EEL 4742C: Embedded Systems

Christopher Badolato

12/2/2019

EEL4742-0011

Lab 11 SPI to LCD

Introduction

In this lab experiment we will be programming the serial peripheral interface (SPI) and the LCD pixel display. SPI will be configured to talk to the between the LCD and the board. in part one we configured the middle driver file to configure the SPI using functions HAL\_LCD\_SpiInit() and HAL\_LCD\_PortInit() (Configuration registers found in family user guide at the end of chapter 31). Once we have configured SPI and the LCD, we can call functions from the grlib.h file to draw or write what we would like to the LCD display.

Part 11.1: Serial Peripheral Interface (SPI)

|  |
| --- |
| Same middle driver used for part 1 and part 2, each has its own main calling configuration functions in main.c which is in the table row below the middle driver code.  This code for main is below. The main is configured to write the given text message to the LCD.  We first configured the SPI pins following the table taken from the lab manual, we will configure the pins accordingly |
| //MIDDLE\_DRIVER.c  //Christopher Badolato  //12/2/2019  //Lab 11.1  //EEL 4742 0011  //SPI AND LCD  **#include** "msp430fr6989.h"  **#include** "LcdDriver/middle\_driver.h"  **#include** "Grlib/grlib/grlib.h"  **#include** <stdint.h>  **void** **HAL\_LCD\_PortInit**(**void**){  // Configure the SPI pins  // P1.4 pin to serial clock  P1SEL1 &= ~BIT4;  P1SEL0 |= BIT4;  // P1.6 pin to SIMO  P1SEL1 &= ~BIT6;  P1SEL0 |= BIT6;  //display pins  //Set reset pin as output  P9DIR |= BIT4;  P9SEL1&=~BIT4;  P9SEL0&=~BIT4;  //data/command pin as output  P2DIR |= BIT3;  P2SEL1&=~ BIT3;  P2SEL0 &=~ BIT3;  //chip select pin as output  P2DIR |= BIT5;  P2SEL1&=~ BIT5;  P2SEL0 &=~ BIT5;  **return**;  }  **void** **HAL\_LCD\_SpiInit**(**void**){  //spi config  // Put eUSCI in reset state while modifying the configuration  UCB0CTLW0 |= UCSWRST;  //clock phase to capture on 1st edge, change on following edge  UCB0CTLW0 |= UCCKPH;  //clock polarity to inactive low  UCB0CTLW0 &=~ UCCKPL;  //data order to transmit MSB first  UCB0CTLW0 |= UCMSB;  //MCU to SPI master  UCB0CTLW0 |= UCMST;  //SPI to 3 pin SPI  UCB0CTLW0 |= UCMODE\_0;  //module to synchronous mode  UCB0CTLW0 |= UCSYNC;  //clock to SMCLK  UCB0CTLW0 |= UCSSEL\_3;  //Set clock divider to 1 (SMCLK is from DCO at 8 MHz; we'll run SPI at 8 MHz)  UCB0BRW =1;  //Exit the reset state at the end of the configuration  UCB0CTLW0 &= ~UCSWRST;  // chip select bit to 0  P2OUT &= ~BIT5;  //DC bit to 0  P2OUT &= ~BIT3;  **return**;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Writes a command to the CFAF128128B-0145T. This function implements the basic SPI  // interface to the LCD display.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **HAL\_LCD\_writeCommand**(uint8\_t command)  {  // For command, set the DC' bit to low before transmission  P2OUT &= ~BIT3;  // Wait as long as the module is busy  **while** (UCB0STATW & UCBUSY);  // Transmit data  UCB0TXBUF = command;  // Set DC' bit back to high  P2OUT |= BIT3;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Writes a data to the CFAF128128B-0145T. This function implements the basic SPI  // interface to the LCD display.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **HAL\_LCD\_writeData**(uint8\_t data)  {  // Wait as long as the module is busy  **while** (UCB0STATW & UCBUSY);  // Transmit data  UCB0TXBUF = data;  } |
| //Christopher Badolato  //12/2/2019  //Lab 11.1  //EEL 4742 0011  //SPI AND LCD  **#include** "msp430fr6989.h"  **#include** "Grlib/grlib/grlib.h" // Graphics library (grlib)  **#include** "LcdDriver/lcd\_driver.h" // LCD driver  **#include** <stdio.h>  **#define** redLED BIT0  **#define** greenLED BIT7  **#define** button BIT1  **void** **main**(**void**){  **volatile** **unsigned** **int** counter=0;  **char** mystring[20];  **unsigned** **int** n;  WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer  PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode  P1DIR |= redLED; P1OUT &= ~redLED;  P9DIR |= greenLED; P9OUT &= ~greenLED;  P1DIR &= ~button; P1REN|=button; P1OUT|=button; // button, resistor, pullup  // Configure SMCLK to 8 MHz (used as SPI clock)  CSCTL0 = CSKEY; // Unlock CS registers  CSCTL3 &= ~(BIT4|BIT5|BIT6); // DIVS=0  CSCTL0\_H = 0; // Relock the CS registers  ////////////////////////////////////////////////////////////////////////////////////////////  Graphics\_Context g\_sContext; // Declare a graphic library context  Crystalfontz128x128\_Init(); // Initialize the display  // Set the screen orientation  Crystalfontz128x128\_SetOrientation(0);  // Initialize the context  Graphics\_initContext(&g\_sContext, &g\_sCrystalfontz128x128);  // Set background and foreground colors  Graphics\_setBackgroundColor(&g\_sContext, GRAPHICS\_COLOR\_BLACK);  Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_WHITE);  // Set the default font for strings  GrContextFontSet(&g\_sContext, &g\_sFontFixed6x8);  // Clear the screen  Graphics\_clearDisplay(&g\_sContext);    Graphics\_drawStringCentered(&g\_sContext, "Welcome to", AUTO\_STRING\_LENGTH, 64, 30, OPAQUE\_TEXT);  **sprintf**(mystring, "EEL 4742 Lab!");  Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 55, OPAQUE\_TEXT);  n = 1234;  **sprintf**(mystring, "%d", n);  Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 80, OPAQUE\_TEXT);  **while**(1){}  } |

Part 11.2: Using the Graphics Library

|  |
| --- |
| Part 11.2 is similar to the part 1, we will just change our drawing functions to do what we would like and be creative. We will be using the same middledriver.c to configure the graphics and LCD screen.   1. Set a new background color 2. Set a new foreground color 3. Draw at least one of each: an outline circle, a filled circle, an outline rectangle, a filled   rectangle and a horizontal line.   1. Use at least three different colors on your screen (open the file grlib.h in the project to   see the available colors)   1. Draw an image on the first screen (use the image in the file logo.c) 2. Setup an incrementing 8-bit counter on the second screen (it should continue counting after rollback to zero) 3. Pushing the button transitions back and forth between the two screens 4. Set the GPIO pin P2.6 to high to engage the highest brightness level 5. Use two fonts on the screen (the project contains fonts in the folder GrLib/fonts)   I was having difficulty getting the UCF image to display. |
| //Christopher Badolato  //12/2/2019  //Lab 11.2  //EEL 4742 0011  //SPI AND LCD  **#include** "msp430fr6989.h"  **#include** "Grlib/grlib/grlib.h" // Graphics library (grlib)  **#include** "LcdDriver/lcd\_driver.h" // LCD driver  **#include** <stdio.h>  **#define** redLED BIT0  **#define** greenLED BIT7  **#define** button BIT1  **volatile** **unsigned** **int** counter=0;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Configures ACLK to 32 KHz crystal  **void** **config\_ACLK\_to\_32KHz\_crystal**(){  // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz  // Reroute pins to LFXIN/LFXOUT functionality  PJSEL1 &= ~BIT4;  PJSEL0 |= BIT4;  // Wait until the oscillator fault flags remain cleared  CSCTL0 = CSKEY; // Unlock CS registers  **do** {  CSCTL5 &= ~LFXTOFFG; // Local fault flag  SFRIFG1 &= ~OFIFG; // Global fault flag  } **while**((CSCTL5 & LFXTOFFG) != 0);  CSCTL0\_H = 0; // Lock CS registers  **return**;  }  **void** **main**(**void**){  **char** mystring[20];  **unsigned** **int** n = 0;  //create a new rectangle objects  Graphics\_Rectangle newRectangle, rectangle2;  newRectangle.xMax = 44;  newRectangle.yMax = 0;  newRectangle.xMin = 0;  newRectangle.yMin = 22;  rectangle2.xMax = 100;  rectangle2.xMin = 54;  rectangle2.yMax = 64;  rectangle2.yMin = 54;  //new ucf image object  tImage newImage;      WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer  PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode  P1DIR |= redLED; P1OUT &= ~redLED;  P9DIR |= greenLED; P9OUT &= ~greenLED;  P1DIR &= ~button; P1REN|=button; P1OUT|=button; // button, resistor, pullup  // Configure SMCLK to 8 MHz (used as SPI clock)  CSCTL0 = CSKEY; // Unlock CS registers  CSCTL3 &= ~(BIT4|BIT5|BIT6); // DIVS=0  CSCTL0\_H = 0; // Relock the CS registers  ////////////////////////////////////////////////////////////////////////////////////////////  Graphics\_Context g\_sContext; // Declare a graphic library context  Crystalfontz128x128\_Init(); // Initialize the display  // Set the screen orientation  Crystalfontz128x128\_SetOrientation(0);  // Initialize the context  Graphics\_initContext(&g\_sContext, &g\_sCrystalfontz128x128);      //used timer interrupts to incement counter  config\_ACLK\_to\_32KHz\_crystal();  TA0CTL = (TASSEL\_1|ID\_0|MC\_2|TACLR|TAIE);  TA0CTL &= ~TAIFG;  \_enable\_interrupts();    //turn brightness on high  P2OUT |= BIT6;    //I was having difficulty displaying the image, it wouldn't let me reference tImage &logo4BPP\_UNCOMP  Graphics\_drawImage(&g\_sContext, &newImage, 0, 0);  // Set background and foreground colors  Graphics\_setBackgroundColor(&g\_sContext, GRAPHICS\_COLOR\_ROYAL\_BLUE);  Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_PLUM);  //set the font  GrContextFontSet(&g\_sContext, &g\_sFontCmtt12);  // Clear the screen  Graphics\_clearDisplay(&g\_sContext);  //was having difficulty displaying  Graphics\_drawImage(&g\_sContext, &newImage, 0, 0);  //write my name  Graphics\_drawStringCentered(&g\_sContext, "Chris Badolato", AUTO\_STRING\_LENGTH, 64, 30, GRAPHICS\_COLOR\_LIGHT\_YELLOW);  //draw left circle  Graphics\_drawCircle(&g\_sContext, 11, 11, 10);  //draw right circle  Graphics\_drawCircle(&g\_sContext, 33, 11, 10);  Graphics\_fillCircle(&g\_sContext, 33, 11, 10);  //draw unfilled rectangle aroudn circles  Graphics\_drawRectangle(&g\_sContext, &newRectangle);  //draw filled center rectangle  Graphics\_drawRectangle(&g\_sContext, &rectangle2);  Graphics\_fillRectangle(&g\_sContext, &rectangle2);  //change the font before showing the counter.  GrContextFontSet(&g\_sContext, &g\_sFontFixed6x8);  **while**(1){  // every second the incrementor will be increased and redisplayed on the LCD.  n = counter;  **sprintf**(mystring, "%d", n);  Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 40, GRAPHICS\_COLOR\_LIGHT\_YELLOW);  }  }  **#pragma** vector = TIMER0\_A1\_VECTOR // Link the ISR to the vector  **\_\_interrupt** **void** **T0A1\_ISR**() {  // Toggle both LEDs  counter++;  **if**(counter >= 32768){  counter = 0;  }  P9OUT ^= greenLED;  P1OUT ^= redLED;  // Clear the TAIFG flag  TA0CTL &= ~TAIFG;  } |

Student Q&A

1. Is the SPI implemented as half-duplex or full duplex in this experiment?

|  |
| --- |
| SPI can be configured to full duplex but in this experiment, we are using half-duplex |

1. What SPI clock frequency did we set up?

|  |
| --- |
| 8 MHz clock frequency |

3. What is the maximum SPI clock frequency that is supported by the eUSCI module? Look

in the microcontroller’s data sheet in Table 5-18.

|  |
| --- |
| eUSCI input clock frequency maximum = 16 MHz |

4. Is the display driver software specific to a display model or could it work with any display?

Explain.

|  |
| --- |
| The driver is not specific to the display model, we need to only change the SPI configuration to match the device we are trying to project images to. |

5. Is the graphics library (e.g. grlib) specific to a display model or could it work with any

display? Explain.

|  |
| --- |
| It could work on any display, we need only change the parameters of the functions we are calling as well as the SPI configuration. |

Conclusion

In conclusion, in this experiment we configured the serial peripheral interface to display on the Booster Pack LCD. First, we simply displayed a given program after configuring the middleDriver functions that will configure the LCD and SPI. In the second part of the lab we created our own design with the grlib functions such as, drawing shapes, changing fonts