Memo to: Randy Larimer
From: Erik Andersen
Date: February 19, 2015
Regarding: EELE 465, Lab1 Project – Keypad and Time-Varying Patterns on Eight LEDs

Summary:

The purpose of this lab was to display four distinct patterns on eight LEDs using the MC9S08QG8 microprocessor and a breadboard with a custom built circuit. The four different patterns will be selected based on which button is pressed on the keypad. Three patterns will be time varying and repeating sequences, one will be static.

Preliminary Solutions:

Developing a flowchart gave us a place to start from when it came to building the keypad read write code. The solution involved creating a new file called keypad read write within the sources folder of the project that contained all the code for reading from the keypad. The startup code jumps to the keypad read write subroutine which then after a long process of determining which keypad button was pressed that eight bit code is stored into memory. Based on what that eight bit code is the main code will branch to the correct pattern subroutine. Each pattern subroutine includes the correct macro in the project headers folder of the project in the derivative file to write the correct pattern to the LED's.

Setup:

The setup involved interfacing the MC9S08QG8 microprocessor with the desktop computer by USB cable to be used for downloading the firmware to the processor for debugging and wiring the keypad, LED array and DEMO9S08QG8 onto the breadboard. The breadboard circuit contains three d-flip-flop packages, one bus transceiver package and one de-multiplexer package. The d-flip-flops and bus transceiver are used to read the keypad press store it and then write the correct pattern to the LED's based on the key pressed. All instructions from the MC9S08QG8 microprocessor go through the de-multiplexer before they go to the three d-flip-flops and the bus transceiver. The reset button on the MC9S08QG8 microprocessor was also enabled.

Solution – Keypad Read Write:

First, set the PTBDD register to an output and store binary number 11110111 into the correct memory address in order to read the first row of the keypad. Then, logical shift left that binary number in order to test the next row of the keypad if the first row of the keypad had no press. Next, check to make sure all rows have been tested if not reinitialize memory address sixty until all rows have been checked. Once all rows have been checked branch to the roughly one second delay loop which was determined by using a stop watch and guessing. After the delay loop branch to the re read subroutine to recheck the button press to verify that it is still pressed after one second. Lastly, store the pattern to the correct memory address and based on the pattern stored to memory branch to pattern subroutine A, B, C or D to load to PTBD in order to drive the LED's. In each pattern subroutine read in the correct macro which contains the actual code to implement the pattern requested.

Solution – Pattern A:

First, load the upper byte of PTBD with 1010 (1=LED ON, 0=LED OFF) while selecting the upper d-flip-flop. Then, set the rising clock edge. Next, load the upper byte of PTBD with 1010 while selecting the lower d-flip-flop. Again, set the rising clock edge. Lastly, return to the beginning of the main code file.

Solution – Pattern B:

First, load the binary number 11111111 into the correct memory address then XOR that binary number with 10000000 and store the result 01111111 into two different memory addresses. Then, logical shift one of

those addresses four times from right to left by loading that address into PTBD at the rising edge of a clock, but not before setting bit 0 of that address to a 1. Now, the other address containing 01111111 is AND with 11110000 with the result stored in another address. Then that address is stored into PTBD at the rising edge of a clock. These two different address are stored into PTBD to allow the correct d-flip-flop to be selected at the right time. Additionally, logical shift right the binary number 10000000 to get the LED that is off to travel across the LED array and reset the mask to10000000. Lastly, return to the beginning of the main code.

Solution – Pattern C:

First of all pattern C is divided into two different macros. One where the two lit LED's in the middle of the array travel away from each other and one where the two lit LED's in the middle of the array travel toward each other. Similarly, pattern C is implemented very similar to how pattern B is implemented. Pattern C has two bit masks instead of one for the two macros. For, the macro where the LED's travel toward each other bit mask C logical shifts right and bit mask C2 logical shifts left whereas the macro where the LED's travel away from each other bit mask C logical shifts left and bit mask C2 logical shifts right. The logical shifting of the bit masks is how the LED's travel across the array. Lastly, return to the beginning of the code.

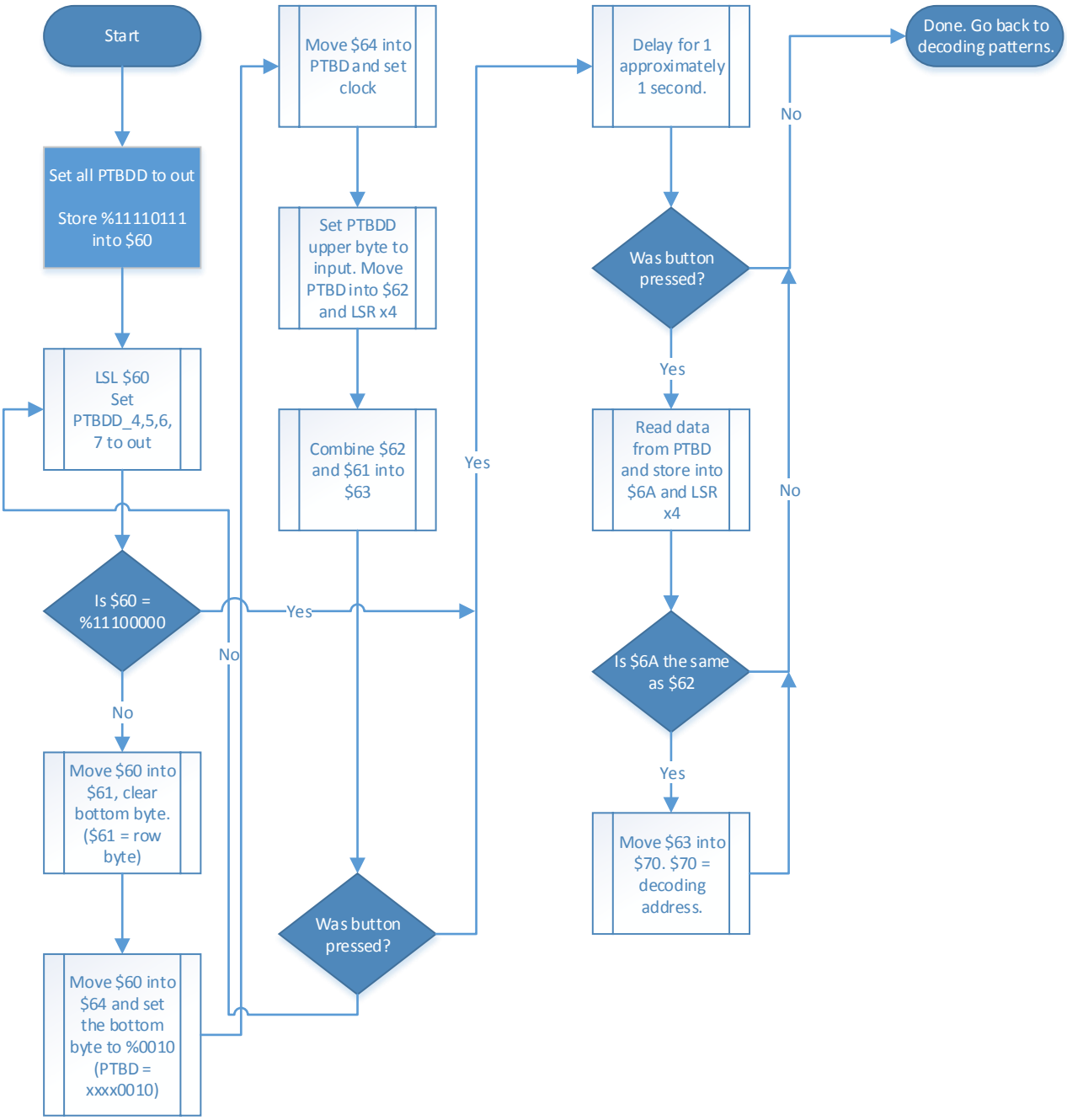
Solution – Pattern D:

To begin with the implementation of pattern D involved the creation of a lookup table in the main code as well as a counter value. The counter was built in a subroutine and the lookup table values set to PTBD were called in the macros. So, based on the value of the counter it branches to the correct subroutine that calls the right macro that assigns the right values from the lookup table to PTBD. Once all the values of the lookup table have been called in the subroutine in the correct order the counter is reset back to its original value so that the sequence can repeat.

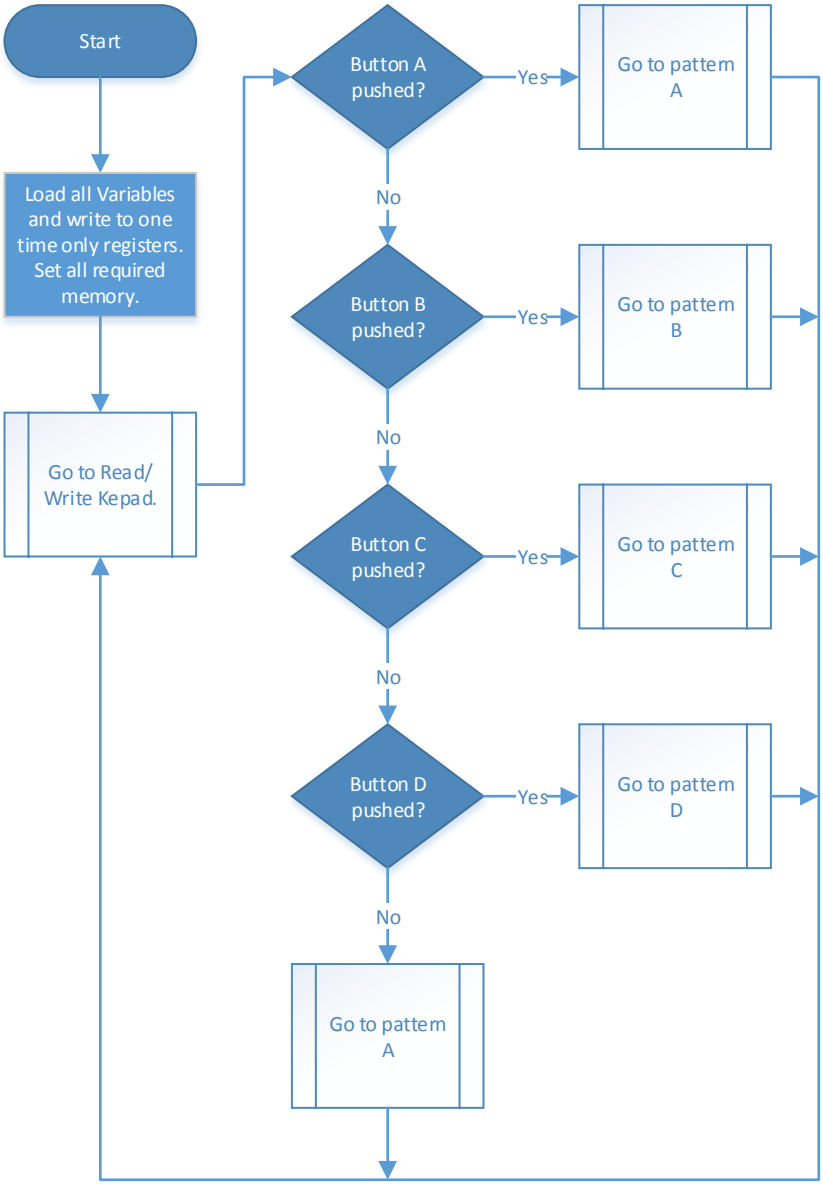
Final Comments:

While performing this lab, many problems and errors in the code were encountered. Some took multiple hours to troubleshoot and debug. The most bothersome and frustrating was figuring out and understanding how design the solution to read a key press from the keypad. For example, how to store the row and column information of a key press in one memory location. The construction of the breadboard circuit and the implementation of the four patterns was relatively straight forward.

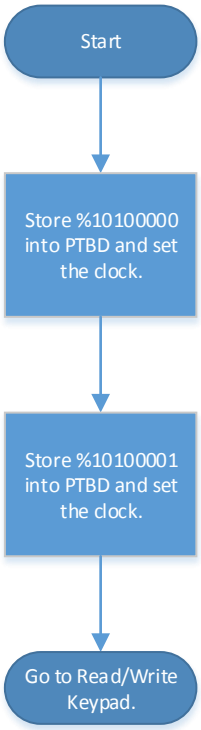
Reading from Keypad Flowchart



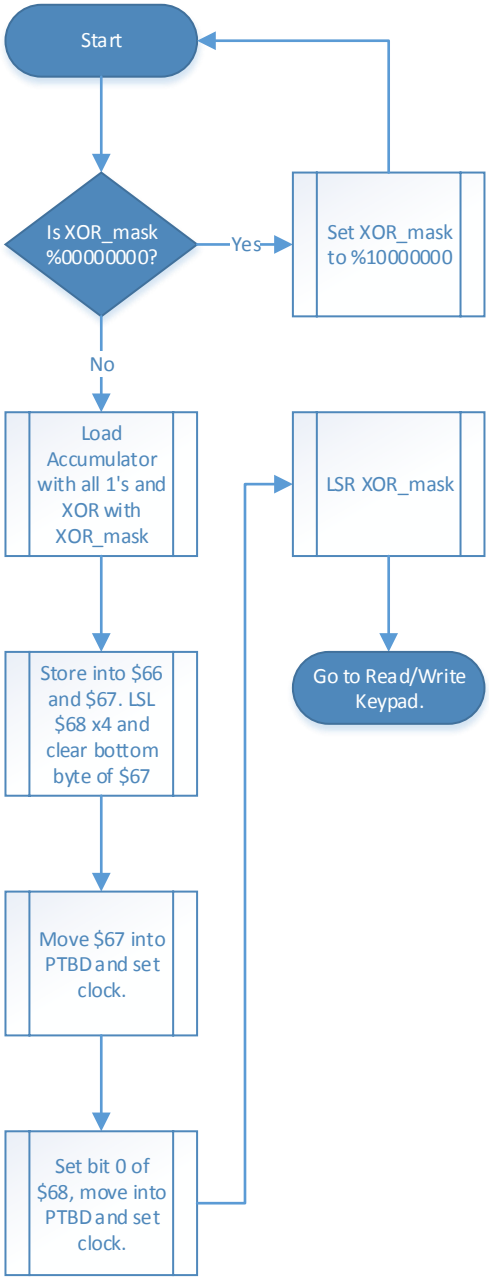
Main.asm Flowchart



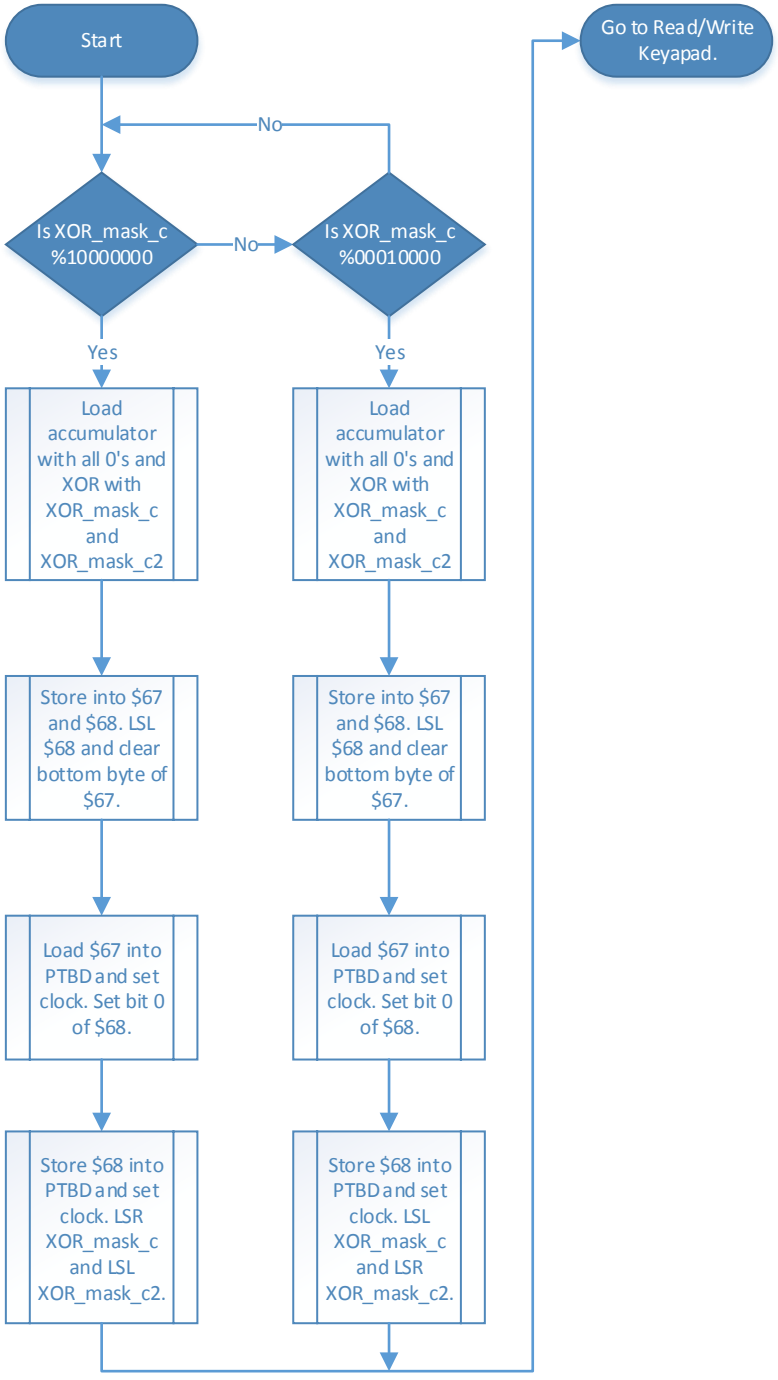
Pattern A


Start
Store %10100000 into PTBD and set the clock.
Store %10100001 into PTBD and set the clock.
Go to Read/Write Keypad.

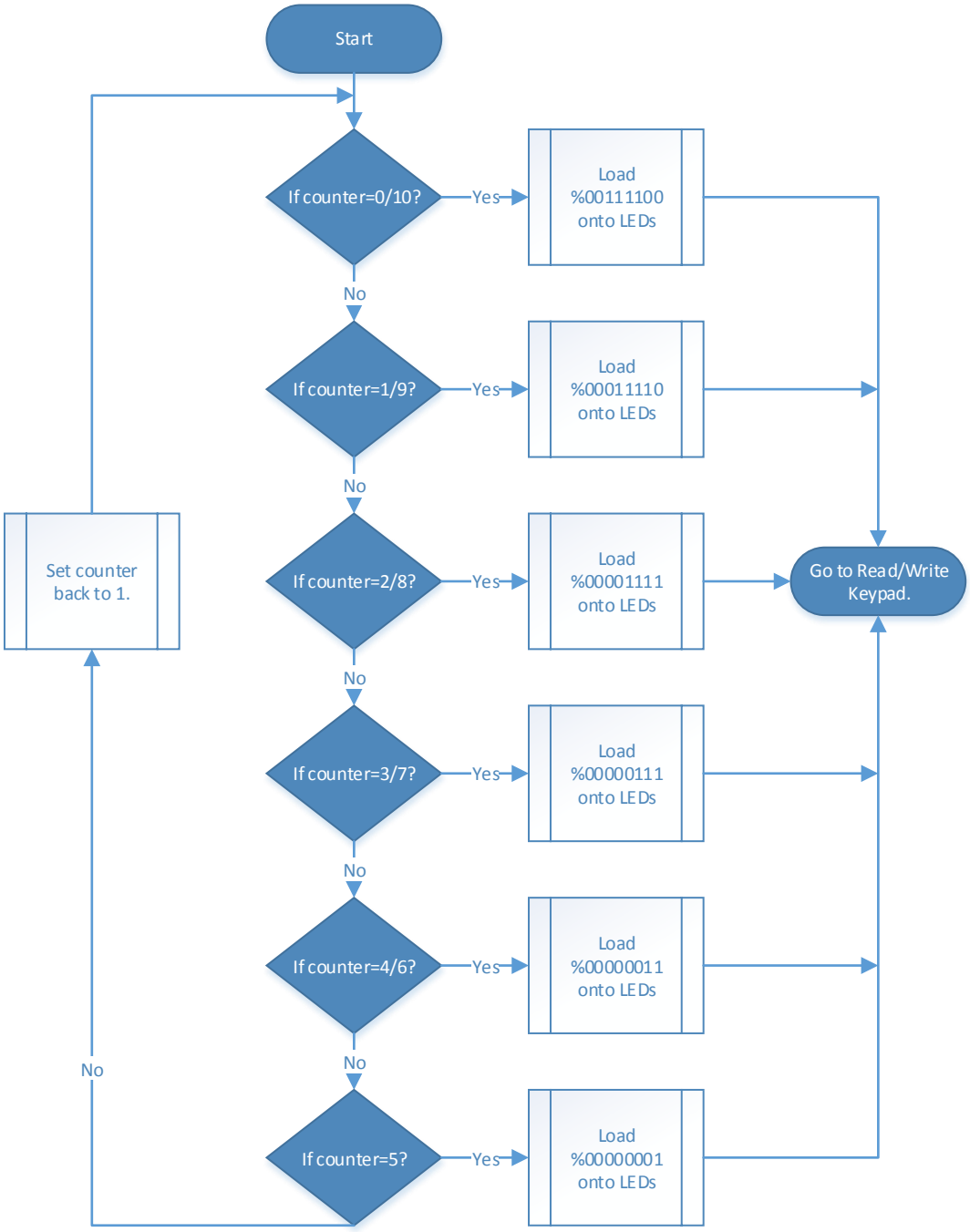
Pattern B


Start
Is XOR_mask %00000000?
Yes
Set XOR_mask to %10000000
No
Load Accumulator with all 1's and XOR with XOR_mask
Store into $66 and $67. LSL $68 x4 and clear bottom byte of $67
Move $67 into PTBD and set clock.
Set bit 0 of $68, move into PTBD and set clock.
LSR XOR_mask
Go to Read/Write Keypad.

Pattern C


Start
Go to Read/Write Keyapad.
No
Is XOR_mask_c %10000000
No
Is XOR_mask_c %00010000
Yes
Yes
Load accumulator with all 0's and XOR with XOR_mask_c and XOR_mask_c2
Load accumulator with all 0's and XOR with XOR_mask_c and XOR_mask_c2
Store into $67 and $68. LSL $68 and clear bottom byte of $67.
Store into $67 and $68. LSL $68 and clear bottom byte of $67.
Load $67 into PTBD and set clock. Set bit 0 of $68.
Load $67 into PTBD and set clock. Set bit 0 of $68.
Store $68 into PTBD and set clock. LSR XOR_mask_c and LSL XOR_mask_c2.
Store $68 into PTBD and set clock. LSL XOR_mask_c and LSR XOR_mask_c2.

Pattern D


Start
If counter=0/10?
Yes
Load %00111100 onto LEDs
No
If counter=1/9?
Yes
Load %00011110 onto LEDs
No
If counter=2/8?
Yes
Load %00001111 onto LEDs
No
If counter=3/7?
Yes
Load %00000111 onto LEDs
No
If counter=4/6?
Yes
Load %00000011 onto LEDs
No
If counter=5?
Yes
Load %00000001 onto LEDs
No
Set counter back to 1.
Go to Read/Write Keypad.