

## EE474: Laboratory Report #3

### Section 1: Procedure

#### Task 1

After learning how to use the ADC by watching the class lectures, how to read the temperature sensor from the laboratory document, and how to interface with the PLL drivers provided by the staff, this task required putting each of the important pieces together. The following is a summary of how each piece interacts with each other. The procedure was to implement each component and test it, then proceed with the next component:

- PLL: The system clock frequency is produced by the PLL engine, whose interface is provided by the course staff. Switch SW1 will be used with PRESET3 to set the system clock to 12MHz, and Switch SW2 will be used with PRESET1 to set the system clock to 120MHz.
- PIOSC: In order to avoid changing the counting value for Timers TIM0 and TIM1, this oscillator will be used to provide a constant 16MHz to the timers as an alternate clock configuration.
- TIM0: This timer is a periodic down-counter that triggers once every second to trigger an ADC conversion (which then converts the analog temperature sensor value to a digital value).
- TIM1: This timer is a periodic down-counter that triggers with a frequency of 2Hz to blink the LEDs. The ISR provides the blinking behavior on the LEDs that should be blinking, and turns OFF all other LEDs.
- LEDs: The on-board LEDs D1, D2, D3, D4 are driven by GPIO pins configured as outputs, and the blinking behavior is produced by TIM1 inside an Interrupt Service Routine that runs when the timer expires.
- Switches: When SW1 is pressed, the ISR records the button press and the main loop sets the system clock to 12MHz if it wasn't before. Similarly, when SW2 is pressed, the ISR records the button press and the main loop sets the system clock to 120MHz if it wasn't before.
- ADC0: Gets triggered by TIM0 every second to perform a conversion coming from the temperature sensor. The ISR decides how many LEDs are blinking. The temperature is calculated by the ISR using the following formula, and the blinking behavior is shown in the table below:

$$temperature = 147.5 - \frac{247.5 \cdot ADC0\_SSFIF03\_R}{4096.0}$$

Blinking LEDs	{PF0, PF4, PN0, PN1}	Temperature (°C)	Temperature (°F)
1	0b0001	0 - 20	32 - 68
1 + 2	0b0011	20 - 24	68 - 75.2
1 + 2 + 3	0b0111	24 - 28	75.2 - 82.4
1 + 2 + 3 + 4	0b1111	28 +	82.4 +

- Finally, the main loop waits until a button is pressed. When a button gets pressed (communicated from the ISR), software decides if the system clock frequency should be changed (used the drivers provided by the staff), or if it should remain the same. Then it waits again until the next button press.

## Task 2.1

The first step was to learn about UART in class. Then, reading the TM4C1294NCPDT datasheet is necessary to understand how to interface with the UART in our boards. Next, I had to read the “Putty for UART Communication” document to understand how to communicate with my computer using PuTTY. Finally, the difference between task 1 and task 2.1 is that task 2.1 uses UART to communicate through a virtual COM port with the aid of the On-Board In-Circuit Debug Interface. The UART that I had to use for this task to work is UART0, the baud rate was set to 115200 bits/s as suggested, and pins A0 and A1 are the RX and TX pins. Besides initializing this UART and GPIO pins, the main change is in the main loop and the ADC0 ISR:

- ADC0: After calculating the temperature and deciding on the number of LEDs that must be blinking, the handler communicates to the main loop requesting this temperature to be printed through UART.
- The loop now waits until either a button is pressed or printing is requested. When a button is pressed, the behavior is the same as task 1. When a print is requested, the string “Temperature in Celsius: [X]” is printed through UART, where [X] is the temperature in Celsius as read from the temperature sensor through an ADC conversion to 2 decimal places.

## Task 2.2

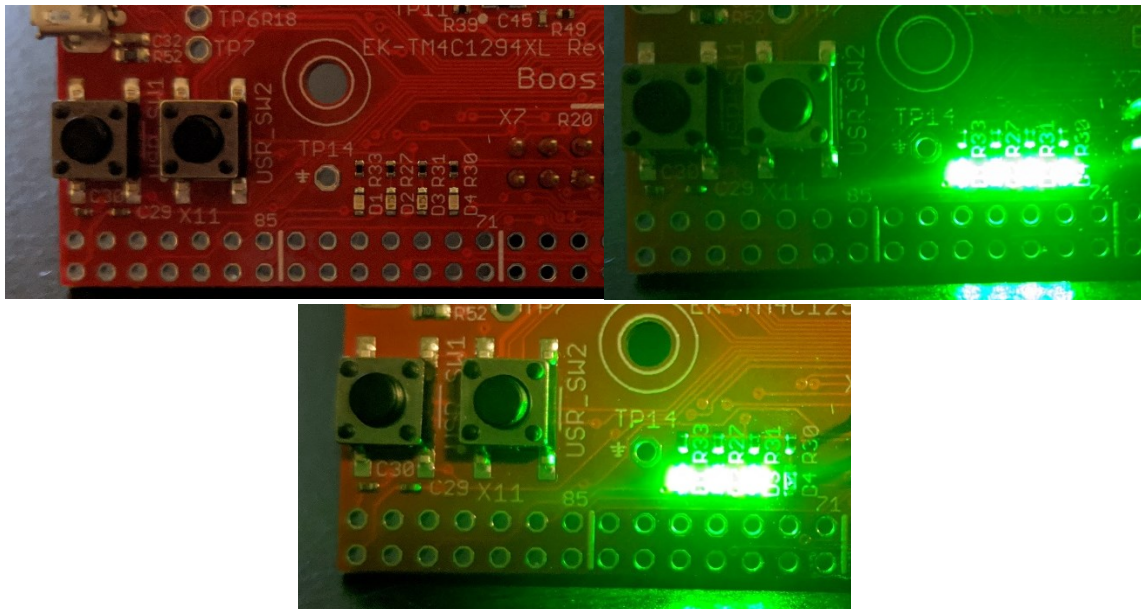
Knowing how to communicate through UART from the previous task, this task reduces to communicating with the HC-06 module through UART which then uses Bluetooth technology to communicate with the computer (replacing the use of the On-board In-circuit Debug Interface). After reading the “Bluetooth for UART Communication” document provided by the staff, it became clear that it was a matter of initializing a UART (UART3 in this case), GPIO pins for TX and RX (A5 and A4, respectively), choosing the correct baud rate (9600 bits/s), and making the physical connections. Then, we have a main loop as follows:

- The main loop is simply waiting until the next character is received, and then that character is sent back. This is repeated indefinitely.

## Section 2: Results

### Task 1

Task 1 successfully meets the specifications outlined in the original laboratory document. When switch SW1 is pressed, the system clock frequency is changed to 12MHz if it wasn't before. In analogous form, when SW2 is pressed the system clock frequency is changed to 120MHz if it wasn't before. Timer TIM0 triggers an ADC interrupt once every second upon time-out. The ADC0 reads the internal temperature of the board and decides which LEDs must be blinking, based on the chart provided by the course staff. At the same time, Timer TIM1 produces the blinking behavior with a frequency of 2Hz. The following images show the output on the LEDs. The top-left image shows an instant where the LEDs are OFF when blinking. The top-right image shows an instant where four LEDs are ON during blinking. The bottom image shows an instant where three LEDs are ON during blinking.



## Task 2.1

The system meets the specifications from the laboratory document. This task produces the same behavior from task 1, with the addition that the temperature is printed through UART after it is calculated (once every second). The value is correctly monitored through PuTTY on my computer running Windows 10. The following screenshot shows the output in PuTTY. During the first measurements, the microcontroller has a system clock frequency of 12MHz, then SW2 is pressed and the temperature starts increasing once the system clock becomes 120MHz:

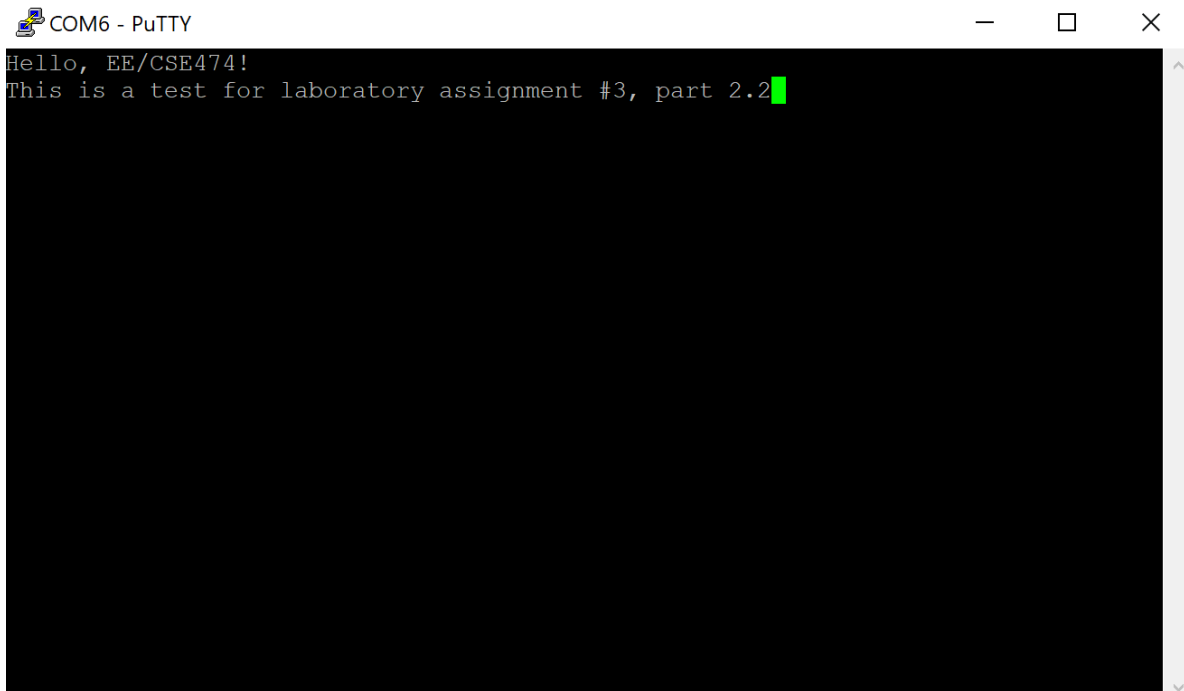
```

COM4 - PuTTY
Temperature in celcius: 26.83
Temperature in celcius: 26.71
Temperature in celcius: 26.83
Temperature in celcius: 26.83
Temperature in celcius: 26.53
Temperature in celcius: 26.83
Temperature in celcius: 26.53
Temperature in celcius: 26.53
Temperature in celcius: 26.83
Temperature in celcius: 26.65
Temperature in celcius: 26.71
Temperature in celcius: 26.83
Temperature in celcius: 26.71
Temperature in celcius: 26.83
Temperature in celcius: 26.53
Temperature in celcius: 26.77
Temperature in celcius: 27.01
Temperature in celcius: 27.38
Temperature in celcius: 27.80
Temperature in celcius: 27.62
Temperature in celcius: 28.10
Temperature in celcius: 28.52
Temperature in celcius: 28.58

```

## Task 2.2

The system correctly uses the Bluetooth module HC-06 and UART communication to receive characters sent from my computer through PuTTY, and each received character is sent back using UART to the Bluetooth module which sends the character to my computer. One can monitor the PuTTY terminal and everything that is typed is printed back. In summary, the system has implemented the “Return-to-Sender” function with success. The following screenshot shows an example of the “Return-to-Sender” feature:



```
COM6 - PuTTY
Hello, EE/CSE474!
This is a test for laboratory assignment #3, part 2.2
```

### **Section 3: Problems Faced & Feedback**

One challenge that I faced was compilation, since task 1 and 2.1 make use of ADC0 but the handler is only slightly different. Therefore, I needed to compile two different versions of ADC0 handler, depending on which task I was running. The solution that I found was to wrap two versions of the ADC0 handler in `#if` directives. If running task 2.1, the version of the ADC0 handler that requests printing temperature is compiled, but if running task 1 then the version of the ADC0 handler that doesn't need to request prints to the UART is used. One consequence is that now the preprocessor must be able to make this replacement on `task1.c` and `task2.c`, so a new header file for choosing which task is ran must be introduced. In summary, the problem was solved satisfactorily.

Feedback about the laboratory assignment is positive. I believe having the opportunity to learn about each of the components necessary for this laboratory are very important. For example, signals in the real world are continuous, which make interfacing with an ADC an essential task in the engineering world. Communication between devices is also very important in modern designs, making learning about UART very valuable. Finally, each year wireless communication is becoming more dominant, so having experience with Bluetooth is definitely important.

One tip for future laboratories is to learn to harness the power of the preprocessor. Sometimes it is easy to forget all the features that it has that can make our lives easier. For instance, I was able to satisfactorily compile the code for both tasks 1 and 2.1 which define the ADC0 handler but have slightly different implementations. Another tip is to write code that is modular. For example, I wrote a *common.c* file that contains the code that is common for both tasks 1 and 2.1, which was only possible by writing code that can be re-used.