2D VECTOR GRAPHICS ON MICROPROCESSOR BASED SYSTEMS

Sashank malladi
Sashank.malladi@sjsu.edu
Department of Computer Engineering, San José State University
1 Washington Sq, San Jose, CA 95192

Abstract- Vector graphics is the use of geometrical primitives such as points, lines, curves, and shapes or polygons—all of which are based on mathematical expressions—to represent images in computer graphics. The purpose of this lab is to create a module with SPI enabled LCD running through a microcontroller unit using LPC1769 as microcontroller and ST7735 as LCD module. The system under consideration is the CPU module and the LCD connected through a Serial Peripheral Interface (SPI) port. Due to the embedded environment and limited port availability, serial connection is preferred over the parallel connection. The objective of this lab is achieved by successfully implementing 2D vector graphics on LCD.

I. INTRODUCTION

This lab is focused on implementing 2-D graphic designs on an LCD module that can collect statistical data from the CPU module. This CPU module uses LPC1769 CPU, based on the ARM cortex M0 core that is provided with a user-friendly IDE called LPCXpresso . In this embedded environment, parallel communication leads to a superfluous usage of large number of CPU pins that are required to process the signals and hence serial communication is the preferred choice of interface between the data LCD module and the CPU module. The main objective of this lab experiment is to understand designing graphics using vectors and display it on LCD module with effective SPI communication between LCD module and CPU. It is essential to have prior programming knowledge of C/C++ languages to program the microcontroller circuit.

II. METHODOLOGY

All aspects of this lab experiment are detailed in this report. This includes, the hardware requirements with its technical specifications, the development board, the flow table depicting the implementation process and source code. Further sections cover these aspect, one by one, in great detail to provide a through insight into this lab experiment. All the instructions were successfully tested and verified.

The technical specifications of the hardware is listed below

CPU	Core	32 bit ARM Cortex M3
	Speed	Up to 120 MHz
	Power	120 – 140 mA
MEMORY	RAM	64 kB SRAM
	ROM	512 kB on chip flash

Table. 1. Technical Specifications of the Hardware.

A. Architectural Overview

This experiment can be distributed into three major blocks with each performing a specific task that is provided below

1. The Central Processing Unit (CPU) module

This block consists of a LPC1769 CPU provided by the NXP semiconductors. It is an ARM Cortex-M3 based microcontroller unit for embedded application. The technical statistics are given above table.

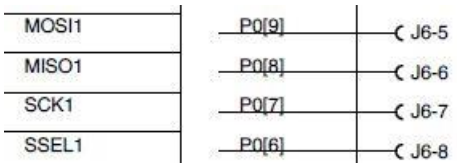


Figure 1. a. Module pinout diagram for SPI1.

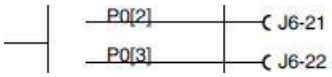


Figure 1. b. Module pinout diagram for GPIOs utilized.

2. Regulated Power Supply (5V)

A regulated power supply of 5V is fed to this system through a power adaptor plugged to a power socket using a 2.1mm power connector and a switch in series.

3. LCD Module

This lab uses ST7735 a single-chip SPI enabled controller/driver for 262K-color, graphic type TFT-LCD.

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

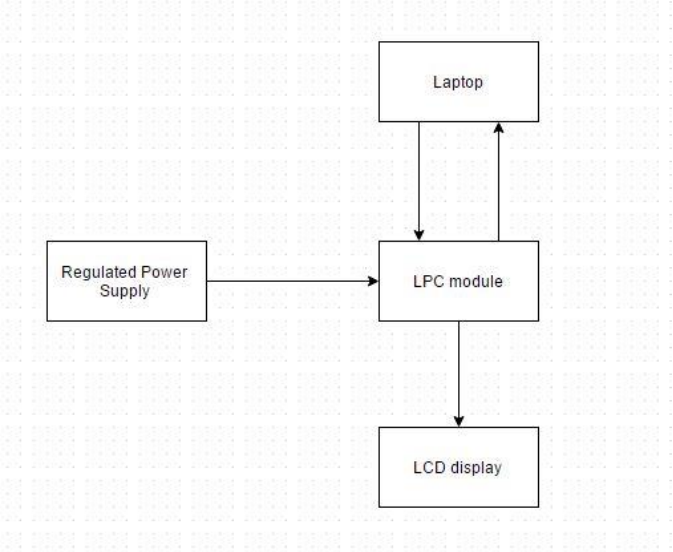
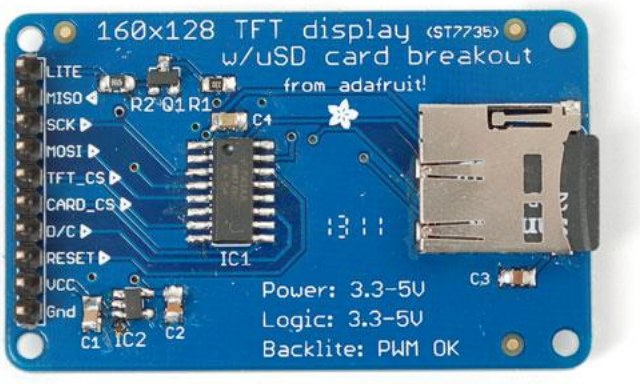


Figure 2. Block Diagram

B. LCD module Interface Design

The CPU module used here is LPC1769 which consists of two inbuilt SPI (serial port interface) controllers for serial communication (SPI0 and SPI1). In this lab experiment, the SPI bus establish the necessary connection with the LCD module allowing transfer of required data to LCD module. The figure below provides the basic pins on the ST7735 LCD module.



C. SPI Overview

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, flash memories, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.
SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

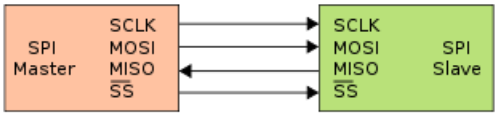


Figure 4. SPI Interface Overview

D. Vectors Overview

Vector equation followed in implementation of rotating square screen saver is as follows
Generalized equation:
V(p)=v(p1)+ c(v(p2)-v(p1))
Based on co-ordinates
X=x1+c(x2-x1)
Y=y1+c(y2-y1)
Where X and Y are new co-ordinates reduced by factor of c(constant) which are used in the rotating square matrix.
As such new points are found on each side of the square and new square is formed with rotation and size reduction.

For the second phase of the lab, as part of developing a tree, we use a transformation matrix.
Steps to be followed are as follows
Translation of origin to the point of interest
Perform rotation
Put back the origin to its original settings
Following shows illustration of the above mentiond steps

xout	=	r00	r01	x - r00*x - r01*y	xin
yout		r10	r11	y - r10*x - r11*y	yin
1		0	0	1	1

where:
r00 = cos(θ), r01 = -sin(θ), r10 = sin(θ), r11 = cos(θ)
θ = angle we are rotating around (in appropriate units for the trig functions we are using)
xin,yin = the coordinates of the input point
xout,yout = the coordinates of the output point (the result)
x,y = the coordinates of the point that we are rotating around

E. Software Overview

The entire system is implemented in C language. This lets the experiment to use the extensive open source library provided by the C as a Board support package for the CPU module.

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

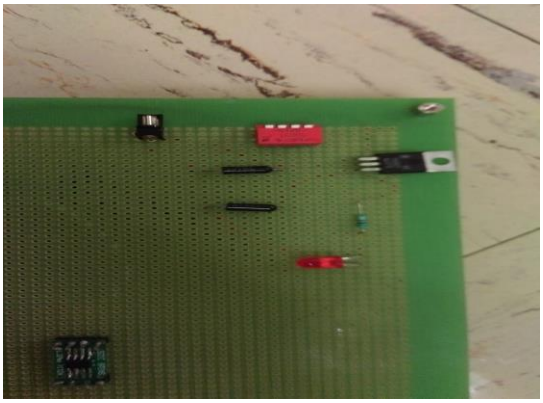


Figure 6A. Power Circuit

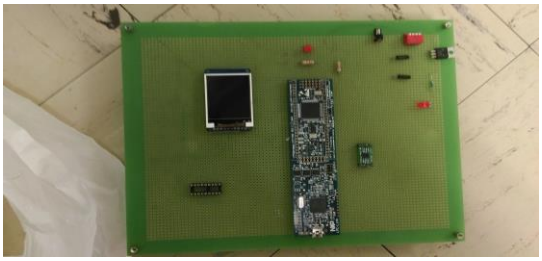


Figure 6B. Complete Circuit

Pseudo Code

The complete working of the code can be briefly depicted using the pseudo code provided below

```
//initialize the ssp control of LPC module
from //following routine.
//power up the ssp port
// to power up the ssp port we need to we
should configure the PCONP register
// set pcssp1 in the pconp register for
activate the ssp1 port
// have to set the 10th bit of pconp register
to set the pcssp1
// we find that register in lpc_sc (system
control)
// setting the clock equal to the clock of
the lpc board so the 21:20 bits of the
pclksel0 register to 01
//select the ssp pins to be used
//p0.6 is the chip select port and this
should be one to avoid writing on to the chip
and should be zero to enable writing on to
the chip.
//and its direction should be out so as p0.6
falls under pinsel0 the direction should be
changed using fiodir0
// there are two types of registers that need
to be configured SSP1CR0,SSP1CR1
// setting the ccpsdvsr to max value so as to
control the ssr(serial clock rate) value in
cr0
initSSP1();
// initialize LCD by using LPC gpio for
rs/rst and
//Use command 0x01for reset
//Use command 0x11 for sleepout after reset
//And use command 0x29 for turning on display
//USE P22 AND P21 GPIOS AS OUTPUT FOR RESET
AND C/D
lcd_init();

// fill the back ground of the lcd with
desired color
fillRect(0,0,ST7735_TFTWIDTH,ST7735_TFTHEIGHT,0
xDBB84D);

// perform desired action based on user
selection
 If user selects to draw a line segment
        drawline(10,10,10,89,BLACK);
else if user selects to draw a rotating square screen saver
        DrawScreenSaver()
else if user selects to draw a forest
        DrawBackgroundorForest();
        DrawAForrest(20,160,10,0x474719);
```

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 saver and design bunch of trees to draw a forest on to the screen.
2. Setup the SPI controller by initializing and configuring the registers associated with it. These configurations include frame-size, bit rate and other settings.
3. Set up and initialize the LCD module.
4. 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.

Flow Chart

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

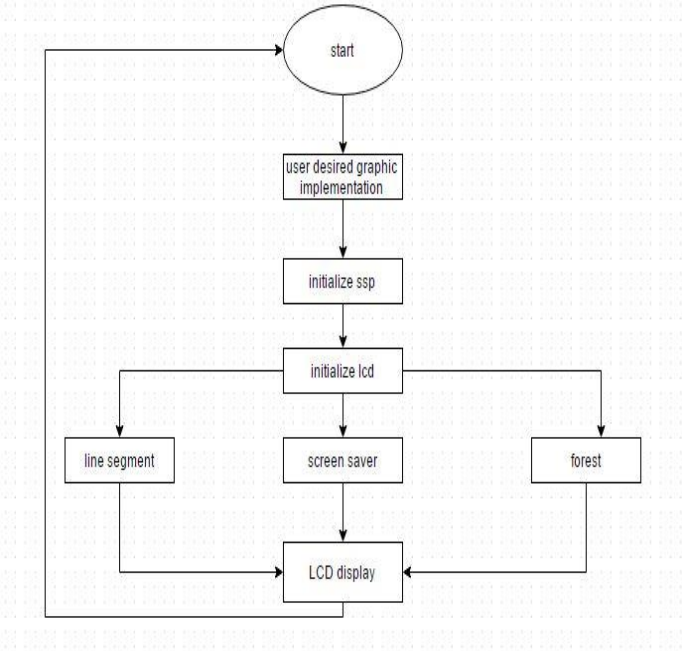


Figure 7. Flow chart

III. IMPLEMENTATION

The entire implementation of this lab experiment is explained in detail following steps-

1. To start with, the LCD module is soldered on a prototype board and ensured that all the pins are working properly.
2. 8 wires are soldered to the prototype board which act as the connections to the LCD module.
3. Wire wrap the LCD module with the CPU module and GND. (Pin connections table is provided in the appendix)
4. The CPU module is then connected with the computer using the USB cable.
5. After connecting the CPU module, the code (provided in appendix) is compiled using the LPCXpresso IDE and downloaded to the CPU module. The required output is then observed on the IDE's Console window or the SPI interfaced peripheral.

Schematics

Operating voltage	LPC Module	LCD module ST7735
	5v through usb cable	3.3 v

The entire schematic of the system is included below for further detailed understanding. The diagram provided below depicts the complete system after performing all the installation and the configurations.

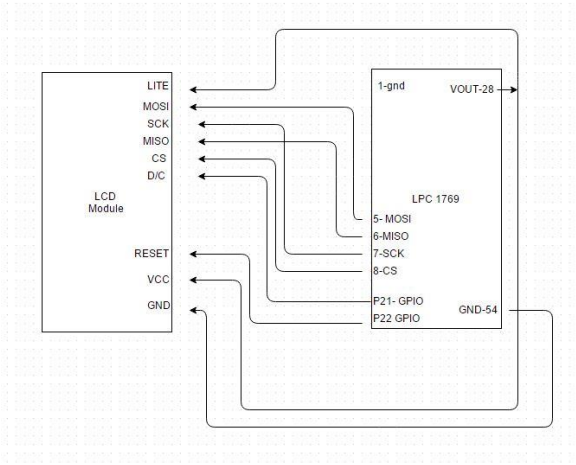


Figure 8. Full Circuit diagram

IV. TESTING AND VERIFICATION

A switch provided on the development board is turned on to

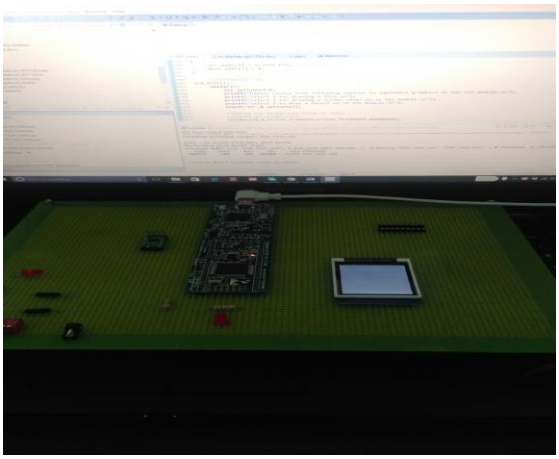


Figure 9. Implementation on board

power ON the central CPU module. This also turns on the LED on the development board which indicates the switching on of the CPU module. To being the installation and configurations on the computer, the LPCXpresso IDE must be installed and run.Following this, the CPU module is connected to the host using the USB cable. Upon initiating the debug session, a link is established with the CPU module and then binary code is uploaded to it. After this, the output console and LCD screen provides the desired output.

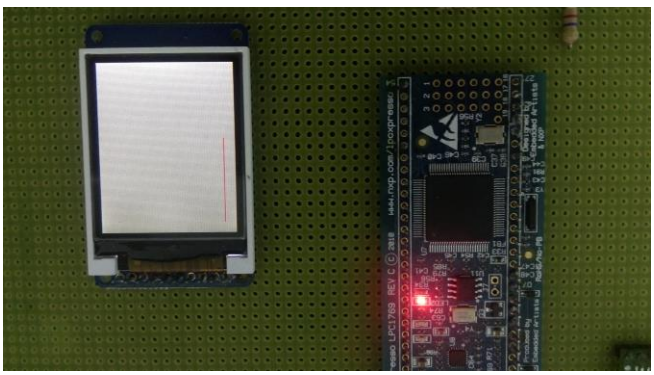


Figure 10. Line segment output

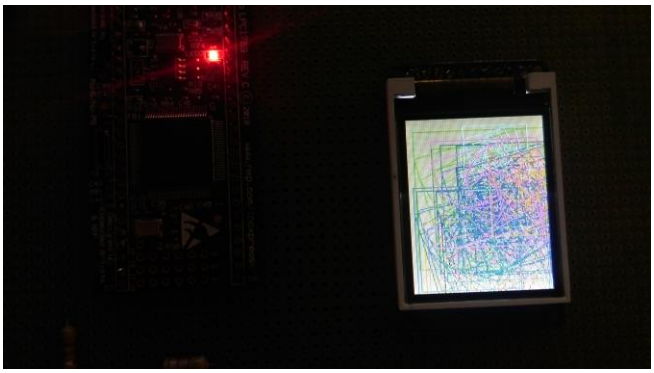


Figure 11. Screen saver output

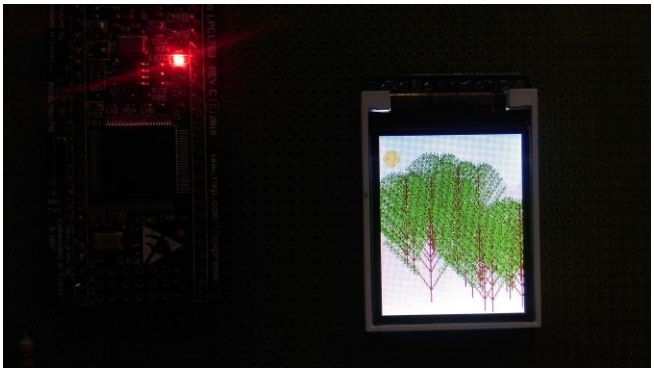


Figure 12. Forest output

V. CONCLUSION

This lab is designed to make use of the LPC1769 CPU along with it 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.

VI. ACKNOWLEDGEMENT

The implementation and the content described in this paper was achieved with the support of Dr. Harry Li.

VII. REFERENCES

1 H. Li, CMPE-240 class , Computer Engineering Department, College of Engineering, San Jose State university.

2 LPC1769 product description, http://www.nxp.com/demoboard/OM13000.html

3 LPCXpresso IDE, http://www.lpcware.com/lpcxpresso/downloads/windows

4 SPI wiki, http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

5 http://www.euclideanspace.com/maths/geometry/affine/aroundPoint/matrix2d/index.htm

6 LCD module , ftp://imall.iteadstudio.com/IM120419001_ITDB02_1.8SP/DS_ST7735.pdf

VIII. APPENDIX

A. Components used
B. Pin Connection
C. Source Code

A. Components used

Description	Quantity
Wire Wrapping Board	1
Wire Wrapping Tool Kit	1
Various Wire Color for Wire Wrapping	3
DC Power Supply 12V 10A	1
1/4 inch stands for board	4
Green color LED	1
LM7805 5V Regulator	1
1 KΩ Resistor	1
1 µF Capacitor	2
Power Switch	1
Slider switch	1
Spi enabled LCD module	1

Table 2. All the Components

B. Pin Connection Tables

CPU LPC 1769	*LCD Module ST7735*
nCS (pin 8)	TFT-CS (pin-5)
stsCK (Pin 7)	sCK (Pin 3)
MOSI (Pin 5)	SI (Pin 4)
MISO(Pin 6)	SO (Pin 2)
V out(pin 28)	Vcc (Pin 9) Lite (pin 1)

Gnd (pin 1)	GND(Pin 10)
GPIO (Pin 21)	D/C (pin-7)
GPIO (Pin 22)	Reset (pin-8)

Table 3. Pin connection

C. Source Code

```
/*****************************************************
*************************
 *   $Id:: ssptest.c 6098 2011-01-08 02:26:20Z nxp12832
$
 *   Project: NXP LPC17xx SSP example
 *
 *   Description:
 *     This file contains SSP test modules, main entry, to 2-D
implement vector graphics.
 *

*****************************************************
*************************
 * Software that is described herein is for illustrative purposes
only
 * which provides customers with programming information
regarding the
 * products. This software is supplied "AS IS" without any
warranties.
 * NXP Semiconductors assumes no responsibility or liability
for the
 * use of the software, conveys no license or title under any
patent,
 * copyright, or mask work right to the product. NXP
Semiconductors
 * reserves the right to make changes in the software without
 * notification. NXP Semiconductors also make no
representation or
 * warranty that such application will be suitable for the
specified
 * use without further testing or modification.
*****************************************************
*************************/
#include <cr_section_macros.h>
#include <NXP/crp.h>


// Variable to store CRP value in. Will be placed automatically
// by the linker when "Enable Code Read Protect" selected.
// See crp.h header for more information
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;

#include "LPC17xx.h"                        /* LPC17xx definitions
*/
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "font.h"
#include "SJSU.h"


/* Be careful with the port number and location number,
because
some of the location may not exist in that port. */
#define PORT_NUM            1
#define LOCATION_NUM        0

#define pgm_read_byte(addr) (*(const unsigned char *)(addr))

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];
int colstart = 0;
int rowstart = 0;

/*****************************************************
*************************
** Function name:       LCD_TEST
**
** Descriptions:        2-d vector graphics- line segment, screen
saver, forest
**
** parameters:          None
** Returned value:      None
**
*****************************************************
*************************/

//LCD
#define ST7735_TFTWIDTH  130
#define ST7735_TFTHEIGHT 160
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define swap(x, y) { x = x + y; y = x - y; x = x - y; }

//Colours
#define BLACK  0xFF0000
//Axes
int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;
int cursor_x = 0, cursor_y = 0;

//for writing data into the SPI
void spiwrite(uint8_t c)
{
   int portnum = 1;
   src_addr[0] = c;
   SSP_SSELToggle( portnum, 0 );
   SSPSend( portnum, (uint8_t *)src_addr, 1 );
   SSP_SSELToggle( portnum, 1 );
}
//writing commands into SPI
void writecommand(uint8_t c) {
   LPC_GPIO0->FIOCLR |= (0x1<<21);
   spiwrite(c);
}
//making LCD ready to write data
void writedata(uint8_t c) {

   LPC_GPIO0->FIOSET |= (0x1<<21);
   spiwrite(c);
}
//writing data to the LCD
void writeword(uint16_t c) {
```

```
    uint8_t d;
    d = c >> 8;
    writedata(d);
    d = c & 0xFF;
    writedata(d);
}
//write colour
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 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);
}

//drawLetters()

void HLine(int16_t x0,int16_t x1,int16_t y,uint16_t color){
    width = x1-x0+1;
    setAddrWindow(x0,y,x1,y);
    writecommand(ST7735_RAMWR);
    write888(color,width);
}
void VLine(int16_t x,int16_t y0,int16_t y1,uint16_t color){
    width = y1-y0+1;
    setAddrWindow(x,y0,x,y1);
    writecommand(ST7735_RAMWR);
    write888(color,width);
}

void drawCircle(int16_t x0, int16_t y0, int16_t r, uint32_t
color)
{
            int16_t f = 1 - r;
            int16_t ddF_x = 1;
            int16_t ddF_y = -2 * r;
            int16_t x = 0;
            int16_t y = r;
            drawPixel(x0 , y0+r, color);
            drawPixel(x0 , y0-r, color);
            drawPixel(x0+r, y0 , color);
            drawPixel(x0-r, y0 , color);
            while (x<y)
            {
                        if (f >= 0)
                        {
                                    y--;
                                    ddF_y += 2;
                                    f += ddF_y;
                        }
                        x++;
                        ddF_x += 2;
                        f += ddF_x;
                        drawPixel(x0 + x, y0 + y, color);
                        drawPixel(x0 - x, y0 + y, color);
                        drawPixel(x0 + x, y0 - y, color);
                        drawPixel(x0 - x, y0 - y, color);
                        drawPixel(x0 + y, y0 + x, color);
                        drawPixel(x0 - y, y0 + x, color);
                        drawPixel(x0 + y, y0 - x, color);
                        drawPixel(x0 - y, y0 - x, color);
            }
}

//Initialize LCD
void lcd_init()
{
/*
 * portnum     = 0 ;
 * cs          = p0.16 / p0.6 ?
 * rs         = p0.21
 * rst         = p0.22
 */
    uint32_t portnum = 1;
    int i;
    printf("LCD initialized\n");
    /* Notice the hack, for portnum 0 p0.16 is used */
    if ( portnum == 0 )
      {
       LPC_GPIO0->FIODIR |= (0x1<<16);       /* SSP1,
P0.16 defined as Outputs */
      }
      else
      {
       LPC_GPIO0->FIODIR |= (0x1<<6);       /* SSP0 P0.6
defined as Outputs */
      }
    /* Set rs(dc) and rst as outputs */
    LPC_GPIO0->FIODIR |= (0x1<<21);       /* rs/dc P0.21
defined as Outputs */
    LPC_GPIO0->FIODIR |= (0x1<<22);       /* rst P0.22
defined as Outputs */


    /* Reset sequence */
    LPC_GPIO0->FIOSET |= (0x1<<22);
```

```
   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */
   LPC_GPIO0->FIOCLR |= (0x1<<22);
   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */
   LPC_GPIO0->FIOSET |= (0x1<<22);
   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */

   //ask?
    for ( i = 0; i < SSP_BUFSIZE; i++ )    /* Init RD and WR
buffer */
      {
         src_addr[i] = 0;
         dest_addr[i] = 0;
      }

    /* do we need Sw reset (cmd 0x01) ? */

    /* Sleep out */
    SSP_SSELToggle( portnum, 0 );
    src_addr[0] = 0x11;    /* Sleep out */
    SSPSend( portnum, (uint8_t *)src_addr, 1 );
    SSP_SSELToggle( portnum, 1 );

    for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */

   /* delay 200 ms */
   /* Disp on */
    SSP_SSELToggle( portnum, 0 );
    src_addr[0] = 0x29;    /* Disp On */
    SSPSend( portnum, (uint8_t *)src_addr, 1 );
    SSP_SSELToggle( portnum, 1 );
   /* delay 200 ms */
    for(i = 0; i < 1000000; i++);//lcddelay(500);
/*delay 500 ms */
}

//Draw line function
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 makeBackGroundBlack()
{
        int i, j;
        for(i = 0; i < ST7735_TFTWIDTH; i++)
                for(j = 0; j < ST7735_TFTHEIGHT; j++)
                        drawPixel(i, j, 0x00);
}

void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t
w, int16_t h, uint16_t color)
{
        int16_t i, j, byteWidth = (w + 7) / 8;
        for(j=0; j<h; j++)
        {
                for(i=0; i<w; i++ )
                {
                        if(pgm_read_byte(bitmap + j *
byteWidth + i / 8) & (128 >> (i & 7)))
                        {
                                drawPixel(x+i, y+j, color);
                        }
                }
        }
}

void write(uint8_t c, uint8_t textSize, uint16_t textbgcolor,
uint16_t textcolor)
{
        uint8_t wrap = 1;
        if (c == '\n')
        {
                cursor_y += textSize*8;
                cursor_x = 0;
        }
        else if (c == '\r')
        {
                // skip em
        }
        else
        {
                drawChar(cursor_x, cursor_y, c, textcolor,
textbgcolor, textSize);
```

```
                cursor_x += textSize*6;
                if (wrap && (cursor_x > (100 -
textSize*6)))
                {
                        cursor_y += textSize*8;
                        cursor_x = 0;
                }
        }

}

// Draw a character
void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t
color, uint16_t bg, uint8_t size)
{
        int8_t i, j;
        if((x >= _width) || // Clip right
        (y >= _height) || // Clip bottom
        ((x + 6 * size - 1) < 0) || // Clip left
        ((y + 8 * size - 1) < 0)) // Clip top
        return;
        for (i=0; i<6; i++ ) {
        uint8_t line;
        if (i == 5)
        line = 0x0;
        else
        line = pgm_read_byte(font+(c*5)+i);
        for (j = 0; j<8; j++) {
        if (line & 0x1) {
        if (size == 1) // default size
        drawPixel(x+i, y+j, color);
        else { // big size
        fillRect(x+(i*size), y+(j*size), size, size, color);
        }
        } else if (bg != color) {
        if (size == 1) // default size
        drawPixel(x+i, y+j, bg);
        else { // big size
        fillRect(x+i*size, y+j*size, size, size, bg);
        }
        }
        line >>= 1;
        }
        }
}

void fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
uint32_t color)
{
        int16_t i;
        // Update in subclasses if desired!
        for (i=x; i<x+w; i++) {
                drawFastVLine(i, y, h, color);
        }
}



void drawFastVLine(int16_t x, int16_t y, int16_t h, uint32_t
color)
{
        // Update in subclasses if desired!
        drawline(x, y, x, y+h-1, color);
}



void DrawScreenSaver(){
        int
baseCoOrdinates[4][2]={{20,20},{20,70},{70,70},{70,20}};
        int
newCoOrdinates[4][2]={{0,0},{0,0},{0,0},{0,0}};
        uint32_t *color[9]
={0xFF0000,0x003399,0x00CC00,0x9933FF,0x00FFFF,0xFF
6600,0xFF3399,0x006666,0x80000};
        int numSavers=0;
        while(numSavers<10){
        int intSquareSize=rand()%(100)+50;
        int x=rand()%(90)+0;
        int y=rand()%(110)+0;
        baseCoOrdinates[0][0]=x;
        baseCoOrdinates[0][1]=y;
        baseCoOrdinates[1][0]=x;
        baseCoOrdinates[1][1]=y+intSquareSize;
        baseCoOrdinates[2][0]=x+intSquareSize;
        baseCoOrdinates[2][1]=y+intSquareSize;
        baseCoOrdinates[3][0]=x+intSquareSize;
        baseCoOrdinates[3][1]=y;
        int intColor=rand()%(8)+0;
        int saverItrations=0;
        while(saverItrations<10){
        int i=0;

        while(i<4){

                if(i==3){

        drawline(baseCoOrdinates[i][0],baseCoOrdinates[i][
1],baseCoOrdinates[0][0],baseCoOrdinates[0][1],color[intCol
or]);

        newCoOrdinates[i][0]=baseCoOrdinates[i][0]+0.2*(b
aseCoOrdinates[0][0]-baseCoOrdinates[i][0]);

        newCoOrdinates[i][1]=baseCoOrdinates[i][1]+0.2*(b
aseCoOrdinates[0][1]-baseCoOrdinates[i][1]);
                }
                else{

        drawline(baseCoOrdinates[i][0],baseCoOrdinates[i][
1],baseCoOrdinates[i+1][0],baseCoOrdinates[i+1][1],color[int
Color]);

        newCoOrdinates[i][0]=baseCoOrdinates[i][0]+0.2*(b
aseCoOrdinates[i+1][0]-baseCoOrdinates[i][0]);

        newCoOrdinates[i][1]=baseCoOrdinates[i][1]+0.2*(b
aseCoOrdinates[i+1][1]-baseCoOrdinates[i][1]);
                }
                i++;
        }
```

```
	memcpy(baseCoOrdinates, newCoOrdinates,
sizeof(baseCoOrdinates));
	saverItrations++;

	}
	numSavers++;
	}

}

//drawing background for forest
void DrawBackgroundorForest(){

	fillRect(0,0,130,50,0x99D6FF);
	int mountinArray[130][2]={};

	int xAxis=0;
	int yAxis=50;
	int loop10=0;
	int Done10=0;
	while(xAxis<130){
		if(Done10){
			yAxis++;
			mountinArray[xAxis][0]=xAxis;
			mountinArray[xAxis][1]=yAxis;
		}
		else{
			yAxis--;
			mountinArray[xAxis][0]=xAxis;
			mountinArray[xAxis][1]=yAxis;
		}
		;
		if(++loop10==20){
			loop10=0;
			Done10 = !Done10;
		}
		xAxis++;
	}
	int i=0;
	for(i=0;i<130;i++){

	drawline(i,80,mountinArray[i][0],mountinArray[i][1]
,0xC2A385);
	}

	//drawing water

	xAxis=70;
	yAxis=60;
	loop10=0;
	Done10=0;
	while(yAxis<160){
		if(Done10){
			xAxis++;

	drawline(xAxis,yAxis,xAxis+20,yAxis+20,0x99D6F
F);
		}
		else{
			xAxis--;

	drawline(xAxis,yAxis,xAxis+20,yAxis+20,0x99D6F
F);
		}
		;
		if(++loop10==15){
			loop10=0;
			Done10 = !Done10;
		}
		yAxis++;
	}

	i=0;
	for(i=0;i<8;i++){
	drawCircle(10, 20, i, 0xFF6600);
	}
	//fillRect();


}

//drawing a forest
void DrawAForest(){
	int p=0;
	while(p<10){
	//30,150
	int branchAng=120;
	int baseX=rand()%(130)+0;
	int baseY=rand()%(70)+90;
	int trunkBase[]={baseX,baseY};
	int trunkPrime[]={0,0};
	int branchBase[]={0,0};
	int branchPrime0[]={0,0};
	int branchPrime1[]={0,0};
	int branchPrime2[]={0,0};
	//10
	double trunkLength=rand()%(10)+10;
	trunkPrime[0]=trunkBase[0];
	trunkPrime[1]=trunkBase[1]-10;
	branchBase[0]=trunkPrime[0];
	branchBase[1]=trunkPrime[1];
	drawline(trunkBase[0],trunkBase[1],trunkPrime[0],tr
unkPrime[1],0x480000 );
	int mainTrunkIteration=10;
	while(mainTrunkIteration>0){
		trunkBase[0]=trunkPrime[0];
		trunkBase[1]=trunkPrime[1];
		//reducing the length by 70%of the trunk
length
		trunkLength=0.8*trunkLength;
		trunkPrime[1]=trunkBase[1]-trunkLength;
		//draw trunk

	drawline(trunkBase[0],trunkBase[1],trunkPrime[0],tr
unkPrime[1],0x480000 );
		//draw first branch to one side of the trunk

	branchPrime0[0]=cos(branchAng)*trunkPrime[0]-
sin(branchAng)*trunkPrime[1]+trunkBase[0]-
cos(branchAng)*trunkBase[0]+sin(branchAng)*trunkBase[1];

	branchPrime0[1]=sin(branchAng)*trunkPrime[0]+co
```

```
s(branchAng)*trunkPrime[1]+trunkBase[1]-
sin(branchAng)*trunkBase[0]-cos(branchAng)*trunkBase[1];

        drawline(trunkBase[0],trunkBase[1],branchPrime0[0]
,branchPrime0[1],0x480000 );
                int branchIteration=0;
                while(branchIteration<mainTrunkIteration){
                        //shift the branch
                        branchBase[0]=branchPrime0[0];
                        branchBase[1]=branchPrime0[1];
                        //calculate branch prime new
locaton

        branchPrime0[1]=branchPrime0[1]-5;
                        //trunk of the branch - 45*

        branchPrime0[0]=cos(branchAng)*branchPrime0[0]-
sin(branchAng)*branchPrime0[1]+branchBase[0]-
cos(branchAng)*branchBase[0]+sin(branchAng)*branchBase[
1];

        branchPrime0[1]=sin(branchAng)*branchPrime0[0]+
cos(branchAng)*branchPrime0[1]+branchBase[1]-
sin(branchAng)*branchBase[0]-
cos(branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
0[0],branchPrime0[1],0x480000);


        branchPrime1[0]=cos(branchAng)*branchPrime0[0]-
sin(branchAng)*branchPrime0[1]+branchBase[0]-
cos(branchAng)*branchBase[0]+sin(branchAng)*branchBase[
1];

        branchPrime1[1]=sin(branchAng)*branchPrime0[0]+
cos(branchAng)*branchPrime0[1]+branchBase[1]-
sin(branchAng)*branchBase[0]-
cos(branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
1[0],branchPrime1[1],0x006600);

                        branchPrime2[0]=cos(-
branchAng)*branchPrime0[0]-sin(-
branchAng)*branchPrime0[1]+branchBase[0]-cos(-
branchAng)*branchBase[0]+sin(-branchAng)*branchBase[1];
                        branchPrime2[1]=sin(-
branchAng)*branchPrime0[0]+cos(-
branchAng)*branchPrime0[1]+branchBase[1]-sin(-
branchAng)*branchBase[0]-cos(-branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
2[0],branchPrime2[1],0x006600);

                        branchIteration++;
                }

                // draw second branch of the tree

                        branchPrime0[0]=cos(-
branchAng)*trunkPrime[0]-sin(-
branchAng)*trunkPrime[1]+trunkBase[0]-cos(-
branchAng)*trunkBase[0]+sin(-branchAng)*trunkBase[1];
                        branchPrime0[1]=sin(-
branchAng)*trunkPrime[0]+cos(-
branchAng)*trunkPrime[1]+trunkBase[1]-sin(-
branchAng)*trunkBase[0]-cos(-branchAng)*trunkBase[1];

        drawline(trunkBase[0],trunkBase[1],branchPrime0[0]
,branchPrime0[1],0x480000 );
                        branchIteration=0;
                        while(branchIteration<mainTrunkIteration){
                                //shift the branch
                                branchBase[0]=branchPrime0[0];
                                branchBase[1]=branchPrime0[1];
                                //calculate branch prime new
locaton

        branchPrime0[1]=branchPrime0[1]-5;
                                //trunk of the branch - 45*
                                branchPrime0[0]=cos(-
branchAng)*branchPrime0[0]-sin(-
branchAng)*branchPrime0[1]+branchBase[0]-cos(-
branchAng)*branchBase[0]+sin(-branchAng)*branchBase[1];
                                branchPrime0[1]=sin(-
branchAng)*branchPrime0[0]+cos(-
branchAng)*branchPrime0[1]+branchBase[1]-sin(-
branchAng)*branchBase[0]-cos(-branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
0[0],branchPrime0[1],0x480000);


        branchPrime1[0]=cos(branchAng)*branchPrime0[0]-
sin(branchAng)*branchPrime0[1]+branchBase[0]-
cos(branchAng)*branchBase[0]+sin(branchAng)*branchBase[
1];

        branchPrime1[1]=sin(branchAng)*branchPrime0[0]+
cos(branchAng)*branchPrime0[1]+branchBase[1]-
sin(branchAng)*branchBase[0]-
cos(branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
1[0],branchPrime1[1],0x006600);

                        branchPrime2[0]=cos(-
branchAng)*branchPrime0[0]-sin(-
branchAng)*branchPrime0[1]+branchBase[0]-cos(-
branchAng)*branchBase[0]+sin(-branchAng)*branchBase[1];
                        branchPrime2[1]=sin(-
branchAng)*branchPrime0[0]+cos(-
branchAng)*branchPrime0[1]+branchBase[1]-sin(-
branchAng)*branchBase[0]-cos(-branchAng)*branchBase[1];

        drawline(branchBase[0],branchBase[1],branchPrime
2[0],branchPrime2[1],0x006600);

                        branchIteration++;
```

```
                }
                mainTrunkIteration--;
        }
        p++;
        }
};


/*****************************************************
*****************************
**   Main Function  main()
*****************************************************
*****************************/
int main (void)
{
        //EINTInit();
 uint32_t i, portnum = PORT_NUM;
 portnum = 1 ; /* For LCD use 1 */
 if ( portnum == 0 )
        SSP0Init();          /* initialize SSP port */
 else if ( portnum == 1 )
        SSP1Init();
 for ( i = 0; i < SSP_BUFSIZE; i++ )
 {
        src_addr[i] = (uint8_t)i;
        dest_addr[i] = 0;
 }

        //initialize LCD
 lcd_init();
        while(1){
                int optionSel=0;
                printf("Please choose from following
options to implement graphics on the LCD module.\n");
                printf("select 1 for drawing a line.\n");
                printf("select 2 for drawing a screen saver
on to the module.\n");
                printf("select 3 to draw a forest on to the
module.\n");
                scanf("%d",& optionSel);

                //Making the background black in color
                //makeBackGroundBlack();

fillRect(0,0,ST7735_TFTWIDTH,ST7735_TFTHEIGHT,0xD
BB84D);


        if(optionSel==1){
                drawline(10,10,10,89,BLACK);
        }
        else if(optionSel==2){
                DrawScreenSaver();
        }
        else if(optionSel==3){
                DrawBackgroundorForest();
                DrawAForest();
        }
  }
  return 0;
}


/*****************************************************
*****************************
**                End Of File
*****************************************************
*****************************/
```