COE3DQ5 Lab #2
Finite State Machine Design for the PS/2 and LCD Interfaces

Objective

To understand the importance of finite state machines (FSMs) in digital systems design. Gain experience with coding styles for FSMs and implement digital systems using the PS/2 and LCD peripherals on the DE2-115 board. PS/2 is a serial port used to interface a keyboard or a mouse to a personal computer. LCD is a low-power/flat-panel display that uses liquid crystals and it is predominant in embedded devices.

Preparation

- Revise the first lab and FSMs
- Read this document and get familiarized with the source code and the in-lab experiments

Experiment 1

Figure 1(a) shows the FSM that corresponds to the source code from ***experiment1***, which describes an FSM that detects if push-button 0 has been pressed three times consecutively. Whenever push-buttons 1, 2 or 3 are pressed, the FSM returns to an idle state and it waits until push-button 0 is pressed again. Before returning to the state that indicates whether push-button 0 has been pressed three times consecutively, the FSM goes through two intermediate states that record if push-button 0 has been pressed once or twice consecutively. In any other state except the one where push-button 0 has been pressed for three times consecutively, the value displayed on the 7-segment-display should be "F".

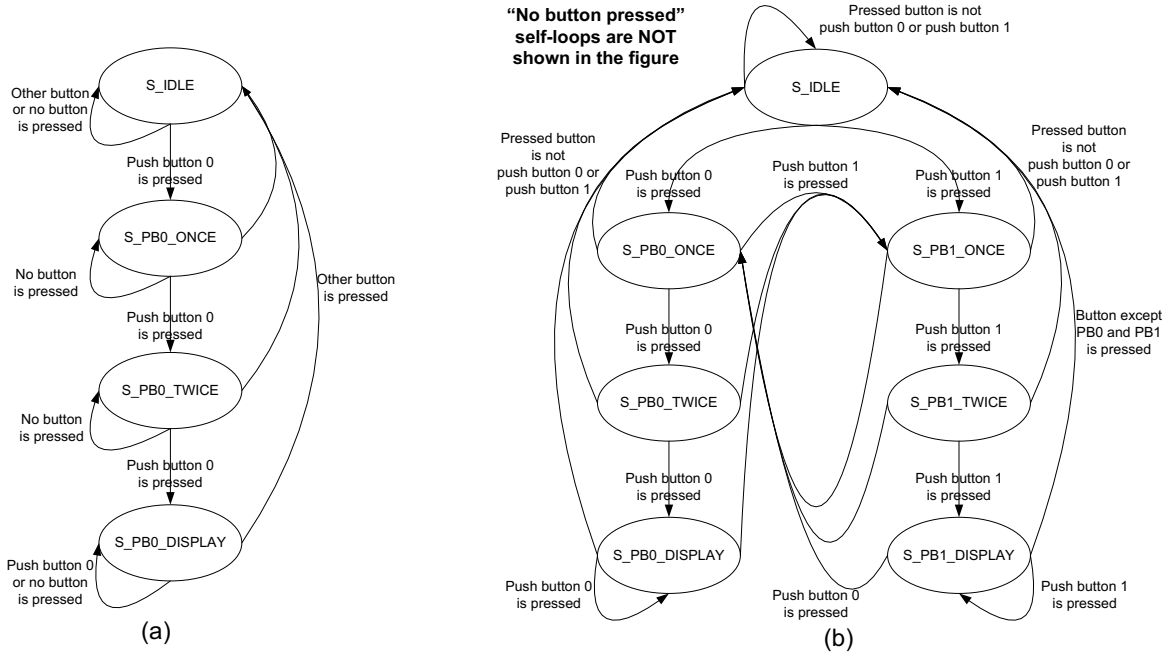


Figure 1 – State diagrams for experiment1

You have to perform the following tasks in the lab for this experiment:

- verify if the FSM shown in Figure 1(a) is described correctly and if its circuit implementation works
- change the FSM description in such way that if push-button 1 is pressed three times consecutively the number displayed on the 7-segment-display is "1"; note, if push-buttons 2 or 3 are pressed then the system returns to the idle state; for clarity Figure 1(b) shows the FSM to be implemented;

Experiment 2

In this experiment you will learn about the PS/2 port and you will modify the FSM of the PS/2 controller.

Figure 2 shows the timing of the PS/2 interface. This interface has two bidirectional signals: PS2_clock and PS2_data. The given implementation supports only the receiver part of the PS/2 interface (i.e., we only receive data from the keyboard). If a key is pressed or released 1 byte of info is sent approx every ms from the PS/2 keyboard. There is a reference clock (PS2_clock) which is "1" when no data is sent. When the PS/2 device initiates the communication a PS2_clock pulse must be accompanied by a start bit (active low). This is detected in the S_PS2_IDLE state. In the next state (S_PS2_ASSEMBLE_CODE), on each edge of the PS2_clock, the value of the PS2_data is fed into a right-shift register (PS2_shift_reg). After 8 clock cycles the parity bit is checked in the S_PS2_PARITY state (in our implementation no action is taken on this parity bit, nonetheless the state must exist in order to comply with PS/2 protocol initiated by the keyboard). The final state S_PS2_STOP checks if the stop bit (active high) is passed with the last PS2_clock. In the same state we set a flag PS2_code_ready that together with the PS2_code (loaded from PS2_shift_reg) will be passed for further processing outside of the controller. It is important to note that the FSM is clocked at 50MHz and it is enabled when an edge is detected on the PS2_clock.

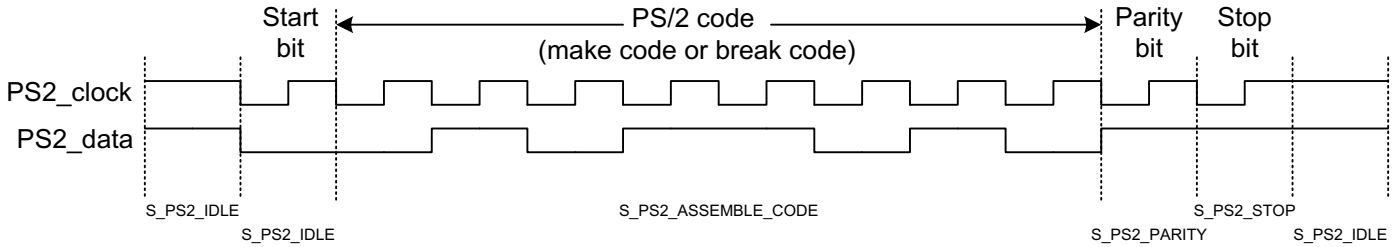


Figure 2 – Timing of the PS/2 interface

Figure 3 illustrates the processing of the PS2_codes received from the PS/2 controller. If an edge is detected on the PS2_code_ready signal the PS2_code is placed into the 8 least significant bits of a 24-bit left-shift register. This register controls the 6 rightmost 7-segment displays. Note, when a key is pressed a "make code" is generated and when a key is released a "break code" is generated by the PS/2 keyboard. For the keys that we are concerned with in this lab, the make code is 1 byte and the break code is 2 bytes (the first byte is 8'hF0 and the second one is the same as the make code). The keys supported in our simplified implementation are available in the "PS2_keyboard_codes.pdf" file provided in the ***experiment2*** directory. The PS2_make_code flag that comes out of the PS/2 controller module can be used to differentiate between the make codes and the break codes.

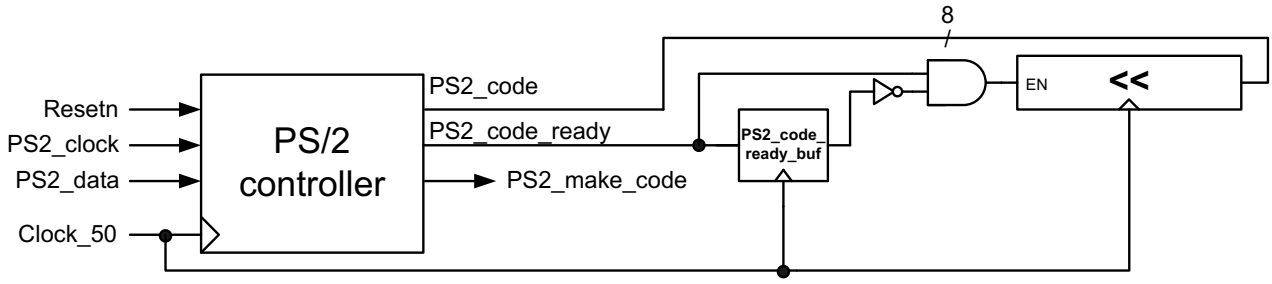


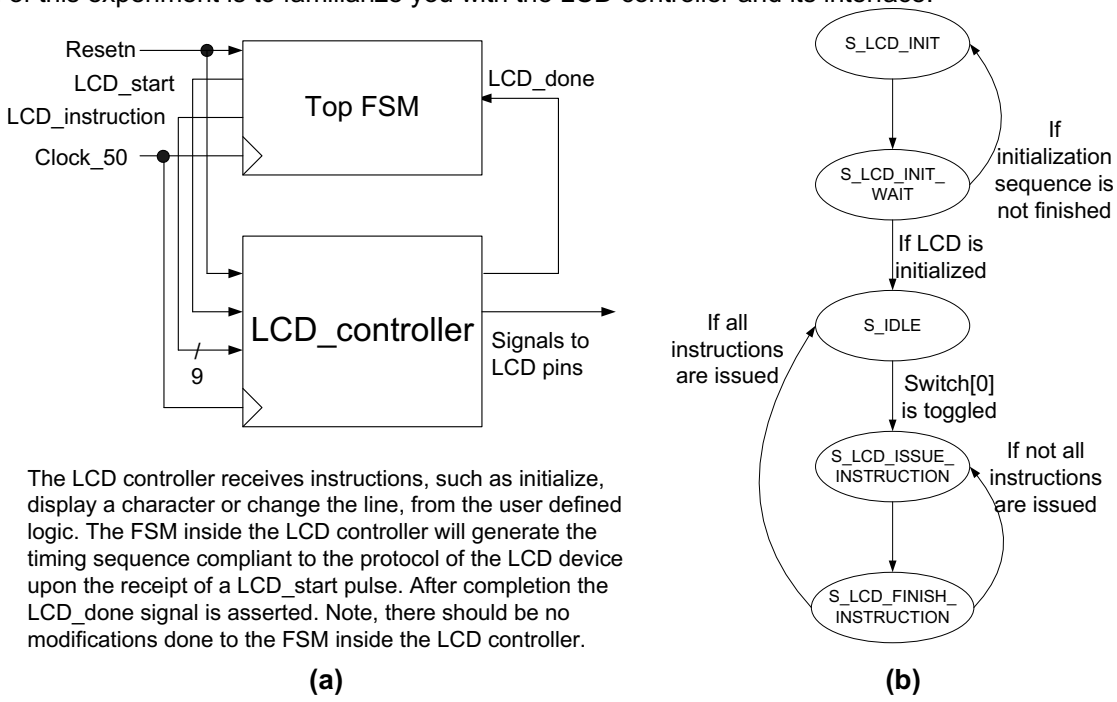
Figure 3 – Interfacing the PS/2 controller to the top level design

You have to perform the following tasks in the lab for this experiment:

- check on the 7-segment displays that if a PS/2 keyboard key is pressed and released then both the make code and the break code are properly assembled by the PS/2 controller
- implement the functionality of the PS2_make_code flag as follows: in the state machine from the PS2_controller set the PS2_make_code to a "1" only if a make code is generated; in the top level file the PS2_code should be passed to the shift register only if the PS2_make_code is set; this will enable us to "filter out" the break codes and display only the make codes on the 7-segment displays

Experiment 3

The aim of this experiment is to familiarize you with the LCD controller and its interface.



The LCD controller receives instructions, such as initialize, display a character or change the line, from the user defined logic. The FSM inside the LCD controller will generate the timing sequence compliant to the protocol of the LCD device upon the receipt of a LCD_start pulse. After completion the LCD_done signal is asserted. Note, there should be no modifications done to the FSM inside the LCD controller.

(a) **(b)**

Figure 4 – LCD controller interface to top FSM and the state transition diagram of the top FSM

Although the LCD interface is bidirectional, in our simplified implementation the user defined logic is only sending instructions to the LCD. The behavior of the LCD controller can be explained using Figure 4(a). After power-up an initialization sequence is provided after which the LCD controller waits for instructions such as print character or change the line. When the user defined logic sends a new instruction to the LCD controller it must generate a pulse on the LCD_start signal and provide the appropriate instruction. The instruction is on 9 bits and if the most significant bit is a "1" then the remaining 8 bits will uniquely identify the character to be displayed. In our implementation the character codes can be found in "LCD_codes.pdf" provided in the ***experiment3*** directory. If the most significant bit is a "0" then the instruction is a system instruction (in our implementation we cover only the "initialization" and "change line" instructions). The transition diagram of the top FSM that decides what kind of characters are displayed on the LCD screen is shown in Figure 4(b). The first two states are for initializing the LCD device and should not be modified. The bottom three states supply 32 character display instructions and 1 change line instruction. These instructions are initiated by toggling SWITCH_I[0], which is required to leave the S_IDLE state.

The LCD instructions are provided as constants to the LCD_data_sequence signal and they are looked up using a counter (LCD_data_index) updated within the S_LCD_FINISH_INSTRUCTION state. Note, after LCD_start is asserted in the S_LCD_ISSUE_INSTRUCTION, it will be de-asserted immediately in the following clock cycle in the S_LCD_FINISH_INSTRUCTION state. Only then the LCD_done signal will be monitored to see if the LCD controller has finished its write cycle. If this is the case, the next instruction will be sent to the LCD controller. We will return to S_IDLE when all the characters have been displayed.

You have to perform the following tasks in the lab for this experiment:

- understand the behavior of the FSM from the top module and verify if the design works correctly
- change the displayed message as you wish (use the character codes from "LCD_codes.pdf")

Experiment 4

This experiment puts together the PS/2 and LCD controllers, thus displaying typed characters.

Embedded memories are part of the field-programmable array (FPGA) fabric and they can be configured either as a read-only memory (ROM), single-port random access memory (RAM) or dual-port RAM. In this lab we will be using only a ROM, which has the following ports: input address, clock and output data. The ROM can be initialized at compile time and its content is programmed into the FPGA together with the rest of the configuration stream required for logic elements and switches. The ROM content is specified in the memory initialization file ("MIF"). The ROM content can also be initialized with an "HEX" file format. The "HEX" file format will be used later in this course when simulating embedded memories.

The previous two experiments have introduced the PS/2 and LCD controllers. You have displayed the make codes on the 7-segment displays and you have printed characters on the LCD. As it is obvious that one would like to print the typed characters, the question is how can we put together the two controllers? This objective is achieved using a ROM that is employed as a code converter. Figure 5 illustrates the basic principle on how the PS/2 and LCD controllers are bridged using a ROM.

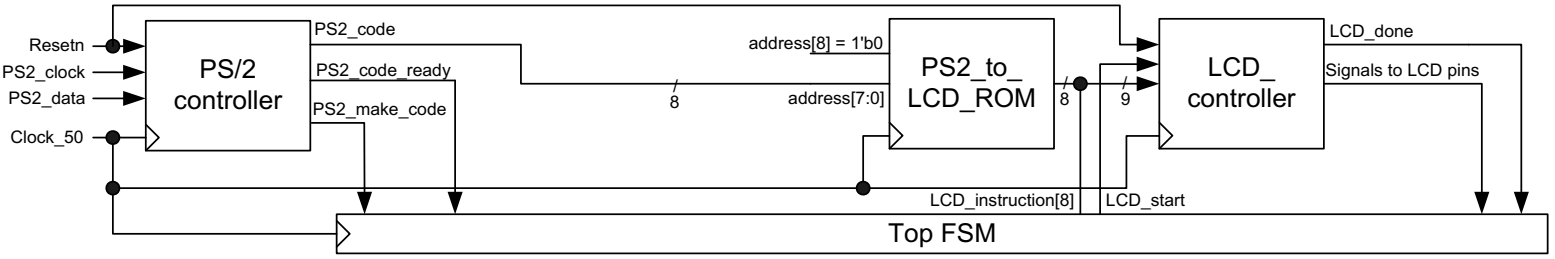


Figure 5 – Using a ROM to bridge the PS/2 and LCD controllers

The PS/2 controller from the figure has been updated to provide the make codes of the typed keys when the PS2_make_code flag is set. The ROM from the figure, stores the LCD character code in the location given by the make code. For example, since the make code for "z" is "8'h1A", as given in the file from ***experiment2*** ("PS2_keyboard_codes.pdf"), and the LCD character code for "z" is "8'h7A", as given in the file from ***experiment3*** ("LCD_codes.pdf"), we will store 8'h7A in location 9'h01A. The ROM address is on 9 bits (and not on 8 bits as given by the make code width as shown in the figure) because the ROM has two segments. The lower segment (i.e., most significant bit is a "0") stores the LCD code values for the lower-case characters and the upper-segment (currently unused) can be used to store the LCD code values for the upper-case characters.

The content of the ROM is provided in the "MIF" file from the working directory (***experiment4***). When a character has been typed, it will be displayed by hooking up the lower 8 bits of the LCD_instruction word to the output of the ROM. The S_IDLE state is left when a new make code has been detected (by monitoring a transition on the PS_code_ready and ensuring that PS_make_code is set). Because the LCD instruction is on 9 bits, the most significant bit (i.e., "1" for printing a character) is provided by the FSM from the top level design file. To change the line on the LCD display, the state machine from the previous experiment has been extended with two extra states. These two states S_LCD_ISSUE_CHANGE_LINE and S_LCD_FINISH_CHANGE_LINE ensure that after 16 characters have been typed, the LCD display advances to a new line (i.e., from the top line to the bottom line or the other way around). The instruction for changing lines is issued also by the FSM.

You have to perform the following tasks in the lab for this experiment:

- understand the behavior of the FSM from the top module and verify if the design works correctly
- update the ROM file to support keys 0 to 9 (use "PS2_keyboard_codes.pdf" and "LCD_codes.pdf")

Experiment 5

The objective of this experiment is to put together a large FSM by showing how to buffer several keys typed on the PS/2 keyboard and subsequently displaying all of them at once on the LCD screen.

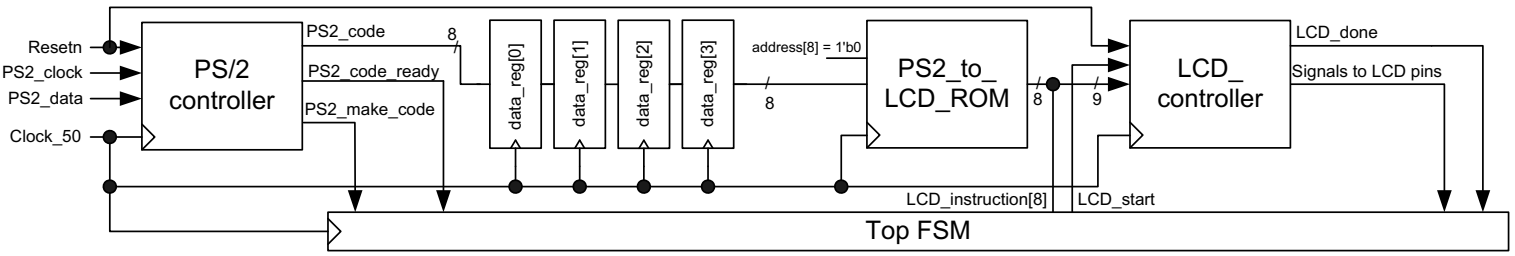


Figure 6 – Using a shift register structure to buffer 4 typed keys before displaying all of them on the LCD

Figure 6 shows how 4 data registers are connected into a shift-register structure between the PS2/controller and the code-converter ROM. These 4 data registers are used to buffer the typed keys by storing and then shifting the last 4 make codes that were generated by the PS/2 controller. After all the 4 registers have been filled with new values, the top FSM (shown in Figure 7) will switch from the S_IDLE to S_LCD_WAIT_ROM_UPDATE. This state together with S_LCD_ISSUE_INSTRUCTION and S_LCD_FINISH_INSTRUCTION will enable the necessary steps to display the 4 pressed keys. When a key is printed, the shift register structure will provide the stored make code to the ROM by shifting data_reg[i] to data_reg[i+1], while filling data_reg[0] with all zeros.

The output of the most significant register from the shift-register structure (i.e., data_reg[3] in this example) is connected to address lines of the code converter ROM. There is one counter “data_counter” that keeps track of how many make codes have been printed and another counter “LCD_position” that keeps track of the position on the LCD screen. When putting together the state machine shown below with the above-mentioned counters and registers, the ROM can translate the 4 buffered make codes to LCD codes such that the corresponding characters are displayed on the LCD.

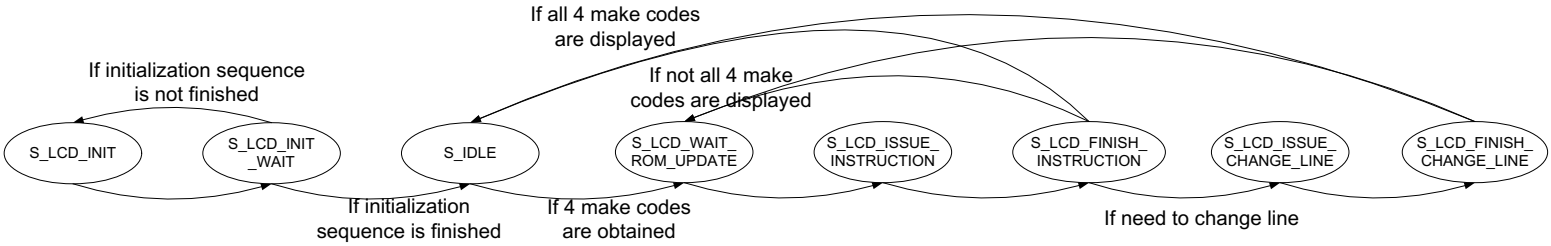


Figure 7 – The revised FSM for buffering 4 typed keys before printing them on the LCD

This last experiment puts together the key concepts explored so far: modulo counters, shift registers, state machines, ROMs, combinational and sequential circuit blocks, design hierarchy and component instantiations, as well as the behavior of the PS/2 and LCD interfaces. Therefore, it is essential that you take your time to understand the Verilog source code, the interactions between design components, the state transitions and the effects of these state transitions on the data path registers, and the relation between the source code and the implemented hardware.

You have to perform the following tasks in the lab for this experiment:

- understand the behavior of the FSM from the top module and verify if the design works correctly
- change the shift register structure to 16 data registers, so an entire line can be displayed at once

Write-up Template

COE3DQ5 – Lab #2 Report
Group number
Student names
McMaster email addresses
Date

Your evaluation is based on one take-home exercise. In addition to your report, you should submit ONLY the source files, i.e., Verilog, MIF and QSF files.

Exercise (3 marks) – In the refence designs from the lab only the lower memory half (i.e., address range 000h-0FFh) is used for storing the LCD codes of lower-case characters. First, extend the "MIF" file with the LCD codes such that the upper memory half (i.e., address range 100h-1FFh) stores the LCD codes for upper-case characters. Using a single-bit flag you can keep track the mode, i.e., *lower-case* or *upper-case*: the left-shift key on the PS/2 keyboard sets the mode to upper-case and the right-shift key resets it to lower-case. For the sake of simplicity, it is assumed that the mode affects ***only*** the letter keys, i.e., 'a' to 'z'. Note, when the left-shift key or the right-shift key are pressed, nothing should be displayed on the character LCD (i.e., they are used only for setting the upper/lower-case modes). Note also, if the left-shift and right-shift keys are pressed more than once in between typing two alphabet character keys, then only the effect of the last shift key that was typed is considered.

For the top line of the LCD, the characters are displayed as they are typed. For the bottom line of the LCD, an entire line is displayed at once after sixteen keys have been typed. After the entire bottom line has been displayed, all the green LEDs are lightened for five (5) seconds and the two *leftmost* 7-segment displays show the following. The *leftmost* 7-segment display shows letter 'd' (from detect) if the **first** character from the bottom line appears at least once in the top line (otherwise this 7-segment display is turned off). The 7-segment display next to the *leftmost* 7-segment display shows letter 'd' if the **last** character from the bottom line appears at least once in the top line (otherwise this 7-segment display is turned off). You can assume that the first and the last characters from the bottom line are alphanumeric characters ('0' to '9', 'a' to 'z' and 'A' to 'Z). Note however, while finding if a character from the bottom line appears in the top line, a lower-case character, e.g. 'a', and its corresponding upper-case character, e.g. 'A', are considered to be identical. After 5 seconds have elapsed, the green LEDs and the two *leftmost* 7-segment displays are turned off. Subsequently, both the top and the bottom lines are ***erased*** and the above-specified behavior is repeated from the beginning of the top line. Note, one intuitive way of erasing the LCD is to display the space character in each position on each line; however, a much simpler way is to issue a single instruction to the LCD whose 9-bit code is 9'h001. Note also, it is assumed that while the green LEDs are lightened and the character LCD is erased, the PS2 keyboard is not monitored.

You will need to handle one type of exception. If from the last time the character LCD has been erased any of the four push-buttons has been pressed four times in a row, then both the top and the bottom lines are erased immediately and the above specified behavior is repeated from the beginning of the top line. Assume that at power-up, or immediately after switch 17 has been deasserted, the character LCD was erased. Finally, take note that the 7-segment displays whose functionality has not specified above, as well as all the red LEDs, can be used for debugging purposes, as you see fit.

Submit your sources and in report describe your reasoning. Your report is expected to be half-a-page and under no circumstances it should exceed a full page.

VERY IMPORTANT NOTE:

This lab has a weight of 3% of your final grade. The report has no value without the source files. Your report must be in "pdf" format and together with the requested source files it should be included in a directory called "coe3dq5_group_xx_takehome2" (where xx is your group number). Archive this directory (in "zip" format) and upload it through Avenue to Learn before noon on the day you are scheduled for lab 3. Late submissions will be penalized.