**11/23/2020**

Joshua Frazer

Lab 11

Serial Peripheral Interface (SPI) & LCD Pixel Display

**Introduction:**

Much knowledge is gained from this lab about SPI. For part 1, we are required to set up the middle driver to get the whole project up and functional. For part 2, we elaborate more on the display itself creating, shapes, lines, constant output, and text.

**Part 1:**

This part tripped me up a bit. For some reason I didn’t realize until lab day that all pins were given to me via the lab manual. I kept looking through the booster pack manual and the family users guide, which in turn never gave me the true values of the pins. Once I found out that the pins I needed were given, everything was straightforward and followed the same technique as the previous lab.

// This code was ported from TI's sample code. See Copyright notice at the bottom of this file.

**#include** "msp430fr6989.h"

**#include** "LcdDriver/middle\_driver.h"

**#include** "Grlib/grlib/grlib.h"

**#include** <stdint.h>

**void** **HAL\_LCD\_PortInit**(**void**)

{

/////////////////////////////////////

// Configuring the SPI pins

/////////////////////////////////////

// Divert UCB0CLK/P1.4 pin to serial clock

P1SEL1 &= ~BIT4;

P1SEL0 |= BIT4;

// Divert UCB0SIMO/P1.6 pin to SIMO

P1SEL1 &= ~BIT6;

P1SEL0 |= BIT6;

// OK to ignore UCB0STE/P1.5 since we'll connect the display's enable bit to low (enabled all the time)

// OK to ignore UCB0SOMI/P1.7 since the display doesn't give back any data

///////////////////////////////////////////////

// Configuring the display's other pins

///////////////////////////////////////////////

// Set reset pin as output

P9DIR |= BIT4;

// Set the data/command pin as output

P2DIR |= BIT3;

// Set the chip select pin as output

P2DIR |= BIT5;

**return**;

}

**void** **HAL\_LCD\_SpiInit**(**void**)

{

//////////////////////////

// SPI configuration

//////////////////////////

// Put eUSCI in reset state while modifying the configuration

UCB0CTLW0 |= UCSWRST;

// Set clock phase to "capture on 1st edge, change on following edge"

UCB0CTLW0 |= UCCKPH;

// Set clock polarity to "inactive low"

// Set data order to "transmit MSB first"

UCB0CTLW0 |= UCMSB;

// Set MCU to "SPI master"

UCB0CTLW0 |= UCMST;

// Set SPI to "3 pin SPI" (we won't use eUSCI's chip select)

UCB0CTLW0 |= UCMODE\_0;

// Set module to synchronous mode

UCB0CTLW0 |= UCSYNC;

// Set 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 |= 0x0000;

// Exit the reset state at the end of the configuration

UCB0CTLW0 &= ~UCSWRST;

// Set CS' (chip select) bit to 0 (display always enabled)

P2OUT &= ~BIT5;

// Set DC' bit to 0 (assume data)

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;

}

/\* --COPYRIGHT--,BSD

\* Copyright (c) 2015, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\* --/COPYRIGHT--\*/

**Questions:**

1. Which SPI mode does the configuration correspond to?
   1. **3 pin SPI**

**Part 2:**

For this part, much of the code was simple. With the included header file and going to the link of the API given by the professor for examples of how it should be written, everything about this part was simple but took some time.

// Code to print to the LCD pixel display on the Educational BoosterPack

**#include** "msp430fr6989.h"

**#include** "Grlib/grlib/grlib.h" // Graphics library (grlib)

**#include** "LcdDriver/lcd\_driver.h" // LCD driver

**#include** <stdio.h>

**#include** <string.h>

**#define** redLED BIT0

**#define** greenLED BIT7

**#define** button BIT1

**extern** tImage logo4BPP\_UNCOMP;

**void** **config\_ACLK\_to\_32KHz\_crystal**();

**void** **main**(**void**)

{

**volatile** **unsigned** **int** counter = 250, temp;

**bool** flag = **false**;

**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

P1IFG &= ~button;

// 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);

// Part 2

Graphics\_setBackgroundColor(&g\_sContext, GRAPHICS\_COLOR\_WHITE);

Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_BLACK);

// Set the default font for strings

GrContextFontSet(&g\_sContext, &g\_sFontCm12b);

**while**(1) {

// Clear the screen

Graphics\_clearDisplay(&g\_sContext);

////////////////////////////////////////////////////////////////////////////////////////////

**if** (flag)

{

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);

// Part 2

GrContextFontSet(&g\_sContext, &g\_sFontFixed6x8);

**sprintf**(mystring, "Lab 11.2!");

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);

// Part 2

tRectangle rect1 = {68, 78, 77, 84};

Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_LIME);

Graphics\_drawRectangle(&g\_sContext, &rect1);

tRectangle rect2 = {81, 78, 90, 84};

Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_VIOLET);

Graphics\_fillRectangle(&g\_sContext, &rect2);

Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_MAROON);

Graphics\_drawCircle(&g\_sContext, 50, 80, 3);

Graphics\_fillCircle(&g\_sContext, 59, 80, 3);

Graphics\_drawLine(&g\_sContext, 50, 90, 90, 90);

**sprintf**(mystring, "Counter: ");

Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 54, 100, OPAQUE\_TEXT);

// Configure ACLK to the 32 KHz crystal (function call)

config\_ACLK\_to\_32KHz\_crystal();

// Configure Timer\_A

// Set timer period

TA0CCR0 = 10000;

// Timer\_A: ACLK, div by 1, up mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

tRectangle rect3 = {75, 95, 110, 110};

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**memset**(mystring,0,**strlen**(mystring));

**while**(!(TA0CTL & TAIFG)) {}

temp = counter;

**if** (temp >= 100)

{

**sprintf**(mystring, "%d ", counter);

}

**if** (temp >= 10)

{

**sprintf**(mystring, " %d ", counter);

}

**if** (temp < 10)

{

**sprintf**(mystring, " %d ", counter);

}

Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_MAROON);

Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 95, 100, OPAQUE\_TEXT);

counter++;

**if** (counter == 256)

counter = 0;

TA0CTL &= ~TAIFG; // Clear the flag

**if** ((P1IFG & button) != 0)

{

counter = 250;

**break**;

}

}

}

**else**

GrImageDraw(&g\_sContext, &logo4BPP\_UNCOMP, 0, 0);

**while**(1){

**if** ((P1IFG & button) != 0)

{

flag = !flag;

P1IFG &= ~button;

**break**;

}

}

}

}

**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

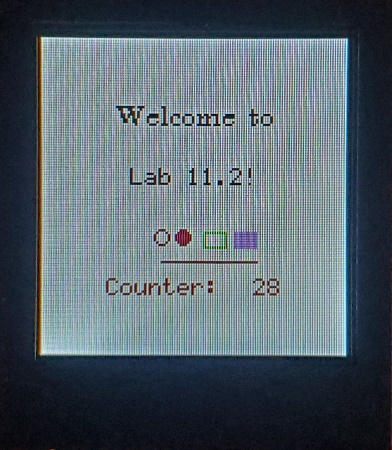
} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers

**return**;

}

**Questions:**

1. Include pictures of the two screens of your demo in your report.
   1. 
   2. 

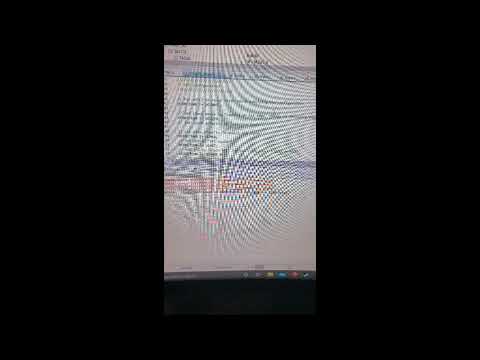
**Student Q&A:**

1. Is the SPI implemented as half-duplex or full-duplex in this experiment?
   1. **The SPI is implemented as full-duplex in this experiment**.
2. What SPI clock frequency did we set up?
   1. **The SPI clock frequency we set up was 8 MHz.**
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.
   1. **The maximum SPI clock frequency that is supported by the eUSCI module is 16 MHz.**
4. Is the display driver software specific to a display model or could it work with any display? Explain.
   1. **The display driver software is specific to a display model because of the pin configurations. Each device has a unique pin setup so to setup SPI, you would need different pins to do so.**
5. Is the graphics library (e.g. grlib) specific to a display model or could it work with any display? Explain.
   1. **The graphics library is not specific to a display model since the header files is run by customizable values to create some type of object. For instance, if the display model is 256x256, you could still use the same graphics library since it can be customizable through pixels and not through the specific display model.**

**Conclusion:**

This lab was simple after figuring out that the pins I needed were given. It’s sad to say that I spent over two hours on this because of that missing piece of knowledge. I should’ve been much more observant of all the info. Due to this, part 1 is once again the winner for being the most significant part of the lab. Without the pins, there is not a way to do the lab.

**Video Link:**

**[](https://www.youtube.com/embed/zaXi6HWazEk?feature=oembed)**