Embedded Programming AVR

C++ for Developers

Content

- Numerical Bases
- Bit Manipulation
- Using Bit Manipulation
- Typical program
- ATMega328p Architecture
- Including AVR files

lafftale
for developers

Numerical Bases

C++ for Developers

Numerical Bases (Radixes)

A base (or radix) is the number of unique digits (including zero) used to represent numbers in a positional numeral system.

Where each position represents a power of the base.

lafftale
for developers

Base 10 (decimal)

The “normal” base we use in everyday life. We have 10 different digits used to describe a number.

For example 9372 in a base 10 system would be:

(9 x 10 ^ 3) + (3 x 10 ^ 2) + (7 x 10 ^ 1) + (2 x 10 ^ 0) =

(9 x 1000) + (3 x 100) + (7 x 10) + (2 x 1) =

9000 + 300 + 70 + 2 = 9372

lafftale
for developers

Base 2 (Binary)

In lower levels we often use the binary system. It works the same but with base 2. To describe the same number 9732 it would like this: 00**1**0 0**1**00 **1**00**1** **11**00

(0x2^15) + (0x2^14) + **(1x2^13)** + (0x2^12) + (0x2^11) + **(1x2^10)** + (0x2^9) + (0x2^8) + **(1x2^7)** + (0x2^6) + (0x2^5) + **(1x2^4)** + **(1x2^3)** + **(1x2^2)** + (0x2^1) + (0x2^0) =

8192 + 1024 + 128 + 16 + 8 + 4 = 9732

lafftale for developers

Bits - Binary Digits

Common technology is based on the binary system where it is:
1 or 0
High or low
On or off

One byte = eight bits.

So for example when we declare an **int** which holds 4 bytes - it hold 32 bits in memory where the largest number can be:
1111 1111 1111 1111 1111 1111 1111 1111 or in decimal 4 294 967 295

lafftale
for developers

Signed / unsigned

Remember when we talked about signed and unsigned variables? This refers to the most significant bit (MSB) deciding if the number is positive(0) or negative(1) - signing the number. That's why an integer can hold a number between:

0111 1111 1111 1111 1111 1111 1111 1111 (2 147 483 647)
1000 0000 0000 0000 0000 0000 0000 0000 (- 2 147 483 647)

The negative number is written in what is called **two's complement**

lafftale for developers

Base 16 (hexadecimal)

Also often used when coding embedded. Adds the digits : A(10) , B(11), C(12), D(13), E(14) and F(15) to the base 10 system. 9372 would look like this:

249C =

(2 x 16 ^ 3) + (4 x 16 ^ 2) + (9 x 16 ^ 1) + (C(12) x 16 ^ 0) =
(2 x 4096) + (4 x 256) + (9 x 16) + (12 x 1) =
8192 + 1024 + 144 + 12 = 9382

lafftale
for developers

Embedded use

Both base 2 (binary) and base 16 (hexadecimal) are commonly used in low-level code.
Memory addresses and hardware registers are often written in hexadecimal, using the prefix 0x to indicate that the value is base 16. For example: 0x00 or 0xFF.

In embedded systems, functionality is often controlled through flags (or bits) that can be either HIGH (1) or LOW (0).
Binary literals can be written directly in C++ using the prefix 0b, for example: 0b00101000

lafftale for developers

Bit Manipulation

C++ for Developers

What is bit manipulation?

This is a way to manipulate and read bits in a logical way.

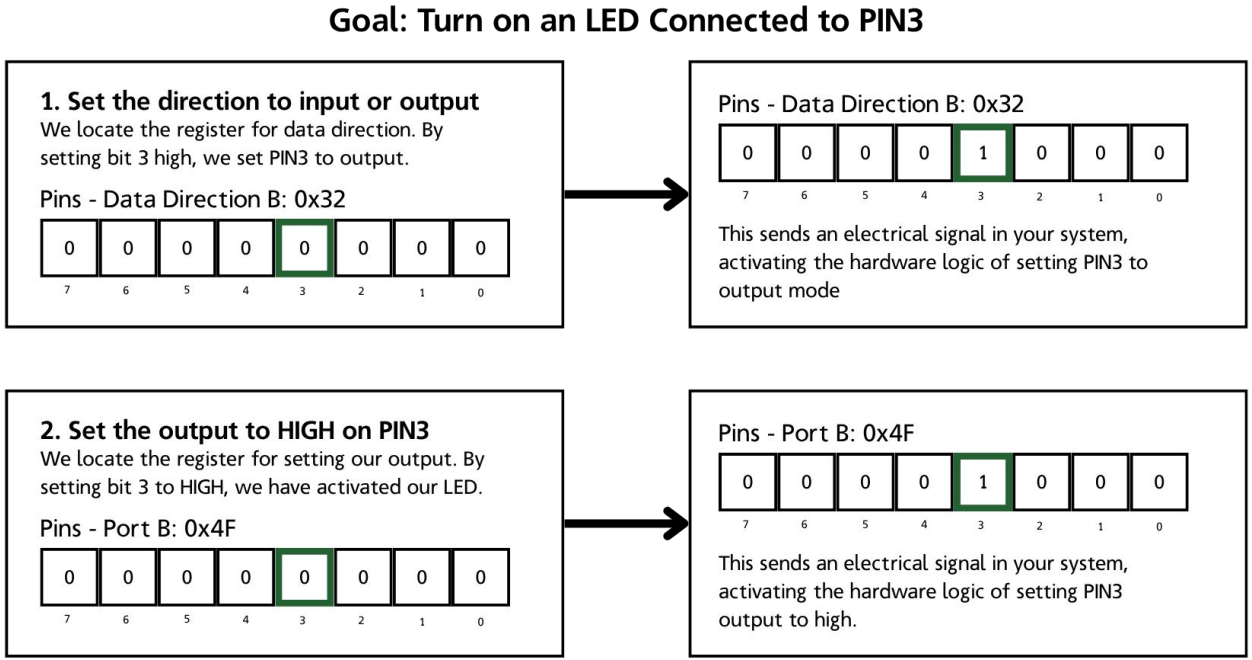
For example:

Instead of having a boolean we can bit 16 booleans into an integer by reading each bit as a true or false value.

0010 0110 1100 0101

lafftale
for developers

Example: Turning on a LED



lafftale for developers

Registers

By using bit manipulation - we can set the registers in our MCU.

By setting bit number 3 to 1 - we have activated the functionality in our hardware.

lafftale
for developers

Shift operators

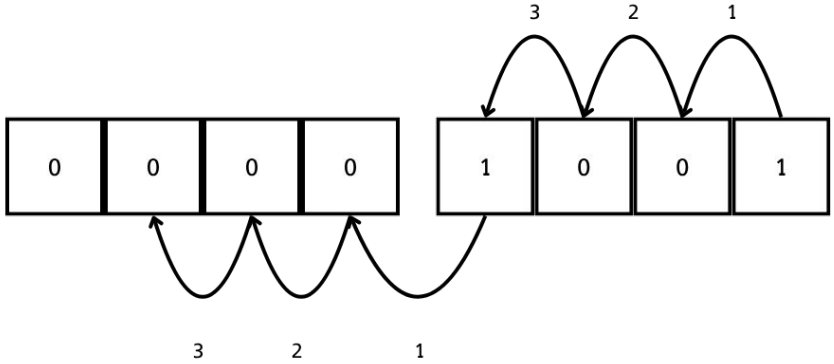
By using << or >> we “shift” the bits to the left (<<) or the right (>>).

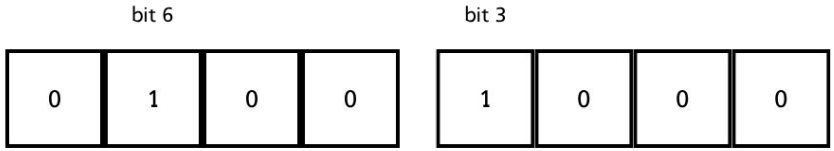
0001 0110 << 2 = 0101 1000
0001 0110 << 3 = 1011 0000

If we take a look at the decimal values:
0010 0110 = 22
1001 1000 = 88 (22 * 2 ^ 2)
1011 0000 = 176 (22 * 2 ^ 3)

lafftale
for developers

Statement: 9 << 3





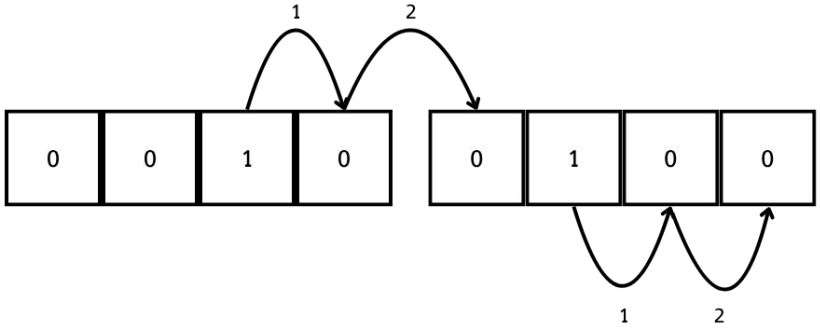
Result: 72

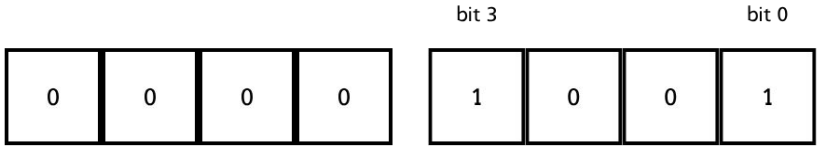
Remembering the binary system:

1 * 2 ^ 6 + 1 * 2 ^ 3 =
64 + 8 =
72

lafftale
for developers

Statement: 36 >> 2





Result: 9

Remembering the binary system:

1 * 2 ^ 3 + 1 * 2 ^ 0 =
8 + 1 =
9

lafftale
for developers

Shift operators

We can now see that:

<< operator

- number * 2 ^ shift

>> operator

- number / 2 ^ shift

lafftale
for developers

Why are they used?

If we want to set bit five in a register, instead of writing:

- 32 (which in binary is 0x0010 0000, the third bit)
  How would we even do this dynamically?

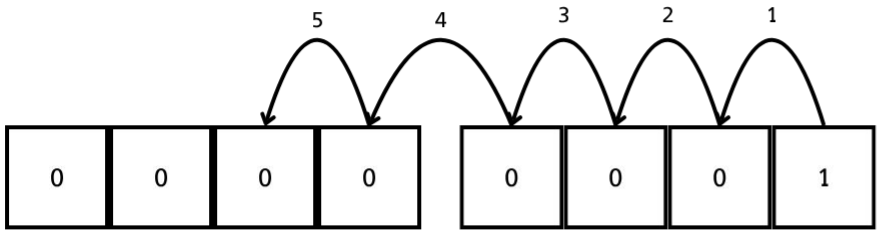
We can instead target it by doing:

- 1 << 5 (which in binary also is 0x0010 0000)

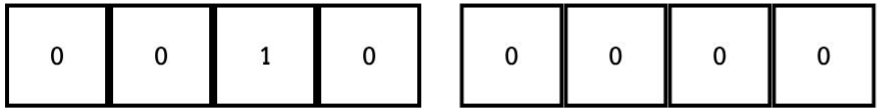
lafftale
for developers

Setting a register would look like this

Statement: 1 << 5



Result: Bit 5 is now set!



lafftale
for developers

Boolean Operators

To further our bit manipulation, we make use of boolean operators.

- | *or*-operator. One need to be true
- & *and*-operator. Both need to be true
- ~ *not*-operator. Negate the statement
- ^ *xor*-operator. True if not equal

lafftale
for developers

| *or*-operator

Statement: 1010 | 0010

1	0	1	0	Number 1
0	0	1	0	Number 2
1 | 0	0 | 0	1 | 1	0 | 0	Comparison
1	0	1	0	Result

Returns 1 if any (or both) of the two comparing bits are 1. Otherwise 0.

lafftale for developers

& *and*-operator

Statement: 1010 & 0010

1	0	1	0	Number 1
0	0	1	0	Number 2
1 & 0	0 & 0	1 & 1	0 & 0	Comparison
0	0	1	0	Result

Returns 1 if both of the two comparing bits are 1. Otherwise 0.

lafftale for developers

~ *not*-operator

Statement: ~(1010 & 0010)

1	0	1	0	Number 1
0	0	1	0	Number 2
1 & 0	0 & 0	1 & 1	0 & 0	Comparison
0	0	1	0	Before negation
1	1	0	1	Result

Negates the the value.

lafftale
for developers

^ *xor*-operator

Statement: 1010 ^ 0010

1	0	1	0	Number 1
0	0	1	0	Number 2
1 ^ 0	0 ^ 0	1 ^ 1	0 ^ 0	Comparison
1	0	0	0	Result

Will return true if the comparing bits are different. False if they are equal.

lafftale for developers

Implementation of bit manipulation

C++ for Developers

Setting registers (or-operator)

```
// Set the data direction to output
DDRB |= (1 << LED_PIN);
```

DDRB is a register with 8 bits: 0000 0000
LED_PIN is PIN5 in this register.

To set it we:

- Bit shift: 0000 0001 << 5 = 0010 0000
- Use the or-operator: 0000 0000 | 0010 0000 = 0010 0000

Now we have set that bit in register!

lafftale for developers

Unsetting registers (and- and not-operator)

```
// Set the output to low
PORTB &= ~(1 << LED_PIN);
```

PORTB is a register with 8 bits where bit number 5 is 1: 0010 0000

To unset it we:

- Bit shift: 0000 0001 << 5 = 0010 0000
- Use not-operator: ~(0010 0000) = 1101 1111
- Use and-operator: 0010 0000 & 1101 1111 = 0000 0000

If other bits where set in the register, they will not be changed!

lafftale for developers

Toggling using the xor-operator

```
while(1) {
    PORTB ^= (1 << LED_PIN);
    _delay_ms(500);
}
```

While the loop runs we want to toggle the 5th bit in register PORTB.

If bit is set: 0010 0000

- shift operation result: 0010 0000
- xor-operation: 0010 0000 ^ 0010 0000 = 0000 0000.

Bit is now unset. This operation does not affect other bits in the register.

If bit is not set: 0000 0000

- shift operation result: 0010 0000
- xor-operation: 0000 0000 ^ 0010 0000 = 0010 0000.

Bit is now set.

lafftale for developers

Reading input of a pin

```c
// GLOBALS
#define LED_PIN     PB5
#define BUTTON_PIN    PB4

int main(void) {

    // SETUP
    DDRB |= (1 << LED_PIN);
    PORTB &= ~(1 << LED_PIN);

    DDRB &= ~(1 << BUTTON_PIN);
    PORTB &= ~(1 << BUTTON_PIN);

    // SUPER LOOP
    while(1) {

        if (PINB & (1 << BUTTON_PIN)) {
            PORTB |= (1 << LED_PIN);
        } else {
            PORTB &= ~(1 << LED_PIN);
        }

        _delay_ms(5);
    }

    return 0;
}
```

```c
if (!(PINB & (1 << BUTTON_PIN))) {
    PORTB |= (1 << LED_PIN);
} else {
    PORTB &= ~(1 << LED_PIN);
}
```

Checking if pin5 is high, then button is pressed.

While not pressed:
PINB = 0000 0000
BTN = 0010 0000
PINB & BTN = false

While pressed:
PINB = 0010 0000
BTN = 0010 0000
PINB & BTN = true

lafftale for developers

Embedded Bare Metal Structure

C++ for Developers

File structure

- Global variables
- Setup
- Super-loop

lafftale
for developers

File structure

Global Variables

Declare variables used in the global scope

Setup

Initialize the components used in our program

Super-loop

The repeated task(s) performed by our MCU

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

/*
GLOBAL VARIABLES
Declare and assign global variables.
*/

int main (void) {
    /*
    SETUP
    Initialize your drivers. This can include
    setting registers, setting a base state in
    other files and so on.
    */

    while(true) {
        /*
        SUPER-LOOP
        This is where the program runs until the MCU
        is turned off.
        */
    }

    // This point should never be reached
    return 0;
}
```

lafftale for developers

Example LED program

- Defining my LED_PIN in global scope
- Setting up data direction and level
- Toggle the pin high and low

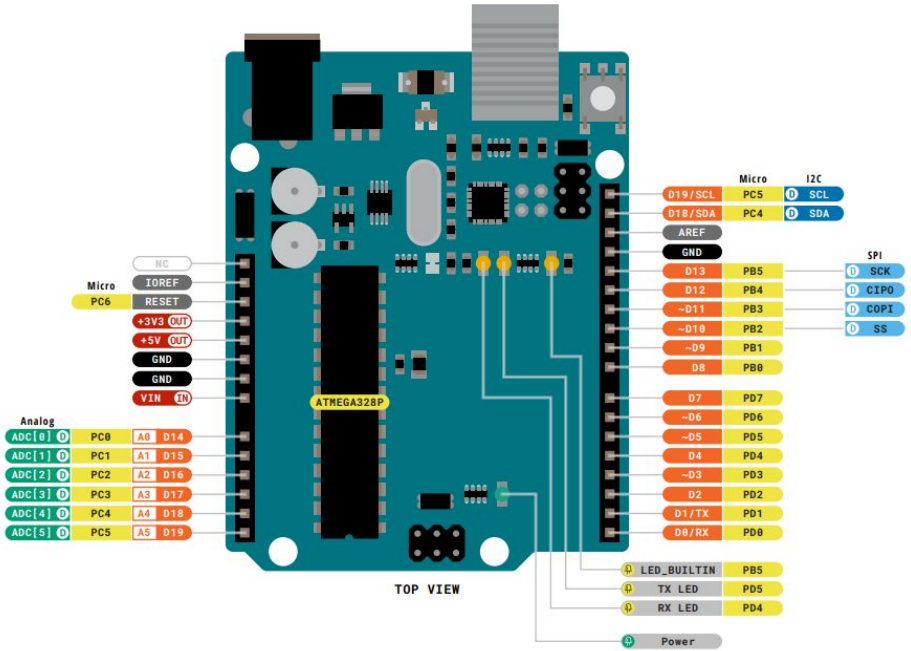
```c
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

// GLOBALS
#define LED_PIN PB5

int main(void) {

    // SETUP
    DDRB |= (1 << LED_PIN);
    PORTB &= ~(1 << LED_PIN);

    // SUPER LOOP
    while(1) {
        PORTB ^= (1 << LED_PIN);
        _delay_ms(500);
    }

    return 0;
}
```

lafftale for developers

ATMega328p Architecture

C++ for Developers

Pin out



Digital I/O

Can output or read HIGH/LOW

Analog Input

Can read voltage between 0 and 5V using ADC.

PWM

Can pulse at a set frequency, simulating different voltage outputs

Serial Communication

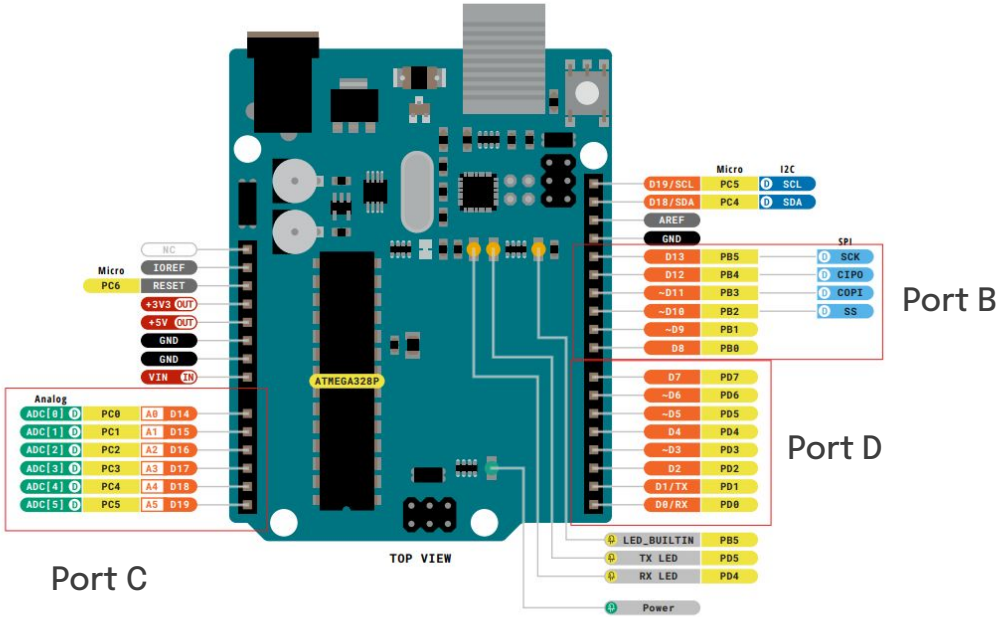
Specific pins connected to the peripherals used for different communication protocols.

Power Pins

5V, 3.3V and GND

lafftale for developers

Pin out

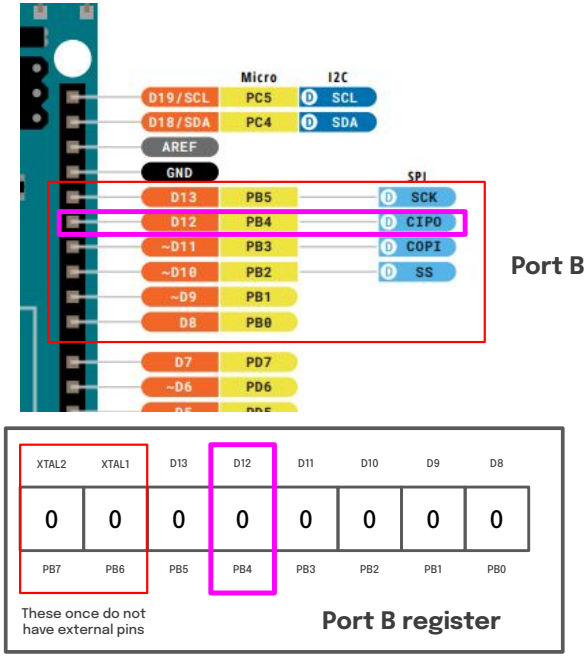


Ports

Instead of controlling each pin separately through different registers - AVR designers grouped 8 pins together in blocks called **Ports**

lafftale for developers

Pin out



Pin mapping

While the pins official name might be DIGITAL12 or GPIO12 - it is represented differently internally.

← In the example we can see that D8→D13 is mapped to PB(Port B) PB0→PB5.

This means that internally, D12 is actually represented as bit 4 in Port B.

The pin mapped names are actually included in the AVR library avr/io.h, so you can use PB4 in your program.

lafftale for developers

Important registers for each PORT

You have the following registers which are good to keep track of. This example will use Port **B**.

- DDR**B** - Data Direction Register B
- PORT**B** - Depending on the Data Direction of the pin:
  - Enables/Disables internal pull-up resistor (handle floating values)
  - Sets pin HIGH/LOW
- PIN**B** - Check the current logical level of a physical pin

lafftale for developers

Create an LED project

C++ for Developers

Project: Light an LED while button is pressed

LED

- Set direction to output
- Set initializing output to 0

Button

- Set direction to input
- Disable internal pull-up resistor (we will have an external)
- Check if button is pushed

lafftale
for developers

1. Include the correct headers

Make sure to include the correct headers. For accessing the GPIOs on the ATmega328p and a blocking delay include:

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

lafftale for developers

2. Use defines to make the program more readable

After your includes, use defines to make your code more readable:

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

// GLOBALS
#define LED_PIN     PB5
#define BUTTON_PIN    PB4

int main(void) {
```

lafftale for developers

3. Initialize the LED

We set bit 5 in data direction register B to 1. Enabling output on GPIO13.

We then set bit 5 in the port register B to 0. Setting the output to low on GPIO13.

```
DDRB |= (1 << LED_PIN);
PORTB &= ~(1 << LED_PIN);
```

lafftale
for developers

4. Initialize the Button

We set bit 4 in data direction register B to 0. Setting GPIO12 to input mode.

We then set bit 4 in port register B to 0. Disabling the internal pull-up resistor for GPIO12.

```
DDRB &= ~(1 << BUTTON_PIN);
PORTB &= ~(1 << BUTTON_PIN);
```

lafftale
for developers

5. Logic in the super-loop

While(the mcu is powered on)

1. Check if the logical input on GPIO12 is high (bit 4 in the pin level register on port B).
   a. *If the logical input on GPIO12 is high*
      i. Set bit 5 in port register B to 1 - setting the output for GPIO13 to high.
   b. *Else*
      i. Unset bit 5 in port register B to 0 - setting the output for GPIO13 to low.
2. Wait 5 milliseconds

```
// SUPER LOOP
while(1) {

    if (PINB & (1 << BUTTON_PIN)) {
        PORTB |= (1 << LED_PIN);
    } else {
        PORTB &= ~(1 << LED_PIN);
    }

    _delay_ms(5);
}
```

lafftale for developers

```c
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>

// GLOBALS
#define LED_PIN     PB5
#define BUTTON_PIN    PB4

int main(void) {

    // SETUP

    DDRB |= (1 << LED_PIN);
    PORTB &= ~(1 << LED_PIN);

    DDRB &= ~(1 << BUTTON_PIN);
    PORTB &= ~(1 << BUTTON_PIN);

    // SUPER LOOP
    while(1) {

        if (PINB & (1 << BUTTON_PIN)) {
            PORTB |= (1 << LED_PIN);
        } else {
            PORTB &= ~(1 << LED_PIN);
        }

        _delay_ms(5);
    }

    return 0;
}
```

Link to wokwi project:
https://wokwi.com/projects/445516715245466625

Link to GitHub project:
https://github.com/lafftale1999/cpp_for_developers/tree/main/week_5/3_use_cases_bit_manipulation/simple_led_project

Link to runnable .hex file:
https://drive.google.com/file/d/1J55hJkRvq13mgTq0UG6hni31FN5hQ3LG/view?usp=drive_link

lafftale for developers

Try running the project

1. Download the .hex file
2. Open the wokwi project
3. Open the command palette:
    a. Right click in the code segment - press command palette
    b. Left click in the code segment and press F1
4. Click on "Upload Firmware and Start Simulation..."
5. Upload the .hex file you downloaded

lafftale for developers


lafftale
for developers