Milestone 5

The Peripheral Target Board (PTB)

"Change is the law of life, and those who look only to the past or present are certain to miss the future."
– John F. Kennedy (1917-1963)

THIS MILESTONE has you start interfacing your ARM-Cortex STM32L4 processor to peripheral devices located on the EduBase-V2 board [1]. You will develop a multitasking ESOS application to exercise some of the target board hardware. Furthermore, you will create a team repository to implement source control management and collaborative development for this milestone and all future lab milestones.

5.1 Hardware

Your team has checked-out a PTB from the department. Your PTB should not require an adapter board.

5.1.1 Schematics

The schematics for the Nucleo board and the PTB can be found on the class website. As the semester progresses, we will create, delete, or modify our connections between the Nucleo board and the PTB. The schematics for the Nucleo, PTB, and adapter board (if used) are critical information for your design progress. Have them handy at all times and become very familiar with them.

Several versions of the PTB exist and are used. They are very similar. Be sure to identify which PTB you have and use the correct schematic.

To facilitate this milestone and the future milestones, study the PTB schematic and all of the peripherals contained therein. You need to be familiar with the justification for component (resistor, capacitor, etc.) selection on the schematic.

[1] *www.trainer4edu.com/edubase_v2/f446.html*

Determine the appropriate signal paths and corresponding pins to drive/read each PTB device. Note that some signal paths will require jumpers be located in specific locations, etc. Create a spreadsheet or a "table" that lists the mapping between the PTB device signal names and the STM32L452 pins. Notice that the "alternate function" of a STM32L452 pin may be the ideal usage. Have your newly created design reference handy while designing.

Tip from the trenches
On my past projects, I often print the pin-peripheral mapping table on large format paper and post it on the wall just above my computer monitor. I usually print the enlarged project schematic and display it in a similar manner. I never want to have to go digging for important information as I develop.

5.1.2 Configuration

Using your interface board (if required), determine the proper connection from the ARM-Cortex processor on the Nucleo to the pushbuttons and LEDs on the PTB. Further determine the method by which you can fully utilize the LEDs and pushbuttons. For this milestone, determine how to allow your ARM-Cortex MCU to properly read[2] the state of the pushbuttons[3]. Determine the proper setup (pull-ups, direction, speed, logic thresholds, etc.) of GPIO for correct and efficient use of each LED and pushbutton.

Do NOT interface with other PTB peripherals at this time, other than to disable them or turn them off as needed.

5.2 Software

5.2.1 Repo preparation

1. Using the tree structure and the repository location provided by the TA or Dr. Bruce, start populating your team repo.
2. Configure your client to access and manage your github repo.
3. Maintain all code and documentation for the remainder of your lab in this repo. The TA will provide instructions for each milestone as to the location of the files.

5.2.2 Software preparation

Develop an appropriate header file[4] called `ptb.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 with other macro names in the distro.

[2] Did I hear a faint voice in the distance say "debounce"?

[3] There are numerous pushbuttons and LEDs on the PTB. Some are accessed in the same way, and others differ. Plan your design to have common code perform similar operations whenever possible.

[4] You can expect this header file to grow longer and more full-featured as the semester progresses.

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, definitions, and/or functions to manage and fully utilize[5] the following components:

- LEDs 0-3 on PTB, and LD2 on the Nucleo
- pushbuttons SW2-SW5 on PTB, KEY0-KEY15 on the PTB, and the user PB1 on the Nucleo

In an earlier milestone, you created macros for the Nucleo-based hardware. Create (or plan on creating) similar macros for each of the PTB-based pushbuttons and LEDs. If immediate access to the data in these macros is not possible, the macros may only access a data structure which is maintained by some other running code you have developed.

```
TURN_ON_LEDx()
TURN_OFF_LEDx()
TOGGLE_LEDx()
IS_LEDx_SET()
IS_LEDx_RESET()
IS_SWx_PRESSED()
IS_SWx_RELEASED()
IS_KEYx_PRESSED()
IS_KEYx_RELEASED()
```

Your design may consider the Nucleo user pushbutton B1 to be “SW1” and Nucleo user LED LD2 to be “LED4”, if convenient.

5.2.3 ESOS Application

Using the ESOS application your team created in Milestone #4, create a new ESOS program that continues to process UART data from your laptop. In this milestone, you should NOT use any interrupts directly as the application programmer. In this milestone’s ESOS application, the functionality is modified slightly to have the following operational modes:

PTB pushbutton	function
B1 (SW1)	accepts only keyboard commands
SW2	removes any non-alphanumeric character
SW3	converts lower-case to upper-case
SW4	encrypts with key “TENNESSEETECH”
SW5	decrypts with key “TENNESSEETECH”
none	echo

When no switch SW2-SW5 is depressed, the application will simply echo the UART stream with no changes. If any switch SW2-SW5 is depressed, the application will “modify” the UART stream as given in the table. In the event of multiple switches being depressed, the lower-number switches have priority, *i.e.* B1/SW1 is the highest priority and SW5 is the lowest.

[5] Initialization, use, and shutdown, as required

When B1 is depressed, the application will only accept the keyboard commands to change the LED timing similar to previous milestone. However, in this application the user can control and inquire about the flashing of five LEDs: LED0-LED3 (on the PTB) and LD1 (on the Nucleo). Obviously, the keyboard commands need to change. "S*x*,*nnnn*" will set the LED*x* to have flashing period *nnnn* ms at a 50% duty cycle. The keyboard command "L*x*" will return the flash period of LED*x* with four decimal digits denoted the period in ms. Keyboard commands are not interpreted in the none and SW2-SW5 operating modes; these modes simply modify the UART data stream as given in the table.

Tip from the Trenches:
With our object-oriented design paradigm and the ability to create concurrent tasks, you encouraged to create modular and focused tasks to acheive your results. Inter-task communication is typically limited to global variable/structures or messaging. Tasks should not be large and should not perform unrelated operations. The LEDs are user interface and only approximate timing accuracy is required. One approach would be to create a single task per LED to manage each LED. Another approach would be to create single task that manages all LED operations.

An addition command "R*hhhh*" where "*hhhh*" is four hex symbols will return the pressed state of keypad switches KEY0-KEY15. The hex symbols compose a "mask" where a one in the mask denotes a request to know the state of the button corresponding to that bit location. KEY0 would be the LSb. KEYA is the the 10th bit. KEYD is the 13th bit. KEY# is the 14th and KEY* is the 15th and MSb in the mask. For example, "RFFFF" request would return a four symbol hex string denoting the state of which keypad buttons are pressed. Request "R0001" would result in a four hex symbol response with only the LSb having the possibility of being one depending on the state of KEY0.

5.3 Bonus Application

Individual team member(s) or your team can earn a milestone bonus. For this milestone, the bonus application is a PC-side GUI application using PyQt to implement a visual representation and control of the PTB LEDs and switches and UART processing using the VCP protocol described above. The code running on the ARM-Cortex must be independent and unchanged whether the PC is using a terminal application or GUI implementation. Consult with Dr. Bruce for requirments of the PC application. Bonus point will be awarded only if the underlying team application works correctly.

If a bonus is submitted, then the submission needs to include a roster of the team member(s) who participated and contributed to the bonus.

5.4 Check Off

Demonstrate the following things to the TA:

- Using your hardware, demonstrate the proper operation of your ability to read the PTB switches and control the PTB LEDs via a serial communications terminal on your PC.

5.5 Submission

Create a folder in your team repo named `m5` to hold this milestone's deliverables. In the milestone folder, store your application code `app.c` along with your header file `ptb.h`. Include the appropriate `makefile` tand any unit/system test files created. Also, create a `readme.txt` to explain how to build your application[6] and simple user instructions. In the milestone folder, also include the following items:

- well written, well commented, MCU code[7] for ARM-Cortex application
- any unit test or test bench files used to verify your code created for this milestone
- bonus GUI application, if applicable
- updated design review forms and software metrics

Email the TA on or before the specified due date to "submit" your deliverables. The TA will refresh his/her local copy of your tree, build the applicaiton code(s) from your source, and run the application on his/her hardware. The TA will not modify code, adjust values or defaults, or in any way modify your code. If your tree does not build "out of the box", then your team's submissions will be graded as "not working".

When your team has completed the milestone and submitted your work to your team repository, each team member must complete the team evaluation feedback per the instructions from Dr. Bruce or the TA.

[6] The TA should be able to simply "pull" from your team repository, and type `make` to fully build everything in your team submission.

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