ECE 2534 – Spring 2018 – Lab 2
The R2D2 Terminal – DESIGN DOCUMENT

This lab is an individual assignment.

Introduction

This document provides additional guidelines to solve the lab.
- It describes details regarding use and documentation of the LCD.
- It describes details on configuring and using MobaXterm.
- It describes where to find UART-related settings.
- It describes details on the recommended hardware abstraction layer (HAL).
- It describes the recommended design steps to build the complete solution.

Using the LCD Display

The LCD is a 128 by 128 graphics pixel display (Crystalfontz CFAF128128B-0145T). It comes with a driver library that provides low-level access to the display from the MSP432P4. The Crystalfonz driver is included in the started code you download for this lab.
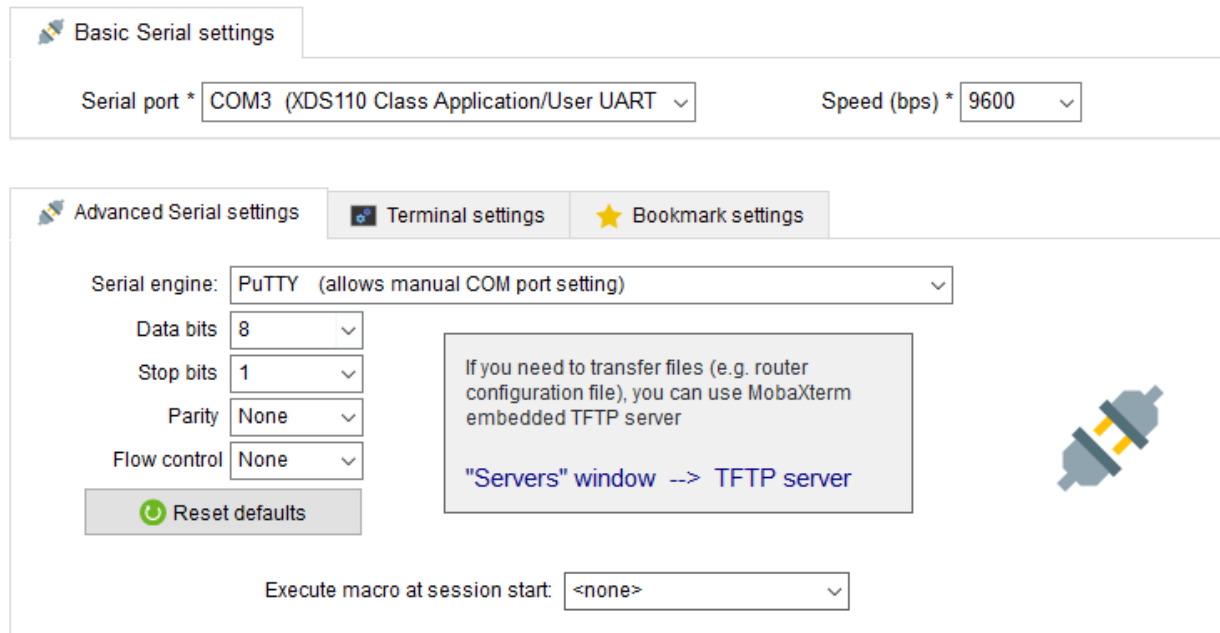
Texas Instruments has developed a generic graphics library, MSP-GRLIB, on top of the Crystalfonz library. MSP-GRLIB is not a part of the driverlib which we use to access other peripherals. Therefore, the graphics library functions are documented in a separate manual, accessible on the web (https://goo.gl/B6jASV). To use the LCD, you will have to consult this manual. In particular, the Context API Section of the GRLIB manual describes how to clear the display, and the String API shows how to display characters.

MobaXterm

The MobaXterm terminal (available from https://mobaxterm.mobatek.net/) will connect to the USB-UART and display the received characters in a window. When you open a session, select 'Serial' to open a serial connection:

Next, select the proper USB Serial Port and the proper baud rate

If you download the starter code, you should see 'Hello!' when you run the application on the MSP432P4. If you type characters on the keyboard of your laptop, they will be echoed in the MobaXterm window.

Baud Rates and other UART settings

This lab needs to use the following UART format: 8 data bits, one stop bit, no parity bits, no flow control. The application of the MSP432P4 needs to support four baud rates: 9600 baud, 19200 baud, 38400 baud and 57600 baud. You will have to compute the settings of the UART peripheral to set the proper baud rate. Consult Section 24.3.9 of the MSP432 User Guide for a description of the UART Baud Rate Generation process. Section 24.3.10 will be especially important, as it describes how to set the baud rate divider hardware of the UART peripheral.

Recommended Hardware Abstraction Layer

We strongly recommend developing a *Hardware Abstraction Layer* (HAL) for your application. A HAL is a set of functions that hide the details of the hardware. We give an example, taken from the starter code.

The software interface to the red LED is captured by the following two functions:

```
void InitRedLED() {
    GPIO_setAsOutputPin    (GPIO_PORT_P1,GPIO_PIN0);
    GPIO_setOutputLowOnPin (GPIO_PORT_P1,GPIO_PIN0);
}
```

```
void RedLEDToggle() {
    GPIO_toggleOutputOnPin(GPIO_PORT_P1,GPIO_PIN0);
}
```

The first function, `InitRedLED()`, will be called upon program initialization. The second function, `RedLEDToggle()`, will be called to toggle the red LED. Note that the HAL functions are written regarding driver library functions. They call GPIO related functions. They also deal with the precise port number and port pin related to the red LED.

The development of a HAL will help you to separate low-level hardware details from high-level application design. They make your program easier to understand. Indeed, just take a look at the cyclic executive of the starter code:

```
while (1) {
    if (UARTHasChar()) {
        c = UARTGetChar();
        RedLEDToggle();
        UARTPutChar(c);
    }
}
```

Thanks to the HAL functions, the program almost explains itself. There are no unneeded details, and as a result, it becomes much easier to see what is going on.

The following is a list of functions that constitute the HAL for Lab 2. You are not required to implement exactly those functions. We recommend the design of a HAL because it will make your application easier. We only provide the function headers and a short description for each function. It's up to you to define the detailed implementation if you decide to use this function.

- Graphics
    - `void InitGraphics()`
      Initializes the LCD and graphics library
    - `void LCDClearDisplay()`
      Clears the LCD to the current background color
    - `void LCDDrawChar(unsigned row, unsigned col, int8_t c)`
      Draws a character c at position row, col
    - `void LCDSetFgColor(color_t c)`
      Sets the foreground color to c
    - `void LCDSetBgColor(color_t c)`
      Sets the background color to c
- UART
    - `void InitUART()`
      Initializes the UART
    - `bool UARTHasChar()`
      Returns true if the UART peripheral has received a character
    - `uint8_t UARTGetChar()`

Returns the most recent received character from the UART peripheral
- o `bool UARTCanSend()`
  Returns true of the UART transmitter register is empty
- o `void UARTPutChar(uint8_t t)`
  Transmits character t using the UART, if the UART can send
- o `void UARTSetBaud(UARTBaudRate_t t)`
  Changes the baud rate to t
- Buttons
  - o `void InitButtonS1()`
    Initializes button S1
  - o `int ButtonS1Pressed()`
    Returns 1 if button S1 is pressed
  - o `void InitButtonS2()`
    Initializes button S2
  - o `int ButtonS2Pressed()`
    Returns one if button S2 is pressed
- LEDs
  - o `void InitLEDS()`
    Initializes the LEDs
  - o `void ColorLEDSet(color_t t)`
    Set the Color LED to color t
  - o `void RedLEDToggle()`
    Toggles the red LED
  - o `void RedLEDOn()`
    Turns the red LED on
  - o `void RedLEDOff()`
    Turns the red LED off
- Timer
  - o `void InitTimer()`
    Initializes the timers. In this lab, we need two times, one for Button Debouncing, and a second for the Color LED flashing. We can use TIMER32_0_BASE and TIMER32_1_BASE for these two functions.
  - o `void Timer200msStartOneShot()`
    Starts a timer for a 200ms one-shot
  - o `int Timer200msExpiredOneShot()`
    Returns 1 if the 200ms one shot timer has expired
  - o `void TimerDebounceStartOneShot()`
    Starts a timer for a debouncing delay
  - o `int TimerDebounceExpiredOneShot()`
    Returns 1 if the debouncing delay timer has expired

How to build your solution

You can solve everything in this lab using only the techniques that we have discussed in the lecture. However, to be successful, you will have to work step-by-step and test your solution as you go forward. The following steps explain a possible strategy which starts with the easy problems, and

then gradually moves to harder problems. You are not required to follow it, but if you are unsure how to start the lab, then follow the steps below. The most important insight in the guidelines below is that you will <u>gradually</u> develop Lab 2 from the starter code, and that you maintain <u>a working application</u> at all times. You can take a very big step as a sequence of many small steps!

1. **Study the echo application**. The starter code for Lab 2 does something very useful: it configures the UART and receives/send characters to it. So carefully studying the starter code is the first thing you should do. In particular, make sure you understand what the following functions do `InitUART()`, `UARTHasChar()`, `UARTGetChar()`, `UARTCanSend()`, `UARTPutChar(uint8_t t)`. To learn what the `UART_` functions do, read the driverlib user manual.

2. **Display characters on the LCD screen.** Next, turn your attention to the LCD screen, and find out how to display characters on it.   s

commands, which each have their implementation: #f changes the foreground color of the display, while #b changes the background color.

To implement this functionality, you have to design a Finite State Machine. The finite state machine parses one character at a time. When a command sequence is found (starting with '#'), the following characters will not be printed until the specific command is known. Not every sequence starting with '#' is a command! For example, '#zoro' is a string that should be printed on the LCD.

b. Design the parsing of commands (#fn and #bn) as a finite state machine that is fed characters received in the main cyclic executive. So the template looks similar to the following.

```c
// FSM
void processChar(uint8_t c) {
    typedef enum {idle, .. } state_t;
    static state_t S = idle;

    switch (S) {
    case idle:
      if (c == '#')
          S = ... (next state);
      else
          display (c);
      break;
    ...
}

// cyclic executive
void main() {
   ...
   while (1) {
      if (UARTHasChar()) {
        uint8_t c = UARTGetChar();
        processChar(c);
      }
      ...
   }
}
```

c. Since the #fx and #bx commands use three characters, the FSM will need to take three steps to parse the full sequence: first, receive #, then receive f or b, then receive a number between 0 and 7. If, at any point, you run into an unsupported inline command (such as '#ft'), then simply display the characters you have parsed so far (in the case of the example, '#,' 'f' and 't').

5. **Buttons to set baud rate.**
   a. To change the baud rate, you have to change the divider values for the UART. Study the example for 9600 baud and find how to set the baud rate for other speeds. Create a function `void UARTSetBaud(UARTBaudRate_t t)` which helps you set the baud rate from anywhere in the application.
   b. Test your baud rate settings by temporarily changing the default baud rate in the main program. If you do this, you have to close the current MobaXterm window and open a new one with the correct baud rate. It's convenient to create shortcuts in MobaXterm that allow you to open a COM3 terminal with a selected baud rate of 9600, 19200, 38400 or 57600 baud.
   c. Design a debouncing FSM for button S2 and test it. If needed, consult the example on button debouncing at https://github.com/vt-ece2534-s18/msp432-button-debounce.
   d. Finally, integrate the baud rate change into the button-down event of button S2.

6. **BONUS: Configure and Use Color LED**. Your application has to use the Color LED of the sensor board. Therefore, write a few functions that help you use the Color LED.
   a. First, find out how it's connected. Trace the connector pins of the sensor board to the Launchpad board and find out what Port and Port Pins are used for the sensor board color LED. Then, write an `InitColorLED()` function and an `ColorLEDSet(color_t t)` function.
   b. Integrate your new functions in the echo application. For example, toggle the color LED for each received character.
   c. Design a one-shot timer to measure precisely 200ms. Design a Timer API with functions such as `InitTimer()`, `Timer200msStartOneShot()` and `Timer200msExpiredOneShot()`. Then, instead of toggling the color LED, turn the Color LED on for each received character, and turn it off again after 200ms. This looks daunting at first, but it's a straightforward application of the cyclic executive.

The previous six steps do not disclose all the details of the solution, and you still have to consult the manuals and guides that come with driverlib, graphics library, MSP432 User Guide, and the board User Guides. Finding the correct information, and finding out how to use it, is part of your work to solve this lab.

Programming Notes

- Your code should be well documented. If you are not sure what level of documentation is enough, please ask your GTAs or instructors.
- You should use #define, constants, and an enum to hardcode the numbers related to the application. It is not allowed to use meaningless constants inside of your code.
- Avoid long chain of if-then-else statements. Use switch-case when appropriate.