2D Graphics Engine using SPI color LCD Display

Karan Daryani *(*013722748*)*
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95112
E-mail: karan.daryani@sjsu.edu

Abstract

In this lab, we create a module of LCD which is operated using SPI through a microcontroller. We use LPC1769 as the microcontroller and ST7735 as the LCD Module. Using SPI operated LCD Display and LPC1769 we implement, design and test a graphics display prototype. Here the we establish a serial communication (SPI Bus) between the Microcontroller and the LCD. The objective of this lab is achieved by successfully implementing 2D vector graphics on LCD.

Keywords*: LPC1769, SPI, LCD, 2-D Graphics.*

1. Introduction

This lab is focused on implementation of a 2D graphic designs on LCD Display by collecting data from a CPU Module. The CPU Module used here is LPC 1769 which is based on ARM Cortex M0 Core. We use MCUXpresso IDE for implementing our design. The most important objective of this lab is to understand the SPI Communication Protocol between a LCD and CPU Module and use it to display different 2D Screen saver . In these screen saver, we have to first implement simple graphics drawing capability by plotting a single line with any user defined color and width. The LCD Module used here is a 18 bit Color Liquid Crystal Display which can work successfully using SPI Bus. It is important to have previous knowledge of programming in C/C++ language to work with the IDE and program the microcontroller.

2. Methodology

All particulars of design methodology of this lab both hardware and software are discussed in this section. The first step is to design the power circuit. The LPC1769 and 1.8" TFT LCD Display are soldered together on a prototype board and connected to the power circuit to work as a standalone module. In SPI Communication, we have a Master – Slave architecture where one part acts as a Master of the communication and the other acts as a Slave. This type of architecture is implemented by connecting LCD module as a slave and LPC CPU Module as a master. There are four main SPI logic signals used. They are SCK (Serial Clock, output from master), MOSI (Master Output Slave Input, output from master), MISO (Master Input Slave Output, output from slave) and CS (Chip Select, active low, output from master). Data Transfer take place from host computer to CPU module through the USB link and then the CPU transfers the data to the LCD Display module.

2.1. Objectives and Technical Challenges

Following were the objectives of this lab:

1. Creating a prototype board including the Microcontroller, LCD and a Power Circuit.
2. Getting familiar with 2D Vector Graphics and implementation of C code for the same.
3. Hands on Experience in MCUXpresso IDE.
4. Implementing, Testing and Debugging of CPU and LCD Modules and accomplishing data transfer among them.

While developing this module we came across a few technical challenged. These technical challenges are listed below.

1. Understanding the Pin Structure of the LCD Module and Bringing it up with SPI Communication.
2. Calculating and Displaying the proper co-ordinates for Square Screensaver.
3. Understanding Rotation, Scaling and Translation Graphic concepts for tree pattern.

2.2. Problem Formulation and Design

To implement a 2D screen saver of rotating squares we select a set of points along the sides the outer square which make the vertices of the inside square. Vector equation followed in implementation of rotating square screen saver is as follows,

$$P = P_i + \lambda d, \text{ where } d = P_{i+1} - P_i$$

In order to draw rotating squares, we select $\lambda = 0.2$. The same process is repeated and squares are drawn inside the main square.

For the second screen saver, we have to draw multiple trees to create an image of forest collectively. The following steps are followed to make this screensaver.

i. Draw the tree trunk and then generate the child nodes i.e. the center, left and right nodes.
ii. The center branch is generated using the equation,
$P_c = P_{end\_pt} + \lambda * d$
where $d = P_{end\_pt} - P_{start\_pt}$ and $\lambda = 0.8$
iii. To draw left and right branches we need to rotate the center branch with reference to the trunk end point.
iv. Translate the reference point to origin,
$X_{transalate} = X_{old} + \Delta x$
$Y_{translate} = Y_{old} + \Delta y$
v. Rotate the vector along the new reference point
$X_{rotation} = X_{translate} \cos\theta - Y_{translate} \sin\theta$
$Y_{rotation} = X_{translate} \sin\theta + Y_{translate} \cos\theta$

vi. Rotate the vector along the new reference point

$X_{rotation} = X_{translate} \cos\theta - Y_{translate} \sin\theta$

$Y_{rotation} = X_{translate} \sin\theta + Y_{translate} \cos\theta$

vii. Translate back to the reference point

$X_{final} = X_{rotation} - \Delta x$

$Y_{final} = Y_{rotation} - \Delta y$

These equations are the basis for implementing the software.

3. Implementation

The overall layout of the board along with the computer is as follows

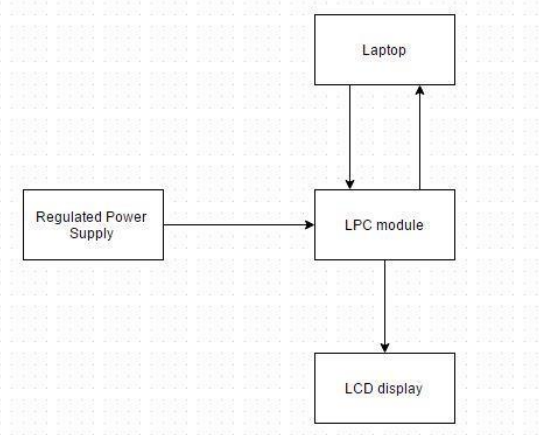

Figure 1. Block Diagram

The implementation of this lab in done in two parts hardware and software. The details of each part is described in this section

3.1. Hardware Design

The hardware components required to implement this lab is as follows:

No.	Item and Description	Notes
1.	Wire Wrapping board	8.5x11 inches
2.	NXP's LPC1769 module (ARM Cortex M3)	
3.	SPI based color LCD display	Driver ST7735R
4.	Colored wires	For connections
5.	Laptop	With MCU Xpresso IDE
6.	Power Regulator IC 7805	Output 3.3V
7.	Wall mount adaptor	Voltage > 7.5V DC

Table 1. Bill of materials

While making a prototype board the most important part is a power regulation circuit to give 5 V output to power up LPC1769. After that interface of LCD and the CPU Module. The CPU Module (Microcontroller) is operated as the Master. Its MISO, MOSI and SCK pins are used to interface with the SPI LCD Display which works as slave. The following pictures shows the hardware developed for of the project.

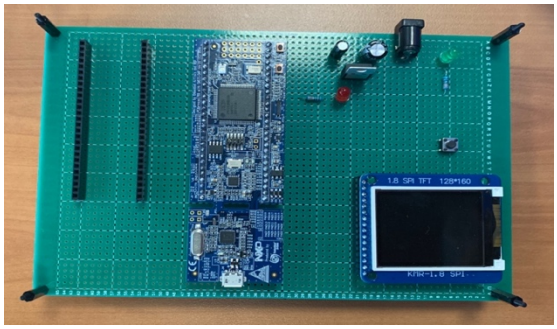

Figure 2. Hardware model of the system

The ST7735R is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source lines and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8-bit/9-bit/16-bit/18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption. In addition, because of the integrated power supply circuits necessary to drive liquid crystal, it is possible to make a display system with fewer components.

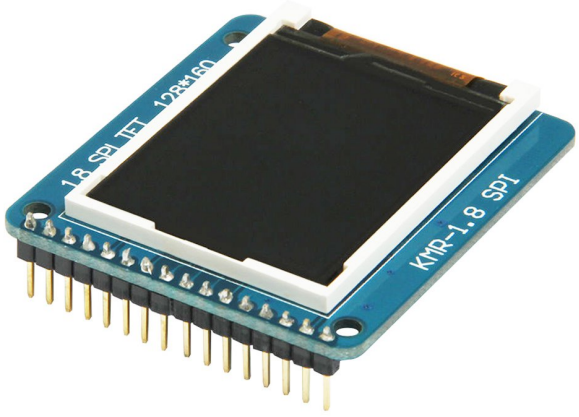

Figure 3. LCD Display Module

3.1.1 Circuit Design

The first thing when making a prototype board is a power regulator circuit. The CPU Module needs 5V to function properly so a voltage regulator is used to reduce any input voltage to 5V. Using a LM 7805 Voltage Regulator IC, capacitor, switch and LEDs a power circuit is made. The circuit diagram for the same is shown.

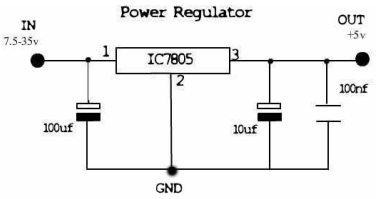

Figure 4. Circuit Diagram for Power Sensor

The following image shows the pin description of LCD Module

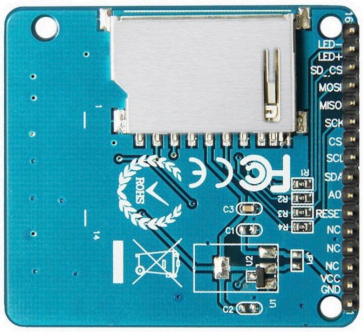

Figure 5. LCD Pin Description

The following table gives the detailed description of interface between the LPC and the LCD.

Pins Connection for LPC and LCD Interface	
LPC Pins	***LCD Pins***
Vcc (J6-28)	LED+ (Pin 2)
GND	LED- (Pin 1)
MISO0 (P0.17) (J6-12)	MISO (Pin 5)
SCK0 (P0.15) (J6-13)	SCK (Pin 6)
MOSI0 (P0.18) (J6-11)	SDA (Pin 9)
SSEL0 (P0.16) (J6-14)	TFT_CS (Pin 7)
GPIO (P0.21) (J2-23)	AO (Pin 10)
GPIO (P0.22) (J2-24)	RESET (Pin 11)
VCC 3.3V	VCC (Pin 15)
GND	GND (Pin 16)

Table 2. Pin connection between LCD and LPC 1769

In the figure 6, the LCD has 10 pins which connect to the CPU Module. The module is connected to the Host Computer via USB cable through which it is powered up. Pin J6-11, J6-13 and J6-14 of the LPC 1769 is used for serial communication with SPI port number 0 for our algorithm. When the CS Pin of the LCD Module is logic 1 the device is deselected, and data output pins are at high impedance. The output of the LCD Display that is the second pin is connected to the MISO (Master In Slave Out) of the CPU module. The SPI instruction uses MOSI and serially writes instructions on the rising edge of the clock. The LED+ pin is connected to the Vcc and the LED- is connected to the GND pin, this lights up the backlight and the display becomes more clear. The Chip Select signal for the TFT Display should be kept low all the time for the duration of the RESET operation to avoid resetting the internal logic state of the device. Instructions vary in length. For some we send only the opcode, for some we send dummy bytes also so that there is some time for the CPU module to recognize the instruction.

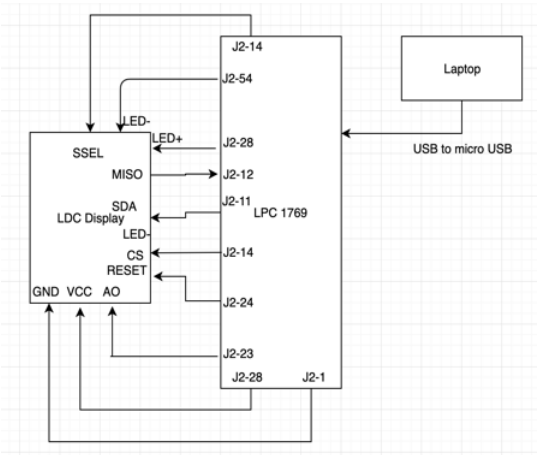

Figure 6. SPI Connection Between LCD and LPC

3.2. Software Design

The implementation of entire system is done in C language. This help us to use the open source library provided by the NXP for LPC 1769 Board support package.

A detailed description of the algorithm is provided below to help in better understanding the architecture. Entire source code with the detailed flow chart is provided in the appendix.

3.2.1 Algorithm

This lab experiment follows following simple sequential steps provided below-

1. Analyze the graphic design that needs to be implemented on LCD module. Develop routines in c language to implement desired figure. In our scope of experiment we use vectors in designing a simple line segment, rotating square as screen

2. saver and design bunch of trees to draw a forest on to the screen.
3. Setup the SPI controller by initializing and configuring the registers associated with it. These configurations include frame-size, bit rate and other settings.
4. Set up and initialize the LCD module.
5. Based on option selected by user, call corresponding C routine in transferring the data through SPI to the LCD module from CPU module to implement desired graphics.

The success of this lab experiment is based on ensuring that all the following graphics are implement on LCD module using vectors.

1. A simple line segment.
2. A rotating square screen saver.
3. A forest with bunch of trees.

3.2.2 Flow Chart

The below flow chart accurately depicts the entire process structure of this lab experiment.

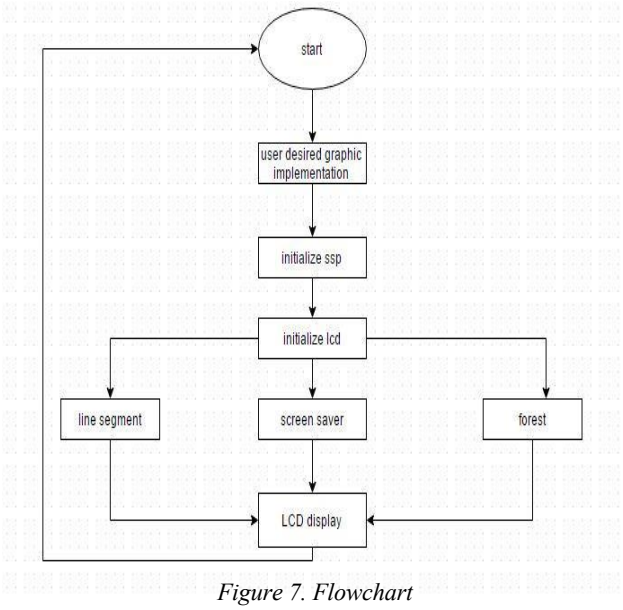

Figure 7. Flowchart

3.2.3 Pseudo Code

To draw the rotating square pattern on the LCD Display we are using the Draw Line Code.

```
void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
 int16_t slope = abs(y1 - y0) > abs(x1 - x0);
 if (slope) {
  swap(x0, y0);
  swap(x1, y1);
 }
 if (x0 > x1) {
  swap(x0, x1);
  swap(y0, y1);
 }
 int16_t dx, dy;
 dx = x1 - x0;
 dy = abs(y1 - y0);
 int16_t err = dx / 2;
 int16_t ystep;
 if (y0 < y1) {
  ystep = 1;
 }
 else {
  ystep = -1;
 }
 for (; x0 <= x1; x0++) {
  if (slope) {
   drawPixel(y0, x0, color);
  }
  else {
   drawPixel(x0, y0, color);
  }
  err -= dy;
  if (err < 0) {
   y0 += ystep;
```

```
  err += dx;
 }
}
}
```

4. Testing and Verification

Each component is soldered onto the prototype board and the its connectivity is checked by using a Digital Multimeter used in connectivity mode. While soldering make sure that there are no cold joint and not to overheat any sensitive components like LCD Display or the CPU Module. Also it is important to ensure that Vcc and Gnd Pins are not shorted since it can create a short circuit and damage the CPU Module. While soldering ensure that no two pins are sorted together causing in malfunction of the whole module.

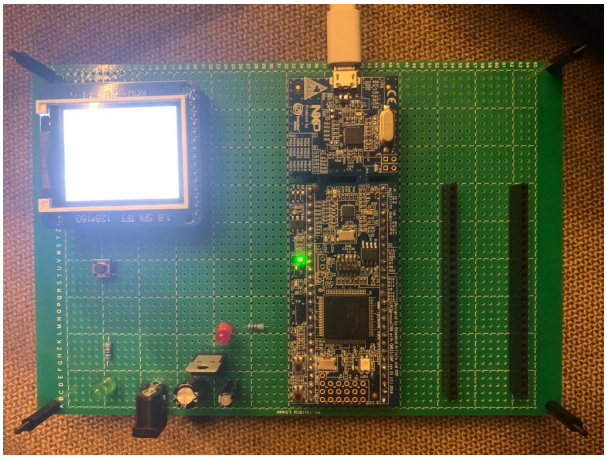

Fig. 8. Prototype board with Working LCD through SPI

The program ran successfully producing the following result.

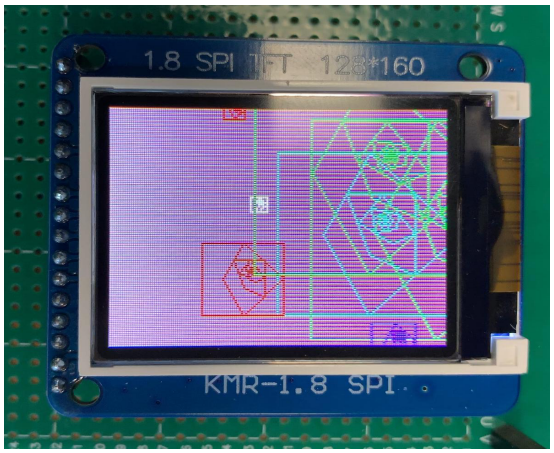

Figure 9. Rotating Squares Screensaver

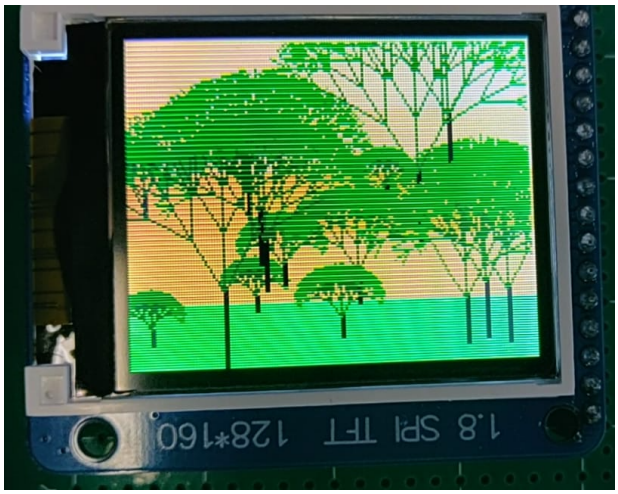

Figure 10. Forrest Screensaver

The code for Rotating Square and Forrest Screen saver is executed to verify that they are built appropriately. From the above images, we can validate that the SPI interface between the LPC 1769 and LCD Display is devised and executed successfully. Also, the objective of the lab to display rotating squares screensaver and patch of forest was satisfactorily achieved.

5. Conclusion

This lab is designed to make use of the LPC1769 CPU along with its inbuilt SPI controller to allow implementation of vector graphics on to LCD module. Upon completion of the experiment we understand how to implement desired graphics using vector concepts and communicate it with LCD for displaying purpose. The experiment related source code, flow chart and the schematic block diagram has been included in the appendix.

6. Acknowledgement

We would like to express our special gratitude to Dr. Harry Li for providing valuable information on LPC 1769, MCU Xpresso IDE and LCD. We have put in special effort to complete this project.

7. References

[1] Dr. H. Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112

[2] Dr. H. Li, CMPE240 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112

[3] NXP LPCXpresso1769 Discussion forums at www.lpcware.com/forum
[4] LPC1769 User manual
[5] http://www.adafruit.com/datasheets/W25Q80BV.pdf
[6] LCD Screen Datasheet – ST7735R http://www.adafruit.com/datasheets/ST7735R_V0.2.pdf

8. Appendix

[A] Source code

```
/*
===============================================
==============================
 Name        : DrawLine.c
 Author      : $KD
 Version     :
 Copyright   : $(copyright)
 Description : main definition
===============================================
==============================
*/

#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#define PORT_NUM 0

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29



#define swap(x, y) {x = x + y; y = x - y; x = x
- y ;}

// defining color values

#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#define BROWN 0xA52A2A
#define YELLOW 0xFFFF00
#define RED1 0xE61C1C
#define RED2 0xEE3B3B
#define RED3 0xEF4D4D
#define RED4 0xE88080
#define GGREEN 0X33FF33
#define GREEN1 0x00CC00
#define GREEN2 0x009900
#define GREEN3 0x006600
#define GREEN4 0x193300
#define DEEPSKYBLUE 0x00bfff
#define DARKORANGE 0xfc9207
#define ORANGE 0xfc9d21
#define ORANGE2 0xfda83a
#define ORANGE3 0xfdb353


int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

typedef struct Point
{
        float x;
        float y;
}Point;

void spiwrite(uint8_t c)

{

 int pnum = 0;

 src_addr[0] = c;

 SSP_SSELToggle( pnum, 0 );

 SSPSend( pnum, (uint8_t *)src_addr, 1 );

 SSP_SSELToggle( pnum, 1 );

}



void writecommand(uint8_t c)

{

 LPC_GPIO0->FIOCLR |= (0x1<<21);

 spiwrite(c);

}
```

```
void writedata(uint8_t c)
{
 LPC_GPIO0->FIOSET |= (0x1<<21);
 spiwrite(c);
}

void writeword(uint16_t c)
{
 uint8_t d;
 d = c >> 8;
 writedata(d);
 d = c & 0xFF;
 writedata(d);
}

void write888(uint32_t color, uint32_t repeat)
{
 uint8_t red, green, blue;
 int i;
 red = (color >> 16);
 green = (color >> 8) & 0xFF;
 blue = color & 0xFF;
 for (i = 0; i< repeat; i++) {
  writedata(red);
  writedata(green);
  writedata(blue);
 }
}

void setAddrWindow(uint16_t x0, uint16_t y0,
uint16_t x1, uint16_t y1)
{
 writecommand(ST7735_CASET);
 writeword(x0);
 writeword(x1);
 writecommand(ST7735_RASET);
 writeword(y0);
 writeword(y1);
}

void fillrect(int16_t x0, int16_t y0, int16_t
x1, int16_t y1, uint32_t color)
{
 //int16_t i;
 int16_t width, height;
 width = x1-x0+1;
 height = y1-y0+1;
 setAddrWindow(x0,y0,x1,y1);
 writecommand(ST7735_RAMWR);
 write888(color,width*height);
}

void lcddelay(int ms)
{
 int count = 24000;
 int i;
 for ( i = count*ms; i > 0; i--);
}

void lcd_init()
{
 int i;
```

```
 printf("LCD Demo Begins!!!\n");
 // Set pins P0.16, P0.21, P0.22 as output
 LPC_GPIO0->FIODIR |= (0x1<<16);

 LPC_GPIO0->FIODIR |= (0x1<<21);

 LPC_GPIO0->FIODIR |= (0x1<<22);

 // Hardware Reset Sequence
 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);

 LPC_GPIO0->FIOCLR |= (0x1<<22);
 lcddelay(500);

 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);

 // initialize buffers
 for ( i = 0; i < SSP_BUFSIZE; i++ )
 {

   src_addr[i] = 0;
   dest_addr[i] = 0;
 }

 // Take LCD display out of sleep mode
 writecommand(ST7735_SLPOUT);
 lcddelay(200);

 // Turn LCD display on
 writecommand(ST7735_DISPON);
 lcddelay(200);

}



void drawPixel(int16_t x, int16_t y, uint32_t
color)

{

 if ((x < 0) || (x >= _width) || (y < 0) || (y
>= _height))

 return;

 setAddrWindow(x, y, x + 1, y + 1);

 writecommand(ST7735_RAMWR);

 write888(color, 1);

}

/*********************************************
*****************************
```

```
** Descriptions:        Draw line function

**

** parameters:           Starting point (x0,y0),
Ending point(x1,y1) and color

** Returned value:        None

**

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


void drawLine(int16_t x0, int16_t y0, int16_t
x1, int16_t y1, uint32_t color)

{

 int16_t slope = abs(y1 - y0) > abs(x1 - x0);

 if (slope) {

  swap(x0, y0);

  swap(x1, y1);

 }

 if (x0 > x1) {

  swap(x0, x1);

  swap(y0, y1);

 }

 int16_t dx, dy;

 dx = x1 - x0;

 dy = abs(y1 - y0);

 int16_t err = dx / 2;

 int16_t ystep;

 if (y0 < y1) {

  ystep = 1;

 }

 else {

  ystep = -1;

 }
```

```
 for (; x0 <= x1; x0++) {

  if (slope) {

   drawPixel(y0, x0, color);

  }

  else {

   drawPixel(x0, y0, color);

  }

  err -= dy;

  if (err < 0) {

   y0 += ystep;

   err += dx;

  }

 }

}

void drawSquare(int16_t x0, int16_t y0, int16_t
x1, int16_t y1, int16_t x2, int16_t y2, int16_t
x3, int16_t y3, uint32_t color, uint8_t level,
float lambda)
{

        for(int i = 0 ; i < level; i++)
        {
                drawLine(x0, y0, x1, y1, color);
                drawLine(x1, y1, x2, y2, color);
                drawLine(x2, y2, x3, y3, color);
                drawLine(x3, y3, x0, y0, color);

//              x0 = x0 + 0.2 * (x1 - x0);
//              y0 = y0 + 0.2 * (y1 - y0);
//              x1 = x1 + 0.2 * (x2 - x1);
//              y1 = y1 + 0.2 * (y2 - y1);
//              x2 = x2 + 0.2 * (x3 - x2);
//              y2 = y2 + 0.2 * (y3 - y2);
//              x3 = x3 + 0.2 * (x0 - x3);
//              y3 = y3 + 0.2 * (y0 - y3);

                x0 = x0 + lambda * (x1 - x0);
                y0 = y0 + lambda * (y1 - y0);
                x1 = x1 + lambda * (x2 - x1);
                y1 = y1 + lambda * (y2 - y1);
                x2 = x2 + lambda * (x3 - x2);
                y2 = y2 + lambda * (y3 - y2);
                x3 = x3 + lambda * (x0 - x3);
                y3 = y3 + lambda * (y0 - y3);
        }
}

//Rotate point p with respect to o and angle
<angle>
Point rotate_point(Point p, Point o, float
angle)
{
        Point rt, t, new;

        // Convert to radians
        angle = angle * (3.14285/180);
        float s = sin(angle);
        float c = cos(angle);

        //translate point to origin
        t.x = p.x - o.x;
        t.y = p.y - o.y;

        rt.x = t.x * c - t.y * s;
        rt.y = t.x * s + t.y * c;

        //translate point back
        new.x = rt.x + o.x;
        new.y = rt.y + o.y;

        return new;
}

void tree(Point start, Point end, int level)
{
        Point c, rtl, rtr;
        if(level == 0)
                return;
        int color[] = {GREEN2, GREEN2, GREEN2,
GREEN3, GREEN3, GREEN3, GREEN3, GREEN4, GREEN4,
GREEN4};

        c.x = end.x + 0.8 * (end.x - start.x);
        c.y = end.y + 0.8 * (end.y - start.y);
        drawLine(c.x, c.y, end.x, end.y,
color[level]);
        tree(end, c, level - 1);

        rtl = rotate_point(c, end, 30);
        drawLine(rtl.x, rtl.y, end.x, end.y,
color[level]);
        tree(end, rtl, level - 1);

        rtr = rotate_point(c, end, 330);
        drawLine(rtr.x, rtr.y, end.x, end.y,
color[level]);
        tree(end, rtr, level - 1);
}

/*

 Main Function main()

*/
```

```
void drawTreetwig(Point start, Point end,
uint16_t color, uint8_t thickness)
{
        int i;
        for(i = 0; i < thickness; i++)
                drawLine(start.x, start.y + i,
end.x, end.y + i, color);
}

int main (void)
{
        Point start, end;

        uint32_t pnum = PORT_NUM, width =
ST7735_TFTWIDTH / 5, len;
        pnum = 0 ;

        srand(time(NULL));
        if ( pnum == 0 )
                SSP0Init();
        else
                 puts("Port number is not
correct");

//               float lambda_value;
//               printf("Enter the value of
lambda:\n");
//               scanf("%f",lambda_value);

         lcd_init();

         float lambda_value;
         printf("Enter the value of lambda:\n");
         scanf("%f",&lambda_value);

         //Start rotating square

        // fillrect(0, 0, ST7735_TFTWIDTH,
ST7735_TFTHEIGHT, BLACK);
         fillrect(0, 0, ST7735_TFTWIDTH,
ST7735_TFTHEIGHT, RED);
         int color[] = {LIGHTBLUE, GREEN, RED2,
WHITE, BLUE, RED, RED4, PURPLE};
while(1)
{
         int background_color=rand()%8;
         fillrect(0, 0, ST7735_TFTWIDTH,
ST7735_TFTHEIGHT,color[background_color]);
         for(int i = 0; i < 30; i++)
         {
                 int x0,y0, x1, y1, x2, y2, x3,
y3, len, cindex;
                 x0 = rand() % 140;
                 y0 = rand() % 125;
                 len = rand() % 120;
                 cindex = rand()%8;

                 x1 = x0;
                 y1 = y0 - len;
                 x2 = x0 - len;
                 y2 = y1;
                 x3 = x2;
                 y3 = y0;

                 //drawSquare(x0, y0, x1, y1, x2,
y2, x3, y3, color[cindex], 10);
                 drawSquare(x0, y0, x1, y1, x2,
y2, x3, y3, color[cindex], 10, lambda_value);
                 lcddelay(70);
```