Microcontrollers

KON-C2004 - Mechatronics Basics, Lecture notes

Ville Klar

26.11.2020

Contents

1 Introduction **2**
1.1 Programmable logic . . . 2
1.2 Types of programmable logic . . . 3

2 Microcontrollers: A great place to start **4**
2.1 What does an MCU look like? . . . 5
2.2 What is inside ? . . . 5
2.3 CPU . . . 6
2.4 Memory . . . 8
2.5 Peripherals . . . 8
2.6 General purpose inputs and outputs (GPIO) . . . 9
2.7 Counters and timers . . . 10
2.8 ADC . . . 11
2.9 DAC . . . 12
2.10 Communication . . . 12

3 Development **13**
3.1 Flashing the firmware . . . 13
3.2 Program structure . . . 14
3.3 Interrupts . . . 14
3.4 Arduino . . . 15
3.5 Abstractions . . . 16

4 Development environments **18**

5 Future **19**
5.1 tinyML . . . 19
5.2 RISC-V . . . 19
5.3 Decreasing abstraction costs . . . 19

6 Suggested reading: **21**

These notes accompany the KON-C2004 - Mechatronics Basics course slides.

1 Introduction

Electronics is one of the “pillars”_ of mechatronics and therefore at least a cursory understanding of it is important. It is vast topic and you have already covered topics like motors, sensors and control. Electronics is typically divided into the digital and analog domains. Most measurement systems require understanding of both domains (*mixed signal*)

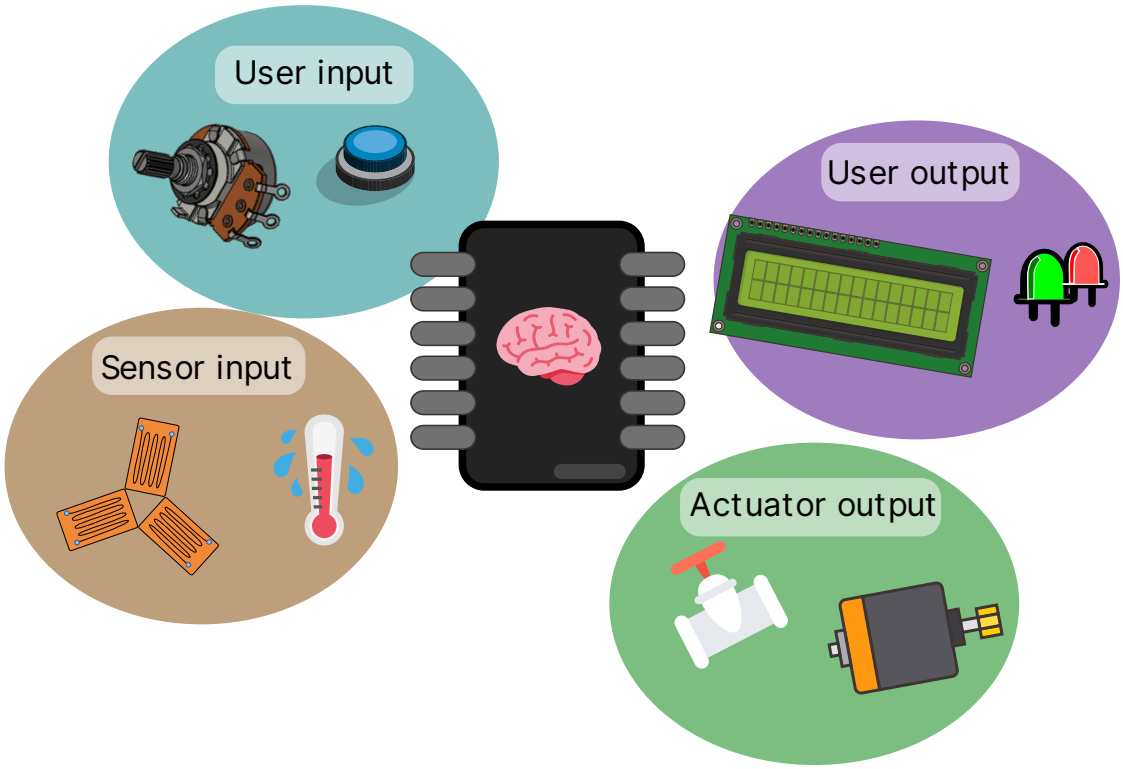
Programmable logic is one of the core topics/ technologies in digital electronics. Programmable logic enables perfroming actions (either internal or relating to outputs) based on inputs according to a defined logic. It is the computational cornerstone on which mechatronic machines rest on.

1.1 Programmable logic

The first programmable logic devices were introduced in the 18th by french textile engineer Basile Bouchon. The purpose of the earliest programmable devices was to automate the control of looms by ‘programming them’ with punched holes in paper tape. In the following centuries the punch cards slowly generalized a medium for storing and executing more general computational task sequences. While punch cards did remain in use until the early 1900s, electromechanical and later semi-conductor based systems replaced these systems.

Mechatronic systems are typically described in terms of inputs and outputs. Both inputs and outputs can relate either to the user or to the environment (sensors and actuators). Programmable logic is the ‘brain-box’ that interfaces system components to each other.

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



1.2 Types of programmable logic

In the semi-conductor era (1960→) there have been several types of programmable logic. The most relevant ones for a 21st century mechatronics engineer are listed below.

Discrete logic

- Individual chips performing simple logic operations (e.g. logic gates and timers)
- Obsolete technology, but a great way to learn fundamentals that apply to state of the art systems

Microcontroller unit (MCU)

- "Small computer" in the form of an integrated circuit that executes instructions that have been programmed into its memory.
- Has the capability to execute simple instructions which when sequentially combined enable completion of complex tasks
- An integrated circuit that contains a central processing unit, memory and peripherals

Microprocessor unit (MPU)

- An integrated circuit that contains a central processing unit
- Similar to an MCU but peripherals and memory not integrated into one chip
  - Requires external memory and peripheral components (similarly to computer CPU)

Field programmable gate array (FPGA)

- An integrated circuit that consists of thousands of programmable logic gates
- Does not execute instructions in the same way as an MCU but logic operations are based on gate configurations
- FPGA are typically highly parallelized and performant
- The software written for FPGAs referred to as gateware
  - The two most common gateware languages are Verilog and VHDL
- Complex programmable logic device (CPLD) is similar FPGAs but contains less programmable gates and is typically more context specific (e.g. purpose-designed for signal processing)

Application specific integrated circuit (ASIC)

- An IC with a specific functionality (for example microwave front panel controller)
  - Usually only warranted with very large production runs (or chips that are used in several product generations)
- All the functionality is programmed 'into the silicon'
- Development usually starts with FPGAs and once the gateware has been deemed functional the same gates can be transferred onto silicon
- Multicomponent module (MCM) is a similar approach but can contain multiple chips and does not necessarily contain custom chips

Programmable logic controller (PLC)

- A 'ruggedized' industrial computational device typically mounted on DIN rails
- The value statement of PLC is that the system integrator does not have to worry about connectors, enclosures, circuit boards, input protection etc.
  - Usually substantially more expensive than using e.g. a development board
- Typically contains one or more MCUs, CPLDs or FPGAs
- PLCs may be restricted to simple program logic and development environments because the PLC manufacturer dictates the capabilities
  - CODESYS is a common PLC development environment

System on Chip (SoC)

- In the past decade an increasing amount of functionality has been integrated into a single chip
- The term System on Chip (SoC) is used for integrated circuits that contain both a microcontroller or microprocessor along with advanced peripherals like graphics processing units (GPU), wireless modules, or coprocessors.

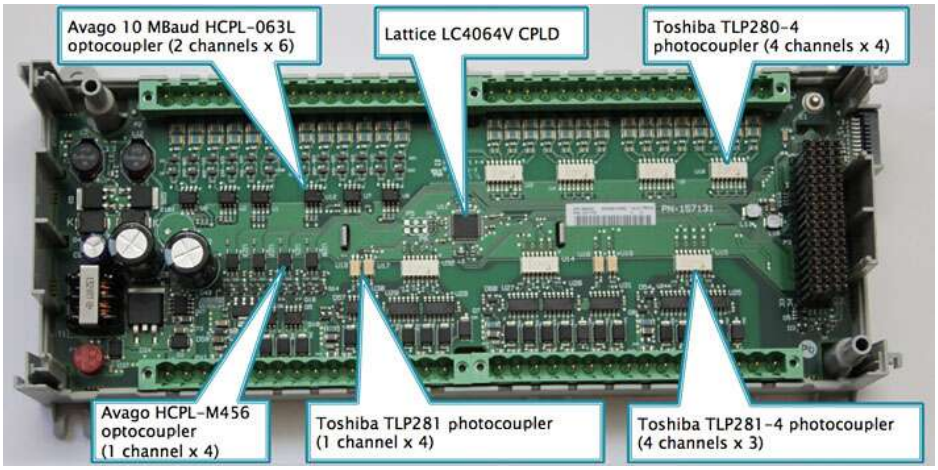


Figure 1: PLC teardown

- Similar to how a microcontroller integrates a microprocessor with peripheral circuits and memory, an SoC integrates a microcontroller with further advanced peripherals.

Single board computer (SBC)

- A full computer running a proper operating system (typically Linux)
- Capable of executing complex parallel tasks where the OS handles low-level timing and prioritization
- *Embedded Linux* development typically involves custom kernels, system calls and working across several layers of firmware & software (with e.g. network stacks)

2 Microcontrollers: A great place to start

Arguably the microcontroller is the most approachable inexpensive low-level form of programmable logic. Therefore, it is a great starting point and a great way to familiarize yourself with the basics of electronics. Microcontrollers are prevalent in embedded systems and the are both highly available and versatile. MCUs are very diverse in terms of cost, performance and features. The cost of a single MCU chip ranges from a few cents to $100 +. Common MCUs cost between $1 and $3.

Other features include

Amount of I/O

Number of IO ranges from 2-3 to hundreds. Number of analog inputs ranges from 0 to dozens (ADC commonly multiplexed). Some pins on an MCU have specific functionality and some have restricted functionality (e.g. input only).

Operating and logic level voltage

Modern MCUs use logic levels of 3v3 or less and it is common to have also 5V tolerant pins. It is important to be aware of the system logic level as incompatible logic level voltages may cause damage.

Power consumption

Most MCUs feature sleep states to facilitate power consumption optimization. Typical current consumption in a sleep state is measured in µA or nA. Sleep states vary from “light” to “deep” to “ultra-deep”. Waking up from a sleep state usually based on external trigger. You can also use a Real time clock module (RTC) to wake up once a day. It is possible to configure an MCU in a way that it wakes up when any button is pressed and the proceeds to read the pressed button immediately after waking up. For example, a remote control is almost always in a sleep state and wakes up when any button is pushed and then sends the light pulse sequence corresponding to that button, and goes back to sleep.

2.1 What does an MCU look like?

MCUs come in a variety of *packages*. Through-hole technology practically obsolete (at least regarding MCUs). Nonetheless, e.g. Arduino UNO designs available with both a surface mount (QFP) and through hole (DIP) atmega328. Try not to confuse the development board with the microcontroller itself.

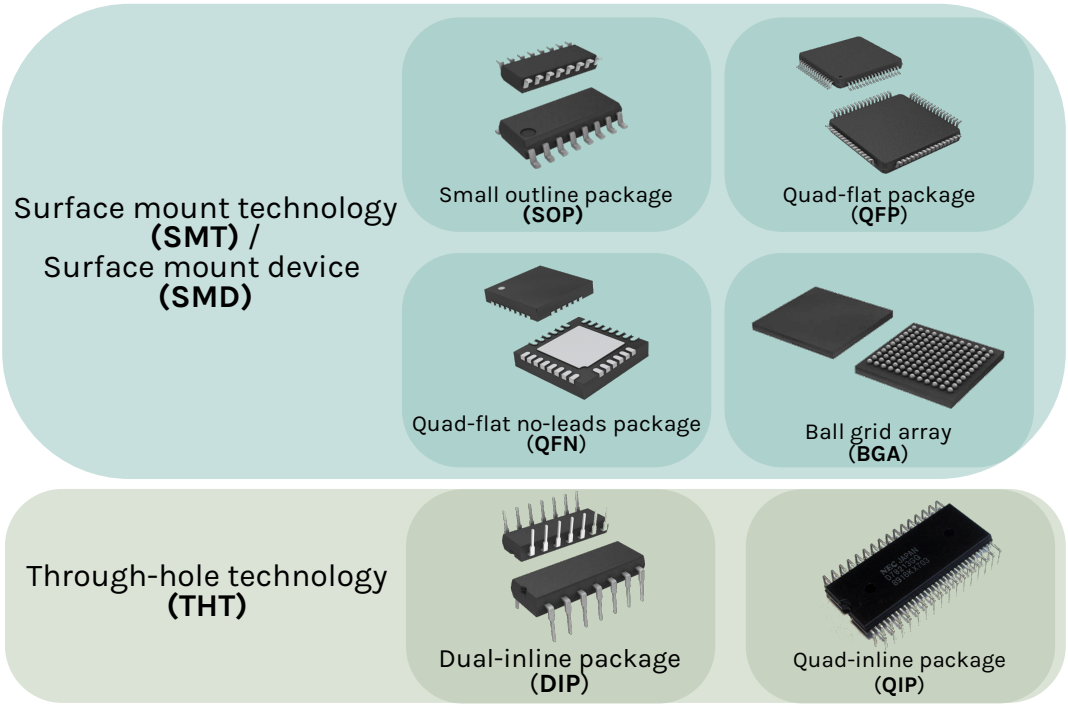


Figure 2: Common packages

It is possible to purchase MCUs for extended temperature and pressure range. Common MCUs available for space or military applications (better materials and more stringent quality control).

2.2 What is inside ?

Figure 3 depicts a simplified diagram of the microcontroller. Three main parts / sections are

- CPU
- Memory
- Peripherals (which can be further divided into e.g. communication specific peripherals)

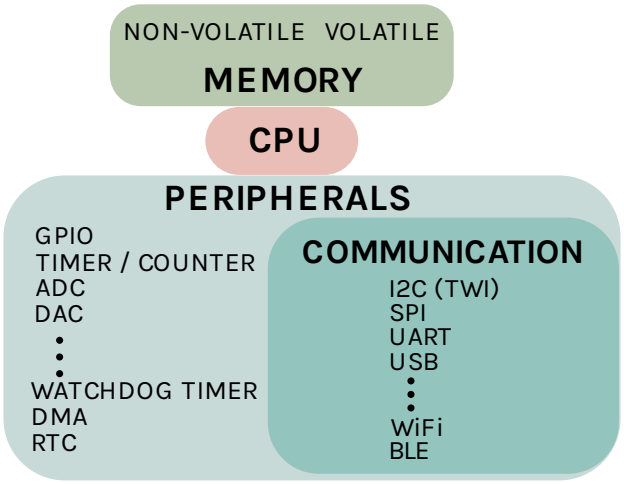


Figure 3: Simplified diagram of an MCU

2.3 CPU

Fetch decode execute loop

The three stages of instruction execution within a CPU.

Arithmetic logic unit

Arithmetic Logic Unit (ALU) is capable of performing arithmetic operations (e.g. Addition, Subtraction etc.) and logical operations (e.g. AND, OR, Ex-OR, Invert etc.). It typically receives two operands and an 'opcode' and output and outputs a result.

Program counter

The program counter is a special register that holds the memory address of the next instruction to be executed.

Stack pointer

The stack is a data structure implemented in volatile memory and is used to store addresses and data quickly. The stack pointer stores the address of the most recent entry that was pushed onto the stack. The stack and stack pointer are the fundamental mechanism with which the CPU accesses and manipulates memory.

Instruction register

To execute an instruction, the processor copies the instruction code from the program memory into the instruction register. From there, the instruction is decoded by the instruction decoder.

System bus

The processor uses and internal bus to communicate address and data. Three different buses are used: control bus, address bus and data bus

Bit-ness: 4-bit, 8-bit, 16-bit, 32-bit The "bit-ness" of a microcontroller refers to the width of the arithmetic logic unit (ALU). Being able to handle larger (e.g. floating point numbers with more decimals) with single instructions typically results in better performance.

You can think of instructions as the atomic programming language of the microcontroller (cannot be divided into further parts). These instructions are very simple (e.g. add numbers, jump to memory location, store number etc.)

Not all MCU employ the same instruction set architecture (ISA). RISC (Reduced instruction set computer) is a common ISA used by e.g. atmega328 MCUs. The atmega328 has an "Advanced RISC Architecture with 131 Powerful Instructions – Most Single Clock Cycle Execution" The available instructions are listed in the

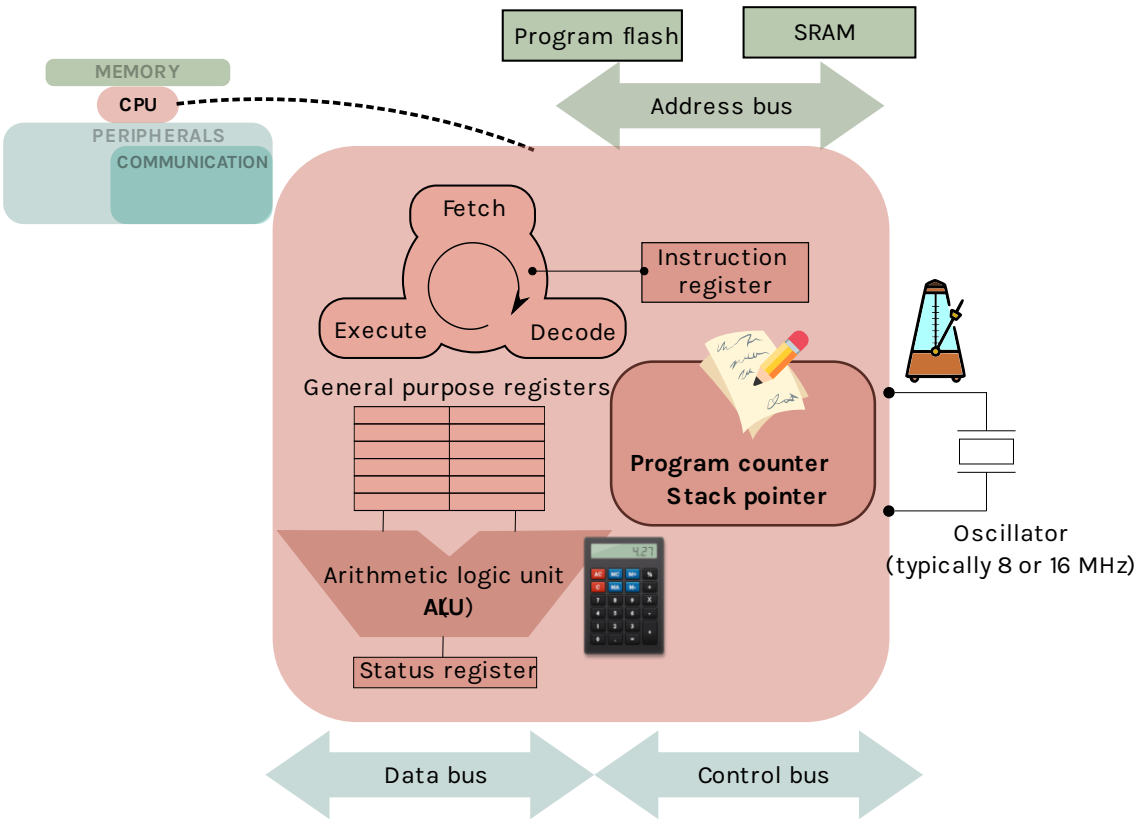


Figure 4: CPU diagram

datasheet in (in the case of atmega328 under section “Instruction set summary”). A well-known company that develops and licences instruction set architectures is ARM (*Advanced RISC Machine*, originally *Acorn RISC Machine*)

The job of the compiler (and linker) is to produce and efficient sequence of instructions from a high-level language such as C. Unless you write assembly you usually do not have to worry about instructions (you should trust the compiler unless you know what you are doing).

CPU Performance

A common metric of CPU performance is clock speed. MCU clock speeds range from sub MHz to over 1 GHz. The clock refers to a source of a square wave signal that triggers the execution of a new instruction. This square wave is typically generated with a crystal oscillator.

While clock speed gives some indication of performance, in reality it is not this simple. Some instruction take more clock cycles to complete and some instructions can be performed simultaneously (in parallel). Consequently, measuring the performance of an MCU is somewhat complicated. MIPS/MHz (million instructions per second) is one metric. Another common metric is is DMIPS (Dhrystone MIPS), where ‘Dhrystone’ refers to a synthetic benchmark program. Arduino Uno (Atmega328p) has a performance of 1 DMIPS/MHz. So if you have a clock speed of 16 MHz you can execute approximately 16 million instruction per second. STM32 ARM (STM32F103×8) is 1.25 DMIPS/MHz and the maximum frequency (clock speed) is 72 MHz (quite a bit more performant). Program memory may also be a performance bottleneck in some applications like using large network stacks or large matrix computations.

The price-performance ratio in MCUs is non-linear. For example, STM32F103×8 has more peripherals and better specifications in general compared to Atmega328p. Nonetheless, Atmega328p currently costs ~1.5 € per unit when bought in bulk whereas STM32F103×8 costs ~1 € per unit when bought in bulk. Software support and driver availability may also be more important factor than cost-performance ratio.

2.4 Memory

Two different memory categories are distinguished; volatile and non-volatile. Volatile memory is lost between power cycles & resets and it is used to store run time variables. A common type of volatile memory is static random access memory (SRAM). Similarly to desktop computers, SRAM is characterized by low capacity but high speed.

The second category, non-volatile memory, is kept between power cycles & resets. Two common types of non-volatile memory is Flash and EEPROM. Flash is mid-capacity, fairly fast in read & write and permanent. Flash memory may be partitioned and can also include a file system. The program is typically stored in flash memory.

Electrically Erasable Programmable Read-Only Memory (EEPROM) is high capacity but slow in terms of read & write speed. Variables stored in EEPROM are changed during program execution and stored between power cycles. It is typically used to store semi-permanent variables (e.g. time stamps of resets or dynamic configuration values). EEPROM is byte erasable whereas flash is only block erasable. Other important factors regarding memory are amount of erase / write cycles and data retention (typically measured in years).

2.5 Peripherals

Peripherals refer to the capabilities of the MCU that extend its functionality beyond CPU specific tasks. In terms of an MPU the peripherals are more clearly distinct as they are on separate discrete chips whereas in an MCU they are integrated into the chip (even memory is a peripheral device in an MCU). Other common peripherals include

- General purpose inputs and outputs (GPIO)
- Timer & counters
- Analog to digital converters (ADC)
- Digital to analog converters (DAC)
- Interrupts
- & many more which are outside the scope of this lecture (DMA, Watchdogs,)

2.6 General purpose inputs and outputs (GPIO)

General purpose inputs and outputs refer to the various pins of an MCU that can be used for a variety of tasks (*general purposes*). The most simple form of such a purpose is digital IO. Pins of the MCU can be written to (set to a high or low state) or read from (measure high or low state). The pins belong to a certain port of the MCU which may have several registers. The values in these registers correspond to the pin value. In Arduino, register operations are rarely encountered and digital IO is abstracted into a simpler *digitalRead* and *digitalWrite* syntax.

```
//The onboard LED of Arduino UNO: pin 13
void setup() {
   pinMode(13, OUTPUT);//Set pin to output
}
void loop() {
   digitalWrite(13, HIGH);//turn LED on
   delay(1000);//wait for a second
   digitalWrite(13, LOW);//turn LED off
   delay(1000);//wait for a second
}
```

Figure 5: Blink program in Arduino

When employing digital logic it is important to make sure that all digital inputs always have a determined state. A situation where an input is in an indeterminate state (voltage) is called floating. Floating should be avoided as it can result in erratic system behaviour. A common way to avoid floating is to add pull-up or pull-down resistors (typical value 1k to 10k) to tie the pin to a specific default state. Some MCUs also have internal pull-ups resistor which can be enabled in firmware.

MCUs (and generic GPIO in general) can typically sink/source ~50 mA. In cases where larger currents are necessary, transistors may be used to drive larger loads.

2.7 Counters and timers

Counter & timer refer to a peripheral that is designed to count and time tasks (usually counting state changes or setting / unsetting a pin). Counter & timers have many use cases but for a mechatronic engineer, pulse width modulation (PWM) is arguably the most important one. In PWM, the output power is modulated by adjusting the ratio between on and off state (i.e. **pulse width**) instead of adjusting amplitude. Such an approach is more suitable for MCUs as they are *"better"* at generating digital signals. Turning something on and off is easier than generating an arbitrary value. PWM performed with different frequencies (**periods**), therefore pulse width is quantified by **duty cycle** (%)

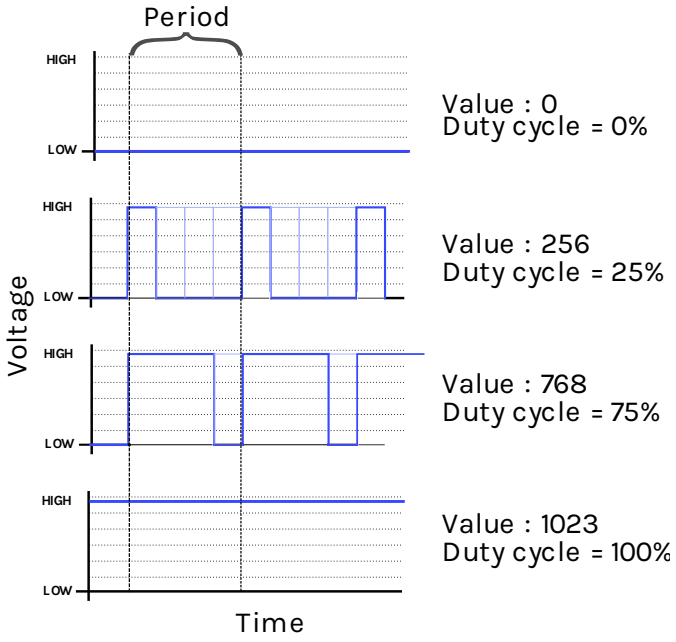


Figure 6: 10-bit PWM with value and duty cycle

Timers and counters can have different sizes and the can also be scaled using prescalers (multipliers).

- 8-bit counter → 2^8 → values 0-255
- 16-bit counter → 2^{16} → values 0-65535

In traditional MCU programming, the generation of PWM involves calculating suitable prescalers and Output Compare Register (OCR) values. In Arduino the somewhat misleadingly named *analogWrite* function is used for generating a PWM.

2.8 ADC

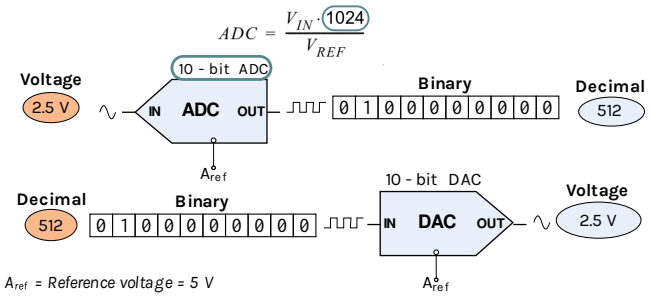


Figure 7: ADC and DAC

An analog digital converter turns a voltage into ones and zeros. The ADC value is defined by equation,

$$ADC = \frac{V_{in} \cdot 2^n}{V_{ref}}$$

,

where ADC is value outputted by conversion, V_{in} is the voltage being measured, n is the amount of bit used in the conversion and V_{ref} is the reference voltage.

The amount bits used in the conversion gives the resolution of the measurement

Given V_{ref} = 5 V

- 8-bit (0-255), resolution is ~ 20mV
- 10-bit (0-1023), resolution is ~ 5 mV
- 12-bit (0-4095), resolution is ~ 1,2 mV

Note that measurement error, resolution and noise are linked but distinct topics.

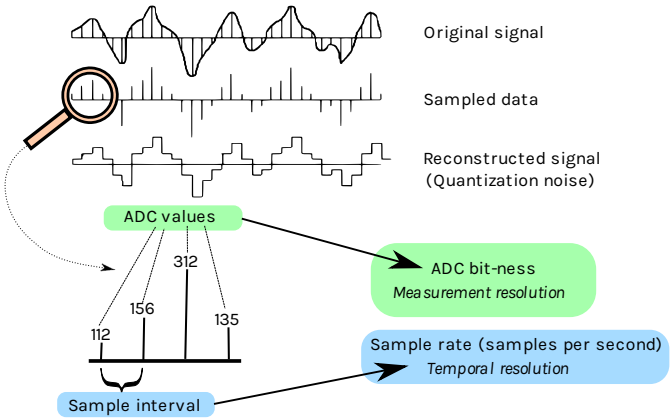


Figure 8: ADC resolution

Typically higher bit conversions take longer time to complete compared to lower bits. Also note that the reference voltage can be dimensioned according to the amplitude of the measured signal.

There are several different types of ADCs (e.g. successive approximation, parallel comparator, delta-sigma). Most common MCUs employ a successive approximation ADC which is based on use of multiple comparator circuits in succession. Sample and hold circuitry is used to 'retain' the sample during this successive approximation. Additionally, ADC are typically multiplexed meaning one ADC is used for several input pins sequentially.

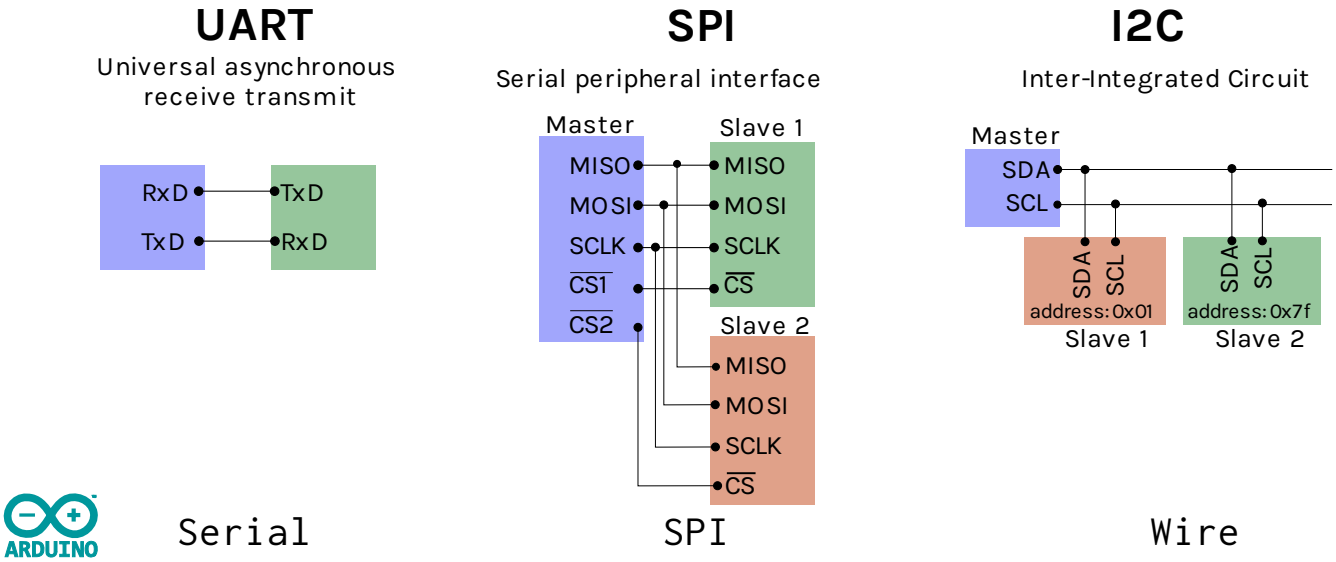
2.9 DAC

The DAC can be though of as the inverse of an ADC, it turns ones and zeros into voltage. Unlike ADCs, DACs are not not available in lower tier MCUs where PWM is a more common approach for modulating power. DACs are very useful in signal generation and commonly used for audio purposes.

2.10 Communication

An MCU has a wide variety of methods to communcate with external sensors and actuators. It is common to employ a *communication bus* for communication. Wired communication buses can be either serial (bits sent on single conductor sequentially) or parallel (bits sent on multiple conductors in parallel). As modern semi-conductors are capable of high speed switching and reading, serial buses are more common in the domain of conventional MCU communication. If you are tinkering with Arduino you may encounter vestigial parallel interfaces e.g. (Hitachi LCD 4-bit parallel interface)

The three most common communication buses are



SPI

- MISO, MOSI, SCLK and CS
- SD cards support SPI-bus communication

I2C

- SDA and SCL
- WiNunChuk uses I2C bus

UART

- RX and TX
- Sometimes referred to as Serial (do not confuse with RS-232)

Don't confuse between electrical standards and communication protocol standards. Modern MCUs may also feature wireless communication capapbilites (e.g. BLE and WiFi).

3 Development

Developing for MCUs involves writing *firmware*, which referes to software for hardware. Firmware is typically written in C or C++. In the domain if firmware, C is considered a high level language.

Compiling, linking and generating a suitable output to upload to the MCU requires software toolchains. Typically the MCU manufacture supplies a suitable software toolchain and many MCUs support multiple different toolchains. With Arduino, the software toolchain is abstracted away and little user configuration is necessary to get up and running with different MCUs.

3.1 Flashing the firmware

Once the source files have been transformed into a sequence of instructions in suitable format for the MCU they are written (or ”burned” / ”flashed”) into the non-volatile memory of the MCU. These programs stay in memory between power cycles. Flashing firmware can be done using USB (typically requires extra chips and bootloader) or external programmer.

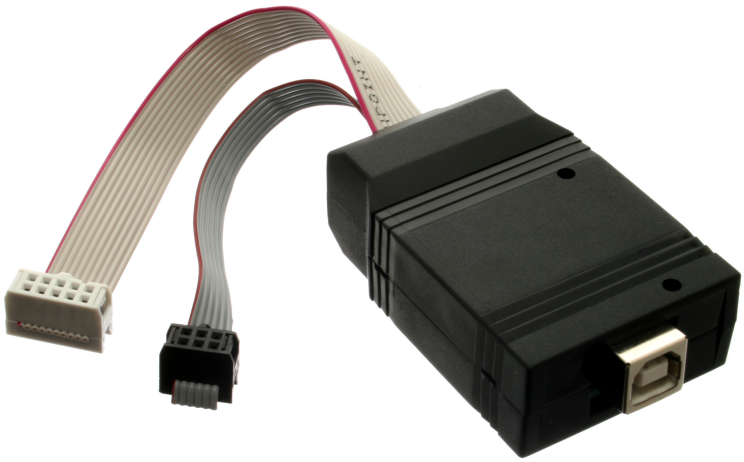


Figure 9: Adafruit TinyISP

Nowadays programmers typically have additional debug functionality such as the ability to use breakpoints and inspect memory addresses during program execution.

3.2 Program structure

The typical simple embedded program follows the structure shown in the figure below.

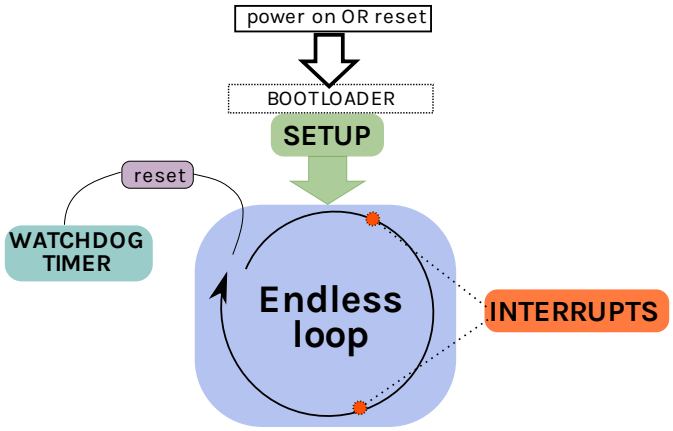


Figure 10: Structure of an embedded program

There is a function (or section of main) where all inputs and outputs are setup and then there is a perpetual loop where the desired functions are completed. Most 'advanced' firmwares implement more sophisticated class based sequences. It is also common to adopt a real-time operating system (RTOS) for more advanced projects where task and dataflow management can be done using e.g. semaphores and queues. See FreeRTOS, Zephyr or Riot for MCU / SoC RTOS.

3.3 Interrupts

In many situations polling (repeated reading) of an input may be suboptimal. Interrupts are a mechanism that can be used to interrupt the main sequence of instructions

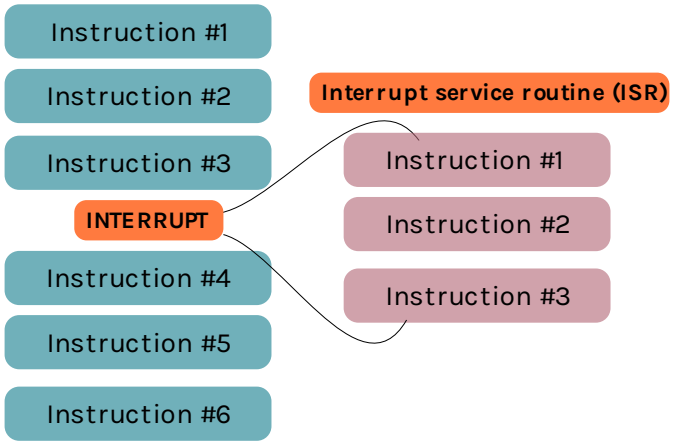


Figure 11: Interrupts and main prorgram

They are triggered by an interrupt source which can be external of internal. Internal interrupts include timer interrupts, adc ready etc. External interrupts (rising, falling or change on a pin) e.g. button is pushed and immediate response is required. **Note**: E.g. Arduino UNO,NANO and Mini (based on the same MCU) only have 2 interrupt capable pins. It is also common to use interrupts to trigger communication sequences e.g. measurement ready interrupt on an external sensor IC.

The function run in interrupt is called an interrupt service routine (ISR) Rule of thumb: Try to minimize time spent in ISR

3.4 Arduino

1. Integrated development environment (IDE)
2. Development boards (AVR-based and other)
3. Bootloader / arduino core
4. Online community and libraries

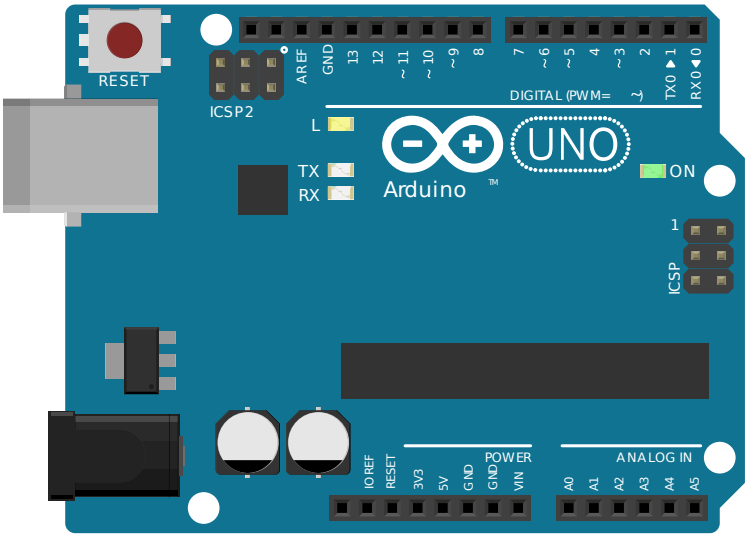


Figure 12: Arduino UNO

At mechatronics projects here at the University Arduinos are commonly used. If you are in the prototyping stage, you should optimize speed of development and ease of use rather price or performance. For that, Arduino is a great choice. The common heuristic here is: UNO is “vanilla”, NANO is compact “vanilla”, Pro Micro if even more compact, ZERO is performant “vanilla”, MEGA is more performant and has more IO You can buy official boards fro 8 – 50 €. Chinese clones are available for less than a tenth of the price (close to BOM cost). Beware of counterfeit chips which can behave in strange ways.

Selling Arduino clones is completely legal but some Chinese vendors operate close to and beyond boundaries of trademark infringement. Chinese clones may also have quality / software compatibility issues. You get what you pay for.

Arduino is sometimes used as a colloquial term to refer to microcontrollers in general. You might upset some embedded engineers by equating the two though. You have been warned. For a hobbyist, Arduino very approachable. The whole Arduino ecosystem is optimized for ease and speed. Also the most commonly known environment outside of the embedded engineering world. When you buy an Arduino, the development board comes with all the supporting circuitry.

3.5 Abstractions

Trade-off between granularity of control and ease of development. Typically it is good to use high-level abstractions as they make development fast. The source code of digitalWrite is quite verbose as the function needs to check many things in order to be 'universal'.

```
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;
}
```

Figure 13: Source code of digitalWrite function

This results in both a large memory footprint and slow execution (compared to direct register manipulation).

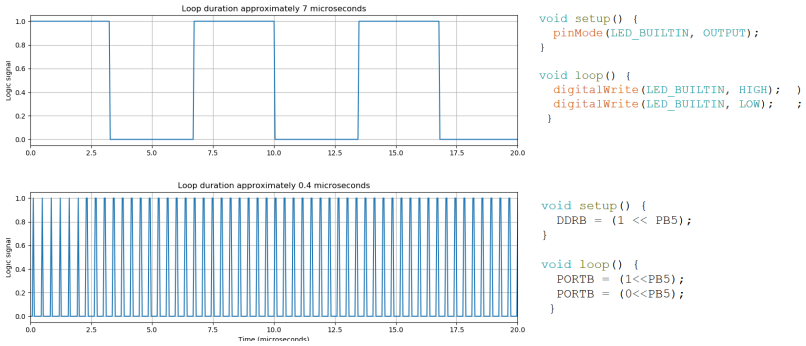


Figure 14: Digitalwrite cost in terms of time

```
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

Figure 15: Digitalwrite cost in terms of memory

Sometimes abstractions may also result in difficult to diagnose bugs.

STM32 Blink utilizes the hardware abstraction layers (HALs) written by STM which are slightly more verbose compared to standard Arduino. This is because the HALs are intended for embedded developers who prefer to have granularity of control over ease of use. The STM32 Cubemx software is a good example of a project configurator software which are common in production environments.

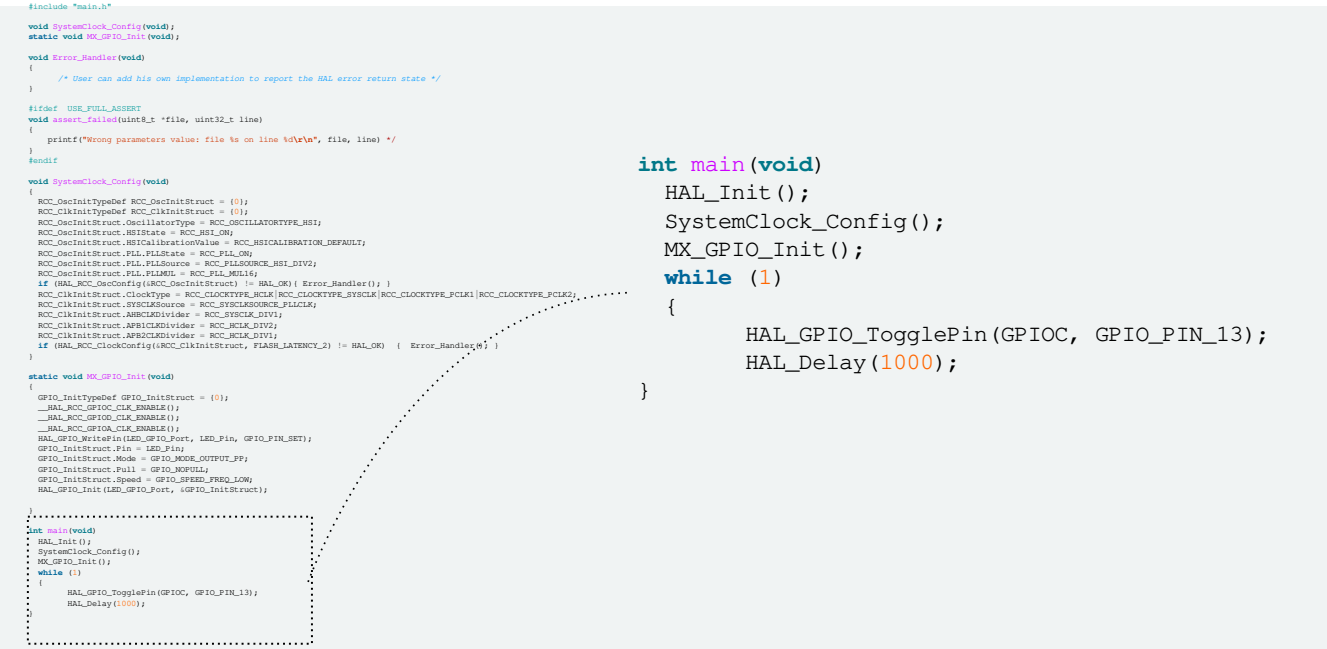


Figure 16: stm32 blink

4 Development environments

There are several development environments and they range from prototype-centric to production-centric.

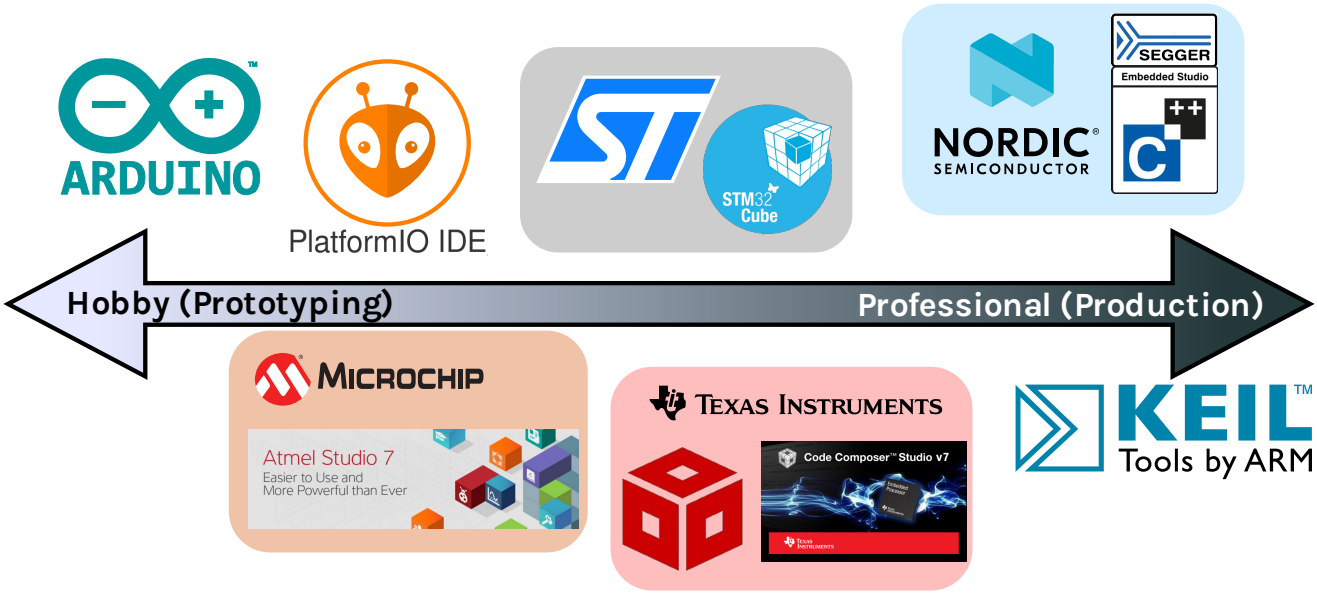


Figure 17: Development environments

5 Future

5.1 tinyML

It will become more and more common to run machine learning (e.g. neural networks) on resource contrained devices. The data acquistion and cloud training required to perform ML also means connectivity (e.g. wifi and ble) will be more necessary / ubiquitous. tinyML is a project / framework to keep and eye on.

Figure 18: tinyML book

The ML / AI trend also presents a new data driven programming paradigm where models and feeding data to them is a central part of the functionality of the embedded device.

5.2 RISC-V

RISC-V is an open standard instruction set architecture (ISA) based on reduced instruction set computer (RISC) principles. Unlike most other ISA designs, the RISC-V ISA is provided under open source licenses that do not require fees to use. This means more players will be able to enter the chip design field and optimize chips for more distinct use cases.

Figure 19: RISC-V

5.3 Decreasing abstraction costs

Future programming languages will be better at providing easy to read abstractions without any impact on performance, memory footprint etc (zero-cost abstractions). Rust is a new-ish system programming language that can also be used with microcontrollers.

```rust
// std and main are not available for bare metal software
#![no_std]
#![no_main]
extern crate stm32f1;
extern crate panic_halt;
extern crate cortex_m_rt;
use cortex_m_rt::entry;
use stm32f1::stm32f103;
// use `main` as the entry point of this application
#[entry]
fn main() -> ! {
    // get handles to the hardware
    let peripherals = stm32f103::Peripherals::take().unwrap();
    let gpioc = &peripherals.GPIOC;
    let rcc = &peripherals.RCC;

    // enable the GPIO clock for IO port C
    rcc.apb2enr.write(|w| w.iopcen().set_bit());
    gpioc.crh.write(|w| unsafe{
        w.mode13().bits(0b11);
        w.cnf13().bits(0b00)
    });

    loop{
        gpioc.bsrr.write(|w| w.bs13().set_bit());
        cortex_m::asm::delay(2000000);
        gpioc.brr.write(|w| w.br13().set_bit());
        cortex_m::asm::delay(2000000);
    }
}
```

Figure 20: Blink in Rust

6 Suggested reading:

- Laplante, Phillip A., and Seppo J. Ovaska. Real-time systems design and analysis: tools for the practitioner. John Wiley and Sons, 2011.
- For more information on microcontroller basics: (https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf)
- The Art of Electronics book