CE-2812, Lab #2, Knight Rider Lights

1 Purpose

The purpose of this lab is to write a complete simple application that interacts with STM32F4 peripherals.

2 Prerequisites

- The Nucleo-F446RE board had been mounted onto the Computer Engineering Development board.

3 Activities

3.1 Project Requirements

Write a program that sweeps the LED lights on the dev board back and forth similar to KITT in TV's Knight Rider. For those of you who are so unfortunate to be unaware of KITT, be sure to watch the show's into: https://www.youtube.com/watch?v=oNyXYPhnUIs.

While we will want to structure our code for reuse, let's focus on functionality this time around and restructure later if needed.

As a first step, create a new project from the template project you set up last week. We will not necessarily need the serial console for this project, but it does not hurt to have it ready to go if we want to use it to print debugging messages or the like.

For simplicity, I suggest implementing this entire project in a single source file.

3.1.1 Delay Subroutine

We will need a delay subroutine to slow down the action a bit. As discussed in lecture, we cannot easily use instruction cycle counts to create a delay. Instead, we will use a timer peripheral, namely, the SysTick timer (STK). This timer is best documented In the STM32F4 programming manual, PM0214.

Create a subroutine **`void delay_ms(XXX theDelay)`** that will use the SysTick timer to effect the requested delay in milliseconds. As a void function, it does not return a value. In the prototype, XXX represents the type of the parameter and it is left up to you to decide what type it should be.

You may include this routine within your main 'C' source file, or a separate file if you wish. If it is in a separate file, you will need a header file as well. Be sure to use defined symbols (**#define**) where appropriate.

3.1.2 LED Driver Subroutine

While we can achieve the Knight Rider effect by bit shifting, we may also benefit from a generic subroutine that simply lights up the LEDs based on a number and the LED will essentially display that number in binary. Since we have 10 LEDs, a number between 1 and 1023 will light up 1 to all of the LEDs. In other words, the parameter supplied will be displayed in binary on the LEDs. No "conversion" to binary is needed – a number is a number…

Create a subroutine **`void light_LED(XXX number)`** that will use the parameter, number, to determine which LEDs to light. In the prototype, XXX represents the type of the parameter, and it is up you to determine what actual type to use.

As an example, the following should be the behavior of this routine:

light_LED(0); ➔ 

light_LED(1); ➔ 

light_LED(2); ➔ 

light_LED(3); ➔ 

light_LED(4); ➔ 

light_LED(1023); ➔ 

Your light_LED routine should use various operators (masks and shifts) to manipulate the argument's variable. You should NOT have a series of if/else statements or similar logic.

3.1.3 LED Driver Initialization Subroutine

Recall that just because we are using a different programming language than last quarter, that doesn't mean the peripherals behave any differently than before. We still need to initialize the GPIO ports, which includes enabling the clocks, setting the mode to output, etc. It will be best to do this in a subroutine, such as **`void light_LED_init(void)`** which you will need to call near the beginning of your program.

3.1.4 Main Logic

Now that you have the support routines in place, tie it together with your main logic. You will need to initialize the LEDs first, then you can enter a loop that takes an integer variable and manipulates it and passes it to **`light_LED`** on each iteration. For this assignment, it will be acceptable to contain logic within main() although later on we may structure our programs a bit differently.

3.1.5 Optional UI

This is absolutely optional, but how about letting a user on the serial console set the speed and/or start and stop the action?

3.2 Hints and Tweaks

- You will find the debugger to be fully functional and able to inspect the values of variables as well as registers and memory. Use it to verify your expressions.
- You will likely find it necessary to do some common operations on variable/numbers. The regular mathematical operations (**+,-,*,/**) work like you would expect them to. You might also be interested in bitwise OR (**|**), bitwise AND (**&**), bitwise NOT (**~**). There are bit shift operations as well: shift-left (**<<**) and shift-right (**>>**). The C Book covers these in section 2.8.2 accompanied by numerous examples.
- You may find the serial console is useful for debugging. Use printf to print messages to the console when certain program milestones are reached. You can use printf to print numbers as well. The C Book covers printf in section 9.11. Don't forget to initialize the UART at the beginning of your program.
- Remember our discussion on prototypes? Assuming all of your code is in one file, the three routines you need to write will either need to appear before main() in the source file, or, you will need prototypes at the top of the source file if the bodies of the methods are below main().
- Don't forget how the LEDs work. There is a gap in the port pin assignments. Refer to the schematic as needed.

3.3 Deliverables

It is intended that you can complete this exercise by the end of the lab period. However, you will have an additional week if needed.

When completed:

1. Submit to Canvas a **single pdf** printout of your completed source code to Canvas. Include in a comment block at the top of your code a summary of your experience with this project.
2. Ask to demo your lab to instructor. You can do this via writing your name on the whiteboard.
    a. If you demo during lab in Week 2, you will earn a 10% bonus on this lab.
    b. If you demo during lab in Week 3, you will be eligible for full credit.

- Demos are ONLY accepted during lab periods. If you are unable to demo by the end of lab in Week 3, you lose the 10% of the assignment attributed to the demo (per syllabus).

- Demos must be ready a reasonable amount of time before the end of the lab period. If you write your name on the board at 9:45 and lab ends at 9:50, and there are five names in front of yours, you will be unlikely to complete your demo by the end of lab and hence lose the bonus or demo points.

3.4 Grading Criteria

For full credit, your solution must:

- Implement the delay_ms() routine as described using the SysTick timer to effect the prescribed delay.
- Implement the light_LED_init() routine to initialize the appropriate GPIO peripheral to control the dev board's LEDs.
- Implement light_LED() routine to display the provided number on the LEDs as specified.
- Implement a "main loop" to output a sequence resulting in lit LEDs bouncing back and forth on LED bar.
    - Loop will need to use a variable to keep track of current position of LED.
    - Loop should use operators to manipulate the variable to calculate next position to be lit.
    - You will need a way to detect if sequence has reached one end or the other and reverse direction.
- Minor errors usually result in a deduction of ~ 3 points (three such errors results in ~ a letter grade reduction)
- Major errors, such as not achieving a requirement, usually result in a deduction of 5 to 10 points.