QWERTY PEN

PERSONAL WRITING/DRAWING MACHINE

Praveen Gnanasekaran
Vijoy Sunil Kumar

Final Project Report
ECEN 5613 Embedded System Design
December 10, 2016

CONTENTS

1 INTRODUCTION

The QWERTY Pen was taken up as a fun and challenging project which was initially meant to draw alphabets and numbers. It uses the AT89C51RC2 microcontroller running at 11.0592 MHz to drive stepper and servo motors which controls the plotter motion which realizes it on paper.

After the implementation of the 3D printing, the hardware setup complete and numbers and alphabets were drawn, the idea of drawing images using the QWERTY Pen was pursued. Then the project took a completely different path of using bitmaps to draw images on the paper. Pixel chunks of data were fed to the 8051 through UART and the motors were moved accordingly to draw the picture.

1.1 HARDWARE BLOCK DIAGRAM

The picture below gives a brief overview of the various components involved in the project. The PC is connected to the 8051 through RS232 to send bitmap data. Additionally, the user can also manually control the plotter using the 4 buttons and the LCD display interface. Some manual controls include quick draw option, origin set and to start and stop a drawing operation.

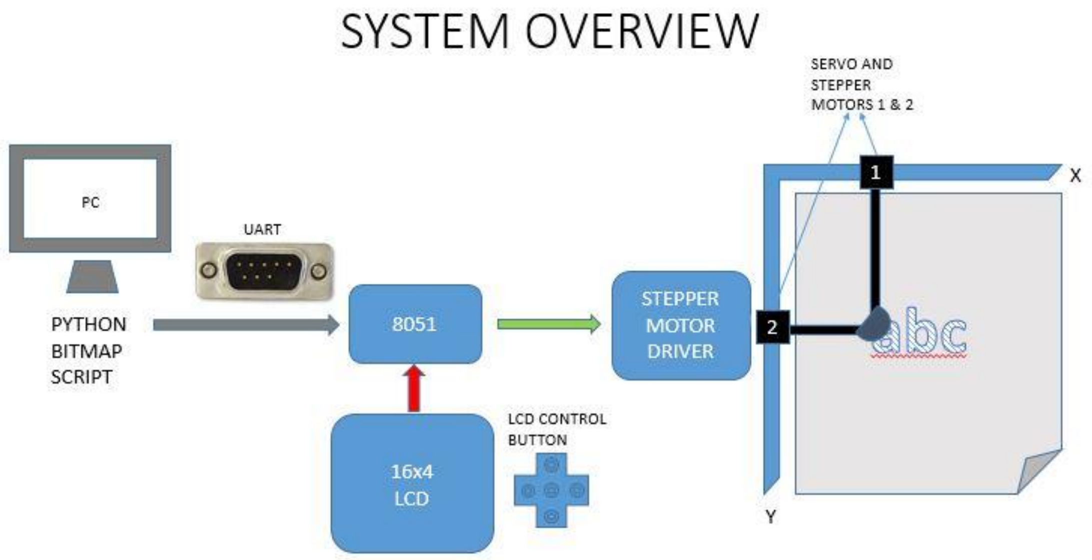


Figure 1 System overview

2 TECHNICAL DESCRIPTION

The following sections detail the design and implementation of the QWERTY Pen. The sections are outlined as follows: Plotter hardware setup includes all new hardware components related to the plotter itself, general hardware descriptions, firmware Design covers 8051 driver code and Software Design describes PC development tools.

2.1 Plotter hardware Setup

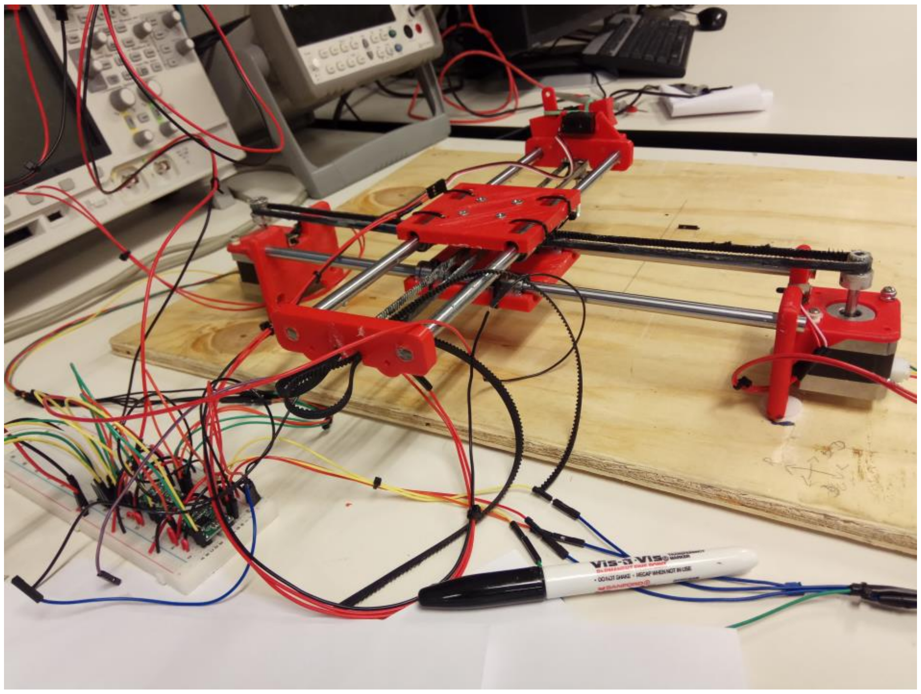


Figure 2 Plotter setup

The picture above shows the plotter arrangement and the stepper motor drivers, attached to the bread board. The parts in red were 3D printed from the STL files (mentioned in the reference)

The rods, motor grooves and the conveyor belt were purchased separately and assembled and velcro strips were used to secure the entire setup to the wooden board. This was vital as the setup vibrates when the stepper motors turn- which led to the figure being drawn being blurred and slant sometimes.

The end 3d prints where the rods are attached were filed to just the right diameter so that the rods were fit tight inside them and didn't move in or out. The conveyor belts were held tight by being fold into two and jammed into the edge of the rod holder as seen in the picture above. Zip ties were also used to secure the wires- which otherwise kept coming in the way of the pen or the rods while in motion.

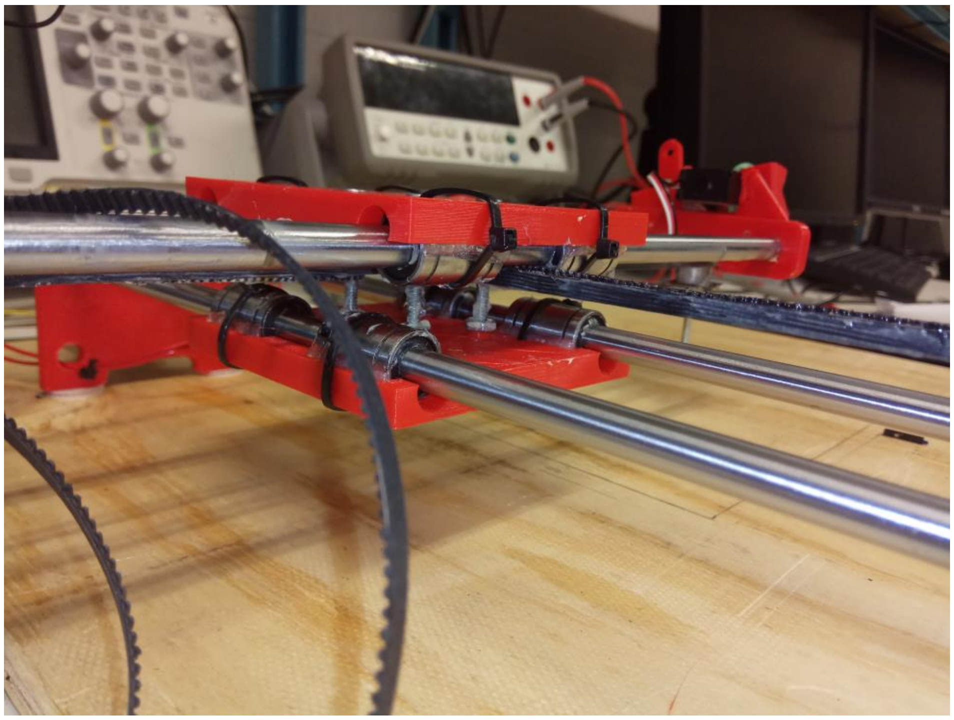


Figure 3 Top and base plates arrangement

The bearings in which the rods slide in and out were lubricated to avoid any friction. The top and bottom plates were hot-glued and secured with zip ties so they don't detach from the rods. Two bolts were used on the screws to keep the top and bottom plates from moving in either direction. This gave some assurance in the pen holder maintaining the same level when the pen moved from one end of the paper to the other.

The picture below shows how the pen holder, servo motor and the belt slider is setup on one end of the setup. Cardboards were kept in place to keep the conveyor belt from falling off the slider. Rubber bands were used to secure the pen to the pen holder. Springs were not used to secure the pen as the right spring with the correct elasticity to hold the pen in place but not pull it up or let it drop too much, was not found for our application. The servo motor was hot glued as well to keep it from moving due to vibrations from the stepper motor and the pen hitting the paper while drawing.

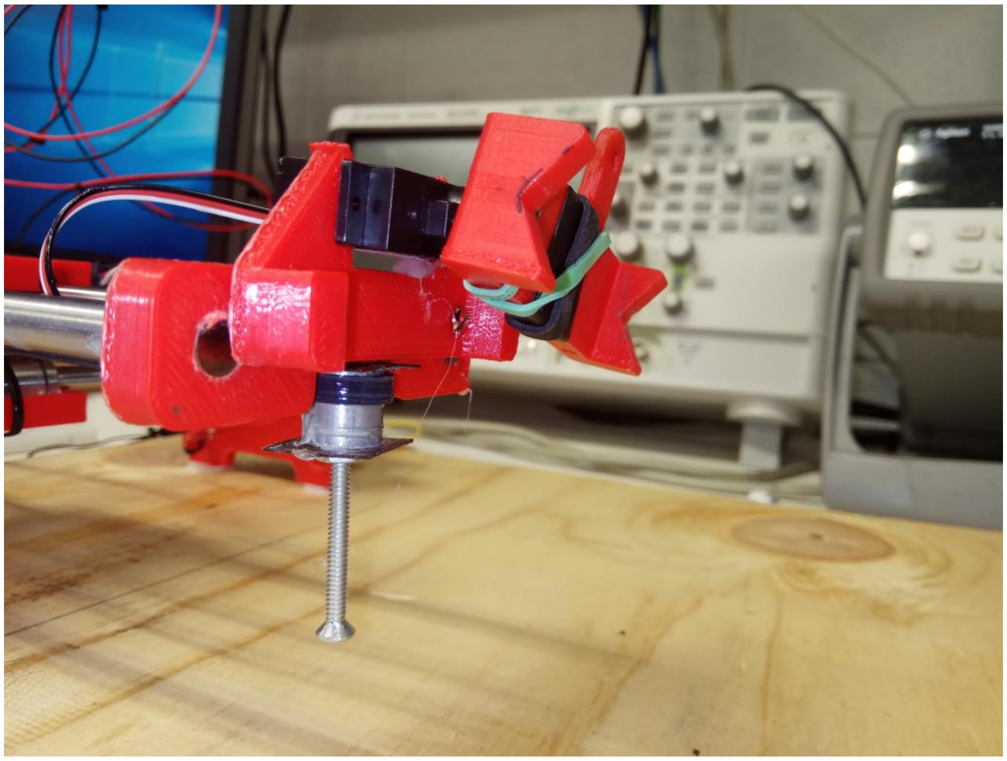


Figure 4 Servo motor and pen holder

2.2 SHORT VIDEO LINK

Click the icon on the right to view project video

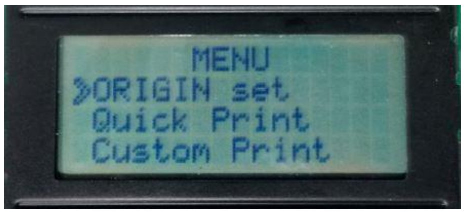
final .avi

2.3 GENERAL HARDWARE DESCRIPTION

Menu control

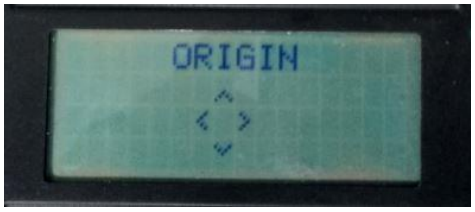
A 16x4 LCD is provided along with 4 push button switches for user control options. The following screens show the different options made available to the user.

Screen 1: Power ON

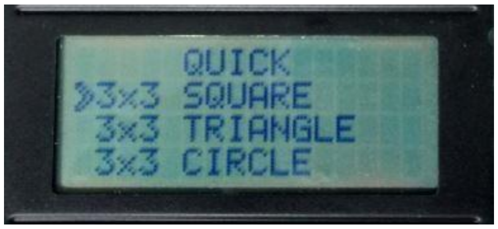


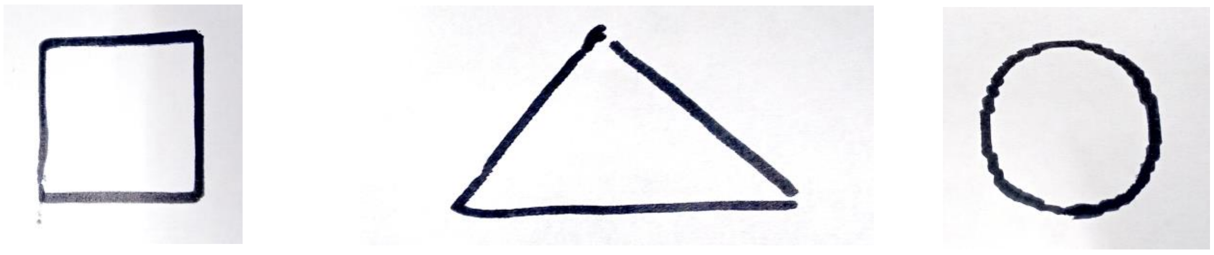
Startup screens

Screen 2: Origin Setting – gives the option to start your print from any point on the paper.



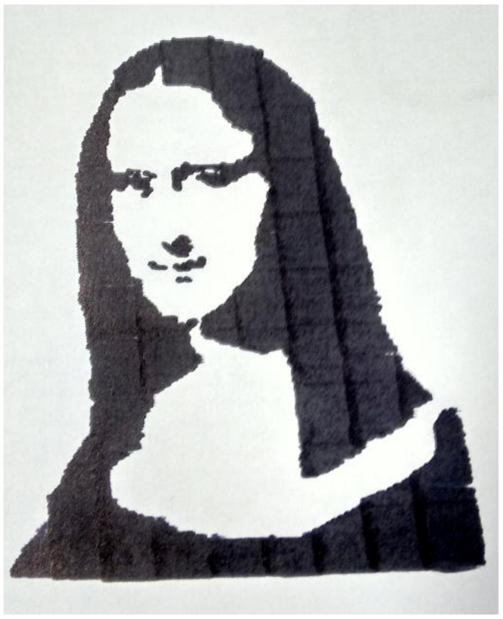
Screen 3: Quick Print – Allows the user to print quick shapes (circle, square or triangle)





PRINTED QUICK SHAPES

Screen 4: Custom Print – On pressing START, an ACKNOWLEDGE byte is sent to the PC connected through UART, which prompts the user to enter a text or an image to print. Pressing STOP can cancel the ongoing print.

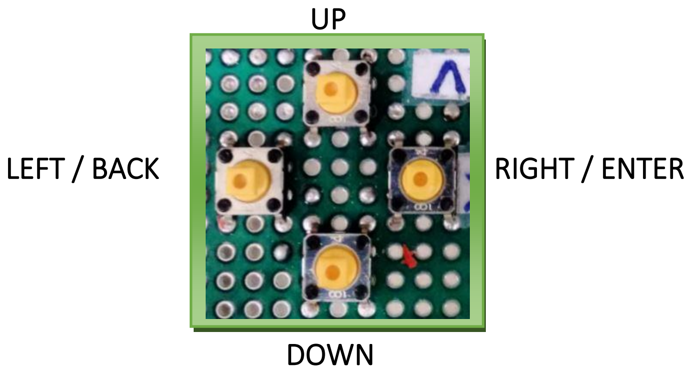

PRINTED IMAGE

ORIGINAL IMAGE

CUSTOM PRINTED IMAGES

Push button switches

There are 4 switches provided for menu control. All 4 of these switches are connected to the PCF8574 I2C I/O Expander.



NEMA 17 Stepper motor

Two of these Stepper motors control the left, right, up and down motion of the pen axis with the help of GT2 16T pulleys, GT2 belt and two 8mm x 300mm linear motion shaft.

Stepper motor wire color code:

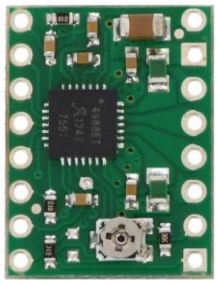
1A	GREEN
1B	YELLOW
2A	GREY
2B	RED

Rods, conveyor belt with grooves and stepper motor used in the plotter

A4988 – Stepper motor driver

The A4988 is a complete micro stepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes.

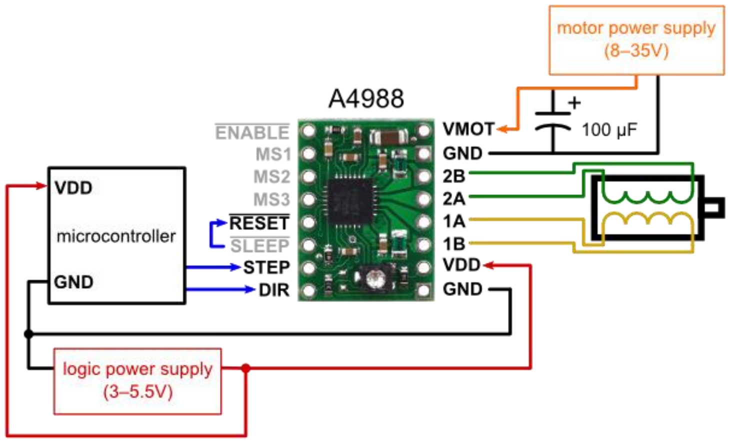
Two of these drivers are used to drive the 2 NEMA 17 stepper motors.



The ENABLE pin is an active low pin and the port pin connected to the ENABLE is low only when the motor has to be activated.

The STEP pin is provided with pulses of 50 % duty cycle and an on time of 15us and the DIR pin controls the direction of rotation. Each pulse to the STEP input corresponds to one micro step of the stepper motor in the direction selected by the DIR pin.

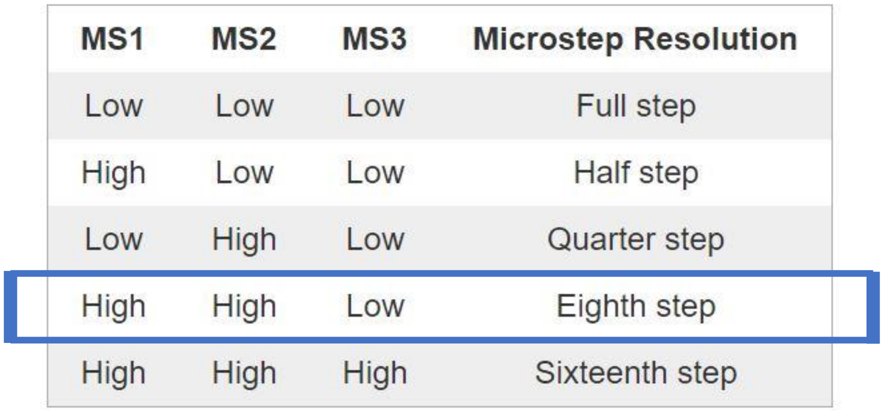
The RESET and the SLEEP pin is not used in this project, so both of these pins are tied. 1A,1B and 2A,2B are connected to the two ends of the two coils in the stepper motor.



STEPPER MOTOR DRIVER CIRCUIT 1

VMOT pin is the motor supply pin and it is supplied with 8V DC from a programmable lab power supply.

MS1, MS2, MS3 pins can be configured to run the stepper motors in any of the below configurations. In this project, MS1 and MS2 are tied to VCC and MS3 is tied to GND, making the stepper resolution to 1/8th of full step (1.8^o)



MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

STEP SIZE OF STEPPER MOTOR 1 AND 2

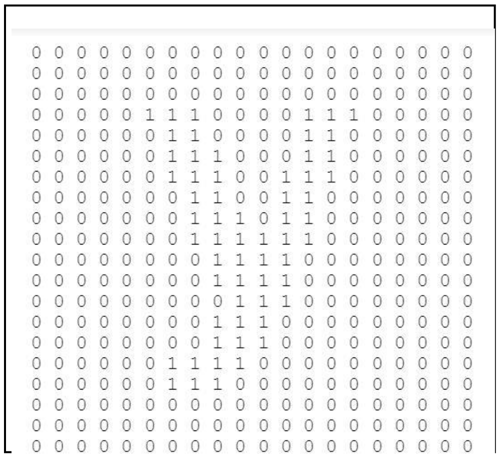
9g A0090 Micro Servo motor

The servo motor is used to move the pen holder up and down depending upon the bitmap array.

The pen holder along with the pen is moved down to make impression on the paper at locations where the bitmap array element is 1 and moved up where the bitmap array element is 0.

MICRO SERVO MOTOR

Sample bitmap array of letter ‘y’

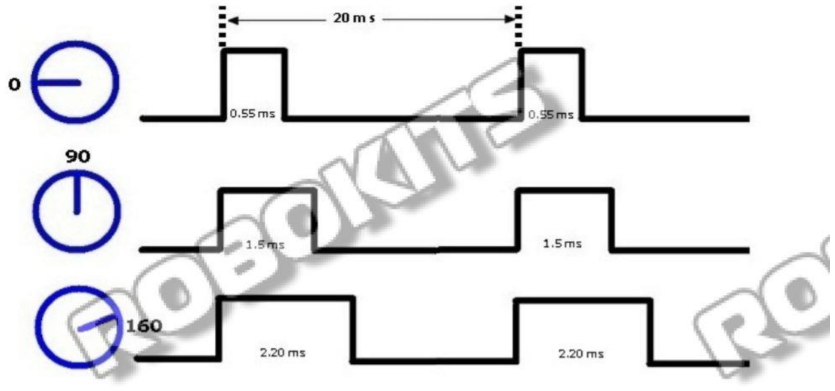


```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

BITMAP IMAGE OF 'y'

The below image shows the pulse timing to control the servo motor rotation.
Pen up position is set at 0 degree and the pen is set to touch down on the paper at 90 degrees.

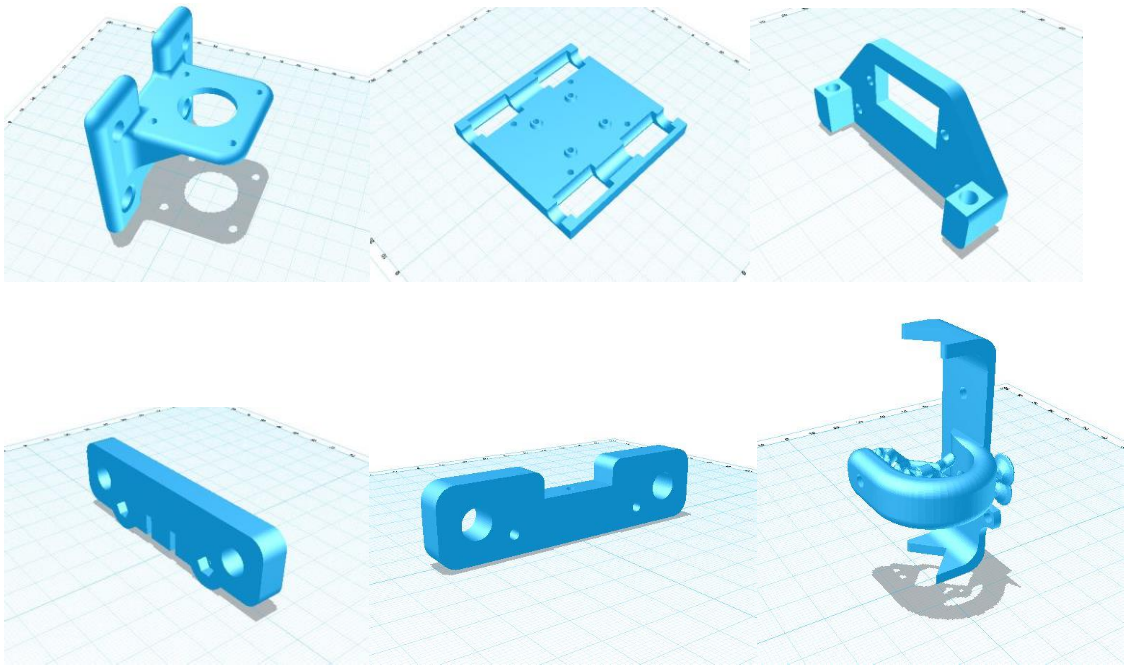
The servo motor is provided a 0.55 ms pulse every 20 ms to turn the servo motor to 0 degrees and a pulse of 1.5 ms every 20 ms to turn to 90 degree position.



MICRO SERVO MOTOR PWM DIAGRAMS

3D printed parts

All the mounts, bases and holders were 3D printed with the help of open source .stl files obtained from the below website. The material used to print them is PLA.



3D print files https://www.youmagine.com/designs/4xidraw

2.4 FIRMWARE DESIGN

Mapping stepper motor motion

At the lower most layer, the following functions were created to move the pen axis in different directions. These functions are called in the upper layers of firmware to draw quick shapes, move to next line and in printing 200 px x 200 px images.

```
void move_left(unsigned int l);
void move_right(unsigned int l);
void move_down(unsigned int l);
void move_up(unsigned int l);
void move_down_right(unsigned int dr);
void move_up_right(unsigned int dr);
void move_up_left(unsigned int dr);
void move_down_left(unsigned int dr);
```

void top_down_right(float rd);
void right_down_left(float rd);
void down_up_left(float rd);
void left_up_right(float rd);

These function are written such that they are calibrated to move in cm units based on the parameter passed.

Layer	files	
Upper layer	shapes.c	shapes.h
Lower layer	stepper_xy.c	steeper_xy.h

Servo motor control

The servo motor control functions are given below. These functions are used to move the pen up and down by providing precise timed pulses to the servo motor

void pen_up(void);
void pen_down(void);

Files	
servo.c	servo.h

LCD menu control

The LCD menu along with the 4 push button switches gives the user an interface to interact with the project.

From powering up the board, setting origin to Printing the 200px x 200px image, we have used numerous custom characters that makes the experience more enjoyable.

The I2C IO expander provided a way to connect up to 4 switches and the data was on which switch was read on expander interrupt.

custom_logo.c	custom_logo.h
expander.c	expander.h
lcd.c	lcd.h
menu.c	menu.h
i2c.c	i2c.h

UART communication

The draw function in text.c takes care of the UART communication with the PC running the python script. After the first packet of 400 bytes is received, the 20x20 block is printed and an ACK signal is sent to the PC for receiving the next 400 bytes.

The packet structure is as below:

Start	row_len	col_len	pixel_size	data[1]	ACK	data[2]	ACK	data[3]	ACK

After receiving each byte, the 8051 acknowledges the byte and this continues until all 400 bytes are received. After printing the first block, another ACK is sent requesting the PC to send the next 400 bytes

The pixel size is either	1: for printing text 20px x 20px
	2: for printing image 200px x 200px

row length and column length are set as 20.

(See Appendix - Firmware Source Code)

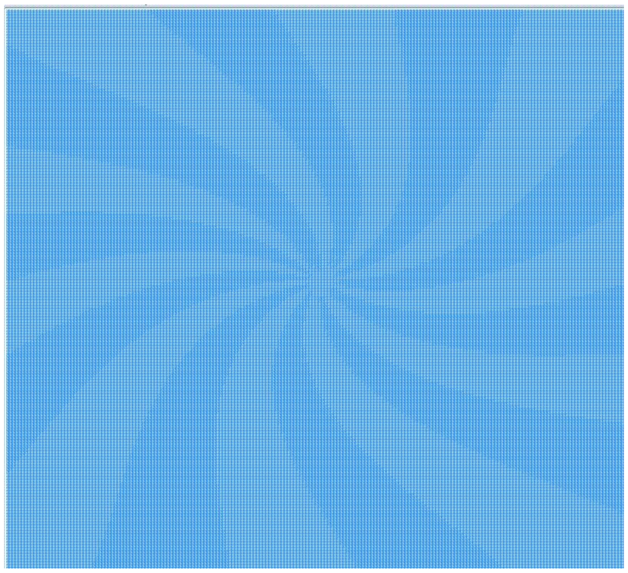
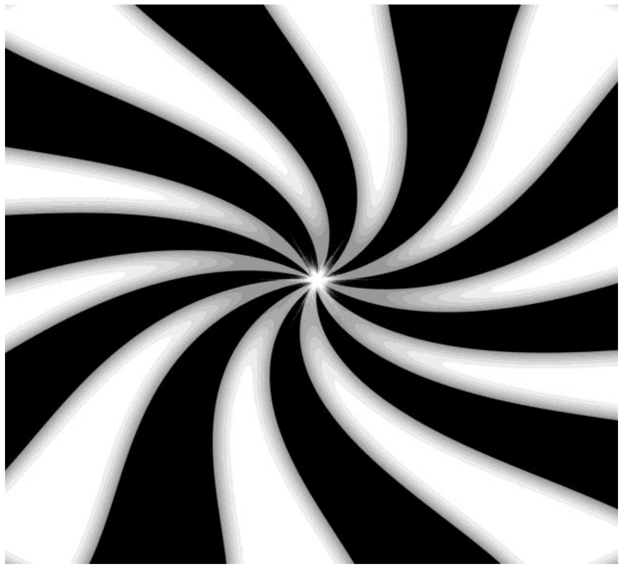
2.5 SOFTWARE DESIGN

For the software design, Python 2.7 was used to write a script that could convert an image into a bitmap array of two dimensions. Some of the notable modules that were used in the script are cv2(computer vision) to read and convert an image into a bitmap array. Pyserial was used to serially send data from the PC to the 8051 microcontroller through the serial port. Numpy was also used as a support module for cv2.

We developed a python script that could scale down a jpeg image into 200 px x 200 px (set in our project) and then convert it into a bitmap array. The script is also designed in a way that it can send the bitmap array serially through UART in groups of 400 bytes (20 px * 20 px) that makes up one small block of the entire image to be printed.

When the user selects the START option in the PRINT menu, the processor AT89C51RC2 sends out a START byte serially that is received by the python script. This tells the script to start sending blocks of 400 bytes data.

The picture below is a bitmap array of a spiral. It consists of zeros and ones, which when scaled down looks like a spiral. This bitmap is sent to the 8051 via the serial port and an actual spiral image is drawn on the paper.

BITMAP CREATED BY THE SCRIPT VS ORIGINAL IMAGE OF A SPIRAL

After the 8051 receives the first 400 bytes, it scans through the array and makes an impression on the paper at locations where the data is ‘1’. After it prints out the entire 400 bytes, the 8051 sends out an ‘ACK’ signal to the PC running the python script which asks for the next block of 400 bytes. This continues until the entire 40000 (200 x 200) bytes are received.

(See Appendix - Software Source Code for python script)

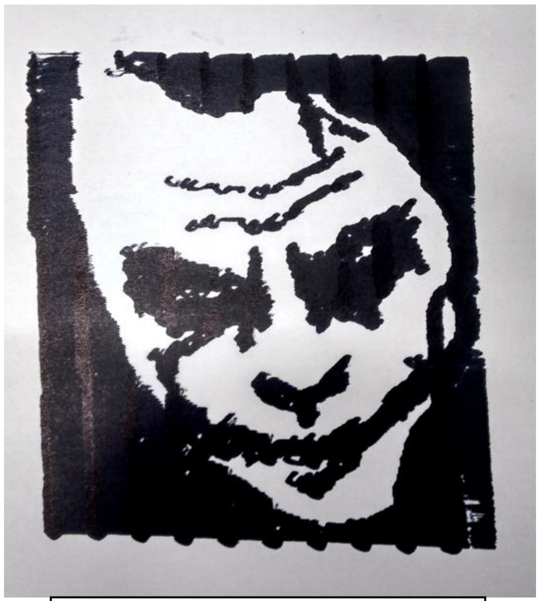
2.6 TESTING PROCESS

We used an incremental development approach to build up the project, starting from making a sturdy base structure to fixing the stepper motor step calibration.

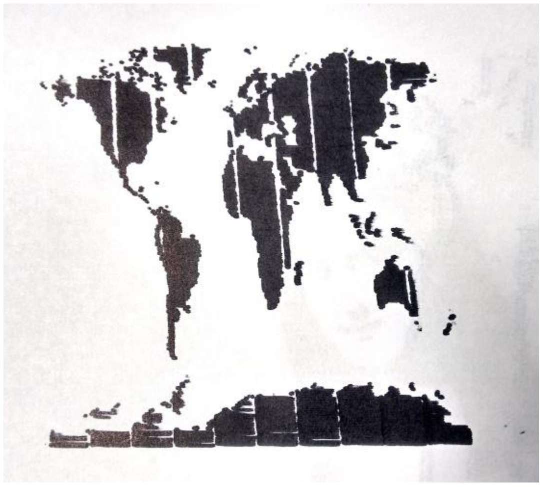
UART communication was tested with small blocks of data (400 bytes) and then tried on with larger block (40000 bytes).

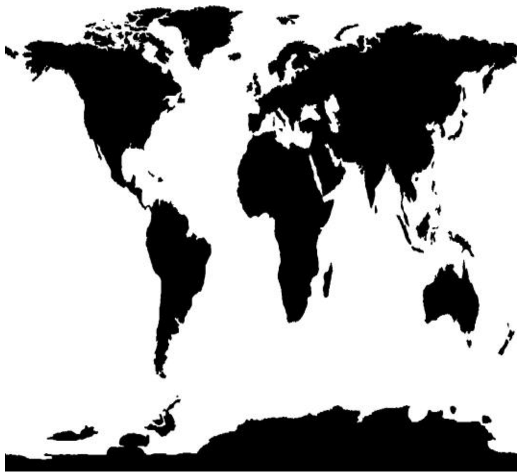
Once the base structure was stable and the communication was tested successfully, the focus was to get a sharper image printed by adjusting the step resolution and also decreasing the print time.

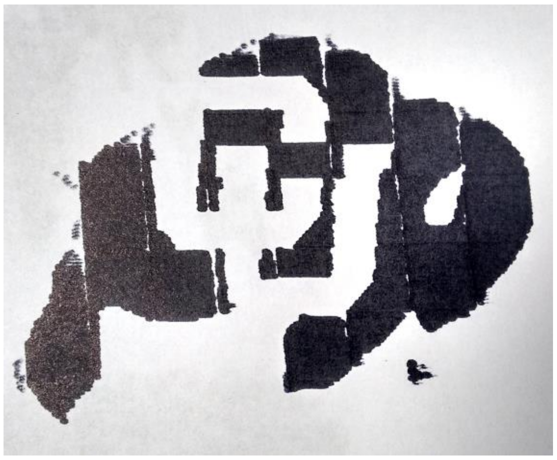
2.7 Final outputs


PRINTED IMAGE

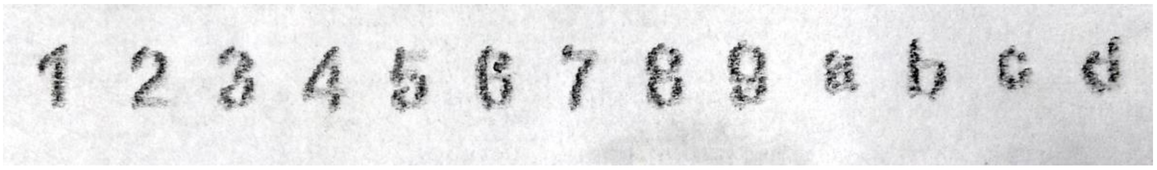
ORIGINAL IMAGE


PRINTED IMAGE


ORIGINAL IMAGE


PRINTED IMAGE

ORIGINAL IMAGE



PRINTED TEXT

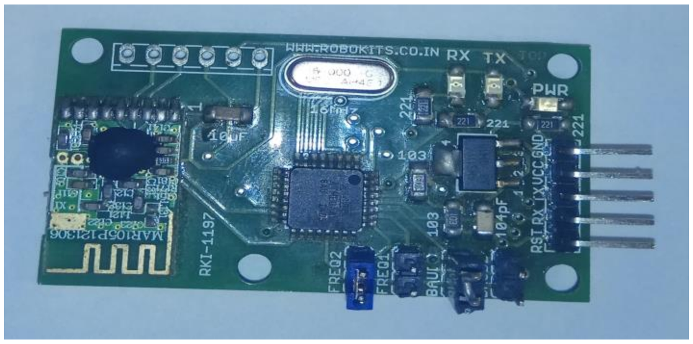
3 RESULTS AND ERROR ANALYSIS

We were able to implement the idea that we initially started out with, so the project was a success. Here are some of the notable roadblocks we faced along the way.

3.1 BLUETOOTH PACKET LOSS

We experienced problems with packet loss of data when using bluetooth. This error was inconsistent in occurrence- for the same code simulation, the error occurred sometimes and didn’t during another trial. We tried increasing the baud rate of transmission and the frequency channel on which the data was transmitted but none of this seemed to fix the problem.

We suspect that the problem might be with the bluetooth module. One future development idea would be to use another bluetooth module to try out the same experiment to check for packet loss.



PICTURE OF THE BLUETOOTH MODULE

3.2 CONVEYOR BELT JERK PROBLEMS

Initially, there was a brief but very noticeable jerk when the stepper motor moved in the up-down direction. This resulted in the pen drawing unsymmetrically and in some cases, the belt not moving at all. We assumed it was with the conveyor belt being too wide or the stepper motor grooves being at a higher position in the structure.

To tackle this, we tried reducing the speeds of the motor belt movements. Later, we thought the problem could be because of lack of a lubricating element between the motor joints and the belt. We tried applying lubricating oil and the performance increased drastically. The jerks were no longer felt. The problem with lubricating oil was that it dried out very soon and had to be reapplied constantly. We finally fixed this issue by using “White Lithium Grease” to lubricate our setup.

4 CONCLUSION

When deciding to implement a mechanical plotter for our final project, we wanted it to not just draw letters and numbers but for it to be able to draw any picture. We faced a lot of roadblocks along the way with the mechanical implementation of the structure, getting the motor resolution and step size correct and ideas for bitmap implementation.

We overcame these difficulties and learnt a lot of new skills along the way. We found a way around some problems (like the conveyor belt jerking, drawing surface not perfectly flat, etc.) and tackled it with different approaches till we found the one that worked.

We realized implementing a mechanical structure in a project idea is bit difficult but by planning ahead and starting to experiment early, it is achievable.

5 FUTURE DEVELOPMENT IDEAS

- Implement different shades of grey- by varying the servo motor's tilt, the impression the pen makes on the paper can be controlled. This can be used to control the level of darkness the pen imprints on the paper. Pictures with more contrast can be drawn with this method.

- Implement wireless connectivity through Bluetooth- this way the user can select an image from a phone and the Qwerty Pen could plot the image on the plotter.

- Increase the picture resolution- by increasing the pixel map for the bitmap from 200x200 to something higher, the resolution of the picture can be increased.

- Restart from a point on power failure- by storing the pixel location flag on a non-volatile on board memory such as the EEPROM or the NVRAM, the plotter can resume printing from the point where it had left off before the power loss. This can be implemented by writing to the non-volatile after every pixel map and reading from it before the next pixel chunk is drawn.

- Use of serial ISR to receive and store data- from the PC to 8051, serial ISR (with a FIFO array) can be used instead of polling to receive every 200 bytes.

- Increasing stepper motor step to size to 1 step (from 1/8th of a step which is currently used) per every PWM pulse- this would decrease the time it takes to draw a picture.

6 ACKNOWLEDGEMENTS

We would like to thank our Professor Linden McClure and all the TAs Anish, Meher and Virag for helping us out through the semester and clarifying all our doubts and suggesting ideas for our project.

We would also like to thank our fellow student, Nestor Lobo, who helped us setup the initial Python scripts and suggesting bit map libraries that we could use in our project.

We would like to acknowledge the Integrated Teaching and Learning Laboratory (ITLL) and Blow Things Up (BTU) lab of the University of Colorado Boulder for letting us borrow a lot of the parts we needed during the initial testing which allowed us to experiment and find the right component.

We would also like to acknowledge the author of the 4xidraw project, “misan” for giving us an idea of the hardware setup and providing us with the 3d printing files to help us build the mechanical structure of the plotter.

7 REFERENCES

1) “3D printing stl files and assembly reference” http://www.instructables.com/id/4xiDraw/
2) “How servo motors work” http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html
3) “Servo motor interfacing with 8051” http://circuitdigest.com/microcontroller-projects/servo-motor-interfacing-with-8051
4) “Sparkfun Micro servo motor” https://www.sparkfun.com/products/9065
5) “Servo Motor datasheet” http://cdn.sparkfun.com/datasheets/Robotics/Small%20Servo%20-%20ROB-09065.pdf
6) “What is a stepper motor” https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor
7) “NEMA 17 stepper motor” https://www.adafruit.com/product/324
8) “Stepper motor driver datasheet” https://www.pololu.com/file/0J450/A4988.pdf
9) “RF 2.4GHz Bluetooth datasheet” http://robokits.download/downloads/RF2_4GHzSerialLink.pdf

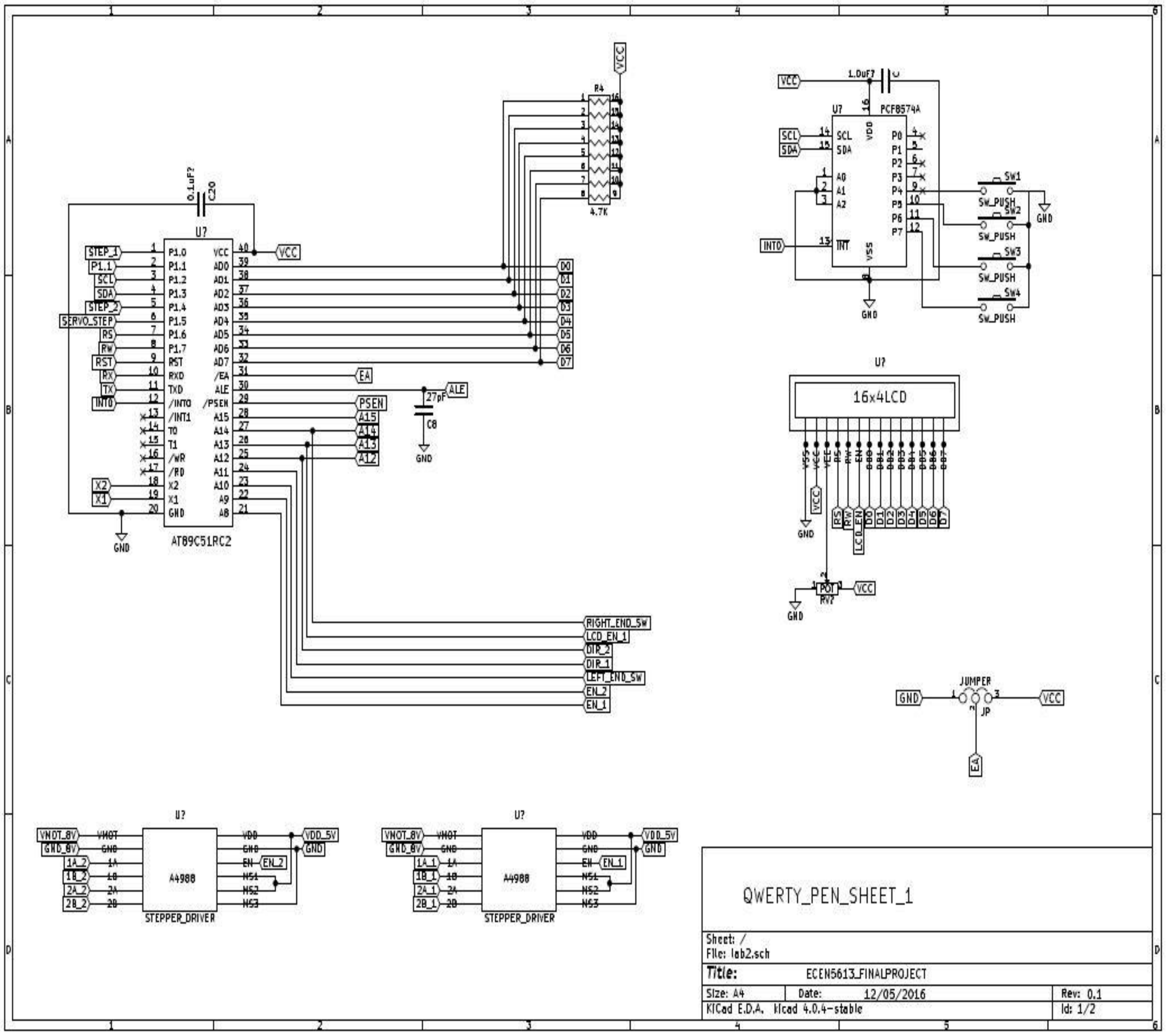
8 APPENDICES

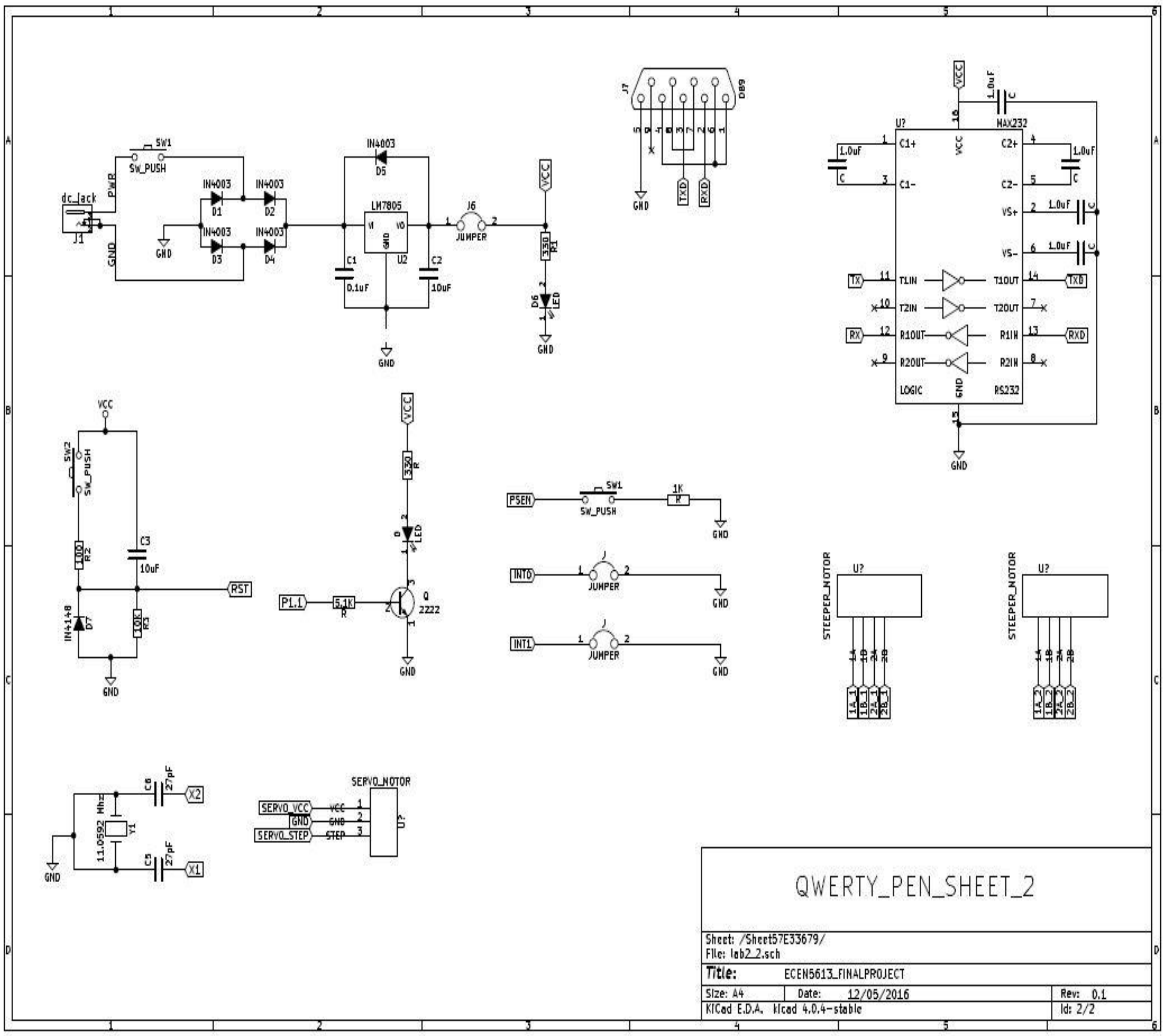
Several appendices have been attached to this report in the order shown below.

8.1 Appendix - Bill of Materials

Part Description	**Source**	**Quantity**	**Cost/unit**
AT89C51RC2	Embedded Systems Lab Kit	1	$8.00
TI PCF8574 I2C I/O Expander	Embedded Systems Lab Kit	1	$2.00
Stepper Motor NEMA17	www.reprapchampion.com	2	$8.49
A4988 Stepper driver	www.reprapchampion.com	2	$2.39
A0090 9g Servo Motor	www.sparkfun.com	1	$8.00
16x4 LCD	Embedded Systems Lab kit	1	$8.00
MAX232	Embedded Systems Lab kit	1	$1.00
PLA for 3D print	ITLL	1	$25.00
8mm x 300mm linear motion shaft	www.reprapchampion.com	4	$5.19
GT2 16T pulleys	www.reprapchampion.com	2	} $8.49
GT2 belt	www.reprapchampion.com	2m	}
LM8UU linear motion bearing	www.amazon.com	8	$0.80
Push button switch	Embedded Systems Lab kit	4	$0.05
7805T voltage regulator	Embedded Systems Lab kit	1	$0.25
DE9 Serial connector	Embedded Systems Lab kit	1	$1.00
TOTAL			**$110.86**

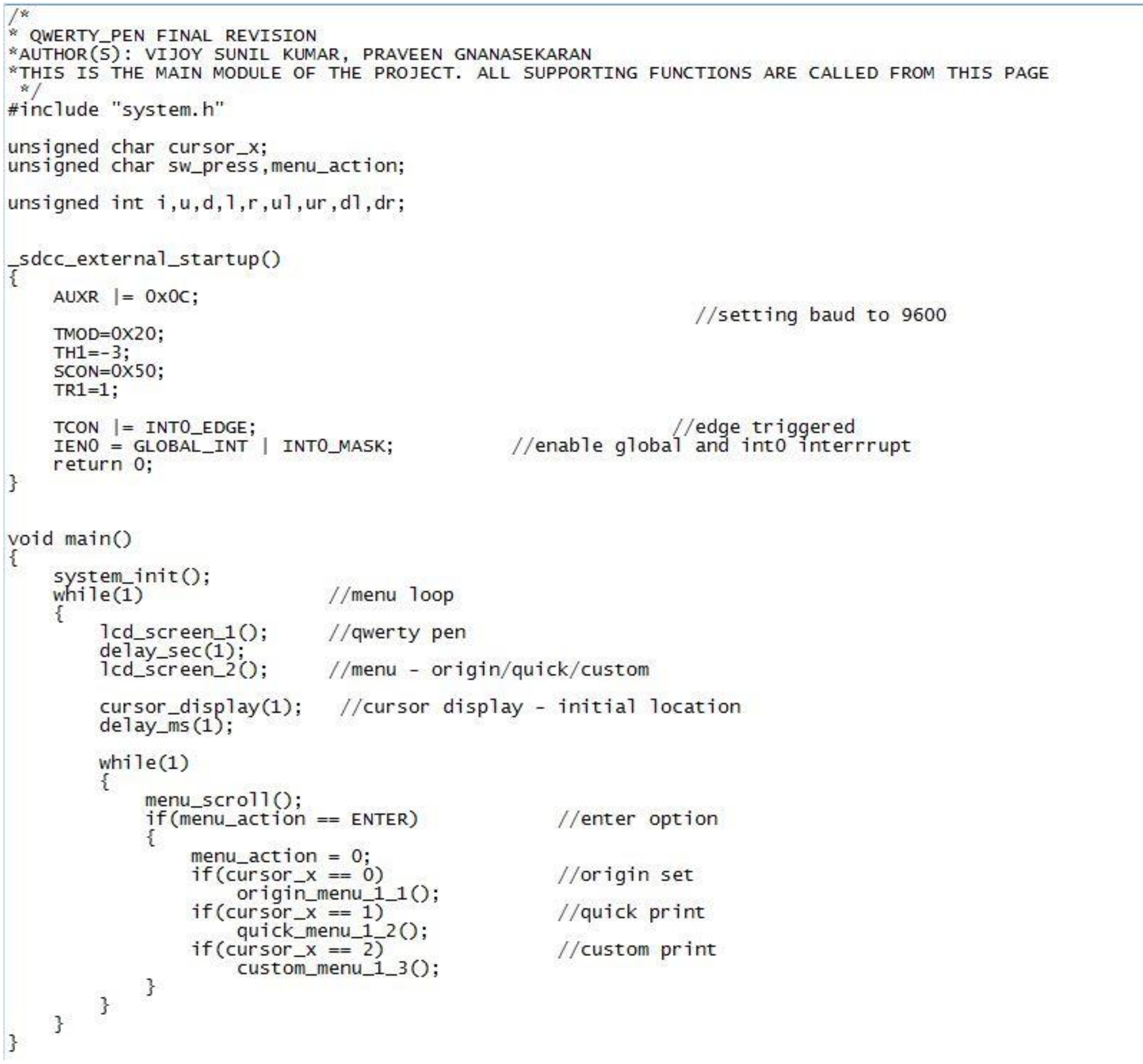
8.2 Appendix – Schematics





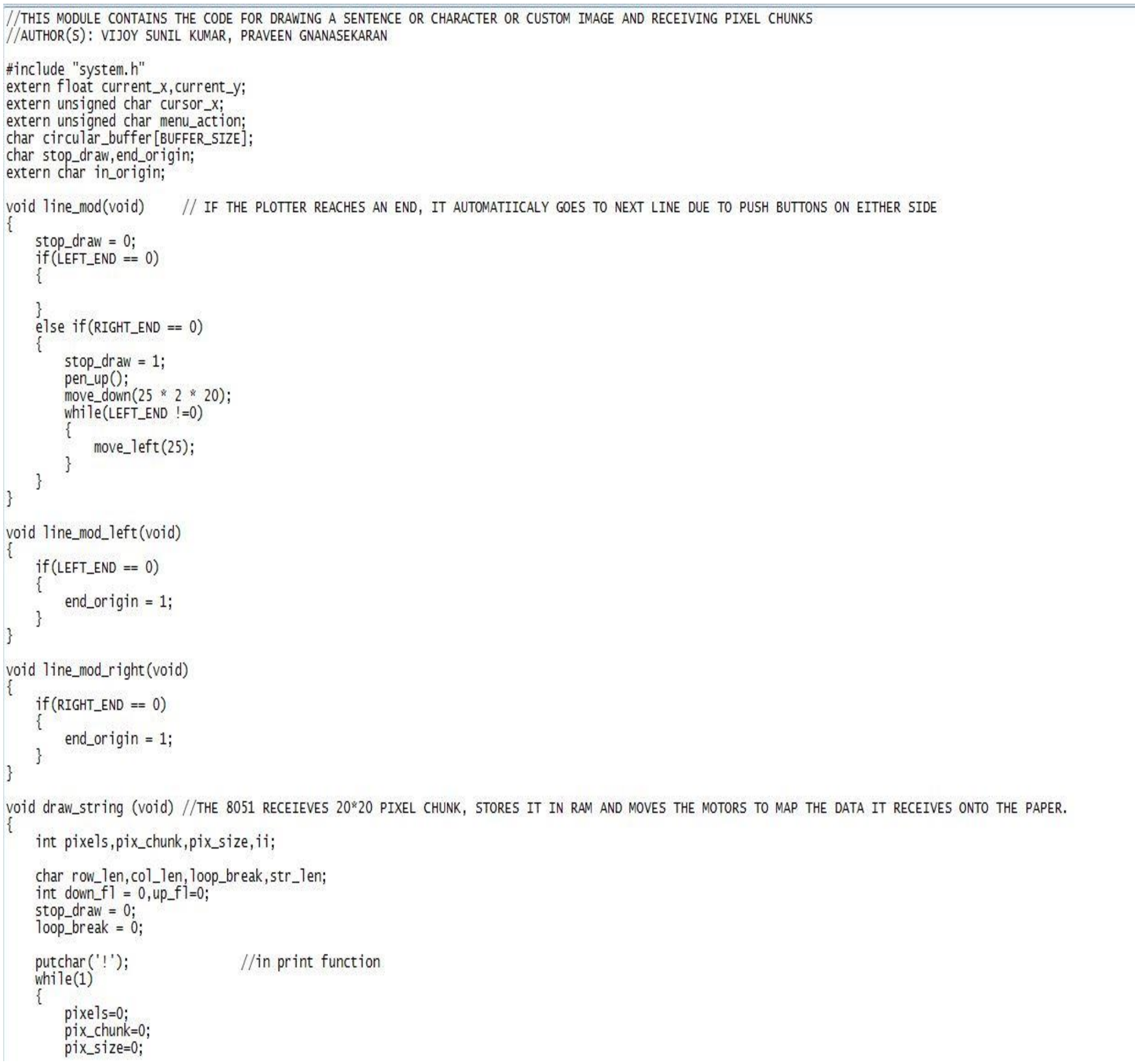
8.3 Firmware Source Code

IDE : CodeBlocks
Version : 12.11
OS : Windows 10
Given below are some snippets from the firmware running on the 8051 microcontroller.



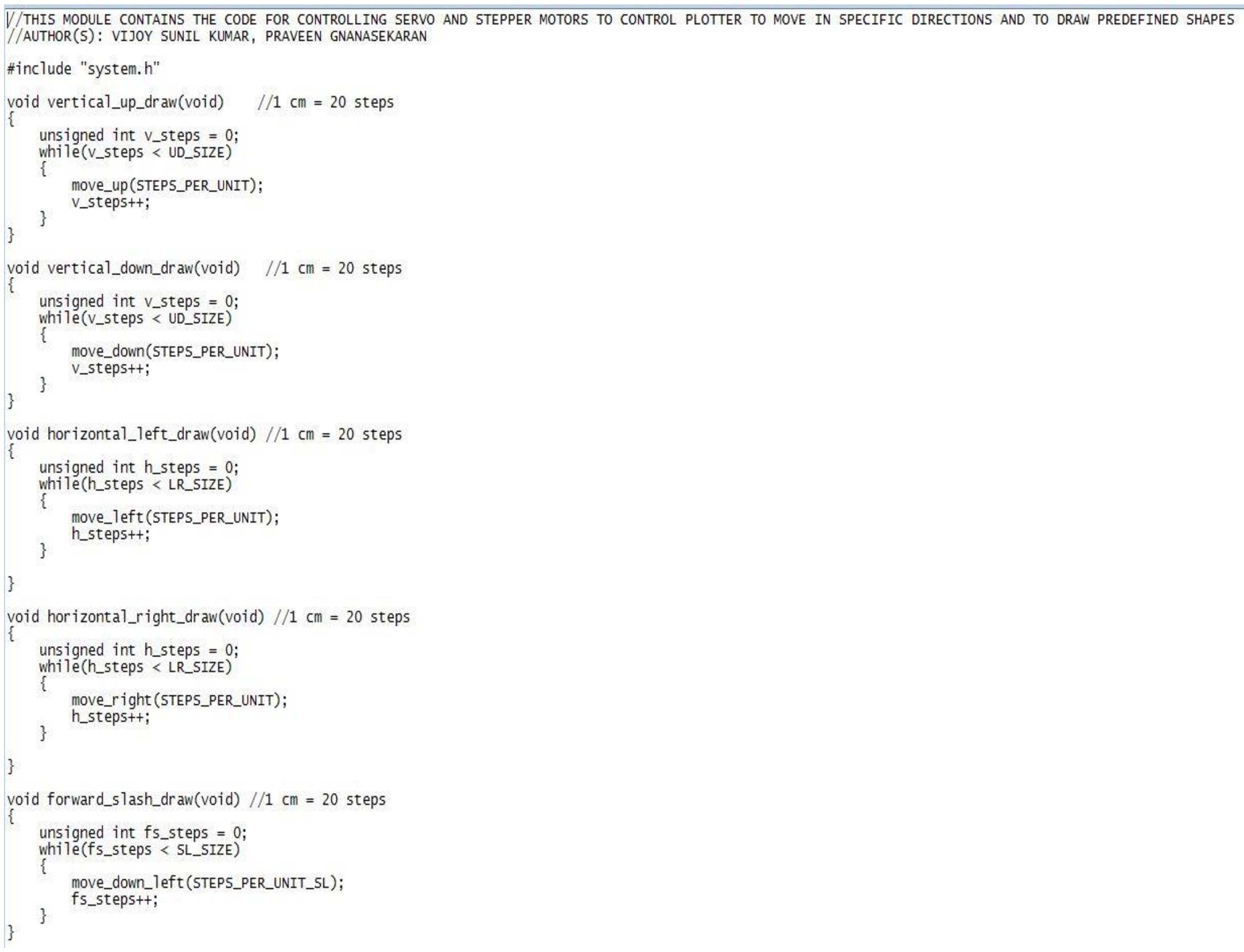
```
/*
* QWERTY_PEN FINAL REVISION
*AUTHOR(S): VIJOY SUNIL KUMAR, PRAVEEN GNANASEKARAN
*THIS IS THE MAIN MODULE OF THE PROJECT. ALL SUPPORTING FUNCTIONS ARE CALLED FROM THIS PAGE
 */
#include "system.h"

unsigned char cursor_x;
unsigned char sw_press,menu_action;

unsigned int i,u,d,l,r,ul,ur,dl,dr;


_sdcc_external_startup()
{
    AUXR |= 0x0C;
                                                      //setting baud to 9600
    TMOD=0X20;
    TH1=-3;
    SCON=0X50;
    TR1=1;

    TCON |= INT0_EDGE;                               //edge triggered
    IEN0 = GLOBAL_INT | INT0_MASK;           //enable global and int0 interrrupt
    return 0;
}


void main()
{
    system_init();
    while(1)              //menu loop
    {
        lcd_screen_1();     //qwerty pen
        delay_sec(1);
        lcd_screen_2();     //menu - origin/quick/custom

        cursor_display(1);   //cursor display - initial location
        delay_ms(1);

        while(1)
        {
            menu_scroll();
            if(menu_action == ENTER)           //enter option
            {
                menu_action = 0;
                if(cursor_x == 0)              //origin set
                    origin_menu_1_1();
                if(cursor_x == 1)              //quick print
                    quick_menu_1_2();
                if(cursor_x == 2)              //custom print
                    custom_menu_1_3();
            }
        }
    }
}
```

MAIN.C WHICH CALLS HIGH LEVEL FUNCTIONS



```
//THIS MODULE CONTAINS THE CODE FOR DRAWING A SENTENCE OR CHARACTER OR CUSTOM IMAGE AND RECEIVING PIXEL CHUNKS
//AUTHOR(S): VIJOY SUNIL KUMAR, PRAVEEN GNANASEKARAN

#include "system.h"
extern float current_x,current_y;
extern unsigned char cursor_x;
extern unsigned char menu_action;
char circular_buffer[BUFFER_SIZE];
char stop_draw,end_origin;
extern char in_origin;

void line_mod(void)     // IF THE PLOTTER REACHES AN END, IT AUTOMATIICALY GOES TO NEXT LINE DUE TO PUSH BUTTONS ON EITHER SIDE
{
    stop_draw = 0;
    if(LEFT_END == 0)
    {

    }
    else if(RIGHT_END == 0)
    {
        stop_draw = 1;
        pen_up();
        move_down(25 * 2 * 20);
        while(LEFT_END !=0)
        {
            move_left(25);
        }
    }
}

void line_mod_left(void)
{
    if(LEFT_END == 0)
    {
        end_origin = 1;
    }
}

void line_mod_right(void)
{
    if(RIGHT_END == 0)
    {
        end_origin = 1;
    }
}

void draw_string (void) //THE 8051 RECEIEVES 20*20 PIXEL CHUNK, STORES IT IN RAM AND MOVES THE MOTORS TO MAP THE DATA IT RECEIVES ONTO THE PAPER.
{
    int pixels,pix_chunk,pix_size,ii;

    char row_len,col_len,loop_break,str_len;
    int down_fl = 0,up_fl=0;
    stop_draw = 0;
    loop_break = 0;

    putchar('!');               //in print function
    while(1)
    {
        pixels=0;
        pix_chunk=0;
        pix_size=0;
```

TEXT.C WHICH DECODES THE BITMAP

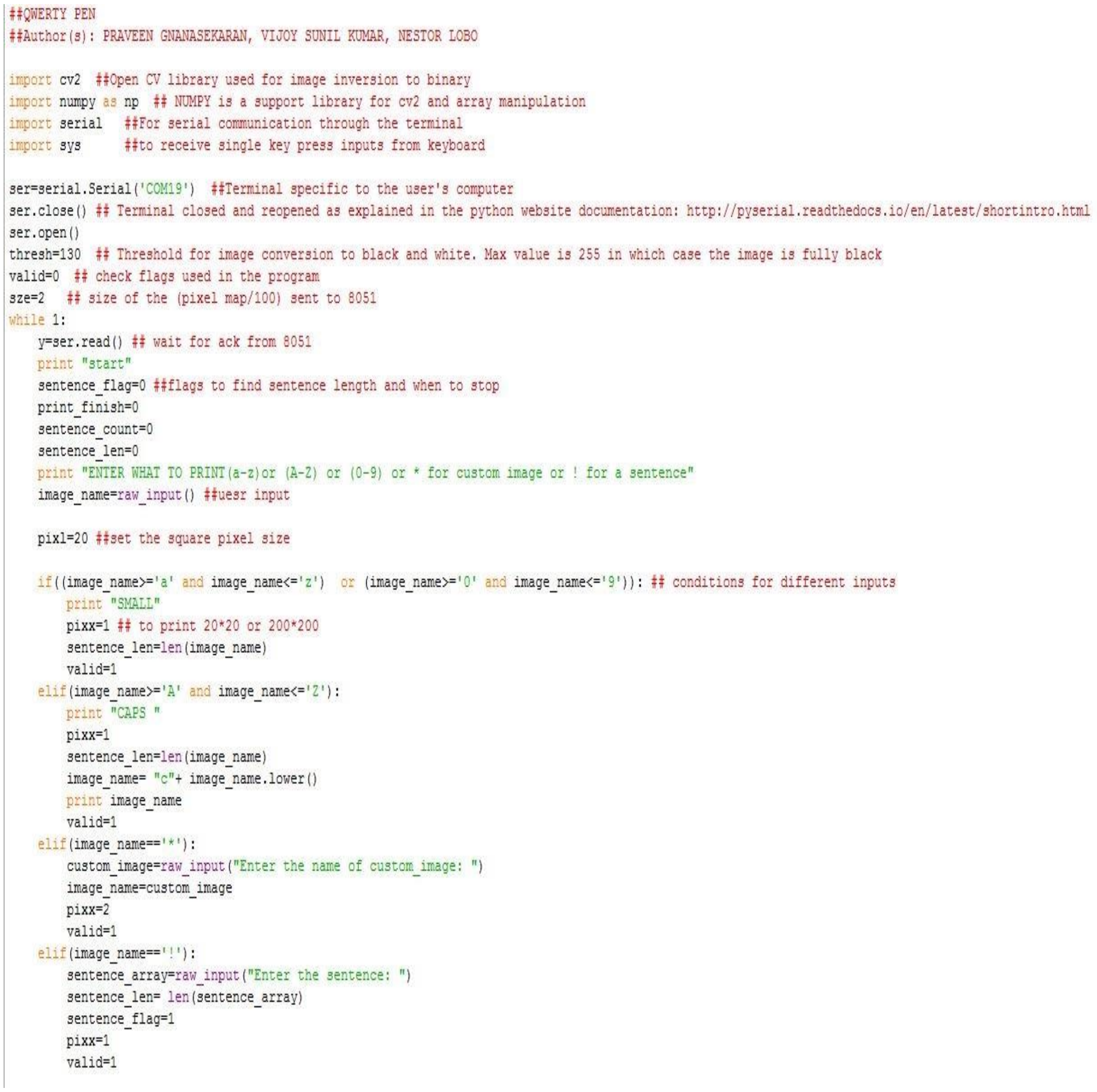


```
//THIS MODULE CONTAINS THE CODE FOR CONTROLLING SERVO AND STEPPER MOTORS TO CONTROL PLOTTER TO MOVE IN SPECIFIC DIRECTIONS AND TO DRAW PREDEFINED SHAPES
//AUTHOR(S): VIJOY SUNIL KUMAR, PRAVEEN GNANASEKARAN

#include "system.h"

void vertical_up_draw(void)    //1 cm = 20 steps
{
    unsigned int v_steps = 0;
    while(v_steps < UD_SIZE)
    {
        move_up(STEPS_PER_UNIT);
        v_steps++;
    }
}

void vertical_down_draw(void)   //1 cm = 20 steps
{
    unsigned int v_steps = 0;
    while(v_steps < UD_SIZE)
    {
        move_down(STEPS_PER_UNIT);
        v_steps++;
    }
}

void horizontal_left_draw(void) //1 cm = 20 steps
{
    unsigned int h_steps = 0;
    while(h_steps < LR_SIZE)
    {
        move_left(STEPS_PER_UNIT);
        h_steps++;
    }

}

void horizontal_right_draw(void) //1 cm = 20 steps
{
    unsigned int h_steps = 0;
    while(h_steps < LR_SIZE)
    {
        move_right(STEPS_PER_UNIT);
        h_steps++;
    }

}

void forward_slash_draw(void) //1 cm = 20 steps
{
    unsigned int fs_steps = 0;
    while(fs_steps < SL_SIZE)
    {
        move_down_left(STEPS_PER_UNIT_SL);
        fs_steps++;
    }
}
```

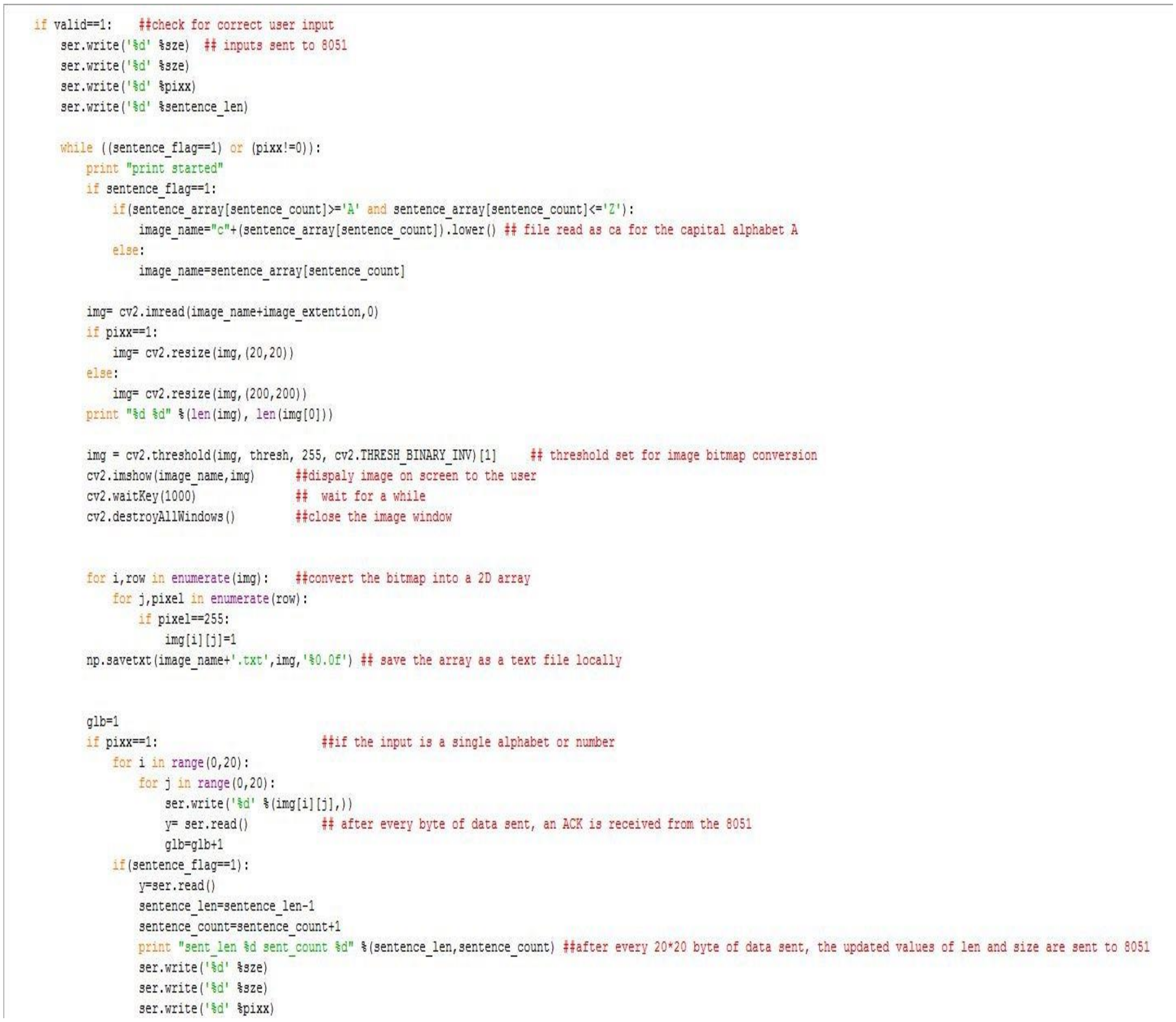
SHAPES.C WHICH DRAWS PREDEFINED IMAGES

8.4 Software Source Code

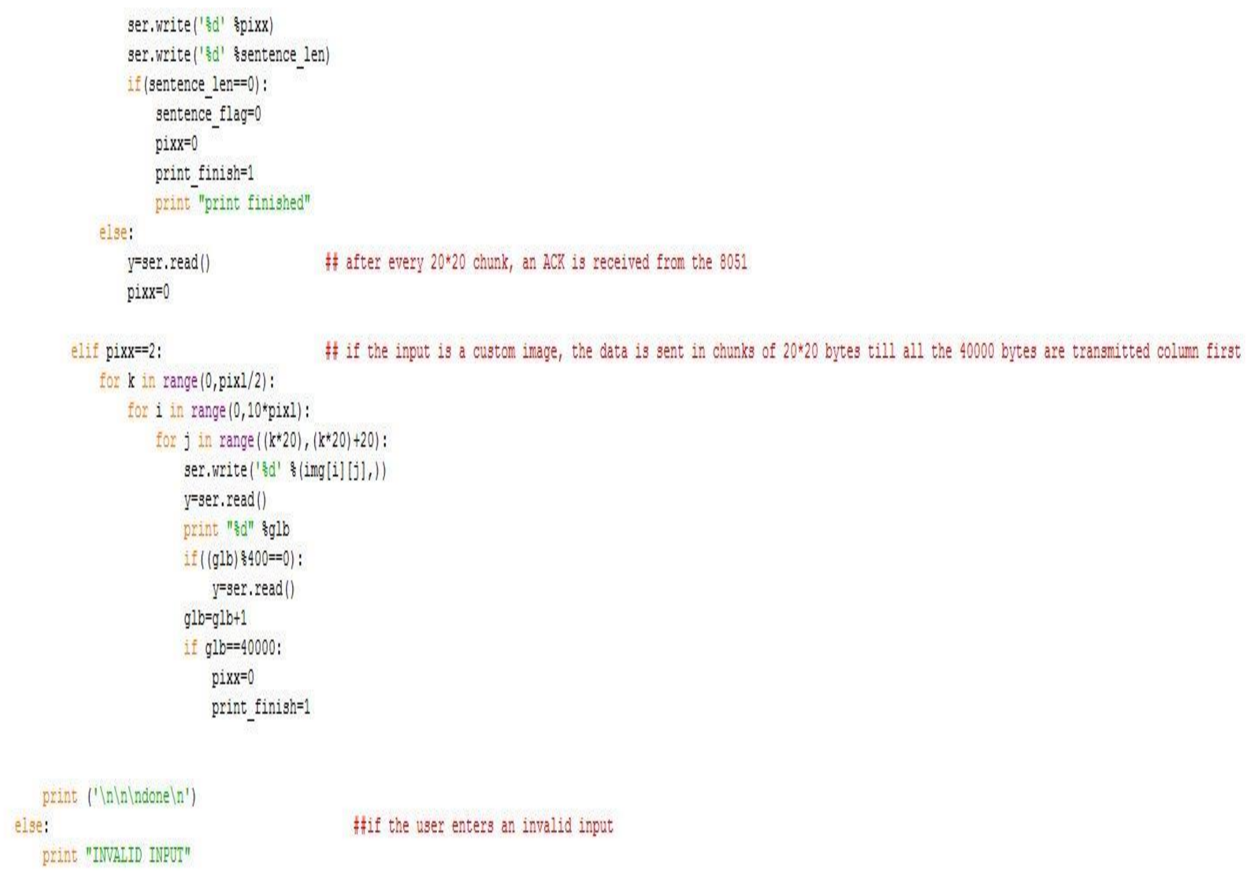
Python Script
Version : 2.7
IDE : IDLE
OS : Windows 10/7
This is the entire commented Python script used in the project



```
##QWERTY PEN
##Author(s): PRAVEEN GNANASEKARAN, VIJOY SUNIL KUMAR, NESTOR LOBO

import cv2  ##Open CV library used for image inversion to binary
import numpy as np  ## NUMPY is a support library for cv2 and array manipulation
import serial   ##For serial communication through the terminal
import sys      ##to receive single key press inputs from keyboard

ser=serial.Serial('COM19')  ##Terminal specific to the user's computer
ser.close() ## Terminal closed and reopened as explained in the python website documentation: http://pyserial.readthedocs.io/en/latest/shortintro.html
ser.open()
thresh=130  ## Threshold for image conversion to black and white. Max value is 255 in which case the image is fully black
valid=0  ## check flags used in the program
sze=2   ## size of the (pixel map/100) sent to 8051
while 1:
    y=ser.read() ## wait for ack from 8051
    print "start"
    sentence_flag=0 ##flags to find sentence length and when to stop
    print_finish=0
    sentence_count=0
    sentence_len=0
    print "ENTER WHAT TO PRINT(a-z)or (A-Z) or (0-9) or * for custom image or ! for a sentence"
    image_name=raw_input() ##uesr input

    pixl=20 ##set the square pixel size

    if((image_name>='a' and image_name<='z')  or (image_name>='0' and image_name<='9')): ## conditions for different inputs
        print "SMALL"
        pixx=1 ## to print 20*20 or 200*200
        sentence_len=len(image_name)
        valid=1
    elif(image_name>='A' and image_name<='Z'):
        print "CAPS "
        pixx=1
        sentence_len=len(image_name)
        image_name= "c"+ image_name.lower()
        print image_name
        valid=1
    elif(image_name=='*'):
        custom_image=raw_input("Enter the name of custom_image: ")
        image_name=custom_image
        pixx=2
        valid=1
    elif(image_name=='!'):
        sentence_array=raw_input("Enter the sentence: ")
        sentence_len= len(sentence_array)
        sentence_flag=1
        pixx=1
        valid=1
```



```
if valid==1:    ##check for correct user input
    ser.write('%d' %sze)  ## inputs sent to 8051
    ser.write('%d' %sze)
    ser.write('%d' %pixx)
    ser.write('%d' %sentence_len)

    while ((sentence_flag==1) or (pixx!=0)):
        print "print started"
        if sentence_flag==1:
            if(sentence_array[sentence_count]>='A' and sentence_array[sentence_count]<='Z'):
                image_name="c"+(sentence_array[sentence_count]).lower() ## file read as ca for the capital alphabet A
            else:
                image_name=sentence_array[sentence_count]

        img= cv2.imread(image_name+image_extention,0)
        if pixx==1:
            img= cv2.resize(img,(20,20))
        else:
            img= cv2.resize(img,(200,200))
        print "%d %d" %(len(img), len(img[0]))

        img = cv2.threshold(img, thresh, 255, cv2.THRESH_BINARY_INV)[1]     ## threshold set for image bitmap conversion
        cv2.imshow(image_name,img)      ##dispaly image on screen to the user
        cv2.waitKey(1000)               ##  wait for a while
        cv2.destroyAllWindows()         ##close the image window


        for i,row in enumerate(img):    ##convert the bitmap into a 2D array
            for j,pixel in enumerate(row):
                if pixel==255:
                    img[i][j]=1
        np.savetxt(image_name+'.txt',img,'%0.0f') ## save the array as a text file locally


        glb=1
        if pixx==1:                     ##if the input is a single alphabet or number
            for i in range(0,20):
                for j in range(0,20):
                    ser.write('%d' %(img[i][j],))
                    y= ser.read()           ## after every byte of data sent, an ACK is received from the 8051
                    glb=glb+1
            if(sentence_flag==1):
                y=ser.read()
                sentence_len=sentence_len-1
                sentence_count=sentence_count+1
                print "sent_len %d sent_count %d" %(sentence_len,sentence_count) ##after every 20*20 byte of data sent, the updated values of len and size are sent to 8051
                ser.write('%d' %sze)
                ser.write('%d' %sze)
                ser.write('%d' %pixx)
```



```
                ser.write('%d' %pixx)
                ser.write('%d' %sentence_len)
                if(sentence_len==0):
                    sentence_flag=0
                    pixx=0
                    print_finish=1
                    print "print finished"
            else:
                y=ser.read()            ## after every 20*20 chunk, an ACK is received from the 8051
                pixx=0

        elif pixx==2:                   ## if the input is a custom image, the data is sent in chunks of 20*20 bytes till all the 40000 bytes are transmitted column first
            for k in range(0,pixl/2):
                for i in range(0,10*pixl):
                    for j in range((k*20),(k*20)+20):
                        ser.write('%d' %(img[i][j],))
                        y=ser.read()
                        print "%d" %glb
                        if((glb)%400==0):
                            y=ser.read()
                        glb=glb+1
                        if glb==40000:
                            pixx=0
                            print_finish=1


    print ('\n\n\ndone\n')
else:                                   ##if the user enters an invalid input
    print "INVALID INPUT"
```

8.5 Appendix - Data Sheets and Application Notes