MILESTONE 3: LINKED LISTS & MCU I/O

LAB OBJECTIVES

1. Gain a thorough understanding of pointers in C, and the implementation of pointers on the MCU.
2. Understand the benefits of implementing a dynamic data structure (i.e. linked list or queue) in comparison to that of a static structure (i.e. array).
3. Learn to implement both input and output functions on the MCU, and implement the required circuitry for basic I/O.
4. Understand the limitations of polling for an input, and implement an example of a polling circuit.

DELIVERABLES

1. Wire you MCU and kit to allow a 'nibble' of data to read from Port A. Connect your LED bank (8 bits) to Port B. Example circuit diagrams have been provided for you at the end of this write-up.

2. Digital I/O and Coding Exercise
   a. Implement a system to read a value from two digital I/O lines (PORTA0: 1) when another digital input line (PORTA2) is activated (Active LOW). Store the two-bit value from PORTA0: 1 in a FIFO queue as discussed in class. When the number of list items equals 4, remove the first entry that was detected and display each of the remaining two bit items on the PORTB LED bank (i.e. Item 0 on PORTB0:1, Item 1 on PORTB2:3, Item 2 on PORTB4:5). The items should be displayed using the following timing requirements (use your code from last week): Item 0 should appear first in its placeholder, after 2 seconds both Items 0 and 1 should appear in their placeholders, and finally after another 2 seconds Items 0, 1, and 2 should appear in their placeholders. During each of the four steps, if an item is not required on the display during the respective step, that placeholder should be LED = OFF (no light). See figure at the end of this write-up.
   b. Submit your coded routine for the application above.
   c. Submit your implemented linked list (FIFO queue) library of functions as described in class. Skeleton code will be provided to you.
   d. Provide a demonstration of the description provided in Part 2 (a), and provide a verbal explanation of your coded routine the week following.

TECHTALK

1. Proper prototype wiring procedures, running wires on bread boards, color coding

INSTRUCTIONS

THE LINKED LIST

In class, you learned how to create functions for implementing a dynamic data structure known as a linked list in the form of a first-in-first-out (FIFO) queue. A linked list is an effective and dynamic method to store and retrieve data. The linked list is much more flexible then a regular array for several reasons including having the ability to shrink or grow upon demand. In contrast to an array, a linked list eliminates the need to guess how many elements are needed to store, in other terms, a linked list could store an unknown quantity of items. Linked lists are also effective such that the elements within the list can be associated (using pointers) to multiple entities. In this lab, you will be required to implement a linked list.

Download the skeleton from the web and complete the linked FIFO queue functions specified in class. It is very important that you understand how to implement the linked list to be able to effectively complete your final milestone and for your Module 1 exam. The functions you develop in this exercise can be used in the final milestone if you desire.

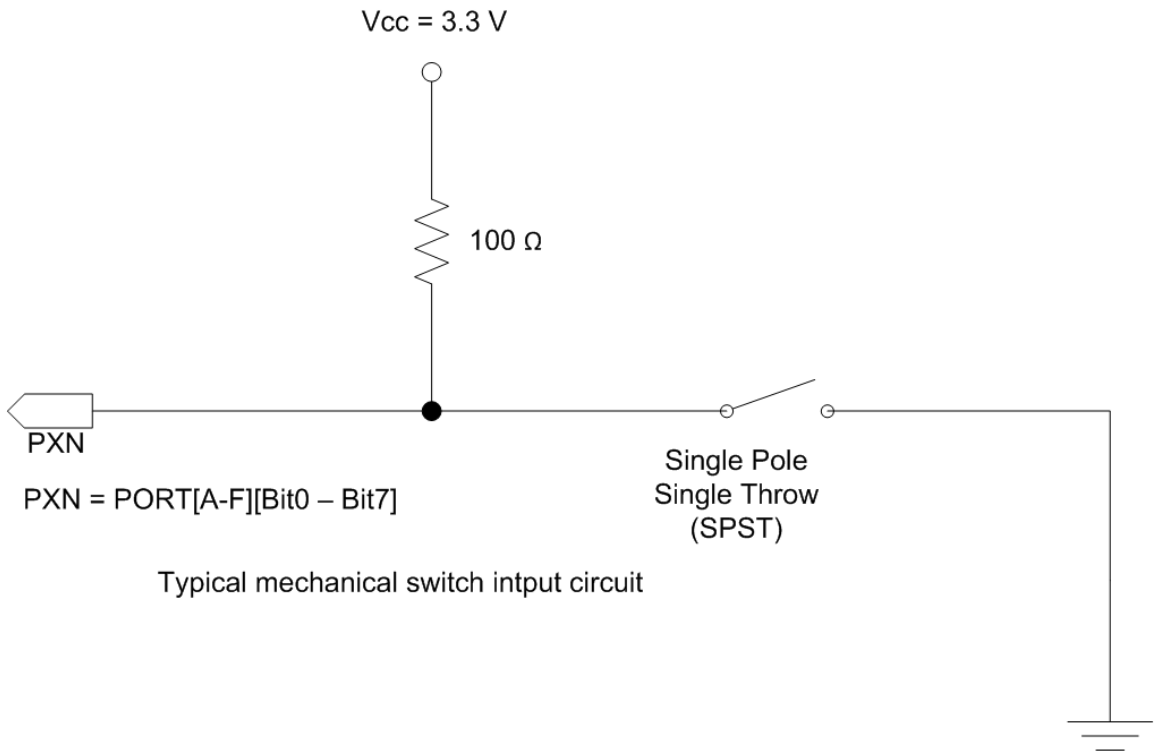
INPUT AND POLLING A SWITCH

Understand the limitations of two details used in this lab. Firstly, a mechanical switch has some limitations in producing logical levels for digital I/O. A closer inspection of the switch may reveal that during the transition from the OPEN state to the CLOSED state of the switch an action known as ‘contact bounce’ can occur. Contact bounce is produced when the metal contacts bounce off of each other producing oscillating state. You will be shown in class several methods to avoid contact bounce, and will implement one of these methods in your final milestone requirements. For now, you will implement a simple polling routine to acquire an input from the user. The mechanical switch is sufficient in this exercise as the frequency at which the user is able to transition the switch from an OPEN state to a CLOSED state is very low. This ensures that a transition will not be missed during the polling loop.

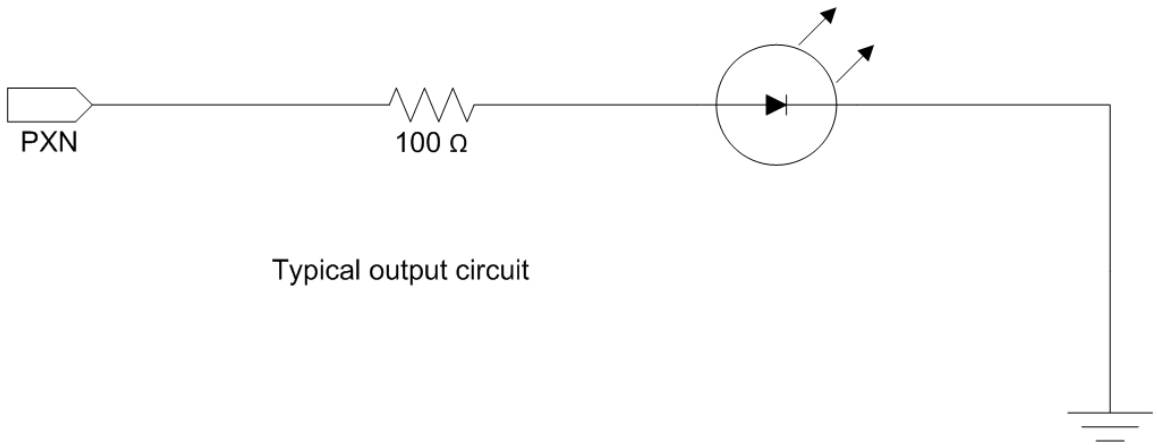
The AVR MCU makes life simple to perform digital input. Like the digital output, the data direction registers (DDRX) need to be set. It is also important that the inputs are not ‘floating’. You will see this term in many reference manuals, and you are required to provide a definitive HIGH or LOW value to the input port otherwise the MCU will not know how to respond. The following code shows how to take input from Port A, and display the output to your LED bank on Port B. In this simple example, a circuit is polled to determine whether the correct logic level is obtained. From the circuit diagram attached at the end of this exercise what state of the switch (OPEN, or CLOSED) will produce a logic 0 for the MCU input. If you don’t understand this circuit please ask for assistance.

```c
#include<avr/io.h>

int debug(char intput);

int main(int argc, char *argv[]){

    char readInput;

    /* Set all of Port A to input bits */
    DDRA = 0x00;

    /* Set all of Port B to output bits */
    DDRB = 0xFF;

    /* Poll for Port C input and adjust the onboard LEDS */
    /* Make sure all inputs have a value...ie. not floating!!!*/
    while(1){
        /* NOTE: we read off the register PINC NOT PORTC */
        readInput = PINA;
        debug(readInput);
    }

    return(0);
}

int debug(char input){

    switch (input){
        case (0x01) :
            PORTB = 0b00000001;     /* GRN  GRN */
            break;
        case (0x02) :
            PORTB = 0b00000010;     /* GRN  RED */
            break;
        case (0x04) :
            PORTB = 0b00000100;     /* RED  GRN */
            break;
        case (0x08) :
            PORTB = 0b00001000;     /* RED  RED */
            break;
        default     :
            PORTB = 0b00000000;     /* OFF  OFF */
            break;
    }/*switch*/

    return(input);
}/* debug */
```

MILESTONE 1: CIRCUIT DIAGRAM FOR INPUT PORTS



Typical mechanical switch intput circuit



Typical output circuit

MILESTONE 2B DIAGRAM FOR INPUT

SW3	SW2	SW1	SW0
X	1	1	0
PA3	PA2	PA1	PA0

High level nothing read

SW3	SW2	SW1	SW0
X	0	1	0
PA3	PA2	PA1	PA0

Low level (active low) on PORTA2 – read PORTA0:1 = 10_B

SW3	SW2	SW1	SW0
X	1	0	1
PA3	PA2	PA1	PA0

High level nothing read

SW3	SW2	SW1	SW0
X	0	0	1
PA3	PA2	PA1	PA0

Low level (active low) on PORTA2 – read PORTA0:1 = 01_B

SW3	SW2	SW1	SW0
X	1	0	0
PA3	PA2	PA1	PA0

High level nothing read

SW3	SW2	SW1	SW0
X	0	0	0
PA3	PA2	PA1	PA0

Low level (active low) on PORTA2 – read PORTA0:1 = 00_B

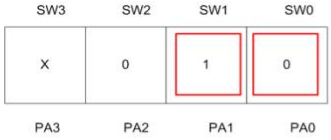
SW3	SW2	SW1	SW0
X	1	1	1
PA3	PA2	PA1	PA0

High level nothing read

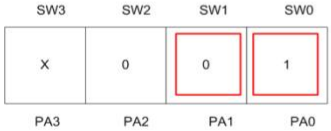
SW3	SW2	SW1	SW0
X	0	1	1
PA3	PA2	PA1	PA0

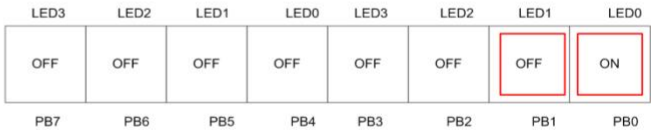
Low level (active low) on PORTA2 – read PORTA0:1 = 11_B

MILESTONE 2B DIAGRAM FOR OUTPUT

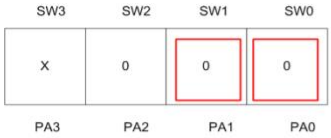


Remove and discard

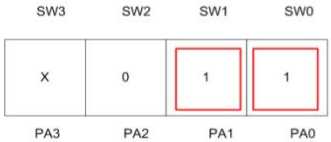


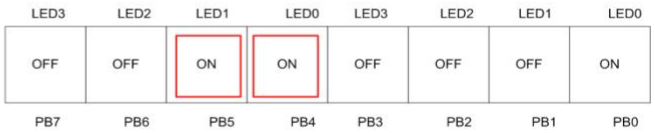


Wait 2 seconds +



Wait 2 seconds +





NOTE: The other entries are still present from the previous steps