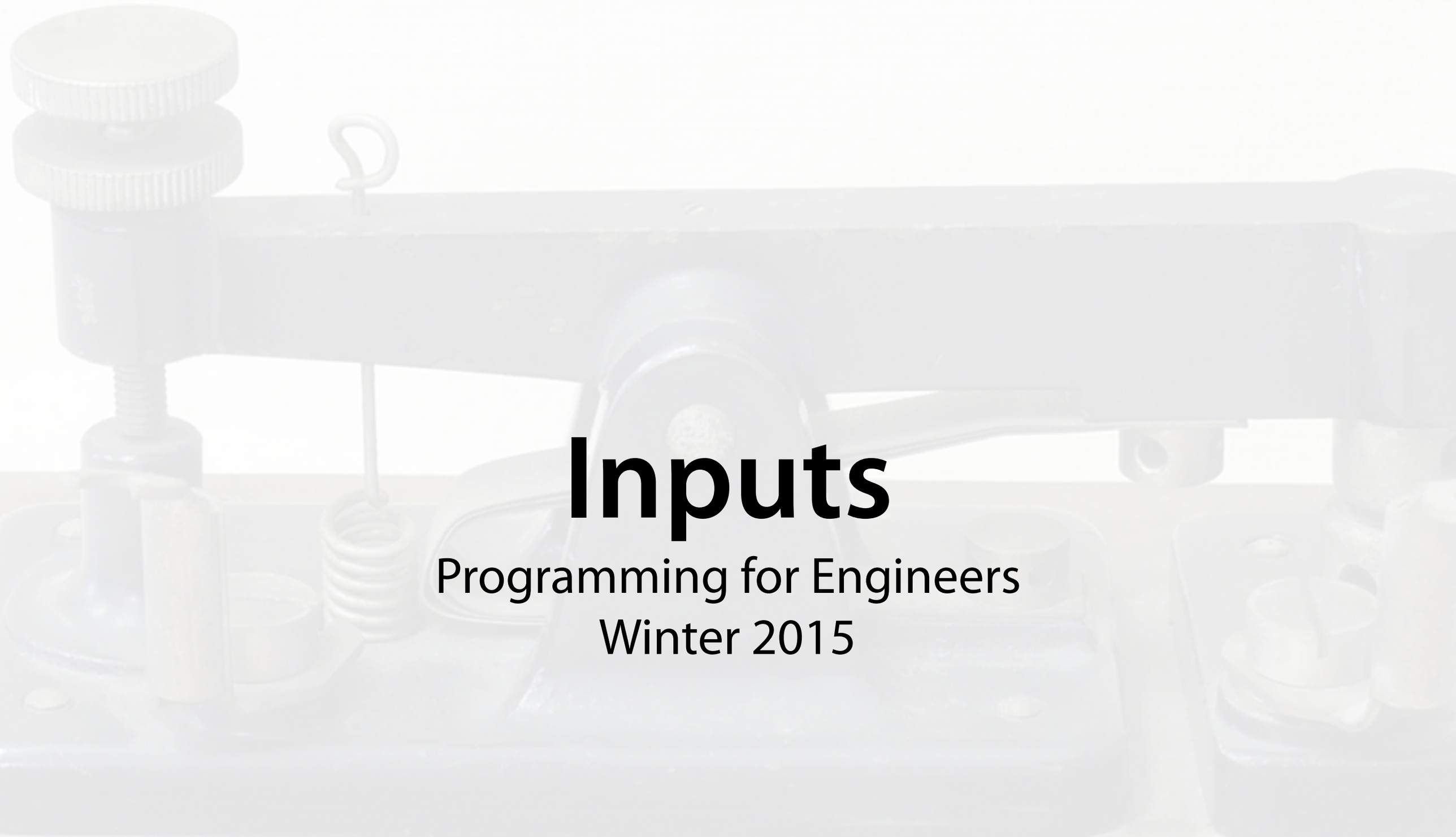


Inputs

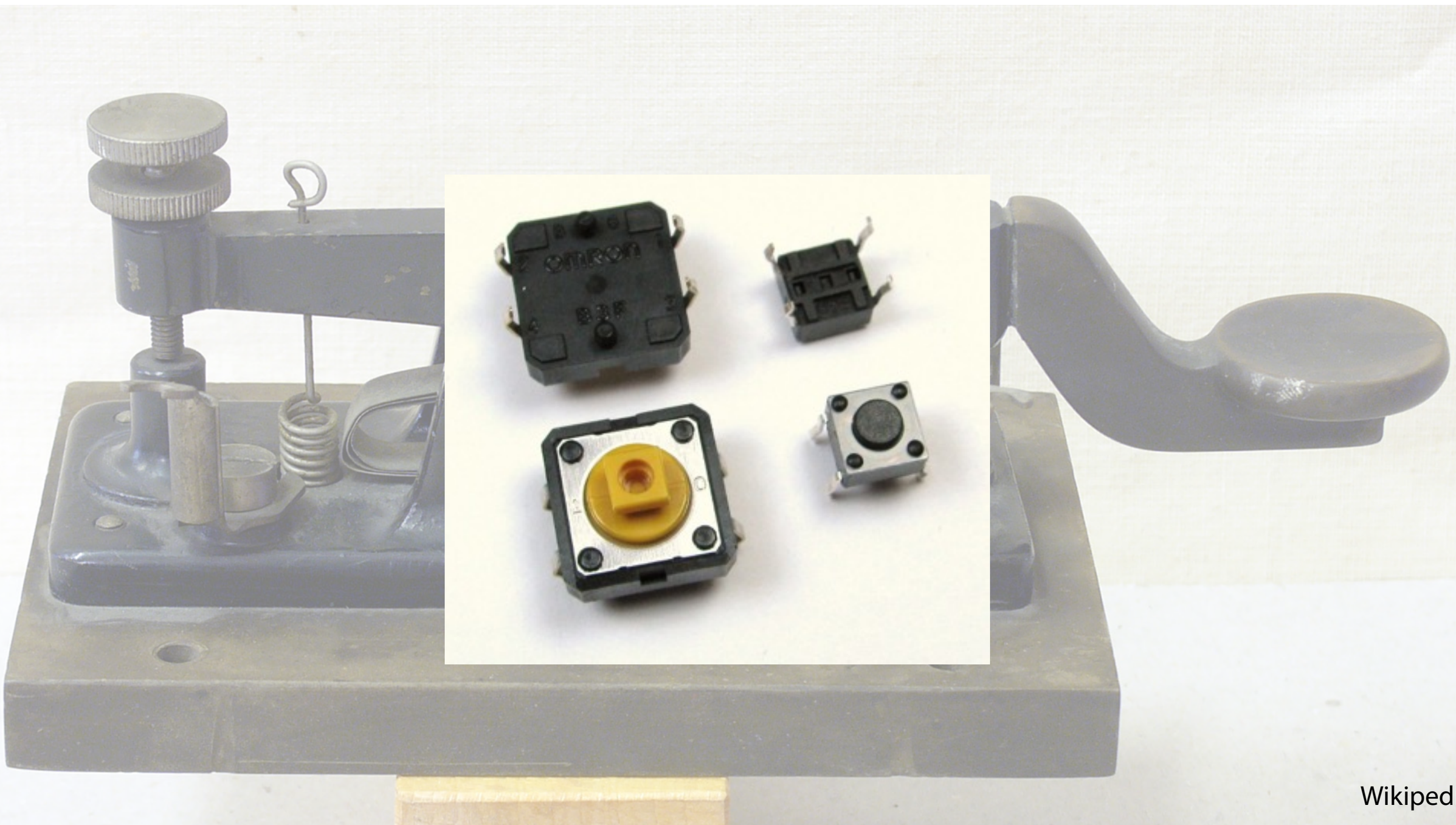
Programming for Engineers
Winter 2015

Andreas Zeller, Saarland University

Today's Topics

- Inputs
- Assignments
- Time Measurements

Button



Wikipedia

Goal

When button pressed, LED shall light up

Querying Sensors

- We already know digitalWrite(), which prints out data
- New: digitalRead() reads in data digitally

```
digitalRead(pin_number)
```

has value HIGH if there is + at the Pin; and LOW otherwise.

- A program that uses digitalRead() must check its value

Querying Sensors

- If digitalRead() = HIGH,
  the LED shall light up
- If digitalRead() = LOW,
  the LED shall turn off
- … and again and again…

Querying Sensors

```
int ledPin = 13;    // The LED
int buttonPin = 8; // The button

void setup() {
     pinMode(ledPin, OUTPUT);
     pinMode(buttonPin, INPUT);
}
```

Querying Sensors

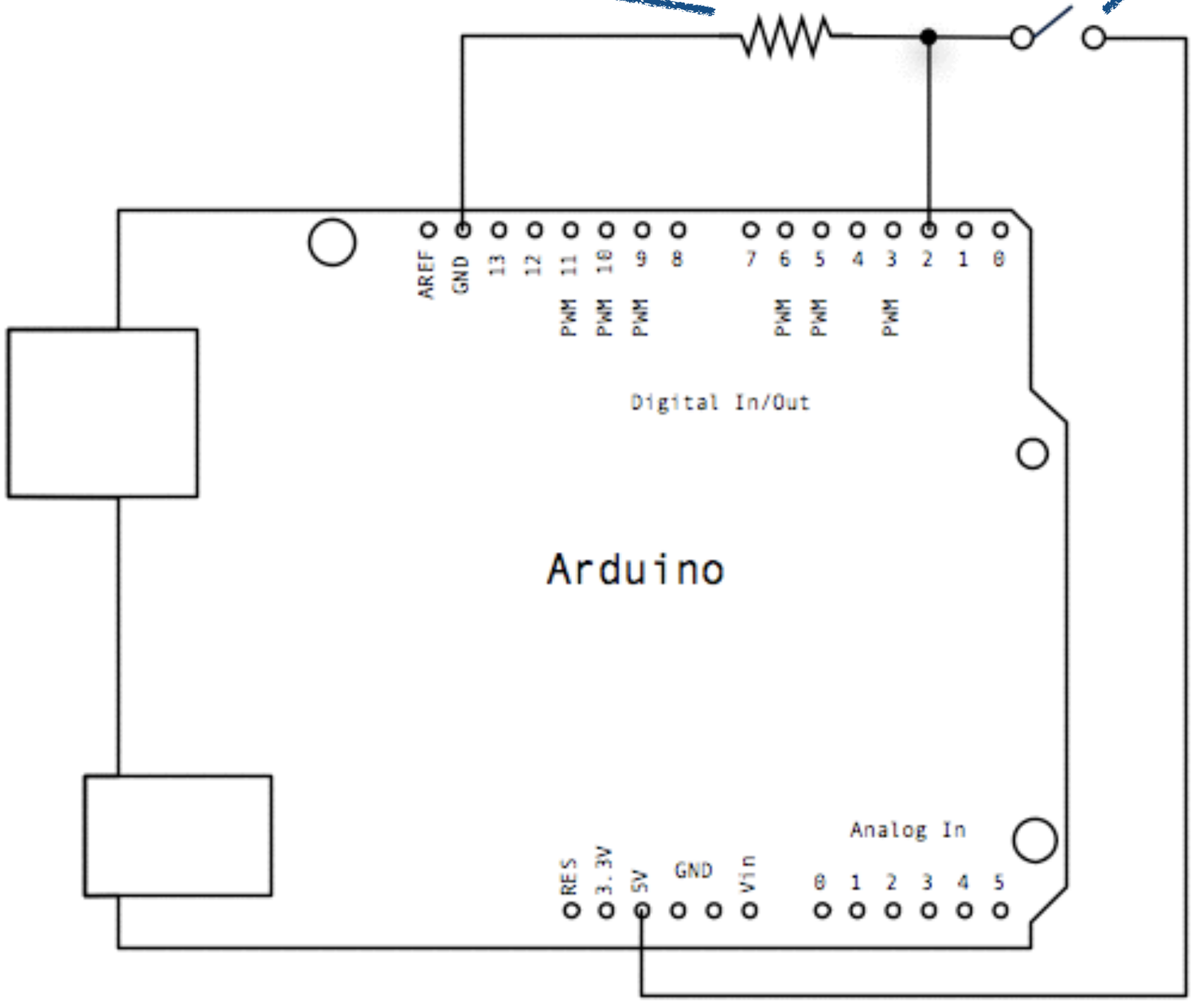
```
int ledPin = 13;    // The LED
int buttonPin = 8; // The button

void setup() {
     pinMode(ledPin, OUTPUT);
     pinMode(buttonPin, INPUT);
}

void loop() {
     if (digitalRead(buttonPin) == HIGH) {
          digitalWrite(ledPin, HIGH);
     }
     if (digitalRead(buttonPin) == LOW) {
          digitalWrite(ledPin, LOW);
     }
}
```

query the sensor

Circuit Diagram

Pulldown Resistor

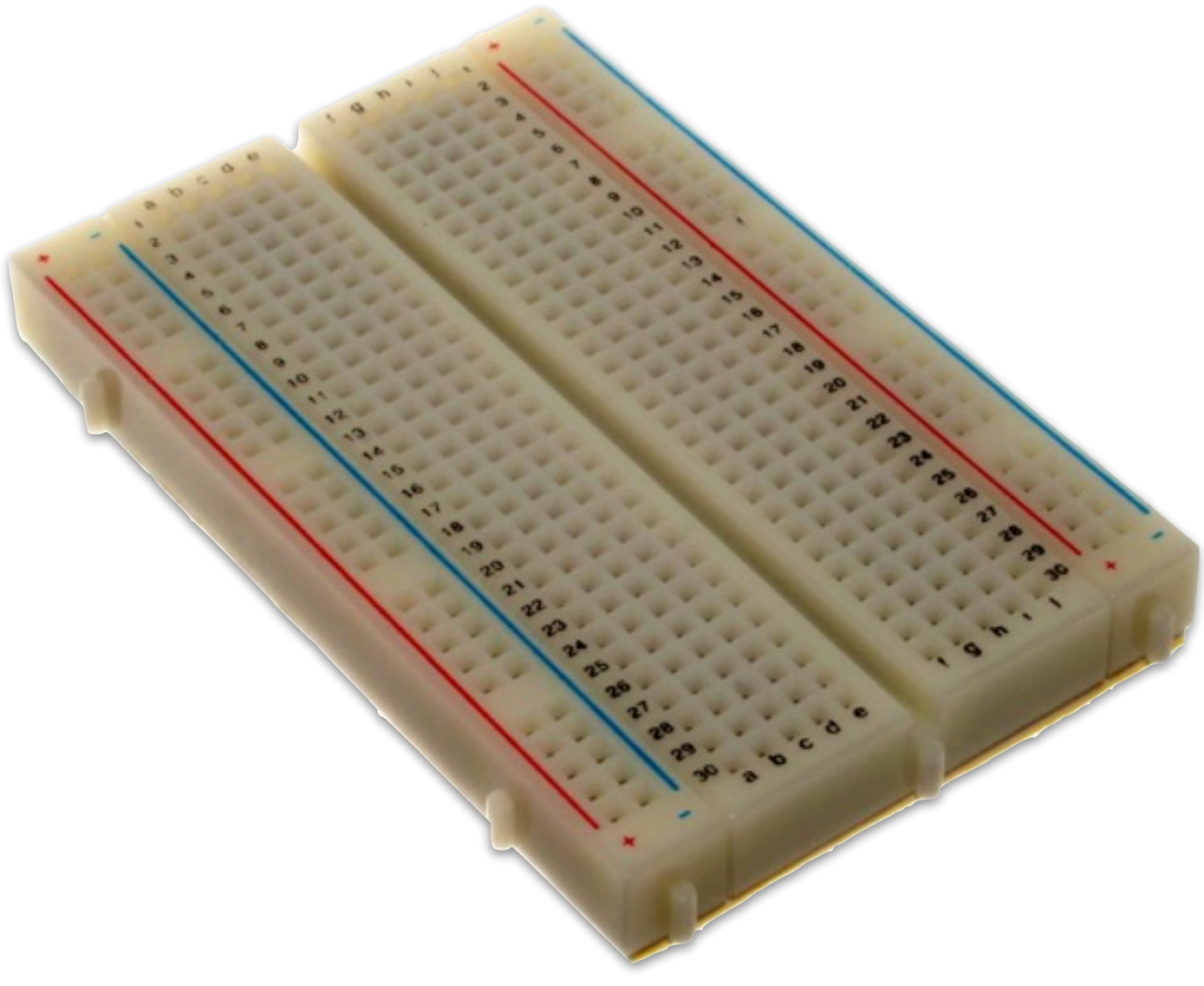
Button

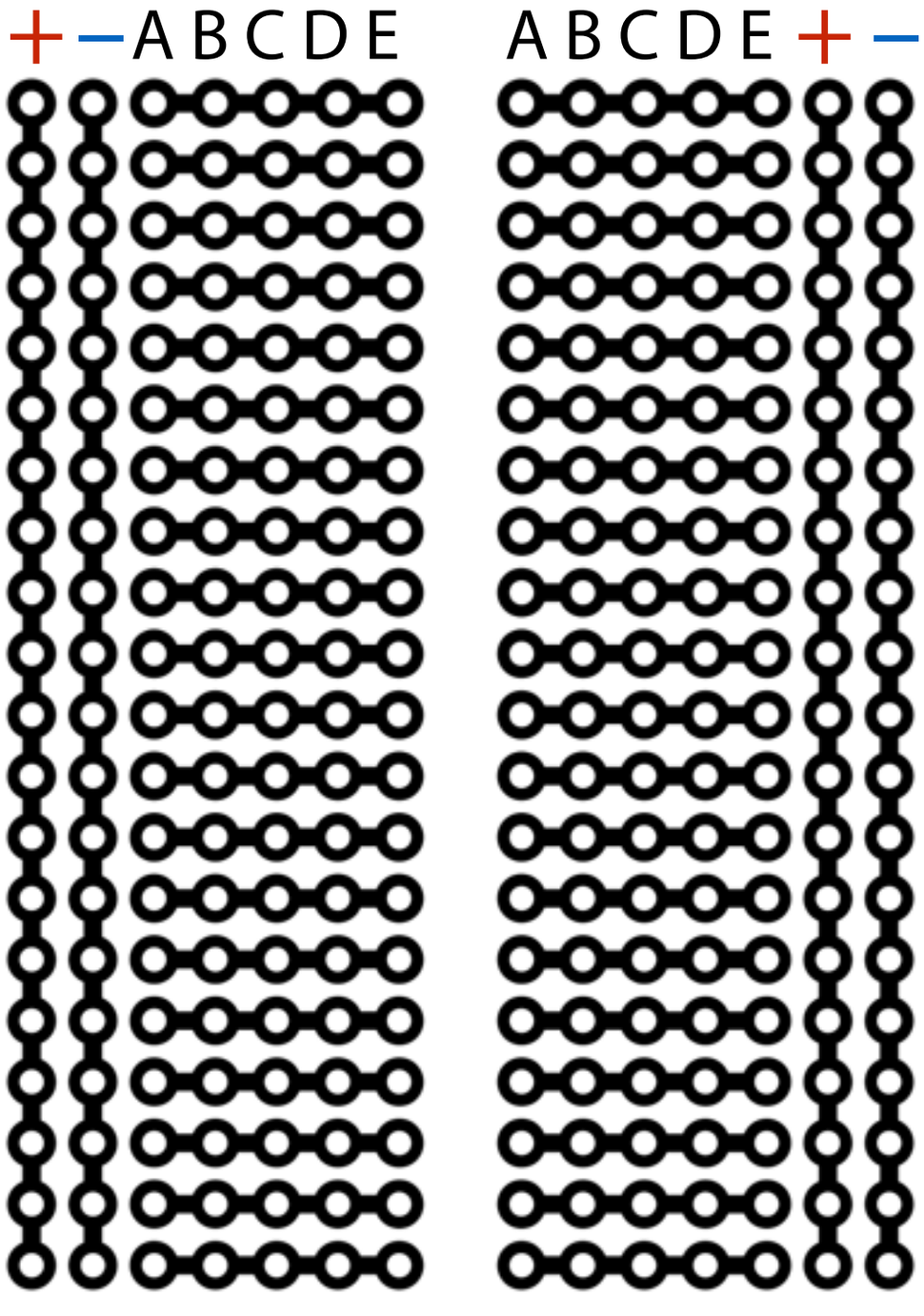


Pullup and Pulldown

- When nothing is connected to a digital input (neither + nor –), the input value is undefined
- A pullup or pulldown resistor (resp.) (Arduino: 10kΩ) defines the level if there is no signal
- Gets shorted by pressing a button

Breadboard





Querying Sensors

```
int ledPin = 13;    // The LED
int buttonPin = 8; // The button

void setup() {
     pinMode(ledPin, OUTPUT);
     pinMode(buttonPin, INPUT);
}

void loop() {
     if (digitalRead(buttonPin) == HIGH) {
          digitalWrite(ledPin, HIGH);
     }
     if (digitalRead(buttonPin) == LOW) {
          digitalWrite(ledPin, LOW);
     }
}
```

Storing Values

- In our program the sensor gets queried twice successively, even though once would suffice
- We must store the result
- We can assign the result to a variable

Assignment

- The assignment

  name = *value*

  causes the variable name to have the new value

- As the program continues, every access to the variable name produces this value (until the next assignment)

Querying Sensors

```
int ledPin = 13;      // The LED
int buttonPin = 8;    // The button


void loop() {
      if (digitalRead(buttonPin) == HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      if (digitalRead(buttonPin) == LOW) {
          digitalWrite(ledPin, LOW);
      }
}
```

query the sensor

Querying Sensors

```
int ledPin = 13;       // The LED
int buttonPin = 8;     // The button
int buttonState;      // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState = HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      if (buttonState = LOW) {
          digitalWrite(ledPin, LOW);
      }
}
```

Querying Sensors

```
int ledPin = 13;       // The LED
int buttonPin = 8;     // The button
int buttonState;      // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState = HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      if (buttonState = LOW) {
          digitalWrite(ledPin, LOW);
      }
}
```

What is wrong here?

Querying Sensors

```
int ledPin = 13;       // The LED
int buttonPin = 8;     // The button
int buttonState;      // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState == HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      if (buttonState == LOW) {
          digitalWrite(ledPin, LOW);
      }
}
```

==, not = !

If and Braces

- The {…} can be omitted when there is only one statement after the if-condition:

```
if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
}
```

is the same as

```
if (buttonState == HIGH)
    digitalWrite(ledPin, HIGH);
```

If and Braces

- Omitting {…} can lead to subtle mistakes:

```
if (buttonState == HIGH)
     digitalWrite(ledPin, HIGH);
     Serial.println("HIGH");

Serial.println(buttonState);
```

If and Braces

- Omitting {…} can lead to subtle mistakes:

```
if (buttonState == HIGH)
     digitalWrite(ledPin, HIGH);
     Serial.println("HIGH");

Serial.println(buttonState);
```

What is wrong here?

If and Braces

- Omitting {…} can lead to subtle mistakes:

```
if (buttonState == HIGH) {
     digitalWrite(ledPin, HIGH);
     Serial.println("HIGH");
}
Serial.println(buttonState);
```

- Note: Indentation is for humans, braces are for the computer.

If … Else

- By means of if … else we can define instructions that are executed, when the if-condition does not hold

```
if (condition) {
    Instructions...
}
if (!condition) {
    Instructions...
}
```

⟺

```
if (condition) {
    Instructions...
}
else {
    Instructions...
}
```

Else If

- if … else can be chained:

```
if (condition) {
    Instructions...
}
else if (condition) {
    Instructions...
}
else {
    Instructions...
}
```

Querying Sensors

```
int ledPin = 13;      // The LED
int buttonPin = 8;    // The button
int buttonState;     // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState == HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      if (buttonState == LOW) {
          digitalWrite(ledPin, LOW);
      }
}
```

Querying Sensors

```
int ledPin = 13;       // The LED
int buttonPin = 8;     // The button
int buttonState;      // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState == HIGH) {
          digitalWrite(ledPin, HIGH);
      }
      else {
         digitalWrite(ledPin, LOW);
      }
}
```

Querying Sensors

```
int ledPin = 13;      // The LED
int buttonPin = 8;    // The button
int buttonState;     // The button state


void loop() {
      buttonState = digitalRead(buttonPin);
      if (buttonState == HIGH)
          digitalWrite(ledPin, HIGH);
      else
          digitalWrite(ledPin, LOW);
}
```

Querying Sensors

```
int ledPin = 13;      // The LED
int buttonPin = 8;    // The button



void loop() {
    digitalWrite(ledPin, digitalRead(buttonPin));
}
```

Does the same; however, is not as readable

Goal

When button pressed, LED turns on/off

Approach

- We introduce a variable ledStatus,
  that represents the LED state and can be
  toggled by the button

Toggling State

```
int ledPin = 13;                // Pin LED
int buttonPin = 8;              // Pin button
int ledStatus = HIGH;           // LED state

void setup() { ... }

void loop() {
    if (digitalRead(buttonPin) == HIGH) {
        if (ledStatus == HIGH)
            ledStatus = LOW;
        else
            ledStatus = HIGH;

        digitalWrite(ledPin, ledStatus);
        delay(200);
    }
}
```

Toggling State

```
int ledPin = 13;                // Pin LED
int buttonPin = 8;              // Pin button
int ledStatus = HIGH;           // LED state

void setup() { ... }

void loop() {
    if (digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;  // short form
        digitalWrite(ledPin, ledStatus);
        delay(200);
    }
}
```

Negation

- Boolean values in C:
  zero (false) and non-zero (true)
- ! is the negation (¬):
  - !0 is 1
  - !1 is 0
- HIGH and LOW have values 1 and 0 resp.

Problem

When button pressed, LED blinks

Approach

- The variable pushed is set while the button is pressed
- The variable ledStatus is only changed when the button changes its state

```
int ledPin = 13;          // Pin LED
int buttonPin = 8;        // Pin button
int ledStatus = HIGH;     // LED state
int pushed = 0;           // button state

void setup() { ... }

void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;

        digitalWrite(ledPin, ledStatus);
        delay(200);
    }
    if (pushed && digitalRead(buttonPin) == LOW)
        pushed = 0;

}
```

Logical Operators

- && is a logical AND (∧)

&&	0	1
0	0	0
1	0	1

- || is a logical OR (∨)

||	0	1
0	0	1
1	1	1

Goal

Start/stop blinking on button press

Blinking on Demand

```
void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW)
        pushed = 0;

    if (ledStatus) {
        digitalWrite(ledPin, HIGH);
        delay(200);
        digitalWrite(ledPin, LOW);
        delay(200);
    }
}
```

Problem

Button presses are ignored

Blinking on Demand

```
void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW)
        pushed = 0;

    if (ledStatus) {
        digitalWrite(ledPin, HIGH);
        delay(200);
        digitalWrite(ledPin, LOW);
        delay(200);
    }
}
```

Bouncing

When a button is pressed, there can be bouncing – a short, repeated closing and opening

Blinking on Demand

```
void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;
        delay(50);  // wait for bouncing to stop
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
        pushed = 0;
        delay(50);  // wait for bouncing to stop
    }
    if (ledStatus) {
        digitalWrite(ledPin, HIGH);
        delay(200);
        digitalWrite(ledPin, LOW);
        delay(200);
    }
}
```

Problem

Button presses are ignored occasionally

Goal

Query the button continously

Measuring Time

- During a delay() all inputs are ignored
- The function millis() returns the number of milliseconds since program start
- We can use millis() to measure time

Blinking with Millis

```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin button

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop() {
    int ms = millis() % 1000;
    if (ms < 500)
        digitalWrite(ledPin, LOW);
    if (ms > 500)
        digitalWrite(ledPin, HIGH);
}
```

Button with Millis

```
void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;
        delay(50);
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
        pushed = 0;
        delay(50);
    }
    if (ledStatus) {
        ms = millis() % 1000;
        if (ms < 500)
            digitalWrite(ledPin, LOW);
        if (ms > 500)
            digitalWrite(ledPin, HIGH);
    }
}
```

Button with Millis

```
void loop() {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
        ledStatus = !ledStatus;
        pushed = 1;
        delay(50);
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
        pushed = 0;
        delay(50);
    }
    if (ledStatus) {
      // blinking...
    }
}
```

Still a delay

Debouncing with Millis

```
int previousPush = 0;

void loop() {
  if (millis() - previousPush >= 50) {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
      previousPush = millis();
      ledStatus = !ledStatus;
      pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
      pushed = 0;
      previousPush = millis();
    }
  }
  // blinking...
}
```

Datatypes in C

- long – at least 32 bits, [$-2^{31}\ldots2^{31}-1$]
- int – 16 to (usually) 32 bits, [$-2^{31}\ldots2^{31}-1$]
- short – at least 16 bits, [$-2^{15}\ldots2^{15}-1$]
- char – at least 8 bits, usually [$-2^{7}\ldots2^{7}-1$]
- Also as "unsigned", then [$0\ldots2^{bits}-1$]

(long long >) long > int > short > char

Datatypes

- millis() has the type "unsigned long" – integer values in $[0 \ldots 2^{32}-1]$
- Usual integer numbers ("int") are in $[-2^{n-1} \ldots 2^{n-1}-1]$
- n is (depending on the device) 16 or 32
- 2^{15} Milliseconds = 32767 ms = 32,7 s
  2^{32} Milliseconds = 1193 h = 49,7 days

Overflow

- When we try to store a value that is too large for a datatype, there is an overflow
- Only the last (binary) digits are stored
- This can lead to arbitrary values

Boeing 787: US-Luftfahrtbehörde warnt vor Stromausfall im Dreamliner

REUTERS

Boeing 787: Programmierter Stromausfall nach 248 Tagen

Die US-Luftsicherheitsbehörde warnt: In Boeings Dreamliner kann im Flug der Strom ausfallen, das Flugzeug unkontrollierbar werden. Schuld ist ein Computerfehler.

Samstag, 02.05.2015 – 11:02 Uhr

Drucken | Senden | Merken

Nutzungsrechte | Feedback

Kommentieren | 121 Kommentare

THEMA
Boeing 787 Dreamliner

Flugsicherheit

Die US-Luftfahrtbehörde FAA (Federal Aviation Administration) hat eine Warnung für den Boeing-Langstreckenjet 787 Dreamliner herausgegeben. Demnach kann ein Softwareproblem dazu führen, dass in dem Flugzeug der Strom ausfällt, sodass die Piloten die Kontrolle über den Flieger verlieren würden.

Am Freitag hat die Behörde eine sogenannte Flugtauglichkeitsdirektive veröffentlicht, in der es

Boeing 787: US-Luftfahrtbehörde warnt vor Stromausfall im Dreamliner

- Power generator failure after 248 days of continuous operation
- 248 days = 2^{31} /100 seconds → int-overflow on time measuring
- If all four generators affected: catastrophe
- **Workaround: restarting every 247 days at the latest**

Boeing 787: Programmierter Stromausfall nach 248 Tagen

Datatypes in C

- long – at least 32 bits, [$-2^{31}\ldots2^{31}-1$]
- int – 16 to (usually) 32 bits, [$-2^{31}\ldots2^{31}-1$]
- short – at least 16 bits, [$-2^{15}\ldots2^{15}-1$]
- char – at least 8 bits, usually [$-2^{7}\ldots2^{7}-1$]
- Also as "unsigned", then [$0\ldots2^{bits}-1$]

(long long >) long > int > short > char

Debouncing with Millis

```
int previousPush = 0;

void loop() {
  if (millis() - previousPush >= 50) {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
      previousPush = millis();
      ledStatus = !ledStatus;
      pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
      pushed = 0;
      previousPush = millis();
    }
  }
 // blinking...
}
```

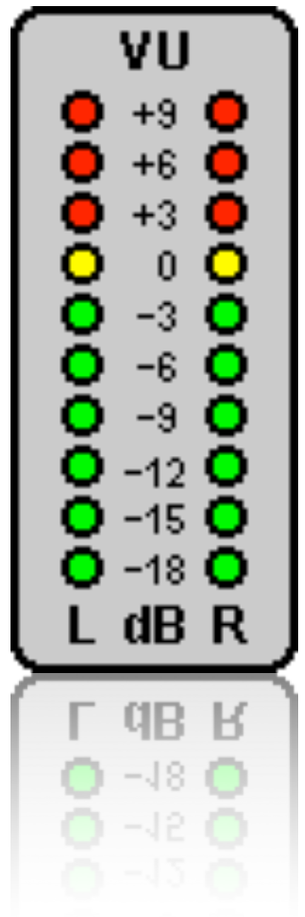
Debouncing with Millis

Now it fits

```
unsigned long previousPush = 0;

void loop() {
  if (millis() - previousPush >= 50) {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
      previousPush = millis();
      ledStatus = !ledStatus;
      pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
      pushed = 0;
      previousPush = millis();
    }
  }
 // blinking...
}
```

Outlook

- Arrays
- Loops
- Level Measuring



Querying Sensors

```
int ledPin = 13;      // The LED
int buttonPin = 8;    // The button


void loop() {
    buttonState = digitalRead(buttonPin);
    if (buttonState == HIGH)
        digitalWrite(ledPin, HIGH);
    else
        digitalWrite(ledPin, LOW);
}
```

Assignment

- The assignment

  name = value

  causes the variable name to have the new value
- As the program continues, every access to the variable name produces this value (until the next assignment)

Blinking with Millis

```
int ledPin = 13;      // Pin LED
int buttonPin = 8;    // Pin button

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop() {
    int ms = millis() % 1000;
    if (ms < 500)
        digitalWrite(ledPin, LOW);
    if (ms > 500)
        digitalWrite(ledPin, HIGH);
}
```

Debouncing with Millis

```
unsigned long previousPush = 0;

void loop() {
  if (millis() - previousPush >= 50) {
    if (!pushed && digitalRead(buttonPin) == HIGH) {
      previousPush = millis();
      ledStatus = !ledStatus;
      pushed = 1;
    }
    else if (pushed && digitalRead(buttonPin) == LOW){
      pushed = 0;
      previousPush = millis();
    }
  }
 // blinking...
}
```