ECE 4501/6501: Advanced Embedded Systems

Mini Project 1
Embedded I/O: Graphics LCD and Periodic Timer Interrupts
Due Date: Wednesday, Sep. 26, 2018, 11:59 PM

In this mini project, you will take the first step towards developing a handheld video game on the TM4C123G LaunchPad and the Educational BoosterPack MKII by implementing a simple program for displaying a crosshair on the BoosterPack LCD module and controlling it using the joystick.

Go to the following link to accept the Mini Project 1 assignment in the **GitHub Classroom**: https://classroom.github.com/a/phBka_iv. You will get access to a private repository created for you in the @UVA-embedded-systems organization on GitHub which contains the starter files for the mini project. You can use this repository for developing and testing your code. Your submission will also be to this repository.

Figure 1 shows the data flow graph for Mini Project 1. You will implement a producer function that runs periodically as a background thread to read raw ADC samples from the joystick device and calculate the next position of the crosshair on the LCD. The position data is passed via a FIFO to a foreground consumer thread that refreshes the LCD and displays the crosshair and its current position value.

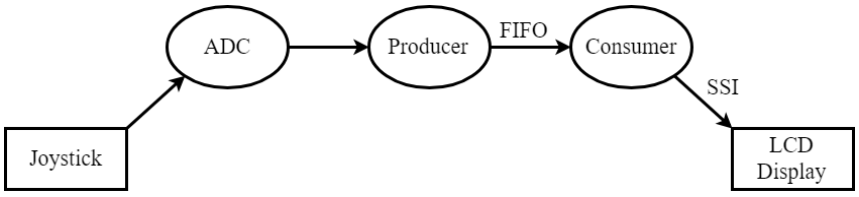

Figure 1: Data Flow Graph for Mini Project 1

The starter files include skeleton code composed of a main function that runs the foreground consumer thread, the drivers for LCD, joystick, ADC, and a periodic timer. Study the data structures and functions provided to you. You are expected to complete the implementation for LCD and timer drivers and develop the producer and consumer functions. Figure 2 shows the overall flowchart for the producer and consumer.

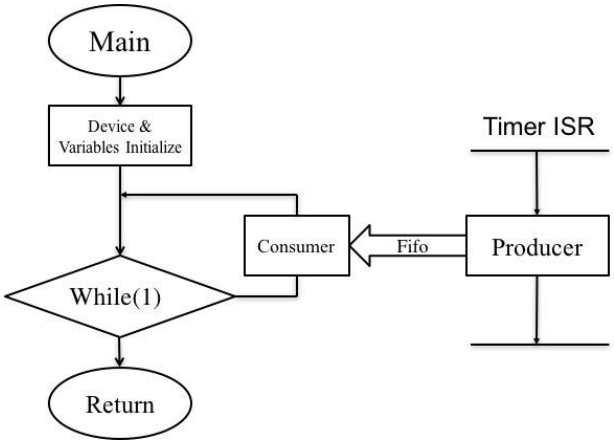

Figure 2: Overall Flowchart for Mini Project 1

Background

LCD: The Crystalfontz CFAF128128B-0145T color 128×128-pixel TFT LCD supports display updates up to 20 frames per second (FPS). This module has a color depth of 262K colors and a contrast ratio of 350. In this mini project, we use the SSI interface on TM4C123G to control the TFT LCD on the Educational BoosterPack MKII. Software drivers for the LCD and other helper functions are provided in the file **`LCD.c`**.

Note: You will need to remove resistor R10 from your TM4C123G LaunchPad, as it is connected to PB7.

Table 1. Color LCD Pinout

BoosterPack Plug-in Module Header Connection	TM4C123GXL Port Pin	Pin Function
J1.7	PB4	LCD SPI clock
J2.13	PA4	LCD SPI chip select
J2.15	PB7	LCD SPI MOSI
J4.31	PF4	LCD reset pin
J4.39(1)	PF3	LCD backlight

(1) Pin is multiplexed with the RGB LED red channel pin through the jumper head J5.

Joystick Input and ADC Module: The ITEAD studio IM130330001 2-axis joystick with pushbutton is simply two potentiometers, one for each axis. The select button is actuated when the joystick is pressed down. The **`BSP_JoyStickInput`** function in **`joystick.c`** reads the voltage present on the joystick axis to provide the position of the joystick to the application (moving the joystick to the left reads $X \approx 0$, moving the joystick to the bottom reads $Y \approx 0$).

Table 2. Joystick Pinout

BoosterPack Plug-in Module Header Connection	TM4C123GXL Port Pin	Pin Function
J1.2	PB5/AIN11	Horizontal X-axis
J1.5	PE4	Select button
J3.26	PD3/AIN4	Vertical Y-axis

In this mini project, the 12-bit ADC module on the TM4C123G is used for sampling the analog inputs from the joystick. Since you are going to sample two analog signals at same time, sample sequencer 1 (SS1) inside the ADC module is utilized. In this case, we do not use any ADC interrupts to read the transformed digital values but read them by a software method. The functions to initialize the joystick and ADC modules are given in the starter file **`joystick.c`**.

Synchronization using FIFO: In this mini project, we use a FIFO data structure for communication between the producer and consumer threads. Feel free to use the functions provided in **`FIFO.h`** and **`FIFO.c`** in your consumer and producer functions.

Mini Project Parts

Part 1) Extend the LCD driver library for displaying messages (i.e., text and associated numeric value) and drawing a crosshair.

A – Design an extended version of the device driver for the LCD so that there are two logically separate displays, one display using the top portion of LCD (rows 0 to 11) and one display for the bottom bar (row 12). The new function will have the following prototype as shown in **`LCD.h`**:

```
void BSP_LCD_Message(int device, int line, int col, char *string,
int32_t value)
```

where **`device`** specifies top or bottom logical screen (0 or 1), **`line`** specifies the line number (0 to 11 when the top logical screen is used (**`device`** = 0) and the 0 when the bottom logical screen is used (**`device`** = 1), **`col`** specifies the column number (0 to 20), **`string`** is a pointer null terminated ASCII string, and **`value`** is a number to display. The function prints the string followed by the numeric value.

(**Hint:** You may use function **`BSP_LCD_DrawString`** in **`LCD.c`** that divides the screen into lines and columns).

B – Write a function to draw a crosshair on the LCD screen with the following header:

```
void BSP_LCD_DrawCrosshair(int16_t x, int16_t y, int16_t color)
```

where **`x`** and **`y`** indicate the cartesian coordinates (0 to 127) of the crosshair and **`color`** indicates the color of the crosshair. You may call **`BSP_LCD_DrawFastVLine`** and **`BSP_LCD_DrawFastHLine`** functions to draw the crosshair.

You may add other functions as you wish. For this task, add prototypes for your public functions to the **`LCD.h`** header file. All your public functions in this part must begin with **`BSP_LCD_`**. Your implementation will be written and debugged in the file **`LCD.c`**.

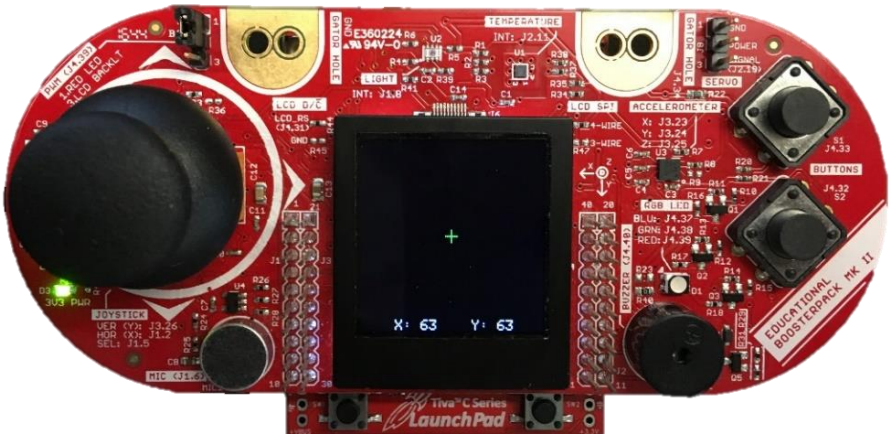

Figure 3: LCD Messaging and Crosshair Drawing Functionality

Deliverables for Part 1: Your code.

◆

Part 2) Design, implement, and test a system timer driver using 32-bit periodic timer interrupts. In this mini project, we will use the general-purpose timer GPTM TimerA.

A – Complete the code for timer initialization function is **`os.c`**:

```
void InitTimer1A(unsigned long period, unsigned long priority)
```

This function configures the timer to create a periodic interrupt that is handled by the corresponding interrupt service routine **`Timer1A_Handler`**. The **`period`** parameter determines the period of the timer. The **`priority`** parameter determines the priority of the interrupt in the NVIC module. The lines you need to complete are commented. Uncomment and complete them.

(**Hint:** Remember that a timer is essentially a counter. It counts the number of ticks in a clock signal (80 MHz in this case). Refer to the datasheet for a description of the timer registers).

B – Implement the timer driver function **`OS_AddPeriodicThread`** in os.c**:**

```
int OS_AddPeriodicThread(void(*task)(void), unsigned long period,
unsigned long priority)
```

This function enables executing a software task at a periodic rate. The **`task`** parameter is a pointer to the function to execute every **`period`**, and **`priority`** is the value to be specified in the NVIC for the timer interrupt. Figure 4 shows the flowchart for this function. In the main function, you should pass a function pointer (pointer to **`task`**) into the driver during initialization and specify the **`period`** and the **`priority`**.

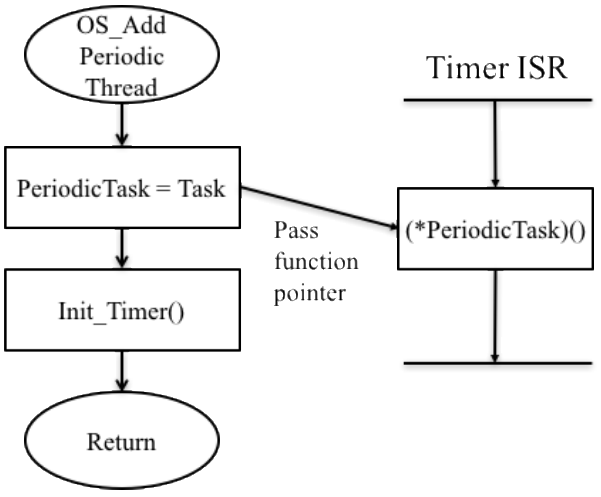

Figure 4: Flowchart for OS_AddPeriodicThread Function

Note: This function is only called once in Mini Project 1. To run multiple background threads, we need additional timers. They will be added in future mini projects.

C – Test your periodic timer by enabling the **`TEST_TIMER`** macro provided in the **`Main.c`**. This macro will enable the **`Producer`** function to generate a heartbeat signal on pin PE1. The **`TEST_PERIOD`** macro specifies the frequency of the **`Producer`**. You can watch the behavior of the heartbeat signal and measure its frequency in simulation in the Keil Logic Analyzer tool. Refer to the following tutorials on how to use the Logic Analyzer:
http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_logicanalyzer.htm
https://www.youtube.com/watch?v=mtsOrzoECbk

Figure 5 shows an example heartbeat signal on the Keil Logic Analyzer tool. Alternatively, you could test the timer by observing the voltage on PE1 on an oscilloscope. Figure 6 shows the same heartbeat signal from the actual board.

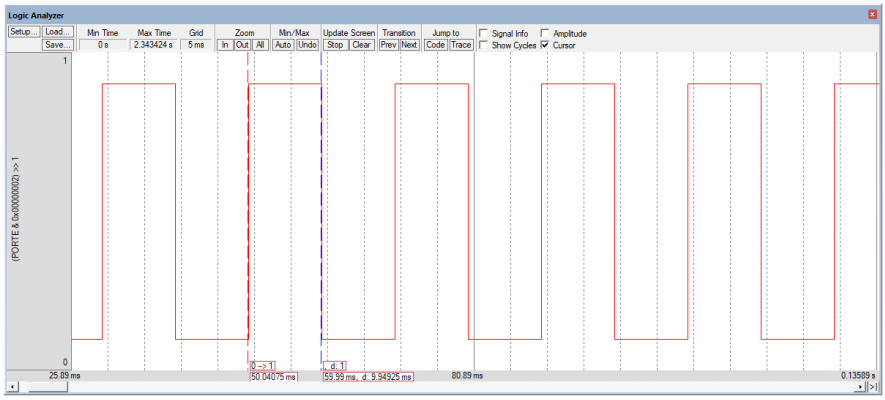

Figure 5: Heartbeat on PE1 in Simulation

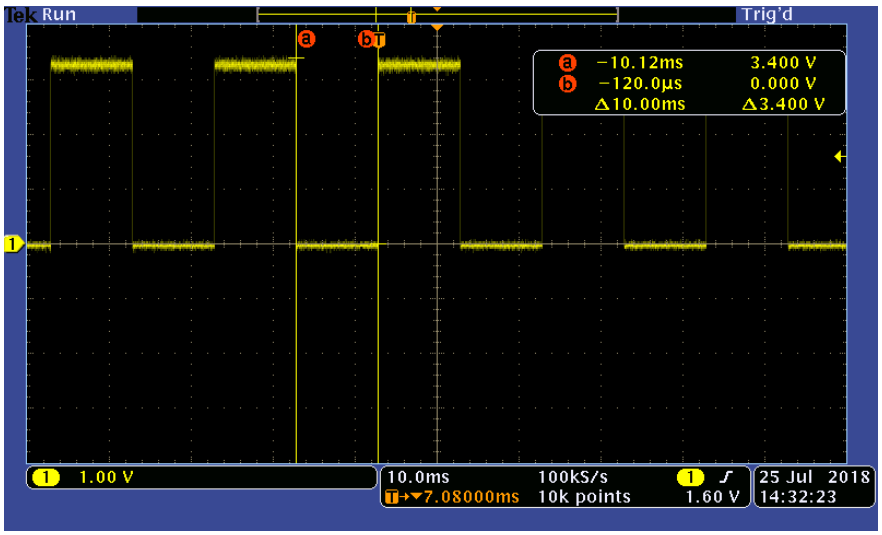

Figure 6: Heartbeat on PE1 on Board

Note: The heartbeat signal is generated by toggling PE1 whenever **`Producer`** runs. The timer frequency is not the same as the frequency of the square wave heartbeat.

Deliverables for Part 2:

- Your code.
- Change the value of **`TEST_PERIOD`** to get a timer frequency of 20 Hz. Show your calculations.
- A **snapshot** of the logic analyzer **or** oscilloscope measuring timer frequency at 20 Hz.

◆

Part 3) Develop the **`Producer`** function and run it from the main function as a periodic thread (i.e., a pointer to this function is passed to **`OS_AddPeriodicThread`**). Specify the **`PERIOD`** macro such that you have a 20 Hz sampling frequency.

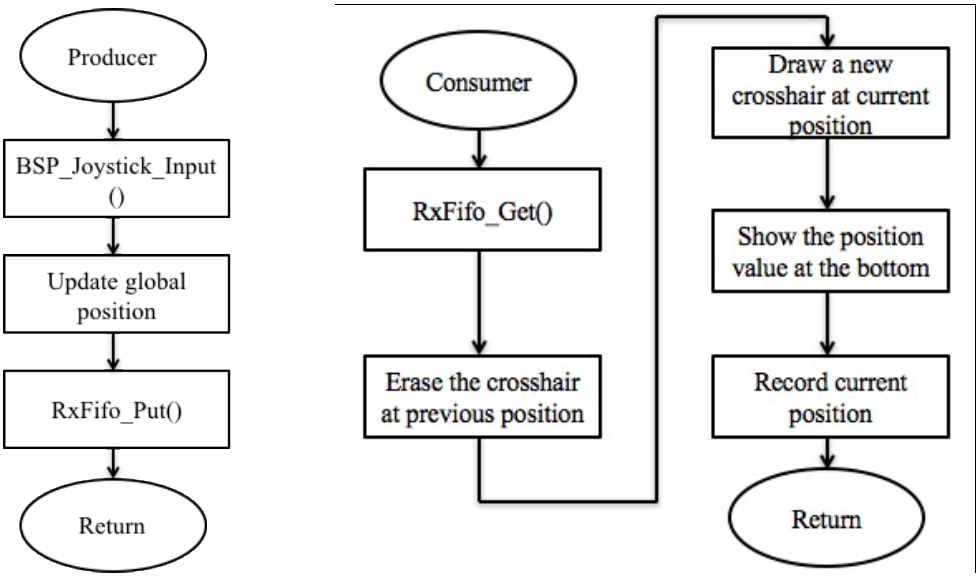

Figure 7: Flowchart for Producer & Consumer Threads

In the producer function, you will use the **`BSP_Joystick_Input`** function to read the raw ADC values into local variables. Notice that the array **`origin`** is defined for you and populated with the initial values of the joystick, which could be used as a reference. You need to calculate the next position of the crosshair based on the raw ADC values and put it into the FIFO. Note that you are moving the crosshair from its current position to a desired position with the joystick. The actual values of the joystick are not directly mapped to the position of the crosshair. You may use the Keil Watch Window to check the range of raw ADC values when you move the joystick. Refer to the following tutorial for the use of the Keil Watch Window:
http://www.keil.com/support/man/docs/uv4/uv4_db_dbg_watchwin.htm.

(**Hint:** Check for the boundary conditions when calculating next crosshair position).

Deliverables for Part 3: Your code.

◆

Part 4) Develop the **`Consumer`** function and call it to run continuously from the main function. This function should display i) the crosshair, and ii) the current crosshair position (X and Y) in the bottom of LCD. You may use **`BSP_LCD_Message`** function developed in Task 1 to show the position values at the bottom of the screen. You can call the **`BSP_LCD_DrawCrosshair`** function to draw or erase the crosshair on the screen.

Deliverables for Part 4: Your code.

◆

Mini Project 1 Deliverables and Grading

The following table lists the deliverables and their corresponding points. Commit and push your changes to your private repository on **GitHub** before the submission deadline. Include a PDF report containing all the deliverables (any calculations, snapshots, answers to questions, and links to videos). The first page of your report should include your name and computing ID. You may upload any required videos to a platform such as YouTube or Google Drive. **Note** that the latest commit before the deadline will be considered your submission. Any major changes to your submitted code and report after the deadline and before posting the grades will be considered a late submission.

Survey: After completion of this mini project, please go to the following link and complete the survey: https://virginia.az1.qualtrics.com/jfe/form/SV_03zq3QPkWQ5eoHb. (Copy the link into browser)
This will be anonymous feedback but will be counted towards your class participation. Attach a **snapshot** of the completion page to your project report.

Deliverables	**Points**
1) Your **code** for the following functions:	
• **`BSP_LCD_Message`**	10
• **`BSP_LCD_DrawCrosshair`**	10
• **`InitTimer1A`**	10
• **`OS_AddPeriodicThread`**	5
• **`Producer`**	15
• **`Consumer`**	10
2) Deliverables for **Part 2**:	
• Show your calculations for **`TEST_PERIOD`** to get a frequency of 20 Hz	5
• A **snapshot** of the logic analyzer **or** oscilloscope measuring timer frequency at 20 Hz	5
3) A **video recording** demonstrating the following functionality:	
• Using joystick to move the crosshair on the LCD	10
• Changes to the position values displayed at the bottom of LCD as crosshair moves	10
• Handling of the boundary cases when crosshair reaches to the LCD borders	10
Total	100