Meghan McLaughlin ECE:3360– Spring 2019 Partner: Dylan McNamara

Tic-Tac-Toe Game

Introduction

This project implements the game Tic-Tac-Toe using a Nokia 5110 cell phone LCD display. The players are able to select their token placement using an attached keypad. The LCD display will update to reflect the placed tokens if the placement was valid. If the game is won by a player, a message will be displayed, and a new game will begin. The players may also choose to restart the game by pressing the 'New Game' button on the keypad. A potentiometer (to dim the display), voltage regulator, and logic level shifter have been included in the design to interface with the LCD.

Motivation

This project was inspired by the games included on the Nokia 5110 and 3310 cell phones. We designed and implemented a simple Tic-Tac-Toe game which interfaces with a keypad reminiscent of those on the Nokia 5110/3310. This type of game is ideal for the simple keypad interface and small display size.

Goals and Specifications

The goal of this project was to create a device which would allow two users to play tic-tac-toe. This required that we have some method to display the status of the game to the user, as well as some intuitive method for the user to provide input.
An additional goal was to provide logical evaluation of the game to the user. This included switching between player turns, verifying if the selected placement is valid, recognizing when the game is won, and providing a method to restart the game without having to power cycle the microcontroller.

Implementation

Overview

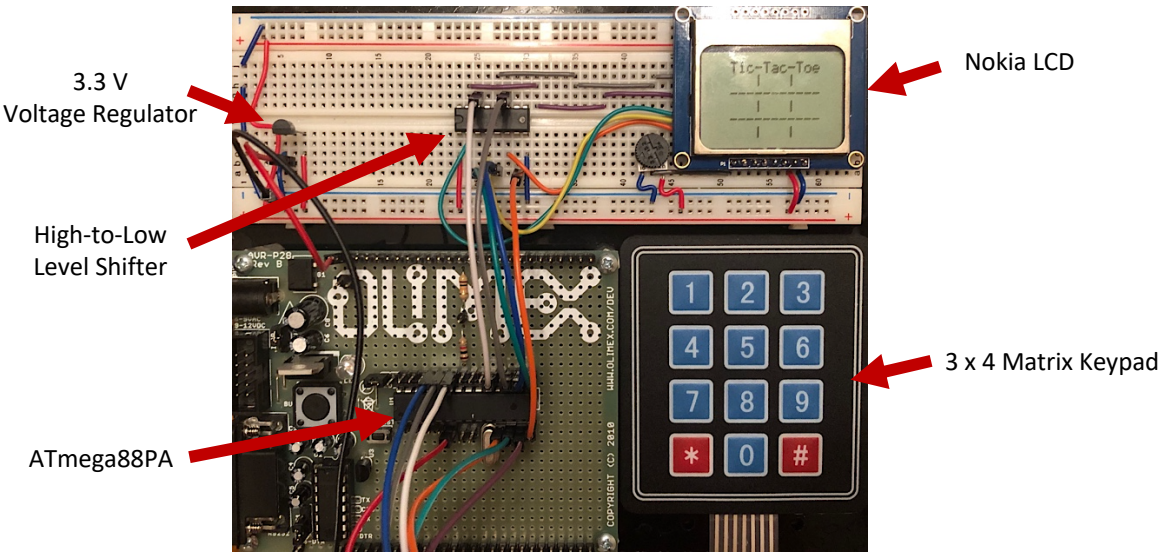
To implement our project, we chose to use a Nokia 5110/3310 LCD unit as the user display, and a 3 x 4 matrix keypad as the interface for user input. Interfacing these components with the ATmega88PA and development board required that we also incorporate a 3.3 V Voltage regulator, and High-to-Low logic level shifter. This hardware will be described in further detail in the sections below.

Hardware

Display

We selected the Nokia 5110/3310 LCD display for our project. This LCD was reclaimed and refurbished from the Nokia cell phones released in the early 2000's. This display operates at 3.3 V, while our development board was in 5 V operating mode. The development board has a jumper select that allows the entire board to operate at 3.3 V, however that would cause the

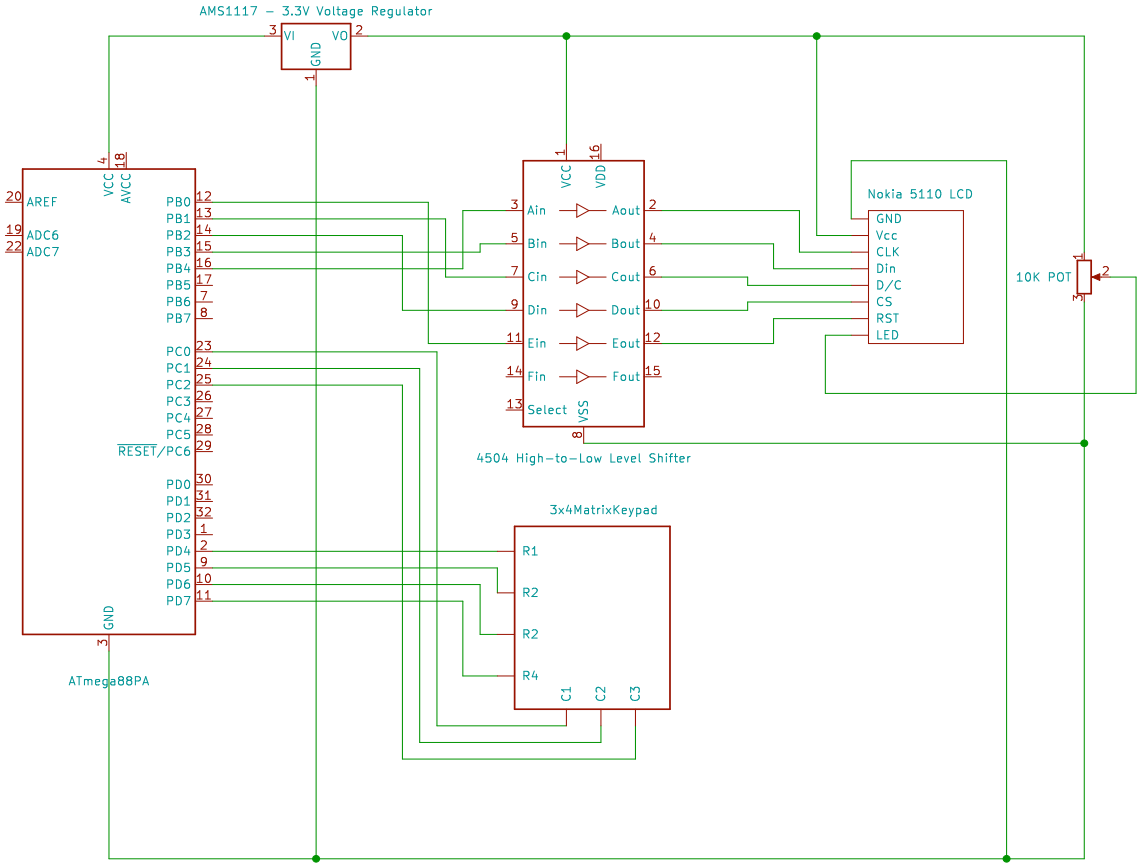
reset generator to be triggered and subsequently reset the board. Rather than choosing this route and unsoldering the reset generator, we chose to use a 3.3 V voltage regulator with a High-to-Low voltage level shifter.
The LCD uses SPI to communicate with the microcontroller. There are 5 connections from the LCD to the microcontroller, Clock/Serial Clock, Data In, Data/Command Select, Chip Select, and Reset. Each of these was connected through the level shifter to adjust the logic from the 5 V output of the microcontroller to the 3.3 V required by the LCD.
The LCD was connected to ground and V_{CC} (3.3 V). We also chose to connect a 10 kΩ potentiometer to the LCD LED input to provide an adjustable backlight.



Project Hardware

Keypad

The keypad selected was a 3 x 4 matrix keypad. The keypad has a single wire for each of the four rows and three columns. The four row pins were wired as inputs to a single microcontroller port, D[7:7]. These input pins also correspond to Pin Change Interrupt pins 20:23. The three column pins were wired as output pins on port C[0:2].



Hardware Schematic

Software

Initialization

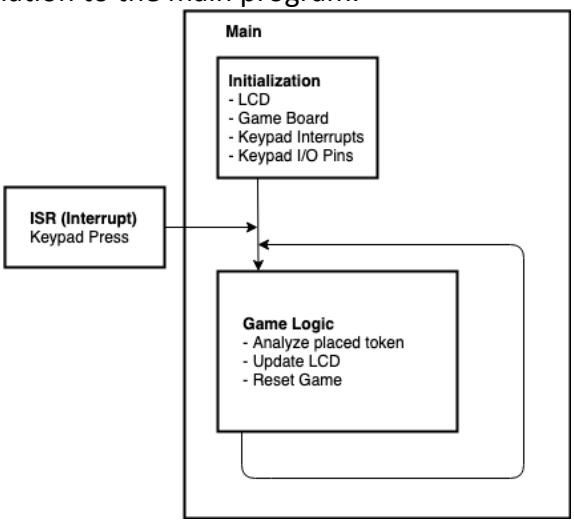
To begin our program, we initialize the connections between the microcontroller and the external components. To initialize the LCD, we use a method provided in our chosen library. To initialize the matrix keypad, we first enable the internal pull-up resistors, configure pins D[4:7] as input, then write logic 1 to those pins to complete the enabling of the pull-up resistors. We also set pins C[0:2] as outputs. After configuring the direction of the connected pins, we also set up pin-change interrupts on the row input pins. This allows us to detect when a button on the keypad has been pressed. Additionally, to initialize the program we set up the required variables for managing the game logic, including an array to represent the game board spots and a Boolean value for determining the current player's turn.

Interrupt Service Routines

As indicated in the software flow diagram below, the main driver of the program is the interrupt service routine triggered by a keyboard button press. The main program waits in an infinite loop and does not take any action unless a button is pressed. The interrupt service routine recognizes when a button has been pressed, decodes the exact value of the button that has been pressed, and relays that information to the main program.

To decode the specific key that is pressed, we rotate through the possible columns – setting one column low and the other two high. When the column that is being pressed is pulled low,

the contact on the matrix keypad will pull current from the row pin (with pull-up resistor to V_{cc}). We then check each bit in the input register. If the current column is correct, one of the pins in the input register will be low, as the current is instead being pulled to the low column pin. By iterating through all of these options, we are able to determine the exact button that is being pressed, and relay that information to the main program.



Software Flow Diagram

Main Program

We also included logic in our software that would compare the currently pressed button to the game board and verify that the selected placement is valid. If the selection is not valid, no action will be taken, and it will remain that player's turn. We also included logic to analyze the game board after each placement to check if the game has been won by the player that just placed a token.

The screen will update accordingly based on the previous action. The placed token will be displayed on the LCD, and a string at the top of the screen will display "X/O's Turn". If the game has been won, the LCD will display "X/O Wins!".

Experimental Methods

Our initial implementation design was to implement a pair of IR remotes, one for each user. When attempting to interface these with our board, we ran into several issues.

The initial plan was to have two separate IR sensors pointing in opposite directions. We wired LCDs to the sensors to test the signals being received. Unfortunately, there was interference between the two IR sensors, even with them pointing in opposite directions. At that point, we chose to continue our design using a single IR sensor (assuming the two players would be nice and not take the other player's turn).

We found several libraries online claiming to successfully implement IR receiving, however we were not able to find one that we were able to integrate with our design. In order to still make a functional product, we chose to switch the user input from an IR remote to a matrix keypad, which is significantly easier to interface with. While this did decrease the complexity of our project, it did allow for us to complete our desired project goals.

We were able to successfully interface with the matrix keypad. One issue that we had was that in the first row, each button would be decoded as being button “1”, i.e. if the user pressed “2” or “3”, it would always be decoded as a “1”. As a workaround to this, we chose to make the entire top row the “New Game” button and shift the nine buttons required for the game board to be rows 2-4.
As we used the keypad more and more, we also noticed a deterioration in keypad functionality. Several buttons (specifically in column 1) had issues being detected consistently. We used a multimeter to verify that the expected logic levels were appearing at the microcontroller pins. Based on this evidence and the fact that the pins do work sometimes, just not every time, lead us to the conclusion that it was a hardware issue with the internals of the keypad.

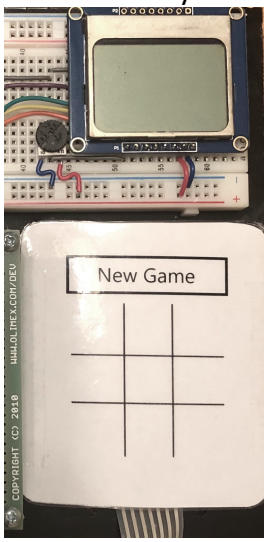
Results

The project successfully completes the following:

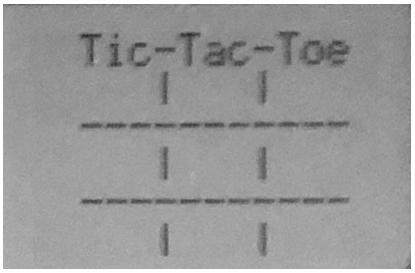
- Automatically switches between Player X and Player O and displays current turn to the user.
- Allows only valid token placement – a token cannot be played on a spot that is taken.
- Recognizes when a game has been won and displays the winner.
- Provides the option to start a new game and reset the game board at any time.
- LCD features a dimmable backlight.

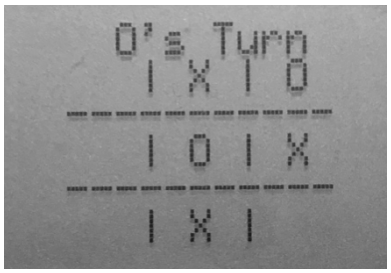
User Hardware

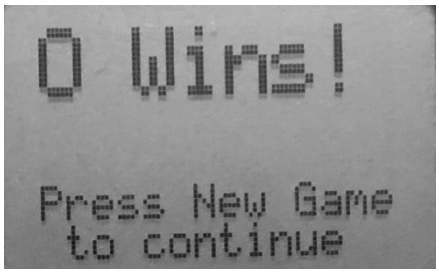
As shown in the image below, we created an overlay for the keypad to aid the user.



Game Screens







Discussion

Overall, we met all of our implementation goals.
In our initial project proposal, we proposed that we would use two IR remotes as the user interface. Throughout project development, we determined that we would not be able to successfully implement the IR receiving. We chose to switch our design to use a single matrix keypad instead. In the future, we could attempt to complete the original implementation using the IR remotes. This would improve the usability of the device.
After our project was completed, we found that there may have been a hardware issue with the matrix keypad. Single rows of the keypad became intermittently faulty and would not recognize user input on the first try. We believe that this issue was with the keypad itself rather than our implementation as all of the buttons do work, but some work more consistently than others. Unfortunately, we were unable to replace the keypad due to time restrictions.

Conclusion

We were able to successfully implement the physical hardware and corresponding game user interface. We believe our final product meets all of the desired features for a game of this kind. Completing this project incorporated several topics from this semester, including interrupt-driven programming, SPI implementation, and C programming.

References

Microchip Technology, “Low Quiescent Current LDO Voltage Regulator”, MCP1700, 2018.
Microchip Technology, “8-bit AVR Microcontroller with 4/8/16K8/16Kbytes In-system”, ATmega88PA, 2014.

Matrix Keypad, Adafruit [no datasheet or model number provided]

Nokia 5110/3310 LCD Library, https://github.com/ss2222/AVR-nokia-5110

Texas Instruments, “CD4050B CMOS Hex Inverting Buffer and Converter”, Sept 2016.

Appendix I: Source Code

```
/*
 * Project.c
 *
 * Created: 4/19/2019 12:24:42 PM
 * Author : mlmclaughlin & dtmcnamara
 */

#define F_CPU 8000000L

#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include <stdbool.h>
#include <avr/interrupt.h>
#include "nlcd.h"          // library for displaying to Nokia 5110 LCD – from
https://github.com/ss2222/AVR-nokia-5110
```

```
#define COL0 3*6
#define COL1 7*6
#define COL2 11*6
#define ROW0 1*8
#define ROW1 3*8
#define ROW2 5*8

/*
LCD INFO:
	48 px-rows
	84 px-col
	8 px/char (hight/row)
	6 px/character (width/col)
	6x14 characters
*/

// our functions
void board_int(LcdNokia *lcd_ptr);
void write_token(LcdNokia *lcd_ptr, bool is_x, uint8_t col, uint8_t row);
void write_player_turn(LcdNokia *lcd_ptr, bool x_turn);
void keypad_init();
void pin_change_init();
uint8_t getKeyPressed();

// Global Values
uint8_t key_val = 0xFF;
int board[9] = {-1,-1,-1,-1,-1,-1,-1,-1,-1};
bool x_turn = true;

ISR(PCINT2_vect){
	// Decode keypress
	key_val = getKeyPressed();
	// Set PORTC back to low
	PORTC = 0x00;
}

int main(void)
{
// Program Initialization
	struct lcd_nokia lcd;
	// Board is initialized as -1's. When tokens are placed, 0 = O token, 1 is X
token
	_delay_ms(500); // give the LCD time to power up
	lcd_nokia_init();
	lcd_nokia_clear(&lcd);
	board_int(&lcd);
	keypad_init();
	pin_change_init();

// End Initialization

// Begin Game Logic

	while(1){
		if (key_val == 0x01){
			// Pressed Row 1 - RESET/ NEW GAME
			for (int i =0; i <9 ;i++){
				board[i] = -1;
			}
			x_turn = true;
			lcd_nokia_clear(&lcd);
```

```
        lcd_nokia_set_cursor(&lcd, 0, 0);
        lcd_nokia_write_string(&lcd, "   New Game   ", 1);
        lcd_nokia_render(&lcd);
        _delay_ms(1000);
        board_int(&lcd);

    }  else if (key_val == 0x04){
        // Pressed "4" i.e. Row 0 Col 0;
        if (board[0] == -1){
              board[0] = x_turn;
              write_token(&lcd, x_turn, COL0, ROW0);
              x_turn = !x_turn;
              write_player_turn(&lcd, x_turn);
              lcd_nokia_render(&lcd);
        }

    } else if (key_val == 0x05){
        // Pressed "5" i.e. Row 0 Col 1
        if(board[1] == -1){
              board[1]= x_turn;
              write_token(&lcd, x_turn, COL1, ROW0);
              x_turn = !x_turn;
              write_player_turn(&lcd, x_turn);
              lcd_nokia_render(&lcd);
        }

    } else if (key_val == 0x06){
        // Pressed "6" i.e. Row 0 Col 2
        if(board[2]==-1){
              board[2]=x_turn;
              write_token(&lcd, x_turn, COL2, ROW0);
              x_turn = !x_turn;
              write_player_turn(&lcd, x_turn);
              lcd_nokia_render(&lcd);
        }

    } else if (key_val == 0x07){
        // Pressed "7" i.e. Row 1 Col 0
        if(board[3]==-1){
              board[3]=x_turn;
              write_token(&lcd, x_turn, COL0, ROW1);
              x_turn = !x_turn;
              write_player_turn(&lcd, x_turn);
              lcd_nokia_render(&lcd);
        }

    } else if (key_val == 0x08){
        // Pressed "8" i.e. Row 1 Col 1
        if(board[4]==-1){
              board[4]=x_turn;
              write_token(&lcd, x_turn, COL1, ROW1);
              x_turn = !x_turn;
              write_player_turn(&lcd, x_turn);
              lcd_nokia_render(&lcd);
        }

    } else if (key_val == 0x09){
        // Pressed "9" i.e. Row 1 Col 2
        if(board[5]==-1){
              board[5]=x_turn;
              write_token(&lcd, x_turn, COL2, ROW1);
```

```
                        x_turn = !x_turn;
                        write_player_turn(&lcd, x_turn);
                        lcd_nokia_render(&lcd);
                }

        } else if (key_val == 0x0A){
                // Pressed "*" i.e. Row 2 Col 0
                if(board[6]==-1){
                        board[6]=x_turn;
                        write_token(&lcd, x_turn, COL0, ROW2);
                        x_turn = !x_turn;
                        write_player_turn(&lcd, x_turn);
                        lcd_nokia_render(&lcd);
                }

        } else if (key_val == 0x0B){
                // Pressed "0" i.e. Row 2 Col 2
                if(board[7]==-1){
                        board[7]=x_turn;
                        write_token(&lcd, x_turn, COL1, ROW2);
                        x_turn = !x_turn;
                        write_player_turn(&lcd, x_turn);
                        lcd_nokia_render(&lcd);
                }

        } else if (key_val == 0x0C){
                // Pressed "#" i.e. Row 2 Col 2
                if(board[8]==-1){
                        board[8]=x_turn;
                        write_token(&lcd, x_turn, COL2, ROW2);
                        x_turn = !x_turn;
                        write_player_turn(&lcd, x_turn);
                        lcd_nokia_render(&lcd);
                }
        }

        // Check for a winner!
        // Possible Combinations 012 345 678 036 147 258 048 246
        bool winner = false;
        if((board[0] != -1) && (board[0] == board[1]) && (board[1] == board[2])){
                winner = true;
        } else if((board[6] != -1) && (board[6] == board[7]) && (board[7] ==
board[8])){
                winner = true;
        } else if ((board[3] != -1) && (board[3] == board[4]) && (board[4] ==
board[5])){
                winner = true;
        } else if((board[0] != -1) && (board[0] == board[3]) && (board[3] ==
board[6])){
                winner = true;
        } else if((board[1] != -1) && (board[1] == board[4]) && (board[4] ==
board[7])){
                winner = true;
        } else if((board[2] != -1) && (board[2] == board[5]) && (board[5] ==
board[8])){
                winner = true;
        } else if((board[0] != -1) && (board[0] == board[4]) && (board[4] ==
board[8])){
                winner = true;
        }  else if((board[2] != -1) && (board[2] == board[4]) && (board[4] ==
board[6])){
```

```
                        winner = true;
                }

                if(winner){
                        lcd_nokia_clear(&lcd);
                        lcd_nokia_set_cursor(&lcd, 0, 0);
                        if(x_turn){
                                lcd_nokia_write_string(&lcd, " O Wins!", 2);
                        } else {
                                lcd_nokia_write_string(&lcd, " X Wins!", 2);
                        }
                        lcd_nokia_set_cursor(&lcd, 0, ROW1+8);
                        lcd_nokia_write_string(&lcd, "Press New Game to continue", 1);
                        lcd_nokia_render(&lcd);
                        for (int i =0; i <9 ;i++){
                                board[i] = -1;
                        }

                }

                // reset key to invalid value
                key_val = 0xFF;

        }
// End Game Logics
}

void board_int(LcdNokia *lcd_ptr){
        char * arr[] = {
                "  Tic-Tac-Toe  ",
                "     |   |     ",
                "  -----------  ",
                "     |   |     ",
                "  -----------  ",
                "     |   |     "
        };

        for (int i = 0; i < 6; i++){
                lcd_nokia_set_cursor(lcd_ptr, 0, (8*i));
                lcd_nokia_write_string(lcd_ptr, arr[i], 1);
        }

        lcd_nokia_render(lcd_ptr);
        return;
}

void write_token(LcdNokia *lcd_ptr, bool x_turn, uint8_t col, uint8_t row){
        lcd_nokia_set_cursor(lcd_ptr, col, row);
        if(x_turn){
                lcd_nokia_write_string(lcd_ptr, "X", 1);
        } else {
                lcd_nokia_write_string(lcd_ptr, "O", 1);
        }
        lcd_nokia_render(lcd_ptr);
        return;
}

void write_player_turn(LcdNokia *lcd_ptr, bool x_turn){
        lcd_nokia_set_cursor(lcd_ptr, 0, 0);
        if (x_turn){
                lcd_nokia_write_string(lcd_ptr, "    X's Turn  ", 1);
```

```
	} else {
		lcd_nokia_write_string(lcd_ptr, "    O's Turn  ", 1);
	}
	lcd_nokia_render(lcd_ptr);
	return;
}

void keypad_init(){
	// Enable Pull-Up Resistors
	//MCUCR |= (0 << PUD);
	MCUCR &= ~(1<<PUD);

	// Configure PIND[4-7] as input
	DDRD = 0x00;

	// Enable the internal pull-ups by writing logic 1 to
	PORTD |= 0xFF;

	// Update: Set PC 0-2 as outputs
	// Drive PC0-2 low
	DDRC = 0x07;
	PORTC = 0x00;

	return;
}

void pin_change_init(){

	//Set Pin Change Masks for PCINT20-PCINT23
	PCMSK2 |= ((1<<PCINT20) | (1<<PCINT21) | (1<<PCINT22) | (1<<PCINT23));

	//Set Pin Change Control Register -  Pin Change Interrupt Enable 2
	PCICR |= (1<<PCIE2);

	//Set global interrupts
	sei();
}

uint8_t getKeyPressed()
{
	// Debounce - wait 10 MS
	_delay_ms(10);
	uint8_t c ;

	for (c = 0; c<3; c++) {
		//Drive a single col pin low
		PORTC = 0x07 & (~(1<<c));
		//c=0  00000111 & (~0000001) = 00000111 & (1111110) = 00000110
		//c=1  00000111 & (~0000010) = 00000111 & (1111101) = 00000101
		//c=2  00000111 & (~0000100) = 00000111 & (1111011) = 00000011

		// Changed all of these from PORTD to PIND
		if (bit_is_clear(PIND, 4)) {
			// Row 1 - PD4
			return (uint8_t)(3*0)+(c+1);
			// Row 1 is always returning 1

		} else if (bit_is_clear(PIND, 5)){
			// Row 2 - PD5
			return (uint8_t)(3*1)+(c+1);
```

```
		} else if (bit_is_clear(PIND, 6)) {
			// Row 3 - PD6
			return (uint8_t)(3*2)+(c+1);

		} else if (bit_is_clear(PIND, 7)){
			// Row 4 - PD7
			return (uint8_t)(3*3)+(c+1);
		}
	}
	return 0xFF; // no key pressed
}
```