Jackson Paull

Jhp2539

Lab 1 Report

1. OBJECTIVES

The objective of this lab was to create 3 drivers, for ADC0, the ST7735 LCD, and a ms precision timer respectively in addition to a continuous interpreter which can call these drivers and communicate over UART.

2. MEASUREMENT DATA

OS_MsTime had 6 instructions in the ISR.

$$\frac{6 * 12.5ns}{1ms} = 0.0075\%\ Utilization$$

DAS_Task measured 0.01% CPU utilization. See figure 1.

10.375µs execution vs 99.995042ms delay

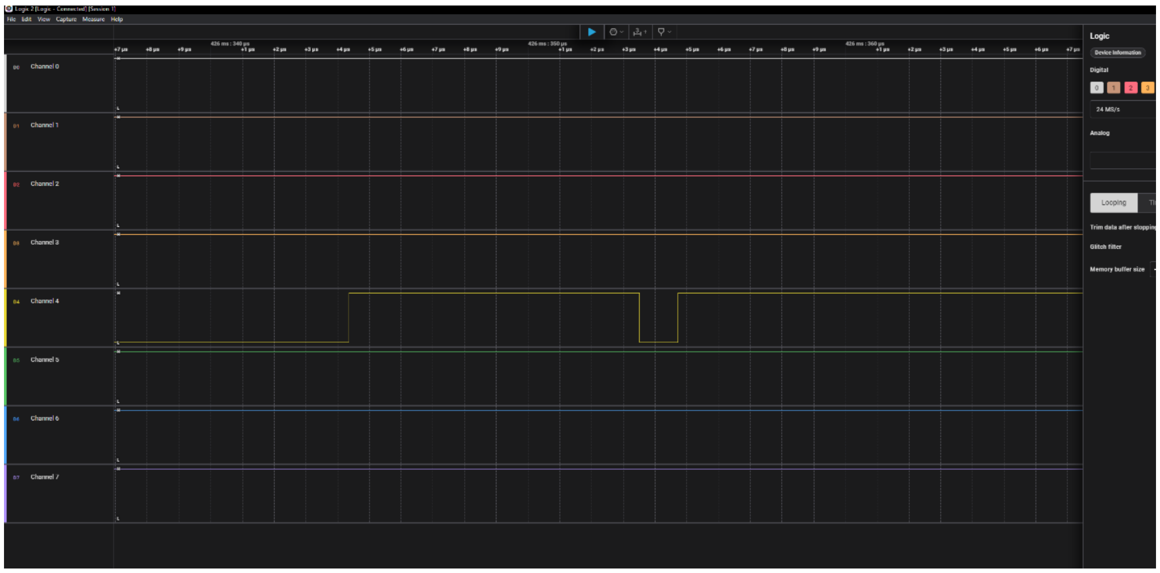

Figure 1. Logic Analyzer capturing LED triple toggle.

3. SOFTWARE DESIGN

Note: See github, tag 'lab1-release' for source code files.

OS_MsTime

I used a simple approach for implementing the time, simply utilizing timer0A to trigger once a ms, and called an ISR which incremented a uint32 counter. Despite the simplicity, it can count for around 50 hours before overflowing, which is acceptable.

Interpreter

For the interpreter I implemented a system which parses an input command string received on UART and then matches the command name with one registered in the system. After a command is found, all of the parameters are passed to the command, which should handle its own typecasting and implement its own functionality. This system has proven to be plenty modular, requiring only to include a function hook paired with a command name in a constant list to add it to the registry.

LCD Driver and ADC Driver

Both of these implementations were as straightforward as they come. For the former, I introduced a device struct which tracks the virtual screen's position on the physical screen. For the later I implemented software triggers with no interrupts.

4. ANALYSIS AND DISCUSSION

1) What are the range, resolution, and precision of the ADC?
    a. The ADC has 12 bits it can distinguish between and saturates at 3.3V. Hence the **range=3.3V**, resolution=80.5mV, precision=2^{12} alternatives
2) List the ways you can start the ADC conversion. Explain why you chose the way you did.
    a. The ADC conversion can be started with an edge triggered interrupt, a timer triggered interrupt, or a software trigger. I chose to use a software trigger because it was explicitly stated to use one in the lab document. However, for the use case needed in this lab, I still would have chosen to implement using a software trigger, as the ADC_In() function is called when needed, and doesn't need to continuously run in the background or pass data to threads as a consistent stream.
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 and output pin. How did you measure it? Compare and contrast your method with these two methods.
    a. I measured ISR run time with triple toggle. The triple toggle method involves flipping a bit at the start of the ISR, waiting a short bit to ensure a logic analyzer will see it, flipping it again, then finally flipping the bit a third time at the end of the ISR. This way, we can measure the time it takes to run the ISR as well as the time in between runs. This is equivalent to the double toggle, we can see the time between runs as the time when the bit is low, and the time for runs as the time when the bit is high. Both of these methods are, in my opinion, superior to measuring the time lost on a program which just flips a bit back and forth. Unless the ISR runs incredibly often, it will be annoying to find an instance where the time loss occurs.
4) Divide the time to execute one instance of the ISR by the total instructions in the ISR to get the average time to execute an instruction. Compare this to the 12.5ns system clock period (80MHz).
    a. When adding bit flipping to OS_MsTime, we find 167ns to execute 12 instructions. This gives an average of 13.9ns for instruction execution time. This is close enough to the actual value that a longer ISR or more accurate logic analyzer could probably achieve better results, but this is good enough.
5) What are the range, resolution, and precision of the SysTick Timer.
    a. Range=209.7ms, resolution=12.5ns, precision=16777216 alternatives.