3DQ5 Lab 2 Exercise Report

Jack Wawrychuk 400145293

Minhaj Shah 400119266

The first step to complete this exercise was to extend the ROM memory using the provided tables in the lab document to include upper case letters. We used a flag called *case_flag* to keep track of the most recent shift key pressed for both the right and left options. The actual flag was not updated until the next key press after the shift key, so the LCD code for the key, before the shift key was pressed, was not modified. A register called *caps_check* is used to store the positions of capital letters. If the *case_flag* is 1 when a PS2 code is shifted into *data_reg*, then a 1 is stored in the corresponding location in *caps_check*. When the bottom line is printed out further along in the code, *caps_check* will be indexed at the same index as the register having its PS2 code converted, ensuring that a capital letter LCD code is returned.

The code for storing the PS2 was in an else statement after the shift key detection, so that the PS2 code from the shift keys aren't stored in the data registers or displayed on the LCD. When the first key is pressed, the PS2 code gets stored in the variable that sends the PS2 code to be converted to an LCD code. The PS2 code is also stored in a variable called *top_line*. This is used when the bottom line is displayed to check if the top character is repeated in the bottom line. A flag is also raised to tell the code that the first character has been displayed on the top row, and no other ones should be. Finally, the state is changed to WAIT_ROM_UPDATE so that the converted LCD code actually gets displayed. The state then returns to idle to wait for the next key press. For the next 16 key presses nothing gets displayed on the LCD, the PS2 codes just gets loaded into *data_reg* instead. Once all 16 registers are full, a flag called mode is raised to tell the code that the bottom line needs to be displayed, and the state is changed to the ISSUE_CHANGE_LINE state.

Since the mode flag now has the value of 1, after the line is changed the state will transition to the FINISH_INSTRUCTION state in order to print off the bottom line. In each cycle a 0 is shifted into the least significant register, while the most significant register gets shifted into the display variable that converts the PS2 code to LCD code. An if statement also checks if the PS2 code in the display variable is the same as the one stored in the *top_line* variable, and increments the *repeat_counter* if this is true. The state then transitions to the WAIT_ROM_STATE to send the converted LCD code from the display variable to the LCD. This repeats 16 times until the *data_reg* register is empty. The mode flag is set to 0 so that the top line will be printed next, the *caps_check* register is reset, two flags are raised, *repeat_delay* and *last_letter*, and the state is transitioned to WAIT_ROM_UPDATE. The *repeat_delay* flag is used to tell the seven segment display that the *repeat_counter* register has finished counting and can be displayed, and the *last_letter* variable ensures that the bottom line doesn't get cut off.

After the state transitions from WAIT_ROM_STATE to ISSUE_INSTRUCTION and displays the final character on the LCD line, the *last_letter* flag is used to tell the state to transition to a state called S_DELAY. In our initial code, the LCD was being sent instructions too frequently and as such would not execute all of them. The delay state is used in this case to implement a 1us delay to allow the LCD controlled to finish the previous instruction. This delay is only implemented once after each new bottom line is displayed on the LCD. Once the delay state finishes its 1us delay, the state is then transitioned to the change line instruction, to prepare the circuit for the next key press. Once a new key has been pressed to be displayed on the top line, the *repeat_delay* flag Is lowered to turn off the seven segment display.