EMBEDDED SYSTEMS CMPE-453

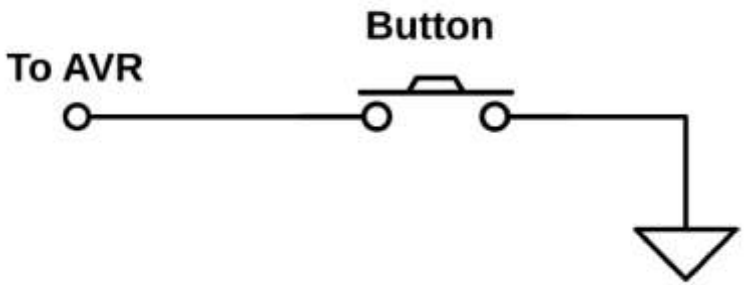
Department of Computer Engineering

Digital Input

IMPORTANCE OF INPUT

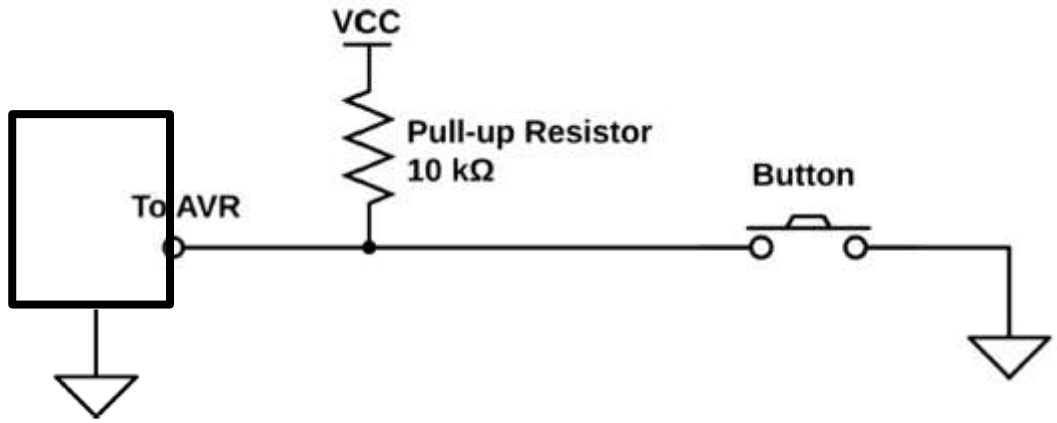
Task is, figure out how to apply logic high and low voltages to an AVR pin by way of a pushbutton.

- Input voltage when button is pressed: 0V
- Input voltage when button is not pressed: unreliable/undefined
- How to make it reliable?



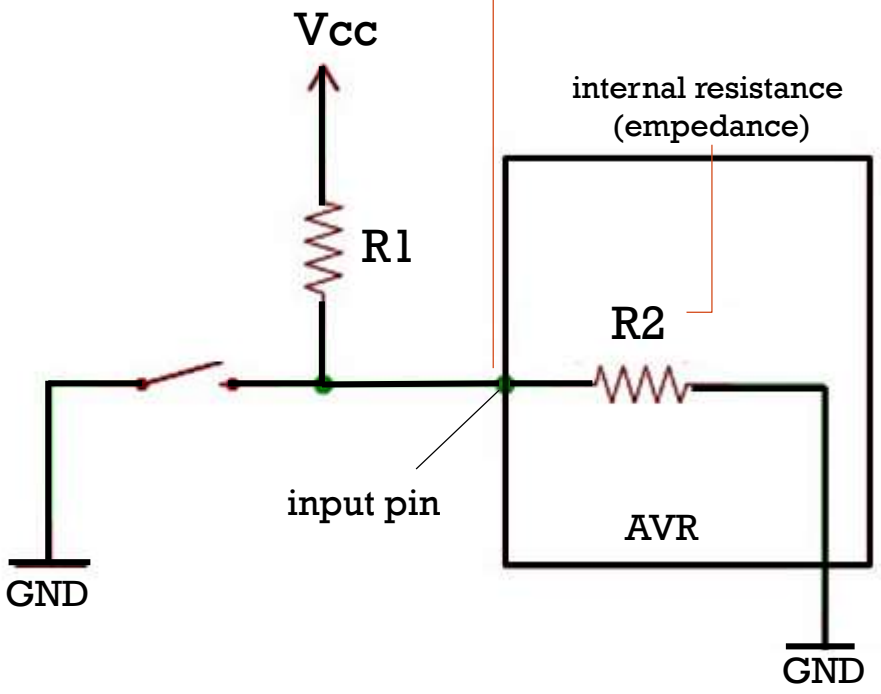
PULL-UP RESISTOR

- Pulls-up the non-ground side to a default high-voltage.
- Relatively large-value resistor (10K to 100 K)



PULL-UP RESISTOR

$$V_{in} = \frac{Vcc}{R_1 + R_2} R_2$$

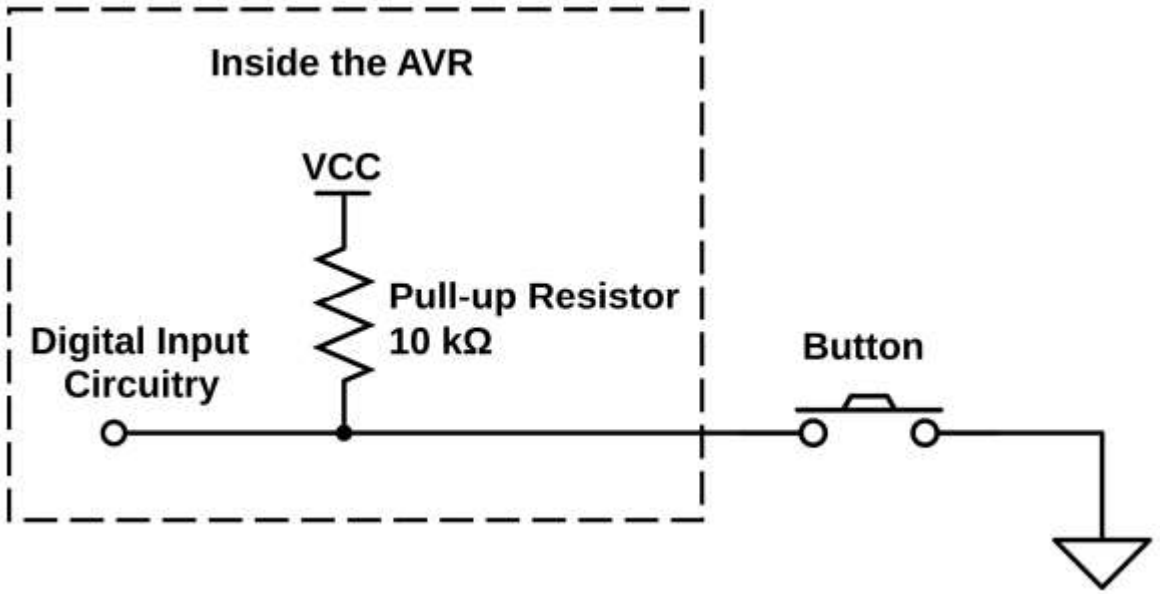


An input pin on a microcontroller has an impedance can vary from 100k-1MΩ

- For example,
- 1MΩ resistor for the pull-up R1, input pin's impedance R2 is on the order of 1MΩ, Vin is half of VCC
- Microcontroller might not register the pin being in a high state.
- On a 5V system, what is read on the input pin if the voltage is 2.5V? Is it a high or a low? MC doesn't know and you might read either a high or a low.
- A resistance of 10k to 100kΩ for R1 should avoid most problems.

PULL-UP RESISTOR

- What if we want to use all eight pins of a port as input pins?
- Do we need to connect 8 pull-up resistors?
- No: AVR has built-in pull up resistors on all pins.
- Just activate them for input pins.



CONFIGURING PIN(S) AS INPUT

- DDR(x):
  - 1 – pin is configured as output pin
  - 0 – pin is configured as input pin (Default)

```
DDRD &= ~(1 << PD2); //makes double-sure we are in input mode
```

- PORT(x): For pin(s) configured as input
  - 0 – disable pull-up resistor (Deafult)
  - 1 – enable pull-up resistor

```
PORTD |= (1 << PD2); //enables pull-up resistor
```

- PIN(x): **P**in-**In**put
  - Contains the input logic values

```
PIND
```

EXAMPLE 1:

- Write c-statements to
  - Configure PORTB as Input
  - Enable its internal pull-up resistor
  - Read the value

```
DDRB = 0x00; //just to be sure
PORTB = 0xFF;//enable pull-up(s)
tmp = PINB;
tmp: 11111111 // when we read PINB
```

let's say we grounding pin PB2 by a push button

```
tmp = PINB;
PINB: 11111011
```

Testing bits with AND

We're interested in testing if bit three is set.

- create a bitmask with a 1 in bit three: (1 << 3) : 00001000
- AND that byte with whatever is in the input register, PIND:

```
PIND :        xxxxxxxx
(1 << 3) :    00001000
& :           0000x000
```

If the value we were interested in knowing is a zero, the result is eight zeros.
If the result isn't zero, then we know the pin in the input register must have been set.

```
if (PIND &  (1<<3)){
doStuff();
}
```

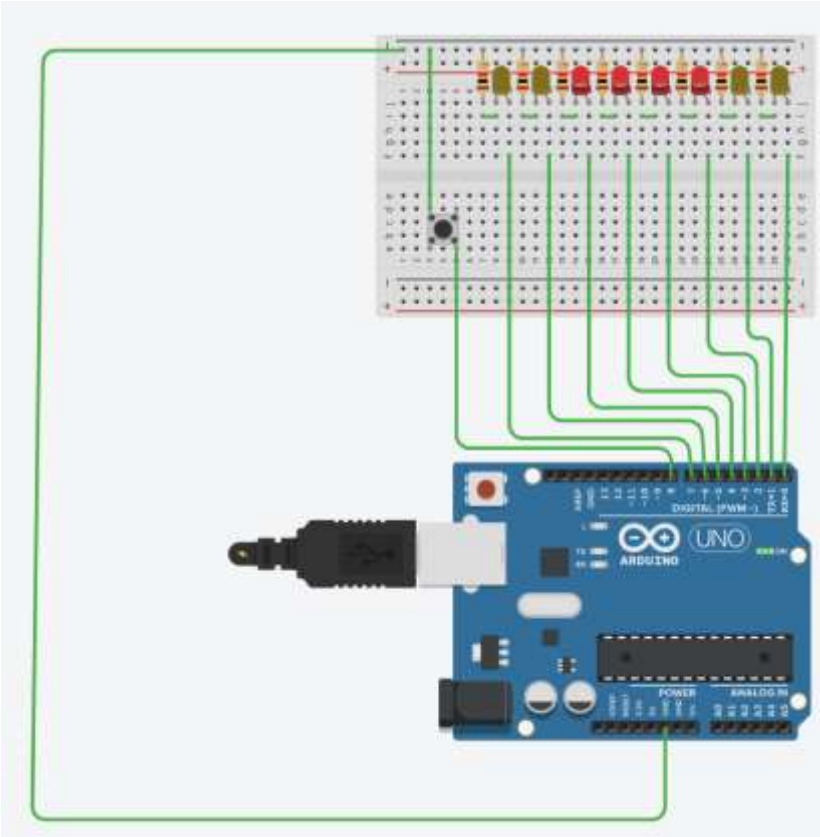
SOME HELPFUL MACROS

- Included with avr/io.h

```
#define _BV(bit): (1 << (bit))                                  // Converts a bit number into a byte value.

#define bit_is_set(sfr, bit): (_SFR_BYTE(sfr) & _BV(bit))       //Test whether bit bit in IO register sfr is set.
                                                                //will return a 0 if the bit is clear, and non-zero if the bit
                                                                //is set.

#define bit_is_clear(sfr, bit): (!(_SFR_BYTE(sfr) & _BV(bit)))  //Test whether bit bit in IO register sfr is clear.
                                                                //will return non-zero if the bit is clear, and a 0 if the bit
                                                                //is set.

#define loop_until_bit_is_set(sfr, bit): do { } while (bit_is_clear(sfr, bit))      //Wait until bit bit in IO register sfr is set.

#define loop_until_bit_is_clear(sfr, bit) do { } while (bit_is_set(sfr, bit))       //Wait until bit bit in IO register sfr is set.
```

EXAMPLE 2

- Build an AVR-based circuit (as shown), which dsplays
  - 0xC3 on LEDs when push-button is pressed.
  - 0x3C on LEDs when push-button is released.



```
/* Demo of the simplest on/off button code Button connected to PB0 LEDs connected to PD0..PD7 */
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
int main(void)
    {
    // -------- Inits --------- //
    /* initialize pullup resistor on our input pin i.e PB0*/
    PORTB |= (1 << PB0);
    /* set up all LEDs for output */
    DDRD = 0xff;
    // ------ Event loop ------ //
    while (1)
        {
        // look for button press
        if(!(PINB & (1<<PB0)))
            { /* pressed */
            PORTD = 0b11000011;
            }
        /* not pressed */
        else
            {
            PORTD = 0b00111100;
            }
        } /* End event loop */
    return 0;
    }
```

```c
/* Demo of the simplest on/off button code Button connected to PB0 LEDs connected to PD0..PD7 */
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
int main(void)
    {
    // -------- Inits --------- //
    /* initialize pullup resistor on our input pin i.e PB0*/
    PORTB |= (1 << PB0);
    /* set up all LEDs for output */
    DDRD = 0xff;
    // ------ Event loop ------ //
    while (1)
        {
        // look for button press
        if (bit_is_clear(PINB, PB0))
            { /* pressed */
            PORTD = 0b11000011;
            }
        /* not pressed */
        else
            {
            PORTD = 0b00111100;
            }
        } /* End event loop */
    return 0;
    }
```

EXAMPLE 3: READING CHANGE IN BUTTON-STATE

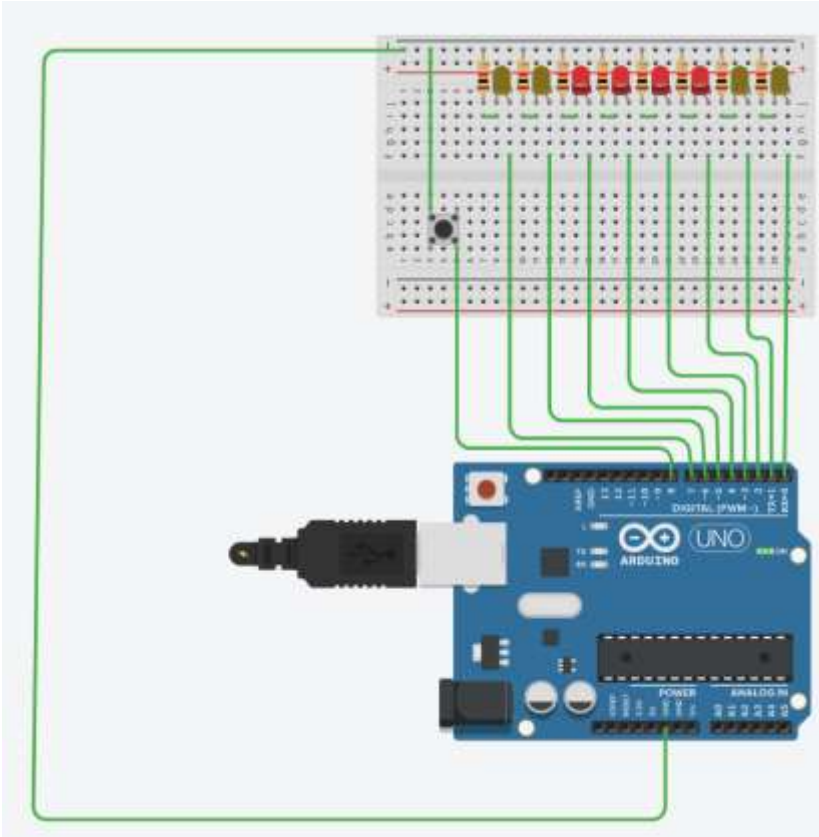
- Program AVR to toggle LEDs when button is pressed i.e state change from open to close.
- Pseudocode:

```
while(1)
        {
        if(button is pressed)
                Toggle LEDs
        }
```

- Won't work
- Why?
- Toggles the LEDs 100,000 times per second when button is pressed.

EXAMPLE 3: READING CHANGE IN BUTTON-STATE

```
old_state = not_pressed
while(1)
        {
        current_state = read_button_state;
        if( (current_state==pressed) && (!old_state))
                Toggle LEDs
        old_state=current_state
        }
```



```
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
int main(void)
    {
    // -------- Inits --------- //
    PORTB |= (1 << PB0);
    DDRD = 0xff;
    bool old_state = false;
    bool current_state;
    PORTD = 0b00111100;
    // ------ Event loop ------ //
    while(1)
        {
        current_state=bit_is_clear(PINB, PB0);
        if(current_state && !old_state)
            PORTD^=0xFF;
        old_state=current_state;
        }
    return 0;
    }
```