ECE3411-Spring 2019
Lecture#3

General Purpose Digital Input

Sung Yeul Park
Department of Electrical and Computer Engineering
University of Connecticut
sung_yeul.park@uconn.edu

Reading a logic value from a Port

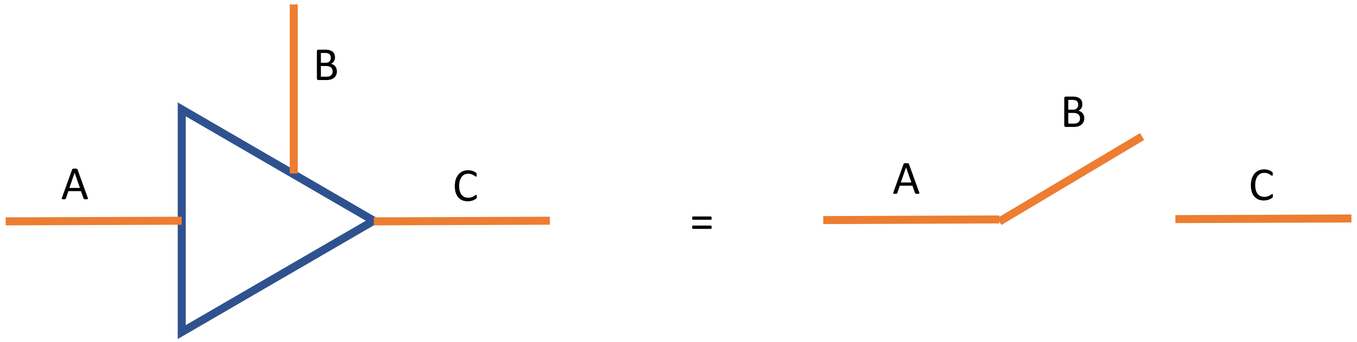
Suppose we want to read the logic value of 7th pin of Port B:

1. Read the register PINB in a character variable, i.e.
   ```
   char reg = PINB
   ```
2. Let PINB register has a value 0b10101010 then
   ```
   reg = 0b10101010
   ```
3. Create a mask to mask out all the bits in ‘reg’ except for 7th bit position, i.e.
   ```
   0b10000000 = (1<<7) = (1<<PINB7)
   ```
4. Use the mask to mask out all the bits except for the 7th bit, and decide based on the resultant value, i.e.
   ```
   if( reg & (1<<PINB7) ) { /* 7th pin is logic 1 */ }
   else                  { /* 7th pin is logic 0 */ }
   ```

Tristate Buffer

- In a naïve button circuit, a closed button connects a pin to the MCU to Gnd:
  - When it opens, the MCU end of the button/switch (i.e. pin) dangles in the air
  - It acts as an antenna picking up high/low voltages depending on what frequency the local radio stations / “noisy” electrical appliances broadcast
  - Unreliable!
- Need a pull-up resistor (10kOhm) at the pin, so that if the switch is open, the voltage at the pin is pulled to high
  - If the switch is closed, the resistance to Gnd is much lower so that the voltage at the pin is close to zero
- The pull-up resistor is implicitly implemented by setting the output of the pin to high as a result of programming PORTx

Tristate Buffer



A (PORT)	B (DDR)	C (PIN)
0	1	Low impedance High out 0
1	1	Low impedance High out 1
0	0	High impedance
1	0	High impedance

- DDR (B) = 0 and PORT (A) = 1: Eliminates static effects/noise and allows to read port/pin in a coherent fashion → PORT (A) = 1 activates the pull-up resistor and makes reading PIN (C) reliable
- DDR (B) = 0 and PORT (A) = 0: Is good for creating high impedance if you do not want the PIN to have any current at all

Debouncing

- Capturing a button push is a very fast process
- When you press a switch closed, two surfaces are brought into contact with each other → no perfect match and electrical contact will be made and unmade a few times till the surfaces are firm enough together
    - The same is true when you release a button, but in reverse
    - Bouncing between high and low voltage is often at a timescale of a few us to a few ms → very often you do not see it
- No debouncing SW:

```
unsigned char counter;

while (1) {
   ...
   //button push of the switch connected to PINB7
   if (!(PINB & (1 << PINB7))) {
      counter++;
   ...
}
```

Debouncing

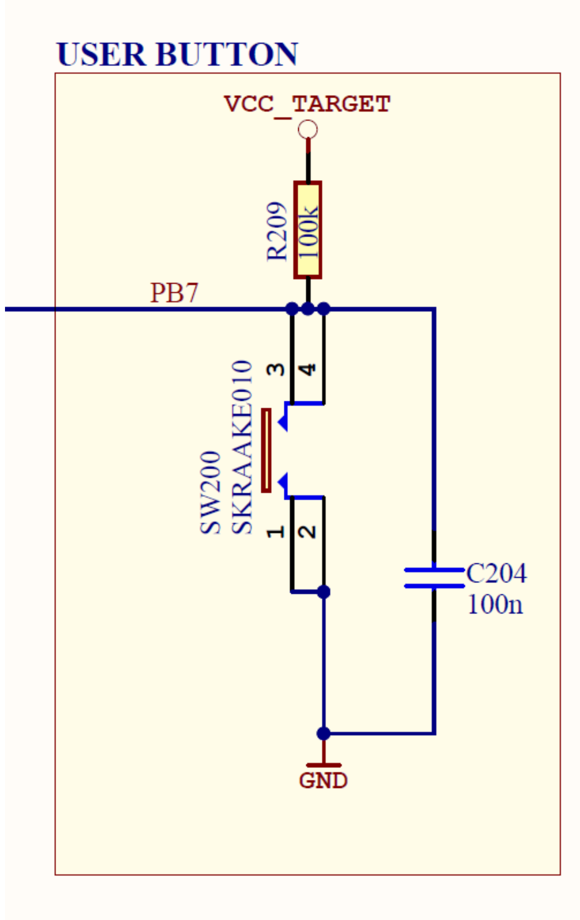
```
unsigned char counter;

while (1) {
   ...
   _delay_ms(10);
   if (!(PINB & (1 << PINB7))) {
      counter++;
   }
   ...
}
```

```
unsigned char counter;
unsigned char pushCount = 0;
unsigned char releaseCount = 0;
while (1) {
   if (!(PINB & (1 << PINB7))) {
      releaseCount = 0;
      pushCount++;
      if (pushCount > 500) {
         counter++;
         pushCount = 0;
      }
   } else {
      pushCount = 0;
      releaseCount++;
      if (releaseCount > 500) {
         ...
         releaseCount = 0;
      }
    }
}
```

Hardware Debouncer

- HW debouncers are also possible:
  - Just by using a low pass filter (a capacitor across the two contacts of the switch)
  - However everyone debounces in SW, saving a few cents per capacitor

- Figure shows the schematic of the push button onboard ATmega328p Xplained Mini kit
  - This is Hardware Debounced switch (Notice the capacitor C204)
  - The switch is connected to PB7



One-shot buttons

- There are times where you may want a button-press to be registered only once
- In other words, the first time you check the button after the press, you will get a ‘1’. Any subsequent checks, will get you a ‘0’.

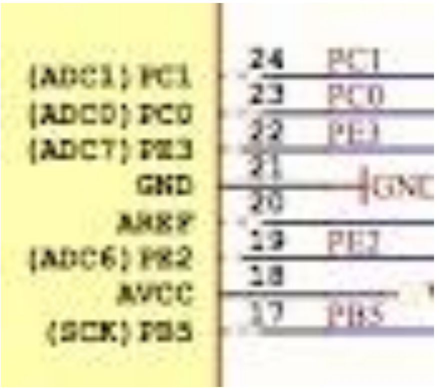
```c
unsigned char counter;

while (1) {
   ...
   //button push of the switch connected to PINB7
   if (!(PINB & (1 << PINB7))) {
      counter++;
   ...
}
```

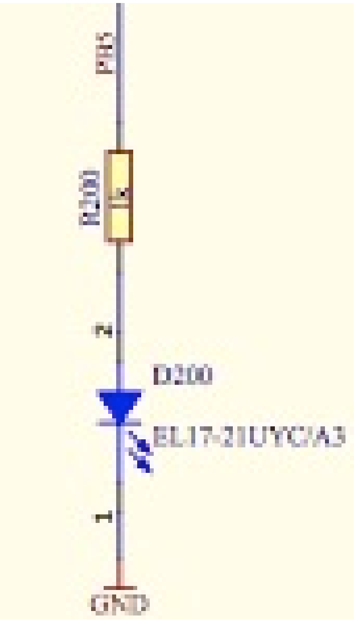
One-shot buttons

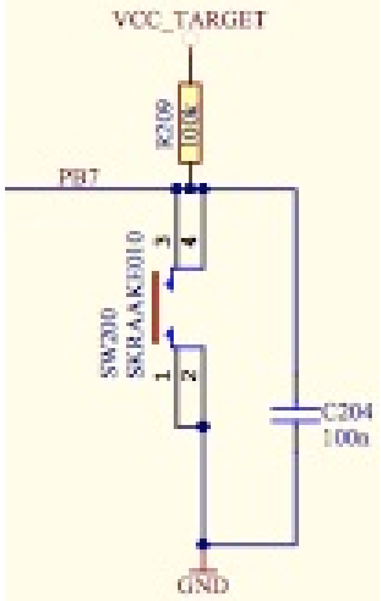
```c
unsigned char counter;
unsigned char buttonHandled = 0; //flag that button was pressed
while (1) {
   ...
   // button push of the switch connected to PINB7
   _delay_ms(10);
   if (!(PINB & (1 << PINB7))) {
      if (!buttonHandled) {
         counter++;
         buttonHandled = 1;
      }
   }
   else {
      buttonHandled = 0;
   }
   ...
}
```

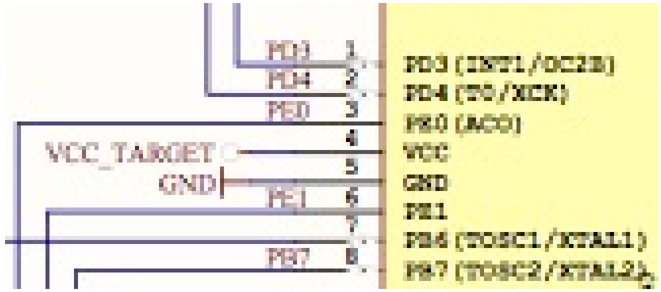
A Simple Test Program

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

int main(void)
{
    //configure LED pin as output
    DDRB |= 1<<DDB5;
    while (1){
        /* check the button status
           (press - 0 , release - 1 ) */
        if( !( PINB & (1<<PINB7) ) ) {
            /* switch off (0) the LED
               until key is pressed */
            PORTB &= ~(1<<PORTB5);
        }
        else {
            /* switch on (1) the LED*/
            PORTB |= 1<<PORTB5;
        }
    }
}
```


(ADC1) PC1
(ADC0) PC0
(ADC7) PE3
GND
AREF
(ADC6) PE2
AVCC
(SCK) PB5
24 PC1
23 PC0
22 PE3
21 GND
20
19 PE2
18
17 PB5


PB5
R200
1k
2
D200
EL17-21UYC/A3
1
GND


VCC_TARGET
R209
100k
PB7
SW200
SKRAAKE010
3
4
1
2
C204
100n
GND


PD3
PD4
PE0
VCC_TARGET
GND
PE1
PB7
1
2
3
4
5
6
7
8
PD3 (INT1/OC2B)
PD4 (T0/XCK)
PE0 (ACO)
VCC
GND
PE1
PB6 (TOSC1/XTAL1)
PB7 (TOSC2/XTAL2)