Milestone 7

LCD Character Module and Service

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

IN this milestone, you will add the hardware-specific code for an ESOS32 service to control and use the ubiquitous LCD character modules based on the Hitachi 44780 chip command set. The ESOS32 LCD character module service is buffered to minimize delays to the calling ESOS32 tasks. An LCD character module housekeeping task is created by the service during initialization to manage and update the LCD character module "in the background". Also, you will create a couple of user tasks to exercise the timers on your MCU

The data sheets for the Hitachi 44780 chip set and your specific LCD character module are provided on the course website. Study them thoroughly before beginning.

7.1 ESOS (Write-only) LCD Service

7.1.1 Hardware-Specific Functions and Documentation

Ultimately, the LCD character module service must communicate with the LCD character module hardware via the hardware mechanisms provided by your processor and the target system. To this end, the LCD character module service has a hardware-independent portion in the files `esos_lcd44780wo.c` and `esos_lcd44780wo.h`.

The functions and macros in these files allow for low-level access to the character module at the lowest levels, however knowledge of the hardware specifics is not known by ESOS itself. Each "port" of ESOS32 to a specific processor and hardware target must provide functions, macros, and tasks to perform the hardware-specific actions that are mapped to the generic (not hardware-specific) functions and macros in `esos_lcd44780wo.c` and `esos_lcd44780wo.h`. For our target, these target-specifc files are named `esos_stm32l4_edub_lcd44780wo.c` and `esos_stm32l4_edub_lcd44780wo.h`. These files are based on the templates in `esos_hwxxx_lcd44780wo.c` and `esos_hwxxx_lcd44780wo.h`.

Since your team does not have access to test & measurement equipment like logic analyzers, the specific code to interface with the EduBase-v2 LCD module has been provided to you.owever, the code lacks good documentation. Your job is to understand fully the concept, operations, and code in the `esos_stm32l4_edub_lcd44780wo` files, and augment these files with thorough documentation (using doxygen markup).

The initiatialization of the LCD module occurs in the earliest times of ESOS booting. To support the LCD initializaiton phases, the service provides some "unsafe" functions:

```
_esos_lcd44780_stm32l4_edub_config( )
esos_unsafe_lcd44780_stm32l4_edub_write_u4( )
__esos_lcd44780_stm32l4_edub_hw_write_u8( )
```

7.1.2 Public Functions

User application tasks could theoretically call/spawn the LCD service child tasks directly. However, in order to avoid having to negotiate the "ownership" of the LCD by user tasks, the LCD service uses a "background" LCD service housekeeping task. This LCD44780 service housekeeping task is the only task that will actually talk to the LCD hardware. All user tasks will modify shadow variable memory. These shadow variables are used by the LCD housekeeping task to update the LCD and keep the physical hardware synchronized with the user application's desires. Furthermore, the shadow variables allow most LCD operations become non-blocking as they are only manipulating memory structures instead of waiting on physical hardware to respond.

With the hardware-dependent and private LCD service functions above, a public and hardware-independent ESOS32 task interface to the LCD character module service can be created. Visit the ESOS32 documentation and read through the code to get a sense of the functions available to the application developer.

Initialization and Configuration

The following function completes configuration and initialization:

```
esos_lcd44780_configDisplay()
```

Configures (or reconfigures) the hardware-independent aspect of the LCD character module service. Also initializes LCD service memory buffers[1].

The LCD character module housekeeping task performs the hardware initialization sequence for the Hitachi 44780 devices before the service can begin its infinite loop of processing characters and refreshing the screen. Hitachi 44780 startup sequences are fairly universal, althought different vendor displays sometimes exhibit differing behavior. ESOS will initialize the display using your hardware-access driving functions above.

```
esos_lcd44780_init()
```

which will cause the LCD housekeeping task[2] to execute the LCD initialization (power-up) sequence. See pages 45-46 in the Hitachi 44780 datasheet.

[1]Remember that Hitachi 44780 LCD modules have more DDRAM that visible screen locations. Many applications use this "off-screen" memory for storage purposes. Since your memory buffers are mimicing the real memory on the LCD module, your buffers need to work properly for read and writes to memory locations outside of the visible screen.

[2]This operation must be completed before the LCD service housekeeping task begins normal operation. You must write your LCD service task to ensure this fact.

Operation

Once the LCD service housekeeping task is properly configured and has initialized hardware, it will enter an infinite loop where changes to its memory-based display buffer is efficiently written to hardware as soon as hardware is available. The LCD service housekeeping task creates and maintains any and all structures necessary to maximize the performance and responsiveness to the calling user tasks. Although our EduBase circuit is a write-only interface to the Hitachi 44780, the ESOS LCD service can implement "read-only" functions from its shadow memory versions of LCD data.

Most of following functions are complete. You may suggest code for any incomplete functions to Dr. Bruce for inclusion into the ESOS32 tree. (Of course, any bugs and suggested fixes for the existing code are very much welcome.)

```
esos_lcd44780_clearScreen()
```

Clear LCD screen buffer and move cursor to (0,0)

```
esos_lcd44780_setCursorHome()
```

Move cursor to (0,0) without changing display memory buffer or the display

```
esos_lcd44780_setCursor(u8_row, u8_column )
```

Move cursor to (`u8_row`, `u8_column`) without changing display memory buffer or the display

```
esos_lcd44780_writeChar(u8_row, u8_column, u8_data)
```

Writes the (single) character `u8_data` to the screen location (`u8_row`, `u8_column`)

```
uint8 esos_lcd44780_getChar(u8_row, u8_column)
```

Returns the character currently displayed at screen location (`u8_row`, `u8_column`)

```
esos_lcd44780_writeBuffer( u8_row, u8_column, pu8_data, u8_bufflen)
```

Writes `u8_bufflen` characters from `pu8_data` to the screen location (`u8_row`, `u8_column`)

```
esos_lcd44780_getBuffer( u8_row, u8_column, pu8_data, u8_bufflen)
```

Returns `pu8_data` with the `u8_bufflen` characters currently displayed starting at screen location (`u8_row`, `u8_column`)

```
esos_lcd44780_writeString( u8_row, u8_column, psz_data)
```

Writes the zero-terminated string `psz_data` to the screen location (`u8_row`, `u8_column`)

```
esos_lcd44780_setCursorDisplay( u8_state )
```

Set the cursor display state to the value (TRUE or FALSE) given in `u8_state`

```
uint8 esos_lcd44780_getCursorDisplay( pst_lcdStruct )
```

Returns the cursor display state (TRUE or FALSE)

```
esos_lcd44780_setCursorBlink(u8_state )
```

Set the cursor blink state to the value (TRUE or FALSE) given in `u8_state`

```
uint8 esos_lcd44780_getCursorBlink()
```

Returns the cursor display state (TRUE or FALSE)

```
esos_lcd44780_setDisplayVisible(u8_state)
```

Set the display visible state to the value (TRUE or FALSE) given in `u8_state`

```
uint8 esos_lcd44780_getDisplayVisible( )
```

Returns the current display visible state (TRUE or FALSE)

```
esos_lcd44780_setCustomChar(u8_charSlot, pu8_charData)
```

Sets the custom character memory for the `u8_charSlot` character to the data in `pu8_charData`. Will read either eight or eleven uint8s from `pu8_charData` depending on the LCD character module settings.

```
esos_lcd44780_getCustomChar(u8_charSlot, pu8_charData)
```

Returns with `pu8_charData` filled with the custom character bitmap data for the custom character memory for the `u8_charSlot` character. Will fill either eight or eleven uint8s into `pu8_charData` depending on the LCD character module settings.

Task-Display Synchronization

In general, your user application tasks can write to the LCD service memory buffers using the non-blocking functions above. Although the LCD character module is slow, and made even slower by the serial interface employed on the EduBase, the LCD service housekeeping task runs even slower due to not monopolizing the CPU. The human eye is slower still and the result is a LCD module that appears very responsive.

The actual LCD character display may not – in fact, it will not – always update in the exact order as the LCD operations function (shown above) are called. Usually this is fine as the final state of the LCD display will ultimately be correct. However, it is possible that an user application will want to ensure that the LCD service housekeeping task has fully updated the LCD display hardware before continuing. To this end, the following macro may be called by user application tasks when neeeded:

```
ESOS_TASK_WAIT_ON_LCD44780_REFRESH()
```

Causes the current task to block until the LCD character module service housekeeping task has updated the screen completely.

7.1.3 LCD Service Task

A "hidden" user task for LCD servicing is created. This task can first initialized the LCD module hardware and setup the LCD service software structures. This task also manage the LCD character module for ESOS32 and allow for faster response to the tasks using the service. The LCD service hidden task maintains a buffer containing the LCD character display memories. Therefore, ESOS32 applications that read/write to the LCD character module do not have to wait for hardware accesses to complete. Instead, ESOS32 applications using the LCD character module service will be, in reality, making read requests from and write requests to the memory buffer. The "hidden" LCD module task is constantly running in the background like any other ESOS32 task to keep the LCD hardware in agreement with the display buffer. The LCD service task keeps track of what locations on-screen need to be updated. It is inefficient to redraw/refresh the entire screen when only one or a few characters have changed.

7.1.4 LCD Service Test App

To verify the integrity of your system and demonstrate your mastery of the LCD services, create a test application that exercises several (many?) functions in the ESOS LCD service. Properly document your code with doxygen markup. Provide a test app README file that explains the proper operation. You may use pushbuttons, LEDs and other resources on the EduBase, if you wish.

7.2 ESOS Application

Use your newly created ESOS32 LCD character module service and the IP from the previous lab milestones to create an application that displays information on the LCD screen. All serial communications to/from the EduBase will continue working as specified in Milestone #6, with the following additions:

When the user presses SW4 or SW5, the LCD display will clear. While SW4 or SW5 is pressed, the LCD display will show the incoming text on the first line and the corresponding outgoing text on the second line. The newest text will displayed in the right-most column with older characters displayed in the columns to the left. Figure 7.1 shows the LCD display for encryption when the user has sent "ECE rocks". Upon release of SW4 or SW5, respectively, the LCD display will return to the "default" LCD display for LED selection shown below.

Figure 7.1: LCD screen for text encryption

```
        ECEROCKS
        XGRESUCW
```

The application in this milestone will provide a Edubase user-interface with LCD module feedback for adjusting the LED flashing periods. The Edubase user interface will choose among the LEDs by pressing the keypad "A" key and the keybpad "B" key. These two keys will act as "cursor" buttons on the EduBase, and will, in turn, select the controllable LEDs (LED0-LED3, and LD2). Upon reaching the end of the LED list, an additional click in the same direction will NOT change the screen.Each LED's current state will be denoted on the screen as shown below. The "arrows" on the screen denote that there is another LED "above" and "below" the currently listed LED. If the displayed LED is the first or last in the list, the up and down arrows will disappear, respectively. Figure 7.2 shows the default screen for displaying and selecting the LED flash period.

Figure 7.2: LCD screen for LED display

```
▴LED3    1250 ms
▾press D to edit
```

Upon pressing the keypad "D" key, the LCD screen will change into edit mode and all four digits of the flash period will change to underscores. The period is entered with the numbers on the keypad. Each digit pressed will appear in the "ones" position with the earlier digits moving leftward. Users can enter more than four digits, but only the four most recent digital will be displayed and "used" in the LED period. Pressing the keypad "*" key will accept the currently display period. Figure 7.3 shows the screen for editing a LED flash period after LED3 has been chosen for edit, and the user has pressed the "4", then the "1" keys on the keypad.

Figure 7.3: LCD screen for LED period edit

```
 LED3    __41 ms
   * to apply
```

Changes to the LED's flash period will take effect immediately upon pressing the "*" key, and the LCD will return to the display in Figure 7.2.

7.3 Check Off

Demonstrate the following things to the TA:

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

7.4 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 `m7` that can be recalled at any time to build this milestone's deliverables.

Each team should submit to their team repository the following files[3]:

- The well-documented versions of the files `esos_stm32l4_edub_lcd44780wo.c` and `esos_stm32l4_edu` created by your team.
- Your team-developed LCD service test app and associated support files
- The ESOS32 LED+encryption display application `m7_app.c`
- any unit test or test bench files used to verify your code created for this milestone

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