Lab Assignment # 5: Connect Four

Due Date: Week of October 12st, 2020,
half an hour before your lab session starts
on myMason (Blackboard)

Objectives:

- To gain more experience programming the MSP430 in C.
- To become familiar with coding state machines.
- To become falimiar with debouncing buttons using timers.

Parts:

This lab is based on your setup from Lab 4. Please make sure that you have the following parts before you start the lab.

- Setup from Lab 4
  - MSP-EXP430FR6989 LaunchPad
  - LED Matrix
  - 3 x SN74HC595 8-bit shift register
  - ULN2803A Darlington transistor array
  - 33 Ω Resistor Network
  - 100 Ω Resistor Network
  - 4 push buttons
  - Pin headers
  - Rainbow ribbon cable with headers

No new hardware will be used in this lab.

Introduction:

In this lab you will be using the setup from lab 4 and expand the final code to a complete Connect Four game. Scrolling text will announce the title of the game. Then two users can play the game, one picks the color green, the other the color red. The users take turns in selecting columns to drop their game pieces which then fall down, occupying the lowest empty space. After each drop, the program will check automatically if four game pieces are connected in a horizontal, vertical, or diagonal line. If so, the program will highlight the four connected pieces and then announce the winner.

The new program will:

- Scroll text in 3 possible colors on the LED matrix.
- Use timers to time scrolling text, dropping of game pieces, and flashing of the four connected pieces.
- User timers to debounce the buttons.

The final result of this lab is shown in Figure 1.

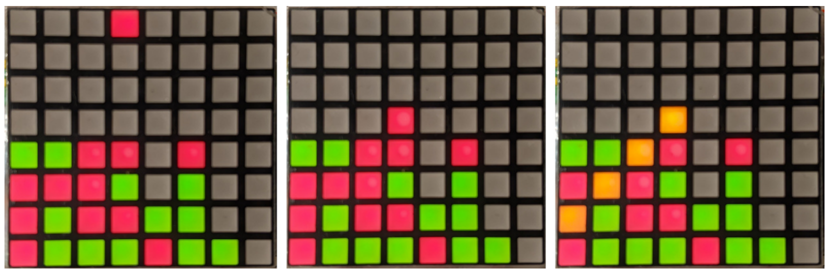
Figure 1: From the left: Red piece ready to drop, red piece dropped, highlight connected four.

Background:

Scroll Text and Font

In the lecture we have seen how a 5x7 font can be defined column based. The 5 refers to the number of columns, the 7 to the number or rows. Therefore a 5x7 font uses a 5x7 pixel matrix. Figure 2 illustrates this. The first column, i.e., the left most column, is defined as '0x7E' which in binary becomes '`0111 1110`'. The MSB of this 8-bit definition is always 0 as this font has only 7 rows. The LSB describes the pixel in the top row.

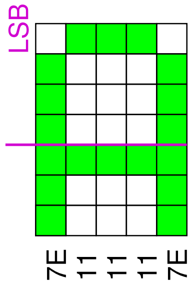

Figure 2: Column based definition of the character 'A' of the 5x7 font.

In order to achieve a scrolling text, each letter that we want to display has to be shifted into the array that our LED matrix shows, one column at a time. The timing with which this scrolling happens is independent of the refresh timing of the LED matrix.

Checking if Four are Connected

The game Connect Four is one as soon as any player manages to get four game pieces next to each other, either vertically, horizontally, or diagonally. Hence, we need a function that can check if this was achieved. This function has to run each time a game piece has dropped. The checking should be based on the most recent dropped piece as it has to be one of the four connected ones. That leaves us with the following possibilities. The row on which the last piece ended is called the *current row.*

Before a game piece is dropped, it can be moved between column 0 and 7. This happens on row 0. The value of row 0 of the active matrix, i.e., the green matrix if its green's turn or the red matrix if its red's turn, can be used as a mask. Let's call this mask *current column.*

- **Vertical**

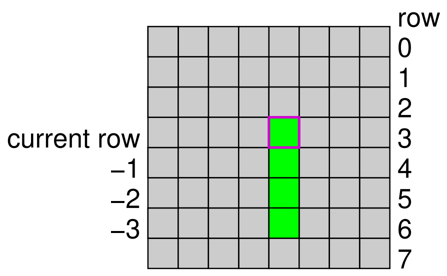

Figure 3: Four Connected vertically

This is the easiest case to check. If three rows below the current row have the LED of the same color lit up in the same column as the current column, then its a win. The current row has to be on row 4 or less, as row 0 is the top most row of the LED matrix, otherwise there won't be three rows below.

- **Horizontal**

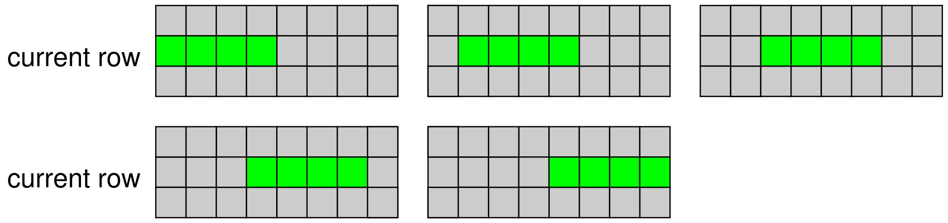

Figure 4: Five possibilities for Four Connected horizontally

In this case it only these 5 possibilities have to be checked for the current row. It is not important in which column that game piece was.

- **Diagonal going up**

A game piece can be the top and right most piece of a connection of four, the bottom left most piece or any in-between. That leads to four cases that have to be checked. Figure 5

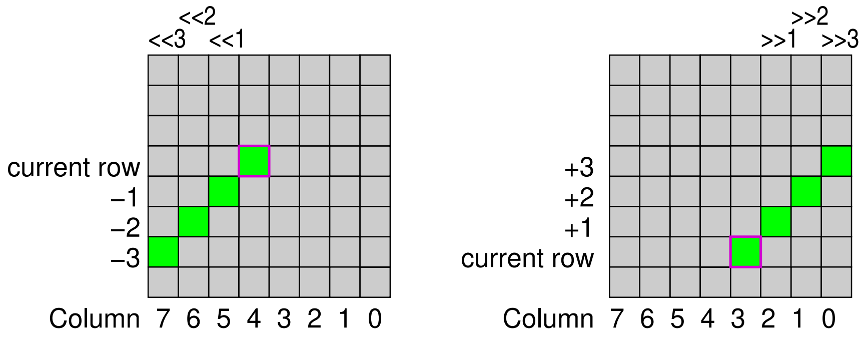

Figure 5: Two extremes for Four Connected diagonally going low left to up right

shows two cases. If the game piece is the top most, then it has to be on row 4 or less, just as we had to check for a vertical row. However, when we check the rows below, we have to shift the current column mask to the left each time. That also means that the game piece has to be on column 4 or less, i.e., less than column 5. As the current column is interpreted in C as a number we can simply check if the current column is less than '0x20', which is BIT5.

The other extreme is when the game piece is the bottom most. Now can't be above row 4 and we have to shift the current column mask to the right each time. In total these are 4 cases that have to be checked relative to the current row and current column.

- **Diagonal going down**

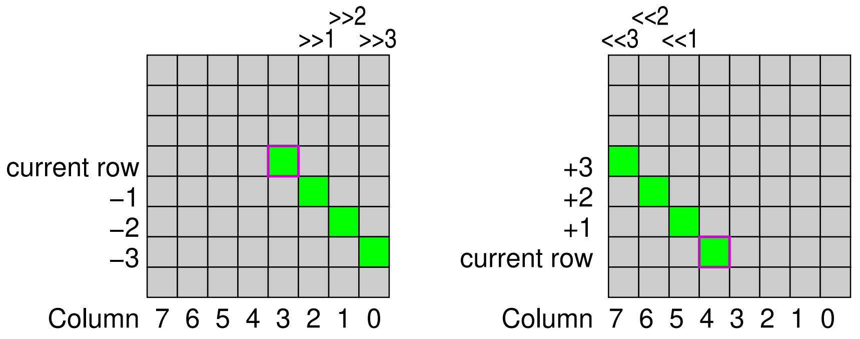

Figure 6: Two extremes for Four Connected diagonally going up left to low right

This case is similar to the *diagonal going up* case, just the other way around and consists again of four cases. If the game piece is the top most, then it still has to be on row 4 or less. However, when we check the rows below, we have to shift the current column mask to the right each time.

On the other hand, when the game piece is the bottom most, we have to shift the current column mask to the left each time. That also means that the current column has to be less than '0x20', i.e., less than BIT5. In total these are 4 cases that have to be checked relative to the current row and current column.

State machine

Due to the complexity of the task, a state machine is an elegant solution. Please familiarize yourself with the statemachine example from class called `statemachinePortISRtimerISR`. While the example descibes a *Moore Machine*, for this lab a *Mealy Machine* might be more suitable. An example is shown in Figure 7.

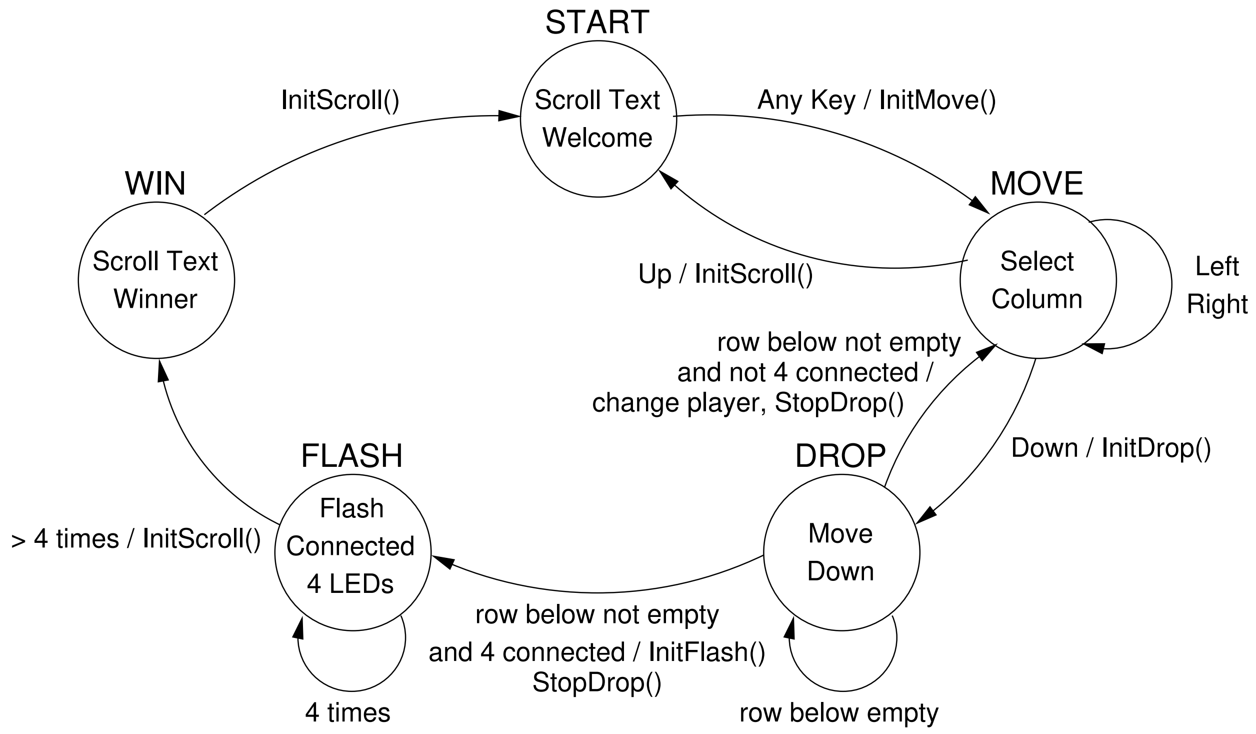

Figure 7: Example state machine for Connect Four

Code Walk Through:

Two files are provided for this lab. The file "fonts.h" contains three fonts of which we will be using the 5x7 font. The file "scrolltext.c" contains two functions namely "getnextchar" and "shiftchar".

In your main program, you have to declare two arrays of 8 unsigned characters for the LED Matrix, one to store which red LEDs are on, and one to store which green LEDs are be on. Each character is an 8-bit binary number describing a row. In addition to these two arrays we need tow more for the next character.

The function "getnextchar" loads one character from the 5x7 font into an array bit by bit. The font has an offset of 0x20 compared to the ASCII table as the first 32 characters in the ASCII table are not printable. The function uses two nested loops to iterate over the rows and for each row it iterates over the columns.

C code	Explanations
`void getnextchar(unsigned char *matrix, char c)`	
`{`	
`    unsigned char col=0;`	Start at column and row 0
`    unsigned char row=0;`	
`    unsigned char bit=0x01;`	Start with rightmost pixel as this corresponds to the top row of the matrix which is row 8
`    unsigned char pc= c - 0x20;`	Offset to ASCII is 0x20
`    while (row<8) {`	Iterate over 7 rows
`        while (col <5) {`	Iterate over 5 columns
`            if (font_5x7[pc][col] & bit)`	If the bit is set in the font
`                matrix[row] |= 0x01;`	set it in the matrix
`            matrix[row]=matrix[row] << 1;`	and shift the matrix to the next column.
`            col++;`	Increment the column of the font.
`        }`	
`        col = 0;`	Go back to the first column of the font.
`        bit = bit << 1;`	Now shift the bit mask to the next bit which corresponds to the next row.
`        row++;`	Increment the row of the matrix.
`    }`	
`}`	

The function "shiftchar" is best described using Figure 8. Each time we shift the rows of the array for the LED matrix to the left, we have to copy in the contents of the fifth column of the next character array. Then the rows of the next character array are also shifted.

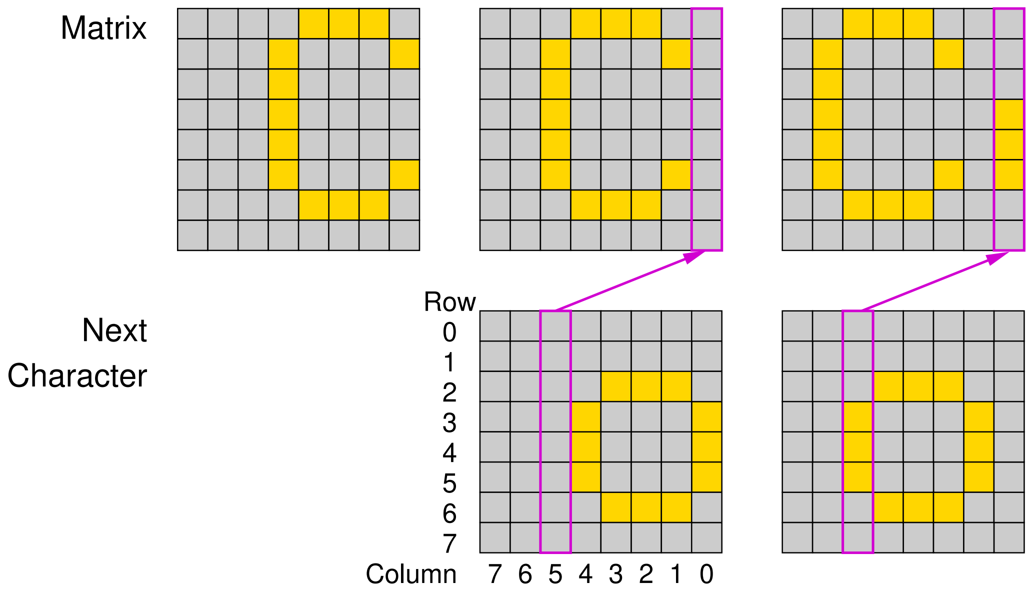

Figure 8: Example of how to scroll text

C code	Explanations
`void shiftchar(unsigned char *matrix,`	
`               unsigned char *nextchar)`	
`{`	
`   unsigned row=0;`	Start with row 0.
`   while (row<8) {`	Iterate over 7 rows.
`      matrix[row] = matrix[row] << 1;`	Shift the row of the array matrix one position to the left.
`      matrix[row] |= (nextchar[row]&0x20) >> 5;`	Copy bit from 5th column of the same row of nextchar into matrix.
`      nextchar[row] = nextchar[row] << 1;`	Shift the row of the array nextchar one position to the left.
`      row++;`	Move on to the next row.
`   }`	
`}`	

In-lab Exercise (week of October 5th):

1. Create a new "Empty Project" in Code Composer Studio and add your solution to lab-4 to it. Also add the provided "scrolltext.c' and "fonts.h" to the project.
2. Change the initialization with a test pattern from you code to initialize both, the red matrix and the green matrix with 0x00.
3. Then call the "getnextchar" function for a letter of your choice and only the green array.
4. Compile the program and see if it displays the letter in green.
5. Add the "getnextchar" call again for the same letter and the red array.
6. Compile the program and see if it displays the letter in Orange.
7. User your webcam and display this to the TA.
8. Lets use TB0CCR4 for scrolling. In the setup section of your code, enable interrupts on TB0CCR4.
9. Make sure that your program is in low power mode in the main while loop.
10. Add an interrupt service routine for timer B. The skeleton of this is shown below. In the interrupt for CCR4 toggle the red LED on board of your launchpad and increment the current value of TB0CCR4 by a value of your choice. Don't forget to add the initialization steps for the port the LED is connected to, to the setup section of your code.

C code	Explanations
`#pragma vector = TIMER0_B1_VECTOR`	
`__interrupt void T0B1_ISR(void)`	
`{`	
`    switch(__even_in_range(TB0IV,14))`	
`    {`	
`      case  0: break;`	No interrupt
`      case  2: break;`	CCR1
` ...`	now you can complete the rest
`    }`	
`}`	

11. User your webcam to show the blinking LED to the TA.

12. Add the command `__low_power_mode_off_on_exit();` to the ISR for TB0CCR4.

13. Now initialize two global strings, one for red and one for green, with some text of your choice. Text that appears in both strings at the same location will be shown in orange on the LED matrix.

14. Modify the "getnextchar" functions to initialize the next character arrays instead of the arrays for the red and the green matrix.

15. In the main while loop, after the low-power-mode command, add the code to shift the next column from next character into the red and green arrays. If the next character has been completely shifted into the red and green arrays for the LED matrix, get the next character. If the message has completed, start all over again. This code will be executed once each time the timer TB0CCR4 has an event.

16. Once you managed to get scrolling text in red, green, and orange, use your webcam to show this to the TA.

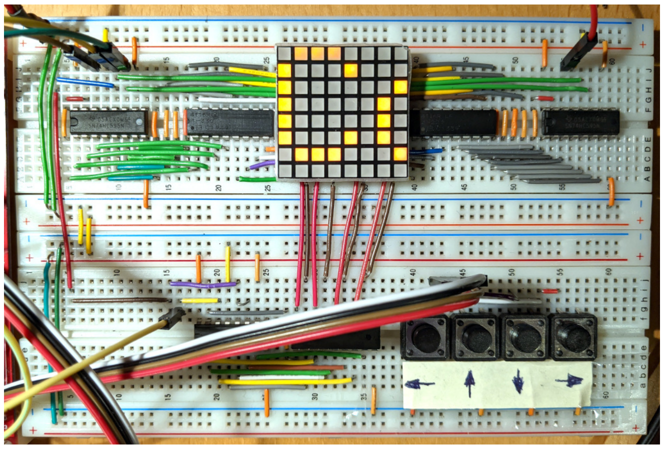

Figure 9: Scrolling text on the LED matrix

Exercises:

In this exercise, you will be turning your code from the in-class exercise into a complete *Connect Four* game. Here is a description of the final outcome. Also have a look at the state machine in Figure 7 and follow along. You can also see an example video of how the game should work on our myMason page in the Labs section.

1. When the board is connected to power, it should display a welcome message as scrolling text. You are free to choose the content of this message. However, it should contain text in red, green, and orange.
2. When any button is pressed, the LED matrix is cleared and the green player gets a turn.
3. The current player can now move the game piece to the desired column by pressing the ‘left’ and ‘right’ buttons. With these the player selects which LED in the top row, row 0, lights up.
4. When the player presses the ‘up’ button the game will end and it will go back so step 1
5. When the player presses the ‘down’ button the game piece will drop.
6. The game piece will drop one row at a time at a speed that is easy to observe. The LED of the current color in the current row is turned off, and the LED of the current color in the row below is turned on. Then the current row is incremented. This is repeated until the game piece is in the bottom most row, i.e., row 7, or the same column in the row below already contains a game piece.
7. When the piece has reached its final position, the program will check if it makes a connection of four game pieces of the same color. If so, the current player won. If not, the other player gets a turn in step 3.
8. Once a player has won, the four connected game pieces will flash orange and then back to the original color four times at a speed that is easy to observe.
9. Now, the display should be cleared and a message announcing the winner should be displayed as scrolling text.
10. Once the message has complete the game starts all over again with step 1.

For a good game experience, all buttons should be debounced. Have a close look at the posted “statemachinePortISRtimerISR.c” code. It contains not only a state machine example, but also the code necessary to use a timer for debouncing buttons. In this lab you can use timer TB0CCR5 for the debounce as it is not needed for the LED matrix refresh and timer B0 is already configured. This leads to the following timer assignments. As you can see, we are only using Timer B0. Our MSP has several more timers which can be used for additional functionality, such as sound.

Port	Timer	Interrupt	Explanation
N/A	TB0CCR0	Yes	LED Matrix timing
P3.6	TB0CCR2	No	Column Clock with TB0CCR0
P3.7	TB0CCR3	No	Column Done and Row Clock
N/A	TB0CCR4	when needed	Blink Delay / Scroll Text Delay
N/A	TB0CCR5	when needed	Debounce Delay
P2.7	TB0CCR6	No	Row Initialization

When writing the program, code one step at a time and test it. Create a state machine with all five states but have it stuck in the START state. Move your code for the scrolling text into the start state and see if it still works. Save the code.

Then add code to move from the START state to the MOVE state on a button press. When going to the MOVE state have a function clear the LED matrix and stop the interrupt on the CCR for the scroll text delay (TB0CCR4). In the MOVE state display a single green LED on the top row and have the user move it with the left and right buttons. You already had part of that code in Lab-3. Pressing 'up' should go back to state START. Ignore the 'down' button for now and see if the code works. Save this code under a new name.

Now add the code to debounce the buttons and test it. Save this code again under a new name.

Continue in this manner. Each time after your code works, keep a copy before you make further modifications, so that you can always get back to a working state.

Questions to be answered in the report:

- How much time in total did you spend on lab-5.
- How much time in total did you spend on labs 2–5.
- What was the most difficult part of labs 2–5.
- Was the outcome, hopefully a fully functioning game, worth the effort.
- As this lab is due after the midterm, do you think working on the labs helped you understanding the material.

Points to be covered in the demo:

- Scroll welcome text.
- Moving game pieces all the way left and right. Show that each button press is exactly one step and that the buttons don't bounce.
- Dropping game pieces, making sure they stack properly.
- Blink connected four. Play the game several times to show that your code can detect four connected game pieces in all four directions described in background section.
- Scroll text to announce winner. And return to the welcome text.
- Show that a game can be quit in the 'MOVE' state by pressing the 'up' button.