Hardware Abstraction Layer: I/O Using Macros
LEDs and Switches

Objective:

Students will utilize knowledge learned from previous Bit Fields lab 04 to extend program code efficiency by use of Hardware Abstraction Layer (HAL). Students will write a simple program to demonstrate use of C language pre-processor directives and macro definitions to monitor and control I/O bits and memory mapped (assign appropriate addresses) for two I/O ports, Dip switches and LEDs.

For this lab you will run your program on the hardware. Therefore, for the project set-up wizard set your "connections" to "P&E USB BDM Multilink". Save your Project as LastName_Lab05.

Program Development Design worksheet:

- Since this lab uses the same program flow logic as lab 04, (only difference is: you will use macros with your bit fields), you **will not** create new outline.
- You are required to create a table to define your macros and indicate where you will place these macro definitions.
- Read instructions completely and before you program, make sure you have clearly defined your macros and understand how to use them in code.
- Example Macros table:

Identify Macros		
Macro definitions	**Example Macro calls in code**	**purpose**
#define	LED_ENABLED(Bit_01);	//make LED port bit an output

Lab 05 instructions:

- In this exercise your will write one function. This function is called from the endless loop in main function. The function will check bits and control bits on the microprocessor trainer. You are required to use macros as listed below in **Using Macros** section and exampled in Lecture. **Your program is required to follow suggested ANSI C standard coding practices as discussed in lecture.**
- You will define your own I/O ports in this program, so you will need to remove the MC9S12XEP100.h file from your project.
  - To do this: Expand *Libs* in the project explorer and then right-click on MC9S12XEP100.h 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 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
  - *you may add additional files if you deem necessary*
  - Each of these files should comments, including a commented header and brief comment on file purpose

- Put your bit fields and macro definitions in a file called defs.h. Put your port declarations in a file called ports.h. Put your function prototypes in a file called protos.h.
- The program code will make use of macros and bit fields definitions to check the DIP switches at port B (address 0x01) and update the LEDs at port C (address 0x04) according to the state of the switches.
- Make all bits of port C outputs by writing 0xff to its data direction register (address 0x06).

- **Code operation**:
  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:

    Bit0 Switches => Bit7 LEDs
    Bit1 Switches => Bit6 LEDs
    Bit2 Switches => Bit5 LEDs
    Bit3 Switches => Bit4 LEDs
    Bit4 Switches => Bit3 LEDs
    Bit5 Switches => Bit2 LEDs
    Bit6 Switches => Bit1 LEDs
    Bit7 Switches => Bit0 LEDs

  Surround the main program logic (function call) with an endless loop so it loops continuously, updating LEDs as you flip switches. You can use the endless loop already in main.

- **Test** your program by flipping each switch, and combinations of several switches, to see if the correct LEDs light. A series of '*if else*' statements can be used. **Do not use the** '*if, else if*' statement since multiple switches may be on at the same time.

- **Using Macros**
  Macros must be used to check a specified switch, turn on or off a specified LED (without affecting other LEDs) and to initialize LED bits as outputs. The macros should be placed in the defs.h header file.

  The macro calls which you must "#define" are as follows:

```
LED_ENABLED(n);      //make LED port bit an output
LED_ON(n);           //set LED bit high
LED_OFF(n);          //set LED bit low
SWITCH_STATE(n);     //return true if high, false if low
```

  Note:
  For this program, when the macro is called, 'n' is replaced with a numeric constant from 0 – 7 that specifies the LED or switch number. When defining your macros, use the C language connector (##) to append the device name (Port B or Port C) to the literal BIT to specify a bit

number. All eight bits need to be #defined, but there need only be one "#define" line for each of the macro definitions listed above.

- **Test**: Once your program is working, try to modularize functions operations.
  Add additional comments for what you changed, if you were successful or not, if successful, how do you think it made your code better?

  Have your code checked by instructor or TA
  Put your "add additional comments" in your code. Run your program to ensure all files include most recent changes.

Demonstrate: your working program 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 05