LAB REPORT 5: SERIAL COMMUNICATION

Written by: [redacted]
Class:
Instructor:
Teaching Assistant:
Deadline:

INTRODUCTION

In this lab report, I will be using the serial transmission system of the MSP430FR2433 to interface through USB with the RealTerm Serial Capture Program. The primary objectives of this lab are twofold. One is to learn how to send data through RealTerm as an input for subsequent actions on the MSP430. For this lab, the blinking frequency of an LED will be adjusted using the numbers that are sent through the MSP430. The other objective is to learn how to receive information from the MSP430 and show it on the RealTerm terminal. This will be used to create a user interface (UI) that will make it so that the user will know what to do and whether their inputs gave results. In addition to this, the protection and utilization of RAM data will be imperative for this lab as the RealTerm program will have to adjust table values as the program is running.

MATERIALS, METHODS, AND RESULTS

For this lab, I primarily used the MSP430FR2433 microcontroller and the RealTerm Serial Capture Program (2.0.0.70). An oscilloscope was occasionally used to measure frequency of the results. For this specific lab report, it was difficult to separate methods and results, so I've placed them both into one section.

To begin, the *UART_Example2.asm* program was used to get familiar with the receiver buffer and transmitter buffer registers and the overall UART interrupt system. The program was used to display my name in the RealTerm terminal.

Figure 1: My name displayed on the terminal.

PART ONE

After this began the first part of the lab. In this part, I was assigned to program the MSP430 to use the values received from the serial communication on RealTerm to adjust the blinking frequency of the LED on MSP430. For the serial communication aspect of this project, the following parameters were required:

Serial I/O	USART 0
Serial Data Frame	1 start bit, 8 data bits, 1 stop bit
Baud Rate	9600
Main Clock Frequency	8 MHz

Table 1: Required UART parameters.

I started with the main clock frequency. For this part of the lab, I used the following equation to figure out how to get an 8 MHz DCO clock frequency:

$$\mathrm{f_{DCOCLK}} = (FLLN + 1) * (\mathrm{f_{FLLREFCLK}} \div \mathrm{FLLD})$$

I used the following settings to achieve a clock frequency of 8 MHz:

FLLN (divider)	243
FLLREFCLK	REFOCLK
Frequency of FLLREFCLK	32768 Hz
FLLD	1

Table 2: FLL DCO Clock Parameters.

Inputting these parameters into our equation, we see that it gives us 8 MHz.

$$\mathrm{f_{DCOCLK}} = (243 + 1) * (32768\ \mathrm{Hz} \div 1) = \ 7{,}995{,}392\ \mathrm{Hz} = \ 8\ \mathrm{MHz}$$

For the configuration of USART 0, I had to ensure that our baud rate was 9600, so I used the following parameters from Table 22-5 of the MSP430 user guide:

UCOS16	1
UCBR	52
UCBRF	0x10
UCBRS	0x0490

Table 3: USART 0 Parameters to Ensure a 9600 Baud Rate.

With these parameters set, our main clock frequency was at 8 MHz and our baud rate was set at 9600. After this, I confirmed that USART 0 was configured correctly by having a startup message sent to RealTerm. This is shown Figure 2 in the following page.

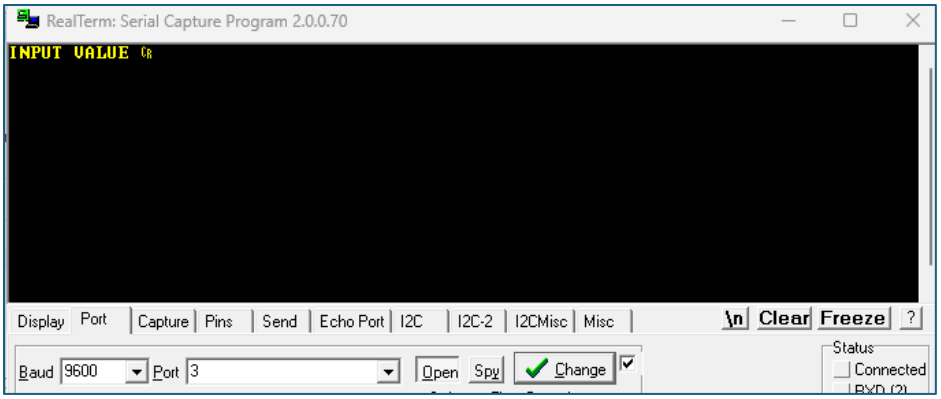

Figure 2: Startup message for Lab 3 Part 1. Baud rate is clearly at 9600.

After confirming that the USART was configured correctly, it was time to set up the LED blinking system. Much of the programming for the LED was borrowed from previous lab projects. However, to ensure a 50% duty cycle I used the following configuration for Timer A0 with the blinking being toggled by CCR1 using toggle mode.

Clock Source	ACLK
Clock Mode	Up Mode
Input Divider	/8

Table 4: TA0CTL configuration.

To connect the blinking frequency to the UART receiver, I had the value that was received from the RX buffer placed into bothTA0CCR1 and TA0CCR0. Ensuring that the blinking frequency is directly controlled by the input value of RealTerm and that the duty cycle remains 50%. With this, I just added a few lines of code to have the program transmit a success message to RealTerm, and this part of the program was almost done.

The last feature I had to add to the program was an insurance that the blink frequency would be at most 16 Hz. This was done with a simple check of the saved RX value in the main loop:

```
update      cmp     #125, rxInd             ; Ensure that timer A is at most 16 Hz
            jge     skipMin
            mov     #125, rxInd
```

Figure 3: Insurance that the blink frequency is a maximum of 16 Hz.

The code in *Figure 3* simply checks to see if the period of the blink signal is too short, if so, then it sets the blink frequency to 16 Hz.

Once all of this was done, a video of the result was taken and attached to this report.

PART TWO

The same parameters and configurations in the first part of the lab were used in the second part. However, for this part of the lab, the objective was to create a program that will integrate button 2.3 and 2.7. Button 2.7 would increment through three LED blink states with different blink frequencies. However, button 2.3 should create a user interface on RealTerm that would allow the user to change the frequency value of the blink state that they are in by sending a number value on RealTerm.

Most of the button programming was taken from the servo section of lab four. However, the difficult part of this lab was to use Random Access Memory (RAM) to store the table that held the timer control values. This is because the RAM table was constantly being corrupted by interrupts and other processes. To mitigate this, I added a few extra values to the table that were meaningless so that those values would be corrupted instead of the needed table values.

```
;-------------------------------------------------------------------------------
;           Table (RAM)
;-------------------------------------------------------------------------------
speedTable:     .word   1000, 2500, 5000, 0, 0, 0, 0    ; Address starts at 0x2ff0
; This stupid interrupt was messing up my table, so I had to add some zeros to
; the end so that it corrupts the zeros. This took me hours to figure out.
```

Figure 4: RAM Table includes extra values to avoid corruption.

However, looking back on the lab lecture for this section, it became clear that I needed to use the '.data' command to ensure protection of this table instead. Either way, this system worked and I was able to move on with the program.

The rest of the program followed this simplified state diagram:

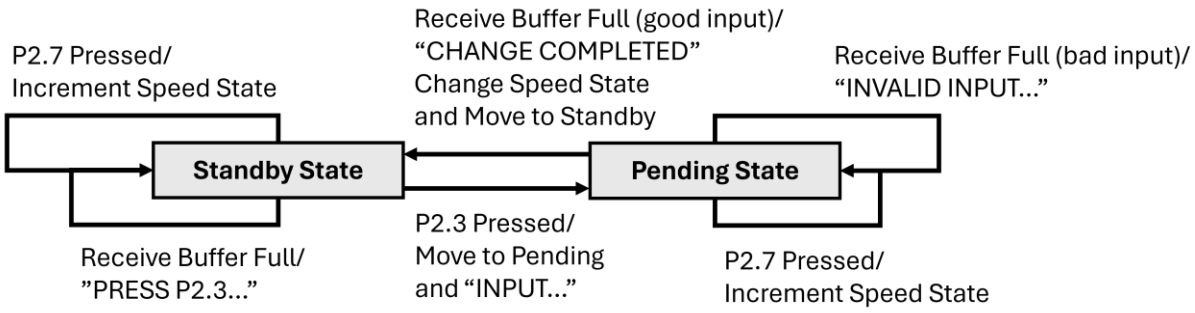

Figure 5: Simplified State Diagram of the Second Part of the Program

With all of this done, a video of the program was taken and attached to this report.

DISCUSSION AND CONCLUSION

In this lab report, the serial communication applications of the MSP430FR2433 microcontroller were explored. These experiments with the blinker speed have shown that the MSP430 can be given and can process inputs that are much more than that of the button inputs by using a serial receiver. Moreover, the microcontroller can now give outputs that allow for straightforward messaging to the user. Together, these two abilities allow for a much easier user experience.

The ease of use of a biomedical instrument is often a result of the user interface of the biomedical instrument. It is because of serial communication like this that we can create user interfaces that will allow non-technical people to begin using the instrument, raising productivity by massive amounts. Ultimately, serial communication has shown to be an imperative tool in an Embedded Biomedical Engineer's arsenal.