Milestone 2

ARM Cortex M4 UART and TIMERs

"When a man sits with a pretty girl for an hour, it seems like a minute. But let him sit on a hot stove for a minute and it's longer than any hour."
– Albert Einstein (1879-1955)

THIS MILESTONE has you explore the UART of the ARM Cortex-M4. Your goal in this milestone is to write a couple of simple programs to exercise the UART and communicate with your laptop.

2.1 Hardware

2.1.1 Schematic

The target hardware schematic[1] is indispensable for the embedded systems designer. For any component used in this milestone – and going forward, all future milestones – you should be familiar with the electrical paths and circuits that are associated with that component. You should be able to justify the circuit topology chosen by the board designer and any component values. The datasheets for these components will also be useful for your investigation.

2.1.2 GPIO Inventory

Your GPIO pins are your most precious resource. For this milestone[2], you must create a spreadsheet named `netID_gpio_m2.xls` or `netID_gpio_m2.ods`, where *netID* is your Tennessee Tech network login ID. This file must enumerate every available GPIO pin on your processor, to what devices it is connected, and any/all possible pin configurations (input with pull-up, push-pull output, open-drain output, A/D input channel, PWM output, etc.) that is used in your design.

[1]The Nucleo board schematic can be found on the lab website and at STMicroelectronics.

[2]You will need to create a new version of this spreadsheet for every assigned milestone this semester. Change the filename to contain *m3* for milestone3, *m4* for milestone4, etc.

2.2 Software

2.2.1 Software preparation

Using the schematic for your board, develop an appropriate header file[3] called `embsysS20.h` that will create user-friendly definitions and macros for manipulating *all* hardware on the target board. Your macros should be written to ensure atomic operation and prevent errors if the header file is included multiple times. Macro and definition names should be consistent.

All code and hardware developed in this lab should follow the Python and C language coding conventions as well as the hardware conventions.

You should create macros and definitions to manage and fully utilize[4] the following components:

- LD2 (equipped with green LED)
- B1 (equipped with blue button)

Create macros (with the appropriate software structures to support) to read and control the resources above:

```
TURN_ON_LD2()
TURN_OFF_LD2()
TOGGLE_LD2()
IS_LD2_SET()
IS_LD2_RESET()
IS_B1_PRESSED()
IS_B1_RELEASED()
```

2.2.2 System Initialization

Write your system initialization code to setup your hardware and software for the milestone. Use a system clock of 40 MHz. Initialization code is artitrary and often cryptic. Init code is famously difficult to decipher in maintenance. Your code, especially the init code, should be well commented, maintainable, and use the macros created above. Be sure your code complies with the coding conventions.

Do not use magic numbers. Use the bit-field constants provided by the libOpenCM3 library whenever possible. If you must "create" a magic number, it should be encapsulated within a #define with a very detailed comment block. All bit-fields should be calculated invidivually and OR-ed together to allow easy changes in maintenance.

[3]You can expect this header file to grow longer and more full-featured as the semester progresses.
[4]Initialization, use, and shutdown, as required

2.3 Program m2: UART and TIMERs

2.3.1 Circular buffer

You will need to write a small library to manage circular buffers. Your code should be modular, portable, and well-documented. Buffers should be byte-sized widths and the length is determined at compile-time. You may assume that the buffer length is a power-of-two. You will need routines to (1) initialize the buffers, (2) write data to the buffer, (3) read the next items from the buffer, and (4) read the most-recent item from the buffer. Your circular buffer code should be written such that it does not suffer from foreground-background data corruption problem and is safe to use in ISRs.

2.3.2 Program to cipher/decipher Vigenere codes

This program should be named `m2a.c`. This single progrm will encode and decode text with a Vigenere[5] cipher. The key text for your Vigenere cipher is "TENNESSEETECH". Plain text is supplied to your Nucleo from your PC over the serial interface at 57600 baud with 8 data bits, 1 start bit, and 1 stop bit with odd parity. Compute the actual baud rate you design implements and the percentage error.

The program returns the encrypted text to the PC over the serial link. Your code will "poll" the UART transmit and receive flags to determine when to send and receive a byte, respectively. Use your circular buffer routines. $\boxed{\text{Do NOT use interrupts!}}$

When the pushbutton B1 is held, the program will deciper the encrypted text using the key. Encrypted text is sent to the PC to the Nucleo with decrypted test returned to the PC ver the serial port.

For this program, you may wish to investigate the follow libOpenCM3 functions and constants:

```
gpio_set_af()
usart_disable_tx_interrupt()
usart_set_baudrate()
usart_set_databits()
usart_set_stopbits()
usart_setmod()
usart_parity()
usart_set_flow_control()
usart_sent()
usart_recv()
USART_CRx_RXNEIE
USART_CRx_TXEIE
USART_ISR_RXNE
```

Identify a free timer (not SYSTICK) on your processor and investigate the timer configuration functions and constants in liOpenCM3.

[5] If you are not familiar with Vigenere ciphers, Wikipedia has a very nice and gentle introduction.

When encrypting text from the UART (B1 is not pressed), configure your selected timer to expire every 0.5 s, thereby setting its flag. Detect the timer expiration via its flag by polling. $\boxed{\text{Do NOT use interrupts!}}$ Toggle the Nucleo LED at each timer expiration. When decrypting text (B1 is pressed), reconfigure the flag to expire every 0.125 s. This will provide a more rapid flash notifying the user of the decryption mode. When B1 is released, restore the timer period back to 0.5 s.

2.4 In-lab Checkoff

- Demonstrate the proper operation of your program m2 by first send a paragraph of text from your PC terminal program to your Nucleo. Your terminal program should display the returned “encrypted” text. Save this text.
- Next, pressing button B1, send the encrypted text from your laptop terminal program to the Nucleo. Your terminal program should receive the decrypted text from the Nucleo. The text should match the text in the first step.

2.5 Submission

- Sumbit a single archive `imstudent42_m2.zip` file or `imstudent42_m2.tgz` file via iLearn. Your archive should have a consistent directory structure. Apart from changing the specific locations of the libraries in the Makefile, the TA should be able to simply type “make” to build your application to run on his/her hardware. If this is not the case, you will receive zero credit.