Lab 1: Graphics, LCD, Timer and Interpreter

Xinyuan (Allen) Pan & Qinyu (Jack) Diao

Objective

The goal of this lab is for us to familiarize with TM4C123 MCU development. We implement a ST7735 LCD function, the UART interrupt serial port driver, the periodic timer interrupt driver and the timer-triggered ADC driver. We also design an interpreter that takes command and controls the different modules.

Software Design

```
/*
 * Divide the LCD into two logical partitions and provide
 * an interface to output a string and an integer
 * inputs:          device: 0 for top screen, 1 for bottom screen
 *                          line: 0 - 3
 *                          string: the string to display
 *                          value: the value to display following the string
 * output: none
 */
void ST7735_Message (int device, int line, char *string, int32_t value);

/*
 * Open ADC0SS3 to collect 1 piece of data
 * ADC0
 * SS3
 * Timer2A interrupt (one-shot mode)
 * 10000 Hz sampling rate on 80 MHz clock
 * (This function does not match what is required by Lab 2, and will be redesigned)
 * input: the channel number to open
 * output: none
 */
void ADC_Open(uint32_t channelNum);

/*
 * read the previously recorded ADC value
 * input:  none
 * output: the previously read ADC value
 */
uint16_t ADC_In(void);
```

```
/*
 * Open ADC0SS3 with the specified channelNum at interrupting frequency fs, take numberOfSamples of samples,
 * store them in buffer, then disable the interrupt
 * ADC0
 * SS3
 * Timer2A triggered
 * (This function does not match what is required in Lab2. Will be redesigned.)
 * inputs:  channelNum: the channel number to open
 *                      fs: the interrupting frequency
 *                      buffer: pointer to the array that stores data
 *                      numberOfSamples: number of samples to collect
 */
void ADC_Collect(uint32_t channelNum, uint32_t fs, uint16_t buffer[], uint32_t numberOfSamples);

//******** OS_AddPeriodicThread ***************
// add a background periodic task
// typically this function receives the highest priority
// Inputs: pointer to a void/void background function
//         period given in system time units (12.5ns)
//         priority 0 is the highest, 5 is the lowest
// Outputs: 1 if successful, 0 if this thread can not be added
int OS_AddPeriodicThread(void(*task)(void), uint32_t period, uint32_t priority);

/*
 * Clears the count for the running thread
 * input: none
 * output: none
 */
void OS_ClearPeriodicTime(void);

/* Outputs the number of times the thread has run
 * input: none
 * output: count
 */
uint32_t OS_ReadPeriodicTime(void);

/*
 * the command interpreter for serial port I/O
 * input: none
 * output:          none
 */
void interpreter(void);
```

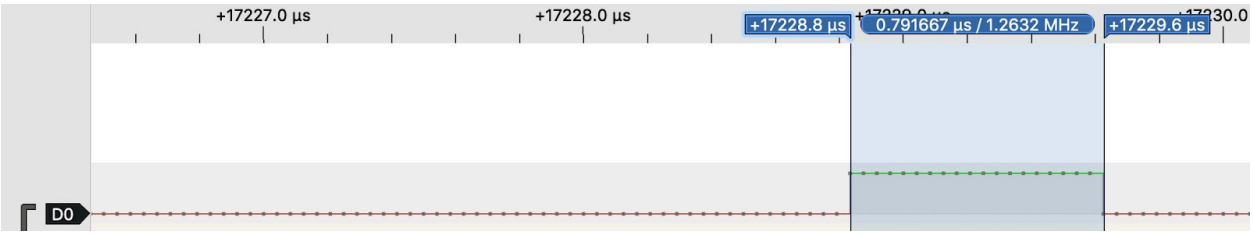
Measured Data

1. Estimated time:

```
00000000 <Timer1A_Handler>:
   0:   b580            push    {r7, lr}
   2:   af00            add     r7, sp, #0
   4:   f7ff fffe       bl      0 <LED_RED_TOGGLE>
   8:   4b07            ldr     r3, [pc, #28]   ; (28 <Timer1A_Handler+0x28>)
   a:   2201            movs    r2, #1
   c:   601a            str     r2, [r3, #0]
   e:   4b07            ldr     r3, [pc, #28]   ; (2c <Timer1A_Handler+0x2c>)
  10:   681b            ldr     r3, [r3, #0]
  12:   4798            blx     r3
  14:   4b06            ldr     r3, [pc, #24]   ; (30 <Timer1A_Handler+0x30>)
  16:   681b            ldr     r3, [r3, #0]
  18:   3301            adds    r3, #1
  1a:   4a05            ldr     r2, [pc, #20]   ; (30 <Timer1A_Handler+0x30>)
  1c:   6013            str     r3, [r2, #0]
  1e:   f7ff fffe       bl      0 <LED_RED_TOGGLE>
  22:   bf00            nop
  24:   bd80            pop     {r7, pc}
  26:   bf00            nop
  28:   40031024        .word   0x40031024

00000000 <LED_RED_TOGGLE>:
   0:   b480            push    {r7}
   2:   af00            add     r7, sp, #0
   4:   4b04            ldr     r3, [pc, #16]   ; (18 <LED_RED_TOGGLE+0x18>)
   6:   681b            ldr     r3, [r3, #0]
   8:   4a03            ldr     r2, [pc, #12]   ; (18 <LED_RED_TOGGLE+0x18>)
   a:   f083 0302       eor.w   r3, r3, #2
   e:   6013            str     r3, [r2, #0]
  10:   bf00            nop
  12:   46bd            mov     sp, r7
  14:   bc80            pop     {r7}
  16:   4770            bx      lr
  18:   400253fc        .word   0x400253fc
```

Here are 38 assembly instructions (including the two LED TOGGLE procedure).
We am not able to estimate in a unit of time, since we do not know how many cycles each instruction take.
2. Measured Time:



The measured time is 791 ns.

Analysis and Discussion

1) What are the range, resolution, and precision of the ADC?

ADC Precision: the number of distinguishable ADC inputs (e.g. 4096 alternatives)
ADC Range: the maximum and minimum ADC input (e.g. 0 to 3.3 V)
ADC Resolution: the smallest distinguishable change in input (in volt). The change in input that causes the digital output to change by 1.

2) List the ways you can start the ADC conversion. Explain why you choose the way you did.

Sampling event triggering sources include:
• processor (software start)
• analog comparators
• external (GPIO pins)
• general-purpose timer
• PWM generator
• always (continuous sampling)
We chose timer-triggering because that provides us the most precise periodic interrupt, also allows the foreground task to keep running while the conversion takes place.

3) You can measure the time to run the periodic interrupt directly by setting a bit high at the start of the ISR and clearing that bit at the end of the ISR. You could also calculate it indirectly by measuring the time lost when running a simple main program that toggles

an output pin. How did you measure it? Compare and contrast your method to these two methods.

Method 1: We can use logic analyzer or scope to measure the interval between two toggles (how long the pin stayed high)

Method 2: for this method, output pin will keep toggling when there is no interrupt. That means when ISR is being executed, the output pin stays constant. We measure the duration where pin stays constant with the scope or logic analyzer.

These two methods are essentially the same in terms of idea. But method 2 will only work if there is only one interrupting thread. We used method 1.

4) Divide the time to execute once instance of the ISR by the total instructions in the ISR it to get the average time to execute an instruction. Compare this to the 12.5 ns system clock period (80 MHz).

791ns / 38 instructions = 20.8 ns/instruction

That means an average of 1.644 cycles per instruction

5) What are the range, resolution, and precision of the SysTick timer? I.e., answer this question relative to the NVIC_ST_CURRENT_R register in the Cortex M4 core peripherals.

Range: the interrupt frequency, bus frequency / (NVIC_ST_CURRENT_R + 1)

I think dividing by NVIC_ST_RELOAD_R makes more sense, but it says relative to the NVIC_ST_CURRENT_R register.

Resolution: the time between each tick, 1/bus frequency

Precision: number of bits in NVIC_ST_CURRENT_R register, 24 bits