ECE-3120
Spring 2008

LAB 9 – Parallel I/O, part 2

The purpose of this lab is to learn the use of the 68HCS12 parallel ports to interface to simple devices, in a program that counts and displays switch activations on the LEDs.

PRE-LAB:

Prepare pseudocode and the first draft of the program. This must be completed before coming to the lab and shown to the lab instructor at the start of the lab session. Note: The Pre-Lab must be typed into a proper *.ASM source file, following our standard Program Format requirements.

Approved: Lab TA _____ *Date* _____

PROGRAMMING ASSIGNMENT:

Write a fully-commented program for the Dragon12 board, following our standard Program Format requirements, including appropriate directives and labels for memory operands and constants, called **SwitchLeds.asm**. The program should do the following:

- This program will run forever in a loop, as most real embedded systems do. So run the program continuously and never exit, except when the reset button is pressed or power is cycled off/on.
- The LEDs can be constantly enabled and all Digits must be disabled, so no scanning or multiplexing is required of the two.
- Read the changes in values of two pushbutton switches (labeled on-board as SW2 and SW5) reliably. So **the switches must be de-bounced and edge-detected so that one and only one count event is produced each time the switch is pressed, no matter how long it is between pressing and releasing the switch**.
- Initially reset the counter to zero and turn off the LEDs.
- Whenever SW2 is pressed, increment the counter. Whenever SW5 is pressed, decrement the counter. Display the current value of the counter in binary (%00000000 to %11111111, $00 to $FF, 0 to 255) on the 8 LEDs. Wrap-around the count to 0 after incrementing above $FF and wrap-around to $FF after decrementing below 0, which is the natural behavior of binary counters. LED PB7 is the MSB. A LED should be ON if that count bit = 1 and OFF if that count bit = 0.

- Only one of the two switches may be pressed at any given moment, never both simultaneously.
- Use polling (not interrupts) for all I/O operations. It is important to your projects to learn this well. Ask questions ahead of time if you do not understand.
- Include and use the standard files, HCS12.INC and DELAY.ASM, as appropriate.
- Use a 10ms delay for debouncing both the press and release of a pushbutton (SW2 and SW5).
- Follow the overall program organization that was recommended in class.
- As always, the code must start at $1000.

The program may use directives to create data in memory locations beginning at $1200 as needed. Clearly define all data in your code!

Procedure: First, use D-Bug12 to fill memory locations $1000 through $14FF with zeros. Then assemble, download, and debug/execute the program as follows.

a. Set breakpoints at the end of all major steps, as necessary to debug your program, pausing at each breakpoint to display the important values. Verify that each value is correct at each breakpoint and the final results are correct at the end of the program. Use the listing file to determine the breakpoint address. TIME is a very important factor in the operation of this program, so using breakpoints will definitely mess up all your program timing plans. You will have to keep this in mind and work around this problem to verify the logic of your program.
b. Then reset the processor (which also removes the breakpoint), download the program, run it at full speed, and verify that the program runs properly under all conditions.
c. When finished debugging and executing, copy the entire terminal window output and paste it into a Notepad or Word document for inclusion in the report. You should edit out mistakes and unnecessary repetitions before submission. Good hand-written comments and explanations are ESSENTIAL to make this meaningful to the grader.

Approved: Lab TA _____ *Date* _____

Things to turn in as your Lab Report, attached in this order:

1. This assignment sheet, with your name at the top, signed by the TA where shown.

2. Your uncorrected pre-lab document (commented source code).

3. A printout of the final **SwitchLeds.asm and SwitchLeds.lst** files. You'll need to print the listing file in landscape mode to make it fit. Use Notepad to print them.

4. A printout from the terminal screen (method: highlight text, type ctrl+c, and paste into a Notepad or Word document, using Courier New as the font) that includes everything done. Add brief hand-written comments and highlight important values at each step in the procedure, showing the relevant memory contents and register contents. Good hand-written comments and explanations are ESSENTIAL to make this meaningful to the grader.

5. Answer the following questions, in your document:

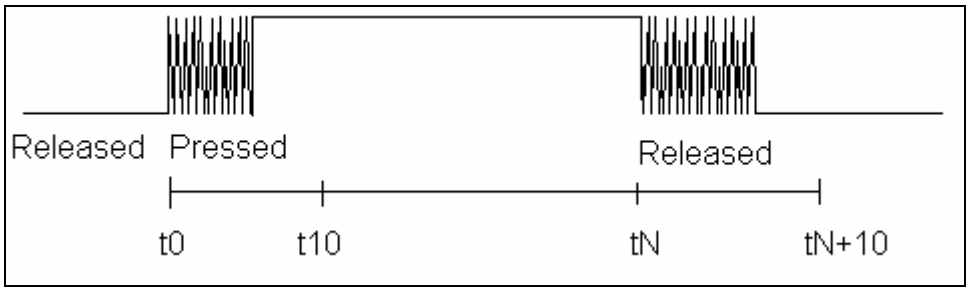
    1) (a) Do all switches in all applications need to be de-bounced and edge-detected? (b) Why must the pushbuttons be de-bounced and edge-detected in this lab's application? (c) Describe how a program can de-bounce and edge-detect a switch, in text-only style without showing any code.
    2) On our Dragon12 board, the LEDs and all four 7-segment digits are all driven by the same data port (B). (a) Why is the circuit designed this way? (b) How else could the circuit be designed, to simplify the programming required? In this Lab 6, we only show data on the LEDs, so it is not a problem here. (c) But explain how a program can show different values on the LEDs and each of the four digits simultaneously (in text-only style, no code).

Notes:

Pushbutton and switch interfacing is an important task in microcontrollers. Many devices use them and proper interpretation of their inputs are required. However, these buttons and switches are mechanical and, as such, require more care.

There are typically two positions in a switch: one that connects the port on the microcontroller to ground and one to 5V, resulting in a logic low and high, respectively. (Active low switches must be interpreted as such.) When a mechanical switch transitions from one position to the other, there are very small bounces that cause the switch to go back and forth between positions before finally settling on the desired position. The time it takes for this "bouncing" to stop is less than 10ms. Noise is also a concern. If the transient noise is strong enough to reach the threshold for a logic high, the microcontroller could detect it as a button press.

Debouncing is the act of negating the effect of this rapid, unwanted transitioning. Very often, you want to identify that the button was pressed once and released. Consider the application of a TV remote with volume control. Without debouncing, a single press of the "Volume Up" button could result in the microcontroller seeing many presses, causing the volume on the TV to instantly be maxed out.



One way debouncing can be done is by sampling and polling.

1. First, the switch is continually checked to determine if there was a press, or transition from 0 to 1.
2. Once a transition is detected, wait for 10ms.
3. Sample the switch. If the current switch is still high (1), the transition was a legitimate press. Else, the switch is a low (0), meaning the press was not valid and you should return to Step 1 – waiting for a press.

Often, there is a need for the button to be released. Without detecting a release, a valid press could be reported every 10ms. Some applications rely on this, others do not. To detect the release of a button, do the same steps given above, but for the transition from 1 to 0.