```c
// Graphics LCD Example
// Jason Losh

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

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

// Hardware configuration:
// Red Backlight LED:
//   PB5 drives an NPN transistor that powers the red LED
// Green Backlight LED:
//   PE5 drives an NPN transistor that powers the green LED
// Blue Backlight LED:
//   PE4 drives an NPN transistor that powers the blue LED
// ST7565R Graphics LCD Display Interface:
//   MOSI (SSI2Tx) on PB7
//   MISO (SSI2Rx) is not used by the LCD display but the pin is used for GPIO for A0
//   SCLK (SSI2Clk) on PB4
//   A0 connected to PB6
//   ~CS connected to PB1

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

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "tm4c123gh6pm.h"

#define RED_BL_LED   (*((volatile uint32_t *)(0x42000000 + (0x400053FC-0x40000000)*32 + 5*4)))
#define GREEN_BL_LED (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 5*4)))
#define BLUE_BL_LED  (*((volatile uint32_t *)(0x42000000 + (0x400243FC-0x40000000)*32 + 4*4)))

#define CS_NOT       (*((volatile uint32_t *)(0x42000000 + (0x400053FC-0x40000000)*32 + 1*4)))
#define A0           (*((volatile uint32_t *)(0x42000000 + (0x400053FC-0x40000000)*32 + 6*4)))

// Set pixel arguments
#define CLEAR  0
#define SET    1
#define INVERT 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
```

```
169 //-----------------------------------------------------------------------------
170
171 // Initialize Hardware
172 void initHw()
173 {
174     // Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz
175     SYSCTL_RCC_R = SYSCTL_RCC_XTAL_16MHZ | SYSCTL_RCC_OSCSRC_MAIN | SYSCTL_RCC_USESYSDIV | (4
    << SYSCTL_RCC_SYSDIV_S);
176
177     // Set GPIO ports to use APB (not needed since default configuration -- for clarity)
178     // Note UART on port A must use APB
179     SYSCTL_GPIOHBCTL_R = 0;
180
181     // Enable GPIO port B and E peripherals
182     SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOB | SYSCTL_RCGC2_GPIOE;
183
184     // Configure three backlight LEDs
185     GPIO_PORTB_DIR_R |= 0x20;  // make bit5 an output
186     GPIO_PORTB_DR2R_R |= 0x20; // set drive strength to 2mA
187     GPIO_PORTB_DEN_R |= 0x20;  // enable bit5 for digital
188     GPIO_PORTE_DIR_R |= 0x30;  // make bits 4 and 5 outputs
189     GPIO_PORTE_DR2R_R |= 0x30; // set drive strength to 2mA
190     GPIO_PORTE_DEN_R |= 0x30;  // enable bits 4 and 5 for digital
191
192     // Configure A0 and ~CS for graphics LCD
193     GPIO_PORTB_DIR_R |= 0x42;  // make bits 1 and 6 outputs
194     GPIO_PORTB_DR2R_R |= 0x42; // set drive strength to 2mA
195     GPIO_PORTB_DEN_R |= 0x42;  // enable bits 1 and 6 for digital
196
197     // Configure SSI2 pins for SPI configuration
198     SYSCTL_RCGCSSI_R |= SYSCTL_RCGCSSI_R2;           // turn-on SSI2 clocking
199     GPIO_PORTB_DIR_R |= 0x90;                        // make bits 4 and 7 outputs
200     GPIO_PORTB_DR2R_R |= 0x90;                       // set drive strength to 2mA
201     GPIO_PORTB_AFSEL_R |= 0x90;                      // select alternative functions for MOSI,
    SCLK pins
202     GPIO_PORTB_PCTL_R = GPIO_PCTL_PB7_SSI2TX | GPIO_PCTL_PB4_SSI2CLK; // map alt fns to SSI2
203     GPIO_PORTB_DEN_R |= 0x90;                        // enable digital operation on TX, CLK
    pins
204
205     // Configure the SSI2 as a SPI master, mode 3, 8bit operation, 1 MHz bit rate
206     SSI2_CR1_R &= ~SSI_CR1_SSE;                      // turn off SSI2 to allow
    re-configuration
207     SSI2_CR1_R = 0;                                  // select master mode
208     SSI2_CC_R = 0;                                   // select system clock as the clock
    source
209     SSI2_CPSR_R = 40;                                // set bit rate to 1 MHz (if SR=0 in CR0)
210     SSI2_CR0_R = SSI_CR0_SPH | SSI_CR0_SPO | SSI_CR0_FRF_MOTO | SSI_CR0_DSS_8; // set SR=0,
    mode 3 (SPH=1, SPO=1), 8-bit
211     SSI2_CR1_R |= SSI_CR1_SSE;                       // turn on SSI2
212 }
213
214 // Approximate busy waiting (in units of microseconds), given a 40 MHz system clock
215 void waitMicrosecond(uint32_t us)
216 {
217                                                  // Approx clocks per us
218     __asm("WMS_LOOP0:   MOV  R1, #6");           // 1
```

```
219     __asm("WMS_LOOP1:   SUB  R1, #1");          // 6
220     __asm("             CBZ  R1, WMS_DONE1");   // 5+1*3
221     __asm("             NOP");                  // 5
222     __asm("             B    WMS_LOOP1");       // 5*3
223     __asm("WMS_DONE1:   SUB  R0, #1");          // 1
224     __asm("             CBZ  R0, WMS_DONE0");   // 1
225     __asm("             B    WMS_LOOP0");       // 1*3
226     __asm("WMS_DONE0:");                        // ---
227                                                 // 40 clocks/us + error
228 }
229
230 // Blocking function that writes data to the SPI bus and waits for the data to complete
    transmission
231 void sendGraphicsLcdCommand(uint8_t command)
232 {
233     CS_NOT = 0;                         // assert chip select
234     __asm (" NOP");                     // allow line to settle
235     __asm (" NOP");
236     __asm (" NOP");
237     __asm (" NOP");
238     A0 = 0;                             // clear A0 for commands
239     SSI2_DR_R = command;                // write command
240     while (SSI2_SR_R & SSI_SR_BSY);
241     CS_NOT = 1;                         // de-assert chip select
242 }
243
244 // Blocking function that writes data to the SPI bus and waits for the data to complete
    transmission
245 void sendGraphicsLcdData(uint8_t data)
246 {
247     CS_NOT = 0;                         // assert chip select
248     __asm (" NOP");                     // allow line to settle
249     __asm (" NOP");
250     __asm (" NOP");
251     __asm (" NOP");
252     A0 = 1;                             // set A0 for data
253     SSI2_DR_R = data;                   // write data
254     while (SSI2_SR_R & SSI_SR_BSY);     // wait for transmission to stop
255     CS_NOT = 1;                         // de-assert chip select
256 }
257
258 void setGraphicsLcdPage(uint8_t page)
259 {
260   sendGraphicsLcdCommand(0xB0 | page);
261 }
262
263 void setGraphicsLcdColumn(uint8_t x)
264 {
265   sendGraphicsLcdCommand(0x10 | ((x >> 4) & 0x0F));
266   sendGraphicsLcdCommand(0x00 | (x & 0x0F));
267 }
268
269 void refreshGraphicsLcd()
270 {
271     uint8_t x, page;
272     uint16_t i = 0;
```

```
273     for (page = 0; page < 8; page ++)
274     {
275         setGraphicsLcdPage(page);
276         setGraphicsLcdColumn(0);
277         for (x = 0; x < 128; x++)
278             sendGraphicsLcdData(pixelMap[i++]);
279     }
280 }
281 
282 void clearGraphicsLcd()
283 {
284     uint16_t i;
285     // clear data memory pixel map
286     for (i = 0; i < 1024; i++)
287         pixelMap[i] = 0;
288     // copy to display
289     refreshGraphicsLcd();
290 }
291 
292 void initGraphicsLcd()
293 {
294     sendGraphicsLcdCommand(0x40); // set start line to 0
295     sendGraphicsLcdCommand(0xA1); // reverse horizontal order
296     sendGraphicsLcdCommand(0xC0); // normal vertical order
297     sendGraphicsLcdCommand(0xA6); // normal pixel polarity
298     sendGraphicsLcdCommand(0xA3); // set led bias to 1/9 (should be A2)
299     sendGraphicsLcdCommand(0x2F); // turn on voltage booster and regulator
300     sendGraphicsLcdCommand(0xF8); // set internal volt booster to 4x Vdd
301     sendGraphicsLcdCommand(0x00);
302     sendGraphicsLcdCommand(0x27); // set contrast
303     sendGraphicsLcdCommand(0x81); // set LCD drive voltage
304     sendGraphicsLcdCommand(0x04);
305     sendGraphicsLcdCommand(0xAC); // no flashing indicator
306     sendGraphicsLcdCommand(0x00);
307     clearGraphicsLcd();           // clear display
308     sendGraphicsLcdCommand(0xAF); // display on
309 }
310 
311 void drawGraphicsLcdPixel(uint8_t x, uint8_t y, uint8_t op)
312 {
313     uint8_t data, mask, page;
314     uint16_t index;
315 
316     // determine pixel map entry
317     page = y >> 3;
318 
319     // determine pixel map index
320     index = page << 7 | x;
321 
322     // generate mask
323     mask = 1 << (y & 7);
324 
325     // read pixel map
326     data = pixelMap[index];
327 
328     // apply operator (0 = clear, 1 = set, 2 = xor)
```

```
329     switch(op)
330     {
331         case 0: data &= ~mask; break;
332         case 1: data |= mask; break;
333         case 2: data ^= mask; break;
334     }
335 
336     // write to pixel map
337     pixelMap[index] = data;
338 
339     // write to display
340     setGraphicsLcdPage(page);
341     setGraphicsLcdColumn(x);
342     sendGraphicsLcdData(data);
343 }
344 
345 void drawGraphicsLcdRectangle(uint8_t xul, uint8_t yul, uint8_t dx, uint8_t dy, uint8_t op)
346 {
347     uint8_t page, page_start, page_stop;
348     uint8_t bit_index, bit_start, bit_stop;
349     uint8_t mask, data;
350     uint16_t index;
351     uint8_t x;
352 
353     // determine pages for rectangle
354     page_start = yul >> 3;
355     page_stop = (yul + dy - 1) >> 3;
356 
357     // draw in pages from top to bottom within extent
358     for (page = page_start; page <= page_stop; page++)
359     {
360         // calculate mask for this page
361         if (page > page_start)
362             bit_start = 0;
363         else
364             bit_start = yul & 7;
365         if (page < page_stop)
366             bit_stop = 7;
367         else
368             bit_stop = (yul + dy - 1) & 7;
369         mask = 0;
370         for (bit_index = bit_start; bit_index <= bit_stop; bit_index++)
371             mask |= 1 << bit_index;
372 
373         // write page
374         setGraphicsLcdPage(page);
375         setGraphicsLcdColumn(xul);
376         index = (page << 7) | xul;
377         for (x = 0; x < dx; x++)
378         {
379             // read pixel map
380             data = pixelMap[index];
381             // apply operator (0 = clear, 1 = set, 2 = xor)
382             switch(op)
383             {
384                 case 0: data &= ~mask; break;
```

```
385                 case 1: data |= mask; break;
386                 case 2: data ^= mask; break;
387             }
388             // write to pixel map
389             pixelMap[index++] = data;
390             // write to display
391             sendGraphicsLcdData(data);
392         }
393     }
394 }
395
396 void setGraphicsLcdTextPosition(uint8_t x, uint8_t page)
397 {
398     txtIndex = (page << 7) + x;
399     setGraphicsLcdPage(page);
400     setGraphicsLcdColumn(x);
401 }
402
403 void putcGraphicsLcd(char c)
404 {
405     uint8_t i, val;
406     uint8_t uc;
407     // convert to unsigned to access characters > 127
408     uc = (uint8_t) c;
409     for (i = 0; i < 5; i++)
410     {
411         val = charGen[uc-' '][i];
412         pixelMap[txtIndex++] = val;
413         sendGraphicsLcdData(val);
414     }
415     pixelMap[txtIndex++] = 0;
416     sendGraphicsLcdData(0);
417 }
418
419 void putsGraphicsLcd(char str[])
420 {
421     uint8_t i = 0;
422     while (str[i] != 0)
423         putcGraphicsLcd(str[i++]);
424 }
425
426 //-----------------------------------------------------------------------------
427 // Main
428 //-----------------------------------------------------------------------------
429
430 int main(void)
431 {
432     // Initialize hardware
433     initHw();
434
435     // Turn-on all LEDs to create white backlight
436     RED_BL_LED = 1;
437     GREEN_BL_LED = 1;
438     BLUE_BL_LED = 1;
439
440     // Initialize graphics LCD
```

```
441     initGraphicsLcd();
442
443     // Draw X in left half of screen
444     uint8_t i;
445     for (i = 0; i < 64; i++)
446         drawGraphicsLcdPixel(i, i, SET);
447     for (i = 0; i < 64; i++)
448         drawGraphicsLcdPixel(63-i, i, INVERT);
449
450     // Draw text on screen
451     setGraphicsLcdTextPosition(84, 5);
452     putsGraphicsLcd("Text");
453
454     // Draw flashing block around the text
455     while(1)
456     {
457         drawGraphicsLcdRectangle(83, 39, 25, 9, INVERT);
458         waitMicrosecond(500000);
459     }
460 }
461
```