Assignment 5: Bare Metal LED Flasher

- Submit deliverables to CourSys: https://courses.cs.sfu.ca/
- This assignment is to be done **individually or in pairs**.
- Post questions to Piazza discussion forum.
- Do not give your work to another student/group, do not copy code found online, and do not post questions about the assignment to other online forums.
  - You may use all sample code Dr. Fraser has posted on the course website this semester, and anything in the TI AM335x StarterWare package (must conform to license).
- See the marking guide for details on how each part will be marked.

1. Application Description

Create a bare metal application named `lightBouncer.bin` to run on the BeagleBone with a Zen cape which provides the following functionality:

1.1 Start-up

On the serial port at start-up:

1. Print a welcome message with your name.
2. Print the reset sources (and clear this register) (see page ~1336 in TRM).
   - The reset source register indicates all sources of resets since the register was cleared; print out a comma separated list of all the indicated reset sources.
   - The reset source register is located at base address 0x44E00F00.
   - Bits labeled "R/W1toClr-0h" mean that if you write a 1 to the bit it will clear it. You'll need to do this after you have read the current reset source(s) in order to clean it up for the next reboot.
   - Do not use any StarterWare methods to access this register (if you find any), other than the `HWREG()` macro. You'll likely have to define your own base and offset constants because they are not in StarterWare header files.
   - Hint:
     - Test with each of: cold boot (unplugging power), external reset (press the reset button beside the LEDs), and watchdog reboot.
3. Print the help information about accepted serial commands.

1.2 10ms Timer

1. Configure a timer for repeated 10ms interrupts.
   - This interrupt will be used for multiple parts of the application.

1.3 Watchdog

1. Configure the watchdog to reset the board after 5s.
2. In the background task, hit the watchdog every 10ms.
   - Hint: Have the timer ISR set a flag, which is checked in the main loop. Have the flag indicate that 10ms have elapsed, and hence will be time to hit the watchdog.
3. A serial port command can disable hitting the watchdog; more below.

1.4 Serial

1. Use interrupts for the serial port to read a single character when a key is typed.
   - Use the StarterWare `UARTCharGetNonBlocking()` function to read the character.
   - Do not require the user to hit ENTER.
   - *Hint*: You should only call `IntAINTCInit()` once in your entire program (it resets the ARM Interrupt Controller).
     - This function will remove any currently configured interrupts, and is called in the sample code for both the serial port Rx and the timer (both using interrupts).
     - Comment-out/delete the call from the second interrupt that you setup. For example, if your `main()` first initializes the timer and then the serial port, have the serial port *not* call `IntAINTCInit()`.
2. In the background task, process the key typed which was read in the ISR.
   - You may assume that the application will have completed processing the first keystroke before a second one is entered by the user (i.e, you don't need a queue to read in key presses for processing: the CPU is fast, the serial port is *slow*).
   - *Hint*: Use a variable to store the key pressed in the ISR which is to be processed in the background task.
   - *Hint*: Have your background process repeatedly call a function in your serial prompt module to check if there is any work to be done.
3. Accepted commands (must be case insensitive):

```
?   : Display this help message.
0-9 : Set speed 0 (slow) to 9 (fast).
a   : Select pattern A (bounce).
b   : Select pattern B (bar).
x   : Stop hitting the watchdog.
```

   - When the user enters a command to change the speed or flash-pattern, call functions in your LED module (below).
   - When the user enters '`x`', stop hitting the watchdog in the background task (board should reboot due to the watchdog).
   - If you enter an unknown command, print an error message and show the help list.

Hint: In addition to the `serial.h/.c` module provided on the course web page in the "fake typer" program, also create a new serial prompt module which handles your your applications use of the serial port at a higher level.

1.5 LEDs

1. Flash the four LEDs on the actual BeagleBone in one of two patterns:
   - **Bouncing:** Turning on one LED at a time, make the lit LED appear to bounce back and forth. The illumination sequence will be (one time-period each):
     0, 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, 1, 0, ….
   - **Bar:** Turn on multiple LEDs, as needed, such that a bar of LEDs appears to grow and shrink. For one time-period each, turn on LEDs 0 through *N* (inclusive), where
     N = 0, 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, 1, 0, ….
2. The time-period for each flash is given by the following formula:
   ```
   flash time-period = 10ms * (speed divider factor)
   ```
   where
   ```
   speed divider factor = 2 (9 - speed)
   ```
3. *Hints:*
   - Have the 10ms timer ISR call a function in your LED code which controls the LEDs.

1.6 Joystick

Pressing and releasing a direction on the joystick has the following effect:

1. Left: Toggle the flash-mode between bounce (pattern A) and bar (pattern B).
   i.e., which ever mode it is currently in, switch to the other mode.
2. Up (bonus): increments the speed by one (clamped to 9).
3. Down (bonus): decrements the speed by one (clamped to 0).

Suggestion:

- Have the 10ms timer ISR call button call a function to check the state of the joystick.
- Note that holding the joystick has no effect; it is only on release that has an effect.
- When a joystick action is detected, call into the LED module to change the pattern/speed.
- If attempting the bonus:
  - The board support package for the BeagleBone Black in StarterWare defines the following two function which you may want to call to initialize the GPIO: `GPIO0ModuleClkConfig`, `GPIO1ModuleClkConfig`.
  - Find pin numbers for joystick directions by consulting Zen cape schematic. Find GPIO mapping via “header table” files on course website.

1.7 General Requirements

1. Must have multiple modules, each presenting a clear interface to the rest of the code through a `.h` file.
   Suggested modules include:
   - controlling the LEDs,
   - reading the joystick,
   - managing the serial port prompt, and
   - provided hardware abstraction modules: timer, watchdog, serial.

   You may add more modules if it improves the readability of your code.
2. No global variables with external linkage.
3. Only the functions exposed through `.h` files may be non-static (except `main()`). All other non-exported functions must be `static`.
4. You must not have any `printf`-style statements called from within code that is called

while processing an interrupt (i.e., the ISR must not trigger a `printf`-style):
All `printf`'s must be done in the background task.

5. Code quality is always important:
   - All functions, variables, constants, and files must be well named.
   - Use perfect indentation (as always!).
   - Add a brief comment at the top of each `.h` file (at least) to explain that module's purpose.
6. Suggestion: Try to use the size-specific C types. For example use `uint8_t` instead of `char`, and `uint32_t` instead of `unsigned int`. (No marks for this, but good to work with; must `#include <stdint.h>`).

2. Deliverables

Submit the items listed below to the CourSys: https://courses.cs.sfu.ca/

1. `as5-bouncer.tar.gz`
   Compressed copy of source code and build script (`Makefile`).

   Archive must expand into the following (without additional nested folders to find the `Makefile`)

   ```
   <assignment directory name>
    |-- Makefile
    \-- <dependencies such as .c, .h files>
   ```

   `Makefile` must support both the `make` and `make all` commands to build your program to `$(HOME)/cmpt433/public/baremetal/lightBouncer.bin`
   (Do not use relative paths for getting to the `cmpt433/public/baremetal/` directory because the TA may build from a different directory than you. Assume the marker has the setup recommended in the bare metal guide.)

   Hint: Compress the `as5/` directory with the command

   ```
   $ tar cvzf as5-bouncer.tar.gz as5
   ```

   You may use a different build system than make (such as CMake). If you do, include a file named `README` which describes the commands the TA must execute to install the necessary build system under Ubuntu 16.04, and the commands needed to build and deploy your project to the `~/cmpt433/public/myApps/` directory. The process must be straightforward and not much more time consuming than running `make`.

To submit, you'll need to create a group. If you worked individually, it will be a group of 1.
Remember that all submissions will automatically be compared for unexplainable similarities.

Sample output on following page

3. Sample Serial Output

```
LightBouncer:
  by Brian Fraser
-----------------
Reset source (0x1) = Cold reset,

Commands:
 ?   : Display this help message.
 0-9 : Set speed 0 (slow) to 9 (fast).
 a   : Select pattern A (bounce).
 b   : Select pattern B (bar).
 x   : Stop hitting the watchdog.
 BTN : Push-button to toggle mode.
?
Commands:
 ?   : Display this help message.
 0-9 : Set speed 0 (slow) to 9 (fast).
 a   : Select pattern A (bounce).
 b   : Select pattern B (bar).
 x   : Stop hitting the watchdog.
 BTN : Push-button to toggle mode.
z
Invalid command.

Commands:
 ?   : Display this help message.
 0-9 : Set speed 0 (slow) to 9 (fast).
 a   : Select pattern A (bounce).
 b   : Select pattern B (bar).
 x   : Stop hitting the watchdog.
 BTN : Push-button to toggle mode.
0
Setting LED speed to 0
3
Setting LED speed to 3
b
Changing to bar mode.
a
Changing to bounce mode.
a
Changing to bounce mode.
x
No longer hitting the watchdog.
<<< Here the board actually reboots, after 5s>>>
```