EE 472 Lab 2

Winter 2015

I/O and Peripherals

University of Washington - Department of Electrical Engineering

Mitch Ishihara, Joshua Fromm, Tien Lee

Lab Objectives:

This lab is the first phase in the development of the RoboTank control system. This phase will focus on developing the infrastructure required for the Stellaris board to communicate with and utilize a keypad and LCD display.

The deliverables include a full specification of the user interface composed of the display and keypad that will be implemented. Because quite a bit of the design will be left to you, it is critical that you convey exactly how your system operates. This means that you will need to document not just the flow of the user interface, but also the connections between your processor and the peripherals as well as the interaction of key pieces of software.

The system implemented this week must be capable of capturing user input through a built-in 5 key keypad and using that information to manipulate a menu on the provided LCD display. In developing this system we will,

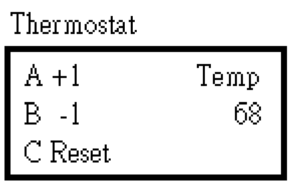
1. Introduce formal specifications,
2. Learn how to set up and use the GPIO pins of a microcontroller,
3. Write functions that get and debounce simple key presses,
4. Wire the connection between GPIO and an LCD display,
5. Develop a function that satisfies the timing requirements of an LCD display,
6. Create a suite of functions that allow software interaction with an LCD display,
7. Develop the infrastructure to allow key presses to interact with the display,
8. Create a full user interface that adheres to specifications.

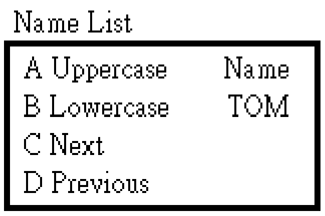
User Interface Specifications:

In this lab you will be developing the user interface for RoboTank system. There is a set of base requirements that each user interface must fulfill, however you are encouraged to add more features and if they’re cool or challenging you will be awarded additional points. The required features for the interface are:

- Input from 5 distinct keys.
- A main menu that prints on the LCD display that contains at least a commands sub menu and a status option.
- When selected, the commands menu must display at least some strings as commands like “Go”, “Left”, and “Right”. The user should be able to use the keys to navigate to a command and select it. When a command is selected, string like “OK” should be displayed to show current status.
- A detailed description of the function of each key and menu option must be provided.
- Note that these requirements are intentionally vague, you are encouraged to come up with creative solutions and implementations so long as they satisfy the specifications provided.

- Some Examples





- Possible Bonus:
  - Make a longer menu and use keypad to navigate through it
  - Use more than the LCD as output

Step 1: Blinking an LED

To accomplish any of the goals in this lab, it is necessary to first understand how to use the GPIO pins provided on the Stellaris board. To learn the basics of software control of GPIOs, we will make a simple program that blinks an LED on the board.

For this step, it will be critical to examine the datasheets included in the homework zip, specifically Stellaris.pdf and LM3S8962.pdf. It is recommended that you read Section 5 (system control) and section 8 (GPIO) of LM3S8962.pdf before starting. Use of the Class Discussion Board is also highly encouraged.

1.0 Setting up

To begin, create a new project using the same procedure learned in lab 1. Once the project is made, create a new blank c file and create a main loop. The main loop needs to first set the clock speed to run directly from the on board crystal. This can be done by either directly writing the SysCtl RCC and RCC2 registers (See the LM3S8962 datasheet) or using the function SysCtlClockSet function used in lab 1.

1.1 Blinking the LED

We now will write a function that blinks the onboard status LED (LED1). First, go to the Stellaris evaluation datasheet and find which GPIO pin controls LED1. Once the pin is determined, we need to set up that pin to be an output. Create a new function called LED_init that does the following **in order.** Be careful not to change the settings of any pin that you are not using. Hint: what bit manipulation process and considerations are required to do this?

1. Set SysCtl Run Mode Clock Gating Control Register 2 to enable the GPIO port for LED1.
2. Set the direction of the pin controlling LED1 to be output.
3. Disable alternate functionality of the pin using the GPIO Alternate Function Select Register.
4. Enable the pin in the GPIO Digital Enable Register.
5. Set the output of the pin to 1, turning the LED on.

In your main loop, simply call LED_init to confirm that the status LED does indeed turn on. If it does not, you have not properly set up the registers.

Next, create function called LED_toggle. This function takes no arguments and simply toggles the state of LED1. Then, use your main loop and a delay function to make your LED blink at a rate of your choosing.

Step 2: Keypad Input

2.0 Basic Functions

Next, we will develop functions that detect and handle when a key is pressed on the Stellaris Evaluation Board. First, use the data sheet to determine which pins connect to each of the 5 on board switches (up, down, left, right, and select).

Create a new file called keypad.c and write a function called key_init that sets up the control registers for the key pins similar to how the LED was initialized previously, noting that the pins must be inputs instead of outputs (since we are reading the key values). Additionally you must set the pins to be pulled up internally using the GPIO_PUR register. This is needed because the switches on the board are not tied to a discrete resistor.

Continue by writing a function named is_a_key that returns a Boolean value indicating whether ANY key is pressed.

Set breakpoints to verify that these two functions work as intended, specifically, set a breakpoint after is_a_key is called in your main loop (which should be in a separate file) and confirm that the return value is true when you hold down a key and false when you do not.

Continue by writing a function called getkey that returns an integer value indicating which key was pressed. The value corresponding to each key is not important so long as you are consistent and document the behavior of the function well.

Finally, write a function called keymaster that returns the value of a key if one is pressed and 0 if no key is pressed. Keymaster should use only calls to is_a_key and getkey rather than checking register values itself. Modify your main loop so that each time the SELECT key is pressed, the LED toggles. Run your code and observe the behavior.

2.1 Debouncing

When running your key code, you've probably noticed that the LED does not always behave how you expect. Specifically, pressing SELECT sometimes does not toggle the LED. Additionally, you may notice that holding down a key causes the LED to be less bright. We will address both of these issues by adding new functionality to our key code.

The first issue occurs because switches tend to bounce when they are pressed. When the switch is pushed down it actually takes a little time before the value stabilizes. Until the value is stable, the output of is_a_key will fluctuate between true and false many times for a single press, making it appear as if the key was pressed many times in quick succession. Modify your code by adding a function called debounce that only returns true if the same key value is read continuously for 100 milliseconds. You may do this by creating a delay loop that checks the key values and only returns true once the loop finished, but exits and returns false if the value changes before timing out. Add a call to debounce from keymaster so that keymaster only returns a non-zero value for a successfully debounced key. Note that debounce should call getkey to check key values.

The other issue encountered is that your code has no way of telling if a press is new or not. When a key is held down, the software can interpret it as having been repressed. Write a new function called fresh_key that returns true if a key press is not a repeat from a previous press. Note that this will require creating shared variables in your file. A shared variable is one defined outside the scope of a function, so that all functions in the file can access them. You'll need a variable that keeps track of the last successfully debounced key press, make sure the variable is cleared when the key is released! Add a call to fresh_key to the keymaster function.

Run your code again and confirm that each key press toggles the LED only once and is consistent.

Step 3: LCD Display

Next, we will develop the software and hardware infrastructure needed to put characters onto the LCD display of the Stellaris board. For this section, you will need to be comfortable with the Optrex Manual included with the lab. Start by figuring out what each pin of LCD does (note we are only interested in the first 11 pins, pos and neg aren't important).

The display requires an input of 8 data pins along with 3 control pins. We will use GPIO for each of these pins. The data pins will be driven by Port B0-6. Since we actually don't need the high bit (D7 on the display) simply connect this line to the board's ground. Make these connections. Next connect the Port D5-7 pins of GPIO to the control lines of the display (E, R/W, and RS). Which GPIO pins are used as control are up to you, just be consistent.

3.0 Display Timing

Because we are setting up a peripheral completely from scratch, there's a number of things we have to do. First it is important to note that LCD displays have very specific start up sequences. Look at page 32 of the included OptrexMan.pdf. The diagram shown details the steps required to properly turn on the display, including required waits. Create a new file called display.c and implement a function called display_init that first **sets up the GPIO registers needed** and then goes through the LCD 8-bit initialization process, which will basically be outputting a value to GPIO then calling a delay function. When this function is called from your main loop, a cursor on the display should begin to blink if you set the B and C bit options in the final step of initialization.

We will continue by writing a function that writes a single value to the display. Go to page 47 of the Optrex Manual. Here you will find the timing characteristics of the display. What is shown is the relationships between when signals are expected to be valid. What we can see here is that RS and R/W should be set T_{AS} seconds before E is set. Similar, timing for each signal is given and on page 49 you can find the values of each delay. Create a function called display_write that takes a char as input and the desired value for RS (1 or 0) and writes it to the display. This will be done by setting bits and delaying to satisfy the timing diagram. Confirm that the function works by calling it once for a single character, you should see it appear on the screen. Note that for writing to the display, RS should be 1.

Write a function called display_string that takes a (char *) argument and puts the corresponding string on the display by iterating through characters and calling display_write.

3.1 Advanced Display

Next, go to page 34 of the Optrex Manual. Here you can find functions that allow more advanced control of the display. Functions that you will need to implement the user interface for this lab are: clear display, Cursor or Display Shift, Set DD RAM Address, and Write DD RAM. Implement functions for each of these commands, each taking appropriate arguments.

Using your set of functions, determine how you want to display the menus required for the lab. Because you may not know what you want at first, it is recommended you iteratively design the interface before attempting to code it. Include a drawing of the final user interface, a table indicating what each key / menu option does, and include it in your report.

Step 4: Put it Together

Now that we have all the tools we need, modify your main loop to call functions that:

1. Initialize the keys and display,
2. Put an initial menu screen on the display,
3. Wait for and debounce key presses,
4. Control and write to the display so that it conforms to the specifications in section 1.

How you go about implementing the menu (and quite a bit of this lab) is intentionally left open ended. You are encouraged to start simple, first getting it to work. Then iteratively expand your creativity. Extra effort and cool features will be rewarded as innovation credits.

Grading Guidelines

You are expected to demo your user interface to a TA, showing that it does all the required features along with anything extra you coded. Be prepared to explain how the software or hardware components used in the lab works as well as any issues encountered and solutions to the issues you encountered.

You will need to turn in a report comprised of the content described in this lab. Clearly indicate the contributions of each team member in the report.. As always, you should also turn in your **thoroughly** documented source code.