Lab 1
ARM Programming Refresher and CPLD Intro

Due: Before your lab section Feb. 28 or Feb. 29

Introduction

In this lab you will accomplish several goals:

- Develop an embedded application using IAR Embedded Workbench
- Write a C program to display a counter on the LCD of the STM32L-Discovery board (ARM board)
- Write an ARM assembly program to blink the LEDs of the STM32L-Discovery board (ARM board)
- Develop a PLD application using Lattice Diamond
- Solder header pins to your MachXO Breakout board (CPLD board)
- Write a Verilog program to count in Gray code on your MachXO Breakout board (CPLD board)

All lab assignments must be done **individually**.

Requirements

ARM Board

You have two main objectives involving your ARM board. First, you need to produce a counter on the LCD of your ARM board using C. Second, you need to blink the LEDs on your ARM board using only assembly language instructions. See later sections on how to mix C and assembly.

Your counter value must be present on the LCD and update at reasonable period (0.5 Hz–1 Hz). You may select the counter base (e.g., decimal, hex, octal). The LCD should clear between values so only one number appears on the LCD at any point in time. You may use any of the libraries present in the empty project on Learn. In particular, look at the file `stm32l_discovery_lcd.h` in the empty project and the demo program that you can download from ST's website (look for the STM32L-Discovery firmware). If you use code from another source, you *must* state that in your comments.

Your LEDs must update at the same frequency as the counter. You may select how to toggle the LEDs (on/off pattern) as long as at least one LED changes per counter update. All operations on the LEDs, including clock setup, port configuration, and value changes, must be done in ARM assembly.

Since the LEDs and counter are synchronized, it is recommended that you use a single loop to toggle the LEDs and update the counter value. An empty `for` loop can provide the delay needed between counter and LED updates.

CPLD Board

Your objective for the CPLD board is to create a counter using Gray code and display the value on the LEDs present on the CPLD board. The counter must be at least 4 bits wide, but you may create a larger counter if you desire. Your counter must be implemented in Verilog.

Additionally, you must solder at least a quarter of the header pins to your CPLD board by the demonstration day. If you need help soldering, be sure to attend the Solder Workshop on Feb. 22 at 1 PM in OCNL 346.

Mixing C and ARM Assembly

Throughout the semester, including this lab, you will need to create projects that include both C and assembly code. This section will detail two ways to do this.

Assembly Function Called by C Program

In this setup you declare and call a function in C and then define the operation of the function in a separate assembly file. The templates below show you how to setup your files to define two C functions, `assembly1` and `assembly2`, that take no arguments and produce no output and whose implementation is done in assembly. Change these functions to suit your needs.

The C file would include:

```
extern void assembly1(void); /* Declare assembly1 */
extern void assembly2(void); /* Declare assembly2 */

int main(void){
    /* ... */
    assembly1(); /* Call assembly1 */
    /* ... */
    assembly2(); /* Call assembly2 */
    /* ... */
    return 0;
}
```

The assembly file (including some constant definition examples) would include:

```
SIX_HEX         EQU   0x06   ; Create a constant named SIX_HEX
THIRTYTWO_HEX   EQU   0x20   ; Create a constant named THIRTYTWO_HEX

    PUBLIC assembly1         ; Declare assembly1 as a public label
    PUBLIC assembly2         ; Declare assembly2 as a public label
    SECTION .text : CODE (2) ; Place the following code in the .text section

assembly1:
                             ; Your code for assembly1 goes here
    BX LR                    ; Return to calling function

assembly2:
                             ; Your code for assembly2 goes here
    BX LR                    ; Return to calling function

    END                      ; End of assembly file
```

Inline Assembly

The alternative approach is to use inline assembly, where you insert assembly instructions directly into a C program. This approach is useful when you only need to insert a small number of instructions. To produce inline assembly, use the `asm` macro to insert the given assembly instruction exactly where the `asm` macro is located. The example below inserts the instruction `ADD R0, R0, R1` into the C program.

```
int main(void){
    /* ... */
    asm("ADD R0, R0, R1");
    /* ... */
    return 0;
}
```

Results

Your work for this lab will be evaluated based on:

- Source code (C and assembly) that you submit to Learn
- Demonstration of your working system to your lab TA on the due date

Grading

Your lab will be graded based on:

- 20 points: LCD counter program in C
- 15 points: LED blinking program in ARM assembly
- 15 points: Verilog code for Gray counter
- 10 points: At least a quarter of the header pins soldered to the CPLD board
- 10 points: Code quality and neatness
- 30 points: Demonstration of a working system to your lab TA