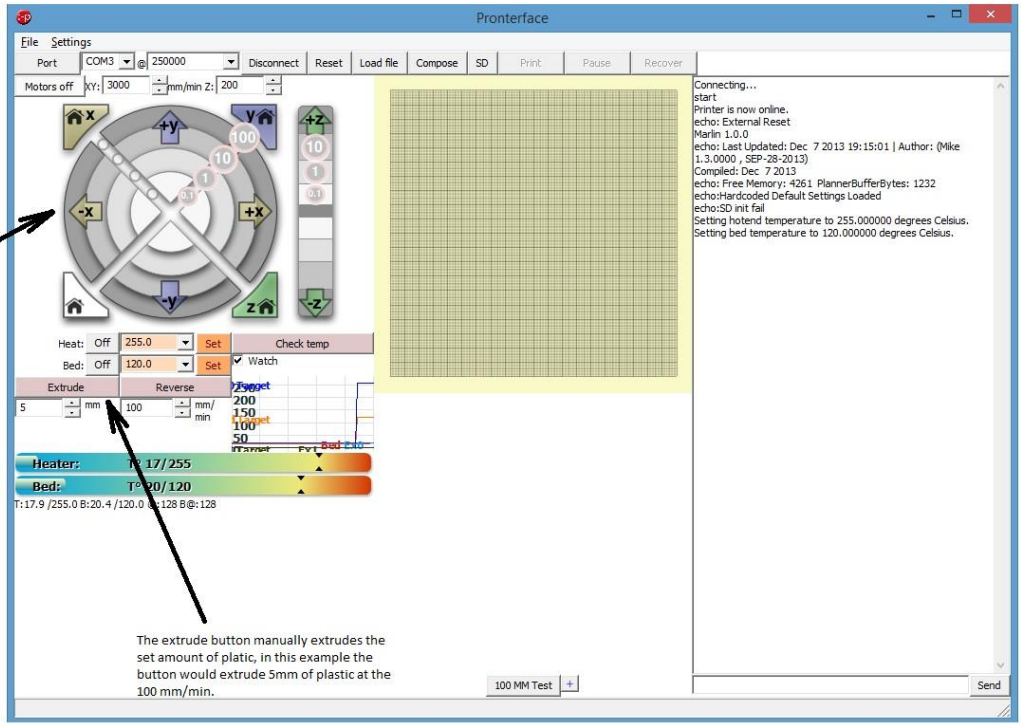


Homing and Moving the X, Y and Z Axis

Once connected you can use Pronterface to manually control the X, Y, and Z axis. Before manually moving an axis, you may want to home the axis beforehand to prevent the axis from over shooting its boundary. After the axis is homed it will not be able to travel past its maximum set limit of 204mm. To home an axis (moving the axis to its zero start position), press the X-home, Y-home, or Z-home icon to home the X, Y, or Z axis respectively. When a print is started the plotter will automatically home each axis so it is not necessary to home the axes before printing.

After homing you may move any axis, X, Y, or Z by pressing the X+, X-, Y+, Y-, or Z+, Z- directional icons. The X and Y directions have 4 distances to choose from to move at a time—.1mm, 1mm, 10mm, or 100mm in either the plus or negative direction. The Z axis directions has 3 distances to choose from–.1mm, 1mm, and 10mm. When an axis is at its home

position it is at zero. You can move each axis of your plotter 204mm from the home position. Starting from zero press the icon in the positive direction. To move towards the home position move in the negative direction. You can also change the rate at which the X, Y and Z axis move by entering a value in the XY mm/min dialog box and the Z mm/min dialog box, however, we recommend using the default values of 2000 mm/min for the X and Y axis and 200 for the Z axis.



G-Code

G-Code is the generic name for a control language for CNC (or Reprap) machines. It is a way for you to tell the machine to move to various points at a desired speed, control the spindle speed, turn on and off various coolants, and all sorts of other things. It is fairly standard, and is a useful tool.

Gcode is a file with X,Y and Z coordinates. Header of this file is set to:

M300 S30.00 (Servo down) G1 X10.00 Y10.00 F2500.00

G1 X20.00 Y10.00 F2500.00

M300 S50.00 (Servo up)

CONCLUSION

Inkscape is a graphical software which helps us to draw and edit images and here in this project, we used the Inkscape to create the Gcode for the plotter. The picture or the words are converted to Gcode and Pronterface software is used to plot the picture.

ADVANTAGES, DISADVANTAGES AND APPLICATIONS

ADVANTAGES AND APPLICATIONS

1. CNC machines can be used continuously 24 hours a day, 365 days a year and only need to be switched off for occasional maintenance.
2. CNC machines are programmed with a design which can then be manufactured hundreds or even thousands of times. Each manufactured product will be exactly the same. Less skilled/trained people can operate CNCs unlike manual lathes / milling machines etc... which need skilled engineers.
3. CNC machines can be updated by improving the software used to drive the machines Training in the use of CNCs is available through the use of 'virtual software'. This is software that allows the operator to practice using the CNC machine on the screen of a computer. The software is similar to a computer game.
4. CNC machines can be programmed by advanced design software such as Pro/DESKTOP®, enabling the manufacture of products that cannot be made by manual machines, even those used by skilled designers / engineers.
5. Modern design software allows the designer to simulate the manufacture of his/her idea. There is no need to make a prototype or a model. This saves time and money. One person can supervise many CNC machines as once they are programmed they can usually be left to work by themselves. Sometimes only the cutting tools need replacing occasionally.
6. A skilled engineer can make the same component many times. However, if each component is carefully studied, each one will vary slightly. A CNC machine will manufacture each component as an exact match.

Applications

The main applications of CNC machines comes in industrial field. Some of them are discussed below:

1. **Metal Removal Applications**: CNC machines are extensively used in industries where metal removal is required. The machines remove excess metal from raw materials to create complex parts. A good example of this would be the automotive industries where gears, shafts and other complex parts are carved from the raw material. CNC machines are also used in the manufacturing industries for producing rectangular, square, rounded and even threaded jobs. All processes, such as milling, grinding, turning, boring, reaming, etc, can be controlled and carried out by these CNC machines using specific machine tools for each task.
2. **Metal Fabrication Industry**: Many industries require thin plates for different purposes. These industries use CNC machines for a number of machining operations such as plasma or flame cutting, laser cutting, shearing, forming and welding to create these plates. CNC plasma or laser cutters are used for shaping metal, while CNC turret presses are used for operations like punching holes. Other operations like bending metal plates can also be carried out with very high precision using CNC press brakes.
3. **Electrical Discharge Machining Applications**: Electrical Discharge Machines, or EDMs as they are also known, remove metal from the raw material by producing sparks that burn away the excess metal. EDM machining through CNC automation is carried out in two different ways; first through Wire EDM and second through Vertical EDM. CNC automated Wire EDM is used to punch and then die combinations for creating die sets used in the fabrication industry. CNC automated Vertical EDM requires an electrode in the same size and shape as the cavity that needs to be carved out.

DISADVANTAGES

1. CNC machines are more expensive than manually operated machines, although costs are slowly coming down.
2. The CNC machine operator only needs basic training and skills, enough to supervise several machines. In years gone by, engineers needed years of training to operate centre lathes, milling machines and other manually operated machines. This means many of the old skills are been lost.
3. Less workers are required to operate CNC machines compared to manually operated machines. Investment in CNC machines can lead to unemployment.
4. Many countries no longer teach pupils / students how to use manually operated lathes / milling machines etc... Pupils / students no longer develop the detailed skills required by engineers of the past. These include mathematical and engineering skills.

FUTURE SCOPE

In modern CNC systems, end-to-end component design is highly automated using computer aided design (CAD) and computer-aided manufacturing (CAM) programs. The programs produce a computer _le that is interpreted to extract the commands needed to operate a particular machine by use of a post processor, and then loaded into the CNC machines for production. Since any particular component might require the use of a number of different tools - drills, saws, etc., modern machines often combine multiple tools into a single "cell".

In other installations, a number of different machines are used with an external controller and human or robotic operators that move the component from machine to machine. In either case, the series of steps needed to produce any part is highly automated and produces a part that closely matches the original CAD design.

The pen of the machine can be replaced by a laser to make it work like a laser engraving or cutting machine. Engraving machine can be used on wood. The pen can also be replaced with a powerful drill so that it can be used for both milling and drilling purposes. The servo can be replaced with a stepper motor and the pen with a 3-D pen to make it a 3-D printer which can print objects with dimensions. By extrapolation of the axes, the working area of the machine can be extended keeping the algorithm unaltered.

We can use Bluetooth module or a SD card reader to plot instantly and without wired connection.

PCB Mill (Future)

A PCB Mill is a device that etches out a pattern on a copper clad board such that it makes a Printed Circuit Board (PCB). PCBs are used everywhere in the field of electrical engineering to connect electrical components to one another. Typically, after a board is designed, the layout _les are sent to a manufacturer who then makes the board and ships it back to the customer. When prototyping, the delay and setup costs associated with sending a layout to a manufacturer can often mean days of down time. While this may not seem costly at first, it can prove to be a significant nuisance since most boards contain a wiring bug that was overlooked or misunderstood and must then be remade.

Module 3: Programming Section

Program

```
/*

Convert SVG to GCODE with MakerBot Unicorn plugin for Inkscape available

  */


#include <Servo.h>

#include <Stepper.h>



#define LINE_BUFFER_LENGTH 512



// Servo position for Up and Down

const int penZUp = 80;

const int penZDown = 40;



// Servo on PWM pin 6

const int penServoPin = 2;
```

```
// Should be right for DVD steppers, but is not too important here

const int stepsPerRevolution = 20;



// create servo object to control a servo

Servo penServo;



// Initialize steppers for X- and Y-axis using this Arduino pins for the L293D H-bridge

Stepper myStepperY(stepsPerRevolution, 3,4,5,6);

Stepper myStepperX(stepsPerRevolution, 8, 9, 10, 11);



/* Structures, global variables    */

struct point {

  float x;

  float y;

  float z;

};



// Current position of plothead

struct point actuatorPos;
```

```
//  Drawing settings, should be OK

float StepInc = 1;

int StepDelay = 0;

int LineDelay = 50;

int penDelay = 50;



// Motor steps to go 1 millimeter.

// Use test sketch to go 100 steps. Measure the length of line.

// Calculate steps per mm. Enter here.

float StepsPerMillimeterX = 6.0;

float StepsPerMillimeterY = 6.0;



// Drawing robot limits, in mm

// OK to start with. Could go up to 50 mm if calibrated well.

float Xmin = 0;

float Xmax = 40;

float Ymin = 0;

float Ymax = 40;

float Zmin = 0;

float Zmax = 1;
```

```
float Xpos = Xmin;

float Ypos = Ymin;

float Zpos = Zmax;



// Set to true to get debug output.

boolean verbose = false;



//  Needs to interpret

//  G1 for moving

//  G4 P300 (wait 150ms)

//  M300 S30 (pen down)

//  M300 S50 (pen up)

//  Discard anything with a (

//  Discard any other command!



/**********************

 * void setup() - Initialisations

 **********************/

void setup() {
```

```
//  Setup

Serial.begin( 9600 );



penServo.attach(penServoPin);

penServo.write(penZUp);

delay(200);



// Decrease if necessary

myStepperX.setSpeed(20);

myStepperY.setSpeed(20);



//  Set & move to initial default position

// TBD



//  Notifications!!!

Serial.println("Mini CNC Plotter alive and kicking!");

Serial.print("X range is from ");

Serial.print(Xmin);

Serial.print(" to ");

Serial.print(Xmax);
```

```
  Serial.println(" mm.");
  Serial.print("Y range is from ");
  Serial.print(Ymin);
  Serial.print(" to ");
  Serial.print(Ymax);
  Serial.println(" mm.");
}

/************************
 * void loop() - Main loop
 ************************/
void loop()
{
  delay(200);
  char line[ LINE_BUFFER_LENGTH ];
  char c;
  int lineIndex;
  bool lineIsComment, lineSemiColon;

  lineIndex = 0;
```

```
lineSemiColon = false;
lineIsComment = false;


while (1) {


  // Serial reception - Mostly from Grbl, added semicolon support
  while ( Serial.available()>0 ) {
    c = Serial.read();
    if (( c == '\n') || (c == '\r') ) {              // End of line reached
      if ( lineIndex > 0 ) {                         // Line is complete. Then execute!
        line[ lineIndex ] = '\0';                    // Terminate string
        if (verbose) {
          Serial.print( "Received : ");
          Serial.println( line );
        }
        processIncomingLine( line, lineIndex );
        lineIndex = 0;
      }
      else {
        // Empty or comment line. Skip block.
```

```
      }
      lineIsComment = false;
      lineSemiColon = false;
      Serial.println("ok");
    }
    else {
      if ( (lineIsComment) || (lineSemiColon) ) {   // Throw away all comment characters
        if ( c == ')' )  lineIsComment = false;     // End of comment. Resume line.
      }
      else {
        if ( c <= ' ' ) {                           // Throw away whitepace and control characters
        }
        else if ( c == '/' ) {                      // Block delete not supported. Ignore character.
        }
        else if ( c == '(' ) {                      // Enable comments flag and ignore all characters until
')' or EOL.
          lineIsComment = true;
        }
        else if ( c == ';' ) {
          lineSemiColon = true;
```

```
        }
        else if ( lineIndex >= LINE_BUFFER_LENGTH-1 ) {
          Serial.println( "ERROR - lineBuffer overflow" );
          lineIsComment = false;
          lineSemiColon = false;
        }
        else if ( c >= 'a' && c <= 'z' ) {        // Upcase lowercase
          line[ lineIndex++ ] = c-'a'+'A';
        }
        else {
          line[ lineIndex++ ] = c;
        }
      }
    }
  }
}

void processIncomingLine( char* line, int charNB ) {
  int currentIndex = 0;
```

```
char buffer[ 64 ];                    // Hope that 64 is enough for 1 parameter

struct point newPos;



newPos.x = 0.0;

newPos.y = 0.0;



// Needs to interpret

//  G1 for moving

 // G4 P300 (wait 150ms)

//  G1 X60 Y30

 // G1 X30 Y50

//   M300 S30 (pen down)

//  M300 S50 (pen up)

 // Discard anything with a (

//  Discard any other command!



while( currentIndex < charNB ) {

  switch ( line[ currentIndex++ ] ) {          // Select command, if any

  case 'U':

    penUp();
```

```
  break;
case 'D':
  penDown();
  break;
case 'G':
  buffer[0] = line[ currentIndex++ ];          // /!\ Dirty - Only works with 2 digit commands
       buffer[1] = line[ currentIndex++ ];
      buffer[2] = '\0';
  buffer[1] = '\0';

  switch ( atoi( buffer ) ){                  // Select G command
  case 0:                                     // G00 & G01 - Movement or fast movement. Same here
  case 1:
    // /!\ Dirty - Suppose that X is before Y
    char* indexX = strchr( line+currentIndex, 'X' );  // Get X/Y position in the string (if any)
    char* indexY = strchr( line+currentIndex, 'Y' );
    if ( indexY <= 0 ) {
      newPos.x = atof( indexX + 1);
      newPos.y = actuatorPos.y;
    }
```

```
    else if ( indexX <= 0 ) {
      newPos.y = atof( indexY + 1);
      newPos.x = actuatorPos.x;
    }
    else {
      newPos.y = atof( indexY + 1);
      indexY = '\0';
      newPos.x = atof( indexX + 1);
    }
    drawLine(newPos.x, newPos.y );
        Serial.println("ok");
    actuatorPos.x = newPos.x;
    actuatorPos.y = newPos.y;
    break;
  }
  break;
case 'M':
  buffer[0] = line[ currentIndex++ ];        // /!\ Dirty - Only works with 3 digit commands
  buffer[1] = line[ currentIndex++ ];
  buffer[2] = line[ currentIndex++ ];
```

```
buffer[3] = '\0';
switch ( atoi( buffer ) ){
case 300:
 {
  char* indexS = strchr( line+currentIndex, 'S' );
  float Spos = atof( indexS + 1);
          Serial.println("ok");
  if (Spos == 30) {
    penDown();
  }
  if (Spos == 50) {
    penUp();
  }
  break;
 }
case 114:                          // M114 - Repport position
 Serial.print( "Absolute position : X = " );
 Serial.print( actuatorPos.x );
 Serial.print( "  -  Y = " );
 Serial.println( actuatorPos.y );
```

```
        break;
      default:
        Serial.print( "Command not recognized : M");
        Serial.println( buffer );
      }
    }
  }
}
```

```
/*********************************
 * Draw a line from (x0;y0) to (x1;y1).
 * Bresenham algo from https://www.marginallyclever.com/blog/2013/08/how-to-build-an-2-axis-arduino-cnc-gcode-interpreter/
 * int (x1;y1) : Starting coordinates
 * int (x2;y2) : Ending coordinates
 *********************************/
void drawLine(float x1, float y1) {

  if (verbose)
```

```
{
  Serial.print("fx1, fy1: ");
  Serial.print(x1);
  Serial.print(",");
  Serial.print(y1);
  Serial.println("");
}

//  Bring instructions within limits
if (x1 >= Xmax) {
  x1 = Xmax;
}
if (x1 <= Xmin) {
  x1 = Xmin;
}
if (y1 >= Ymax) {
  y1 = Ymax;
}
if (y1 <= Ymin) {
  y1 = Ymin;
```

```
}


if (verbose)
{
  Serial.print("Xpos, Ypos: ");
  Serial.print(Xpos);
  Serial.print(",");
  Serial.print(Ypos);
  Serial.println("");
}


if (verbose)
{
  Serial.print("x1, y1: ");
  Serial.print(x1);
  Serial.print(",");
  Serial.print(y1);
  Serial.println("");
}
```

```
//  Convert coordinates to steps

x1 = (int)(x1*StepsPerMillimeterX);

y1 = (int)(y1*StepsPerMillimeterY);

float x0 = Xpos;

float y0 = Ypos;



//  Let's find out the change for the coordinates

long dx = abs(x1-x0);

long dy = abs(y1-y0);

int sx = x0<x1 ? StepInc : -StepInc;

int sy = y0<y1 ? StepInc : -StepInc;



long i;

long over = 0;



if (dx > dy) {

  for (i=0; i<dx; ++i) {

    myStepperX.step(sx);

    over+=dy;

    if (over>=dx) {
```

```
      over-=dx;
      myStepperY.step(sy);
    }
    delay(StepDelay);
  }
}
else {
  for (i=0; i<dy; ++i) {
    myStepperY.step(sy);
    over+=dx;
    if (over>=dy) {
      over-=dy;
      myStepperX.step(sx);
    }
    delay(StepDelay);
  }
}

if (verbose)
{
```

```
  Serial.print("dx, dy:");
  Serial.print(dx);
  Serial.print(",");
  Serial.print(dy);
  Serial.println("");
}

if (verbose)
{
  Serial.print("Going to (");
  Serial.print(x0);
  Serial.print(",");
  Serial.print(y0);
  Serial.println(")");
}

//  Delay before any next lines are submitted
delay(LineDelay);
//  Update the positions
Xpos = x1;
```

```
  Ypos = y1;
}
//  Raises pen
void penUp() {
  penServo.write(penZUp);
  delay(LineDelay);
  Zpos=Zmax;
  if (verbose) {
    Serial.println("Pen up!");
  }
}
//  Lowers pen
void penDown() {
  penServo.write(penZDown);
  delay(LineDelay);
  Zpos=Zmin;
  if (verbose) {
    Serial.println("Pen down.");
  }
}
```

CONCLUSION

With the increasing demand for small scale high precision parts in various industries, the market for small scale machine tools has grown substantially. Using small machine tools to fabricate small scale parts can provide both flexibility and efficiency. In this thesis, a small scale three axis CNC plotter machine is designed and analyzed under very limited budget.

A plotter is a graphics printer that uses a pen or pencil to draw images. Plotters differ from printers in that plotters use continuous lines to create images while printers use a collection of dots. Like printers, plotters are connected to computers and are used to produce complex images and text. However, plotters are much slower than printers because of the mechanical motion necessary to draw detailed graphics using continuous lines. Architects and product designers use plotters for technical drawings and computer-aided design purposes. Additionally, many garment and sign manufacturers use cutting plotters in which the plotter's pen is replaced with a sharp razorblade.

This project is about building a mechanical prototype of a CNC plotter machine which is able to draw a PCB layout of 4x4 cm (or any image/text) on a given solid surface. It consumes low power and works with high accuracy due to precise controlling of stepper motors. This is a low cost project as compared to other CNC product. It is made with easily available components and spare parts. It is designed for private manufacturing and small scale applications in educational institutes. The machine is designed with a very simple construction scheme and can be carried anywhere without much effort. The algorithm used is simple. The pen can be replaced with a pinhead or laser head or any other tool for different purpose of use. Software that has been used is open source and user-friendly.

The existing CNC machines are of high cost, difficult to maintain and requires highly skilled operators. Our CNC plotter overcomes these problems. It is of low cost and easy to control and there is no need of highly skilled operators. It can be used for long hours at a stretch which is not possible in existing ones. It is hoped to extend this work for future development.

BIBLIOGRAPHY

1. Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi AVR Microcontroller and Embedded Systems, Pearson Education, Inc., 2011.
2. http://www.instructables.com
3. arduino.cc/en/Main/*ArduinoBoardUno*
4. Electronics 4 U magazine
5. wikipedia.org/wiki/*ATmega328p*
6. https://www.quora.com/What-is-the-difference-between-bipolar-unipolar-stepper-motor
7. https://learn.adafruit.com/all-about-stepper-motors/types-of-steppers.

APPENDIX