Milestone 3

ARM Cortex M4 Interrupts

"I've always objected to doing anything over again if I had already done it once."
– Rear Admiral Grace Hopper (1906-1992)

THIS MILESTONE has you explore the interrupt system of the ARM Cortex-M4. Your goal in this milestone is to extend your previous ARM program to exercise the timer and UART utilizing interrupts.

3.1 Hardware

3.1.1 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.

3.1.2 GPIO Inventory

Update your GPIO inventory spreadsheet to reflect the GPIO pins used in this milestone. Rename your spreadsheet `netID_gpio_m3.xls` or `netID_gpio_m3.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.

3.2 Software

3.2.1 System Initialization

Write your system initialization code to setup your hardware and software for the milestone. Use a system clock of 80 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.

3.3 Program m3

In this milestone, we will modify and augment the functionality in the previous milestone m2.

3.3.1 Timers

Use your source code and Makefile from Milestone #2 as a template to start this program. This project's main file should be named `m3.c`. The program will flash the user LED LD2 at different frequencies depending on the user's commands via the UART. Use timer #2 to create the LD2 timing and the timer #2 ISR to toggle the LED. Upon power-on reset, flash LD2 once per second with a 50% duty cycle.

Verify your code operation with an oscilloscope.

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

```
nvic_enable_irq()
rcc_periph_reset_pulse()
rcc_periph_clock_enable()
timer_set_prescaler()
timer_disable_preload()
timer_set_period()
timer_enable_counter()
timer_enable_irq()
timer_clear_flag()
timer_get_flag()
```

3.3.2 UART with interrupts

Modify your code to use UART interrupts on completed UART hardware RX and TX operations. Continue to implement the Vigenere coding and decoding specified in `m2` via your circular buffers. Configure your UART to use 57.6k/8/N/1.

Remember that ISR should do the minimal amount of processing to service the hardware. Therefore, your UART RX ISR should simply copy the received byte into the incoming circular buffers, and the UART TX ISR should copy the next out-going character from the out-going circular buffer. The Vigenere operations should be performed in the main loop. Your program will encrcypt or decrypt UART text as prescribed by the button B1.

The UART stream in this milestone will be able to change the LD2 flash frequency or inquire about the LD2 frequency. If the UART contains the string "!!!L", these four characters will NOT be encrypted or decrypted by your Vigenere cipher. Instead, your program will insert the LD2 flash period in milliseconds in human-readable ASCII text into the UART RX stream. For example, after POR the LD2 flash period is 1000 ms. Therefore, your program will respond to "!!!L" with "1000".

The user can also change the LD2 flash frequency via the UART. If the user sends "!!!S*nnnn*" to your program, the LD2 flash period will be set to *nnnn* ms with a 50% duty cycle. Note that the period *nnnn* must always be four (4) ASCII digits. The minimum LD2 flash period ency is therefore 1 ms, and the maximum flash period is 9999 ms. Your program must correctly determine the contents of the timer 2 registers to create the desired LD2 flash period.

3.4 In-lab Checkoff

- Demonstrate the proper operation of program m3 using a large file of text chosen by the TA. Send the plain text to your board. Capture the encrypted text. Then, send the encrypted text to the Nucleo board and capture the original plain text in response. Demonstrate the setting and query of the LD2 flash period.

3.5 Submission

- Sumbit a single archive `imstudent42_m3.zip` file or `imstudent42_m3.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.