EEL 4742C: Embedded Systems
Name: Hamzah Ullah
Lab 11: Advanced Timer Features

Introduction:

This part of the lab introduces us to the LCD screen and the SPI protocol for communications. We communicate with the display through SPI.

Part 1: Serial Peripheral Interface(SPI)

This part of the lab deals with just porting the SPI with a predetermined output on the display. We check the data sheet for the correct pins to port the SPI then use the family users guide to direct the registers in the correct configuration. Below is the code showing the configuration for the display to output that was prewritten for us. It does this by putting a background color and a for-ground color for the text. And outputs it within the context of the screen using the string library from the graphics library.

Functions for configuring the LCD port and SPI

```
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"
    UCB0CTLW0 &= ~UCCKPL;
    // 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_2;

    // Set clock divider to 1 (SMCLK is from DCO at 8 MHz; we'll run SPI at 8 MHz)
    UCB0BRW &= ~(BIT4|BIT5|BIT6);

    // 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;
}
```

Part 2: Using the Graphics Library

This part of the lab uses the graphics library within the files given. Since the graphics library is shared with all drivers, the code is easy to use amongst multiple displays. There are several components that go into displaying the image. The first part is the image information. The UCF logo has several addresses that correspond to each pixel of the screen and designates a color for each one. To draw shapes there are several predetermined shapes given to us like circles and squares. We simply call the value and set parameters for the vertices for squares and rectangles or radius for circles. To implement a counter, we simply count up the value of n as we display it. Finally we implement a push button by setting parameters for the button, and clearing the context and drawing the image on the screen. In my implementation, I put 2 context on 1 screen to demonstrate how the context can have different color shapes by changing the foreground. Below is the code.

Below has relevant code

```
//code to increment
while(n<257){
    sprintf(mystring, "%d", n++);
    Graphics_drawStringCentered(&g_sContext, mystring, AUTO_STRING_LENGTH, 64, 80,
OPAQUE_TEXT);
    if(n==256)
    {
        n=0;
        Graphics_clearDisplay(&g_sContext);
    }
// Set background and foreground colors
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
//UCF LOGO
        Graphics_drawImage(&g_sContext, &logo4BPP_UNCOMP, 0, 0);
//Draw Circle
        Graphics_drawCircle(&g_sContext, 40, 40, 10);
//Code for Switching Screens + Multicontext and colors
```

```
    Graphics_Context g_sContext;        // Declare a graphic library context
    Graphics_Context g_sContext2;       // Declare a graphic library context

// Set background and foreground colors
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_YELLOW);
    Graphics_setBackgroundColor(&g_sContext2, GRAPHICS_COLOR_BLACK);
       Graphics_setForegroundColor(&g_sContext2, GRAPHICS_COLOR_RED);

for(;;){
        while((TA0CTL & TAIFG) != TAIFG) {}
        TA0CTL &= ~TAIFG; // Clear the flag

    if ((P1IN & button) == 0) {
    // Toggle the screens
    if(status == 0)
    {
        Graphics_clearDisplay(&g_sContext);
        Graphics_clearDisplay(&g_sContext2);
       //UCF LOGO
        Graphics_drawImage(&g_sContext, &logo4BPP_UNCOMP, 0, 0);
        status = 1;
        }
        else if(status == 1)
        {
            Graphics_clearDisplay(&g_sContext);
            Graphics_clearDisplay(&g_sContext2);
            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_drawCircle(&g_sContext2, 10,10, 10);
            Graphics_drawStringCentered(&g_sContext, mystring, AUTO_STRING_LENGTH,
64, 80, OPAQUE_TEXT);
            status = 0;
        }
    }
```

QUESTIONS:

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

2. **What SPI clock frequency did we set up?**
   8Mhz

3. **What is the maximum SPI clock frequency that is supported by the eUSCI module? Look in the microcontroller's datasheet in table 5-18.**

16Mhz

4. **Is the display driver software specific to a display module or could it work with any display? Explain.**
   Display driver is software specific because each display has a different set of data that carries the display information which includes color, resolution and features. If you were to use the same driver, the display would not work because the data required for that display would be too large or too small.

5. **Is the graphics library (e.g. grlib) specific to a display model or could it work with any display? Explain.**
   The graphics library is not specific, it can be used with any display driver because they are using the same set of functions to draw onto the display. The display driver already configured the display information to be read by the graphics library to be interfaced with the screen.