I/O Using Logic Operations, and Bit Fields
LEDs and Switches

Objective:
Students will write a simple program to demonstrate use of C language bits fields. They will learn how to read and control I/O port bits by use of C-language logic operations with bit fields in both the traditional format and the modern format. C-language Structures, #define, and bit manipulation are introduced. Students will also learn how to define and memory map (assign appropriate addresses) for two I/O ports, Dip switches and LEDs.

For this lab students will execute and run their program on the hardware. Therefore, for the project set-up wizard, set "connections" to "P&E USB BDM Multilink". Save Project as LastName_Lab04.

Program Development Design worksheet: Read instructions completely and before you program, use the Program Development Design worksheet to outline your program. The logic is the same for both functions, therefore, just identify the logic steps and flow chart for one function.

Lab 04 instructions:
- In this exercise, you will write two functions. Each function will check bits and control bits on the microprocessor trainer.
  1. The first function will use traditional C bit field logic instructions to read and control I/O bits
  2. The second function will use modern C bit field structure to read and control I/O bits.
- The completed project will include header (.h) files you create and add to the program.
- You will define your own I/O ports in this program, so you will need to remove the MC9S12XEP100.c file from your project.
    - To do this: Expand *Libs* in the project explorer and then right-click on MC9S12XEP100.c to remove it from the project.
    - Then, in the main.c file, (once you remove the MC9S12XEP100.c file) comment out the #include derivative.h file. Otherwise you will get multiple definition errors.

- You will write your main logic in the main.c file.
- Your completed project should have the following files added:
    - ports.h: file to hold your I/O port assignments
    - defs.h: file to hold your bit definitions
    - protos.h; file to hold your prototype definitions
    - Each of these files should comments, including a commented header and brief comment on file purpose

- The program code will check the DIP switches at port B (address 0x01) and according to the state of the switches, will update the LEDs at port C (address 0x04)
    - You must come up with appropriate names for your port declarations

  Note: Be sure to make all bits of port C outputs by writing 0xff to its data direction register (address 0x06).

- **Code operation example**:

  if bit 0 of the switches is high, set bit 7 of the LEDs to a high without affecting other LEDS states. If bit 0 of the switches is low, write bit 7 of the LEDs low without affecting other LEDS states. Do the same for the remaining switches (bits 1-7).
  - The bits should track as follows:

    Bit 0 Switches => Bit 7 LEDs

Bit 1 Switches => Bit 6 LEDs
Bit 2 Switches => Bit 5 LEDs
Bit 3 Switches => Bit 4 LEDs
Bit 4 Switches => Bit 3 LEDs
Bit 5 Switches => Bit 2 LEDs
Bit 6 Switches => Bit 1 LEDs
Bit 7 Switches => Bit 0 LEDs

Both function versions are called from the endless loop inside of the main function, so it loops continuously, updating LEDs as you flip switches.

1. Write the first function version as instructed in **Function Version 1: Traditional C Bit Fields**, test it, if it works as instructed, demonstrate to Instructor or TA, then
2. Write the second function version as instructed in **Function Version 2: Bitfield Structures**, test it, then demonstrate to Instructor or TA.
   Note: **do not delete** the code from function version 1, just comment it out. When completed, you should be able to run either function version by commenting out the unnecessary code from the other function version. Both functions should produce identical logic I/O behavior.

- **Test** each function version by flipping each switch, and combinations of several switches, to see if the correct LED(s) light. A series of '*if else*" statements should be used. **Do not use the** *if else- if* statement since multiple switches may be on at the same time. Make sure you understand the previous sentence.
- Put your bit definitions in a file called defs.h. Put your port declarations in a file called ports.h. Put your function declarations in a file called protos.h
- For this assignment write your functions in the main.c file and you are required to write these functions below the main function.

1. **Function Version 1: Traditional C Bit Fields (#defines and Bitwise Logic Instructions)**
   The function that uses Traditional C bit fields logic instructions should check each switch individually and then write the state of that switch to the tracked LED. For each switch of PORTB you can use either simple *if else* decision blocks.

```
if(PORTB & BIT_0)        // test: is switch 0 high?
{
    PORTC |= BIT_7;  // if switch 0 is high, set LED bit 7
}
else
{
    PORTC &=  ~BIT_7;  //if switch 0 is low, clear LED bit 7
}
```

- In your defs.h header file, use define statements to specify values for BIT_0 – BIT_7.
  Example: #define BIT_0 0x01

- In your ports.h header file, Define PORTB, PORTC and DDRC. Use the proper ANSI C coding standard as discussed in Lecture.
  Example: #define PORTB (*(volatile unsigned char*)(0x01))

- **Demonstrate**: Do not continue to Function Version 2 until you have demonstrated your working Function Version 1 to the instructor or TA.

2. **Function Version 2: Modern C Bit Fields (bit fields wrapped in a Structure)**
   Do not delete but comment out the unnecessary parts of previous program code.

   - In your defs.h file,
     - define a structure template with fields for all eight bits of a character-sized port. Give each field a generic name such as BIT_0, BIT_1, BIT_2, etc or some similar name.
     - Use a typedef statement to define a variable type for the structure
     - Then use the variable type to Define variables for the Port switches and the Port LEDs
     - Update your other files to reflect the changes/additions for this Version 2
     - For Function Version 2: Use a similar if - else structure for checking and setting bits.
     - You must use your defined structure variables and structure fields with the dot operator.

Demonstrate: your working Function Version 2 to the instructor or TA.

Before you submit: ensure each of your files includes a commented header, a brief description of the program or file purpose and well organized and commented code.

Submission: Submit your completed well organized design worksheet, main.c, defs.h, ports.h and protos.h files as well as a zipped version of your entire project to Canvas assignments for Lab 04.