```c
// Graphics LCD Library
// Jason Losh

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL with LCD/Keyboard Interface
// Target uC:       TM4C123GH6PM
// System Clock:    40 MHz

// Hardware configuration:
// ST7565R Graphics LCD Display Interface:
//   MOSI on PD3 (SSI1Tx)
//   SCLK on PD0 (SSI1Clk)
//   ~CS on PD1 (SSI1Fss)
//   A0 connected to PD2

//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "tm4c123gh6pm.h"
#include "graphics_lcd.h"
#include "gpio.h"
#include "spi1.h"

// Pins
#define A0 PORTD,2

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------

uint8_t  pixelMap[1024];
uint16_t txtIndex = 0;

// 96 character 5x7 bitmaps based on ISO-646 (BCT IRV extensions)
const uint8_t charGen[100][5] = {
    // Codes 32-127
    // Space ! " % $ % & ' ( ) * + , - . /
    {0x00, 0x00, 0x00, 0x00, 0x00},
    {0x00, 0x00, 0x4F, 0x00, 0x00},
    {0x00, 0x07, 0x00, 0x07, 0x00},
    {0x14, 0x7F, 0x14, 0x7F, 0x14},
    {0x24, 0x2A, 0x7F, 0x2A, 0x12},
    {0x23, 0x13, 0x08, 0x64, 0x62},
    {0x36, 0x49, 0x55, 0x22, 0x40},
    {0x00, 0x05, 0x03, 0x00, 0x00},
    {0x00, 0x1C, 0x22, 0x41, 0x00},
    {0x00, 0x41, 0x22, 0x1C, 0x00},
    {0x14, 0x08, 0x3E, 0x08, 0x14},
    {0x08, 0x08, 0x3E, 0x08, 0x08},
    {0x00, 0x50, 0x30, 0x00, 0x00},
    {0x08, 0x08, 0x08, 0x08, 0x08},
    {0x00, 0x60, 0x60, 0x00, 0x00},
    {0x20, 0x10, 0x08, 0x04, 0x02},
    // 0-9
    {0x3E, 0x51, 0x49, 0x45, 0x3E},
    {0x00, 0x42, 0x7F, 0x40, 0x00},
    {0x42, 0x61, 0x51, 0x49, 0x46},
    {0x21, 0x41, 0x45, 0x4B, 0x31},
    {0x18, 0x14, 0x12, 0x7F, 0x10},
    {0x27, 0x45, 0x45, 0x45, 0x39},
    {0x3C, 0x4A, 0x49, 0x49, 0x30},
    {0x01, 0x71, 0x09, 0x05, 0x03},
    {0x36, 0x49, 0x49, 0x49, 0x36},
    {0x06, 0x49, 0x49, 0x29, 0x1E},
    // : ; < = > ? @
    {0x00, 0x36, 0x36, 0x00, 0x00},
    {0x00, 0x56, 0x36, 0x00, 0x00},
    {0x08, 0x14, 0x22, 0x41, 0x00},
    {0x14, 0x14, 0x14, 0x14, 0x14},
    {0x00, 0x41, 0x22, 0x14, 0x08},
    {0x02, 0x01, 0x51, 0x09, 0x3E},
    {0x32, 0x49, 0x79, 0x41, 0x3E},
    // A-Z
    {0x7E, 0x11, 0x11, 0x11, 0x7E},
    {0x7F, 0x49, 0x49, 0x49, 0x36},
    {0x3E, 0x41, 0x41, 0x41, 0x22},
    {0x7F, 0x41, 0x41, 0x22, 0x1C},
    {0x7F, 0x49, 0x49, 0x49, 0x41},
    {0x7F, 0x09, 0x09, 0x09, 0x01},
    {0x3E, 0x41, 0x49, 0x49, 0x3A},
    {0x7F, 0x08, 0x08, 0x08, 0x7F},
    {0x00, 0x41, 0x7F, 0x41, 0x00},
    {0x20, 0x40, 0x41, 0x3F, 0x01},
    {0x7F, 0x08, 0x14, 0x22, 0x41},
    {0x7F, 0x40, 0x40, 0x40, 0x40},
    {0x7F, 0x02, 0x0C, 0x02, 0x7F},
    {0x7F, 0x04, 0x08, 0x10, 0x7F},
    {0x3E, 0x41, 0x41, 0x41, 0x3E},
    {0x7F, 0x09, 0x09, 0x09, 0x06},
    {0x3E, 0x41, 0x51, 0x21, 0x5E},
    {0x7F, 0x09, 0x19, 0x29, 0x46},
    {0x46, 0x49, 0x49, 0x49, 0x31},
    {0x01, 0x01, 0x7F, 0x01, 0x01},
    {0x3F, 0x40, 0x40, 0x40, 0x3F},
    {0x1F, 0x20, 0x40, 0x20, 0x1F},
    {0x3F, 0x40, 0x70, 0x40, 0x3F},
    {0x63, 0x14, 0x08, 0x14, 0x63},
    {0x07, 0x08, 0x70, 0x08, 0x07},
    {0x61, 0x51, 0x49, 0x45, 0x43},
    // [ \ ] ^ _ `
    {0x00, 0x7F, 0x41, 0x41, 0x00},
    {0x02, 0x04, 0x08, 0x10, 0x20},
    {0x00, 0x41, 0x41, 0x7F, 0x00},
    {0x04, 0x02, 0x01, 0x02, 0x04},
    {0x40, 0x40, 0x40, 0x40, 0x40},
    {0x00, 0x01, 0x02, 0x04, 0x00},
    // a-z
    {0x20, 0x54, 0x54, 0x54, 0x78},
    {0x7F, 0x44, 0x44, 0x44, 0x38},
    {0x38, 0x44, 0x44, 0x44, 0x20},
    {0x38, 0x44, 0x44, 0x48, 0x7F},
    {0x38, 0x54, 0x54, 0x54, 0x18},
    {0x08, 0x7E, 0x09, 0x01, 0x02},
    {0x0C, 0x52, 0x52, 0x52, 0x3E},
    {0x7F, 0x08, 0x04, 0x04, 0x78},
    {0x00, 0x44, 0x7D, 0x40, 0x00},
    {0x20, 0x40, 0x44, 0x3D, 0x00},
    {0x7F, 0x10, 0x28, 0x44, 0x00},
    {0x00, 0x41, 0x7F, 0x40, 0x00},
    {0x7C, 0x04, 0x18, 0x04, 0x78},
    {0x7C, 0x08, 0x04, 0x04, 0x78},
    {0x38, 0x44, 0x44, 0x44, 0x38},
    {0x7C, 0x14, 0x14, 0x14, 0x08},
    {0x08, 0x14, 0x14, 0x18, 0x7C},
    {0x7C, 0x08, 0x04, 0x04, 0x08},
    {0x48, 0x54, 0x54, 0x54, 0x20},
    {0x04, 0x3F, 0x44, 0x40, 0x20},
    {0x3C, 0x40, 0x40, 0x20, 0x7C},
    {0x1C, 0x20, 0x40, 0x20, 0x1C},
    {0x3C, 0x40, 0x20, 0x40, 0x3C},
    {0x44, 0x28, 0x10, 0x28, 0x44},
    {0x0C, 0x50, 0x50, 0x50, 0x3C},
    {0x44, 0x64, 0x54, 0x4C, 0x44},
    // { | } ~ cc
    {0x00, 0x08, 0x36, 0x41, 0x00},
    {0x00, 0x00, 0x7F, 0x00, 0x00},
    {0x00, 0x41, 0x36, 0x08, 0x00},
    {0x0C, 0x04, 0x1C, 0x10, 0x18},
    {0x00, 0x00, 0x00, 0x00, 0x00},
    // Custom assignments beyond ISO646
    // Codes 128+: right arrow, left arrow, degree sign
    {0x08, 0x08, 0x2A, 0x1C, 0x08},
    {0x08, 0x1C, 0x2A, 0x08, 0x08},
    {0x07, 0x05, 0x07, 0x00, 0x00},
};

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

// Blocking function that writes data to the SPI bus and waits for the data to complete
transmission
void sendGraphicsLcdCommand(uint8_t command)
{
    setPinValue(A0, 0);                 // clear A0 for commands
    writeSpi1Data(command);
}

// Blocking function that writes data to the SPI bus and waits for the data to complete
transmission
void sendGraphicsLcdData(uint8_t data)
{
    setPinValue(A0, 1);                 // set A0 for data
    writeSpi1Data(data);
}

void setGraphicsLcdPage(uint8_t page)
{
  sendGraphicsLcdCommand(0xB0 | page);
}

void setGraphicsLcdColumn(uint8_t x)
{
  sendGraphicsLcdCommand(0x10 | (x >> 4));
  sendGraphicsLcdCommand(0x00 | (x & 0x0F));
}

void refreshGraphicsLcd()
{
    uint8_t x, page;
    uint16_t i = 0;
    for (page = 0; page < 8; page ++)
    {
        setGraphicsLcdPage(page);
        setGraphicsLcdColumn(0);
        for (x = 0; x < 128; x++)
            sendGraphicsLcdData(pixelMap[i++]);
    }
}

void clearGraphicsLcd()
{
    uint16_t i;
    // clear data memory pixel map
    for (i = 0; i < 1024; i++)
        pixelMap[i] = 0;
    // copy to display
    refreshGraphicsLcd();
}
```

```c
void drawGraphicsLcdPixel(uint8_t x, uint8_t y, enum operation op)
{
    uint8_t data, mask, page;
    uint16_t index;

    // determine pixel map entry
    page = y >> 3;

    // determine pixel map index
    index = page << 7 | x;

    // generate mask
    mask = 1 << (y & 7);

    // read pixel map
    data = pixelMap[index];

    // apply operator
    switch(op)
    {
        case CLEAR:  data &= ~mask; break;
        case SET:    data |= mask; break;
        case INVERT: data ^= mask; break;
    }

    // write to pixel map
    pixelMap[index] = data;

    // write to display
    setGraphicsLcdPage(page);
    setGraphicsLcdColumn(x);
    sendGraphicsLcdData(data);
}

void drawGraphicsLcdRectangle(uint8_t xul, uint8_t yul, uint8_t dx, uint8_t dy, enum
operation op)
{
    uint8_t page, page_start, page_stop;
    uint8_t bit_index, bit_start, bit_stop;
    uint8_t mask, data;
    uint16_t index;
    uint8_t x;

    // determine pages for rectangle
    page_start = yul >> 3;
    page_stop = (yul + dy - 1) >> 3;

    // draw in pages from top to bottom within extent
    for (page = page_start; page <= page_stop; page++)
    {
        // calculate mask for this page
        if (page > page_start)
            bit_start = 0;
        else
            bit_start = yul & 7;
        if (page < page_stop)
            bit_stop = 7;
        else
            bit_stop = (yul + dy - 1) & 7;
        mask = 0;
        for (bit_index = bit_start; bit_index <= bit_stop; bit_index++)
            mask |= 1 << bit_index;

        // write page
        setGraphicsLcdPage(page);
        setGraphicsLcdColumn(xul);
        index = (page << 7) | xul;
        for (x = 0; x < dx; x++)
        {
            // read pixel map
            data = pixelMap[index];
            // apply operator (0 = clear, 1 = set, 2 = xor)
            switch(op)
            {
                case CLEAR:  data &= ~mask; break;
                case SET:    data |= mask; break;
                case INVERT: data ^= mask; break;
            }
            // write to pixel map
            pixelMap[index++] = data;
            // write to display
            sendGraphicsLcdData(data);
        }
    }
}

void setGraphicsLcdTextPosition(uint8_t x, uint8_t page)
{
    txtIndex = (page << 7) + x;
    setGraphicsLcdPage(page);
    setGraphicsLcdColumn(x);
}

void putcGraphicsLcd(char c)
{
    uint8_t i, val;
    uint8_t uc;
    // convert to unsigned to access characters > 127
    uc = (uint8_t) c;
    for (i = 0; i < 5; i++)
    {
        val = charGen[uc-' '][i];
        pixelMap[txtIndex++] = val;
        sendGraphicsLcdData(val);
    }
    pixelMap[txtIndex++] = 0;
    sendGraphicsLcdData(0);
}

void putsGraphicsLcd(char str[])
{
    uint8_t i = 0;
    while (str[i] != 0)
        putcGraphicsLcd(str[i++]);
}

void initGraphicsLcd()
{
    // Initialize SPI1 interface
    initSpi1(USE_SSI_FSS);
    setSpi1BaudRate(1e6, 40e6);
    setSpi1Mode(1, 1);

    // Enable clocks
    enablePort(PORTD);

    // Configure A0 for graphics LCD
    selectPinPushPullOutput(A0);

    // Initialize display
    sendGraphicsLcdCommand(0x40); // set start line to 0
    sendGraphicsLcdCommand(0xA1); // reverse horizontal order
    sendGraphicsLcdCommand(0xC0); // normal vertical order
    sendGraphicsLcdCommand(0xA6); // normal pixel polarity
    sendGraphicsLcdCommand(0xA2); // set led bias to 1/9 (should be A2)
    sendGraphicsLcdCommand(0x2F); // turn on voltage booster and regulator
    sendGraphicsLcdCommand(0xF8); // set internal volt booster to 4x Vdd
    sendGraphicsLcdCommand(0x00);
    sendGraphicsLcdCommand(0x27); // set contrast
    sendGraphicsLcdCommand(0x81); // set LCD drive voltage
    sendGraphicsLcdCommand(0x16);
    sendGraphicsLcdCommand(0xAC); // no flashing indicator
    sendGraphicsLcdCommand(0x00);
    clearGraphicsLcd();           // clear display
    sendGraphicsLcdCommand(0xAF); // display on
}
```