Task 3

ESOS UI Service

"A lot of times, people don't know what they want until you show it to them."
– Steve Jobs (1955-2011)

PRACTICAL embedded systems that interact directly with users need an ergonomic and efficient user interface. Users need to telegraph their intentions to the systems and the embedded systems needs to provide feedback to the user. In this task, you will develop a new "user-interface service" for ESOS to provide to ESOS application developers with a clean and direct method by which to interace with users. The systems' interaction with users is done via provided hardware devices[1], *e.g.* the pushbuttons, LEDs, rotary encoder, the potentiometer, and the liquid crystal display. You will develop a small demonstration application using your new service.

3.1 ESOS UI Service

User-interface components are often platform-specific, and the ECE4723/6723 lab target board is no different. You will develop an UI service in ESOS to ease the developer's burden in configuring and reading the UI devices on your target board. Use the skeleton ESOS code files `esos_uif14.c` and `esos_uif14.h`. These files will contain the data structures, macros, and code to implement your user interface. Furthermore, these files should be generic in that they only contain code to create the UI functions described here. No application-specific code should be included in these files. You will use the user interface service created in this assignment in the varied user applications assigned to you throughout this semester.

3.1.1 ESOS UI Service Internals

In the UI service files named above, use the data structure variable named `_st_esos_uiF14Data` that contains the "private" variables you need to implement the user interface described here. This "private" variable should be declared in global memory space. The UI service will provide the application programmer with "getters" and "setters". The application programmer should *not* modify the variables in your structure directly as the structure is subject to change per this lab specification.

[1]The LCD and the potentiometer are quite different devices and more involved than the simple buttons and LEDs. We will address these two devices individually in subsequent weeks.

3.1.2 ESOS UI Service Configuration

Write the contents of the function `config_esos_uiF14()` that configures the UI hardware and UI service software structures for use. The function `config_esos_uiF14()` will by called by the application programmer in application's `user_init()` and signals that the user application will employ your UI service.

3.1.3 ESOS UI Service Operation

Complete the ESOS "user" task `__esos_uiF14_task` that runs every 10 ms. Although an "user" task[2], this task will be the crux of the ESOS UI service. This task will debounce the switches and maintain the software-accessible state of the user-interface. All of the function listed in this section comprise the "public" API for the UI service and should be defined in your `esos_uiF14` files. All of these functions are non-blocking and can be called from inside or outside of a task context.

Push-button switches

```
esos_uiF14_isSWxPressed()          where x is 1,2,3
esos_uiF14_isSWxReleased()         where x is 1,2,3
esos_uiF14_isSWxDoublePressed()    where x is 1,2,3
```

These function give the "debounced" state of the three push buttons on the target board. Determine the appropriate period of time to declare the switches "double-pressed". The appropriate duration that defines "double-pressed" will be determined by a consensus of your team members – acting as your human-machine interface focus group. Reading the `esos_uiF14_isSWxDoublePressed()` function will "clear" the double-pressed state of that switch. Obviously, a subsequent press of the switch should also "clear" the double-pressed state.

LEDs

```
esos_uiF14_isLEDyOn()              where y is 1,2,3
esos_uiF14_isLEDyOff()             where y is 1,2,3
esos_uiF14_turnLEDyOn()            where y is 1,2,3
esos_uiF14_turnLEDyOff()           where y is 1,2,3
esos_uiF14_toggleLEDy()            where y is 1,2,3
esos_uiF14_flashLEDy(v)            where y is 1-3, v period in ms
esos_uiF14_turnzLEDOn()            where z is Red, Yellow, Green
esos_uiF14_turnzLEDOff()           where z is Red, Yellow, Green
```

Operation of these functions should be self-evident. Since several of these functions perform similar operations, you should write your code such that the actual hardware change is done in one place. The other functions should call into that function so that any corrections or changes to hardware manipulation in future revisions only occurs in one code location.

[2]Since `__esos_uiF14_task` is defined as an user task, it will be registered during the execution of `user_init()` by calling the function `config_esos_uiF14()` above.

The functionality `esos_uiF14_flashLEDy()` should be wholly implement inside of your ESOS UI service "user" task `__esos_uiF14_task`. The argument of `esos_uiF14_flashLEDy()` is the flash *period* in number milliseconds. Very short duration flashes and many flash periods are not possible given the period of the `esos_uiF14_task`. Your code should strive to create the most accurate flashing period possible under any given circumstance. Your code documentation should clearly document and explain any operational limitations.

Rotary Pulse Generator (a.k.a. Rotary Encoder)

```
esos_uiF14_getRpgValue_u16()        return uint16
esos_uiF14_isRpgTurning()           returns TRUE or FALSE
esos_uiF14_isRpgTurningSlow()       returns TRUE or FALSE
esos_uiF14_isRpgTurningMedium()     returns TRUE or FALSE
esos_uiF14_isRpgTurningFast()       returns TRUE or FALSE
esos_uiF14_isRpgTurningCW()         returns TRUE or FALSE
esos_uiF14_isRpgTurningCCW()        returns TRUE or FALSE
esos_uiF14_getRpgVelocity_i16()     returns int16 (>0 for CW, <0 for CCW)
```

Your ESOS UI service needs to maintain a RPG counter (as an `uint16`) that is incremented or decremented based on RPG rotation. By keeping track of the "time" between RPG state changes, you can estimate the rotational velocity of the RPG. Determine the appropriate RPG velocities that would be considered "slow", "medium", and "fast" through a consensus of your team (your human-machine interface focus group). In general, a "fast" velocity should be quite rare and correspond to the user spinning the RPG as quickly as possible.

3.1.4 ESOS UI Service Task Support

In your ESOS UI service files, flesh out the following blocking calls to aid ESOS task in using the UI serivce from the user task context:

```
ESOS_TASK_WAIT_UNTIL_UIF14_SWx_PRESSED()           where x is 1,2,3
ESOS_TASK_WAIT_UNTIL_UIF14_SWx_RELEASED()
ESOS_TASK_WAIT_UNTIL_UIF14_SWx_PRESSED_AND_RELEASED()
ESOS_TASK_WAIT_UNTIL_UIF14_SWx_DOUBLE_PRESSED()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_UNTIL_TURNS()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_UNTIL_TURNS_CW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_UNTIL_TURNS_CCW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_MEDIUM()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_MEDIUM_CW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_MEDIUM_CCW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_FAST()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_FAST_CW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_TURNS_FAST_CCW()
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_MAKES_REV(y)      where y is # revs
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_MAKES_CW_REV(y)
ESOS_TASK_WAIT_UNTIL_UIF14_RPG_MAKES_CCW_REV(y)
```

3.2 ESOS UI Sample Application

In a file named `t3_app.c`, you will need to implement an user application[3] that exercises all of the functions described here. The task will do the following using this lab's code:

- a heartbeat will flash with period 500 ms on LED3
- LED2 state is determined by the RPG.
  - ✧ If no RPG movement, LED2 is off.
  - ✧ If RPG is moving slowly, LED2 is on.
  - ✧ If RPG is moving medium, LED2 flashes with period 500 ms.
  - ✧ If RPG is moving fast, LED2 flashes with period 100 ms
- LED1 state is determined by the state of the switches. SW3 determines whether LED1 shows the state of SW1 (SW3 released) or SW2 (SW3 pressed)
  - ✧ LED1 is illuminated when SW1/2 is pressed
  - ✧ LED1 is flashed three times rapidly whenever SW1/2 is double-pressed
- All SW state changes are signaled by writing the appropriate string to the serial port, e.g. "SW1 pressed", "RPG turning CW fast", "SW2 double-pressed", etc.
- The serial port should provide a menu or interface whereby the user can dynamically select/experiment with the button double-press periods, and RPG thresholds for determining slow, medium, and fast rotation, etc. The structure and appearance of this menu on your laptop is up to you; however, the menu should be easily understood by the uninitatied users (, i.e. the TA and Dr. Bruce) without external documentation.
- Before the TA check-off, determine experimentally the optimal periods above. Include comments in your code documenting these value. After TA check-off demonstration, modify your UI service code to use these periods by default.

3.3 Check Off

Demonstrate the following things to the TA:

1. Commit your project to your repository to your "trunk" folder in accordance with the procedure provided by the TA. Your Mercurial repository should all files required to produce your "build", including the appropriate build file(s).
2. A "tag" release in your Mercurial repository called `task3` that can be recalled at any time to build this task's deliverables.
3. Using the target board specified by the TA, demonstrate the proper operation of the two sample sensor applications.

[3] Recall that use of `printf()` and `floats` is forbidden in this class.

3.4 Submission

Each team should submit the following items to their repo. Send a notification email to the TA as to the location of your submission.

- Schematic errata document
- Component selection and justification spreadsheet
- Stuffing, soldering, and partial-build testing procedures
- well written, well commented, MCU code[4] for the new ESOS sensor service
- well written, well commented ESOS application code for `t3_app.c`
- any unit test or test bench files used to verify your code created for this task
- updated design review forms and software metrics

[4] Your MCU firmware must adhere to the *C* language coding standards.