Lab 1 Graphics, LCD, ADC, Timer and Interpreter

Michael Coulter and Jake Klovenski

A) Objectives (1/2 page maximum)

The objective of this lab is to develop an interpreter that takes commands from the connected computer and execute them on the TM4C. It is also to develop a device driver for the ST7735 LCD screen that splits the screen into two logical devices. We also wrote drivers to use the onboard ADC. We also implemented a rudimentary periodic thread initializer and handler that runs at a rate of once every 1ms.

B) Hardware Design (none in this lab)

C) Software Design (software documentation in the report and checkin of all files into the submission repository)

1) Low level LCD driver(ST7735_.c and ST7735_.h files)

void ST7735_DivideScreen (void) - separates the LCD into two separate halves.

void ST7735_Message (int device, int line, char *string, int32_t value, uint16_t color) - outputs text to the LCD by deciding which half of the screen to write to, which line on that screen to write to, the string to write and in what color to write it.

2) Low level ADC driver (ADC.c and ADC.hfiles)

void ADC_Open(uint8_t channelNum) - opens a single, specified ADC channel that can be sampled by calling ADC_In.

uint16_t ADC_In(void) - returns one sample from the initialized ADC.

int ADC_Collect(uint32_t channelNum, uint32_t period, uint16_t buffer[], uint32_t numberOfSamples) - enables the specified ADC to sample at the specified frequency. It will then sample the ADC the specified number of times and store the results to the specified buffer.

3) Low level timer driver(OS.c and OS.h files)

Utilizes the TM4C's built in periodic timer 3 to keep a system clock to the 1ms. Runs one periodic task at a given period using the OS_AddPeriodicThread function. This function takes a function pointer, and two uint32_t to specify the period in milliseconds and one to specify the priority. There is also a function called OS_ReadPeriodicTime that returns a uint32_t with the current system time in ms and a function named OS_ClearPeriodicTime that clears the the timer.

4) High level main program(the interpreter)

The interpreter utilizes the UART device drivers to communicate over usb with a computer. The interpreter takes commands from the computer and reacts depending on the input. The following commands have been implemented:

Command	Output
ADC_Read	Outputs the current value at channel 0
Time	Ouputs the current time according to OS_ReadPeriodicTime
Screen	Prompts the user for which screen, which line on that screen, and finally a message.
Clear_Time	Runs the OS_ClearPeriodicTime
Clear_Screen	Resets the ST7735

D) Measurement Data

1)Estimated time to run the periodic timer interrupt

The lab manual says explicitly it cannot be measured.

2) Measured time to run the periodic timer interrupt

1.442 µs

E) Analysis and Discussion (1 page maximum) This section will consist of explicit answers to these questions

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

Range: 0 - 3.3V

Resolution: 0 - 4095, 2^12

Precision: .0008V

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

There are two ways to start the ADC conversion. The first is to simply run the function void ADC_Open(uint8_t channelNum). This function will enable the specified ADC, which can then be software triggered by calling ADC_In(). The second is to call int ADC_Collect(uint32_t channelNum, uint32_t period, uint16_t buffer[], uint32_t numberOfSamples), which will enable the specified ADC to sample at the specified frequency. It will then sample the ADC the specified number of times and store the results to the specified buffer. This is a convenient function for initializing an ADC and collecting multiple samples at a consistent rate all with one command.

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.

We triggered a GPIO pin at the start and end of the ISR. This way is more intrusive because it requires adding two more tasks to the ISR, however it is more accurate because we are given a physical sign of when the ISR starts and ends. The other method of running the program with and without the interrupt leaves more room for error because we are inferring when the interrupt happens based on the output of the main program.

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 12.5ns system clock period (80 MHz).

We recorded 22 instructions in the ISR. 1.442 µs is the recorded time for the ISR. Taking these two numbers we get 65ns/instructions. This is much larger than the 12.5 ns clock cycle.

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: 0 - 0.2097152 seconds
Resolution: 2^24
Precision: 12.5ns