Heaven's Light Is Our Guide

Rajshahi University of Engineering & Technology
Department of Computer Science & Engineering

CSE 3105
Computer Interfacing & Embedded System

Digital Inputs, Outputs and PWM

Md. Nasif Osman Khansur
Lecturer
Dept. of CSE, RUET

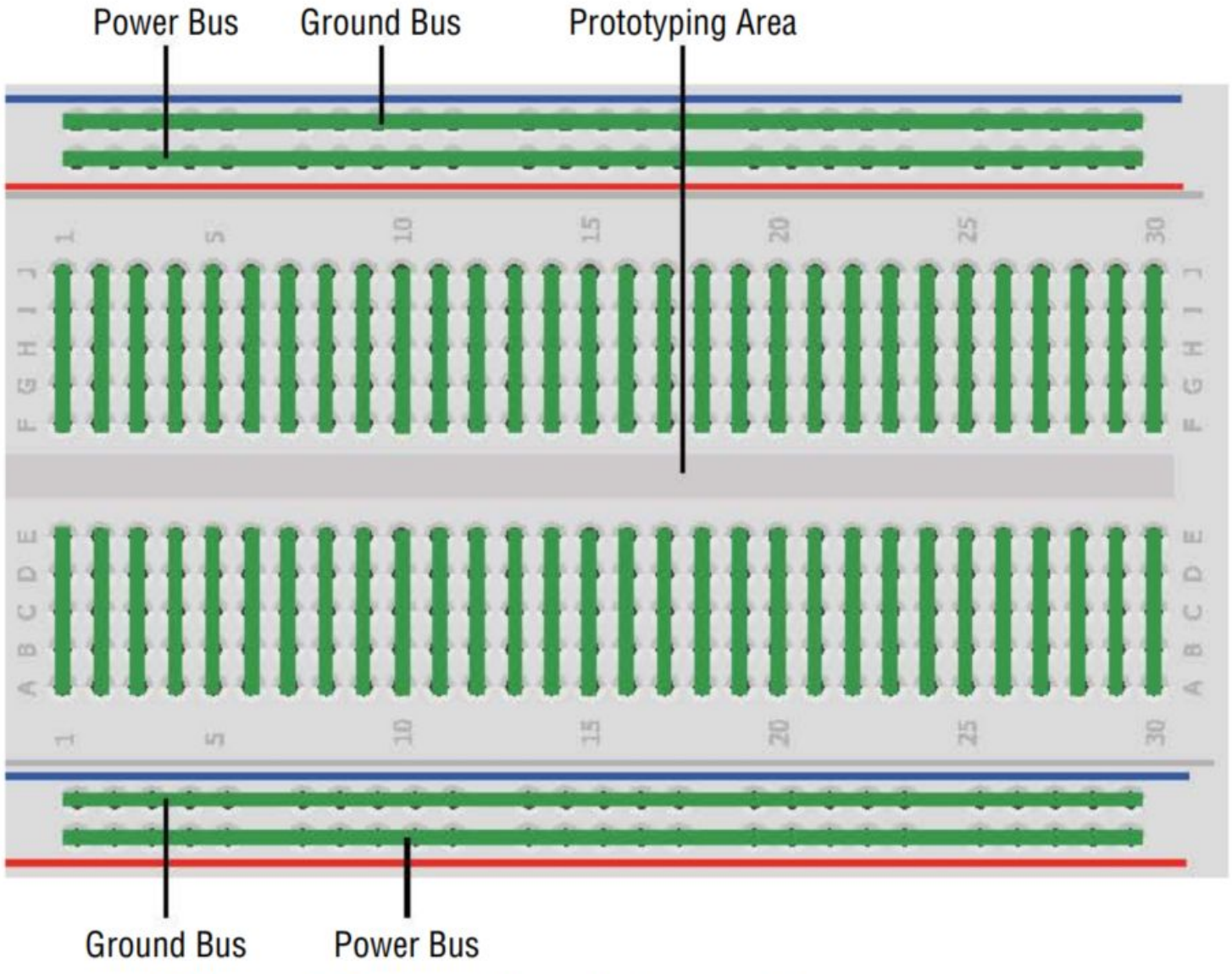
References

1. Exploring Arduino 2e by Jeremy Blum [Chapter 2]

PIN Functions

Functionality	**Arduino Uno**	**STM32 Blue Pill**
Digital Pins (Input/Output)	0 - 13 A0 - A5	PA0 - PA15 PB0 - PB15 PC13 - PC15
Analog Pins (Input only)	A0 - A5	PA0 - PA7 PB0 - PB1
PWM Pins	3, 5, 6, 9, 10, 11	PA0 - PA3 PA6 - PA10 PB0 - PB1 PB6 - PB9

Breadboard



A ***breadboard*** is a simple prototyping tool that allows you to easily wire up simple circuits without having to solder together parts to a custom-printed circuit board.

The pins adjacent to the **red** and **blue** color-coded lines are designed to be used as *power* and *ground* buses. All the **red** pins are electrically connected, and are generally used for providing **power** (5V). All the **blue** pins are electrically connected and are used for the **ground** bus. All the vertically aligned pins are also connected in rows, with a division in the middle to make it easy to mount integrated circuits (ICs) on the breadboard.

Structure of a Program

```
void setup()
{

}

void loop()
{

 }
```

The setup() function runs one time at the start of the program.

Structure of a Program

```
void setup()
{

}

void loop()
{

 }
```

The loop() function runs over and over again.

Structure of a Program

1. /* … */ : This is a multiline comment. Comments are important for documenting your code. Whatever you write between these symbols will not be compiled or even seen by your Arduino. Multiline comments start with /* and end with */.
2. // : This is a single-line comment. When you put // on any line, the compiler ignores all text after that symbol on the same line. This is great for annotating specific lines of code or for "commenting out" a particular line of code that you believe might be causing problems.
3. Void setup() : A function is a piece of code that does a specific task. void setup() is one of two functions that must be included in every Arduino program. Code within the curly braces of the setup() function is executed once at the start of the program. This is useful for one-time settings, such as setting the direction of pins, initializing communication interfaces, and so on.
4. pinMode (): The Arduino's digital pins can all function as inputs or outputs. To configure their direction, use the command pinMode(). All pins default to inputs unless you explicitly tell the Arduino to treat them as outputs. Defining a pin as an output during the setup() will mean that the pin stays configured as an output for the duration of the program execution. The pinMode() command takes two arguments. *An argument gives commands information on how they should operate*. The first argument to pinMode() determines which pin is having its direction set. The second argument to pinMode() sets the direction of the pin: INPUT or OUTPUT.

Structure of a Program

5. Void loop () : The second required function in all Arduino programs is void loop(). The contents of the loop function repeat forever as long as the Arduino is on. If you want your Arduino to do something once at boot only, you still need to include the loop function, but you can leave it empty.

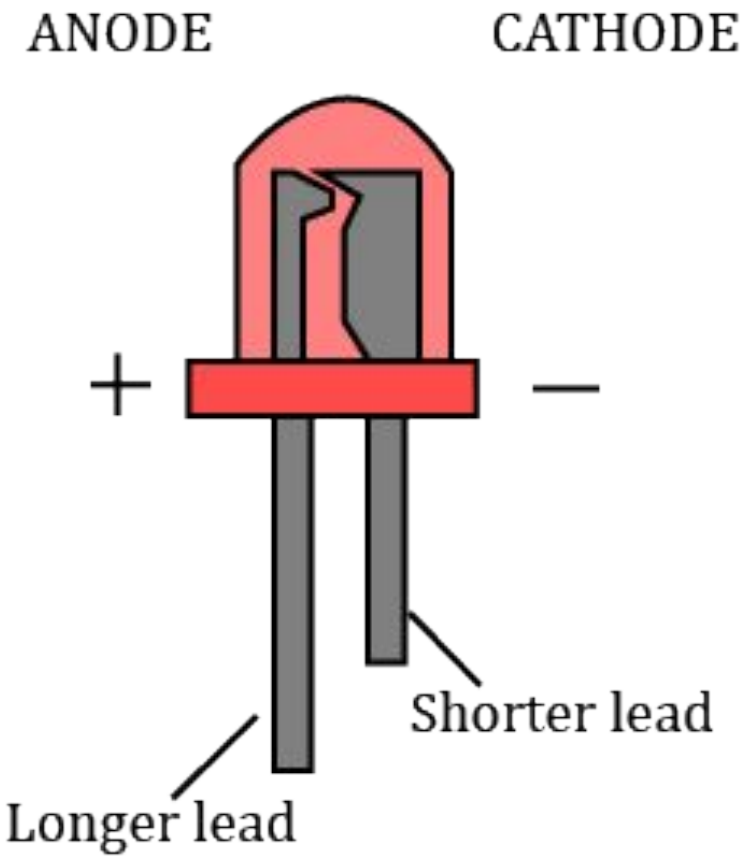
6. digitalWrite() is a command that is used to set the state of an output pin. It can set the pin to either 5V or 0V. When an LED is connected to a pin (through a current-limiting resistor), setting it to 5V will enable you to light up the LED. The first argument to digitalWrite() is the pin you want to control. The second argument is the value you want to set it to, either HIGH (5V) or LOW (0V). The pin remains in this state until it is changed later in the code.

7. The delay() function accepts one argument: a delay time in milliseconds. When calling delay(), the Arduino stops doing anything for the amount of time specified.

Programming Digital Outputs

By default, all Arduino pins are set to inputs. If you want to make a pin an output, you need to first tell the Arduino how the pin should be configured.

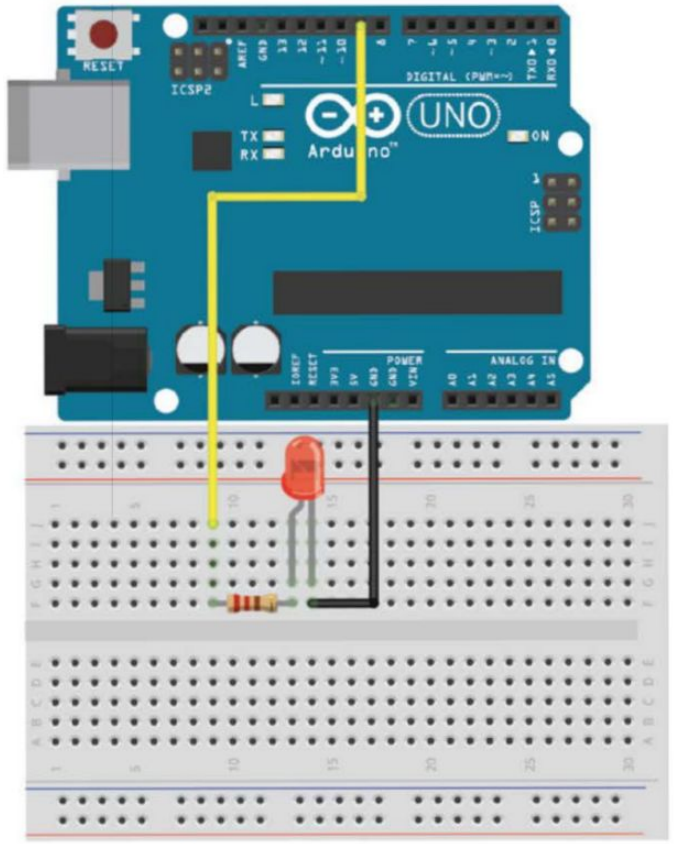
LED

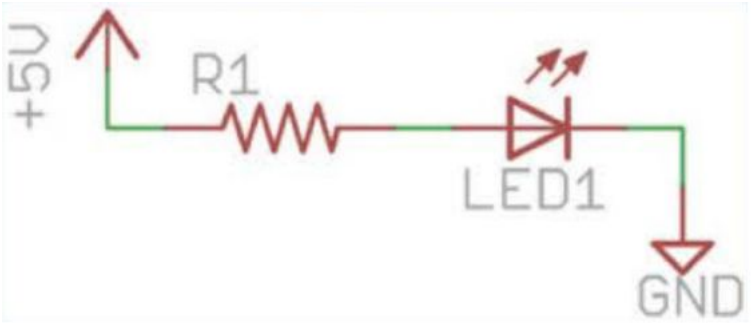


The *longer lead* of the LED is called **anode** and is the positive terminal, so should be connected to the positive terminal of the battery. Similarly, the *shorter lead* of the LED is called **cathode** and is negative terminal, so it is connected to the negative terminal of the battery.

[Link]

Turning On LED





Turning On LED

```
const int LED=PC13; // Define LED for pin PC13
```

Turning On LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}
```

Turning On LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
   digitalWrite(LED, HIGH);
   // Set the LED pin high
}
```

Turning On LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
   digitalWrite(LED, HIGH);
   // Set the LED pin high
}

void loop()
{
   // We are not doing anything in the loop!
 }
```

Blinking LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}
```

Blinking LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   digitalWrite(LED, HIGH);
 }
```

Blinking LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   digitalWrite(LED, HIGH);
   delay(1000); // Time in milliseconds
 }
```

Blinking LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   digitalWrite(LED, HIGH);
   delay(1000); // Time in milliseconds
   digitalWrite(LED, LOW);
 }
```

Blinking LED

```
const int LED=PC13; // Define LED for pin PC13

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   digitalWrite(LED, HIGH);
   delay(1000); // Time in milliseconds
   digitalWrite(LED, LOW);
   delay(1000);
 }
```

Using For Loops

It’s frequently necessary to use loops with changing variable values to adjust the parameters of a program.

In the case of the program you just wrote, you can implement a for loop to see how different blink rates impact your system’s operation. You can visualize different blink rates by using a for loop to cycle through various rates.

LED with changing blink rate

```
const int LED=PA15; // Define LED for pin PA15

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}
```

LED with changing blink rate

```
const int LED=PA15; // Define LED for pin PA15

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   for (int i=100; i<=1000; i=i+100)
   {

   }
 }
```

LED with changing blink rate

```
const int LED=PA15; // Define LED for pin PA15

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
}

void loop()
{
   for (int i=100; i<=1000; i=i+100)
   {
       digitalWrite(LED, HIGH);
       delay(i);
       digitalWrite(LED, LOW);
       delay(i);
   }
 }
```

Pulse-Width Modulation

What if you want to output a voltage other than 0V or 5V? Well, you can’t—unless you are using a digital to-analog converter (DAC) integrated circuit.

However, you can get pretty close to generating analog output values by using a trick called pulse-width modulation (PWM).

Pulse-Width Modulation

The PWM output is an 8-bit value. In other words, you can write values from 0 to 2^8–1, or 0 to 255.

In the case of your LED circuit, setting the output to 255 will result in full brightness, and 0 will result in the LED turning off, with the brightness varying between these two values.

Pulse-Width Modulation

PWM control can be used in a lot of circumstances to emulate pure analog control, but it cannot always be used when you actually need an analog signal.

Pulse-Width Modulation

If you are not actually changing the voltage being delivered to an LED, why do you see it get dimmer as you lower the duty cycle?

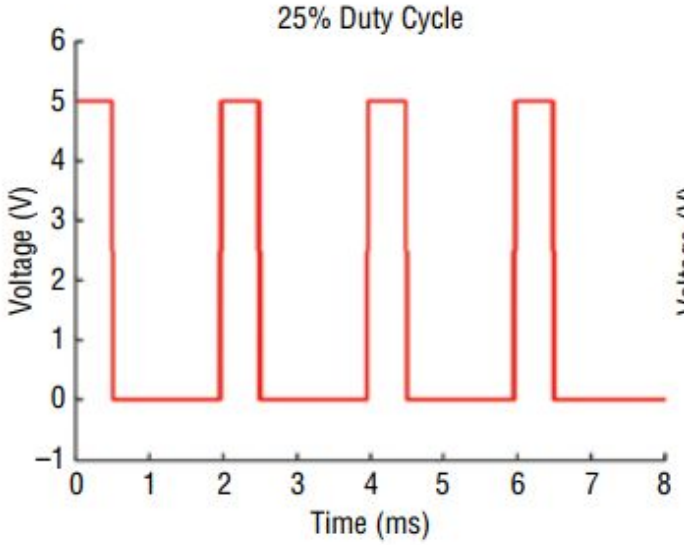
It is really a result of your eyes playing a trick on you! If the LED is switching on and off every 1 ms (which is the case with a duty cycle of 50 percent), it appears to be operating at approximately half brightness because it is blinking faster than your eyes can perceive. Therefore, your brain actually averages out the signal and tricks you into believing that the LED is operating at half brightness

Pulse-Width Modulation

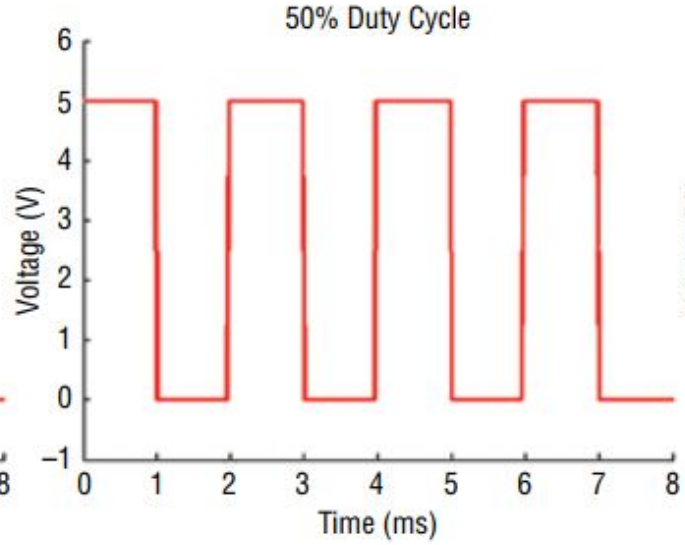
A similar effect is accomplished with DC motors. Because motors can't change speed instantaneously, duty cycling their power at 50 percent results in them running at about 50 percent of their maximum speed

Pulse-Width Modulation

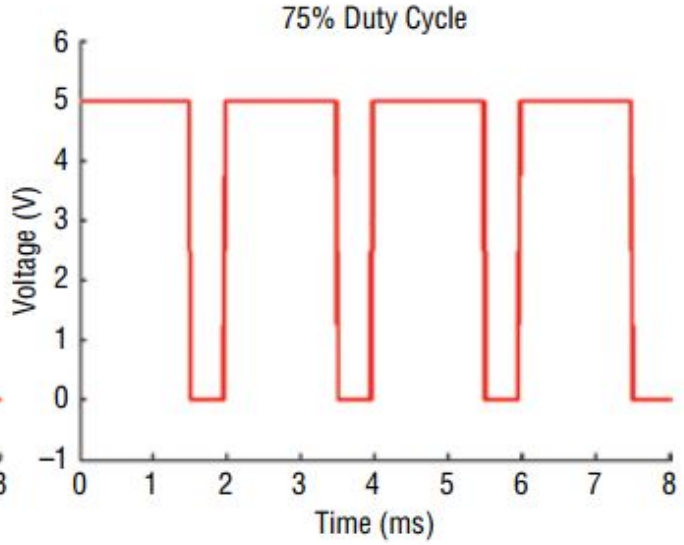
`analogWrite(PIN, 64)`



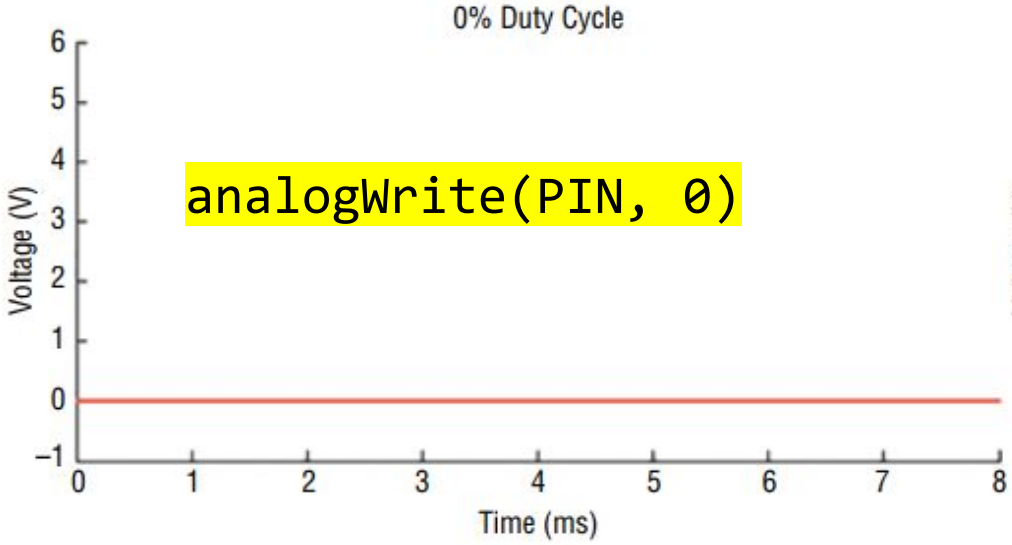
`analogWrite(PIN, 127)`



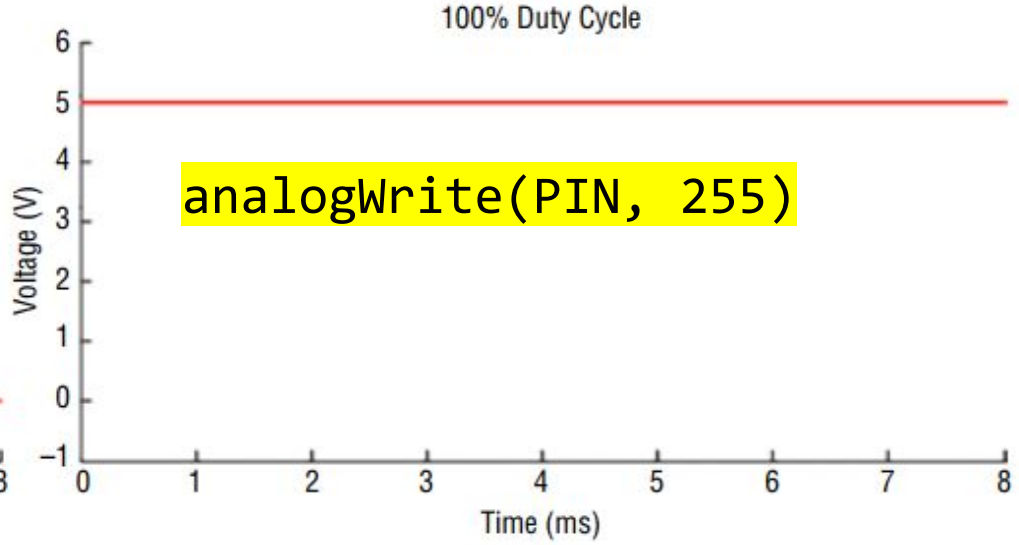
`analogWrite(PIN, 191)`



`analogWrite(PIN, 0)`



`analogWrite(PIN, 255)`



LED fade

```
void setup()
{
   pinMode (PB0, OUTPUT);
   //Set the LED pin as an output
}
```

LED fade

```
void setup()
{
   pinMode (PB0, OUTPUT);
   //Set the LED pin as an output
}

void loop()
{
  for (int i=0; i<256; i++)
  {

  }
 }
```

LED fade

```
void setup()
{
   pinMode (PB0, OUTPUT);
   //Set the LED pin as an output
}

void loop()
{
  for (int i=0; i<256; i++)
  {
    analogWrite(PB0, i);
    delay(10);
  }
 }
```

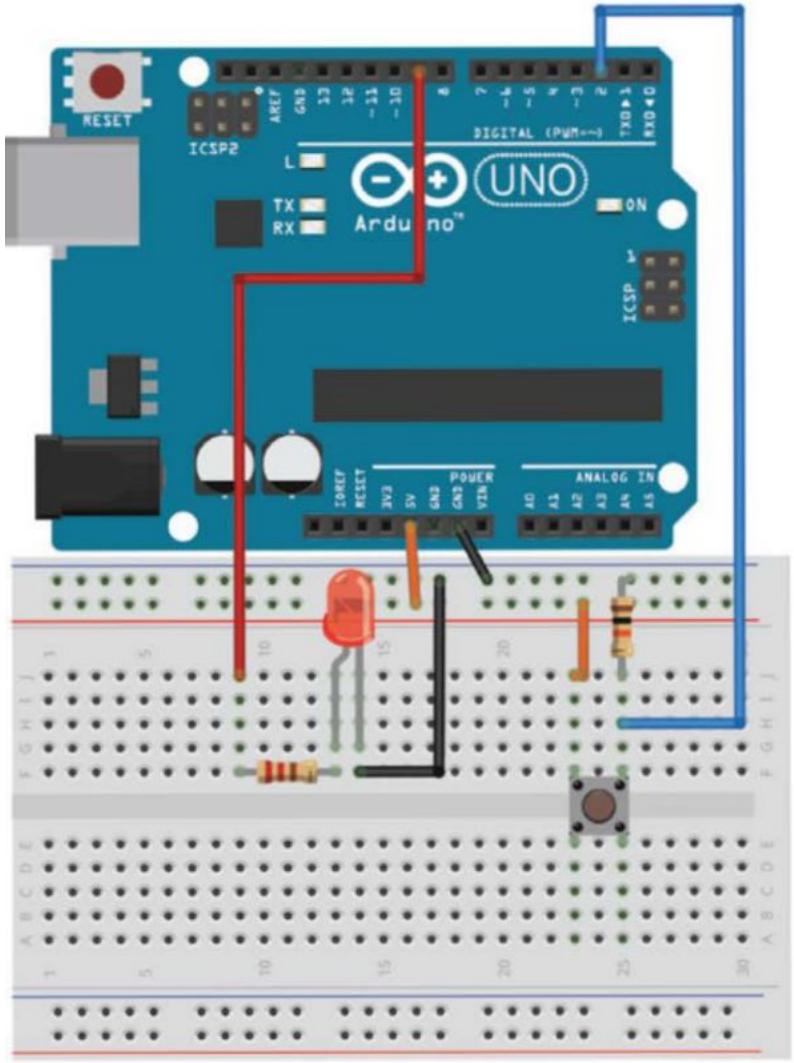
LED fade

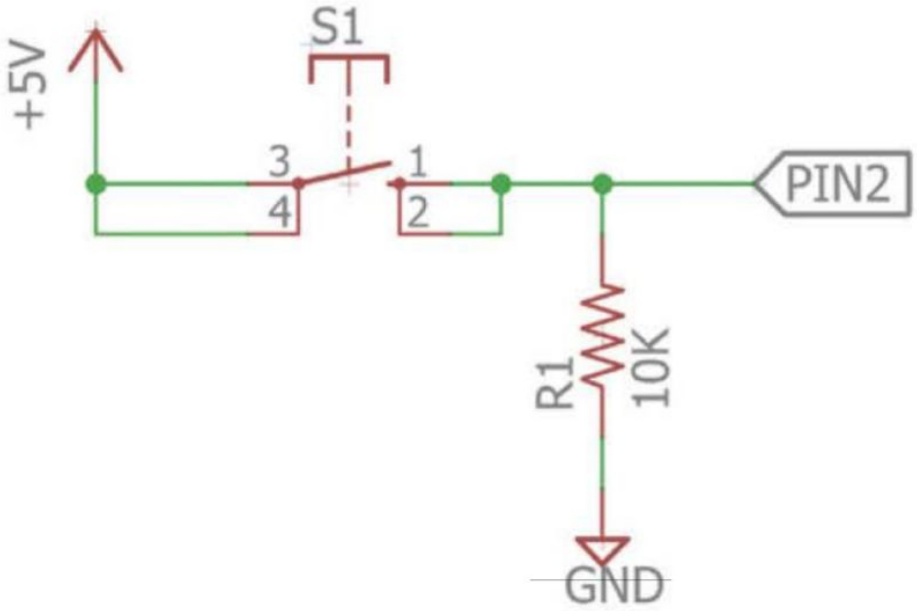
```
void setup()
{
   pinMode (PB0, OUTPUT);
   //Set the LED pin as an output
}

void loop()
{
  for (int i=0; i<256; i++)
  {
    analogWrite(PB0, i);
    delay(10);
  }
  for (int i=255; i>=0; i--)
  {

  }
 }
```

LED fade

```
void setup()
{
   pinMode (PB0, OUTPUT);
   //Set the LED pin as an output
}

void loop()
{
  for (int i=0; i<256; i++)
  {
    analogWrite(PB0, i);
    delay(10);
  }
  for (int i=255; i>=0; i--)
  {
    analogWrite(PB0, i);
    delay(10);
  }
 }
```

Reading Digital Inputs





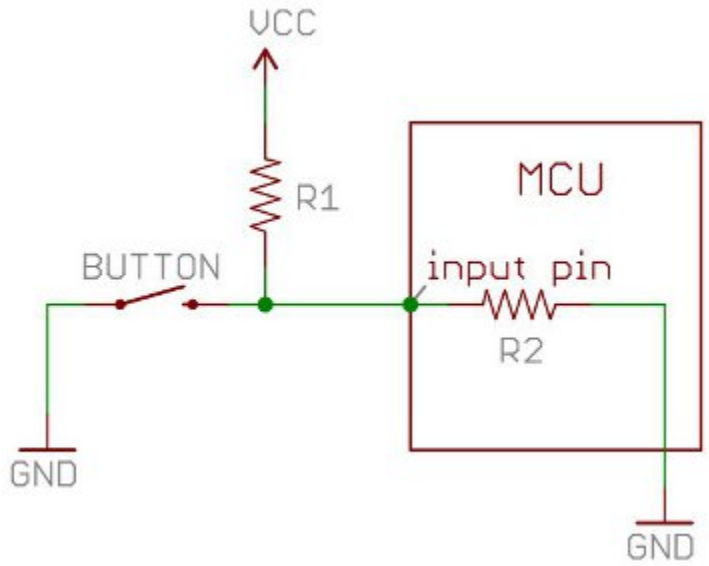
Floating

Imagine the circuit in figure without the 10kΩ resistor. What happens when the button is not being pressed?

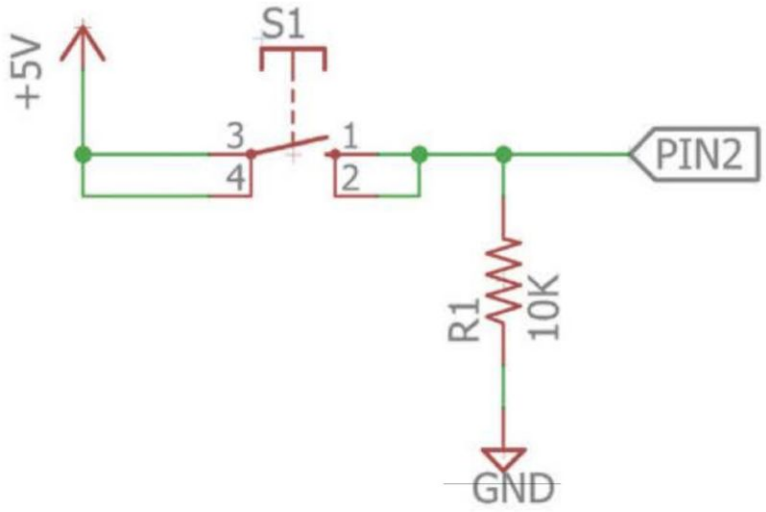
In that scenario, the input pin you are reading is essentially connected to nothing – the input pin is said to be “floating.”

And, because the pin is not physically connected to 0V or 5V, reading it could cause unexpected results as electrical noise on nearby pins causes its value to fluctuate between high and low. To remedy this, the pull-down resistor is installed as shown in the schematic

Pull Up & Pull Down Resistor



Pull up resistor [Link]



Pull down resistor

Simple LED control with a button

```
const int LED=PB0;
// The LED is connected to pin 9
const int BUTTON=PC15;
// The Button is connected to pin 2

void setup()
{
   pinMode (LED, OUTPUT);
   // Set the LED pin as an output
   pinMode (BUTTON, INPUT);
   // Set button as input (not required)
 }
```

Simple LED control with a button

```
void loop()
{
   if (digitalRead(BUTTON) == LOW)
   {
    digitalWrite(LED, LOW);
    }
    else
    {
    digitalWrite(LED, HIGH);
    }
 }
```

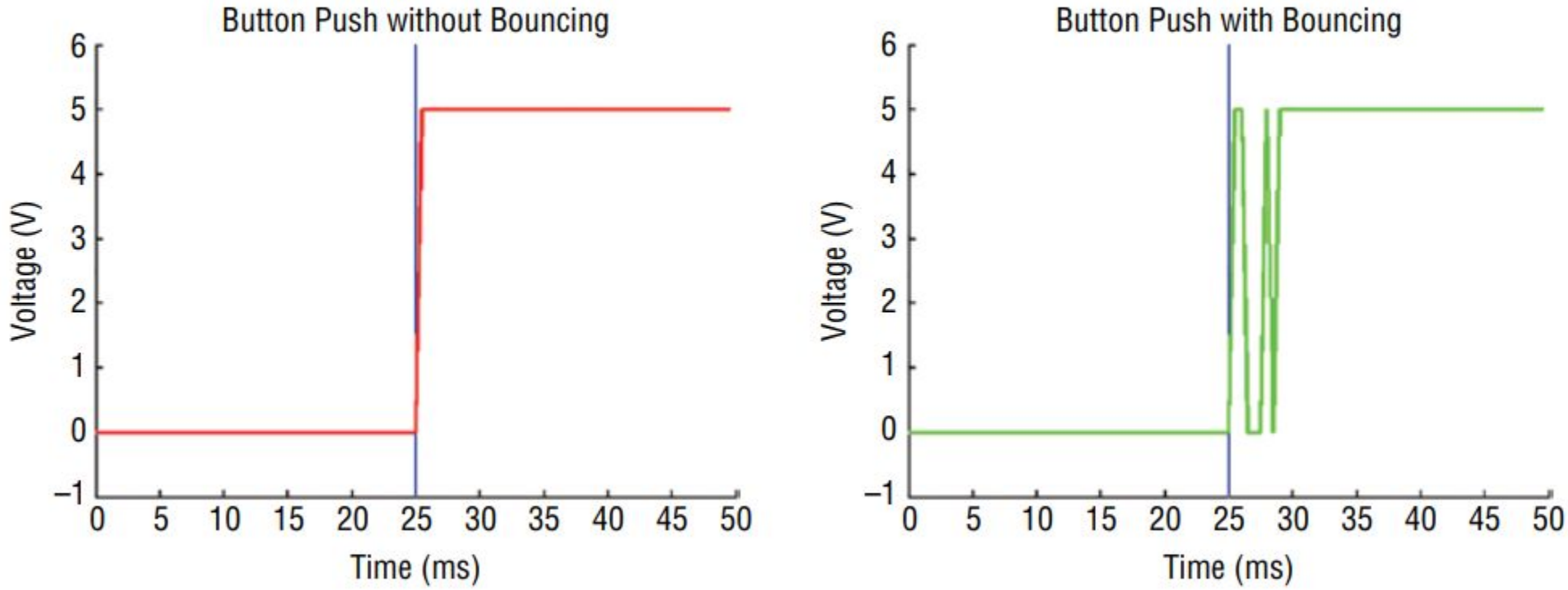
Simple LED control with a button

```
void loop()
{
   if (digitalRead(BUTTON) == LOW)
   {
    digitalWrite(LED, LOW);
    }
    else
    {
    digitalWrite(LED, HIGH);
    }
 }
```

When was the last time you had to hold a button down to keep a light on? Probably never. It makes more sense to be able to click the button once to turn it on and to click the button again to turn it off.

Unfortunately, this is not quite as easy as you might first guess. You cannot just look for the value of the switch to change from low to high; you need to worry about a phenomenon called switch bouncing.

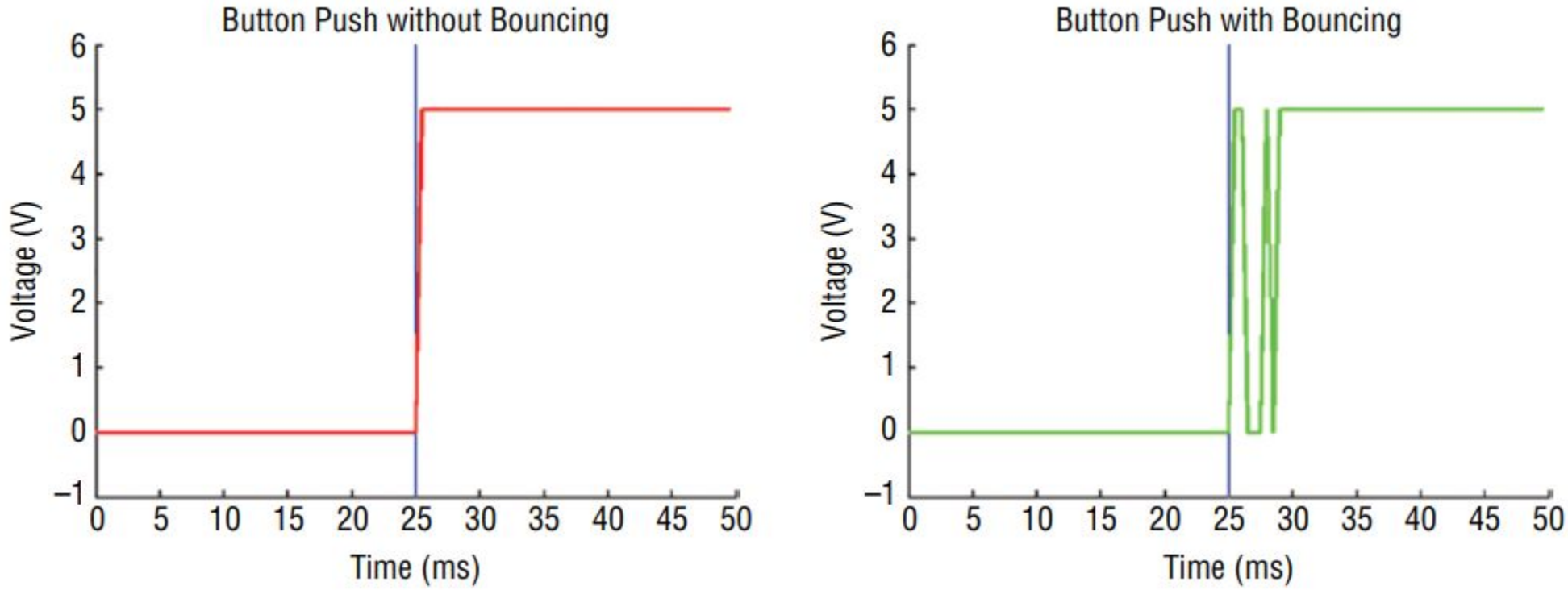
Switch Bouncing



Pressing a switch – Expectation vs Reality

Switch Bouncing

Why switch bouncing occurs?



Pressing a switch – Expectation vs Reality

Debouncing

1. Store a previous button state and a current button state (initialized to LOW).
2. Read the current button state.
3. If the current button state differs from the previous button state, wait 5 ms because the button must have changed state.
4. After 5 ms, reread the button state and use that as the current button state.
5. If the previous button state was low, and the current button state is high, toggle the LED state.
6. Set the previous button state to the current button state.
7. Return to step 2

Debounce Button Toggling

```
const int LED=PB0; // The LED is connected to pin PB0
const int BUTTON=PC15; // The Button is connected to pin PC15

boolean lastButton = LOW; // Variable containing the previous button state
boolean currentButton = LOW; // Variable containing the current button state

boolean ledOn = false; // The present state of the LED (on/off)

void setup()
{
    pinMode (LED, OUTPUT); // Set the LED pin as an output
    pinMode (BUTTON, INPUT); // Set button as input (not required)
}
```

Debounce Button Toggling

```
/*
* Debouncing Function
* Pass it the previous button state,
* and get back the current debounced button state.
*/
boolean debounce(boolean last)
{
    boolean current = digitalRead(BUTTON); // Read the button state
    if (last != current) // if it's different...
    {

    }
 }
```

Debounce Button Toggling

```
/*
* Debouncing Function
* Pass it the previous button state,
* and get back the current debounced button state.
*/
boolean debounce(boolean last)
{
    boolean current = digitalRead(BUTTON); // Read the button state
    if (last != current) // if it's different...
    {
        delay(5); //Wait 5ms
        current = digitalRead(BUTTON); //Read it again
    }
    return current; //Return the current value
 }
```

Debounce Button Toggling

```
void loop()
{
    currentButton = debounce(lastButton); //Read debounced state
}
```

Debounce Button Toggling

```
void loop()
{
    currentButton = debounce(lastButton); //Read debounced state
    if (lastButton == LOW && currentButton == HIGH) //if it was pressed...
    {

    }
}
```

Debounce Button Toggling

```
void loop()
{
    currentButton = debounce(lastButton); //Read debounced state
    if (lastButton == LOW && currentButton == HIGH) //if it was pressed...
    {
        ledOn = !ledOn; //Toggle the LED value
    }
}
```

Debounce Button Toggling

```
void loop()
{
    currentButton = debounce(lastButton); //Read debounced state
    if (lastButton == LOW && currentButton == HIGH) //if it was pressed...
    {
        ledOn = !ledOn; //Toggle the LED value
    }
    lastButton = currentButton; //Reset button value
    digitalWrite(LED, ledOn); //Change the LED state
}
```

Controllable RGB LED Nightlight

Self-study from book

Thank You!