ECE263

Lab 5

Spring 2013

Parallel Ports & User Keypad

Revised: 7 Mar 2013 DAL

(intentionally left blank)

Object:
In this lab you will design and code an assembly language program that will use a 4-key keypad and a 4-digit LED display to simulate a simple user interface for an embedded system.

Material:
user specified

Background:
When embedded systems need to get input data from a human user of a product, one or more momentary single pole single throw (SPST) switches are often used. For a small number of switches, one side (pole) of the switch is normally tied to GND and the other side is tied to an I/O port pin of the microcontroller. In order to bias the signal to a high state when the switch is not engaged (pushed), a pull-up resistor is tied between the signal and Vcc (see Figure 1).

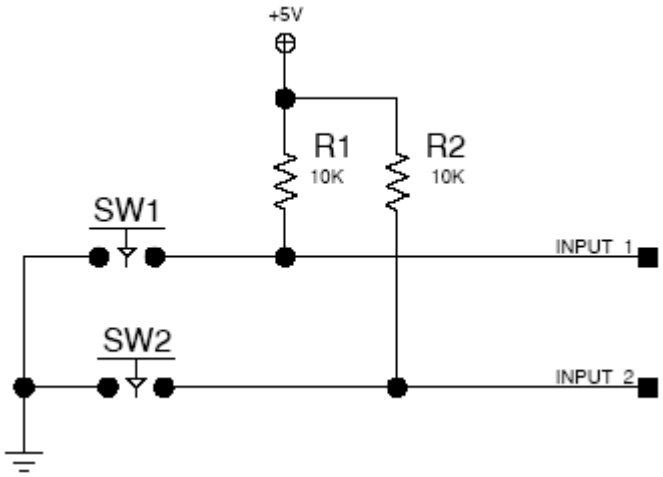

Figure 1. Two Momentaty SPST Switches

When the switch is not activated the input lines will be at a high voltage and will be read as a 1. When a switch is pressed, the pull-up resistor is shorted to GND and the input will be read as a 0. The program running in the microcontroller would look for any input that was a 0 to determine if the corresponding user input (switch) was activated or not. It would be possible to tie the common connections to V_{CC} rather than GND as long as the pull-up resistors were changed to pull-down (to GND) resistors instead and then the program would look for a 1 to determine if a switch was activated, however this is not the normal practice. In either case the port pins need to be biased to either a high or low state when the switch connections are not present.

Many embedded systems use a 4-key keypad for getting user input. The four keys are sometimes labeled: MODE, UP, DOWN and ENTER. The MODE key would be used for selecting the mode of operation of the system or possibly a specific item from a list of possible choices. The UP and DOWN keys would cycle through the possible selections, and the ENTER key would initiate the chosen action.

The Dragon Board has four momentary action push button switches which are wired in parallel with an 8-position DIP switch and are connected to the 9S12 via Port H as shown in Figure 2.

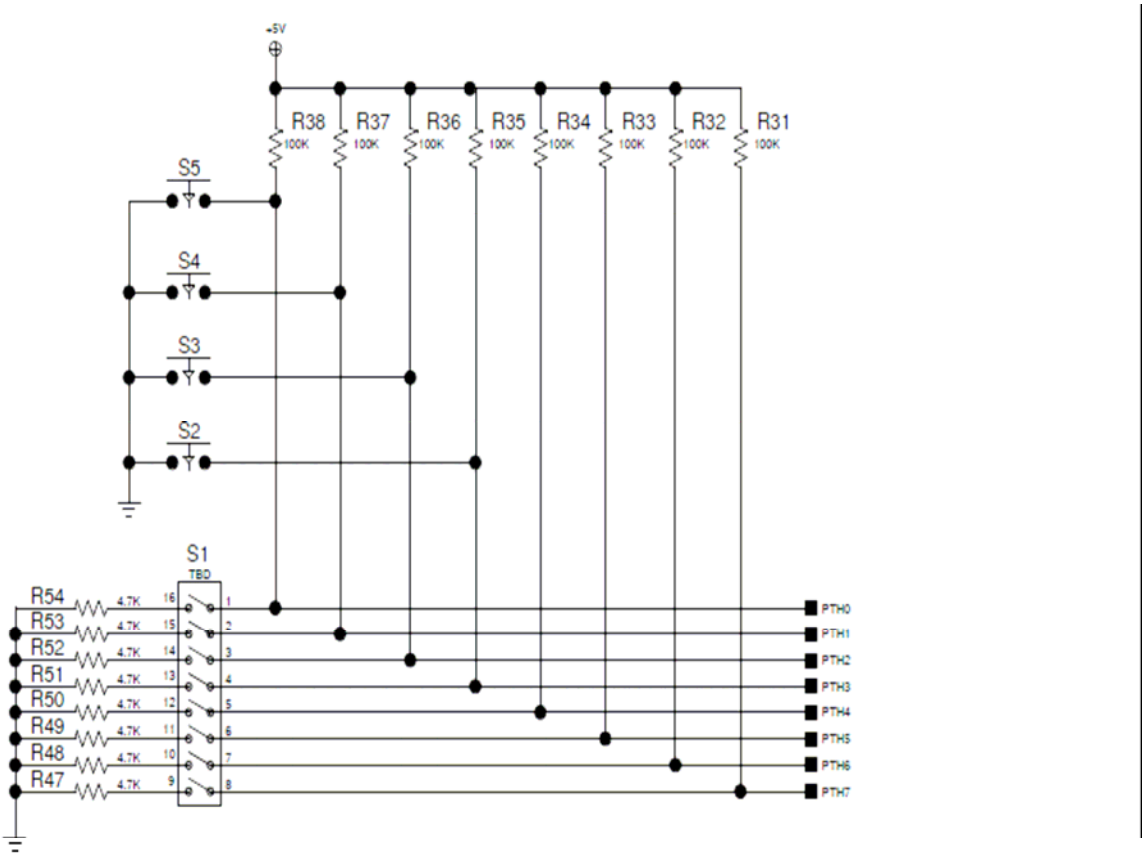

Figure 2. 4 Momentary Push Button Switches and 8-Position DIP Switch

When the DIP switches are left in the OPEN (OFF) position then the state of the push button can be determined by reading PH0 to PH3. The process of determining which, if any, key is pressed is done in a KEY_DETECT subroutine as described previously. KEY_DETECT must be called on a periodic basis to determine if the user has tried to initiate an action. The frequency of the subroutine call is determined by the reaction time of the human user and the desired "responsiveness" of the keypad. If the frequency is too low, there will be an apparent lag between the keypress and the action and will be very annoying. If the frequency is too high, then too much of the available processing time is used by this function with no appreciable gain in performance. Normally a response time of 100msec to 200msec is sufficient.

Another challenge that has to be dealt with when using mechanical switches for data input is caused by a phenomena known as "switch bounce". When a switch is opened or closed the contacts do not break or make cleanly but have mechanical bounce that translates to an electrical signal that goes high to low many times before settling down to the correct state as illustrated in Figure 3. To eliminate bad data caused by the switches, the program must have a "debounce" routine to filter out all of the unintended voltage swings when the switch contacts bounce. The length of time required for the filtering is dependent upon the particular switches or keypads being used and may vary greatly. Another potential problem occurs when the user holds the switch closed for a long period of time. The KEY_DEBOUNCE routine should also insure that a key press is only detected as a single closure and not multiple closures no matter how long the

user holds the key activated. The program specification at the end of this write-up will describe one possible method of filtering which is useful for detecting single key entries.

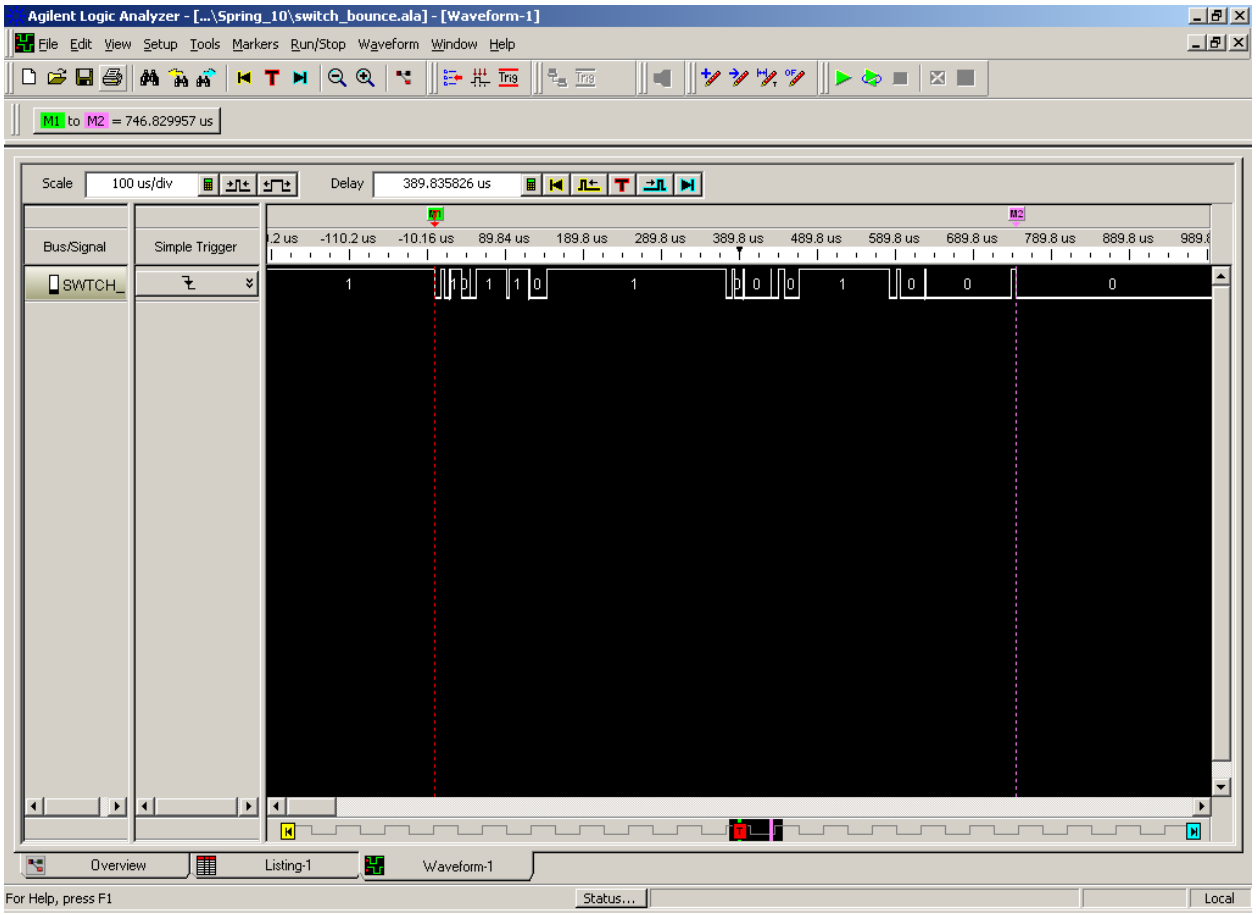

Figure 3. Example of Switch Bounce

After a valid key press has been detected, the program would then decode the entry to determine what action the program is required to take. The decode routine would only be called after the key activation is first detected and must not be called again until the switch has been deactivated.

The keypads are usually used in conjunction with some type of display unit to provide feedback and/or system status.

Preparation:

The program for this lab will added to the program you generated for Lab 3, *Real Time Interrupt*, and Lab 4, *Parallel Ports & LED Display*. This program will add the capability to get data from a 4-key user input port and then perform some simple functions. See the specification for the program attached at the end of this write-up for details on how to write the new code and add it to your existing program.

Procedure: Working with your lab partner, complete the following steps:

1. Create a new project and enter your program as *main.asm*. Generate the executable object file using the *Make* facility and then download the program on the Dragon 12 Plus.
2. Now set up the Logic Analyzer. Get two "flying lead" pod from the box at the front of the room.
3. Recall the Logic Analyzer setup that you (hopefully) save at the end of Lab 4.
4. Change the Port B probes to Port M
5. Add the following new points:
6. Add the signal names MODE, UP, DOWN and ENTER.
7. Attach leads to the pins labeled PH0 through PH3 on the Dragon 12 Plus. This is the data being read from the user input switches.
8. Now add four more signals labeled DSP1, DSP2, DSP3 and DSP4.
9. Attach leads to the pins labeled PP0 through PP3 respectively. These pins are the signals being sent to the cathodes of the seven segment digits.
10. Add the Name ANODES and attach the corresponding probe leads to the pins labeled PB0 through PB7. This is the data being sent to the anodes of the seven segment display.
11. Click OK to finish the setup.
12. The analyzer is set up so now start the debug process using the *True Time Simulator/Debugger*.
13. Start debugging your program as you normally would using *single step* and *breakpoints* as appropriate. You will need to press (activate) keys on the keypad in order to see if the program works correctly.
14. If the program is not working as expected, go correct the source file and re-try.
15. Use the logic analyzer to verify the operation of your program.
16. Use the cursors to take timing measurements.

17. Print out screen shots of the sequences to show that you have generated the correct timing for all of the parts of this program, i.e. Task 1 (DSP1 – DSP4), Task 2 (switch detect and debounce), Task 3 Decode Routine. Include these prints with an explanation of each one in the lab report.
18. Demonstrate the working program to the TA or the instructor and explain what it is doing and how it operates.
19. Have the TA or instructor sign the Verification Sheet.

Lab 5: Programming Assignment

General Description

Your program will display and/or modify the contents of one of four 8-bit variables (NUM_0 to NUM_3) stored in RAM space. You will decode the MODE, UP, DOWN and ENTER keys of the 4-key keypad to determine what actions you will perform on those variables.

You will use the **MODE** switch to select one of the four 8-bit variables and copy it's contents to the most significant byte of the 16-bit variable named NUM_2_DSP. The lower byte of NUM_2_DSP will be separated into two nibbles: the upper nibble will either be blanked or will be a dash the lower nibble will contain a count showing which variable is being displayed (i.e. 0 to 3). You will display NUM_2_DSP on the seven segment LED display using the program you wrote for Lab 4. .

> (NOTE: DS1 and DS2 will display the two hex characters of the selected number, DS3 will be a blank or a dash, and DS4 will be the variable number)

If the **UP** or **DOWN** keys are activated then you will either increment or decrement the most significant byte of NUM_2_DSP. When the **ENTER** key is activated the current contents of the most significant byte of NUM_2_DSP will be written back to the original (stored) variable (NUM_0 to NUM_3).

NUM_2_DSP is being used both as a scratch pad and as the location to be displayed. If the user changes the contents of the scratch pad location it gets displayed but it does not affect the original variable unless and until the **ENTER** key is activated.

As stated previously, these functions are to be added to your program for Lab 4. The "waste time" code of Task 1 is to be replaced by two subroutines. The first subroutine will read the keypad and leave the current data in a global variable and the second routine will *debounce* that global variable. In place of the counter that calls Task 3, you will use the results of debounce. If a valid key was detected you will set Task 3 flag. In Task 3 you will decode the specific key that was activated and then call the appropriate routine to perform the desired action. Ideally, you will have each operation be a separate task that will have its own task flag and will be called by the scheduler. To do this, you will have to use four more bits of T_FLG to signify the tasks for each one of the four possible actions.

Routine Specifics

Key Detect: The key detect routine is going to be a subroutine that is called from Task 1. It will read Port H and check to see if any keys have been pressed If a key press was detected for any of the switches, you should set a variable called NEW_KEY with the key number ($00 through $03). If no keypress was detected you should set NEW_KEY to $FF (minis 1). If more than one switch is pressed, you probably should also set NEW_KEY to $FF (minis 1).

Debounce: The debounce routine is going to be a subroutine that is called from Task 1 right after KEY_DETECT. It will compare the current state of the keypad with the previously debounced entry. If a new condition is detected on a specified number of passes (i.e. filter time) then it is assumed that the switch has stopped bouncing and the entry is now valid. The DEBOUNCE routine will require two variables: CURRENT_KEY and KEY_COUNT. At the start of the program the value of CURRENT_KEY is set to $FF and KEY_COUNT is set to $05. $00. After the KEY_DETECT subroutine has been executed, DEBOUNCE will compare the

value of NEW_KEY with CURRENT_KEY. If they are the same then KEY_COUNT will be set to $05 but if they are different, KEY_COUNT will be decremented by 1. If KEY_COUNT reaches zero, then CURRENT_KEY is set to the value in NEW_KEY, KEY_COUNT is set back to $05, and the task 3 flag bit (TSK_3) is set to 1 to indicate that a change of state was detected and that Task 3 must be called to decode the new entry.

Decode: Task 3 is going to be the DECODE routine and it will determine which key was pressed and it will either call the routine to perform the required action or (ideally) it will set one of four task flags that will be used by the task manager (scheduler) to call the required routine.

The MODE routine will increment the counter pointing to NUM_0 through NUM_3, i.e. the lsbyte of NUM_2_DSP, get the contents of that variable, and put it into the msbyte of NUM_2_DSP. The UP or DOWN routine sill either increment or decrement the msbyte of NUM_2_DSP, and the enter routine will save the contents of the msbyte of NUM_2_DSP back to the appropriate variable.