ECE3411

Lecture 3

Digital Input

Sung-Yeul Park
Department of Electrical & Computer Engineering
University of Connecticut
Email: sung_yeul.park@uconn.edu

Slides adopted from Marten van Dijk & Syed Kamran Haider ECE 3411 - Fall 2016
Slides adopted from John Chandy ECE 3411 - Fall 2024

Lab practice#1: Blinking 8 LEDs

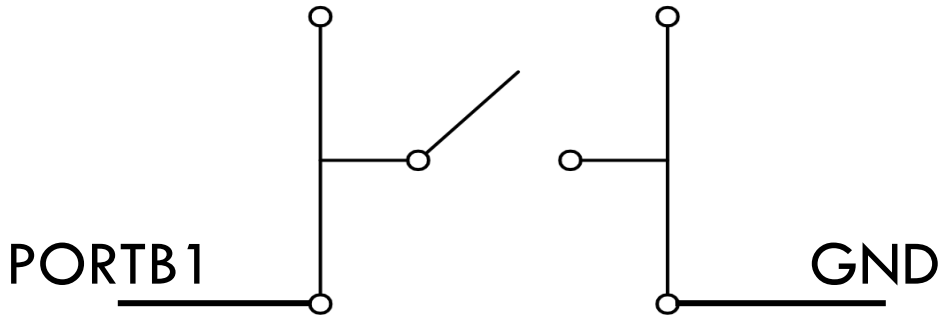
Blinking on all 8 LEDs one after another.

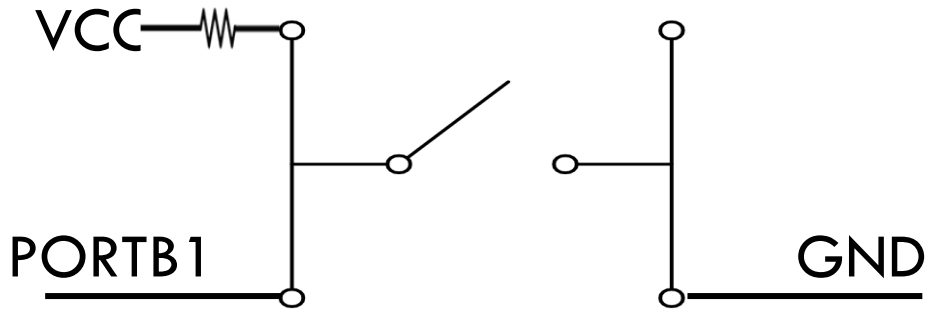
- When the system starts, LED 0 is active and blinks at 2Hz.
- After two seconds, it will blink at 4Hz.
- After two more seconds, the blinking LED will go back to 2Hz and also shift to LED 1
- This will continue
  - Every two seconds, the frequency will flip between 2 and 4 Hz.
  - Every four seconds, the LED will shift to the left. When the active LED reaches the left-most position (LED 7) it should start from 0 again.

- You can do this with nested for loops that do the appropriate number of blinks at each frequency
- Or, without nested loops, you need to
  - Keep track of the current frequency
  - Keep track of the LED bit position
  - Keep track of the second or millisecond count so you know when to switch frequency of position. Update this count after you blink the LEDs

Tristate Buffer

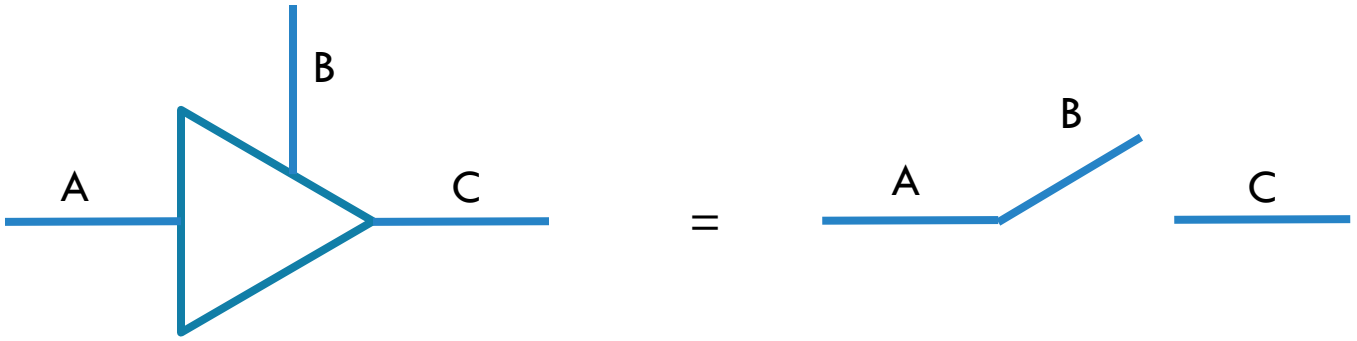
- In a naïve button circuit, a closed button connects a pin to the MCU to Gnd:
  - When it opens, the MCU end of the button/switch (i.e. pin) dangles in the air
  - It acts as an antenna picking up high/low voltages depending on what frequency the local radio stations / “noisy” electrical appliances 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



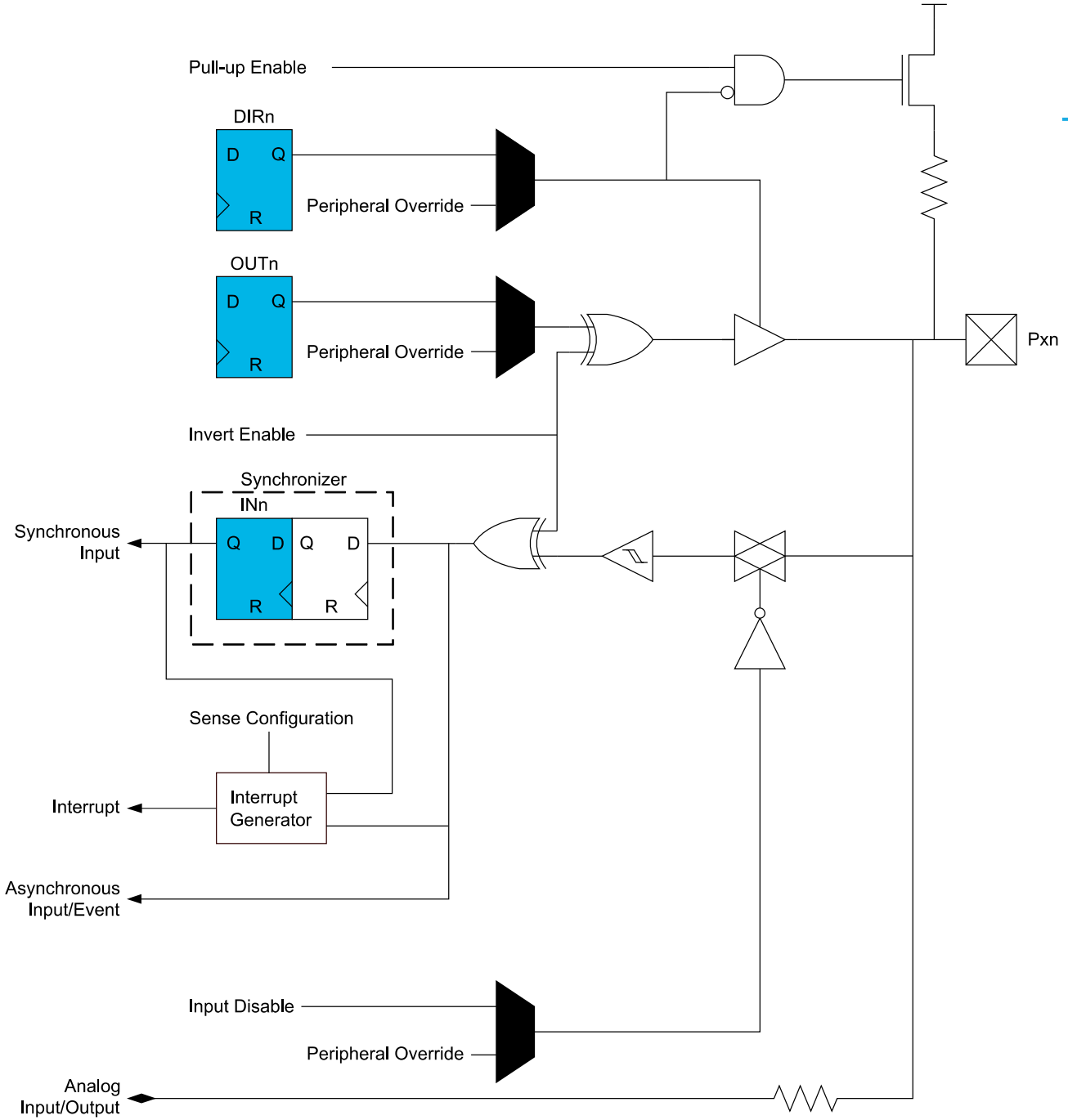


Tristate Buffer

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)



Pin configuration



- **DIRn (Direction Register)**: Determines whether the pin is configured as input or output.
- **OUTn (Output Register)**: Holds the value to be output when the pin is set as an output.
- **Peripheral Override**: Allows internal peripherals to take control of the pin, bypassing manual I/O control.
- **Pull-up Enable**: Activates an internal pull-up resistor when the pin is used as input.
- **Invert Enable**: Inverts the logic level of the pin (useful for active-low configurations).
- **Synchronizer with INn (Input Register)**: Captures the input signal and synchronizes it with the system clock.
- **Sense Configuration**: Defines how the pin detects changes (e.g., rising edge, falling edge, level).
- **Interrupt Generator**: Triggers an interrupt based on the sense configuration.
- **Input Disable**: Turns off the input buffer, often used when the pin is used for analog functions.
- **Analog Input/Output**: Indicates that the pin can also be used for analog signals (ADC or DAC).

Input pins

18.5.16 Pin n Control

Bit	7	6	5	4	3	2	1	0
	INVEN	INLVL			PULLUPEN	ISC[2:0]		
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

Bit 7 – INVEN Inverted I/O Enable

This bit controls whether the input and output for pin n are inverted or not.

Value	Description
0	Input and output values are not inverted
1	Input and output values are inverted

Bit 3 – PULLUPEN Pull-Up Enable

This bit controls whether the internal pull-up of pin n is enabled or not when the pin is configured as input-only.

Value	Description
0	Pull-up disabled
1	Pull-up enabled

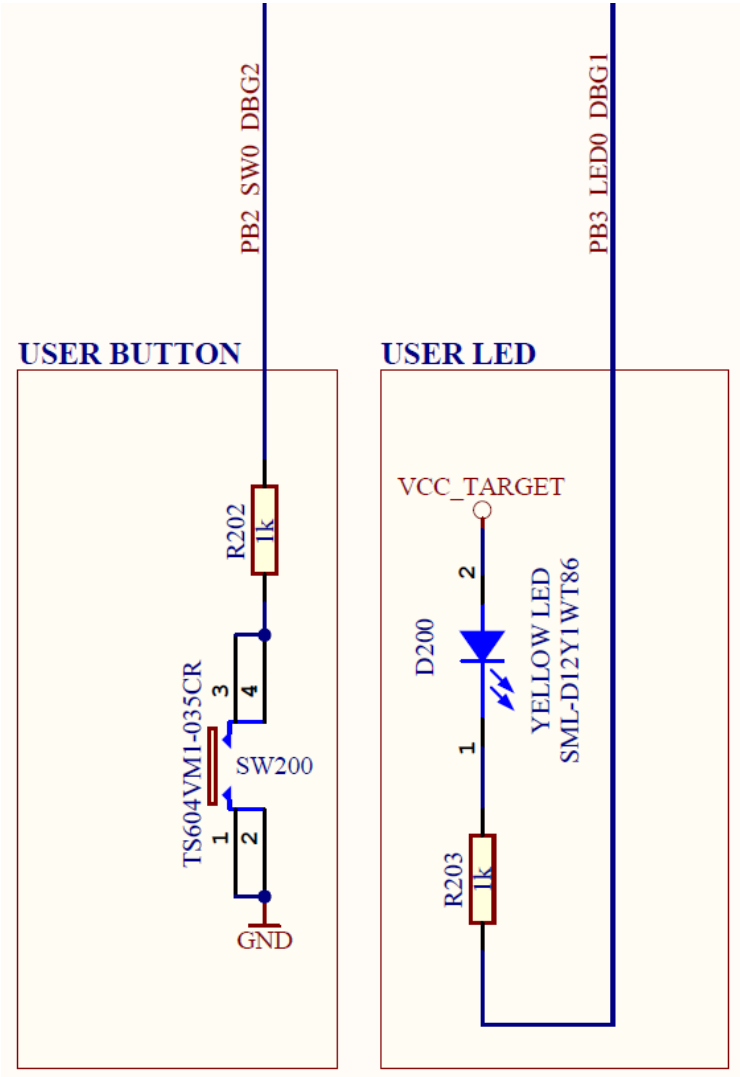
Reading a logic value from a Port

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

1. Read the register `PORTB.IN` in a character variable, i.e.
```
char reg = PORTB.IN;
```
2. Assume `PORTB.IN` 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<<PIN7_bp)
```
4. Use the mask to mask out all the bits except for the 7th bit, and decide based on the resultant value, i.e.
5.
```
if( reg & (1<<PIN7_bp) ) { /* 7th pin is logic 1 */ }
else                     { /* 7th pin is logic 0 */ }
```
OR
```
if( reg & PIN7_bm ){ /* use predefined mask */ }
else               {                              }
```

A Simple Test Program

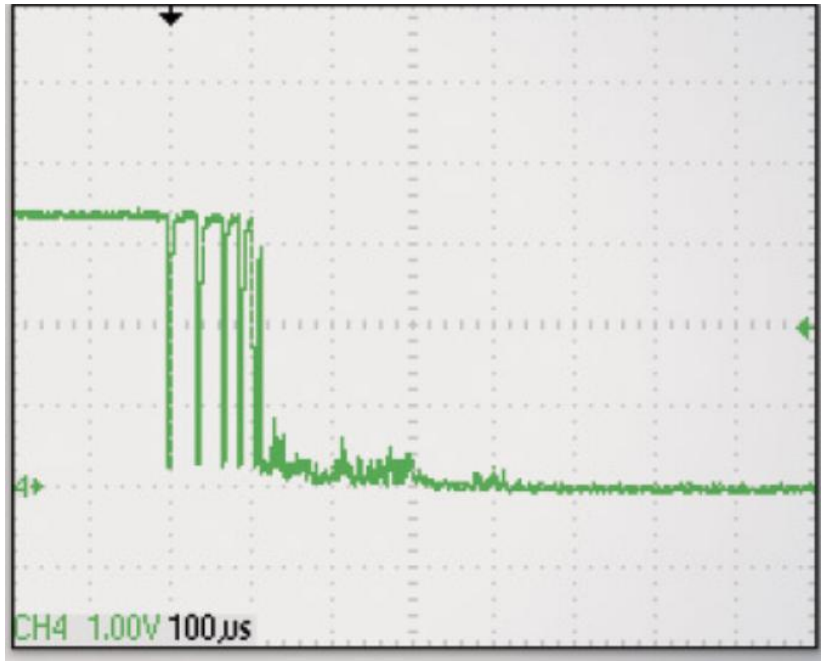
```
#include <avr/io.h>

int main(void)
{
    /* turn on PORTB2 pull up resistor */
    PORTB.PIN2CTRL |= PORT_PULLUPEN_bm;
    /* configure LED pin as output */
    PORTB.DIRSET = PIN3_bm;
    while (1){
        /* check the button status
           (press - 0 , release - 1 ) */
        if( !( PORTB.IN & PIN2_bm ) ) {
            /* button on (0) turn on LED */
            PORTB.OUTCLR = PIN3_bm;
        }
        else {
            /* button off (1) turn off LED*/
            PORTB.OUTSET = PIN3_bm;
        }
    }
}
```



Mechanical Switch Bounce Effects

- 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 - 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 µs to a few ms

  → very often you do not see it



Without debounce code

- 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'.
- In code below, counter will get incremented multiple times as long as button is pressed, not just once.

```
unsigned char counter;

while (1) {
    ...
    //button push of the switch connected to PORTB2
    if ( !(PORTB.IN & PIN2_bm ) ) {
       counter++;
    ...
}
```

One-shot delay button

- You could try a delay and hope that the button is released before the button is checked again.

```
unsigned char counter;
while (1) {
    ...
    // button push of the switch connected to PORTB2
    if ( !(PORTB.IN & PIN2_bm ) ) {
        counter++;
        _delay_ms(1000);
    }
...
}
```

Button pressed flag

- Better approach is to use a flag to keep track of whether button has been pressed

```
unsigned char counter;
unsigned char buttonHandled = 0; //flag that button was pressed

while (1) {
    ...
    // button push of the switch connected to PORTB2
    if ( !(PORTB.IN & PIN2_bm ) ) {
        if (!buttonHandled) {
            counter++;
            buttonHandled = 1;
        }
    }
    else {
        buttonHandled = 0;
    }
    ...
}
```

Debouncing

```
unsigned char counter;

while (1) {
   ...
   if (!(PORTB.IN & PIN2_bm)) {
      _delay_ms(10);
      if (!(PORTB.IN & PIN2_bm )) {
         if (!buttonHandled) {
            counter++;
            buttonHandled = 1;
         }
      }
  }
  else {
     buttonHandled = 0;
  }
  ...
}
```

```
unsigned char counter;
unsigned char pushCount = 0;
unsigned char releaseCount = 0;
while (1) {
   if (!(PORTB.IN & PIN2_bm)) {
      releaseCount = 0;
      pushCount++;
      if (pushCount > 500) {
         counter++;
      }
   } else {
      pushCount = 0;
      releaseCount++;
      if (releaseCount > 500) {
         releaseCount = 0;
      }
   }
}
```

Hardware Debounce

1. RC Low-Pass Filter

Components: Resistor (R) and Capacitor (C)

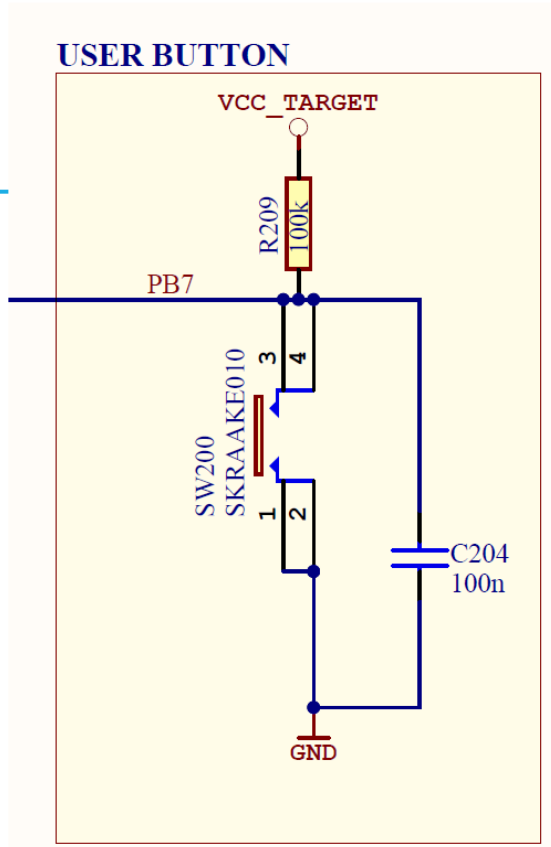
How it works: Smooths out rapid voltage changes caused by bouncing.

Typical values:

R = 10kΩ

C = 0.1µF

Effect: Filters out high-frequency noise and short pulses.

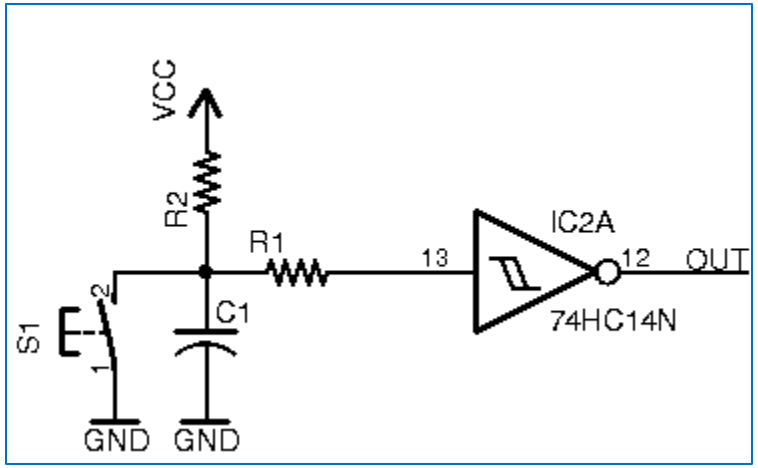


2. Schmitt Trigger

Component: Logic gate or buffer with hysteresis (e.g., 74HC14)

How it works: Converts noisy input into clean digital transitions.

Advantage: Built-in noise immunity and sharp switching.



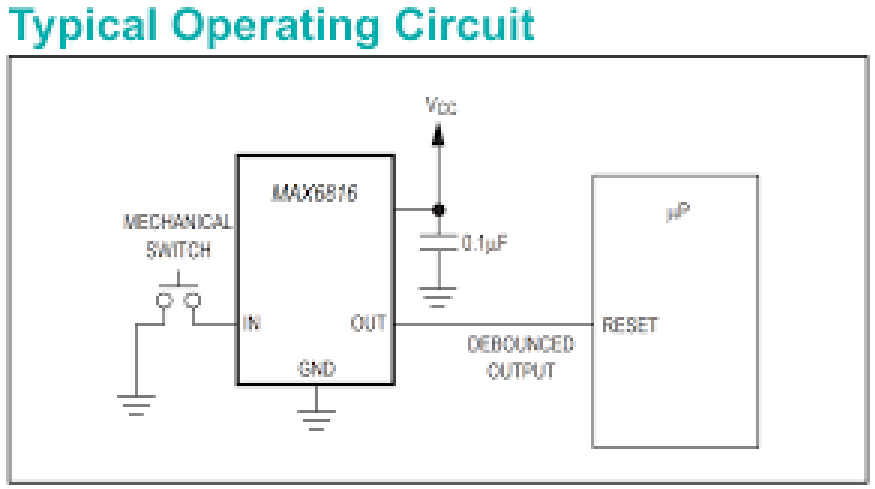
Hardware Debounce

3. Debounce ICs

Example: MAX6816, MAX6818

How it works: Specialized chips designed to debounce mechanical switches.

Advantage: Reliable and compact solution for multiple switches.

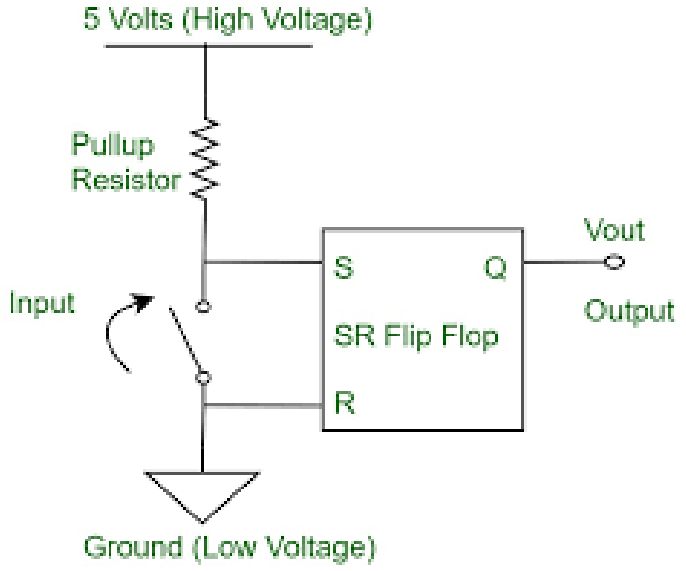


4. Flip-Flop Based Debounce

Components: D-type flip-flops or SR latches

How it works: Captures stable transitions after bouncing settles.

Used in: More complex digital circuits or programmable logic.



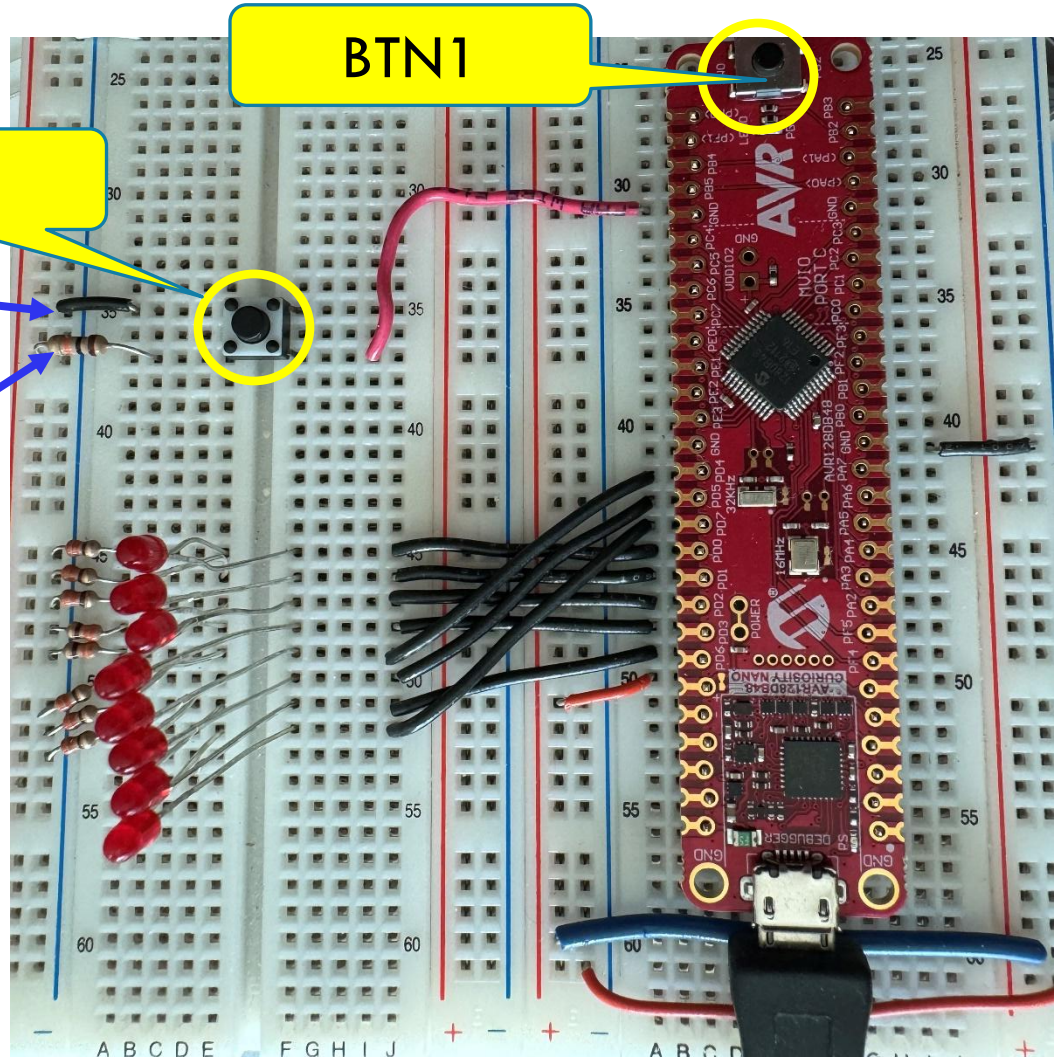
Lab practice #2

- Based on the lab practice #1 setup, connect an external push-button switch to Port B with a 10kΩ pull-down resistor. You will use two button switches: one is built into the board(PB2), and the other is the external switch(PB5).
- Task 1
  - 8 LEDs of port D as output
  - Set up the blinking frequency of the 3rd LED to 3Hz and rest of the LEDs should be turned off.
- Task 2
  - 2 buttons BTN1(on-board) and BTN2(off-board) as input
  - Increase the blinking frequency of the LED by 1 Hz if button BTN1 is pressed
  - Decrease the blinking frequency of the LED by 1 Hz if button BTN2 is pressed
    - Do not go below 1 Hz
- Task 3
  - If both BTN1 and BTN2 button are pressed at the same time, change the position of the blinking LED
  - The position of the blinking LED should go left to right initially
  - After it goes to 8th position the direction will be reversed
  - Once it goes to 1st position the direction will be reversed again

Switches pressed	Status of the LEDs
Initial condition	00100000
BTN1 and BTN2	00010000
BTN1 and BTN2	00001000
BTN1 and BTN2	00000100
BTN1 and BTN2	00000010
BTN1 and BTN2	00000001
BTN1 and BTN2	00000010
BTN1 and BTN2	00000100
BTN1 and BTN2	00001000
BTN1 and BTN2	00010000
BTN1 and BTN2	00100000
BTN1 and BTN2	01000000
BTN1 and BTN2	10000000
BTN1 and BTN2	01000000
BTN1 and BTN2	00100000
.	.
.	.
.	.
	.

Here, 1 means blinking LED and 0 means off LED.

Connections



Button should be across the divider