Microcontrollers

Mechatronics Basics

Tuomas Tiainen

21.11.2024

Tuomas Tiainen

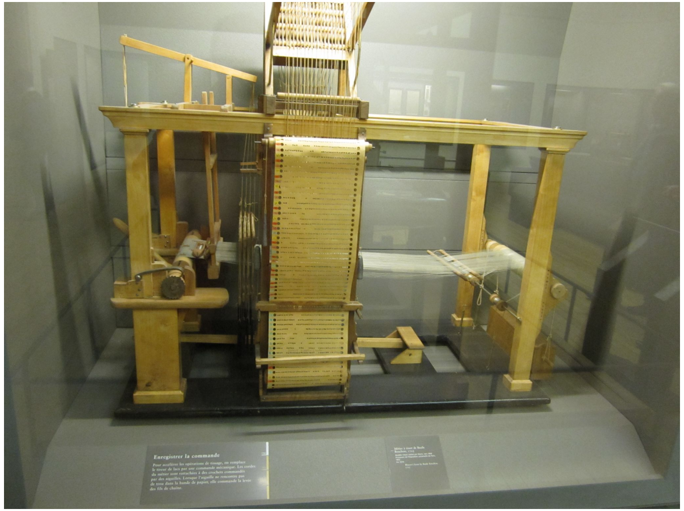
- CEO, Tapio Measurement Technologies Oy
- Aalto University: M.Sc. (2018), D. Sc. (2020)
- Studies in machine design and computer science
- Doctoral degree and postdoc period at Aalto 2018–2023
- Worked as a software developer before starting work on dissertation
- tuomas.tiainen@iki.fi

Learning goals

- What is a microcontroller?
- Hardware and communication
- How to program microcontrollers
- Demystify programmable logic

History of programmable logic

Jacquard machine (1804)
(Image credit: Wikimedia commons)



Bouchon loom (1725)
(Image credit: Wikimedia commons)

What is programmable logic?

Read inputs and write to outputs based on a logic programmed into memory

Reasons to use programmable logic:

- Automation
- React to changes in system by programming
- Reconfigurability
- Meet real-time requirements

Types of programmable logic

Microcontroller unit (MCU)

Microprocessor unit (MPU)

Programmable logic controller (PLC)

(Discrete logic)

Field programmable gate array (FPGA)

System on Chip (SoC)

Application specific integrated circuit (ASIC)

Single board computer (SBC)

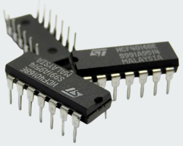
Microcontrollers: Building Blocks of Embedded Systems

Microcontrollers (MCU or μC) are:

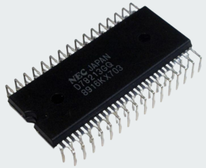
- Prevalent in embedded systems (specialized purpose-built computer system combining hardware and software)
- Highly available and versatile
- Diverse in terms of cost, performance and features
- Easy to use and develop for
- Great way to learn basics of electronics, programmable logic and embedded design

What does an MCU look like?

Through-hole technology (THT)

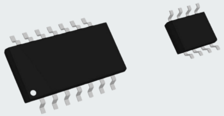


Dual-inline package (DIP)

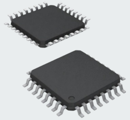


Quad-inline package (QIP)

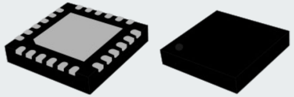
Surface mount technology (SMT) /
Surface mount device (SMD)



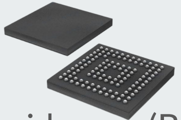
Small outline package (SOP)



Quad-flat package (QFP)



Quad-flat no-leads package (QFN)



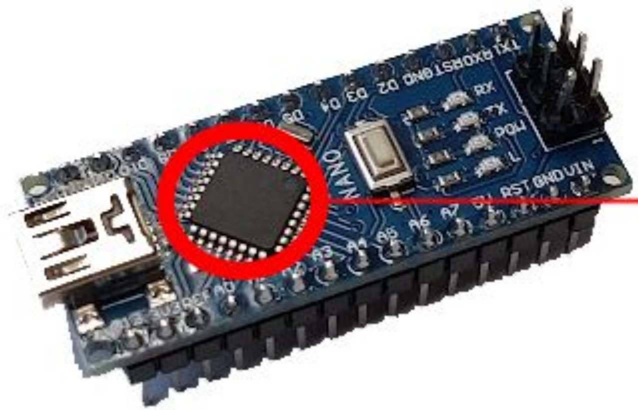
Ball grid array (BGA)

Development board, MCU and CPU

Arduino Nano

ATmega328

AVR CPU

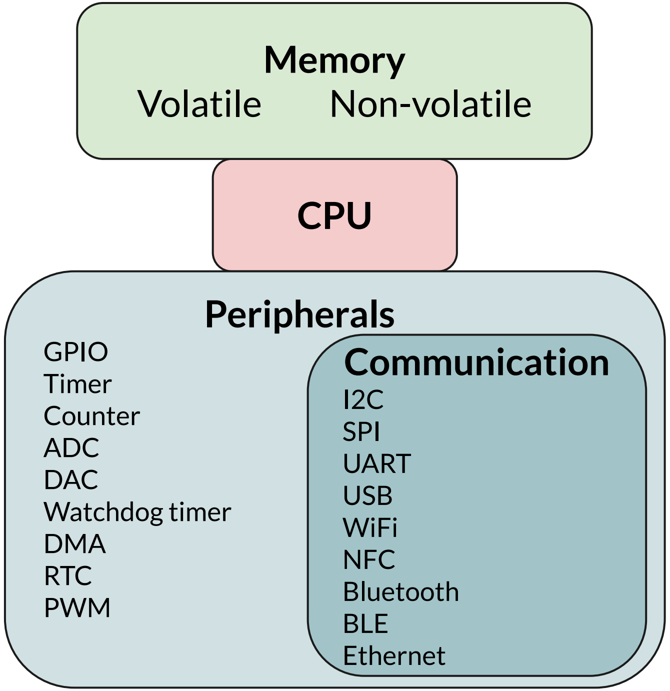


Development board

MCU
Microcontroller Unit

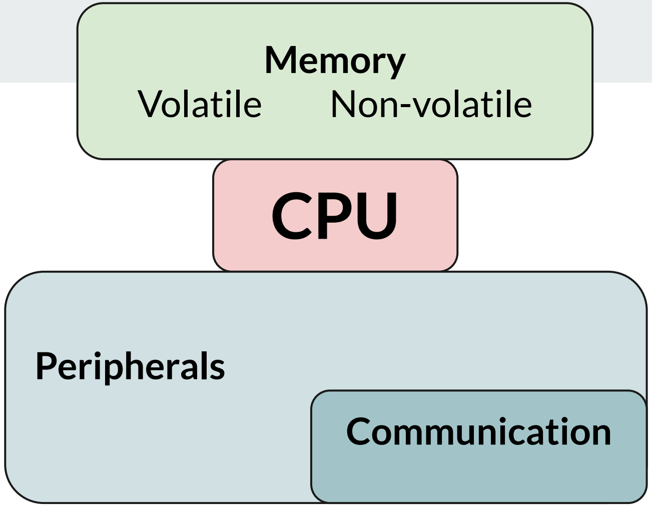
CPU
Central Processing Unit

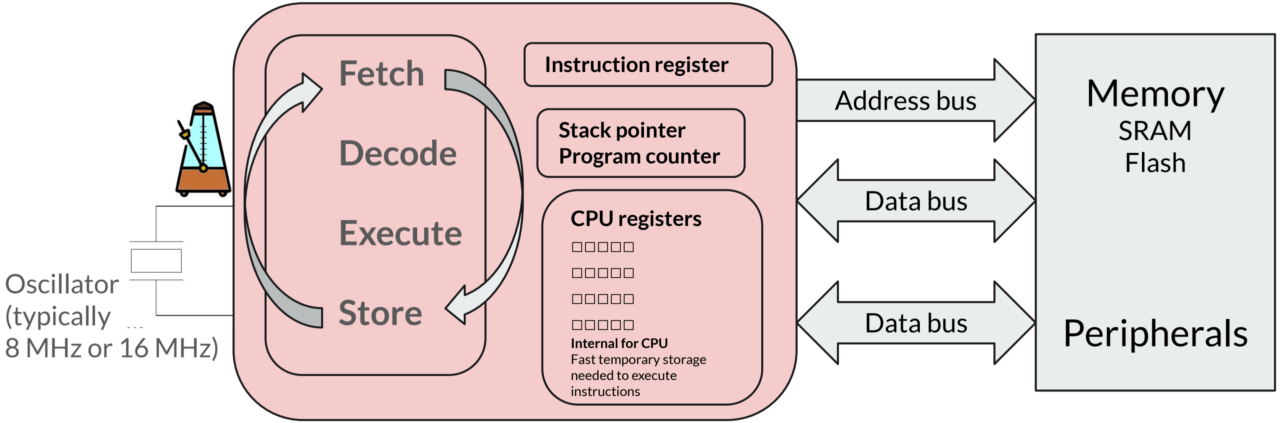
Inside a microcontroller unit



Note: these features are examples. Different microcontrollers have different hardware features.

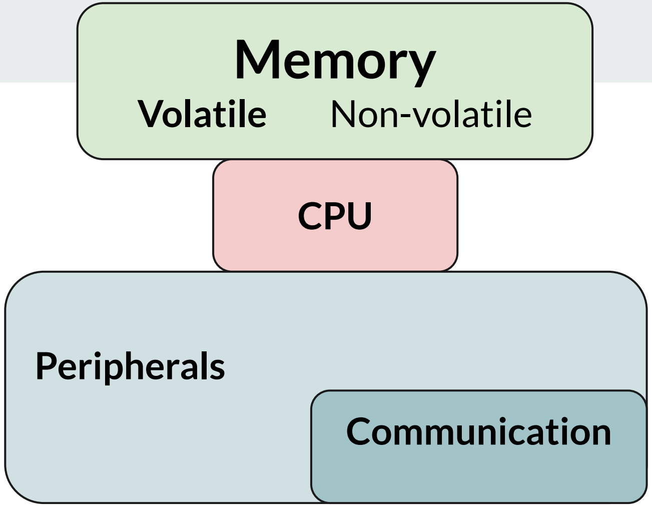
Inside a microcontroller: CPU





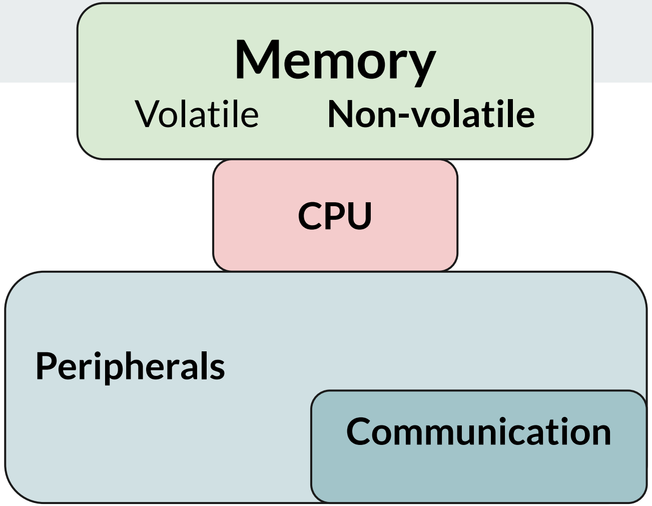
Inside a microcontroller: Volatile memory

- Lost between power cycles and resets
- Example sizes:
  - Arduino Nano: 2 kB SRAM
  - Teensy 4.1: 1 MB SRAM
- Static random access memory (SRAM)
  - Small but fast
  - For run time variables
- Additional PSRAM chips available



8 MB PSRAM chip with SPI interface
Image credit: PJRC

Inside a microcontroller: Non-volatile memory



Kept between power cycles and resets

Flash

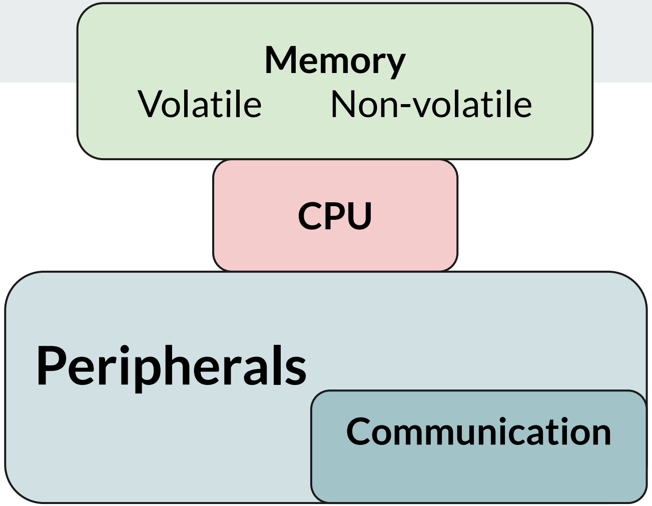
- Mid-size, fairly fast, permanent
- Block erasable
- Program is stored here
- Example sizes:
  - Arduino Nano: 32 kB
  - Teensy 4.1 2 MB

EEPROM

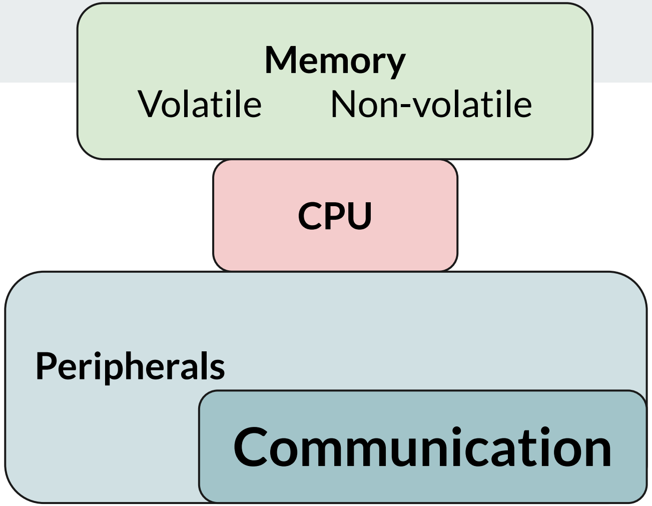
Electrically Erasable Programmable Read-Only Memory

- Flash memory that allows writing individual bytes
- Slow
- Limitations on write cycles

Inside a microcontroller
Peripherals



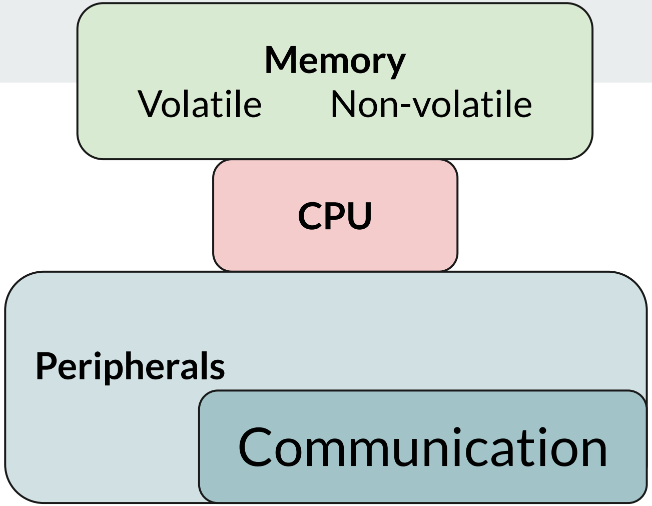
- General purpose inputs and outputs (GPIO)
  - E.g. for buttons and LED's
- Timers, counters, pulse width modulation
  - Timing tasks at set intervals
  - Counting time between events
- ADC and DAC (analog-to-digital and digital-to-analog converters)
  - Read an analog signal (arbitrary voltage between 0 and a reference voltage)
  - Generating an analog signal (arbitrary voltage between 0 and a reference voltage)



Inside a microcontroller GPIO

- Toggle pins high (1) or low (0)
- Read if pin is high (1) or low (0)
- Avoid floating: use pull-ups or pull-downs (many MCU's have integrated)
- Arduino: *digitalRead* and *digitalWrite*

- **Attention:** there are different logic levels
  - Usually 5 V or 3.3 V
  - Use logic-level converters or voltage dividers
  - **Boards are easily damaged by overvoltage**
- **Attention:** GPIO pins cannot be used to directly drive larger loads



Inside a microcontroller
Example: NXP iMX RT1062

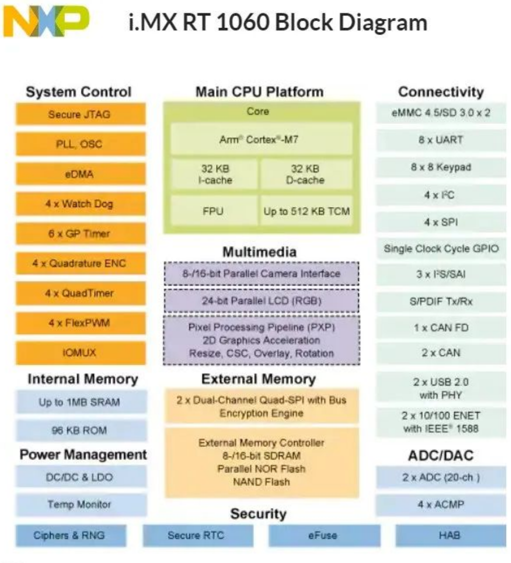


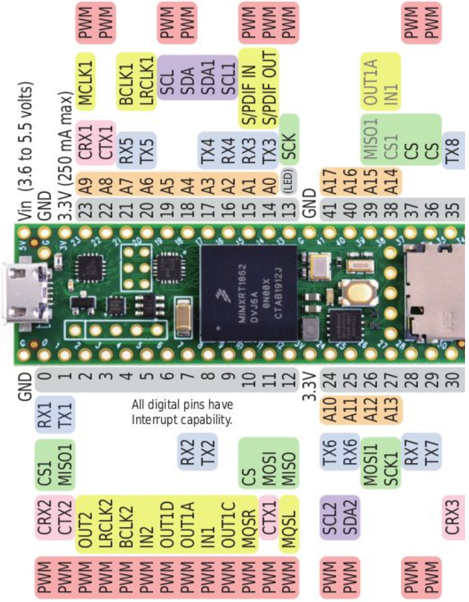
Image credit: NXP

Demo: Blink

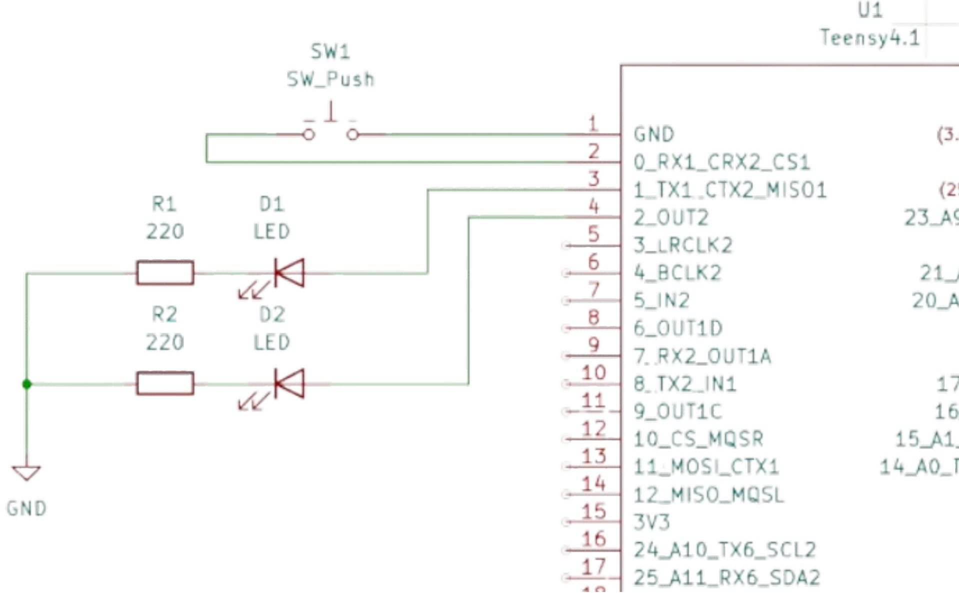
```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop () {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

Teensy 4.1 development board
Image credit: PJRC

Demo: Two-led blink

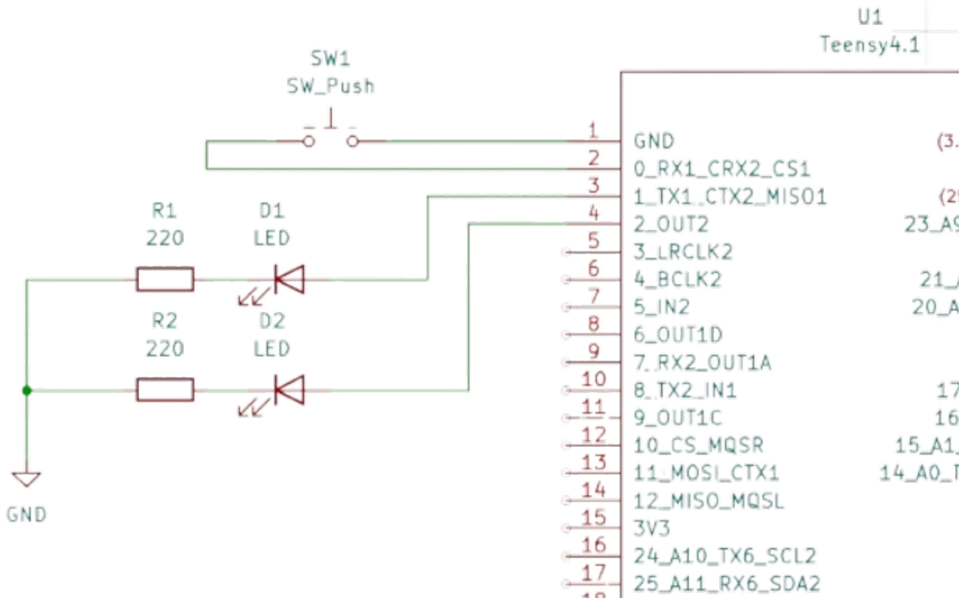


Teensy 4.1 pinout



Demo: Two-led blink

```
void setup() {
  // initialize digital pins 1 and 2 as outputs
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(1, HIGH);  // turn the LED on Pin 1 ON
  digitalWrite(2, LOW);   // turn the LED on Pin 2 OFF
  delay(1000);            // wait for a second
  digitalWrite(1, LOW);   // turn the LED on Pin 1 OFF
  digitalWrite(2, HIGH);  // turn the LED on Pin 2 ON
  delay(1000);            // wait for a second
}
```



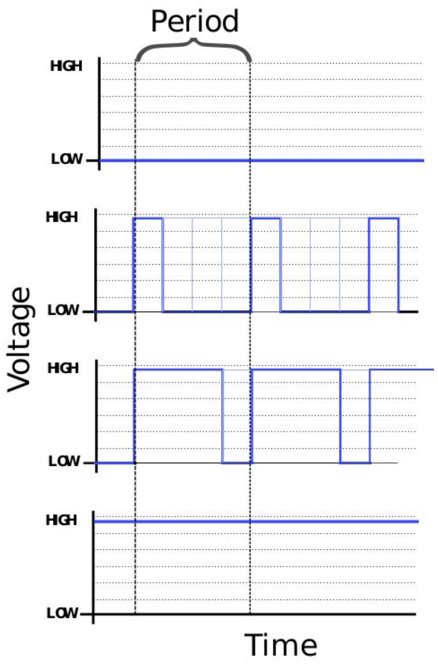
Pulse width modulation (PWM)

Instead of adjusting amplitude, adjust the ration between on and off state (i.e. pulse width)

PWM can be done at different frequencies (period lengths in time), the pulse width is quantified by duty cycle (%)

Arduino: *analogWrite*

Example: 8-bit PWM



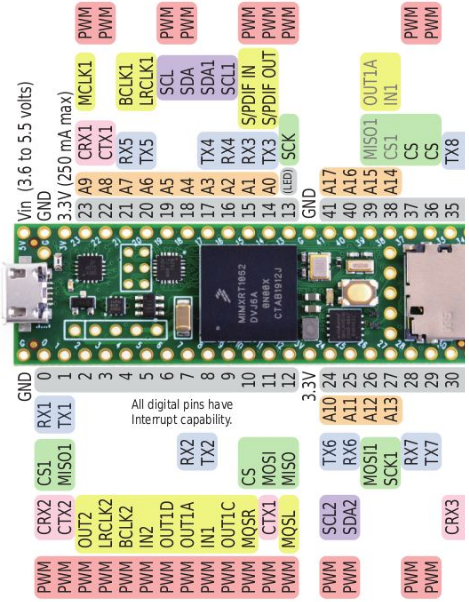
Value 0
Duty cycle 0 %

Value 64
Duty cycle 25 %

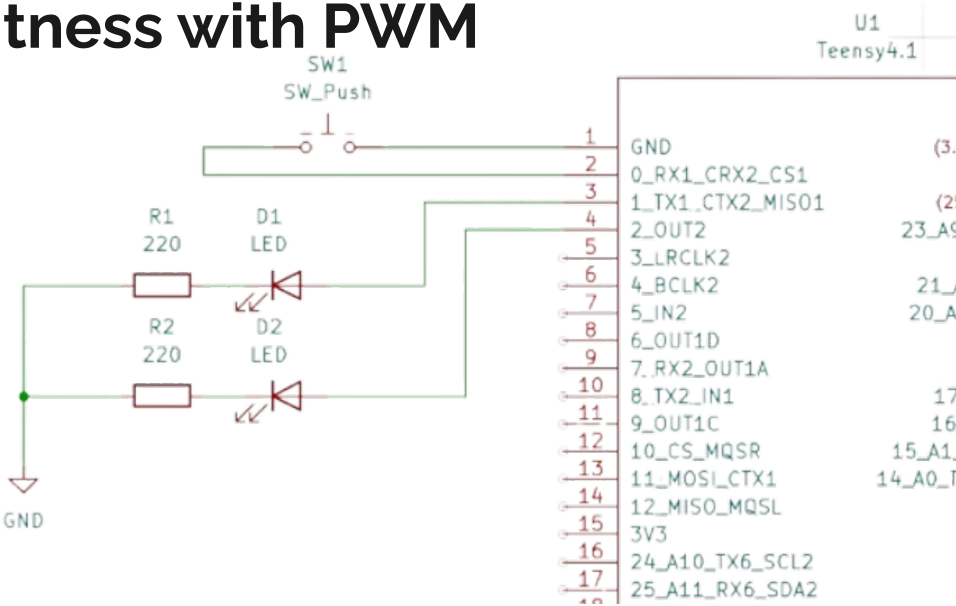
Value 192
Duty cycle 75 %

Value 256
Duty cycle 100 %

Demo: Adjusting LED brightness with PWM

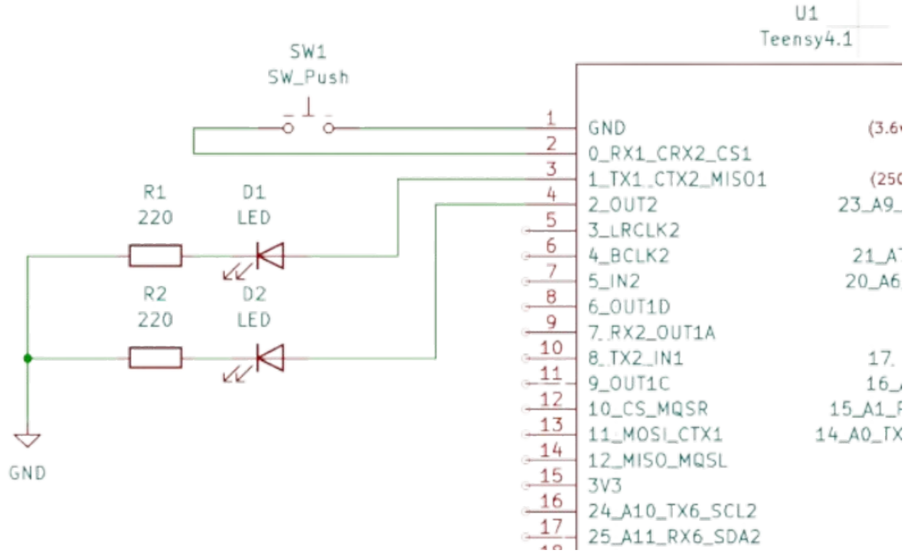


Teensy 4.1 pinout



Demo: Adjusting LED brightness with PWM

```
// analogWrite is used to adjust PWM duty cycle

void setup() {
  // Initialize digital pins 1 and 2 as outputs
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  // analogWriteFrequency(1, 20);
  // analogWriteFrequency(2, 20);
}

void loop() {
  // Ramp up the brightness of both LEDs over 2 seconds
  for (int brightness = 0; brightness <= 150; brightness++) {
    analogWrite(1, brightness); // Set brightness on pin 1
    analogWrite(2, brightness); // Set brightness on pin 2
    delay(8);
  }
  // Ramp down the brightness of both LEDs over 2 seconds
  for (int brightness = 150; brightness >= 0; brightness--) {
    analogWrite(1, brightness); // Set brightness on pin 1
    analogWrite(2, brightness); // Set brightness on pin 2
    delay(8);
  }
}
```



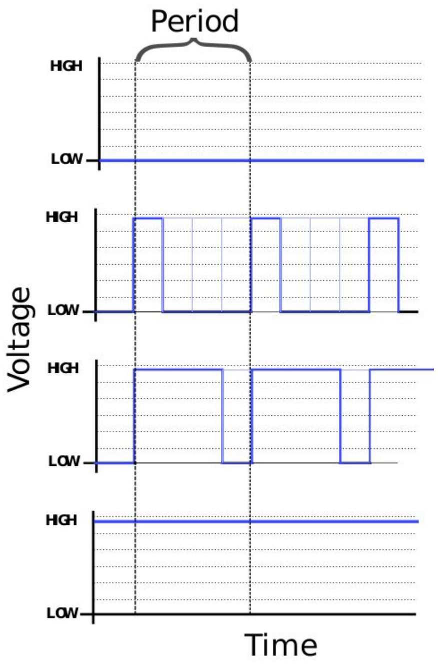
Discussion in pairs: PWM frequency

The PWM frequency in Hz (inverse of the period length) can be adjusted with the command

analogWriteFrequency(1, 50000);

When and why is the PWM frequency important?

Example: 8-bit PWM



Value 0
Duty cycle 0 %

Value 64
Duty cycle 25 %

Value 192
Duty cycle 75 %

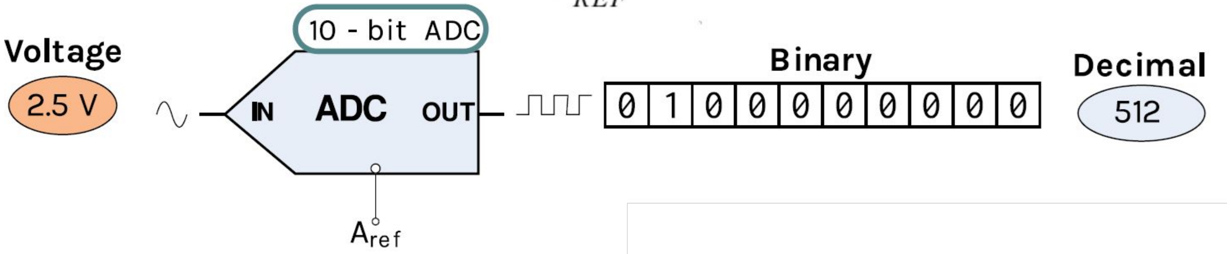
Value 256
Duty cycle 100 %

ADC

Analog-to-digital converter (ADC)

- Turn voltage into binary values
- Different types with different operating principles
- Usually several multiplexed analog inputs

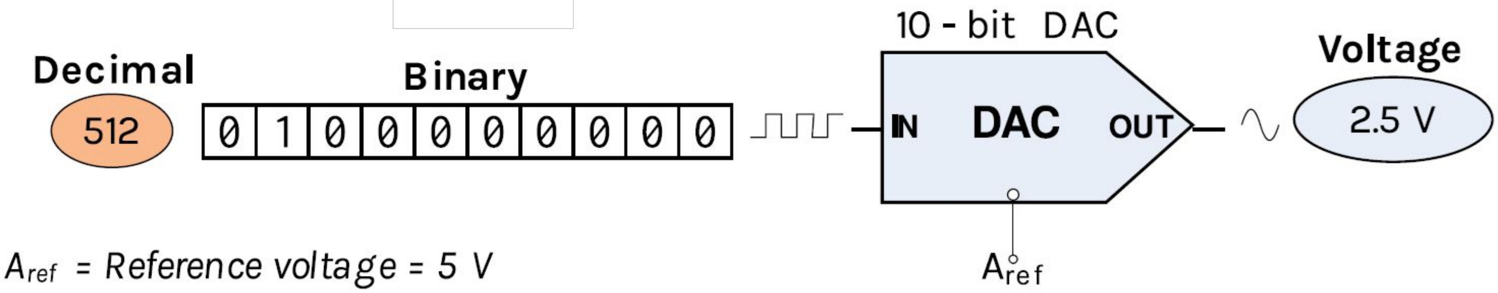
$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$



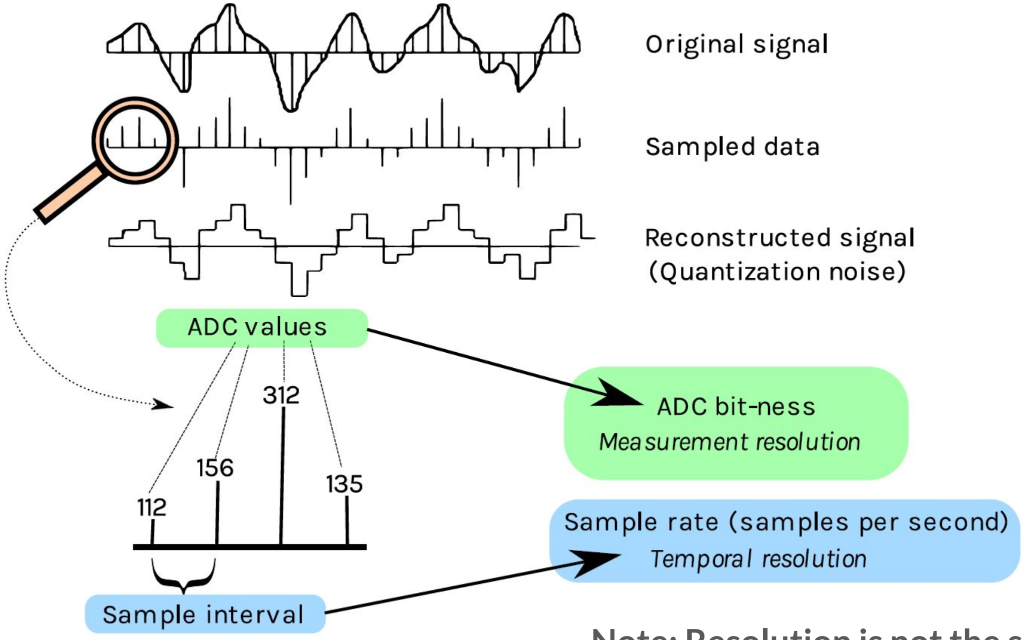
DAC

Digital-to-analog converter (DAC)

- Turn binary values into voltage
- Not available in lower tier MCU's
- Useful e.g. in audio purposes



ADC resolution



Note: Resolution is not the same as accuracy!

Exercise: ADC accuracy

- A signal is measured on 10 bit ADC on atmega328P MCU with a reference voltage of 5.0 V
- The ADC reports a value of 649
- The atmega328p datasheet reports absolute accuracy ± 2 LSB
- Assume that there is no noise on the analog reference or elsewhere.

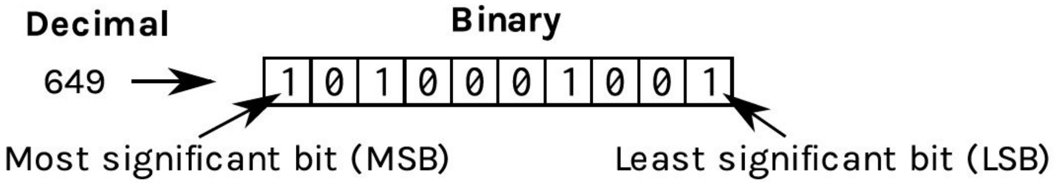
What is the measured voltage and the error of the measurement?

Solution: ADC accuracy

This is simplified.
In reality there are many factors that contribute to the error and uncertainty of the measurement. There are many additional intrinsic and extrinsic noise sources to consider.

10-bit ADC reports a value of 649, accuracy ± 2 LSB, reference voltage 5 V
What is the measured voltage and the error of the measurement?

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}} \longrightarrow V_{in} = \frac{ADC}{1024} \cdot V_{ref} \longrightarrow \frac{649}{1024} \cdot 5.0\ \mathrm{V} = 3.1689\ \mathrm{V}$$



With a 10-bit ADC we have 1 LSB (count) of 5.0 V / 1024 = 4.88 mV

The signal voltage is 3.1689 ± 0.0098 V

Warning: aliasing

Signals which contain high frequencies **must be low-pass filtered prior to sampling** or aliasing can occur.

Aliasing can occur if sampling frequency is too low for the signal (Nyquist limit Fs/2 is the highest frequency that can be resolved).

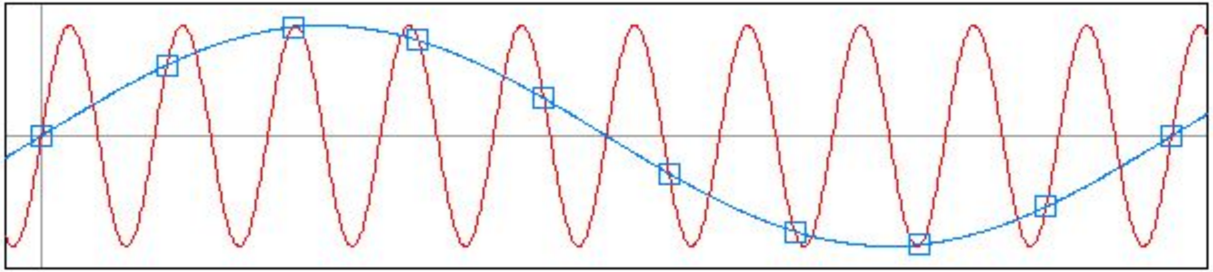


Image credit: Wikimedia commons

Exercise: ADC Resolution

A ±100 g accelerometer sensor outputs voltage between $-V_{ref}$ and V_{ref} linearly depending on the acceleration of the sensor.

A how many bit ADC should we choose if we want to measure 1 g changes in acceleration?

Analog Devices ADXL1001 accelerometer.
(Image credit: Analog Devices)

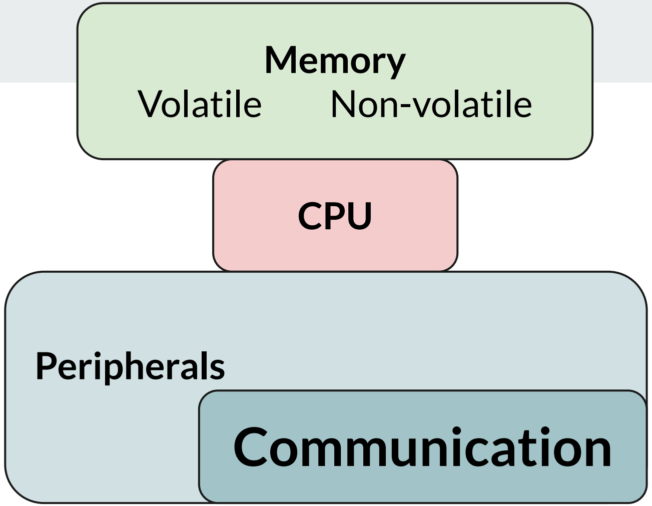
Exercise: ADC Resolution

A ±100 g accelerometer sensor outputs voltage between $-V_{ref}$ and V_{ref} linearly depending on the acceleration of the sensor.

A how many bit ADC should we choose if we want to measure 1 g changes in acceleration?

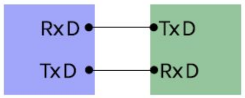
Answer: An 8-bit ADC is needed.

ADC Bits	Discrete Steps	Resolution (g)
1	$2^1 = 2$	$\frac{200}{2} \approx 100$
2	$2^2 = 4$	$\frac{200}{4} \approx 50$
3	$2^3 = 8$	$\frac{200}{8} \approx 25$
4	$2^4 = 16$	$\frac{200}{16} \approx 12.5$
5	$2^5 = 32$	$\frac{200}{32} \approx 6.25$
6	$2^6 = 64$	$\frac{200}{64} \approx 3.13$
7	$2^7 = 128$	$\frac{200}{128} \approx 1.56$
8	$2^8 = 256$	$\frac{200}{256} \approx 0.78$
9	$2^9 = 512$	$\frac{200}{512} \approx 0.39$
10	$2^{10} = 1024$	$\frac{200}{1024} \approx 0.20$



Inside a microcontroller Communication

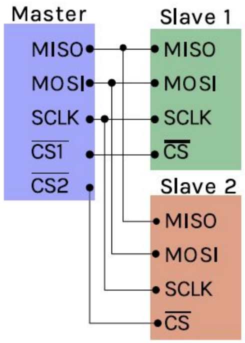
UART
Universal asynchronous receive transmit



`Serial`

Example device: Barcode scanner

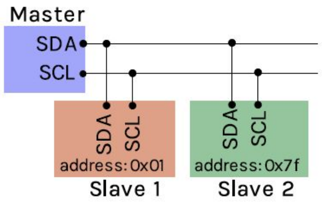
SPI
Serial peripheral interface



`SPI`

Example device: Display

I2C
Inter-Integrated Circuit

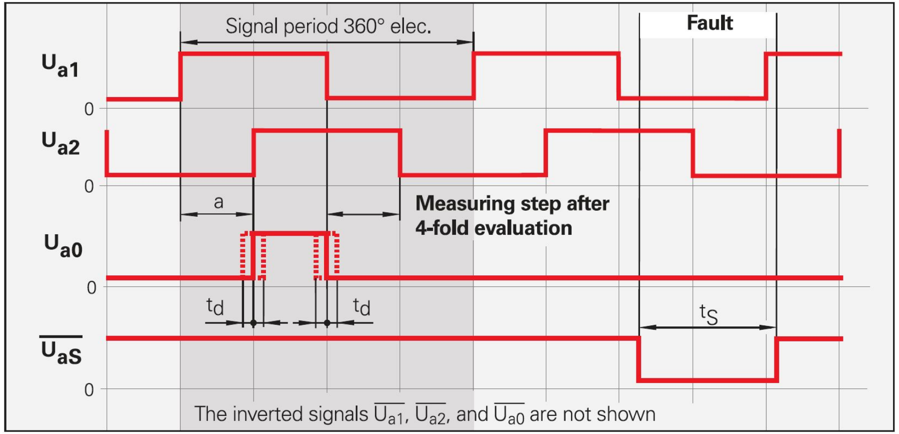


`Wire`

Example device: Battery charger

TTL quadrature signaling

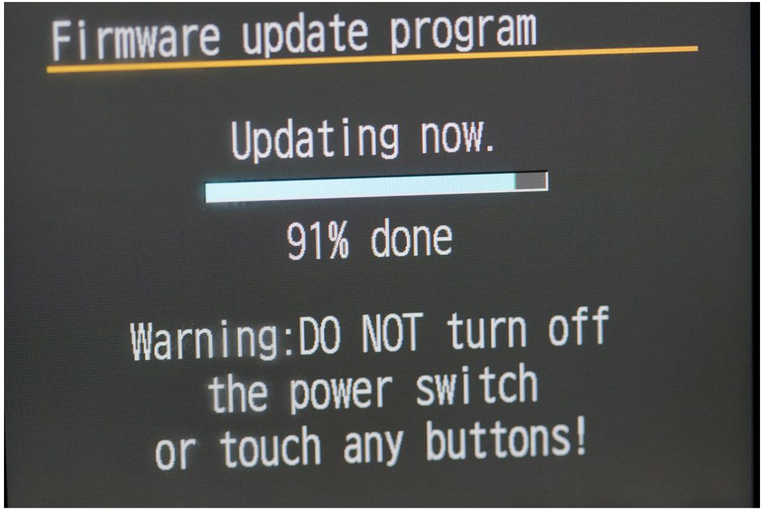
- Can be used in linear and rotary encoders
- Two square wave signals with 90 degree phase offset
- Incremental, can be used to determine the direction the encoder has moved
- Polling or interrupt based libraries
- Hardware quadrature support in some MCU's (much faster)



TTL quadrature signaling diagram
(Image credit: Heidnhain)

Software development

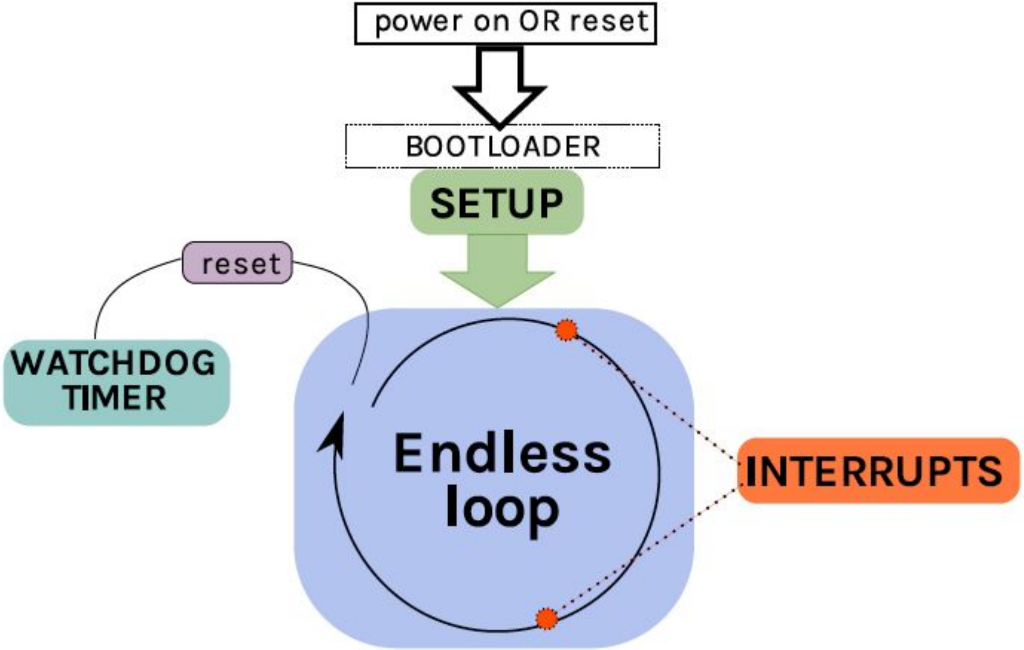
- **Firmware** is “software for hardware”
- Software embedded in a hardware device
- Hardware abstraction layers
- C is a high level language in the embedded world
- **Assembler** converts assembly language into machine code (can be done internally in compiler)



Flashing the firmware

- Programs are written (or “burned”/“flashed”) into non-volatile memory
- Programs stay in memory between power cycles
- Programming devices (in circuit programmer) or over USB (bootloader)

Program structure



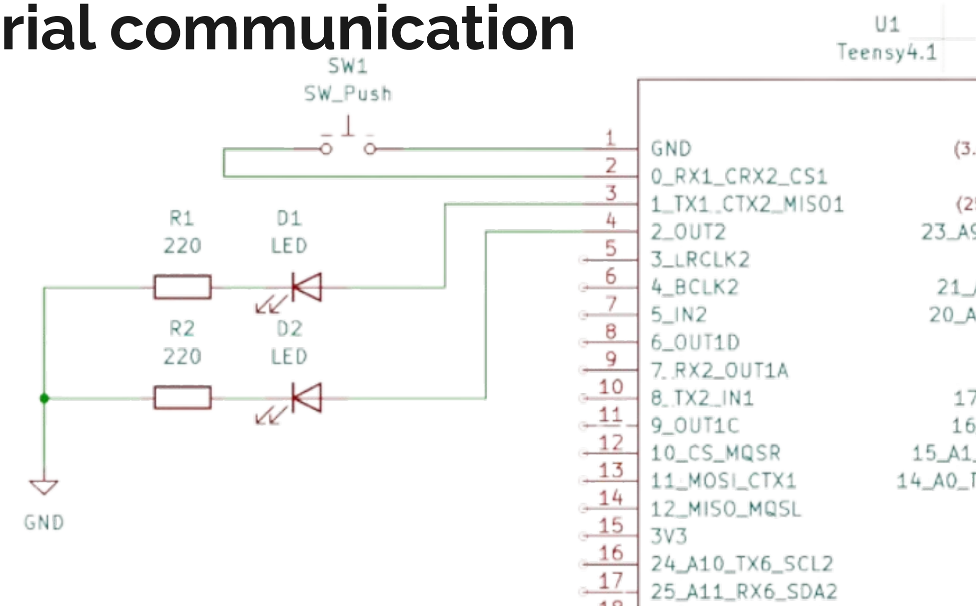
Interrupts

- Interrupts the main sequence of instructions
- Function run in interrupt is called an interrupt service routine (ISR)
- Internal interrupts (timers etc.)
- External interrupts, rising/falling/change on pin (not on all pins in all MCU:s)

Try to minimize time spent in ISR. The ISR can block other interrupts and important events in the program.

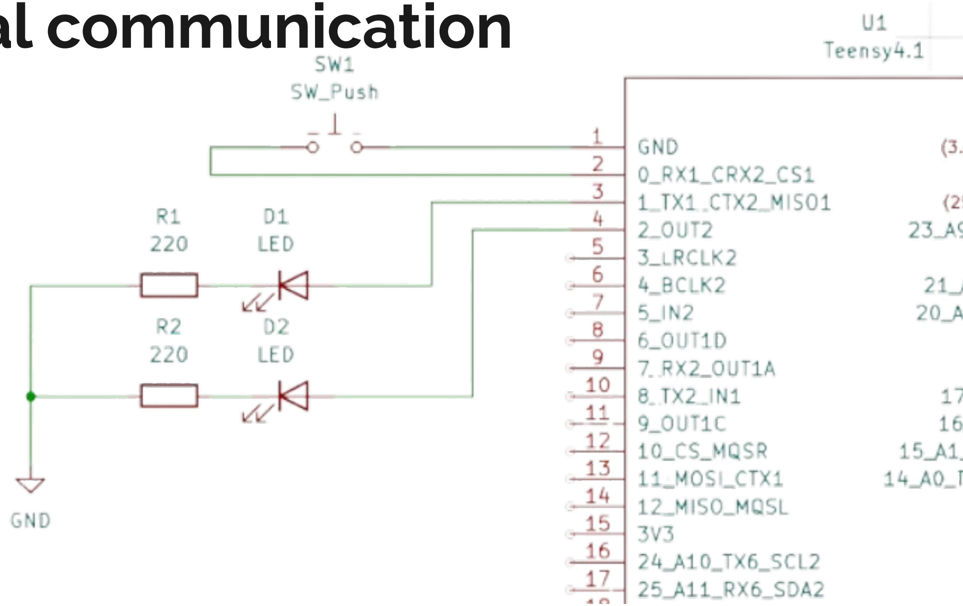
Demo: digitalRead and Serial communication

```
int buttonPin = 0;
int pressCount = 0;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600); // Start the serial communication
}

void loop() {
  digitalWrite(LED_BUILTIN, LOW);
  if (digitalRead(buttonPin) == LOW) {
    digitalWrite(LED_BUILTIN, HIGH);
    pressCount++; // Increment counter
    Serial.print("Button pressed ");
    Serial.print(pressCount);
    Serial.println(" times");
    delay(200); // Simple delay to avoid immediate repeated counting
  }
}
```



Demo: interrupts and Serial communication

```
int buttonPin = 0;
int counter = 0;

void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  Serial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(buttonPin),
buttonPressed, FALLING);
}

void loop() {
  counter++;
  Serial.print("Counter value: ");
  Serial.println(counter);
  delay(100); // Wait for 1 second
}

// Interrupt Service Routine (ISR)
void buttonPressed() {
  counter = 0;
}
```



Arduino

- Integrated development environment
- Development boards with many features, shields
- Bootloaders, easy to program
- Popular, online community with extensive resources
- Portable (usually)
- **However: simplicity and abstraction results in increased bloat**

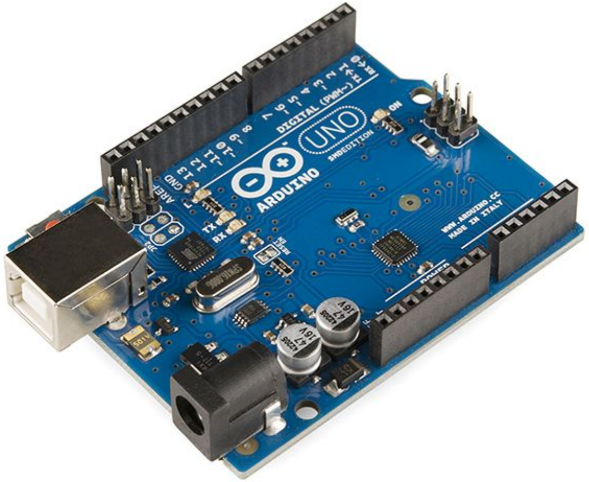


Image credit
Wikimedia commons

Abstractions

```
digitalWrite(PIN,VALUE); =
```

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    uint8_t oldSREG = SREG;
    cli();

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;
}
```

Abstractions

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

```
Sketch uses 924 bytes
```

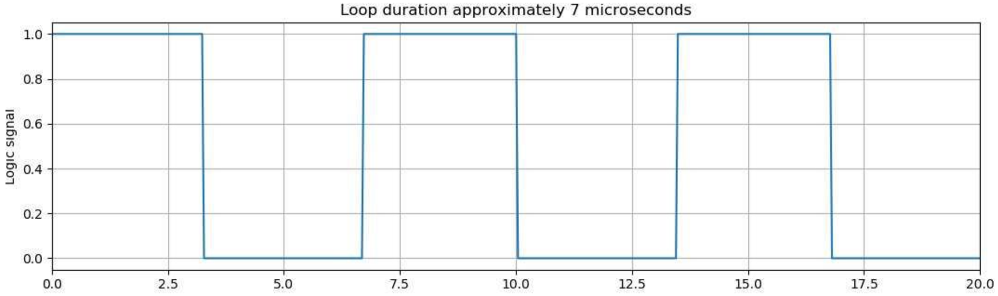
```
void setup() {
  DDRB = (1<<PB5);
}

void loop() {
  PORTB = (1<<PB5);
  _delay_ms(1000);
  PORTB = (0<<PB5);
  _delay_ms(1000);
}
```

```
Sketch uses 492 bytes
```

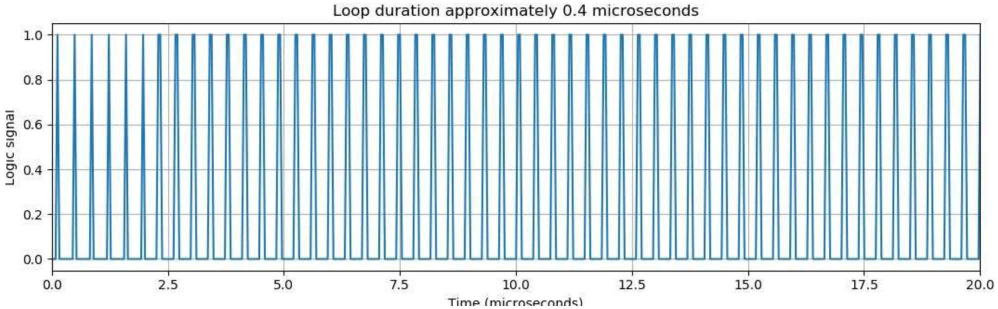
```
int main(){
  DDRB = (1<<PB5);
  while(1){
    PORTB=(1<<PB5);
    _delay_ms(1000);
    PORTB = (0<<PB5);
    _delay_ms(1000);
  }
}
```

```
Sketch uses 178 bytes
```

Abstractions



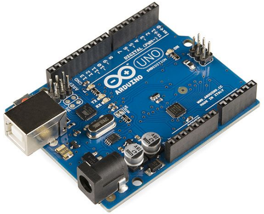

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   )
  digitalWrite(LED_BUILTIN, LOW);    ;
 }
```



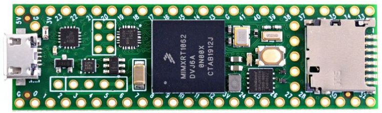

```
void setup() {
  DDRB = (1 << PB5);
}

void loop() {
  PORTB = (1<<PB5);
  PORTB = (0<<PB5);
 }
```

Examples of common microcontroller platforms

Arduino



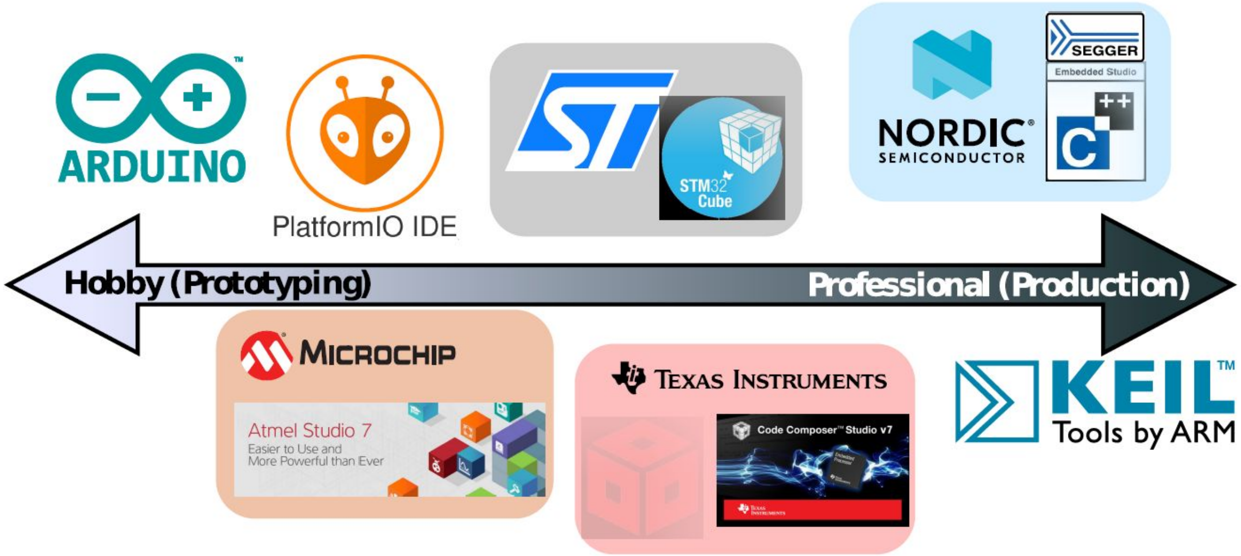
Teensy (PJRC)



STM32

ESP32

Development environments



Real-time requirements

- Real-time operating systems can be used to schedule tasks
- Guaranteed response time to an event

Real time means that there is a deterministic response time!

Real time does not mean fast!

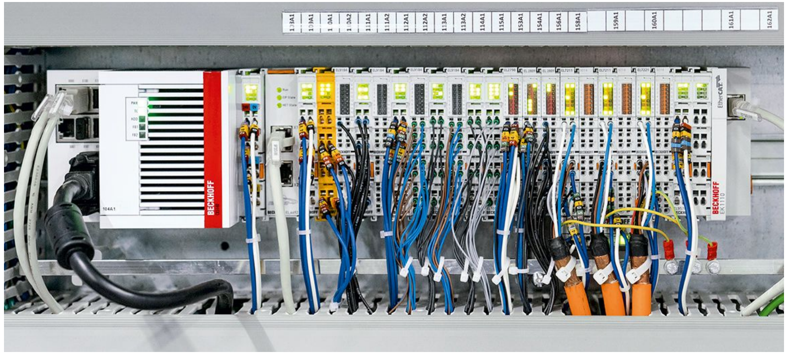
PLC programming

PLC's are ruggedized microcontrollers for industrial use. Can include protection against vibration, overvoltage, short circuit, high or low temperatures, dust, water etc.

Standardized in IEC 61131-3. Manufacturers provide IDE’s.

- Ladder diagram (LD)
- Function block diagram (FBD)
- Structured text (ST)

Siemens, Beckhoff etc.



Beckhoff PLC and modules
Image credit:
Beckhoff

Future developments

- Increasing computing power
- GUI libraries (LVGL)
- Edge computing
- Support for AI and machine learning (inference)
- Lower energy consumption
- Security features (prevent running unsigned code)
- Further improved and integrated connectivity

Demo: PCB design with KiCad

- Open-source EDA

  1. Create schematic from symbols, labels and wires

  2. Assign footprints

  3. Design routing on PCB (autorouters)

- Generate Gerber files for production
- Multiple companies produce PCB's and offer automatic board assembly

Contact

Please feel free to contact me. I will do my best to answer any of your questions.

Tuomas Tiainen
tuomas.tiainen@iki.fi

Special thanks to Dr. Ville Klar for providing slide templates.