

DELIVERABLES

1. Code for the functions:

- **BSP_LCD_Message**

```
void BSP_LCD_Message (int device, int line, int col, char *string, unsigned int value){
    if (device == 0) {
        if (line > 11 || col > 20) return;

        // Setting cursor and draw the string
        BSP_LCD_SetCursor(col, line);
        BSP_LCD_DrawString(col, line, string, ST7735_WHITE);

        // Moving the cursor to where the value should appear
        BSP_LCD_SetCursor(col + strlen(string), line);

        // Outputting the value using BSP_LCD_OutUDec
        BSP_LCD_OutUDec(value, ST7735_WHITE);
    } else if (device == 1) {
        if (line != 0 || col > 20) return;

        // Setting cursor and draw the string
        BSP_LCD_SetCursor(col, 12);
        BSP_LCD_DrawString(col, 12, string, ST7735_WHITE);

        // Moving the cursor to where the value should appear
        BSP_LCD_SetCursor(col + strlen(string), 12);

        // Outputting the value using BSP_LCD_OutUDec
        BSP_LCD_OutUDec4(value, ST7735_WHITE);
    }
}
```

- **BSP_LCD_DrawCrosshair**

```
void BSP_LCD_DrawCrosshair(int16_t x, int16_t y, int16_t color) {
    // The size of the crosshair (half the length of each line)
    int16_t crossSize = 5;

    // Adjusting coordinates to ensure the entire crosshair is within bounds
    if (x - crossSize < 0+4) {
        x = crossSize; // Ensure the left part of the crosshair is within bounds
    } else if (x + crossSize >= ST7735_TFTWIDTH) {
        x = ST7735_TFTWIDTH - crossSize - 1; // Ensuring the right part of the crosshair is within bounds
    }

    if (y - crossSize < 0+4) {
        y = crossSize; // Ensure the top part of the crosshair is within bounds
    } else if (y + crossSize >= ST7735_TFTHEIGHT) {
        y = ST7735_TFTHEIGHT - crossSize - 1; // Ensuring the bottom part of the crosshair is within bounds
    }

    // Calculating the bounds for the horizontal and vertical lines of the crosshair
    int16_t leftX = x - crossSize; // Start of the horizontal line
    int16_t rightX = x + crossSize; // End of the horizontal line
    int16_t topY = y - crossSize; // Start of the vertical line
    int16_t bottomY = y + crossSize; // End of the vertical line

    // Drawing the horizontal line of the crosshair
    BSP_LCD_DrawFastHLine(leftX, y, rightX - leftX + 1, color);

    // Drawing the vertical line of the crosshair
    BSP_LCD_DrawFastVLine(x, topY, bottomY - topY + 1, color);
}
```

- InitTimer1A

```
// Configure Timer1A
void InitTimer1A(unsigned long period, unsigned long priority)
{
    long sr;
    volatile unsigned long delay;

    sr = StartCritical();
    SYSCTL_RCGCTIMER_R |= 0x02;
    while((SYSCTL_RCGCTIMER_R & 0x02) == 0){} // allow time for

    TIMER1_CTL_R &= ~TIMER_CTL_TAEN;
    TIMER1_CFG_R = TIMER_CFG_32_BIT_TIMER;
    TIMER1_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
    TIMER1_TAILR_R = period - 1;
    TIMER1_ICR_R = TIMER_ICR_TATOCINT;
    TIMER1_IMR_R |= TIMER_IMR_TATOIM;
    NVIC_PRI5_R = (NVIC_PRI5_R & 0xFFFF00FF) | (priority << 13);
    NVIC_EN0_R = NVIC_EN0_INT21;
    TIMER1_TAPR_R = 0;
    TIMER1_CTL_R |= TIMER_CTL_TAEN;
    EndCritical(sr);
}
```

- OS_AddPeriodicThread

```
int OS_AddPeriodicThread(void(*task)(void), unsigned long period, unsigned long priority) {
    if (PeriodicTask != 0) { // Checking if a periodic task is already running
        return 0; // Return 0 if the thread cannot be added
    }
    PeriodicTask = task; // Assign the task to the global function pointer
    InitTimer1A(period, priority); // Initializing Timer1A with the specified period and priority
    return 1;
}
```

- Producer

```
***** Producer *****
void Producer(void) {
    #if TEST_TIMER
        PE1 ^= 0x02; // heartbeat
        Count++; // Increment dummy variable
    #else
        // Variable to hold updated x and y values
        int16_t newX = x;
        int16_t newY = y;
        int16_t deltaX = 0;
        int16_t deltaY = 0;

        uint16_t rawX, rawY; // To hold raw adc values
        uint8_t select; // To hold pushbutton status
        rxDataType data;
        BSP_Joystick_Input(&rawX, &rawY, &select);

        // Your Code Here

        int16_t crosshairAreaHeight = 10;

        // Calculating deltas based on raw ADC values and origin
        deltaX = ((int16_t)rawX - (int16_t)origin[0]) / 512;
        deltaY = -((int16_t)rawY - (int16_t)origin[1]) / 512; // Negated deltaY to fix inverted Y-axis

        // Updating crosshair position based on deltas
        newX += deltaX;
        newY += deltaY;

        // Defining the size of the crosshair (half the length of each line)
        int16_t crossSize = 5;

        // Clamping crosshair position to ensure it stays within valid range [0, 127]
        if (newX < crossSize) newX = crossSize;
        if (newX > 127 - crossSize) newX = 127 - crossSize;
        if (newY < crossSize) newY = crossSize;
        if (newY > 127 - crossSize - crosshairAreaHeight) newY = 127 - crossSize - crosshairAreaHeight;

        // Updating global crosshair position
        x = (uint32_t)newX;
        y = (uint32_t)newY;

        // Preparing data for FIFO
        data.x = x;
        data.y = y;

        // Pushing data into the FIFO
        RxFifo_Put(data);
    #endif
}
```

- Consumer

```

//***** Consumer *****
void Consumer(void) {
    rxDataType data;

    // Checking if there's new data in the FIFO
    if (RxFifo_Get(&data)) {

        // Erasing the previous crosshair
        BSP_LCD_DrawCrosshair(prevx, prevy, BG_COLOR);

        // Drawing the new crosshair
        BSP_LCD_DrawCrosshair(data.x, data.y, LCD_RED);

        // Displaying the X and Y positions
        BSP_LCD_Message(1, 0, 4, "X:", data.x);
        BSP_LCD_Message(1, 0, 12, "Y:", data.y);

        // Updating the previous position for the next iteration
        prevx = data.x;
        prevy = data.y;
    }
}

```

2. calculations for TEST_PERIOD to get a frequency of 20 Hz, and the snapshot of the logic analyzer or oscilloscope measuring timer frequency at 20 Hz:

Period = 1 / Frequency

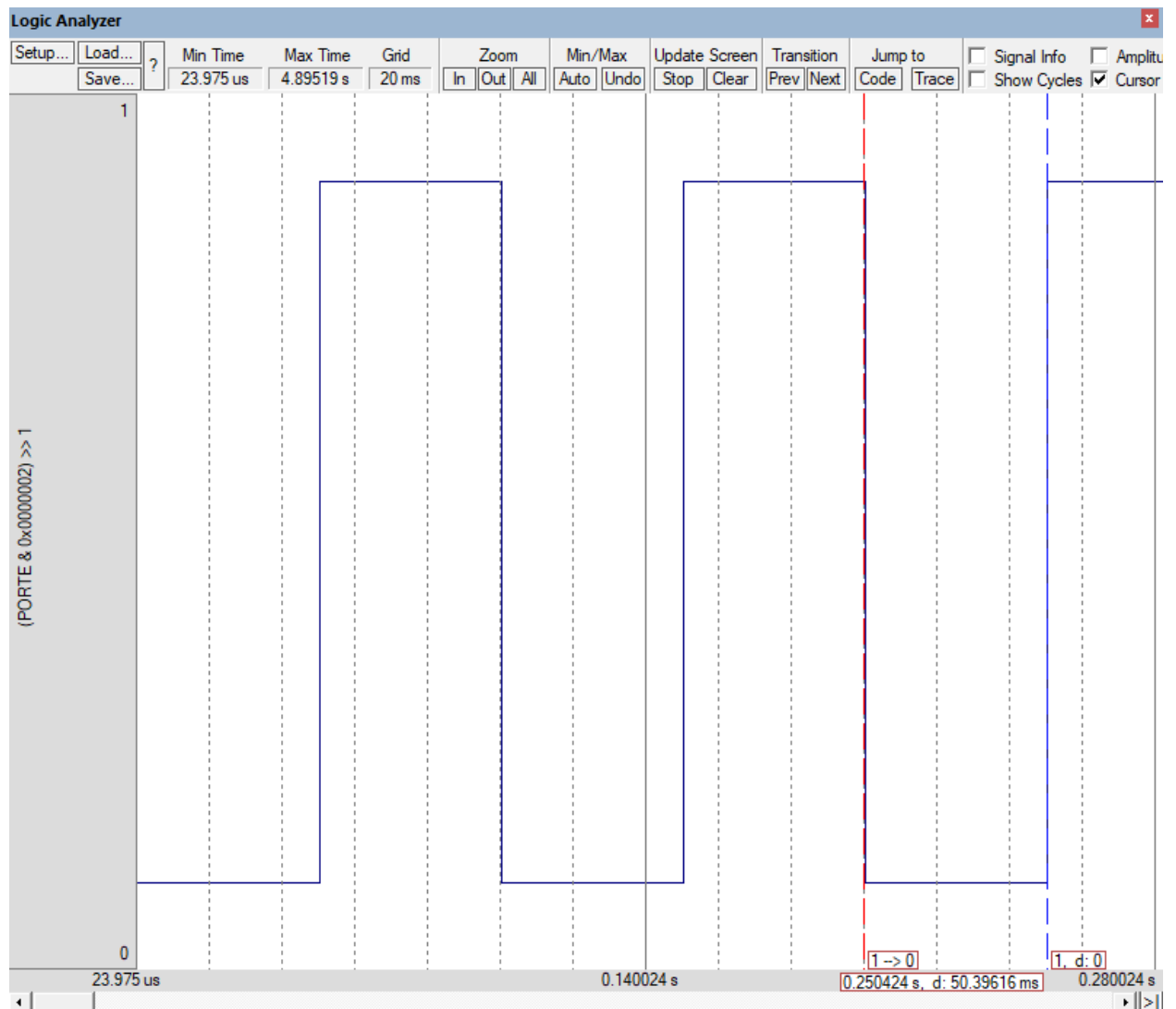
= 1 / 20 = 0.05 seconds (50 ms)

TEST_PERIOD = Period * Clock Frequency

= 0.05 * 80 MHz

= 4000000

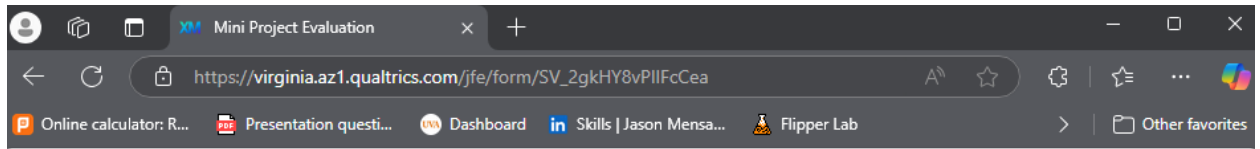
As indicated in the screenshot below:



3. Link to video recording demonstrating the required functionality:
<https://youtube.com/shorts/k4OC2mR9-ul?si=Kmc3wmwtNkPO5yBs>

APPENDIX:

- Class participation: Proof of completed survey



Thank you for your feedback on **Mini Project 1**.