Lab 9 – Serial Peripheral Interface (SPI) & LCD Pixel Display

Youssef Samwel

yo800238@ucf.edu

EEL4742C Embedded Systems

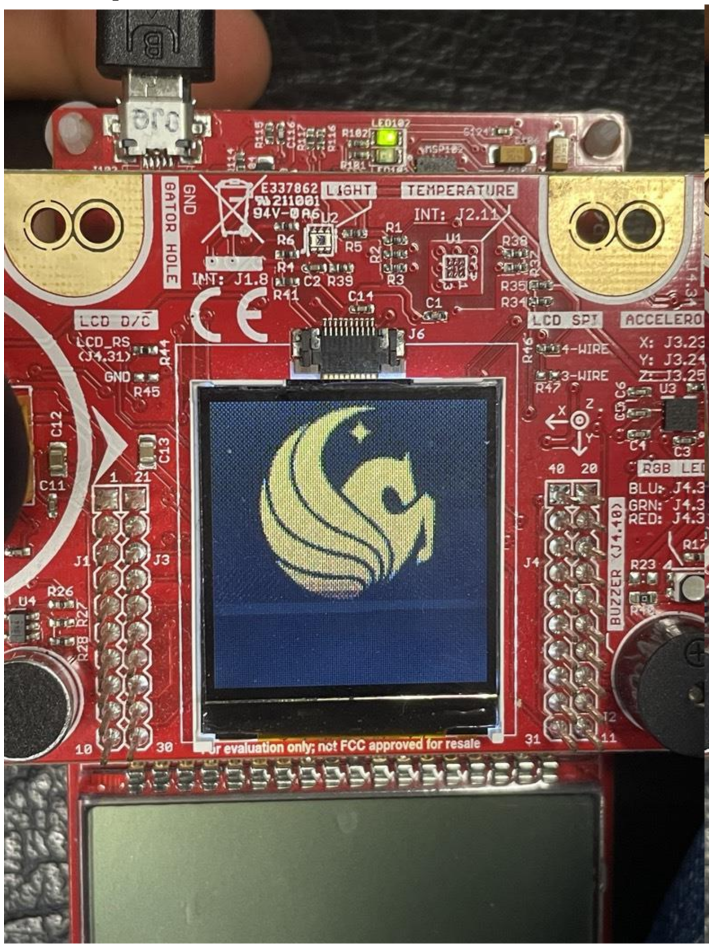
Prof. Dr. Zakhia Abichar - Section 00419

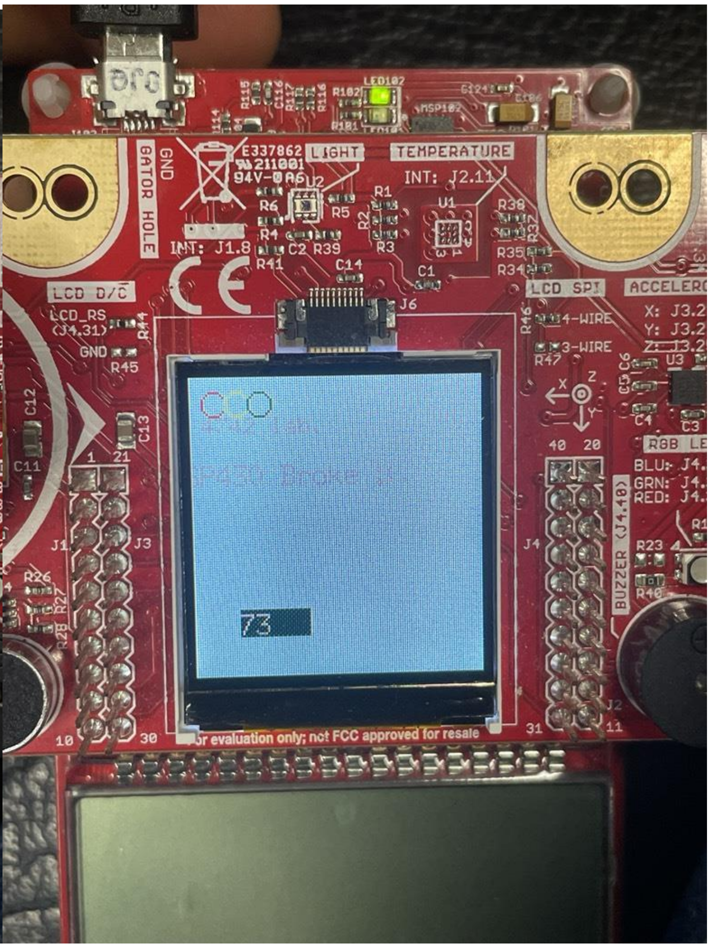
4/4/2024

1.0 Project Description

In In this lab, we will learn using the Serial Peripheral Interface (SPI) to communicate data and use it to interface the pixel display that’s on the BoosterPack board. We will also learn how the graphics software stack works and use it to draw on the pixel display.

2.0 Experiment Code





```c
#ifndef __LAB9_IMPLEMENTATIONS__
#define __LAB9_IMPLEMENTATIONS__

#include "Grlib/grlib/grlib.h"    // Graphics library (grlib)
#include "LcdDriver/lcd_driver.h" // LCD driver
#include <stdio.h>

static Graphics_Context g_sContext; // Declare a graphic library context

extern const tImage UCF_Logo;
extern const tFont g_sFontfixed7x13;

void lab_9_1() {
    char mystring[20];
    // Clear the screen
    Graphics_clearDisplay(&g_sContext);
```

```
///////////////////////////////////////////////////////////////////////////////////////
/////

    // Print message
    Graphics_drawStringCentered(&g_sContext, (int8_t*)"Welcome to", AUTO_STRING_LENGTH,
64, 30, OPAQUE_TEXT);
    sprintf(mystring, "EEL 4742C Lab!");
    Graphics_drawStringCentered(&g_sContext, (int8_t*)mystring, AUTO_STRING_LENGTH, 64,
55, OPAQUE_TEXT);
}

void lab_9_2() {
    Graphics_clearDisplay(&g_sContext);

    // draw UCF logo
    Graphics_drawImage(&g_sContext, &UCF_Logo, 0, 0);

    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BEIGE);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BISQUE);

    // Set the default font for strings
    GrContextFontSet(&g_sContext, &g_sFontFixed6x8);

    __delay_cycles(2e5);
    Graphics_clearDisplay(&g_sContext);

    char szBuf[30];
    int len = sprintf(szBuf, "EEL 4742 lab.");
    Graphics_drawStringCentered(&g_sContext, (int8_t*)szBuf, len, 20, 20, 1);
    Graphics_setFont(&g_sContext, &g_sFontfixed7x13);
    len = sprintf(szBuf, "MSP430 Broke :(");
    Graphics_drawStringCentered(&g_sContext, (int8_t*)szBuf, len, 40, 40, 1);

    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_drawCircle(&g_sContext, 10, 11, 5);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_YELLOW);
    Graphics_drawCircle(&g_sContext, 20, 11, 5);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
    Graphics_drawCircle(&g_sContext, 30, 11, 5);


    int8_t value = 0;
    while(1) {
        Graphics_Rectangle rect;
        rect.xMin = 10;
        rect.xMax = 120;
        rect.yMin = 90;
        rect.yMax = 110;
        Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BEIGE);
        Graphics_drawRectangle(&g_sContext, &rect);

               rect.xMin = 20;
               rect.xMax = 50;
               rect.yMin = 99;
               rect.yMax = 110;
               Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
               Graphics_fillRectangle(&g_sContext, &rect);

        value++;
        sprintf(szBuf, "%d", value);
        Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLANCHED_ALMOND);
        Graphics_drawString(&g_sContext, (int8_t*)szBuf, len, 20, 99, 0);
        __delay_cycles(1e5);
    }

    _low_power_mode_3();

}

#endif

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

#include <LcdDriver/lower_driver.h>
#include "msp430fr6989.h"
#include "Grlib/grlib/grlib.h"
#include <stdint.h>

void HAL_LCD_PortInit(void)
{
    /////////////////////////////////////
    // Configuring the SPI pins
    /////////////////////////////////////

    // Configure UCB0CLK/P1.4 pin to serial clock
    // Dir: X Sel1: 0 Sel0: 1 LCDSz: 0
    P1SEL1 &= ~BIT4;
    P1SEL0 |= BIT4;


    // Configure UCB0SIMO/P1.6 pin to SIMO
    // Dir: X Sel1: 0 Sel0: 1 LCDSz: 0
    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
```

```c
    ///////////////////////////////////////////////
    // 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 and set all fields in the register to 0
    UCB0CTLW0 = UCSWRST;

    // Fields that need to be nonzero are changed below

    // Set clock phase to "capture on 1st edge, change on following edge"
    UCB0CTLW0 |= UCCKPH;
    // Set clock polarity to "inactive low"
    UCB0CTLW0 &= ~UCCKPL;
    // Set data order to "transmit MSB first"
    UCB0CTLW0 |= UCMSB;
    // Set data size to 8-bit
    UCB0CTLW0 &= ~UC7BIT;
    // Set MCU to "SPI master"
    UCB0CTLW0 |= UCMST;
    // Set SPI to "3-pin SPI" (we won't use eUSCI's chip select)
    UCB0CTLW0 &= ~UCMODE_3;
    // Set module to synchronous mode
    UCB0CTLW0 |= UCSYNC;
    // Set clock to SMCLK
    // either 2 or 3 ??
    UCB0CTLW0 |= UCSSEL_3;


    // Configure the clock divider (SMCLK is from DCO at 8 MHz; run SPI at 8 MHz using
SMCLK)
    UCB0BRW = 1;

    // 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--*/

/*
EEL 4742C - UCF

Code that prints a welcome message to the pixel display.
*/

#include "msp430fr6989.h"
#include "Grlib/grlib/grlib.h"    // Graphics library (grlib)
#include "LcdDriver/lcd_driver.h" // LCD driver
#include <stdio.h>
#include <math.h>
#include "lab9.h"
#include "shader1.h"

#define WIDTH 128
#define HEIGHT 128
#define redLED BIT0
#define greenLED BIT7
#define S1 BIT1
#define S2 BIT2

extern void HAL_LCD_writeCommand(uint8_t command);
extern void HAL_LCD_writeData(uint8_t data);
extern void HAL_LCD_writeCommand(uint8_t command);

void Initialize_Clock_System();

// Function to draw a pixel at (x, y) with color
void draw_pixel(float r, float g, float b)
{
    uint16_t red = r * 0b11111;
    uint16_t green = g * 0b111111;
    uint16_t blue = b * 0b11111;
    uint16_t pixelValue = (red << 11) | (green << 5) | (blue);
    // We're in 16-bit pixel mode
    HAL_LCD_writeData(pixelValue & 0xFF);
    HAL_LCD_writeData((pixelValue >> 8) & 0xFF);
}

void run_shader1()
{
    iTime += 0.01;
    uint16_t x, y;
    for (y = 0; y < 128; y++)
    {
        for (x = 0; x < 128; x++)
        {
            vec2 uv;
            uv.x = x;
            uv.y = y;
            vec3 fragColor;
            mainImage(&fragColor, uv);
            draw_pixel(fragColor.r, fragColor.g, fragColor.b);
        }
    }
}

// ****************************************************************************
void main(void)
{
    char mystring[20];

    // Configure WDT & GPIO
    WDTCTL = WDTPW | WDTHOLD;
    PM5CTL0 &= ~LOCKLPM5;

    // Configure LEDs
    P1DIR |= redLED;
    P9DIR |= greenLED;
    P1OUT &= ~redLED;
    P9OUT &= ~greenLED;
    // Configure buttons
    P1DIR &= ~(S1 | S2);
    P1REN |= (S1 | S2);
    P1OUT |= (S1 | S2);
    P1IFG &= ~(S1 | S2); // Flags are used for latched polling

    // Set the LCD backlight to highest level
    P2DIR |= BIT6;
    P2OUT |= BIT6;

    // Configure clock system
    Initialize_Clock_System();


////////////////////////////////////////////////////////////////////////////////////////
/////
    // Graphics functions

    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);
    HAL_LCD_writeCommand(CM_DISPON);

    // lab_9_1();
    lab_9_2();

    //while (1)
    //{
     //   // configure the dispaly
     //   Crystalfontz128x128_SetDrawFrame(0, 0, 127, 127);
      //  HAL_LCD_writeCommand(CM_RAMWR);
       // run_shader1();
    //}
}

// ******************************
void Initialize_Clock_System()
{
    // DCO frequency = 8 MHz (default value)
    // MCLK = fDCO/2 = 4 MHz
    // SMCLK = fDCO/1 = 8 MHz
    CSCTL0 = CSKEY;                  // Unlock clock module config registers
    CSCTL3 &= ~(BIT2 | BIT1 | BIT0); // DIVM = 000
    CSCTL3 |= BIT0;                  // DIVM = 001 = /2
    CSCTL3 &= ~(BIT6 | BIT5 | BIT4); // DIVS = 000 = /1
    CSCTL0_H = 0;                    // Relock clock module config registers

    return;
}
```

3.0 Student Q&A

Which SPI mode does the configuration correspond to?

Mode 0

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

simplex

2. What SPI clock frequency did we set up in this lab?

8 MHz

3. What I2C clock frequency did we set up in this lab?

1 MHz

4. 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. 5. Show how you computed the I2C clock divider in the last part.

16 MHz

4.0 Conclusion

In conclusion, Lab 9 provided a hands-on exploration of SPI communication and its application in interfacing with a pixel display. By configuring SPI in Mode 0 with an 8 MHz clock frequency, we successfully demonstrated drawing capabilities on the display. Operating in simplex mode, we efficiently transmitted data and computed the I2C clock divider for optimal performance. This lab enhanced our understanding of SPI fundamentals and equipped us with valuable skills in hardware interfacing and communication protocols.