Microprocessors, Microcontrollers, and Embedded Systems

CSE 315

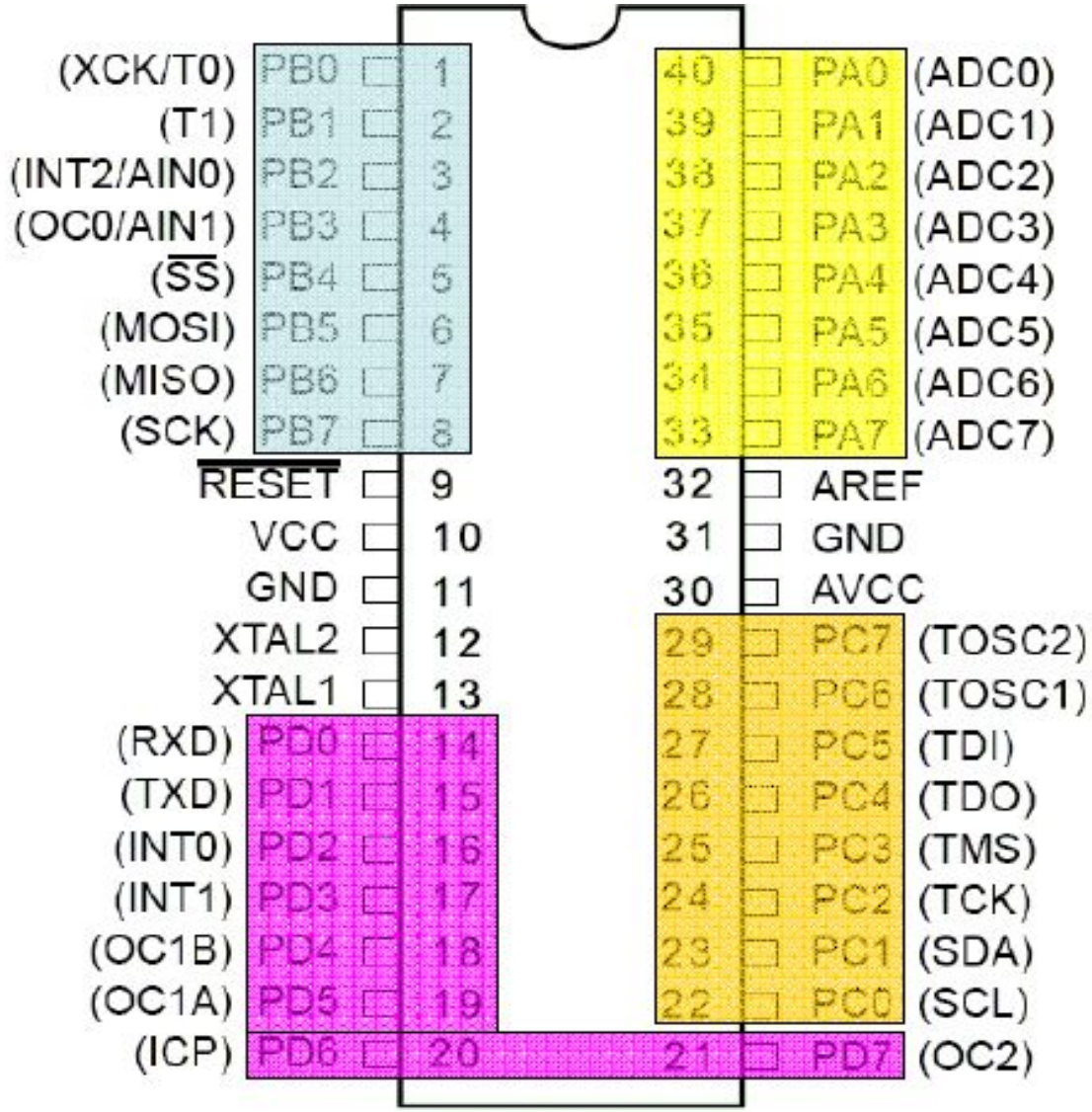
MD. IFTEKHARUL ISLAM SAKIB

BASIC I/O in ATmega32/16

Ports for I/O

- 4 different ports for I/O
  - A, B, C, D
  - 8 bit <-> 8 data pins
  - Every pin is bidirectional, can be used as input or output

I/O Ports of ATmega32 (AVR)



Port Operation Registers

- DDRx – Data Direction Register
- PORTx – Pin Output Register
- PINx – Pin Input Register

Configuration

- For configuration we have to use the Data Direction Registers
  - One register for each port
  - DDRx (DDRA, DDRB, DDRC, DDRD)
  - Configures each pin as input or output
- Pin configuration
  - Input – 0
  - Output – 1

C Code Example

- DDRA = 0b11111111;
  - Sets each bit of port A as output
- DDRB = 0b00000000;
  - Sets each bit of port B as input
- DDRC = 0b01010101;
  - ???

Input

- You have to read the PINx register

```
unsigned char ch;
ch = PINA;
```

- **What to do if only some of the pins are configured as input ??**

Output

- You have to use the PORTx register

```
PORTB = 0b11111111;
```

- **What to do if only some of the pins are configured as output ??**

Write a simple program to set the port B to 0xFF

C Code

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

int main(void)
{

    DDRD= 0b11111111; //initializing portD in
        //output mode
    PORTD=0b11111111;//writing value to portD

    while(1)
    {
        //TODO:: Nothing
    }
}
```

What will this code do ??

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

int main(void)
{

    DDRD= 0b01111111;
    PORTD=0b11111111;
    while(1)
    {
        //TODO:: Nothing
    }
}
```

Write a simple program to blink a LED on port B pin 0

C Code

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

int main(void)
{
unsigned char c = 1;
DDRB= 0b00000001;

    while(1)
    {
      PORTB = c;
      if(c)c=0;
      else c=1;
      delay();
      }
}
```

```
void delay()
{
unsigned char i,j,k;
for(i=0;i<255;i++)
for(j=0;j<255;j++)
for(k=0;k<100;k++);

}
```

Write a simple program to animate 8 LEDs connected to PORT B

- One LED at a time is ON
- The rest are off
- At first LED 0 is on, then LED 1, and so on.

C Code

```
int main(void)
{
    unsigned char c = 0x01;
    DDRB= 0xFF;

    while(1)
    {
      PORTB = c ;
      if(c==1<<7)c = 1;
      else c = c << 1;
      delay();
     }
}
```

Accurately Generating Delay

```
#include <avr/io.h>
#define F_CPU 1000000 // Clock Frequency
#include <util/delay.h>

int main(void)
{

    while(1)
    {
   //1000 ms delay
   _delay_ms(1000);

     }
}
```

Simple Input

- A 8 bit input is connected to PORT A
- Show the input state on PORT B

C Code

```
int main(void)
{
    unsigned char c;
    DDRA = 0x00;
    DDRB = 0xFF;

    while(1)
    {
      c = PINA;
      PORTB = c ;
    }
}
```

Simple Counter

- PA0 is connected with push button
  - 1 when pressed
- PORT B connected to 8 LEDs
- Increment count when pressed the push button

C Code

```
int main(void)
{
unsigned char c=0,in=0;
DDRA = 0xFE;
DDRB = 0xFF;

     while(1)
     {
    PORTB = c;

    in = PINA;
    if(in)
    {
        c++;
        _delay_ms(1000);
    }
     }
}
```

C Code

```
int main(void)
{
unsigned char c=0,in=0;
DDRA = 0xFE;
DDRB = 0xFF;

     while(1)
     {
    PORTB = c;

    in = PINA;
    if(in)
    {
        c++;
        _delay_ms(1000);
    }
     }
}
```

There is still a problem in the design.

C Code

```
int main(void)
{
    unsigned char c=0,in=0;
    DDRA = 0xFE;
    DDRB = 0xFF;

    PORTB = c;
     while(1)
     {
        in = PINA;
        if(in & 0x01)
        {
            c++;
            PORTB = c;
            _delay_ms(1000);
        }
     }
}
```

Write a program to read a byte from PORT A and write it to PORT B

If input (taken from *PORTC*) is less than 100, send it to *PORTB*, otherwise, send it to *PORTD*

Write a program to read a byte from PORT A and write its upper nibble to PORT B (lower nibble) and lower nibble to PORT C (upper nibble)

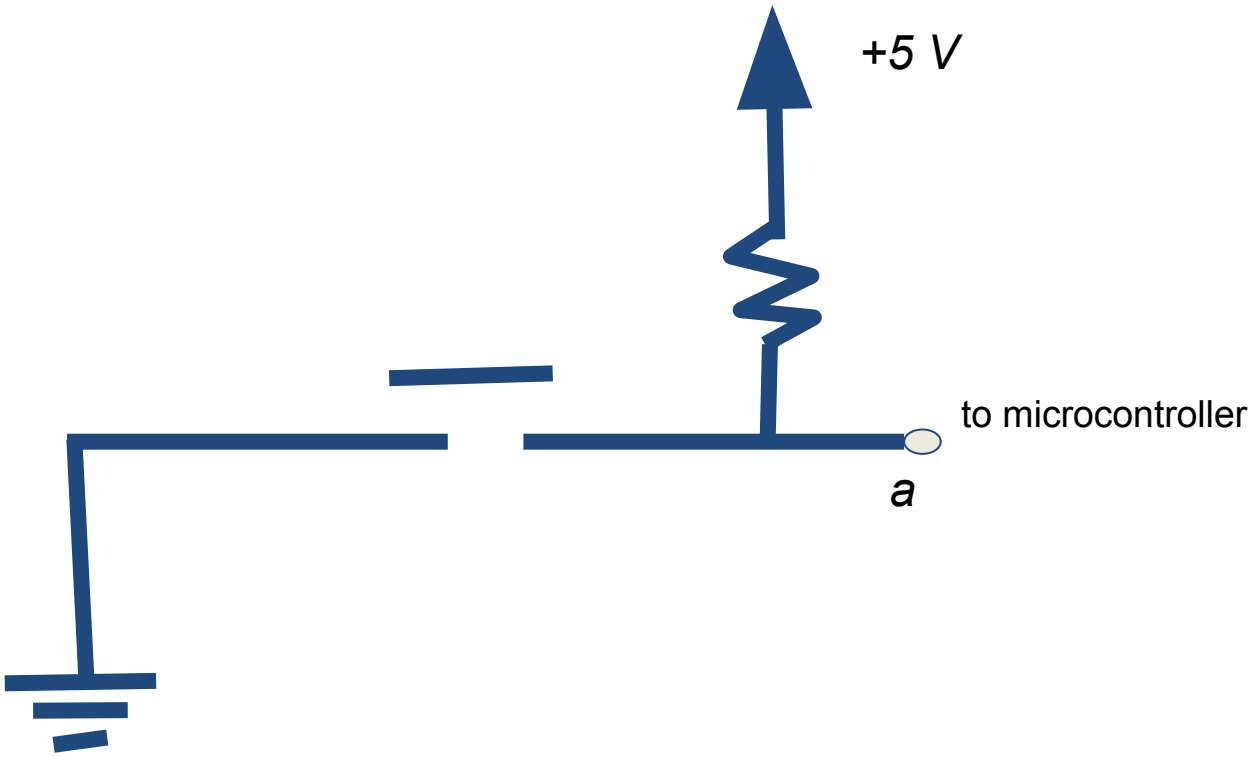
Discussions on push buttons

- This might seem like the reasonable connection
  - what is wrong with it ??

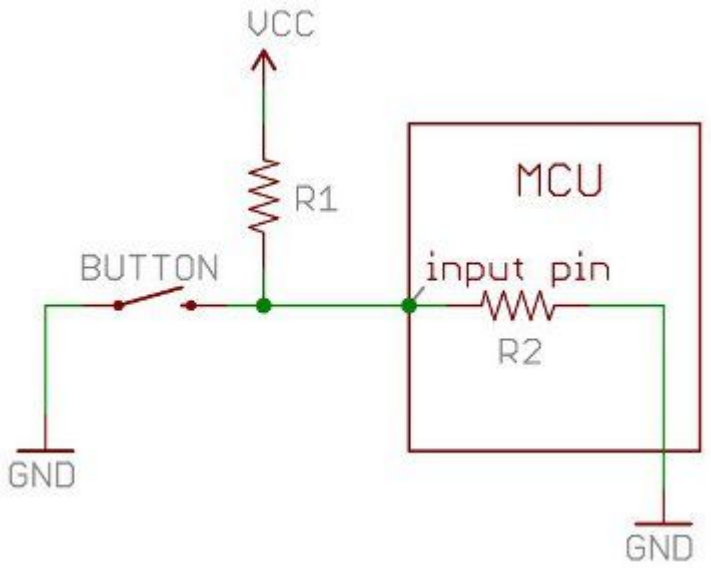
Discussions on push buttons

- This might seem like the reasonable connection
  - what is wrong with it ??
  - What is the voltage at terminal *a* when the button is not pressed?

Pull Up Resistors

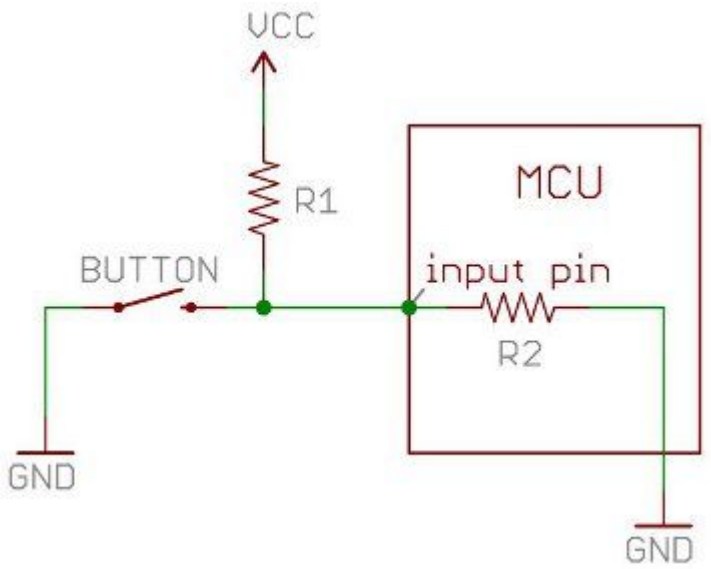


A more deeper look



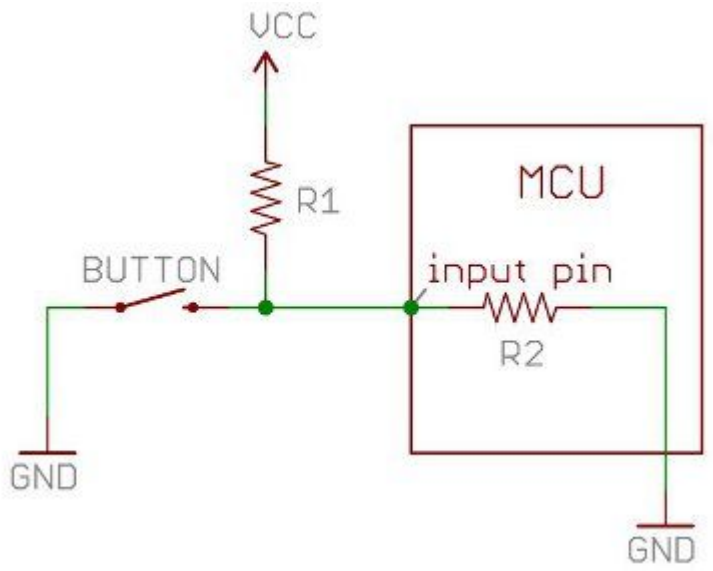
A more deeper look

- When the button is pressed, the input pin is pulled low. The value of resistor R1 controls how much current we want to flow from VCC, through the button, and then to ground.



A more deeper look

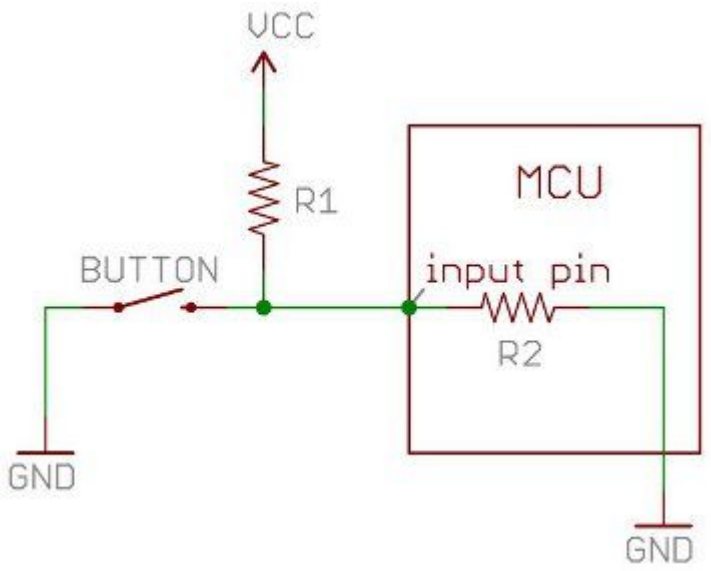
- When the button is not pressed, the input pin is pulled high. The value of the pull-up resistor controls the voltage on the input pin.



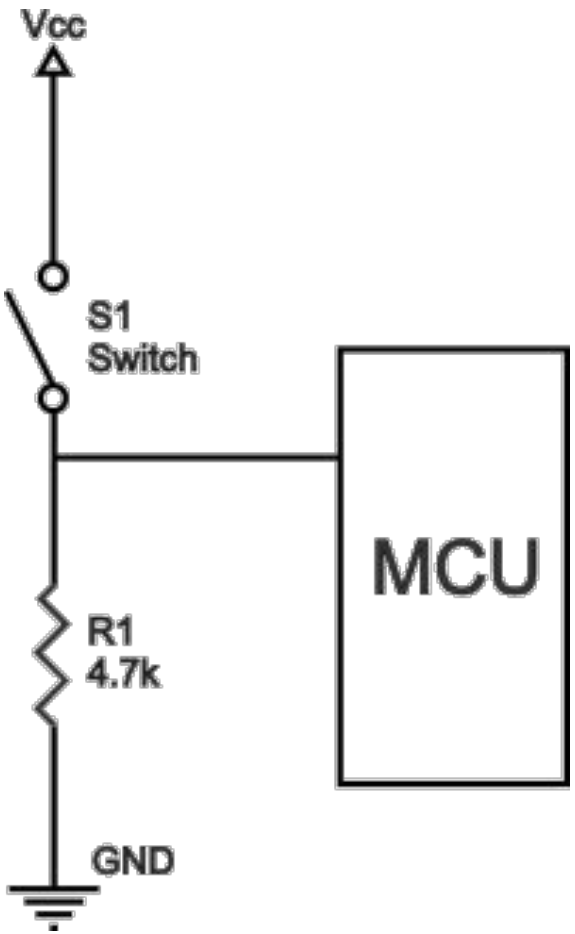
A more deeper look

- Because of the two opposing factors the resistor value cannot be too high not too less

- Depending on the context you choose an appropriate value



Pull Down Resistor



ATmega 32 has internal pull up resistors!!

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated.

Here, x = A,B,C, or D

n is bit number, e.g., 0,1,2,..., or 7

General Discussion

- If some pins are unused, it is recommended to ensure that these pins have a defined level.
- The simplest method to ensure a defined level of an unused pin, is to enable the internal pullup.

Resources

- DataSheet
- Pull Up and Pull Down Resistors
  - http://www.resistorguide.com/pull-up-resistor_pull-down-resistor/
- More on Pull Up and Pull Down resistors
  - https://learn.sparkfun.com/tutorials/pull-up-resistors