Milestone 8

Seven-segment Display Service

"We must all hang together, or most assuredly we shall all hang separately."
– Benjamin Franklin (1705-1790)

In this milestone, you will apply all you have learn to design and implement an entirely new service for ESOS: a seven-segment display service. Your service will involve hardware and software initialization, a hidden housekeeping task, and user API function. Finally, you will demonstrate your new service by augmenting your ongoing user application.

8.1 ESOS Seven-segment display service

Seven-segment displays are a low-cost user interface device. Your Edubase-V2 target board contains four seven-segment displays. Of course, seven-segment displays can easily present numbers to the user. With a bit of imagination, seven-segment displays can display many letters too. You will create the hardware-independent and hardware-dependent (STM32L4 plus EdubaseV2) code for your new service in this milestone.

8.1.1 Initialization

Following the format of the existing ESOS services, you will need to create four files. For the purposes of this milestone, keep these four files in your milestone source folder. Do NOT add the files for this milestone to your ESOS tree.

esos_7seg.c contains the hardware-independent code for the seven-segment display service

esos_7seg.h contains the macros and service variables for the seven-segment display service

esos_stm32l4_edub_7seg.c contains the hardware-dependent code specific to our processor and target board

esos_stm32l4_edub_7seg.h header file for the above

Initialization code in the hardware-independent code will initialize the service data structures and ultimately call the hardware-specific initialization routines `__esos_7seg_hw_config()`. This routine will be specific to the SMT32L4 and Edubase target board.

8.1.2 Hidden service task

Seven-segment displays are GPIO-intensive. Most stystems that use seven-segment displays share pins for the individual displays and only illuminate one display at a time. Our Edubase board is no different. Some "task" must constantly be redrawing the contents of each display repeatedly and quickly for the human user to see four simultaneously illuminated digits. This job will be done by a hidden service task. (See the ESOS services for LCD and UI for an example.) Furthermore, our display is limited to four digits/characters. The hidden task will also coordinate the systematic display of messages longer than four digits/characters. Messages destined for the seven-segment display must be scrolled with a suitable pause between repeated scrollings. This form of display is often called "marquee". Choose your parameters for scrolling to be "smooth" and "pleasing to the eye".

Create a hidden ESOS (user) task for your seven-segment service `__esos_7seg_service`. This service will periodically scan the data structures created by the service and illuminate the appropriate segments on the displays. The task will do this for each digit/character, in turn, rapidly enough that the display appears to be constant. You must determine the appropriate duration for each digit display to balance between loading on the CPU and flickering observed by the human eye. Your hidden user task will also need to control any scrolling display functions, if necessary.

8.1.3 Seven-segment display service API

To provide user applications with the ability to display data on the seven-segment display, your hardware-independent code must provide a programming API. Provide the following API functions in your `esos_7seg.c` file:

- A function `esos_7seg_clearScreen(void)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will blank the display at its earliest opportunity
- A function `esos_7seg_writeCharacter(u8_column, u8_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `u8_data` in the `u8_column` at its earliest opportunity. It may be that you can not display all characters on the seven-segment display. Determine what your service will do in these circumstances.
- A function `esos_7seg_writeBuffer(pu8_data, u8_bufflen)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display the first `u8_bufflen` characters of `pu8_data` at its earliest opportunity. If the buffer exceeds the number of available character displays, the hidden task must "scroll" the buffer message repeatedly until directed otherwise.
- A function `esos_7seg_getDisplay(void)` will return the a string containing what is currently displayed on the seven-segment displays
- A function `esos_7seg_writeString(psz_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display the zero-terminated string `psz_data` at its earliest opportunity. If the string exceeds the number of available character displays, the hidden task must "scroll" the buffer message repeatedly until directed otherwise.

- A function `esos_7seg_blick(bState)` with a TRUE argument will instruct the ESOS seven-segment services hidden task to rapidly blink the contents of the seven-segment displays. A FALSE argument will cause the displays to be constantly illuminated.
- A function `esos_7seg_writeU8Decimal(u8_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `u8_data` as a decimal number (right-justified) on the display.
- A function `esos_7seg_writeU8Hex(u8_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `u8_data` as a hexadecimal number (right-justified) on the display.
- A function `esos_7seg_writeI8Decimal(i8_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `i8_data` as a decimal number (right-justified) on the display. A negative sign, if needed, will be in the left-most "empty" digit.
- A function `esos_7seg_writeI8Hex(i8_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `i8_data` as a hexadecimal number (right-justified) on the display. A negative sign, if needed, will be in the left-most "empty" digit.
- A function `esos_7seg_writeU16Decimal(u16_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `u8_data` as a decimal number (right-justified) on the display. If the number exceed the capability of the available displays, show "dashes" in every digit location instead.
- A function `esos_7seg_writeU16Hex(u16_data)` will modify the ESOS seven-segment services data structures accordingly so that the hidden task will display character `u8_data` as a hexadecimal number (right-justified) on the display.

8.2 ESOS Seven-segment Service Test Application

Create a test application `m8_test` that will fully exercise all aspects of your newly created ESOS32 seven-segment service. You may use any other resources (serial, LCD character module, LEDs, pushbuttons, etc.) in any way you desire. Be sure that your code is fully documented, maintainable, and modular[1].

8.3 ESOS Application

Use your newly created ESOS32 seven-segment service and the IP from the previous lab milestones to create an application `m8_app` that displays information on the LCD screen and seven-segment displays. All serial communications to/from the EduBase will continue working as specified in Milestone #7, with the following changes:

8.3.1 Work Change Order #1

During LED selection, the LED flash period will also be displayed in decimal on the seven-segment displays.

[1] Careful design of this test app could provide your will some valuable reusable code for the user application in the next section

8.3.2 Work Change Order #2

During LED period edits, the currently entered LED flash period will be also displayed in decimal on the seven-segment displays, including the "underscores".

8.3.3 Work Change Order #3

A running sum of total letters encrypted or decrypted since POR will be maintained. As each letter comes into the system for encryption or decryption, the accumulator is incremented. During the encryption (SW4) and decryption (SW5) modes, the accumulator value will be displayed, in real-time, on the seven-segment displays in decimal. During encryption, the display will be constant. During decryption, the display will flash.

8.4 Check Off

Demonstrate the following things to the TA:

1. Your developed seven-segment service test app
2. Your LCD/seven-segment display ESOS application with proper UI behavor (with a "nice" feel), encryption, decryption, and LED flash control

8.5 Submission

1. Commit your project to your repository to your "trunk" folder in accordance with the procedure prescribed by the TA. Your repository should all files required to produce your "build", including the appropriate build file(s).
2. A "tag" release in your repository called `m8` that can be recalled at any time to build this milestone's deliverables.

Each team should submit to their team repository the following files[2]:

- The well-documented versions of the files `esos_7seg.c`, `esos_7seg.h`, `esos_stml4_edub_7seg.c`, and `esos_stm32l4_edub_7seg.h` created by your team.
- Your team-developed seven-segment service test app and associated support files
- The ESOS32 LED+7-segment+encryption display application `m8_app.c`
- any unit test or test bench files used to verify your code created for this milestone

[2]Files should be possess the same licensing header as the other ESOS32 source code files and be fully commented.