**// Lab 11: Serial Peripheral Interface (SPI) & LCD Pixel Display**

* Using Serial Peripheral Interface (SPI) for communicating data!
* Software stack 🡪 Enabling printing to the LCD pixel display!
* LCD Pixel Display (on the Educational BoosterPack)

**11.1: Serial Peripheral Interface (SPI)**

* SPI Interface 🡪 A simple communication protocol 🡪 A master communicates with one or multiple devices!
* Two data wires between the master and the device! 🡪 Full-duplex protocol!
* Master 🡪 Generating clock signal and transmission of it to the device!
* Synchronous protocol!
* Fourth Wire of SPI 🡪 The chip select signal!
* Master 🡪 Activating the signal 🡪 Enabling the device to communicate!
* The device 🡪 Placing high-impedance on its data out line!
* Totally 🡪 The SPI interface 🡪 Using four wires: (A) Serial Data Out; (B) Serial Data In; (C) Serial Clock; (D) Chip Select!
* SPI 🡪 Not officially a standard option!
* Transmitted Data by Master (Serial Out; Serial Data Out; MOSI; SIMO)!
* 3-Pin SPI 🡪 Implementation of SPI with 2 Shift Registers 🡪 One at the master and the other at the device!
* Communication 🡪 Based on exchanging the contents of the two shift registers!
* Bit out of the master’s shift register goes in the device’s shift register!
* 8 Times shifted registers 🡪 The contents of two shift registers are exchanged!
* Typically latched MSB bit into a D flip-flop!
* Transmitting a bit consisting of latch/shift actions!
* Repeating these actions 9 times 🡪 For exchanging the contents of two registers!
* Four modes of operations 🡪 Defined for SPI based on how the clock signal is used!
* Four Cases 🡪 (0/0), (0/1), (1/0), and (1/1)!
* Polarity 0: Clock idle at low
* Polarity 1: Clock idle at high
* Phase 0: Latch at trailing edge, communicate at leading edge
* Phase 1: Latch at leading edge, communicate at trailing edge
* 128x128 pixel resolution and supports 262K colors
* It incorporates a built-in controller, the Sitronix ST7735S and an SPI interface

unmodified.

Code 11.1

#include "msp430fr6989.h"

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

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

#include <stdio.h>

// Function that sets the pins of the LCD Display to SPI functionality

void HAL\_LCD\_PortInit(void)

{

// Diverts UCB0CLK/P1.4 pin to SCLK (SPI Serial Clock)

P1SEL1 &= ~BIT4;

P1SEL0 |= BIT4;

// Diverts UCB0SIMO/P1.6 pin to Slave In Master Out (SPI SIMO)

P1SEL1 &= ~BIT6;

P1SEL0 |= BIT6;

// We ignore UCB0STE/P1.5 since Display's enable bit is set to low so that it can work at all times.

// We ignore UCB0SOMI/P1.7 since Display never gives back any data.

// Reset Pin as output

P9DIR |= BIT4;

// Data/Command pin as output

P2DIR |= BIT3;

// Chip Select pin as output

P2DIR |= BIT5;

return;

}

// Function that configures eUSCI module to SPI

void HAL\_LCD\_SpiInit(void)

{

// eUSCI module is set to reset state while modifying the configuration

UCB0CTLW0 |= UCSWRST;

// Clock Phase 0 of SPI

UCB0CTLW0 |= UCCKPH;

// Clock Polarity 0 of SPI

UCB0CTLW0 &= ~UCCKPL;

// Data transmitted by MSB first

UCB0CTLW0 |= UCMSB;

// Sets MCU to be the SPI Master

UCB0CTLW0 |= UCMST;

// Sets SPI configuration to 3 Pin SPI

UCB0CTLW0 |= UCMODE\_0;

// Sets the mode to synchronous mode

UCB0CTLW0 |= UCSYNC;

// Sets clock to SMCLK

UCB0CTLW0 |= UCSSEL\_2;

// Sets Clock Divider to 1 since we want an SPI clock frequency of 8MHz

UCB0BRW |= 0x01;

// Exits the reset state of eUSCI after configuring for SPI

UCB0CTLW0 &= ~UCSWRST;

// CS bit to 0 to always display

P2OUT &= ~BIT5;

// DC bit to 0 to always assume data

P2OUT &= ~BIT4;

return;

}

**11.2: Using the Graphics Library**

* Using TI’s Graphics Library (grlib) for drawing shapes and text on the display!
* Graphics Software Stack 🡪 Consisting of multiple layers!
* Lowest Layer 🡪 The display driver that performs initializations 🡪 Driving a pulse!
* Next Layer (inside the software stack) 🡪 The graphics library!
* Using the services of the layer!
* Library 🡪 Working with any display once the driver is setup!
* Library 🡪 Higher level of functionalities!
* The library grlib 🡪 A small library and is easy to use

Code 11.2

#include "msp430fr6989.h"

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

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

#include <stdio.h>

#define redLED BIT0 // Red LED location is at P1.0

#define BUT1 BIT1 // Push Button location is at P1.1

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

void config\_ACLK\_to\_32KHz\_crystal() {

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

do {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

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

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

return;

}

// Global Variables used for the demo

unsigned int status = 0, n = 0;

Graphics\_Context g\_sContext;

tRectangle myRectangle1 = {45, 95, 79, 105};

tRectangle myRectangle2 = {85, 95, 95, 105};

tRectangle myRectangle3 = {46, 96, 46, 104};

char mystring[20];

void main(void)

{

volatile unsigned int counter=0;

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode

// Red LED configuration as output and starts as low

P1DIR |= redLED;

P1OUT &= ~redLED;

// Max Brightness on the LCD Display along with J5 jumper select

P2DIR |= BIT6;

P2OUT |= BIT6;

// Configuring buttons with interrupt to change between the screen fast

P1DIR &= ~(BUT1|BUT2);// Sets the push button S1 as input

P1REN |= (BUT1|BUT2); // Enables the internal resistor found in P1.1

P1OUT |= (BUT1|BUT2); // Sets the resistor as a pull-up

P1IE |= (BUT1|BUT2); // Enables the interrupt enable bit of push button S1

P1IES |= (BUT1|BUT2); // Configures the interrupt raise event as falling edge

P1IFG &= ~(BUT1|BUT2);// Clears the interrupt flag of push button S1

// ACLK rerouting to the 32KHz crystal for an accurate timer.

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

// Timer A module 0 that pertains to the 8-bit counter; generates a delay of 1 second

// Configuring Timer\_A0 by setting ACLK, frequency division by 1, continuous mode, and clear TAR.

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

TA0CCR0 = (32768-1);

// This ensures that the Timer\_A Interrupt Flag is set to zero.

TA0CTL &= ~TAIFG;

// Timer A module 1 for the button debouncer

TA1CTL = TASSEL\_1 | ID\_0 | MC\_2 | TACLR;

TA1CCTL1 &= ~CCIE; // Disables the interrupt enable bit of Timer A module 0 channel 1

TA1CCTL1 &= ~CCIFG; // Clears the interrupt flag of Timer A module 0 channel 1

// 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

Crystalfontz128x128\_Init(); // Initialize the display 128x128

// Set the screen orientation

Crystalfontz128x128\_SetOrientation(0);

// Initializes the context to the pertaining display

Graphics\_initContext(&g\_sContext, &g\_sCrystalfontz128x128);

// Set background and foreground colors to Purple and SteelBlue

Graphics\_setBackgroundColor(&g\_sContext, ClrPurple);

Graphics\_setForegroundColor(&g\_sContext, ClrSteelBlue);

// Set the default font for strings to Font Fixed 6x8

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

// Clears the screen display

Graphics\_clearDisplay(&g\_sContext);

// Displays the first screen of the demo to be an image of UCF

Graphics\_drawImage(&g\_sContext, &logo4BPP\_UNCOMP, 0, 0);

// Starts low power mode 0 since we only use the SMCLK and ACLK

\_low\_power\_mode\_0();

}

// Push Button raised an interrupt event to switch screens

#pragma vector = PORT1\_VECTOR

\_\_interrupt void PORT1\_ISR() {

if((P1IN & BUT1) == 0){

TA2CTL &= ~TAIE;

TA1CCTL1 |= CCIE; // Enable the interrupt enable bit of Timer A module 0 channel 1

TA1CCTL1 &= ~CCIFG; // Clears the interrupt flag of Timer A module 0 channel 1

TA1CCR1 = TA1R + 655; // Schedule the next interrupt event to 20ms

P1IFG &= ~BUT1; // Clears the interrupt flag of the push button S1

P1IE &= ~BUT1; // Disables the interrupt enable bit of push button S1

}

}

// Timer raised an interrupt event to ensure that the debouncing is gone

#pragma vector = TIMER1\_A1\_VECTOR

\_\_interrupt void T1A1\_ISR() {

// Checks to see if the button is still pushed

if((P1IN & BUT1) == 0){

P1OUT ^= redLED; // Toggles Red LED

status ^= BIT0; // Toggles the status

if(status == 0){

TA0CTL &= ~TAIE; // shuts off the timer used for the 8-bit counter

myRectangle3.xMax = 46; // Initial position for the loading bar animation

n = 0;

// Set background and foreground colors to Purple and Steelblue

Graphics\_setBackgroundColor(&g\_sContext, ClrPurple);

Graphics\_setForegroundColor(&g\_sContext, ClrSteelBlue);

// Clear the screen

Graphics\_clearDisplay(&g\_sContext);

// First Screen Displays Image

Graphics\_drawImage(&g\_sContext, &logo4BPP\_UNCOMP, 0, 0);

}

else if(status == 1){

// Set background and foreground colors

Graphics\_setBackgroundColor(&g\_sContext, ClrPurple);

Graphics\_setForegroundColor(&g\_sContext, ClrSteelBlue);

// Clear the screen

Graphics\_clearDisplay(&g\_sContext);

// Second Screen Display

Graphics\_drawStringCentered(&g\_sContext, "Where's mah", AUTO\_STRING\_LENGTH, 64, 30, OPAQUE\_TEXT);

// Set the 2nd font for strings

GrContextFontSet(&g\_sContext, &g\_sFontCm12);

sprintf(mystring, "snake?");

Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 55, OPAQUE\_TEXT);

// Draws a pink circle

Graphics\_setForegroundColor(&g\_sContext, ClrPink);

Graphics\_drawCircle(&g\_sContext, 64, 64, 60);

// Draws a yellow filled circle

Graphics\_setForegroundColor(&g\_sContext, ClrYellow);

Graphics\_fillCircle(&g\_sContext, 35, 100, 5);

//Draws an orange rectangle

Graphics\_setForegroundColor(&g\_sContext, ClrOrange);

Graphics\_drawRectangle(&g\_sContext, &myRectangle1);

// Draws a green square

Graphics\_setForegroundColor(&g\_sContext, ClrDarkOliveGreen);

Graphics\_fillRectangle(&g\_sContext, &myRectangle2);

// Draws a cyan line

Graphics\_setForegroundColor(&g\_sContext, ClrCyan);

Graphics\_drawLineH(&g\_sContext, 35, 90, 90);

TA0CTL |= TACLR | TAIE; // enables the 8-bit counter

}

P1IE |= BUT1; // Enables the interrupt enable bit of push button S1

TA1CCTL1 &= ~CCIE; // Disables the interrupt enable bit of Timer A module 0 channel 1

TA1CCTL1 &= ~CCIFG; // Clears the interrupt flag of Timer A module 0 channel 1

}

}

// Timer raised an interrupt event to enable the animation and timer of the second screen

#pragma vector = TIMER0\_A1\_VECTOR

\_\_interrupt void T0A1\_ISR() {

if((TA0CTL & TAIFG)== TAIFG ){

if(status == 1){

GrContextFontSet(&g\_sContext, &g\_sFontFixed6x8); //Changes the font back to default

sprintf(mystring,"%d",n); //Stores the initial variable

Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 84, OPAQUE\_TEXT); // Prints the current number in the 8-bit counter

Graphics\_fillRectangle(&g\_sContext, &myRectangle3); // Prints the current position of the loading animation screen

n++; // Advances the 8-bit counter

myRectangle3.xMax += 1; //Advances the loading screen animation

if(n == 256){ //When the counter reaches the maximum then we start again

n=0;

sprintf(mystring," ");

Graphics\_drawStringCentered(&g\_sContext, mystring, AUTO\_STRING\_LENGTH, 64, 84, OPAQUE\_TEXT);

}

if(myRectangle3.xMax == 78){ // When the loading screen reaches its end, we restart again

Graphics\_setForegroundColor(&g\_sContext, ClrPurple); // Sets the next object to purple

Graphics\_fillRectangle(&g\_sContext, &myRectangle3); // Clears the rectangle by using the same color as the background

myRectangle3.xMax = 46; // Resets the loading animation back to its original position

Graphics\_setForegroundColor(&g\_sContext, ClrCyan); // Sets the loading animation back to cyan

}

TA0CTL &= ~TAIFG; // Clears the interrupt flag.

}

}

}