M. Shifat Hossain
Roll: 1203062
Section: B
Mail: shifataccount@gmail.com
Date of Submission: 6th March 2016

PROJECT REPORT

An Automated CNC PCB Milling Machine

Project supervised by

Dr. Bashudeb Chandra Ghosh
Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology (KUET)
Khulna -9203, Bangladesh. Phone: 041-769471-305/250
Mail: bcg@kuet.ac.bd

TABLE OF CONTENTS

Contents

Abstract ____ 1

Introduction ____ 2

Equipment and Components ____ 5

Construction ____ 7

Functional Description ____ 13

References ____ 16

Abstract

Nowadays it's hard to find a product without electronic components, and they are getting smaller day after day. Consequently the printed circuit boards used on these products are getting more complex and with thinner tracks. There are many ways to produce these boards and one of them is through electrochemical corrosion, but it does not produce good results. A better process is with milling and a CNC machine. There is a lack of CNC machines for this purpose, so this paper presents the development of a CNC milling machine for printed circuit boards with low manufacturing costs for domestic use. The customer requirements are obtained through a market research and then processed with the use of a QFD matrix to acquire the product requirements. A morphological matrix is then used to obtain all possible solutions for each requirement and they are analyzed with an algorithm to find the best concept for this product. The detailed design comprises the 3D model in a CAD system and the selection of materials. Analytic calculations of the critical components and CAE analysis are factors of extreme importance to obtain a precise result. The FMEA allows an analysis of possible flaws in the final product, which can be eliminated during product development. All this to ensure a lean and robust design, which must absorb all vibration from motors and allow the milling of thin tracks. With the detailed design done it is possible to build the first prototype. Individual components are manufactured and then the mechanical part of the machine is assembled, followed by the electronic assembly. With a functional prototype finished many tests are done to assure that all customer requirements were fulfilled. It's also at this stage that the mathematical models used along the development can be validated[1].

Introduction

Numerical control (NC) is the automation of machine tools that are operated by precisely programmed commands encoded on a storage medium, as opposed to controlled manually by hand wheels or levers, or mechanically automated by cams alone[2]. Most NC today is computer (or computerized) numerical control (CNC)[3] in which computers play an integral part of the control.

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 file 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.

A CNC router is a computer controlled cutting machine related to the hand held router used for cutting various hard materials, such as wood, composites, aluminum, steel, plastics, and foams. CNC stands for computer numerical control. CNC routers can perform the tasks of many carpentry shop machines such as the panel saw, the spindle moulder, and the boring machine. They can also cut mortises and tenons.

A CNC router is very similar in concept to a CNC milling machine. Instead of routing by hand, tool paths are controlled via computer numerical control. The CNC router is one of many kinds of tools that have CNC variants.

A CNC router typically produces consistent and high-quality work and improves factory productivity. Unlike a jig router, the CNC router can produce a one-off as effectively as repeated identical production. Automation and precision are the key benefits of CNC router tables.

A CNC router can reduce waste, frequency of errors, and the time the finished product takes to get to market.

INTRODUCTION

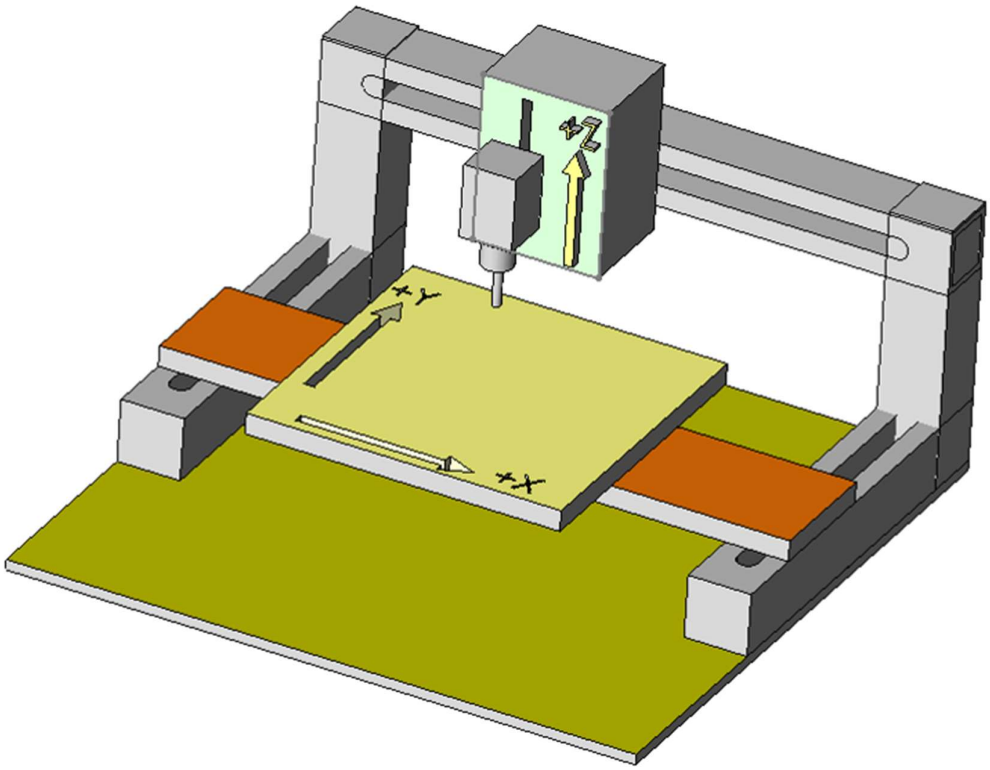


Figure 1: Basic design of a CNC router

Motion is controlled along multiple axes, normally at least two (X and Y)[4] and a tool spindle that moves in the Z (depth). The position of the tool is driven by direct-drive stepper motor or servo motors in order to provide highly accurate movements, or in older designs, motors through a series of step down gears. Open-loop control works as long as the forces are kept small enough and speeds are not too great. On commercial metalworking machines, closed loop controls are standard and required in order to provide the accuracy, speed, and repeatability demanded.

As the controller hardware evolved, the mills themselves also evolved. One change has been to enclose the entire mechanism in a large box as a safety measure, often with additional safety interlocks to ensure the operator is far enough from the working piece for safe operation. Most new CNC systems built today are 100% electronically controlled.

CNC-like systems are now used for any process that can be described as a series of movements and operations. These include laser cutting, welding, friction stir welding, ultrasonic welding, flame and plasma cutting, bending, spinning, hole-punching, pinning, gluing, fabric cutting, sewing, tape and fiber placement, routing, picking and placing, and sawing.

INTRODUCTION

CNC mills use computer controls to cut different materials. They are able to translate programs consisting of specific numbers and letters to move the spindle (or workpiece) to various locations and depths. Many use G-code, which is a standardized programming language that many CNC machines understand, while others use proprietary languages created by their manufacturers. These proprietary languages while often simpler than G-code are not transferable to other machines. CNC mills have many functions including face milling, shoulder milling, tapping, drilling and some even offer turning. Standard linear CNC mills are limited to 3 axis (X, Y, and Z), but others may also have one, or more, rotational axes. Today, CNC mills can have 4 to 6 axes.

Equipment and Components

a.	**Chassis**					Qty.
	1. Steel sheet					
		i) 1mm MS	-	41”x31”	-	1
		ii) 2mm SS	-	12”x24”	-	1
	2. Rack and pinion					
		i) **Pinion** Number of teeth (min) = 30 Dia. (max) = 1” Bore = same as motor shaft	-		-	3
		ii) **Rack** Length = 1ft	-		-	3
	3. Steel rod		-	Dia. = 1 cm Length = 12”	-	2
			-	Dia. = 0.3 cm Length = 10”	-	2
	4. Aluminum Caster Wheel		-	Dia. = 1”	-	4
b.	**Electrical Design**					
	1. Stepper Motor 6 wire hybrid stepper motor Holding torque = 150mN.m Housing = NEMA17 Current rating = 1.2A		-		-	3
	2. Transformer		-	220V:24V 6A	-	1
			-	220V:12V 1A	-	1
	3. LM7805		-		-	1
	4. Capacitor		-	0,1uF	-	2
			-	0.33uF	-	2
			-	22pF	-	4
	5. Push Buttons		-		-	6
	6. Diode		-	1N4007	-	10
	7. ATmega8A		-		-	3
	8. Crystal oscillator		-	16MHz	-	3
	9. Mini Hand Drill		-		-	1

EQUIPMENT AND COMPONENTS

10. LED	-	Red	-	2
	-	Green	-	2
	-	Yellow	-	2
11. Wire	-	25AWG	-	6ft
12. Jumper Wires	-	Male-Male	-	40
	-	Male-Female	-	40
13. MOSFET	-	MTP3N60FI	-	12
14. Transistor	-	BC547	-	12
15. Vero-board	-		-	3
16. Resistor	-	10k	-	
	-	1k	-	
	-	68	-	

Construction

MECHANICAL CONSTRUCTION

The mechanical design of this CNC machine is done using 1mm and 2mm steel sheets for different structural rigidities and different scope of applications. The 1mm steel sheet was used in building the base, plate and attachment mechanisms. The 2mm steel sheet was used in building the sideway support stands. This improves the stability of the design.

This machine is a 3-axis CNC router. The base consists of a single box shaped frame. In which the plate is fixed by wheels (Aluminum Caster Wheel). The plate moves in the Y-axis. The side support stand holds the attachment mechanisms and it provides the movement capability in X-axis.

As the sheets used are of steel, they were arc welded to join them together.

Three motor were attached with this mechanical structures. These motors are of NEMA 17 structure. The sheets were designed as to support the motor screws. The motor were attached with a pinion each. As the pinions can crawl with the rack that is attached with the chassis in three different places.

The attachment mechanism in which the drill machine will be attached is designed so as that can lift the drill machine from the resting position. This is the Z-axis of the machine.

CONSTRUCTION

ELECTRICAL CONSTRUCTION

Electrical construction is the bridge between the software and the mechanical part of this machine. It takes command from the connected computer and sends voltages or currents to drive the motors.

The electrical part of this device is mostly switching mechanism that responds to the direct control through the PC. Arduino is used to control the motors using software.

As this project uses 6 wire stepper motors the wires must be identified before using these motor in practical applications. A stepper motor consists of two coils. In case of 6 wire motors, the 3 wires are from one coil and the remaining 3 are from the other coil.

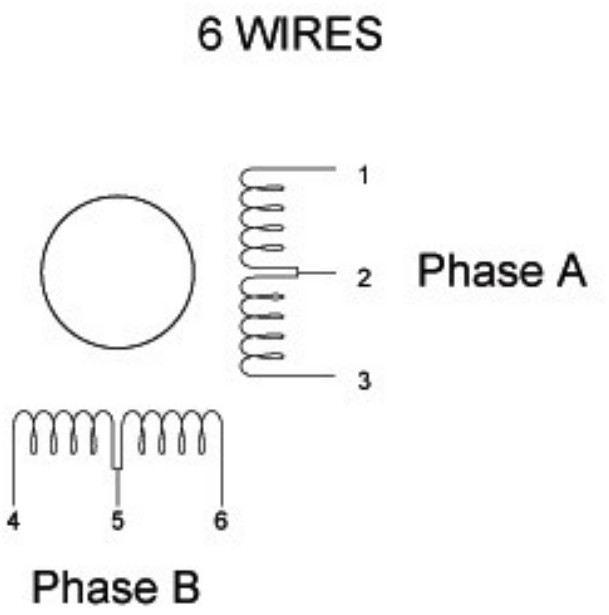


Figure 2: 6-wire hybrid stepper motor connection diagram

Among these wires, two of them are common wires (see figure 2 – the 2nd and 5th wire). To drive a stepper these two wires are connected in +Ve Voltage. And the rest of the wires are connected in ground sequentially.

Now the first work is to find the common wires. To do this step an ohmmeter is used. Now selecting any two wires will or will not give any resistance reading. If the wires does not give any resistance value then it can be concluded that the wires chosen are from different coils.

On the other hand, if those wires gave resistance value then the value should be taken as a note. The keeping any one wire and choosing other wire from the remaining wires. If this time the wires chosen gave same value of resistance then it can be concluded that the wires are from same coil and the wire that is not changed in two consecutive readings is the common wire. Because the resistance between 4-5, 5-6, 1-2 and 2-3 are all the same. So the terminal resistance is the double of single side resistance.

Now the wires are identified it is time for sequence identification. As it is said earlier that except the center or common wires the other 4 wires are connected with ground sequentially. The sequence must be maintained. Otherwise the motor will just vibrate in position.

To identify the wire sequence one of the four wires should be connected with ground constantly. This wire has to be marked as the 4th wire. Now, the three wires should be connected to ground one by one. And it should be noted that, by connecting the three wires one by one the motor rotates slightly (by small degree). It should also be noted that on connecting the wires the motor moves clockwise, counter clockwise and does not move at all. Now the wire causes counter clock-wise rotation should be noted as the 1st wire, the no-rotation is 2nd and the clock-wise rotating wire is the 3rd wire and the 4th wire is already predefined. Now disconnecting the 4th wire from the ground and connecting these wires by using this sequence will cause the motor do a step in a direction. If the motor is required to rotate backward the sequence should be reversed.

Now, to drive these motors by any digital system it needs a driver. A driver is just an arrangement of switches which connect those four wires to ground on the input signal from any digital source (i.e. microcontrollers or microprocessors).

As the motor require about 1.2A to drive in full torque, simple general purpose transistors are not applicable. In this project MOSFETs are used in the driver circuit. But the MOSFETs used (MTP3N60FI) cannot operate in TTL logic voltages. It requires about 12V to saturate. So a TTL to 12V switch is used primarily to operate the MOSFETs. The MOSFETs then control the motor connections.

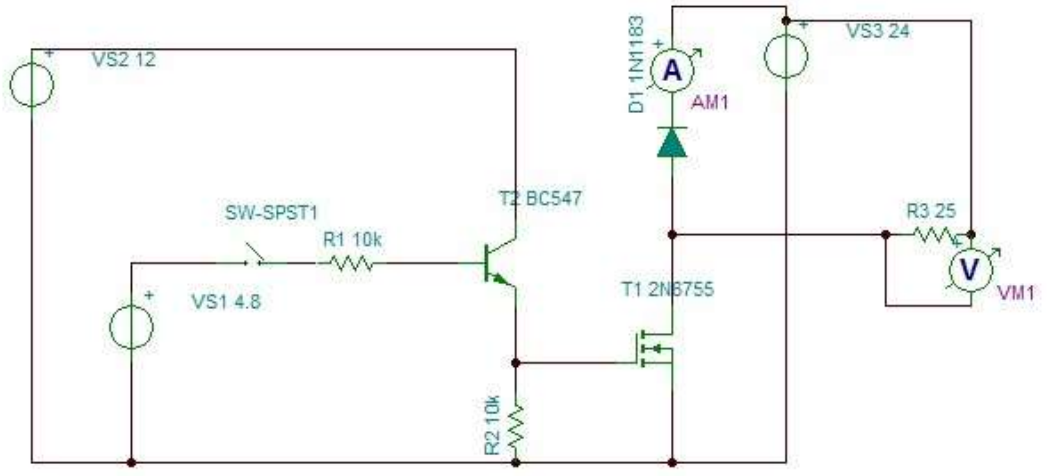


Figure 3: Single unit driver circuit.

CONSTRUCTION

In figure 3 a simulation of the single unit of driver circuit is given. In this circuit MOSFET model is approximated with another closely related enhancement type MOSFET. The SW-SPST1 and VS1 are together representing the logic voltage coming from digital sources. The VS2 is the motor supply. A diode 1N4007 or 1N1183 is connected in reverse direction to the power supply and parallel to the resistor R3. The resistor R3 representing the half coil resistance of stepper motor.

When a digital high is given from the switch SW-SPST1 the output of BC547 transistor saturates and thus MOSFET turns on. Then the MOSFET output taken inversely to get ground connection when the MOSFET saturates. This makes the coils connect to the supply ground.

Each motor consists of 4 of these unit driver circuits. For full circuital diagram see Appendix A and B.

These motor signal in pins from the driver circuit gets connected to the Arduino. The Arduino then gives appropriate signals to drive the motors in the desired directions.

The stepper motor controller can have a current sensing unit. Using this unit the driver can ensure the maximum current is being delivered to the motor and hence ensuring the maximum torque of the motors.

Most of the time PWM signals are used to regulate current flowing through the motor coils. The current is sensed using the current sensing unit of the driver and the PWM signal is generated accordingly.

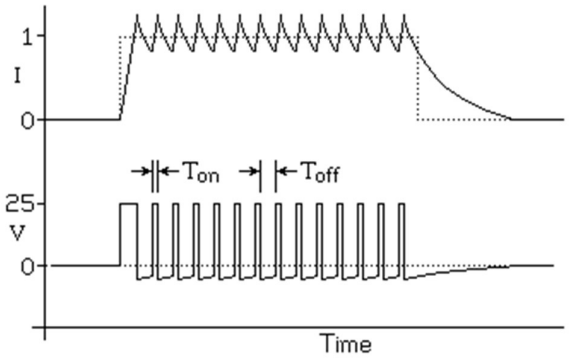


Figure 4: PWM current limiting signal

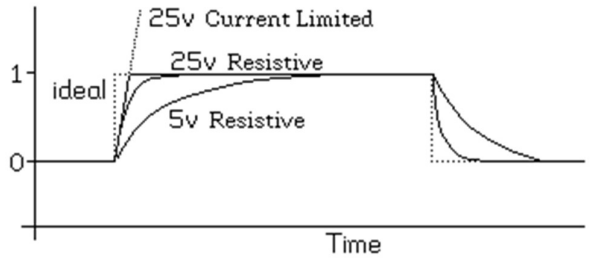
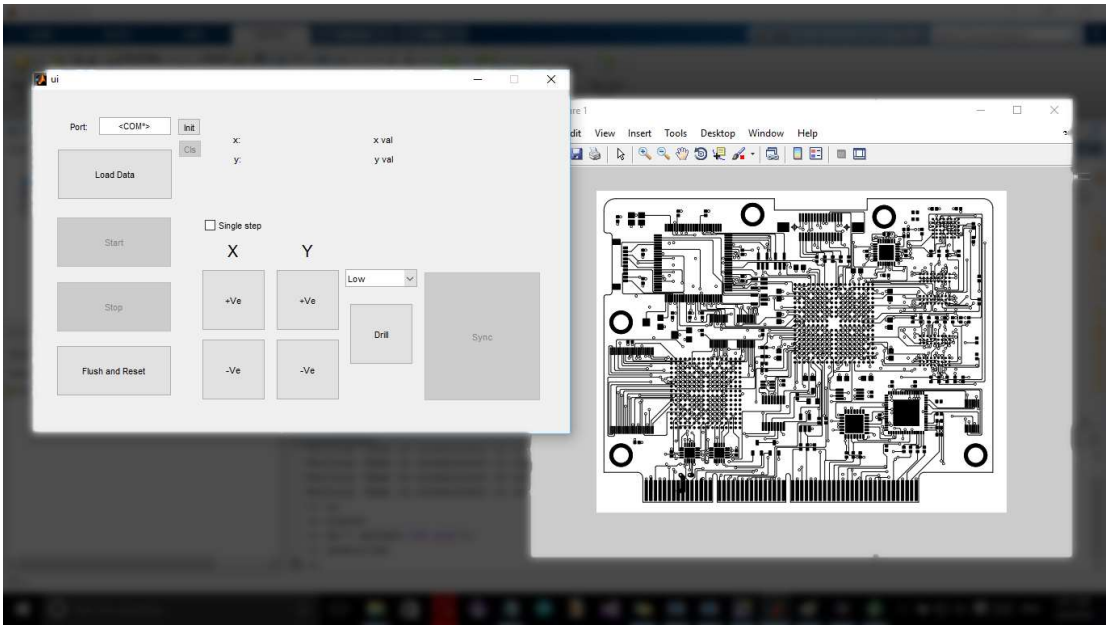


Figure 5: Resistive and transistor based current limiting results

CONSTRUCTION

SOFTWARE CONSTRUCTION



The software part is divided into two parts –

1. PC side
2. Arduino or device side

The Arduino side software is a symbol representing software. Meaning it will not do anything unless the specific run command are given to it. It only knows the sequence and position of the motors and manage them, simplifying the command structures from the PC side application. The Arduino takes PC data or run instructions by using the Serial interface. The serial baud rate is 9600 bps. The symbols defined by the Arduino software is given below:

ARDUINO COMMANDS

PRIMARY COMMAND	SECONDARY COMMAND	DESCRIPTION
1		Move Y-axis to the positive direction by one step
2		Move Y-axis to the negative direction by one step
3		Move X-axis to the positive direction by one step

4		Move X-axis to the negative direction by one step
5	Value (int)	Move Y-axis to the positive direction by the given value
6	Value (int)	Move Y-axis to the negative direction by the given value
7	Value (int)	Move X-axis to the positive direction by the given value
8	Value (int)	Move X-axis to the negative direction by the given value
9	0 or 1 or 2 (int)	Keep drill bit for a predefined time. 0 – 500ms 1 – 1s 2 – 4s These timings are used for different depth of drilling like copper etching or full hole drill
a		Turns off Y-axis supply
b		Turns off X-axis supply
c	h (char)	Continuous positive Y-axis movement. 'h' command after this primary command stops the continuity
d	h (char)	Continuous negative Y-axis movement. 'h' command after this primary command stops the continuity
e	h (char)	Continuous positive X-axis movement. 'h' command after this primary command stops the continuity
f	h (char)	Continuous negative X-axis movement. 'h' command after this primary command stops the continuity

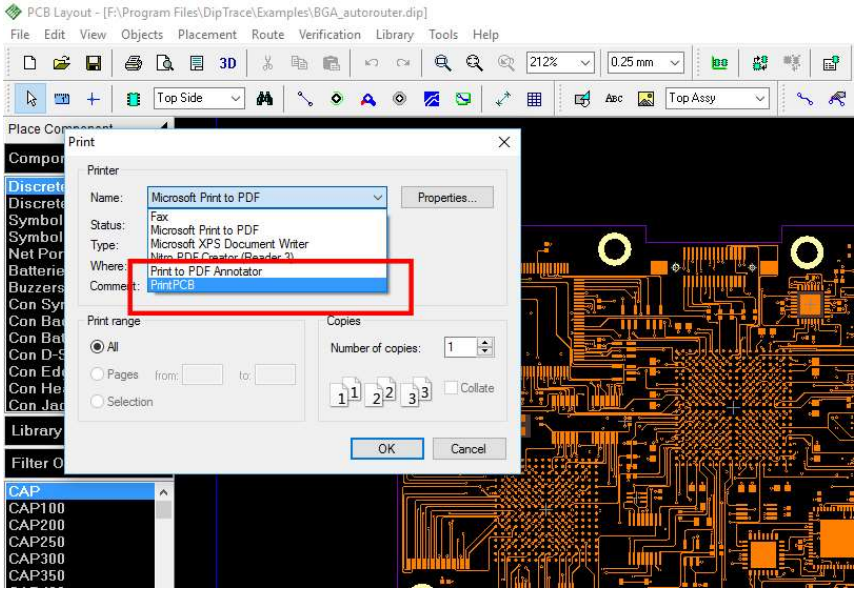
The PC software gives these commands to the Arduino and drives the CNC router. N.B. the data are not generated in Arduino. Instead the PC software generates data from PCB designs given to the software. The software then approximates pixels in real life scale. Then the software gives commands to the CNC router to move different parts of the device. For full Arduino and PC side code see Appendix D and E.

Functional Description

PCB FABRICATION

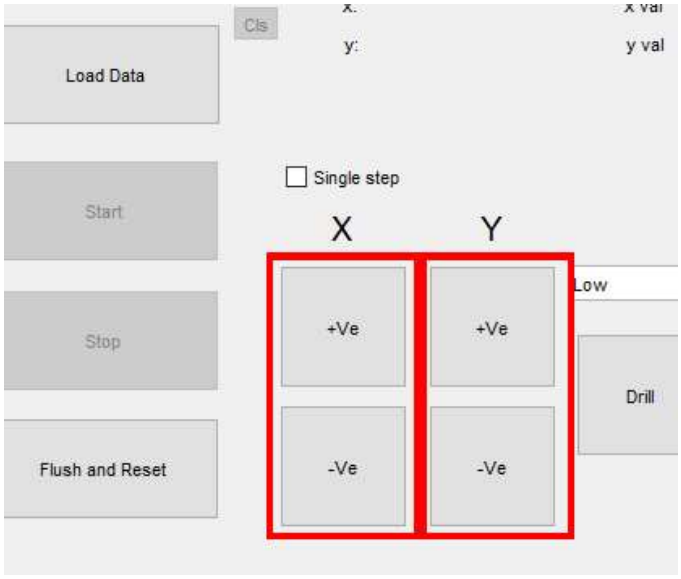
The PCB fabrication pipeline using this CNC PCB milling machine is divided into several sections –

1. Design of a PCB in any CAD or PCB designing software.
2. Print or Export graphics from those CAD applications.
3. If Export of graphics is chosen, then the circuit must be exported with single layer only. No silk layers should be present in the graphics. The graphics must be black and white and not grayscale. As grayscale images will result in error. The file format should be *.jpg, or *.png.
4. If print option is chosen, then the “printPCB” printer must be chosen and the rules applied to the “Export Graphics” option is applicable in this case also.

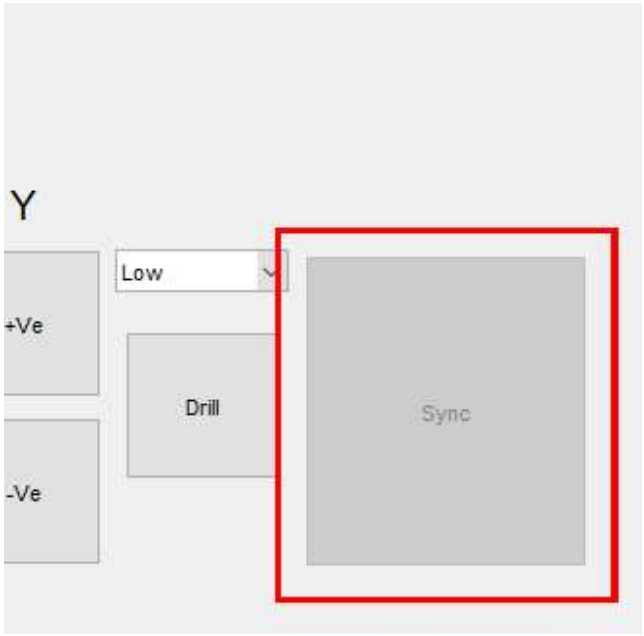


5. Then the CNC PC controller software should be run. And the image should be loaded with the exported graphics content. N.B. the software will run automatically when the PCB is printed using the “printPCB” printer method;
6. Now after loading the image or the printer press the Start button. The router will ask for calibration using a red mark in the figure shown.

7. Then the drill bit should be placed in the marked sign manually by the X and Y buttons.

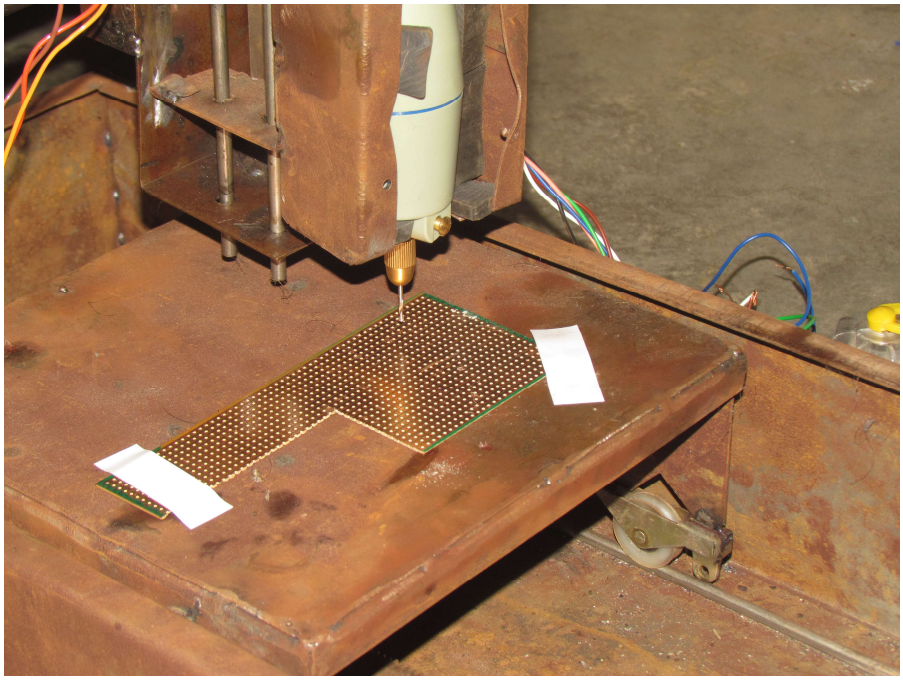


8. Then the Sync button should be pressed.



9. This Records the initial position of the printing. Then the software will again give another point in the drawing to calibrate. Repeat 6-8 steps to calibrate again. This step calculates the relation between the drawing and the real world scale.

10. Now the router or PCB printer will start drawing the PCB over the board.



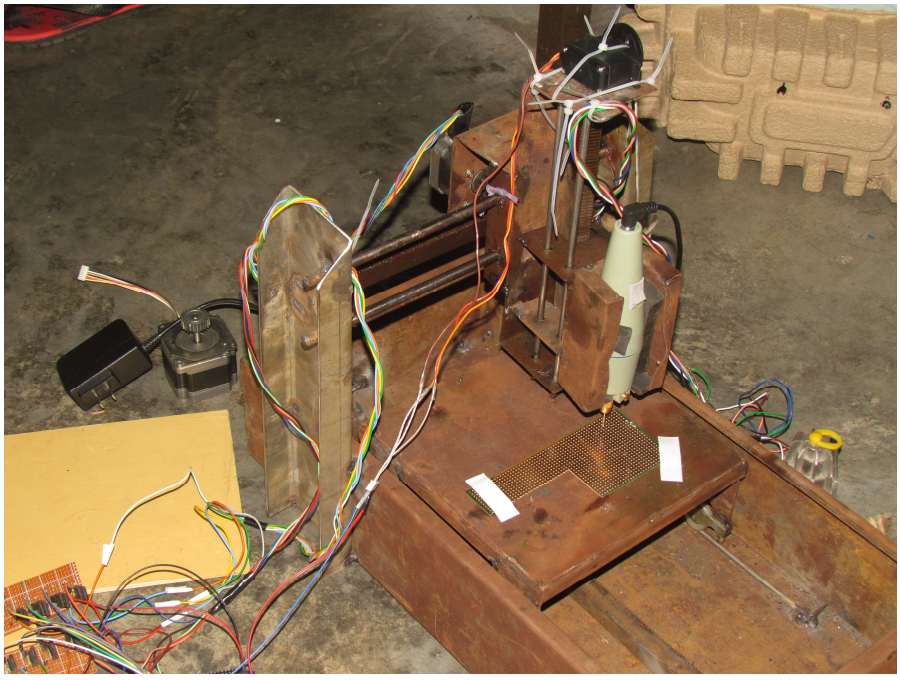


Figure 6: Original photographs of the constructed CNC machine

Reference

[1] Basniak, R., & Fontana Catapan, M. (2012). *DESIGN OF A PCB MILLING MACHINE.* Rio de Janeiro, Brasil: ABCM - Braxilian Society of Mechanical Sciences and Engineering.

[2] *Numerical Control - Wikipedia, the free encyclopedia*. (2016, February 29). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Numerical_control

[3] "Computerized Numerical Control". www.sheltonstate.edu. Shelton State Community College. Retrieved March 24, 2015.

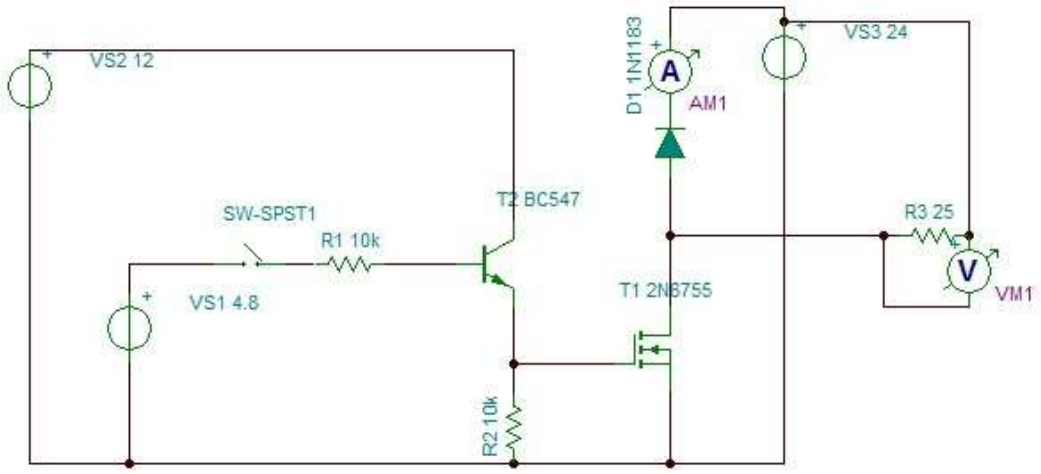
[4] Mike Lynch, "Key CNC Concept #1—The Fundamentals Of CNC", Modern Machine Shop, 4 January 1997. Accessed 11 February 2015


Open
Hardware

APPENDICES

Appendix A

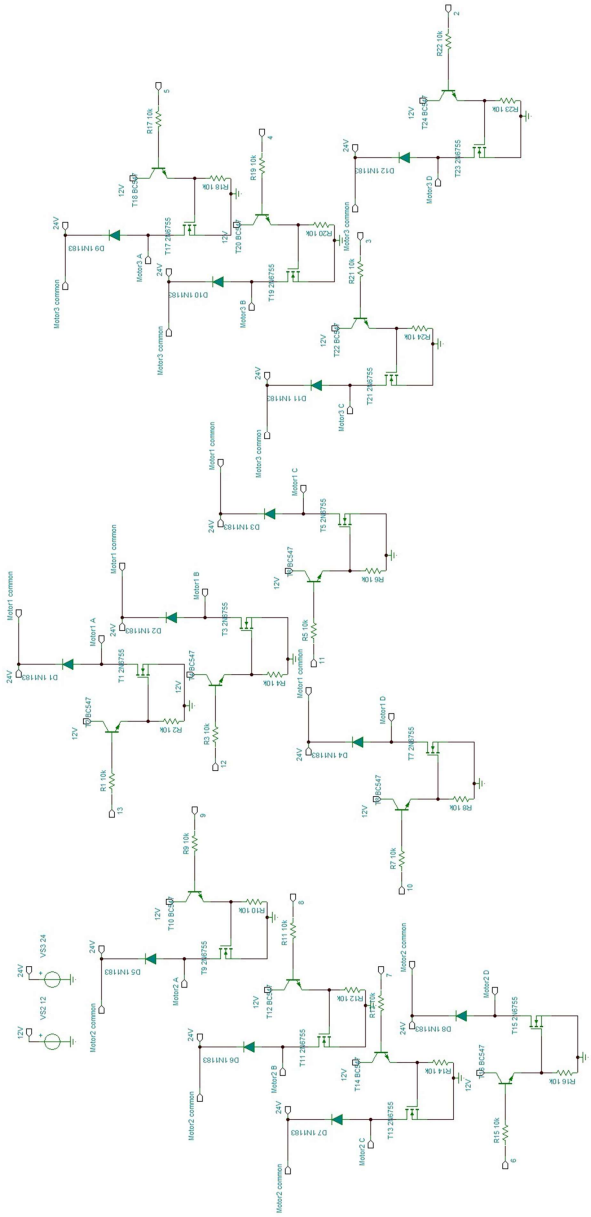
SINGLE MOTOR CIRCUITRY BLOCK



APPENDICES

Appendix B

FULL CIRCUIT DIAGRAM



APPENDICES

Appendix C

COMPONENT DATASHEETS

1. MTP3N60FI
2. Stepper Motor

APPENDICES

Appendix D

ARDUINO CODE

```
#include <Servo.h>

Servo dr;
int tray = 75;
int xgrid = 30;

int sptr = 25;
int spxg = 200;

int stTR = 0;
int stXG = 0;

void setup() {
pinMode(13, OUTPUT);
pinMode(12, OUTPUT);
pinMode(11, OUTPUT);
pinMode(10, OUTPUT);
pinMode(9, OUTPUT);
pinMode(8, OUTPUT);
pinMode(7, OUTPUT);
pinMode(6, OUTPUT);
Serial.begin(9600);
dr.attach(2);
dr.write(5);
rPoTr(20);
```

APPENDICES

```
rNeTr(20);
rPoXg(20);
rNeXg(20);
turnOffTr();
turnOffXg();
Serial.println("Program Started!");
}

char state;
int value;

void loop() {
 if(Serial.available() > 0)
 {
  state = Serial.read();
  Serial.println(state);
  if(state == '1')
  {
   rPoTr(1);
  }
  else if(state == '2')
  {
   rNeTr(1);
  }
  else if(state == '3')
  {
   rPoXg(1);
  }
```

APPENDICES

```
else if(state == '4')
{
 rNeXg(1);
}
else if(state == '5')
{
 while(1)
 {
  value = Serial.parseInt();
  if(value > 0)
  {
   rPoTr(value);
   break;
  }
 }
}
else if(state == '6')
{
 while(1)
 {
  value = Serial.parseInt();
  if(value > 0)
  {
   rNeTr(value);
   break;
  }
 }
}
```

APPENDICES

```
else if(state == '7')
{
 while(1)
 {
  value = Serial.parseInt();
  if(value > 0)
  {
   rPoXg(value);
   break;
  }
 }
}
else if(state == '8')
{
 while(1)
 {
  value = Serial.parseInt();
  if(value > 0)
  {
   rNeXg(value);
   break;
  }
 }
}
else if(state == '9')
{
 while(1)
 {
```

APPENDICES

```
    value = Serial.parseInt();
    if(value >= 1)
    {
     if(value == 1)
     {
       drill(500);
     }
     else if(value == 2)
     {
       drill(1000);
     }
     else if(value == 3)
     {
       drill(2000);
     }
     break;
    }
  }
}
else if(state == 'a')
{
  turnOffTr();
}
else if(state == 'b')
{
  turnOffXg();
}
else if(state == 'c')
```

APPENDICES

```
{
 while(1)
 {
  rPoTr(1);
  if(Serial.read() == 'h')
  {
   turnOffTr();
   break;
  }
 }
}
else if(state == 'd')
{
 while(1)
 {
  rNeTr(1);
  if(Serial.read() == 'h')
  {
   turnOffTr();
   break;
  }
 }
}
else if(state == 'e')
{
 while(1)
 {
  rPoXg(1);
```

APPENDICES

```
      if(Serial.read() == 'h')
      {
        turnOffXg();
        break;
      }
     }
    }
    else if(state == 'f')
    {
     while(1)
     {
      rNeXg(1);
      if(Serial.read() == 'h')
      {
        turnOffXg();
        break;
      }
     }
    }
  }

}

void rPoTr(int steps) // Positive direction -
{
  for(int i=0;i<steps;i++)
  {
    if(stTR == 0)
```

APPENDICES

```
{
 analogWrite(13, 0);
 analogWrite(12, tray);
 analogWrite(11, 0);
 analogWrite(10, 0);
 stTR = 1;
 delay(sptr);
}
else if(stTR == 1)
{
 analogWrite(13, 0);
 analogWrite(12, 0);
 analogWrite(11, tray);
 analogWrite(10, 0);
 stTR = 2;
 delay(sptr);
}
else if(stTR == 2)
{
 analogWrite(13, 0);
 analogWrite(12, 0);
 analogWrite(11, 0);
 analogWrite(10, tray);
 stTR = 3;
 delay(sptr);
}
else if(stTR == 3)
{
```

APPENDICES

```
    analogWrite(13, tray);
    analogWrite(12, 0);
    analogWrite(11, 0);
    analogWrite(10, 0);
    stTR = 0;
    delay(sptr);
   }
  }
}

void rNeTr(int steps) // Negative direction -
{
 for(int i=0;i<steps;i++)
 {
  if(stTR == 0)
  {
   analogWrite(13, 0);
   analogWrite(12, 0);
   analogWrite(11, 0);
   analogWrite(10, tray);
   stTR = 3;
   delay(sptr);
  }
  else if(stTR == 1)
  {
   analogWrite(13, tray);
   analogWrite(12, 0);
   analogWrite(11, 0);
```

APPENDICES

```
    analogWrite(10, 0);
    stTR = 0;
    delay(sptr);
   }
   else if(stTR == 2)
   {
    analogWrite(13, 0);
    analogWrite(12, tray);
    analogWrite(11, 0);
    analogWrite(10, 0);
    stTR = 1;
    delay(sptr);
   }
   else if(stTR == 3)
   {
     analogWrite(13, 0);
     analogWrite(12, 0);
     analogWrite(11, tray);
     analogWrite(10, 0);
     stTR = 2;
     delay(sptr);
   }
  }
}

//////////////////////////////////////////////////////
void rPoXg(int steps) // Positive direction -
{
```

APPENDICES

```
for(int i=0;i<steps;i++)
{
 if(stXG == 0)
 {
  analogWrite(9, 0);
  analogWrite(8, xgrid);
  analogWrite(7, 0);
  analogWrite(6, 0);
  stXG = 1;
  delay(spxg);
 }
 else if(stXG == 1)
 {
  analogWrite(9, 0);
  analogWrite(8, 0);
  analogWrite(7, xgrid);
  analogWrite(6, 0);
  stXG = 2;
  delay(spxg);
 }
 else if(stXG == 2)
 {
  analogWrite(9, 0);
  analogWrite(8, 0);
  analogWrite(7, 0);
  analogWrite(6, xgrid);
  stXG = 3;
  delay(spxg);
```

APPENDICES

```
    }
    else if(stXG == 3)
    {
     analogWrite(9, xgrid);
     analogWrite(8, 0);
     analogWrite(7, 0);
     analogWrite(6, 0);
     stXG = 0;
     delay(spxg);
    }
  }
}

void rNeXg(int steps) // Negative direction -
{
  for(int i=0;i<steps;i++)
  {
    if(stXG == 0)
    {
     analogWrite(9, 0);
     analogWrite(8, 0);
     analogWrite(7, 0);
     analogWrite(6, xgrid);
     stXG = 3;
     delay(spxg);
    }
    else if(stXG == 1)
    {
```

APPENDICES

```
    analogWrite(9, xgrid);
    analogWrite(8, 0);
    analogWrite(7, 0);
    analogWrite(6, 0);
    stXG = 0;
    delay(spxg);
   }
   else if(stXG == 2)
   {
    analogWrite(9, 0);
    analogWrite(8, xgrid);
    analogWrite(7, 0);
    analogWrite(6, 0);
    stXG = 1;
    delay(spxg);
   }
   else if(stXG == 3)
   {
     analogWrite(9, 0);
     analogWrite(8, 0);
     analogWrite(7, xgrid);
     analogWrite(6, 0);
     stXG = 2;
     delay(spxg);
   }
  }
}
```

APPENDICES

```
////////////////////////////////////////////////

void turnOffTr()
{
 analogWrite(13, 0);
 analogWrite(12, 0);
 analogWrite(11, 0);
 analogWrite(10, 0);
}
void turnOffXg()
{
 analogWrite(9, 0);
 analogWrite(8, 0);
 analogWrite(7, 0);
 analogWrite(6, 0);
}

void drill(int del)
{
 dr.write(60);
 delay(del);
 dr.write(5);
}
```

APPENDICES

Appendix E

PC SIDE CONTROLLING SOFTWARE MATLAB CODE

```
function varargout = ui(varargin)
% UI MATLAB code for ui.fig
%      UI, by itself, creates a new UI or raises the existing
%      singleton*.
%
%      H = UI returns the handle to a new UI or the handle to
%      the existing singleton*.
%
%      UI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in UI.M with the given input arguments.
%
%      UI('Property','Value',...) creates a new UI or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before ui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to ui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ui

% Last Modified by GUIDE v2.5 04-Mar-2016 00:05:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ui_OpeningFcn, ...
                   'gui_OutputFcn',  @ui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

APPENDICES

```
% --- Executes just before ui is made visible.
function ui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ui (see VARARGIN)

% Choose default command line output for ui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ui wait for user response (see UIRESUME)
% uiwait(handles.figure1);

set(handles.stp, 'Enable', 'off');
set(handles.str, 'Enable', 'off');
set(handles.sync, 'Enable', 'off');

% --- Outputs from this function are returned to the command line.
function varargout = ui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in ld.
function ld_Callback(hObject, eventdata, handles)
% hObject    handle to ld (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

fname = uigetfile({'*.bmp';'*.png'}, 'Select CNC design');
if fname ~= 0

end
% --- Executes on button press in str.
```

APPENDICES

```
function str_Callback(hObject, eventdata, handles)
% hObject    handle to str (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in xp.
function xp_Callback(hObject, eventdata, handles)
% hObject    handle to xp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mode = get(hObject,'String');
a = evalin('base','ser');
if strcmp(mode, 'Stop')
    fprintf(a,'h');
    set(handles.xn,'Enable', 'on');
    set(handles.yp,'Enable', 'on');
    set(handles.yn,'Enable', 'on');
    set(hObject,'String','+Ve');
else
    if get(handles.SS,'value')
        fprintf(a,'3');
        fprintf(a,'b');
    else
        fprintf(a,'e');
        set(handles.xn,'Enable', 'off');
        set(handles.yp,'Enable', 'off');
        set(handles.yn,'Enable', 'off');
        set(hObject,'String','Stop');
    end
end

% --- Executes on button press in xn.
function xn_Callback(hObject, eventdata, handles)
% hObject    handle to xn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mode = get(hObject,'String');
a = evalin('base','ser');
if strcmp(mode, 'Stop')
    fprintf(a,'h');
    set(handles.xp,'Enable', 'on');
    set(handles.yp,'Enable', 'on');
    set(handles.yn,'Enable', 'on');
    set(hObject,'String','-Ve');
else
    if get(handles.SS,'value')
        fprintf(a,'4');
        fprintf(a,'b');
    else
        fprintf(a,'f');
        set(handles.xp,'Enable', 'off');
        set(handles.yp,'Enable', 'off');
        set(handles.yn,'Enable', 'off');
```

APPENDICES

```
        set(hObject,'String','Stop');
    end
end

% --- Executes on button press in yp.
function yp_Callback(hObject, eventdata, handles)
% hObject    handle to yp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mode = get(hObject,'String');
a = evalin('base','ser');
if strcmp(mode, 'Stop')
    fprintf(a,'h');
    set(handles.xn,'Enable', 'on');
    set(handles.xp,'Enable', 'on');
    set(handles.yn,'Enable', 'on');
    set(hObject,'String','+Ve');
else
    if get(handles.SS,'value')
        fprintf(a,'1');
        fprintf(a,'a');
    else
        fprintf(a,'c');
        set(handles.xn,'Enable', 'off');
        set(handles.xp,'Enable', 'off');
        set(handles.yn,'Enable', 'off');
        set(hObject,'String','Stop');
    end
end

% --- Executes on button press in yn.
function yn_Callback(hObject, eventdata, handles)
% hObject    handle to yn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mode = get(hObject,'String');
a = evalin('base','ser');
if strcmp(mode, 'Stop')
    fprintf(a,'h');
    set(handles.xn,'Enable', 'on');
    set(handles.yp,'Enable', 'on');
    set(handles.xp,'Enable', 'on');
    set(hObject,'String','-Ve');
else
    if get(handles.SS,'value')
        fprintf(a,'2');
        fprintf(a,'a');
    else
        fprintf(a,'d');
        set(handles.xn,'Enable', 'off');
        set(handles.yp,'Enable', 'off');
        set(handles.xp,'Enable', 'off');
        set(hObject,'String','Stop');
    end
```

```
end

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in dr.
function dr_Callback(hObject, eventdata, handles)
% hObject    handle to dr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

a = evalin('base','ser');
fprintf(a,'9');
switch get(handles.popupmenu1,'Value')
    case 1
        fprintf(a,'1');
    case 2
        fprintf(a,'2');
    case 3
        fprintf(a,'3');
    otherwise
end

% --- Executes on button press in stp.
function stp_Callback(hObject, eventdata, handles)
% hObject    handle to stp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in sync.
function sync_Callback(hObject, eventdata, handles)
% hObject    handle to sync (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

APPENDICES

```
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in SS.
function SS_Callback(hObject, eventdata, handles)
% hObject    handle to SS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of SS


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

APPENDICES

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function port_Callback(hObject, eventdata, handles)
% hObject    handle to port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of port as text
%        str2double(get(hObject,'String')) returns contents of port as a
double


% --- Executes during object creation, after setting all properties.
function port_CreateFcn(hObject, eventdata, handles)
% hObject    handle to port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in init.
function init_Callback(hObject, eventdata, handles)
% hObject    handle to init (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = serial(get(handles.port,'String'));
assignin('base', 'ser', a);
b = evalin('base','ser');
fopen(b);
set(handles.pushbutton15,'Enable','on');
set(hObject, 'Enable','off');

% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.init,'Enable','on');
set(hObject, 'Enable', 'off');
a = evalin('base','ser');
fclose(a);
```