Embedded Systems

Lecture 0
Introduction

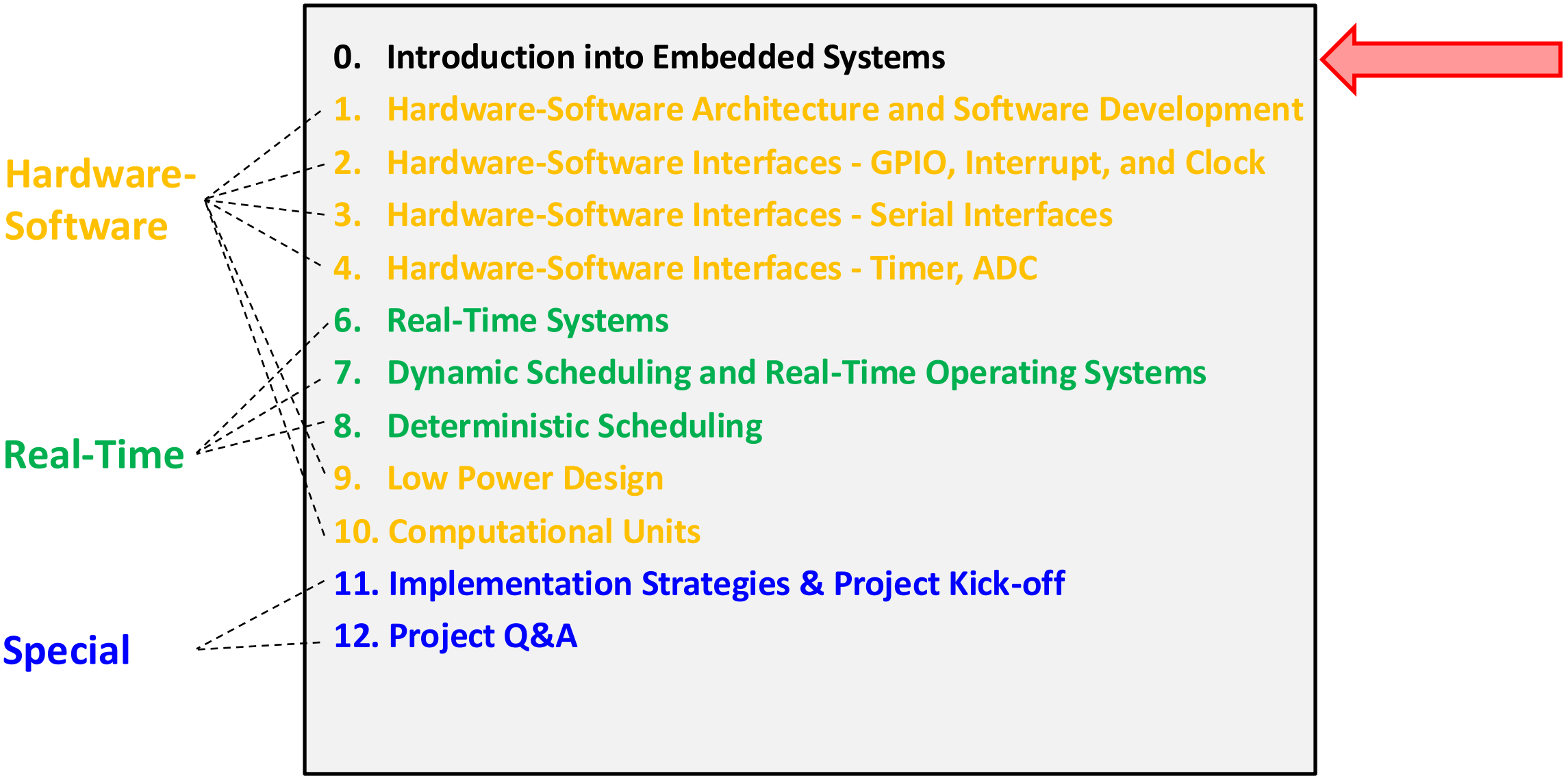
© Michele Magno

D-ITET Center for Project-Based Learning

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Where We are


Hardware-Software
Real-Time
Special
0. Introduction into Embedded Systems
1. Hardware-Software Architecture and Software Development
2. Hardware-Software Interfaces - GPIO, Interrupt, and Clock
3. Hardware-Software Interfaces - Serial Interfaces
4. Hardware-Software Interfaces - Timer, ADC
6. Real-Time Systems
7. Dynamic Scheduling and Real-Time Operating Systems
8. Deterministic Scheduling
9. Low Power Design
10. Computational Units
11. Implementation Strategies & Project Kick-off
12. Project Q&A

What You Will Learn

- **Theoretical foundations** and principles to analyze and design embedded systems.
- **Practical aspects** of embedded system design, mainly software design.

The course has three components:

Lecture: Communicate principles and practical aspects of embedded systems.

Exercise: *Practical*: Introduction into practical aspects of embedded systems design. Use of state-of-the-art hardware and design tools – **ARM Cortex-M4F cores**.

Theory: Use paper and pencil to deepen your understanding of analysis and design principles.

Project-Based Learning practical exercises!

Organization

Webpage: https://moodle-app2.let.ethz.ch/course/view.php?id=22898

Lecture: PD. Dr. Michele Magno

michele.magno@pbl.ee.ethz.ch

Literature:

- Yifeng Zhu: *Embedded Systems with Arm Cortex-M Microcontrollers in Assembly Language and C - Fourth Edition*, E-Man Press LLC, ISBN: 978-0982692677, 2023
- Giorgio C. Butazzo: *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*, Springer, ISBN 978-1-4614-3019-3, 2011

Sources: The slides contain ideas and material of J. Rabaey, K. Keuzer, M. Wolf, P. Marwedel, P. Koopman, E. Lee, P. Dutta, S. Seshia, and from the above cited books.

Who We Are

Webpage: https://pbl.ee.ethz.ch/people.html

Course Coordinator

Lukas Schulthess

lukas.schulthess@pbl.ee.ethz.ch

ETZ K 78.1

Lectures / Exam Coordinator

Julian Moosmann

julian.moosmann@pbl.ee.ethz.ch

ETF F 110

Exercises

Philipp Schilk

schilkp@ethz.ch

ETZ K 78.1

Lectures / Exam Coordinator

Jakub Mandula

jakub.mandula@pbl.ee.ethz.ch

ETF F 110

Lecture and Exercises

Lectures:

Held on **Mondays from 14:15 to 16:00 in ETF C1** until further notice.

Slides are available via the web page of the lecture.

Exercises:

Held **from 16:15 to 18:00 in ETF E1**.

Group A on Wednesday, and group B on Friday. **The group assignment is binding!**

The exercises will include practical programming tasks as well as theoretical questions. They are made available before the date of the exercise. During an exercise, you will apply the learned knowledge in theoretical and practical exercises. Teaching assistants give hints how to approach the solution, answer your questions and discuss the correct solution.

Solutions to the exercises will be provided on Monday of the following week.

Exercises are an integral part of the lecture and contain exam-relevant content.

Lab and Exercise Preparation

Preparation: We urgently ask all students to install the required software for the practical exercises on their own hardware prior the first exercise. You can find the installation guide for Windows, MacOS, and Linux on Moodle.

Moodle: Check the “Install Guide” on Moodle.

Questions: Ask in the exercise sessions or on the Moodle Forum for installation issues.

End-Semester Mini Project

Mini Project: At the end of the semester, we will offer a mini project in the style of the practical exercises. Constantly following and solving the practical sessions is the best way to acquire the knowledge for completing the bonus project successfully.

Bonus: If completed successfully, the final grade will be increased by 0,25.

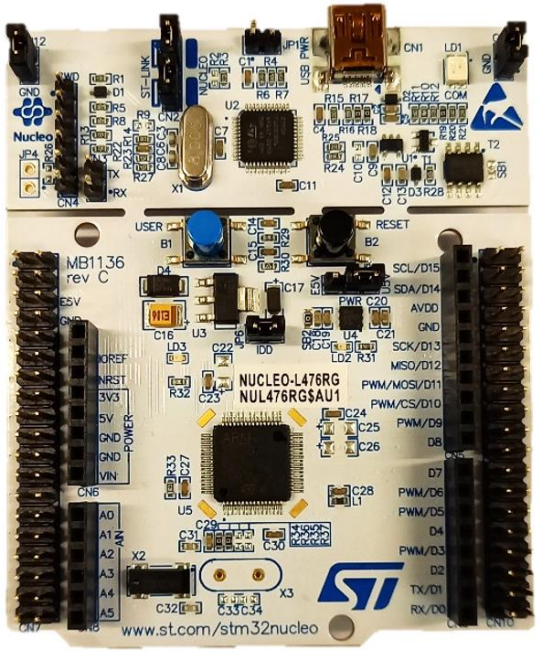
Details: Details will follow later in the semester.

Embedded System «Znüni Böxli»

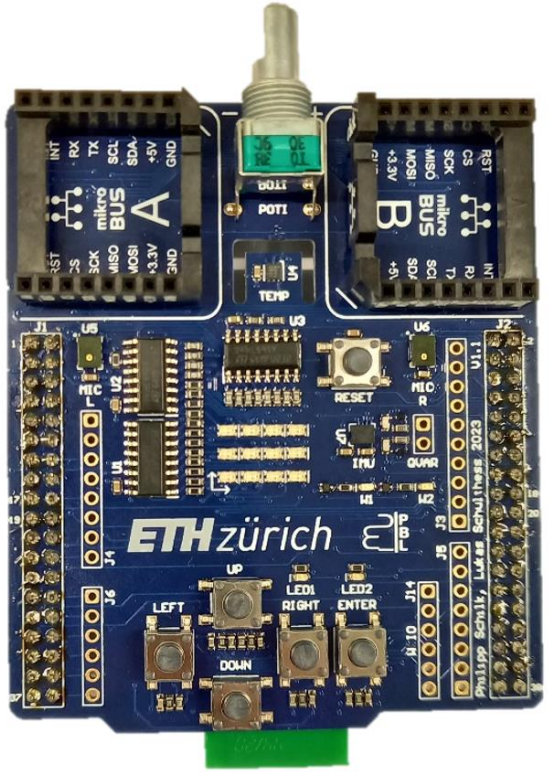
What’s inside the Box

Everyone gets:

1x NUCLEO-L476RG Board

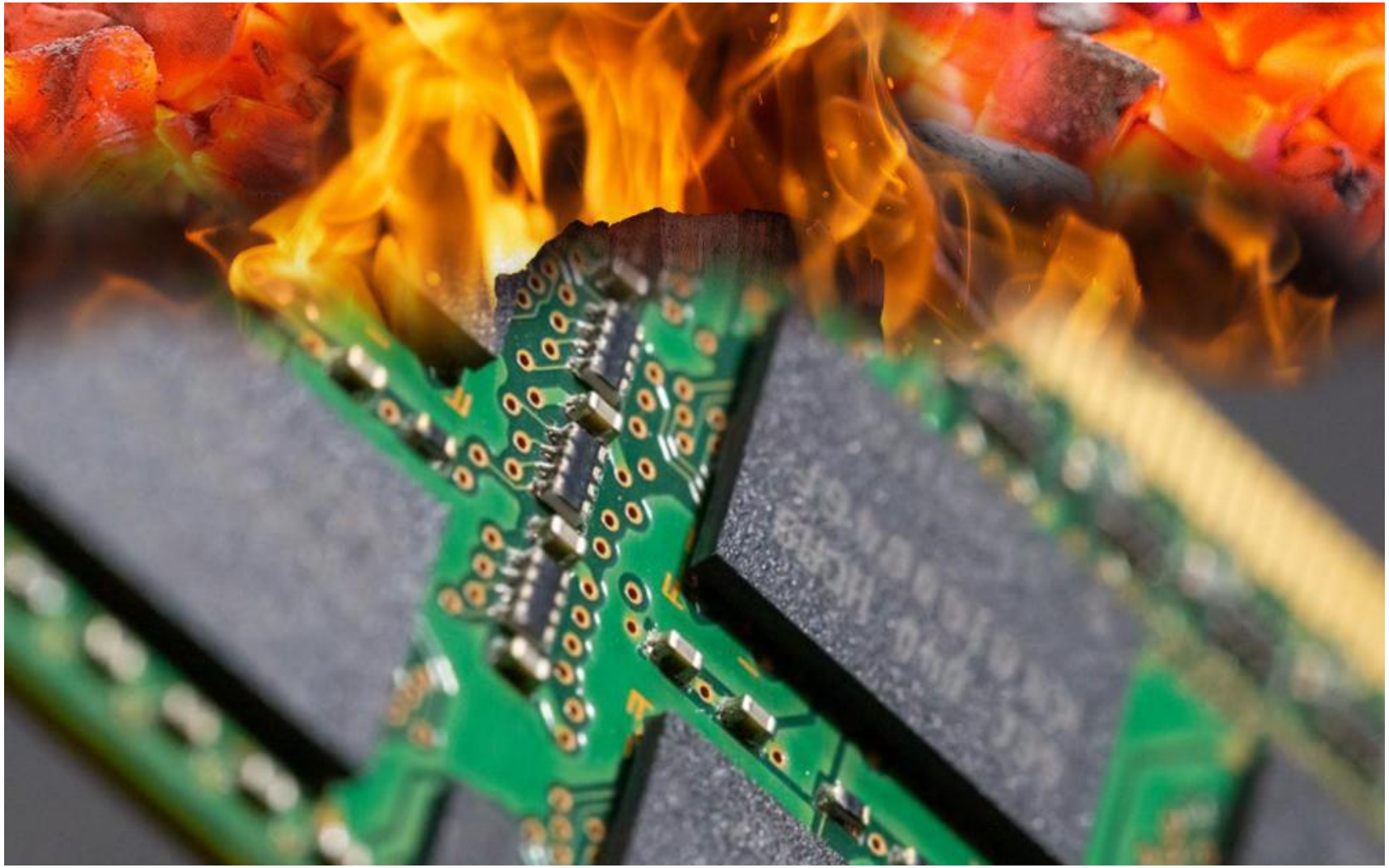


1x Embedded Systems Sensor Shield



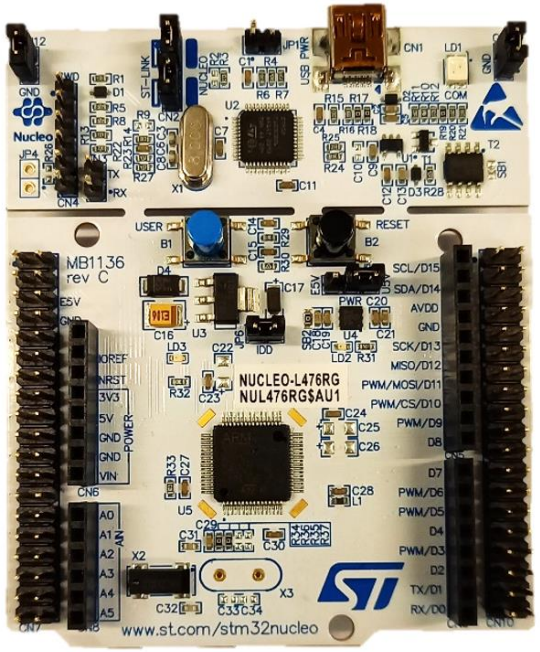
1x Mini-USB Cable

Be Careful and Please do Not …

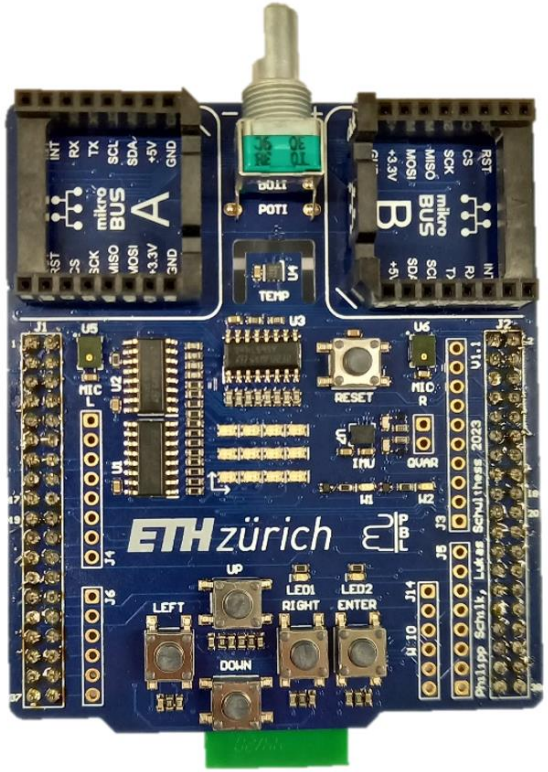


What You Need to Return

1x NUCLEO-L476RG Board



1x Embedded Systems Sensor Shield



1x Mini-USB Cable

Put it back inside the ”Znüni Böxli”.

Hardware Distribution

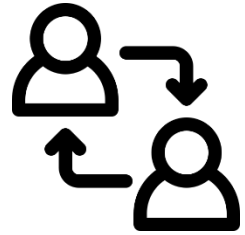
Distribution Days:

Wednesday**, 25.09.2024** beginning of the exercise for Group A

Friday, **27.09.2024** beginning of the exercise for Group B

Make sure to bring your Legi, please!

Interaction: What about You?



Clicker Questions in the EduApp

Why did you choose to attend the lecture on embedded systems?

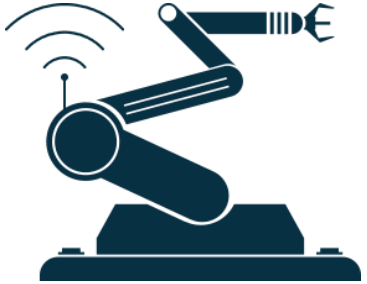
What is an Embedded System?

Embedded Systems - Impact

Embedded Systems

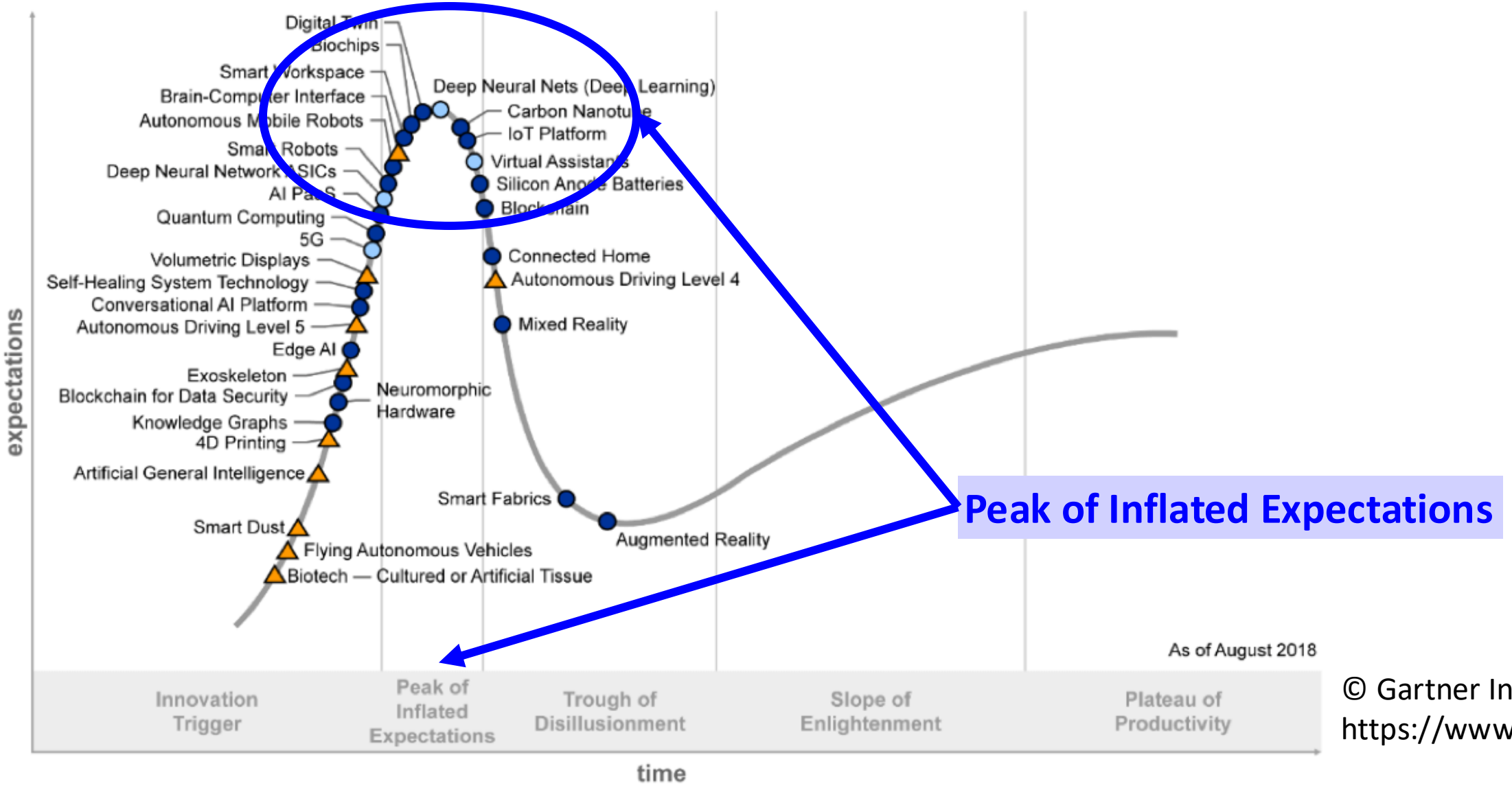
Embedded systems (ES) = **information processing systems embedded into a larger product**

Examples:

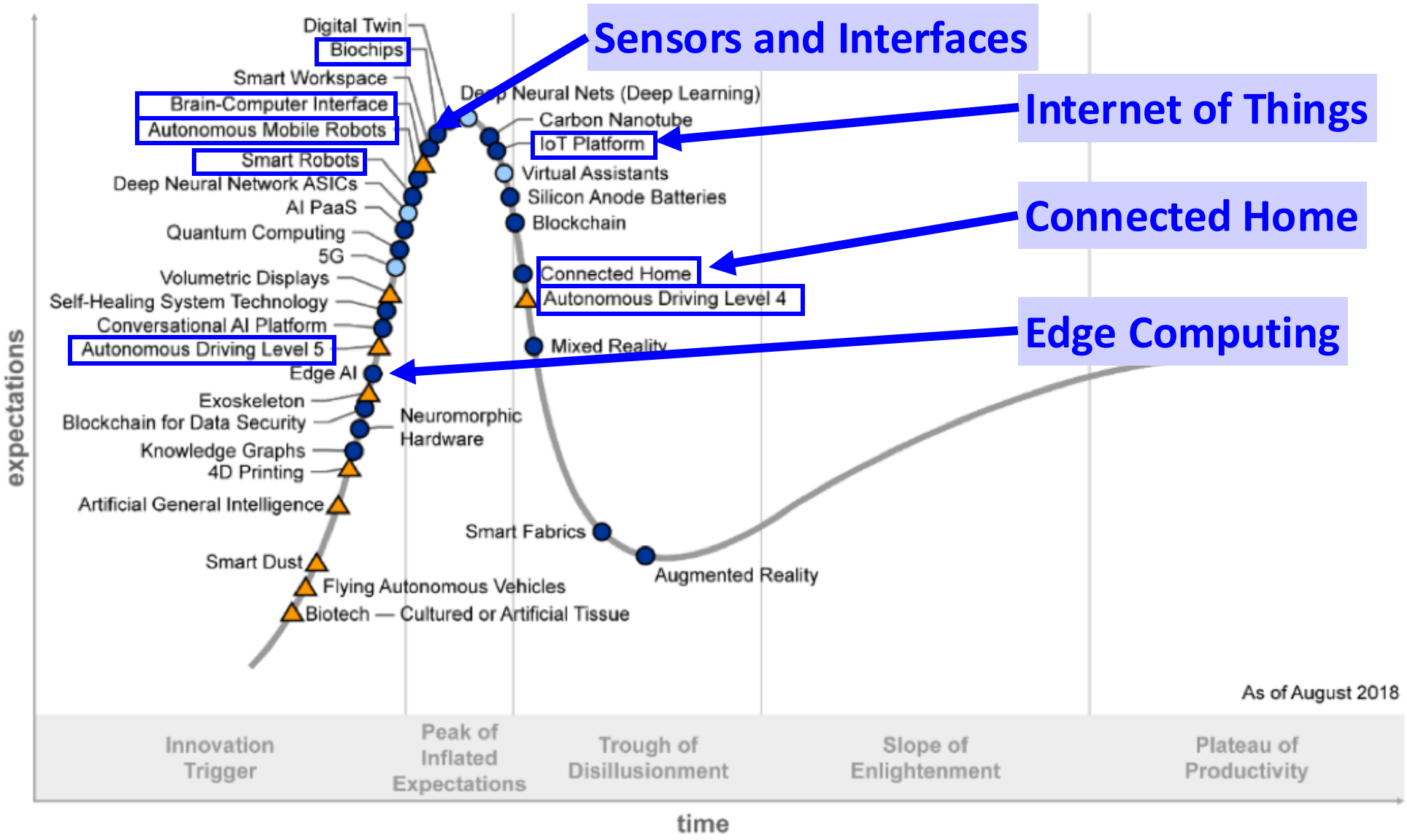


Often, the main reason for buying is not information processing

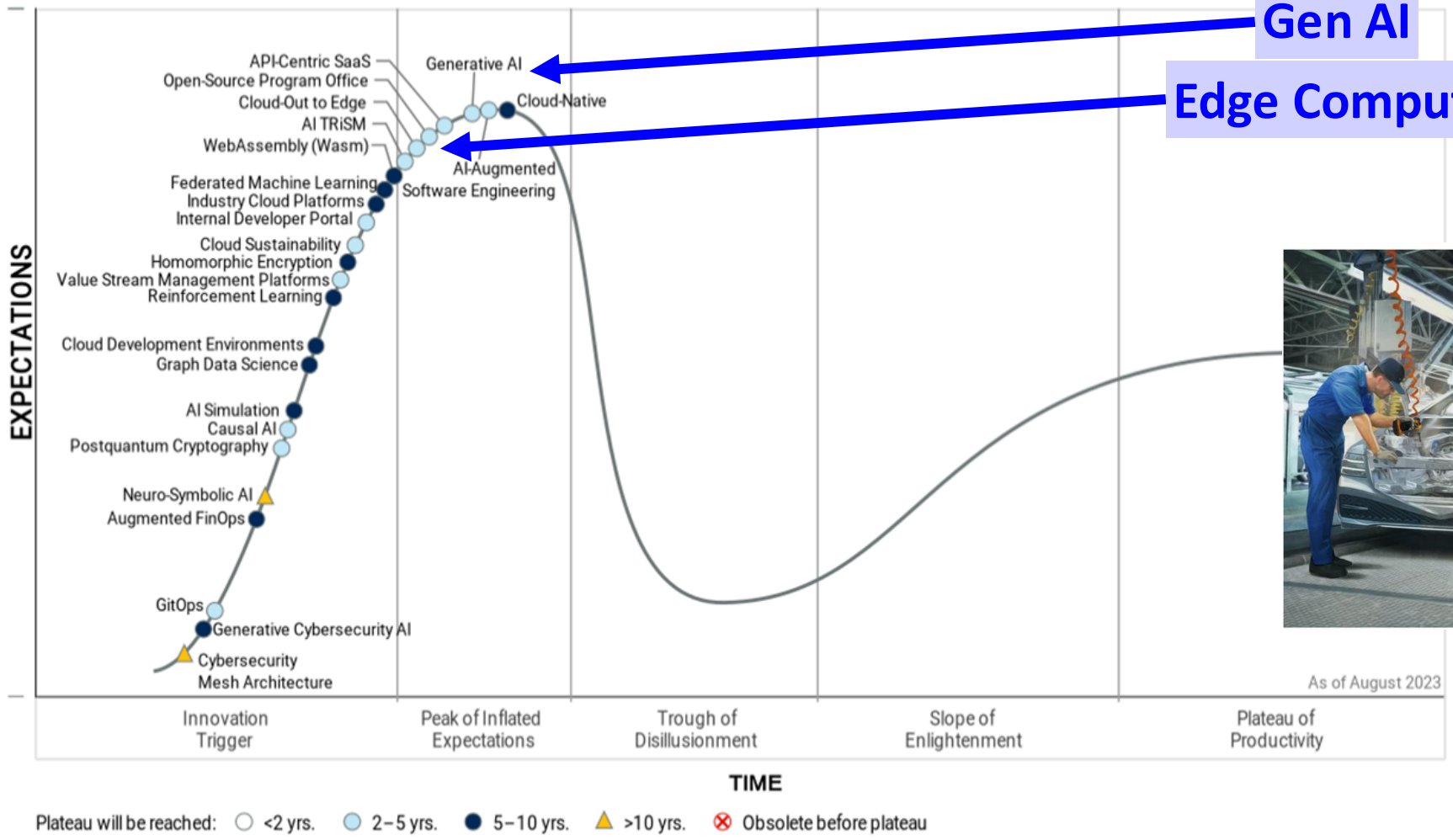
The Hype Cycle



The Hype Cycle



The Hype Cycle Continuous



Gen AI

Edge Computing

Gen AI at the Edge ☺

Cognitive Intelligence

If robots are smart enough, we won't need a fleet of programmers every time a robot needs to do a new task. The robot will understand what it sees, determine how to navigate the space, and figure out what manipulation tasks are needed to perform to successfully complete the assignment. It will also map the observed process onto hardware — its own morphology and the morphology of other robots — and consider how available infrastructure such as tables, fixtures, and tools can be used.

THE AI INSTITUTE

Gartner

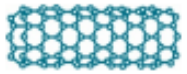
Embedded Systems

Twelve potentially economically disruptive technologies

	**Mobile Internet**	Increasingly inexpensive and capable mobile computing devices and Internet connectivity
	**Automation of knowledge work**	Intelligent software systems that can perform knowledge work tasks involving unstructured commands and subtle judgments
	**The Internet of Things**	Networks of low-cost sensors and actuators for data collection, monitoring, decision making, and process optimization
	**Cloud technology**	Use of computer hardware and software resources delivered over a network or the Internet, often as a service
	**Advanced robotics**	Increasingly capable robots with enhanced senses, dexterity, and intelligence used to automate tasks or augment humans
	**Autonomous and near-autonomous vehicles**	Vehicles that can navigate and operate with reduced or no human intervention

…

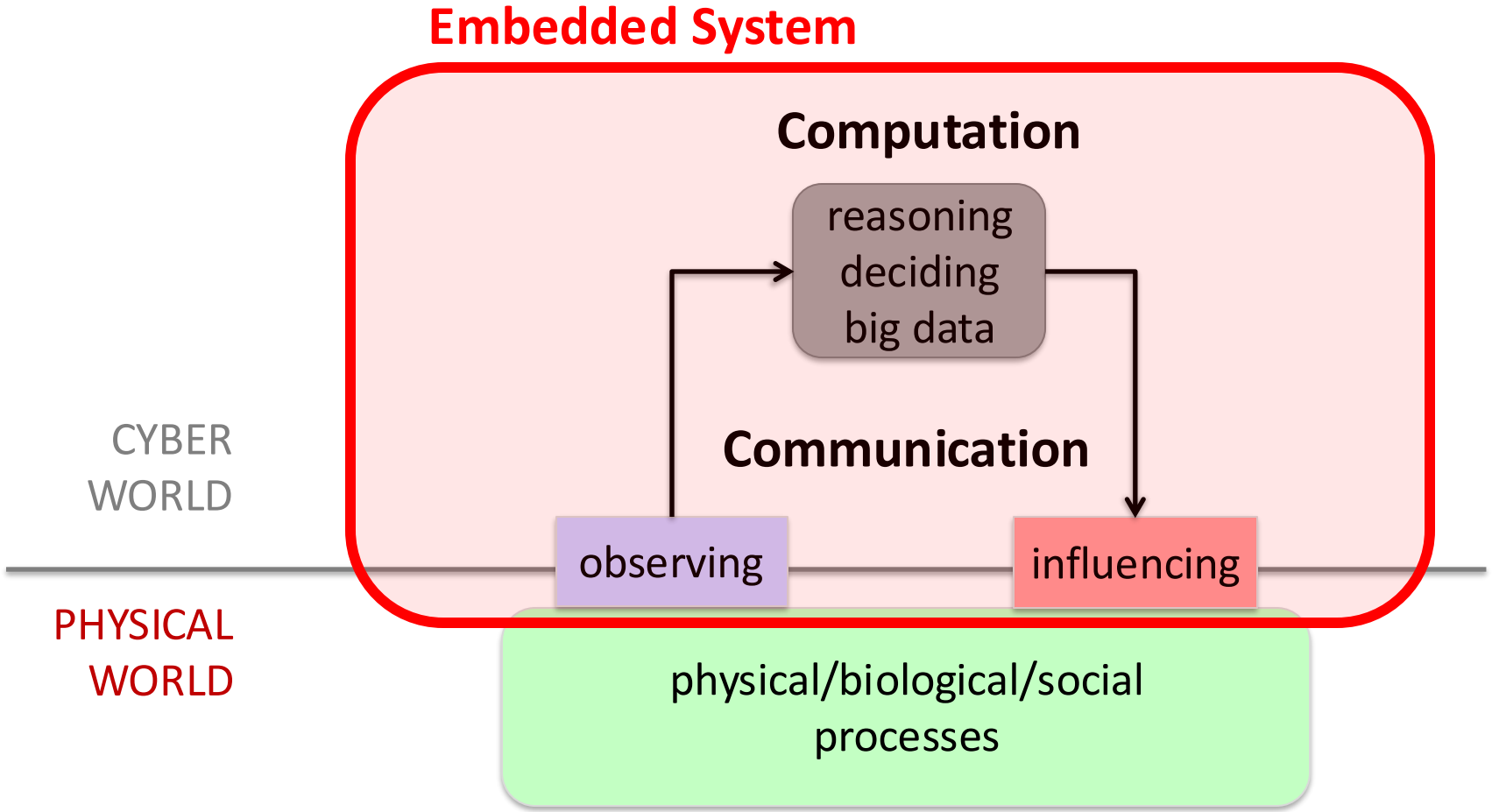
Embedded Systems are an essential component

	**Next-generation genomics**	Fast, low-cost gene sequencing, advanced big data analytics, and synthetic biology ("writing" DNA)
	**Energy storage**	Devices or systems that store energy for later use, including batteries
	**3D printing**	Additive manufacturing techniques to create objects by printing layers of material based on digital models
	**Advanced materials**	Materials designed to have superior characteristics (e.g., strength, weight, conductivity) or functionality
	**Advanced oil and gas exploration and recovery**	Exploration and recovery techniques that make extraction of unconventional oil and gas economical
	**Renewable energy**	Generation of electricity from renewable sources with reduced harmful climate impact

Manyika, James, et al. *Disruptive technologies: Advances that will transform life, business, and the global economy*. Vol. 180. San Francisco, CA: McKinsey Global Institute, 2013.

Some Example of Embedded Systems in our Lab

Embedded Systems



Use feedback to influence the dynamics of the physical world by taking smart decisions in the cyber world

Reactivity & Timing

Embedded systems are often reactive:

- Reactive systems must react to stimuli from the system environment :

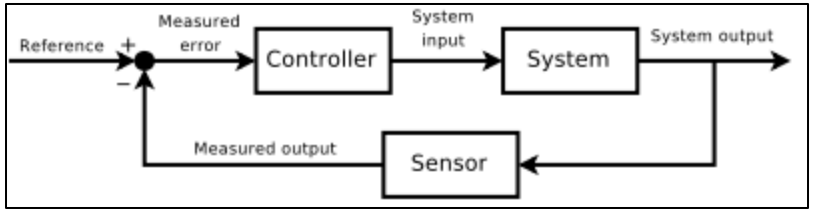
> *„A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment“ [Bergé, 1995]*

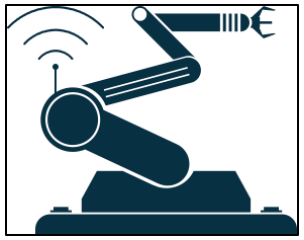
Embedded systems often must meet *real-time constraints:*

- For hard real-time systems, right answers arriving too late are wrong. All other time-constraints are called soft. A *guaranteed system response* has to be explained without statistical arguments.

> *„A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe“ [Kopetz, 1997].*

Predictability & Dependability





CPS = cyber-physical system

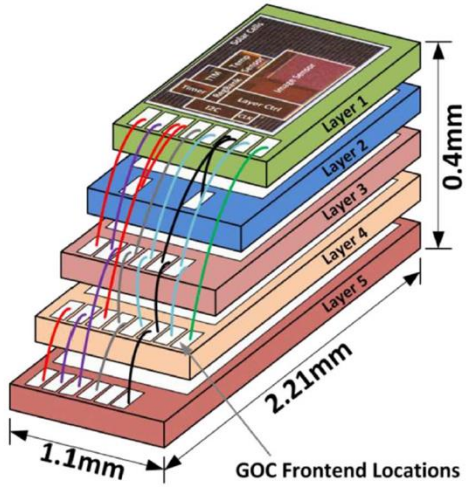
"It is essential to *predict* how a CPS is going to behave under any circumstances […] *before* it is deployed."[Maj14]

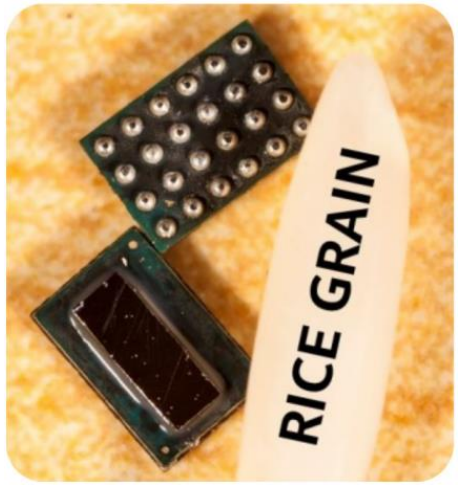
"CPS must *operate dependably*, safely, securely, efficiently and in real-time."[Raj10]

[Maj14] R. Majumdar & B. Brandenburg (2014). Foundations of Cyber-Physical Systems.
[Raj10] R. Rajkumar et al. (2010). Cyber-Physical Systems: The Next Computing Revolution.

Efficiency & Specialization

- Embedded systems must be *efficient:*
  - *Energy* efficient
  - *Code-size* and *data memory* efficient
  - *Run-time* efficient
  - *Weight* efficient
  - *Cost* efficient





Embedded Systems are often *specialized* towards a certain application or application domain:

- Knowledge about the expected behavior and the system environment at design time is exploited to *minimize resource usage* and to *maximize predictability and reliability*.

Comparison

Embedded Systems:

- Few applications that are known at design-time.
- Not programmable by end user.
- Fixed run-time requirements (additional computing power often not useful).

- Typical criteria:
  - Cost
  - Power consumption
  - Size and weight
  - Dependability
  - Worst-case speed

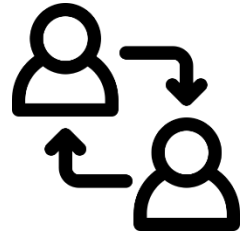
General Purpose Computing

- Broad class of applications.

- Programmable by end user.
- Faster is better.

- Typical criteria:
  - Cost
  - Power consumption
  - Average speed

Interaction: Discussion (5 minutes)



Brainstorm real-world applications of embedded systems.

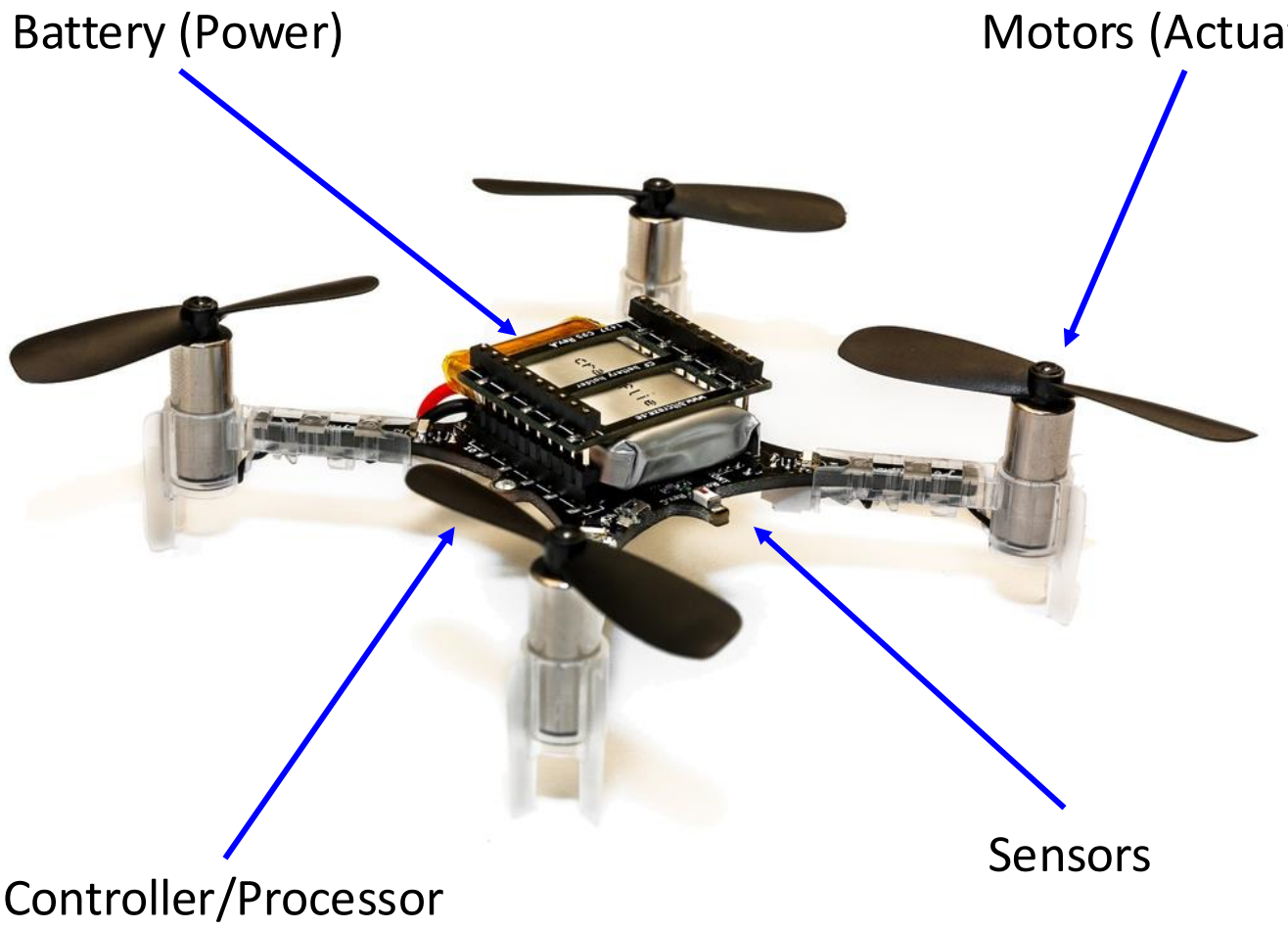
1. Recap the key concepts from the introduction and highlight the importance of real-world applications (1 minute).

2. Brainstorm quickly and write down one real-world application where embedded systems can be applied. Select one sensor and one actuator that might be used in that application (4 minutes).

3. Place your application in one of the fields in the clicker question:
    1. *To which field does your application belong?*
    2. *Which sensors are needed?*

Embedded System: An Example

System Overview

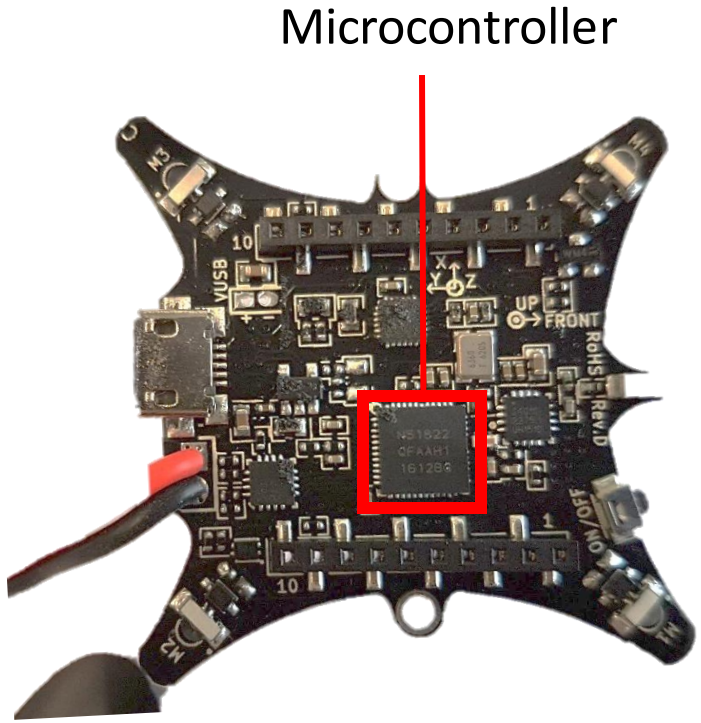


What is a Microcontroller?

- A microcontroller is a **small processor** with a core, memory and integrated I/O peripherals such as timers, analog-to-digital converters, and serial communications [1].

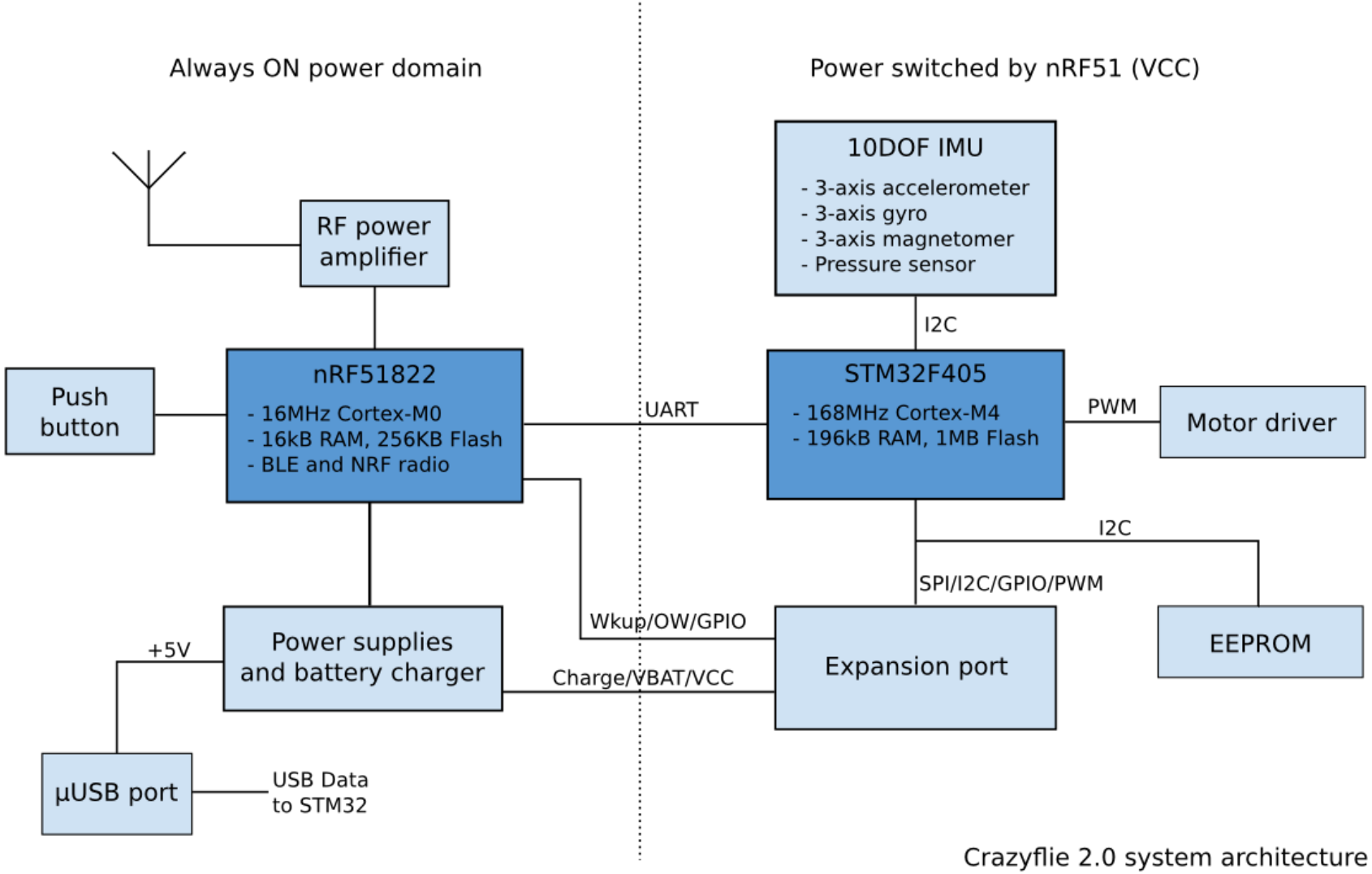
- Features of a microcontroller (MCU):
  - Self Contained (CPU, Memory, I/O)
  - Application or Task Specific (Not a general-purpose computer)
  - Appropriately scaled for the job
  - Small power consumption
  - Low costs ($0.50 to $5.00.)

- Microcontrollers are everywhere, many systems contain even multiple microcontrollers.



[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 56

High-Level Block Diagram



Crazyflie 2.0 system architecture

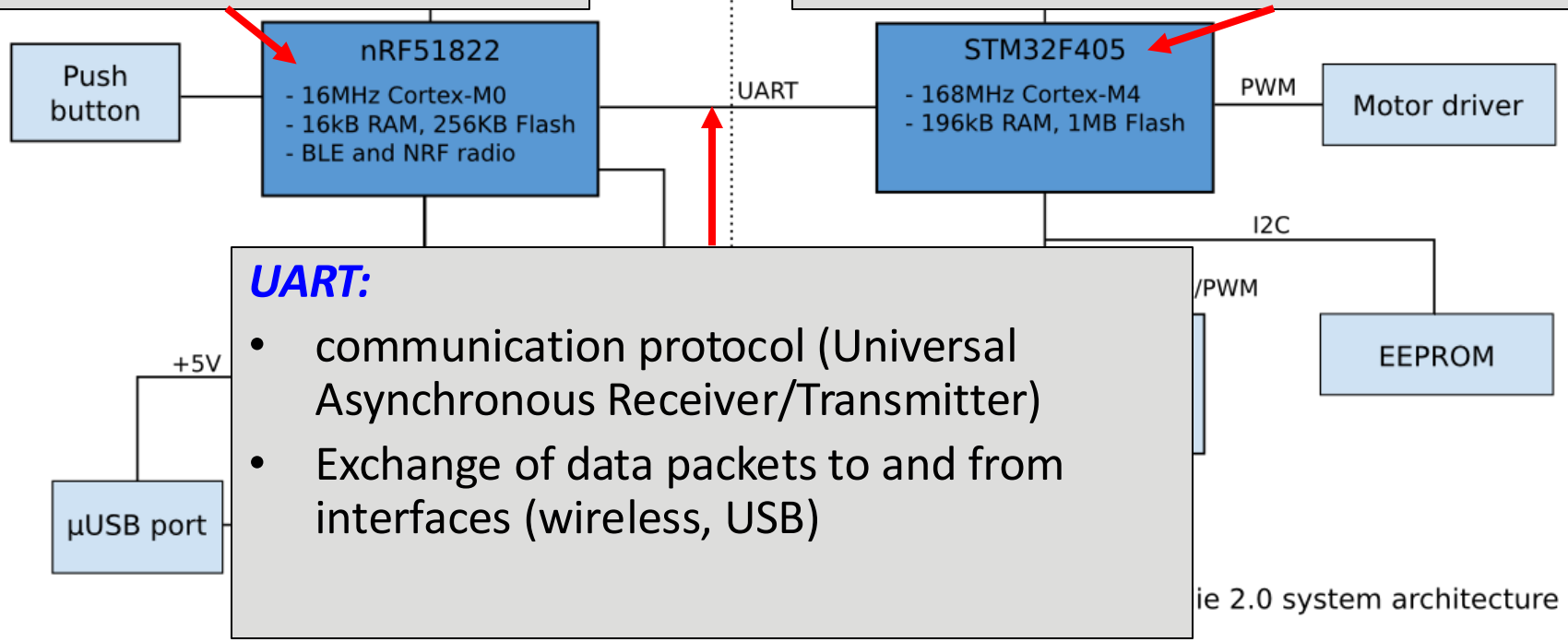
High-Level Block Diagram

Low Power CPU:

- Enabling power to the rest of the system
- Battery charging and voltage measurement
- Wireless radio (boot and operate)
- Detect and check expansion boards

Higher Performance CPU:

- Sensor reading and motor control
- Flight control
- Telemetry (including the battery voltage)
- Additional user development
- USB connection



UART:

- communication protocol (Universal Asynchronous Receiver/Transmitter)
- Exchange of data packets to and from interfaces (wireless, USB)

ie 2.0 system architecture

High-Level Block Diagram

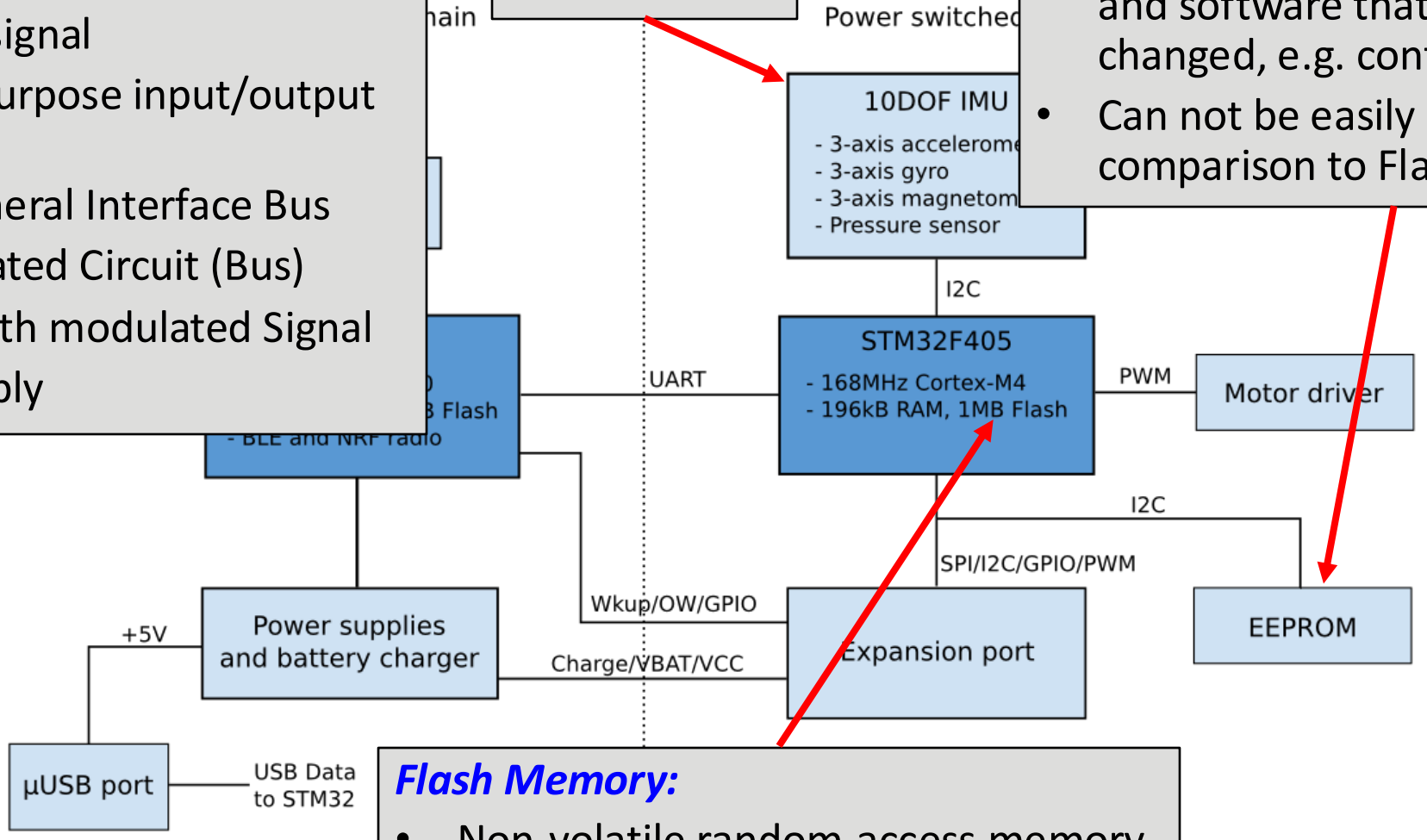
Acronyms:

- Wkup: Wakeup signal
- GPIO: General-purpose input/output signal
- SPI: Serial Peripheral Interface Bus
- I2C: Inter-Integrated Circuit (Bus)
- PWM: Pulse-width modulated Signal
- VCC: power-supply

Sensor Board:

EEPROM:

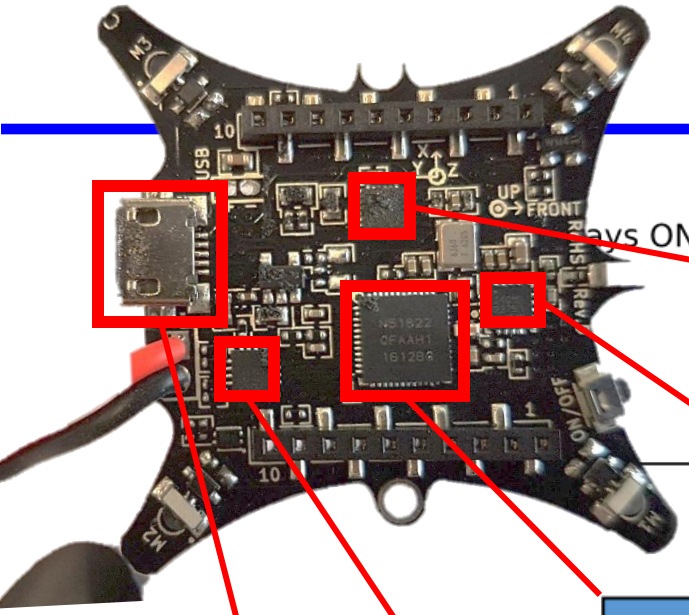
- Electrically erasable programmable read-only memory
- Used for firmware (part of data and software that usually is not changed, e.g. configuration data)
- Can not be easily overwritten in comparison to Flash



Flash Memory:

- Non-volatile random-access memory for program and data

High-Level Physical View



ays ON power domain

Power switched by nRF51 (VCC)

RF power amplifier

10DOF IMU
- 3-axis accelerometer
- 3-axis gyro
- 3-axis magnetomer
- Pressure sensor

I2C

Push button

nRF51822
- 16MHz Cortex-M0
- 16kB RAM, 256KB Flash
- BLE and NRF radio

UART

STM32F405
- 168MHz Cortex-M4
- 196kB RAM, 1MB Flash

PWM

Motor driver

I2C

SPI/I2C/GPIO/PWM

Wkup/OW/GPIO

Power supplies and battery charger

+5V

Charge/VBAT/VCC

Expansion port

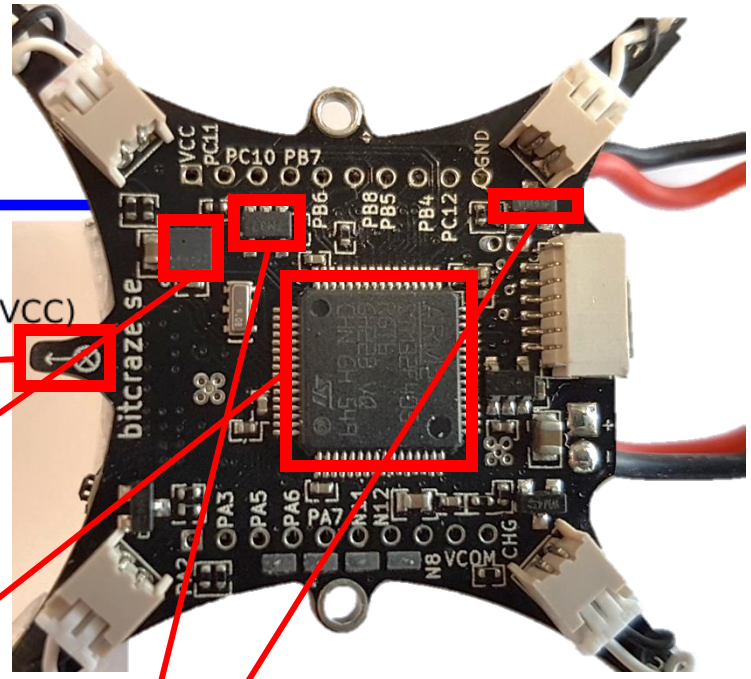
EEPROM

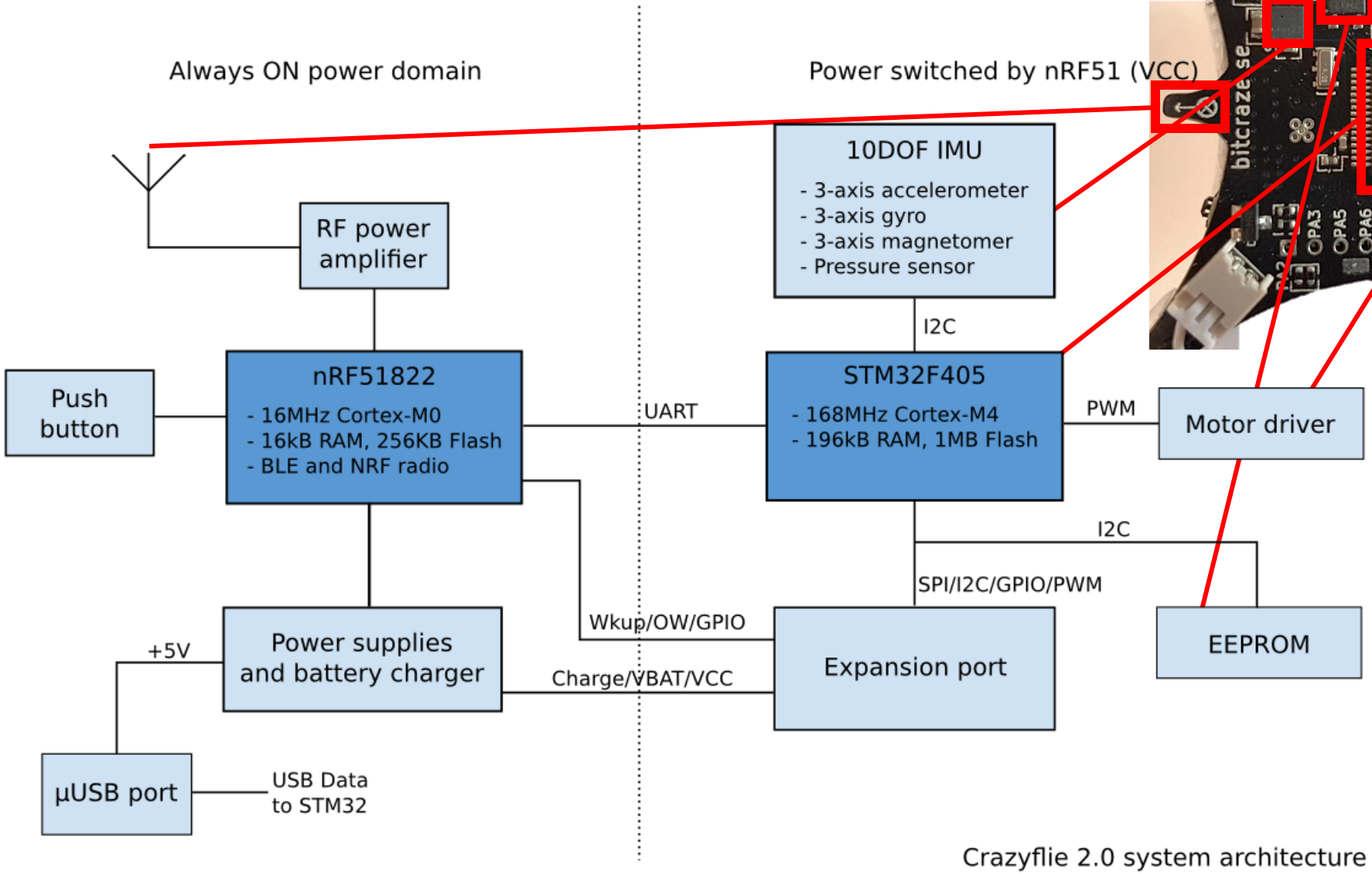
µUSB port

USB Data to STM32

Crazyflie 2.0 system architecture

High-Level Physical View





Crazyflie 2.0 system architecture

Embedded Systems - Basics

Embedded Systems in a Picture – An Overview

Hardware Perspective

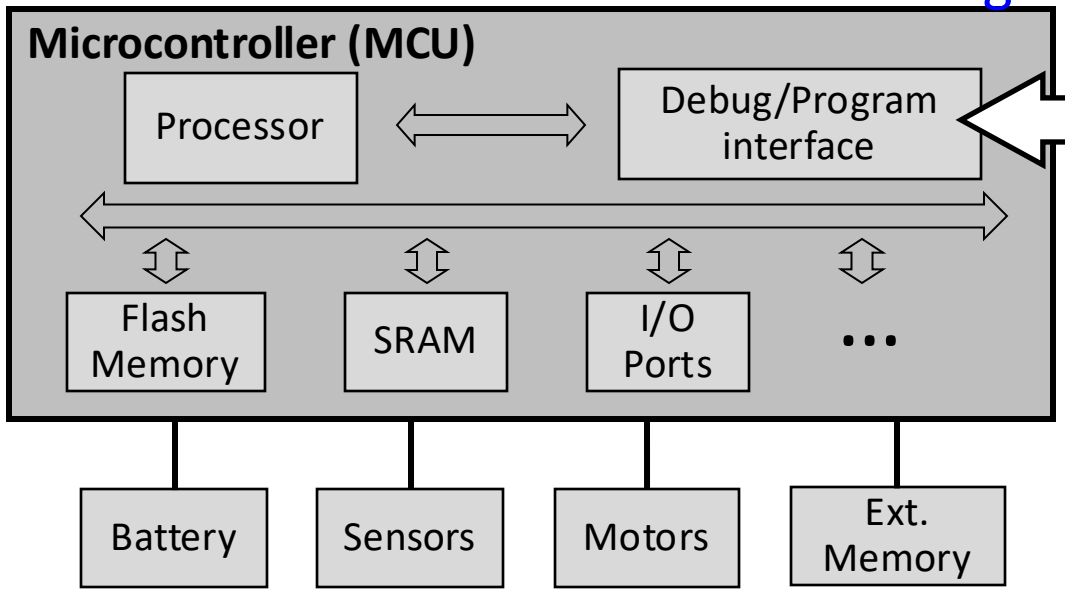
- Defining the physical Connection (Schema)
- Designing the actual electronics

System Overview

Firmware/Software Perspective

- programming functionality
- interfacing HW w. drivers

0. Design HW



3. Using SWD/JTAG to flash the processor with your code

1. Write Code

code.c

```c
// High-level language
// C-source code
int main (void){
    int total = 0;
    for (int i = 0; i < 10; i++) {
        total += i;
    }
    while(1); // Infinite loop
}
```

Compiler

code.asm

```
// Low-level Language
// Target Assembly Code
        MOVS r1, #0
        MOVS r2, #0
        B    check
loop:   ADD  r1, r1, r0
        ADDS r0, r0, #1
check:  CMP  r0, #10
        BLT  loop
self:   B    self
```

Assembler

2. Compile with GCC Compiler

code.hex

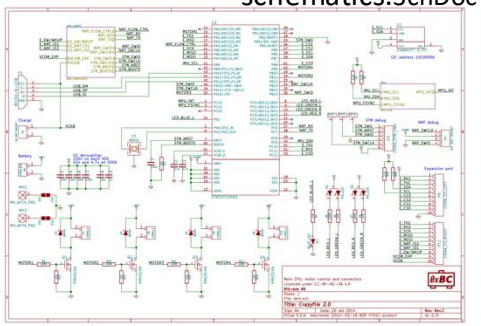
```
0x2100
0x2000
0xE001
0x4401
0x1C40
0x280A
0xDBFB
0x8F00
0xE7FE
```

Target Machine Code

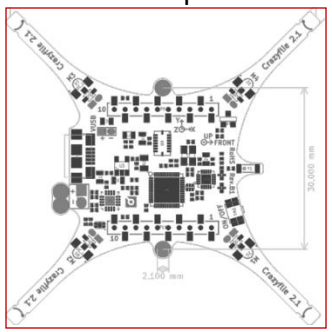
code.bin

```
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
```

schematics.SchDoc

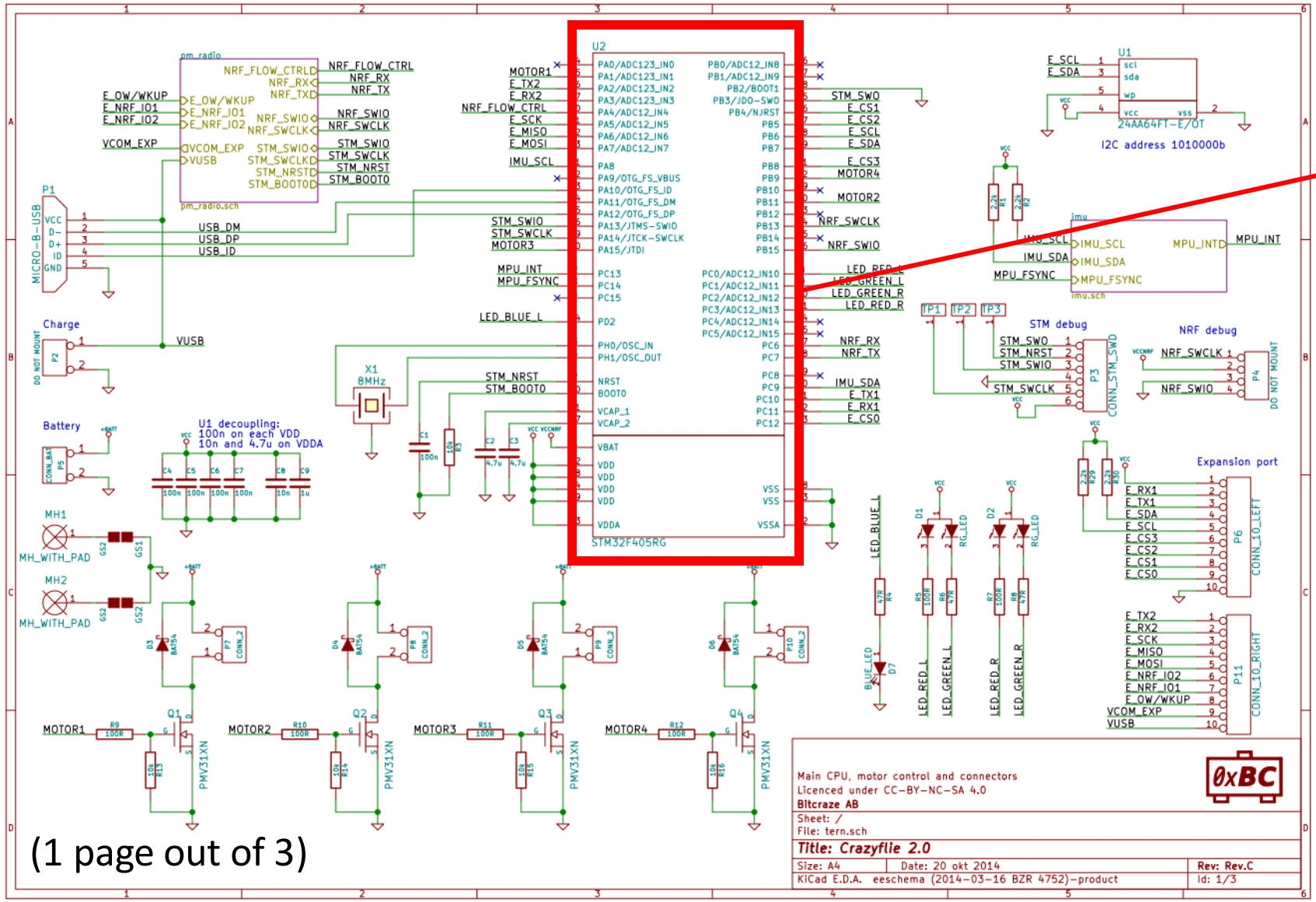


pcb.PcbDoc

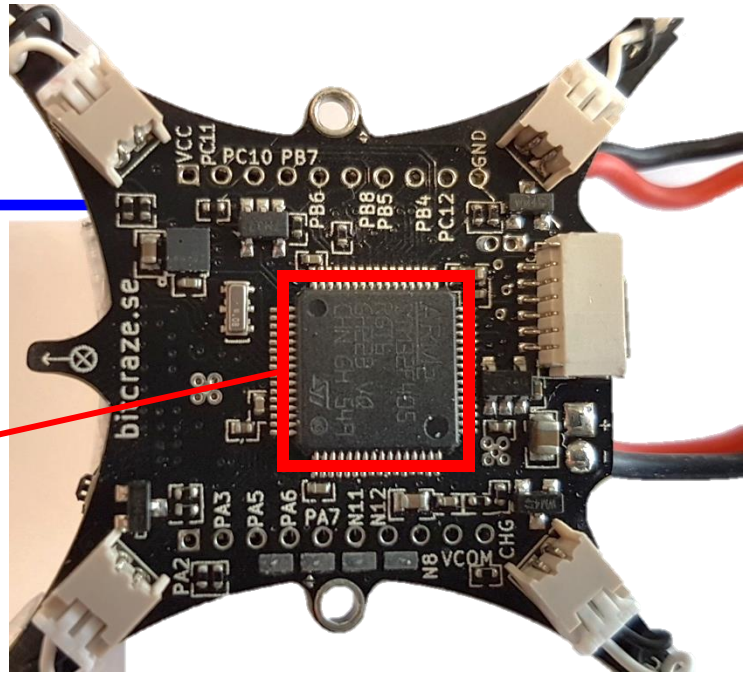


Schematics

Low-Level Schematic Diagram View



(1 page out of 3)



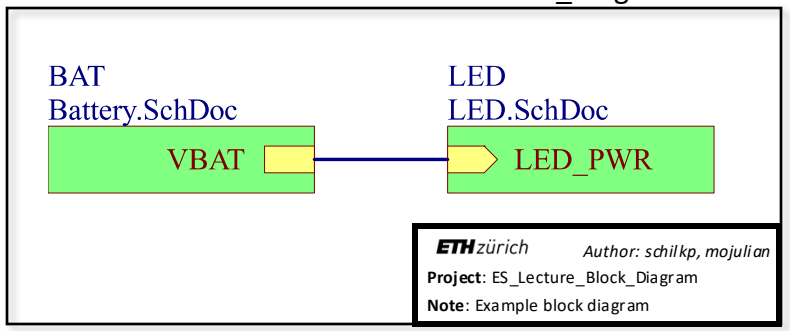
Takes a bit of practice to understand…

How to Read the Hierarchical Schematic?

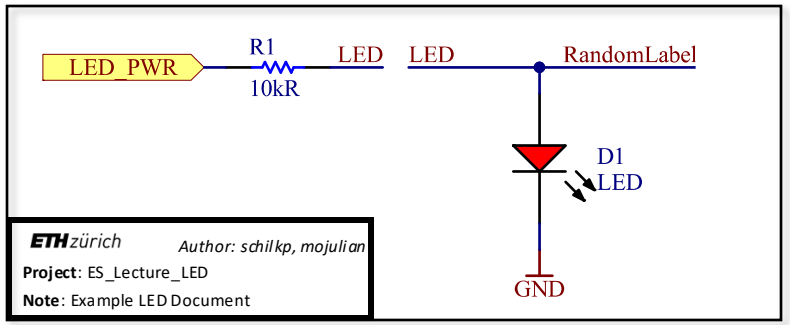
- Complicated schematics are drawn in a hierarchical manner.
  - Circuit is split into separate sheets. Each sheet contains related circuitry, i.e.: Power, MCU, Sensors,…
  - These sub-circuits are represented as blocks in other schematic sheets.

- Think of it like functions while programming:
  - You design the sub-circuit once with defined inputs and output ports. (*"implementing the function"*).
  - You reuse these sub-circuits multiple times in other schematics (*"calling the function"*).

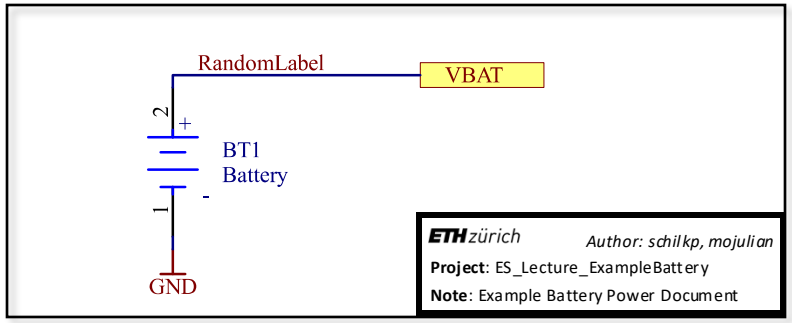
Block_Diagram.SchDoc



LED.SchDoc

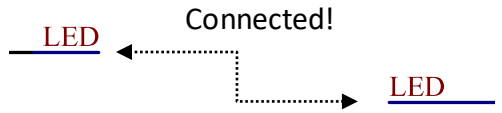


Battery.SchDoc

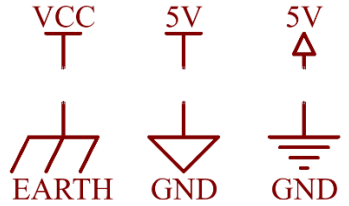


How to Read the Hierarchical Schematic?

- **Local Net Labels:**
  - Avoids ”Kabelsalat”/ “Ratsnest”
  - Matched names connect within current sheet.

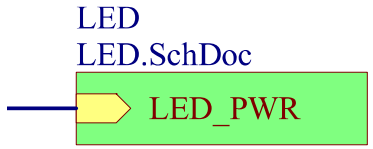


- **Global Power Labels:**
  - Matched symbols connects globally across all sheets.

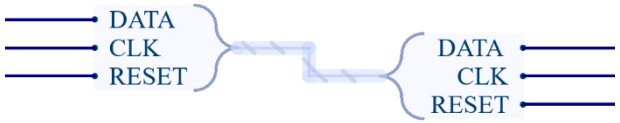


- **Sheet Ports:**
  - Define inputs & outputs of a re-useable schematic block

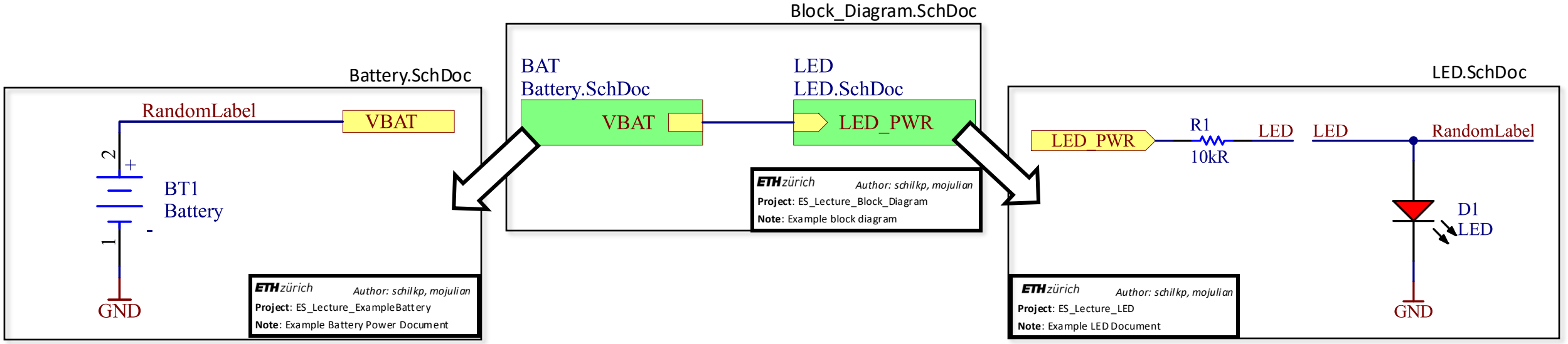
- **Sheet Block/Symbol:**
  - Represents an instance of a schematic sheet defined elsewhere.



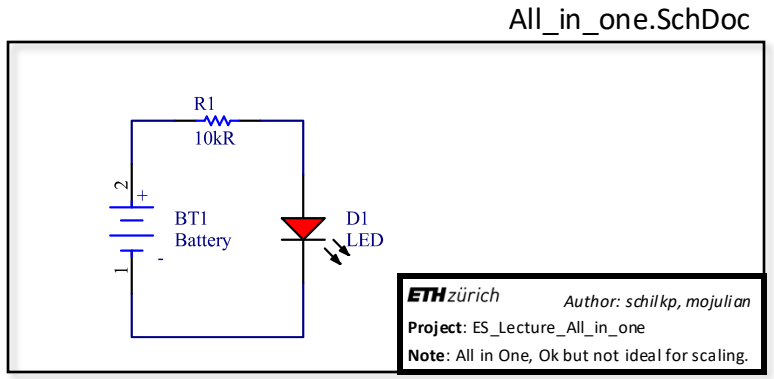
- **Harness/Bus**
  - Groups related signals into one connection.



How to Read the Hierarchical Schematic?

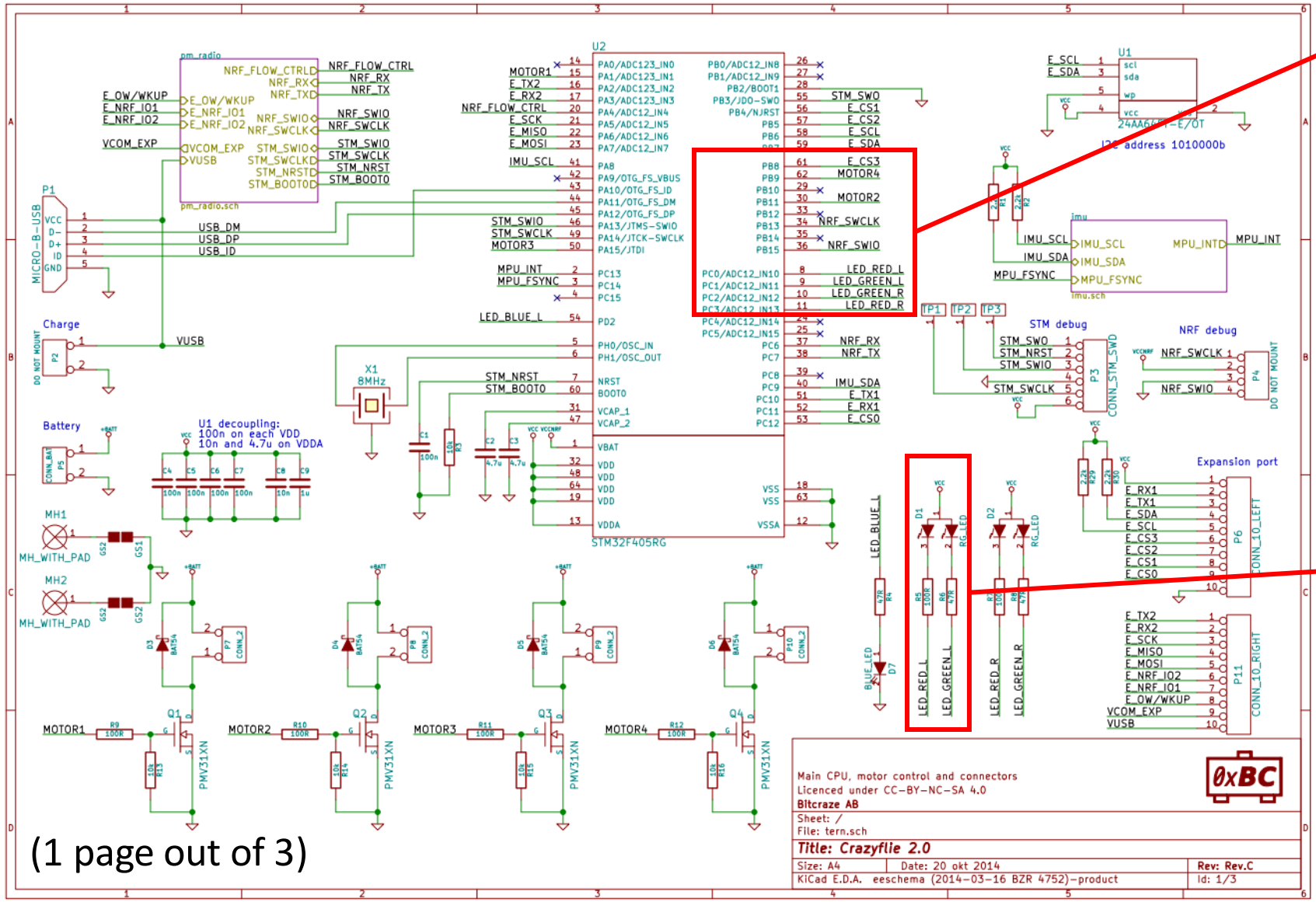


- The 3 schematics above are equivalent to the one shown below.
- This simple example is somewhat contrived, but in a complex system such organization is very helpful.

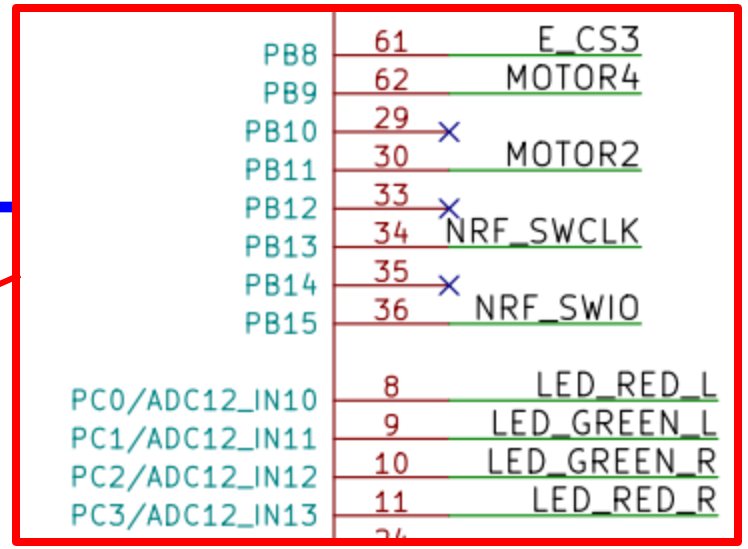


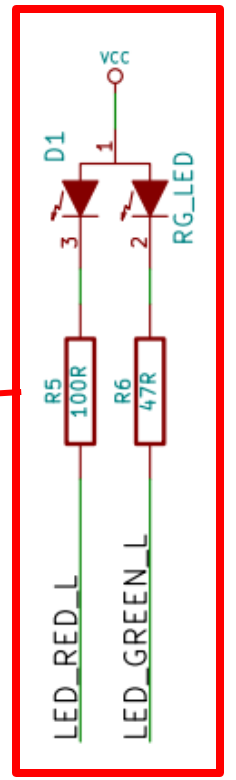
- Note that the two "RandomLabel" labels are **NOT** connected!
- They are LOCAL labels.

Net Labels in Action



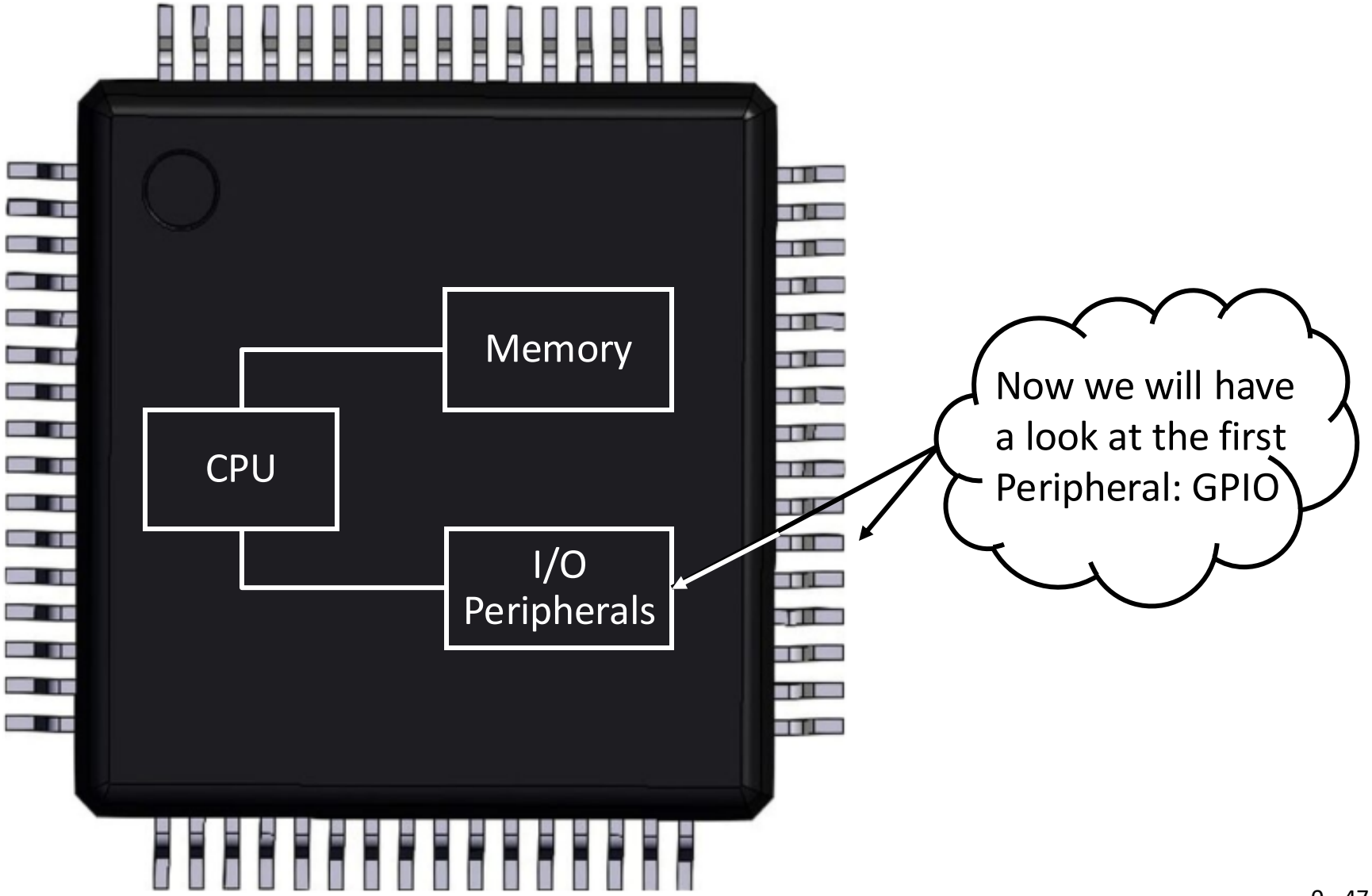
(1 page out of 3)





LEDs

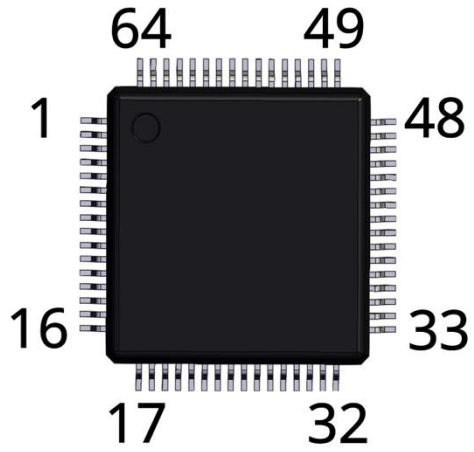
Brief Glimpse Inside the Microcontroller



GPIO (General-Purpose Input-Output)

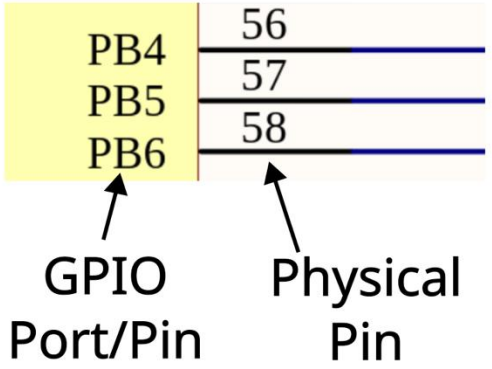
How does the Microcontroller Interact with the World?

- **Feature a number of General-Purpose Input/Output (GPIO) pins.**

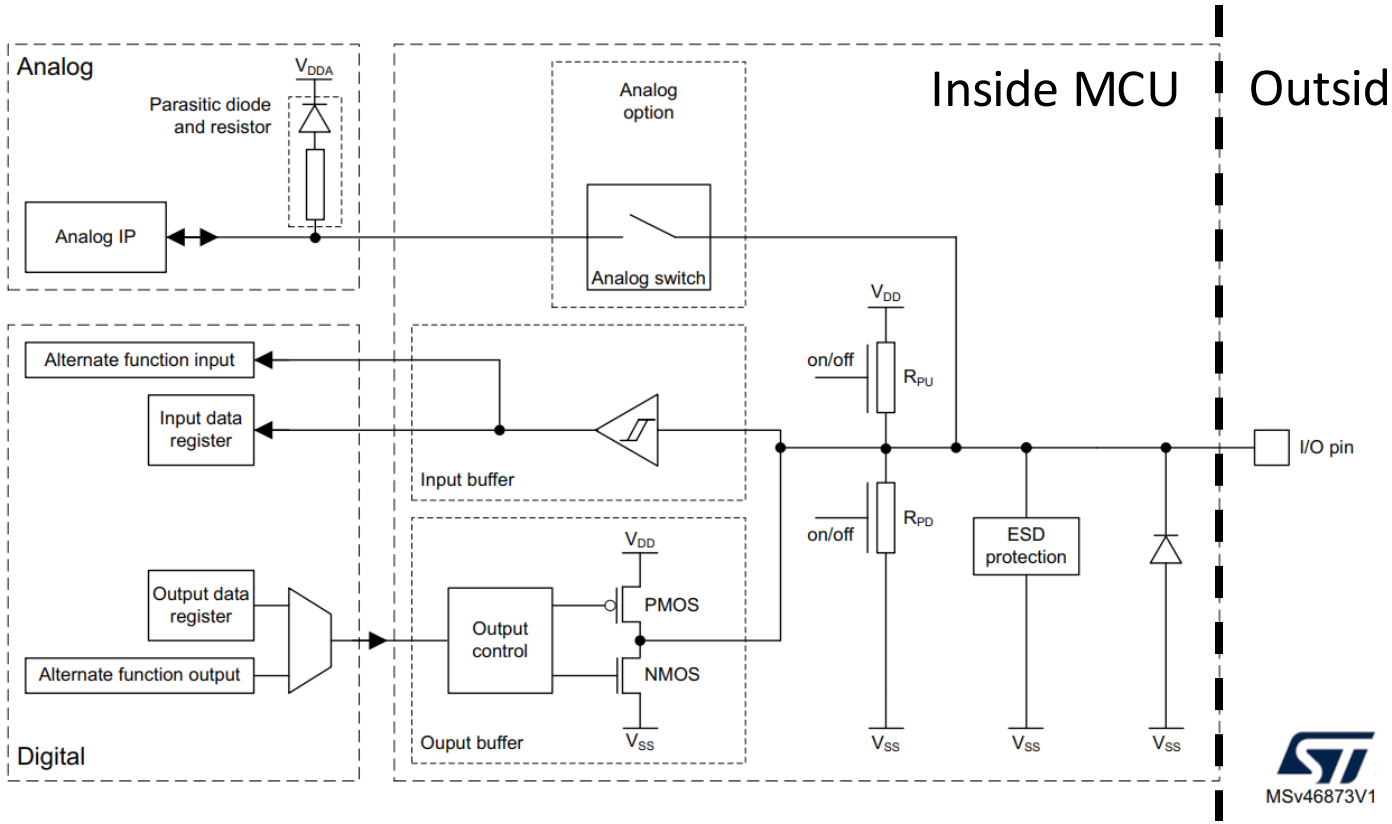


- **They are grouped into ports:**
  - Each port is identified by a letter (A-H).
  - Each port features up to 16 GPIO pins.

- **They can be configured in a variety of ways:**
  - Input/Output/Analog/Alternate (Peripheral)
  - Push-Pull/Open Drain
  - Internal Pull-Ups/Pull-Downs

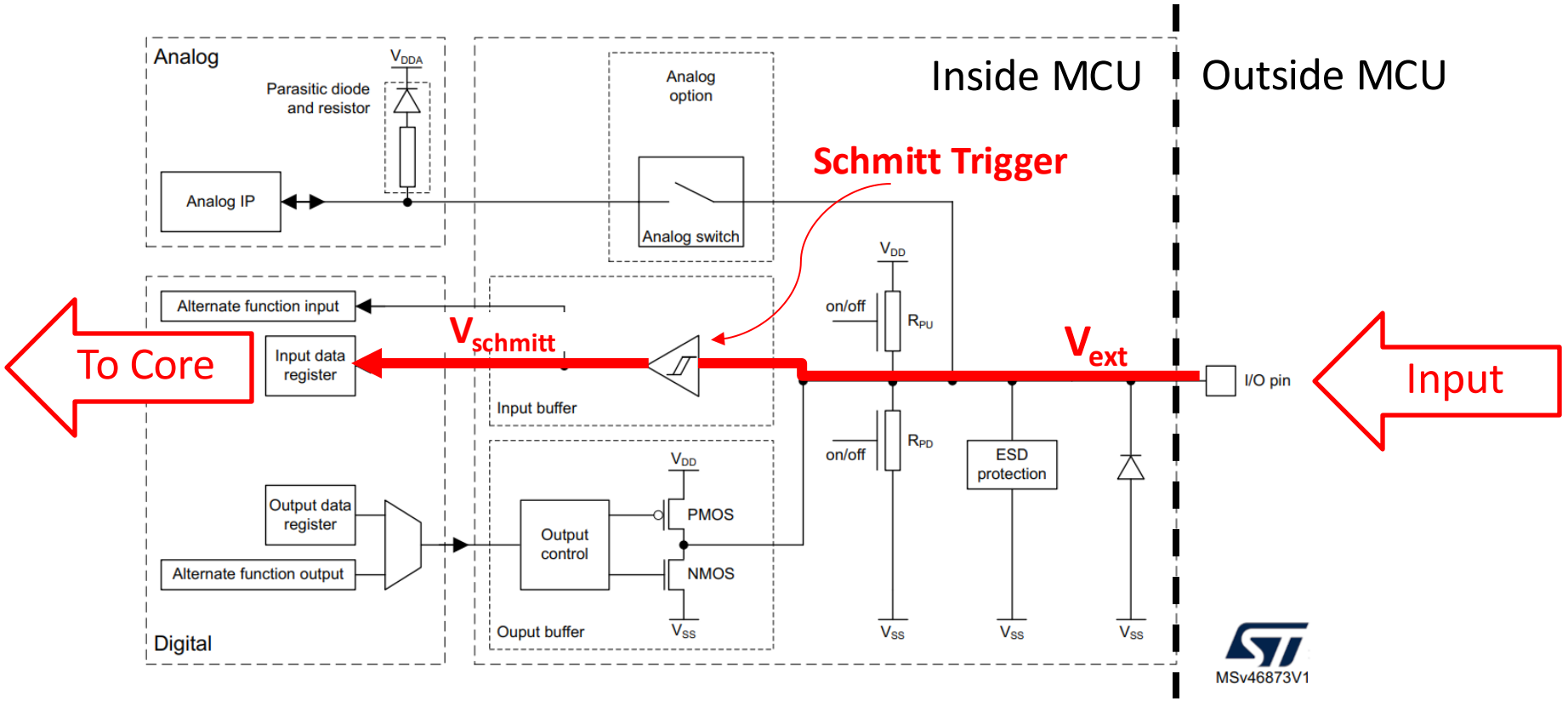


GPIO Pin in Detail



Each pin has many features and components:
We will explore them step by step.

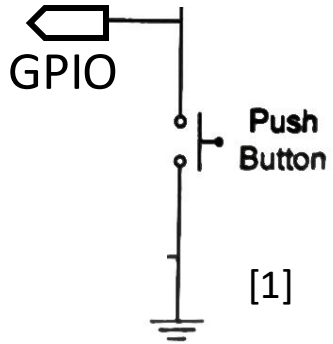
GPIO: Inputs



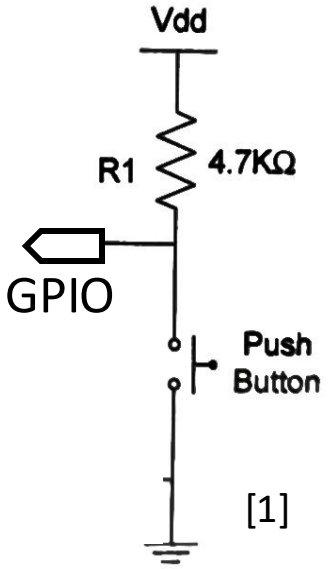
- *Digital inputs* detect whether an external voltage signal is higher or lower than a predetermined threshold.

GPIO: Floating Inputs

- A digital input that is not connected to a voltage source is referred to as floating.
- Floating pins have an undefined state and will randomly read as 1 or 0. This must be avoided!
- Pull-up or Pull-down resistors are used to provide a default voltage when the pin would otherwise be floating.



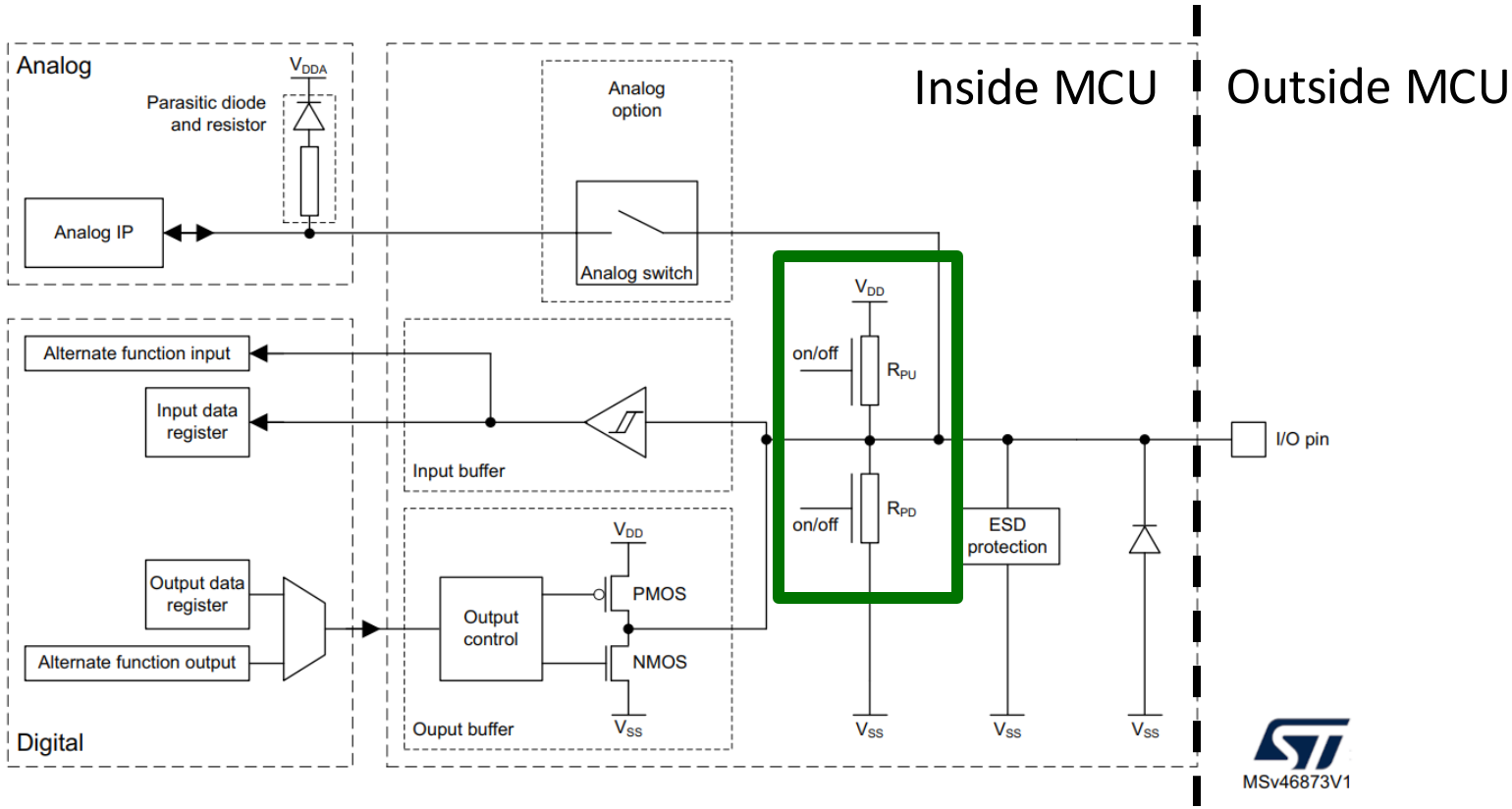
- **No Pull-Up.**
- Pressed: 0V
- Else: **Floating**



- **Pull-Up.**
- Pressed: 0V
- Else: Vdd

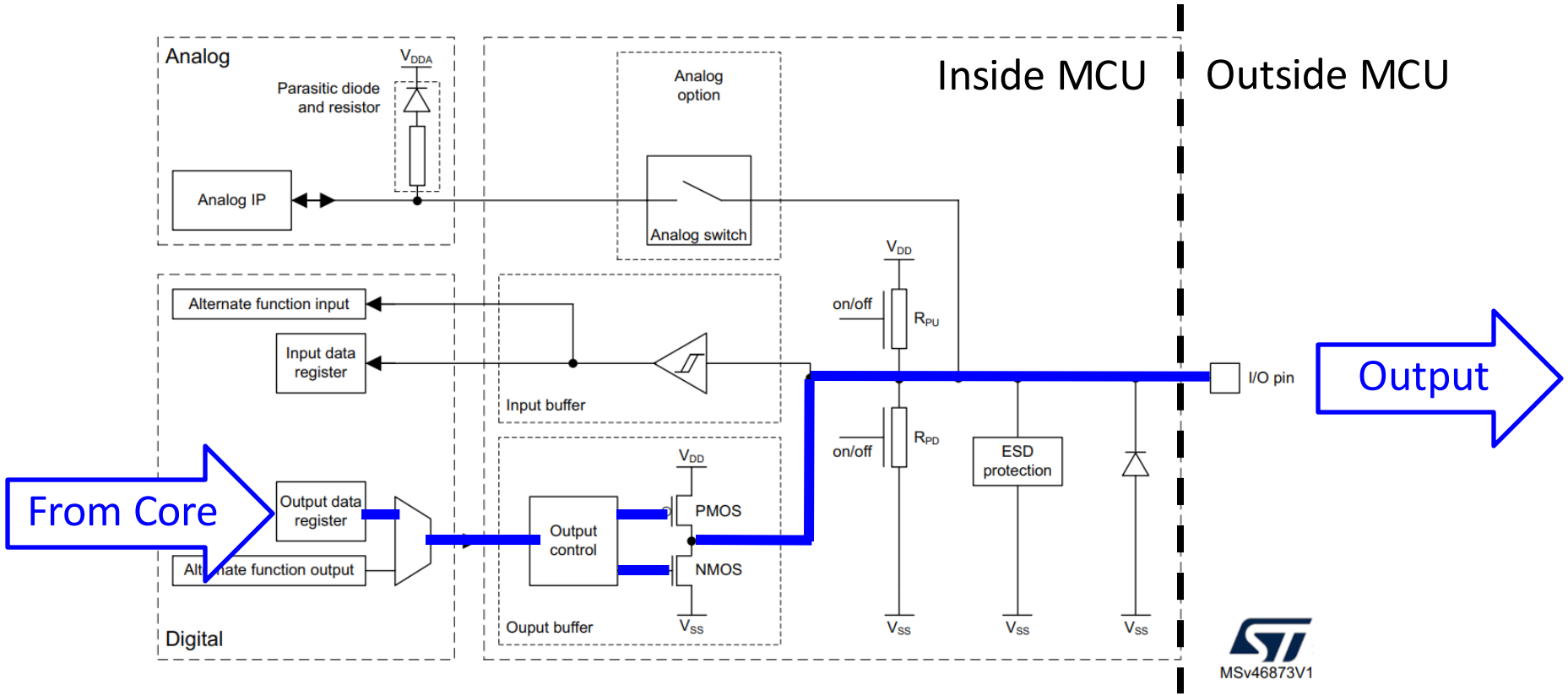
[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 362

GPIO: Pull-Up and Pull-Down Resistors



- Pull-Up or Pull-Down Resistors can be soldered to the PCB, or internally enabled by software.

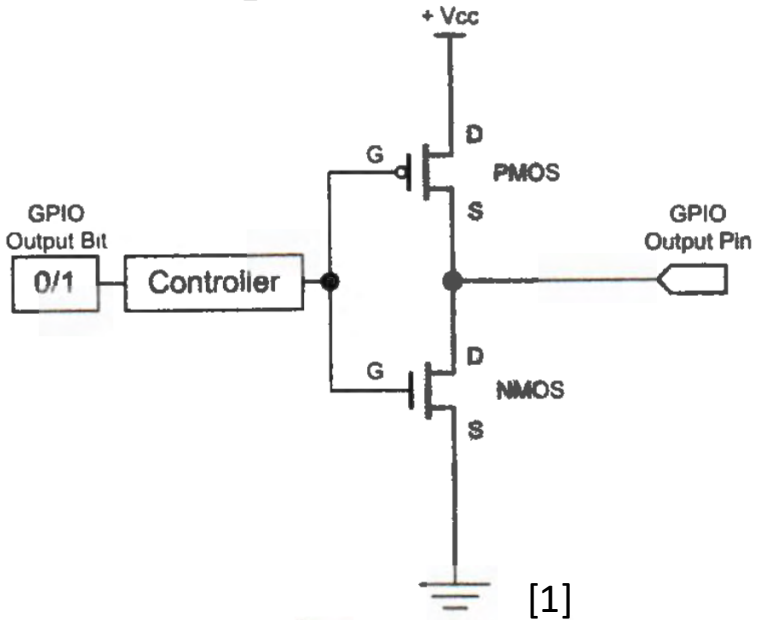
GPIO: Outputs



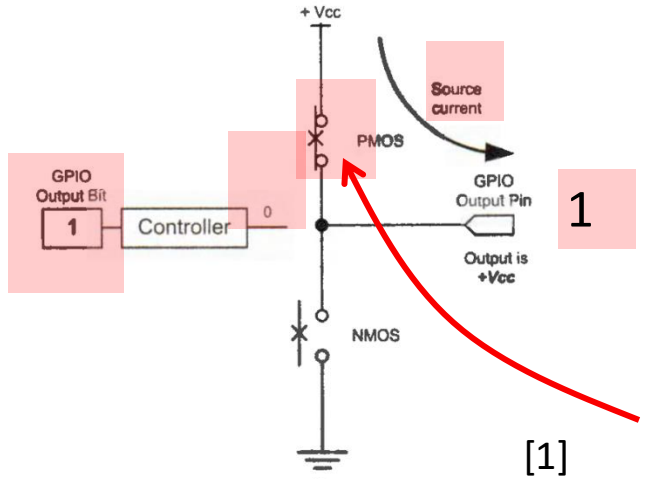
- *Digital outputs* control the voltage on the pin.

GPIO Operating Modes – Digital Output

- For digital outputs, each GPIO can be configured as either push-pull or open-drain.
- *Push-pull* can absorb or supply current:

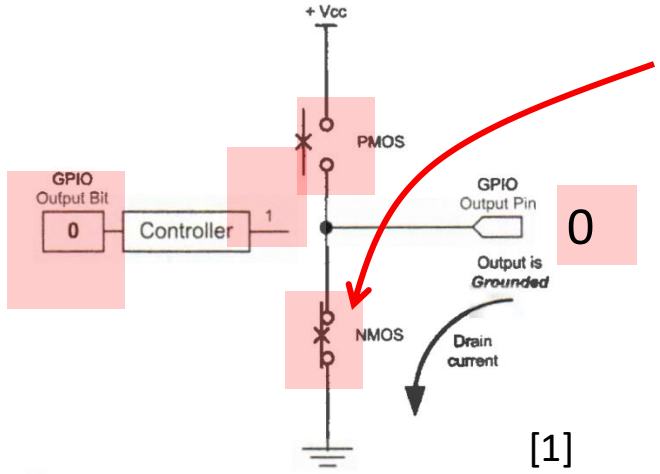


[1]



[1]

max. current is limited by the MOSFET's channel size!

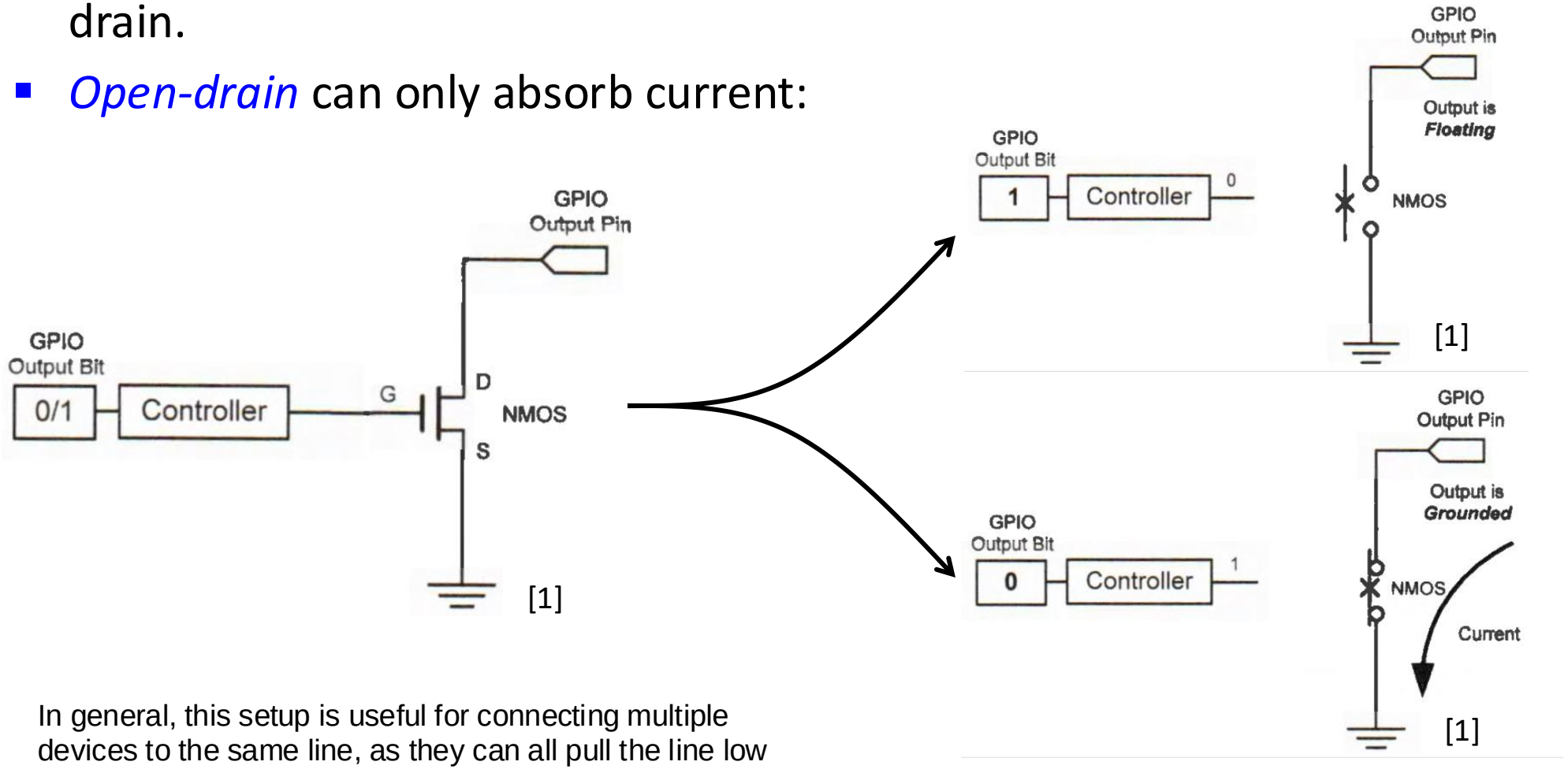


[1]

[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 360

GPIO Operating Modes – Digital Output

- For digital outputs, each GPIO can be configured as either push-pull or open-drain.
- *Open-drain* can only absorb current:



In general, this setup is useful for connecting multiple devices to the same line, as they can all pull the line low without conflicting with each other.

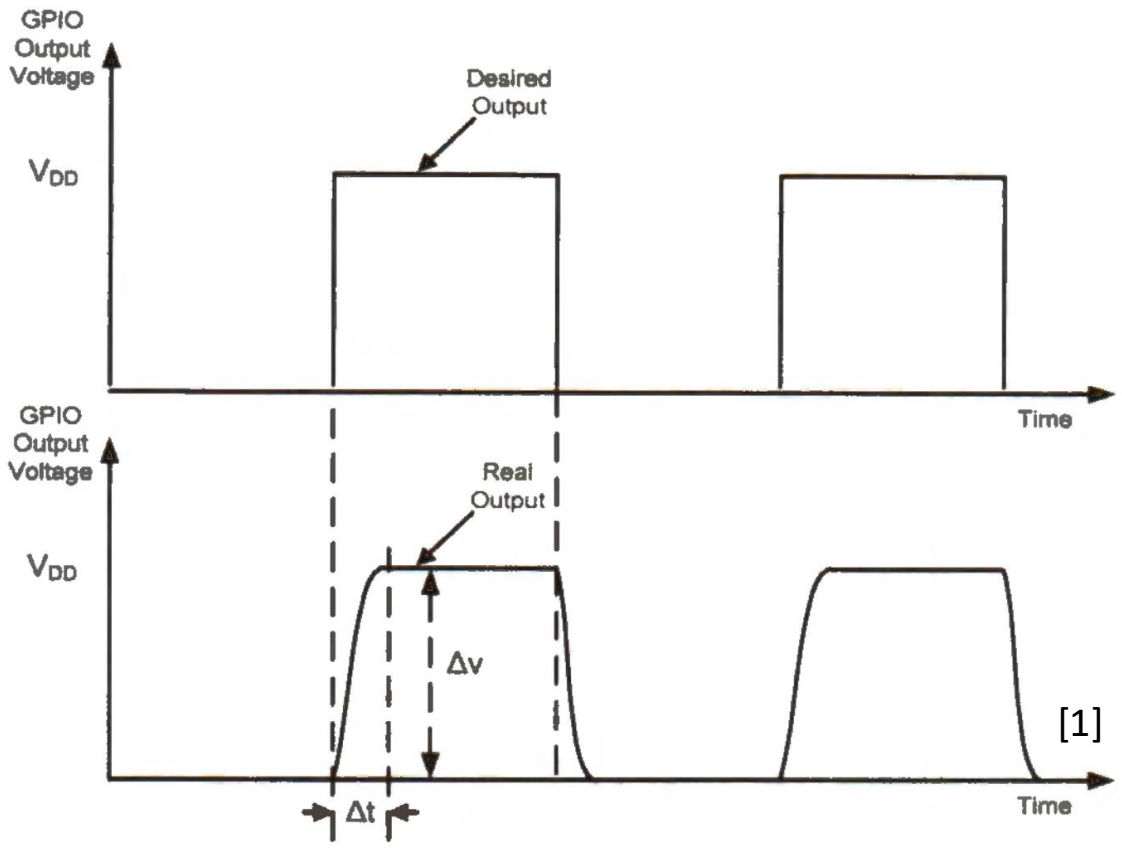
[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 361

GPIO Operating Modes – Digital Output Speed

The slew rate of a GPIO pin is the speed of change of its output voltage per unit of time, as defined as follows:

$$\text{slew rate} = \frac{\triangle v}{\triangle t}$$

- High slew rate: fast signals but causes higher electromagnetic interference (EMI) and consumes more power
- Low slew rate: save power and reduce noise

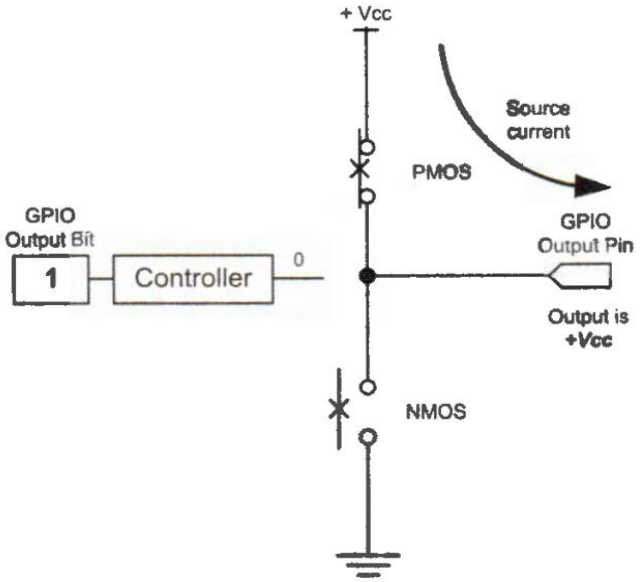


[1]

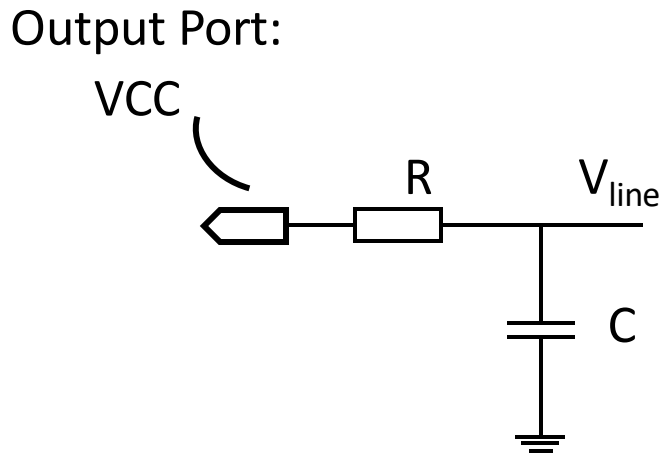
[1] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, page 362

GPIO Operating Modes – Digital Output Speed

- The system can be modeled as an RC equivalent circuit.
- The resistor is given by the strength of the transistors and minor parasitics*.
- The capacitor is given by parasitics.
- RC-constant formed by parasitic elements determines the rising and falling time!
- GPIO slew time can be adjusted in software.



=

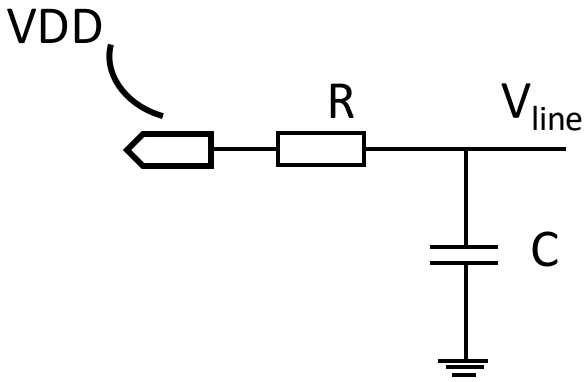


* "Parasitics" model the non-ideal real-world behavior of an electrical circuit (e.g. a real inductor can be modeled by an ideal inductor in series with a resistor and a capacitance in parallel).

Quick Review of RC circuit

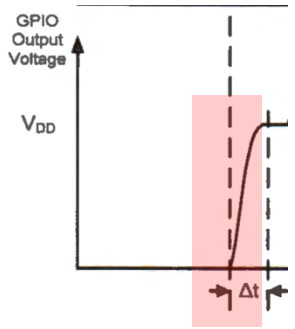
- The **Input** or **Output** path of a GPIO Port/Pin, has always parasitics due to the MCU manufacturing and the PCB design.
- These constants have to be considered.

Charing (initial $V_{line} = 0V$)

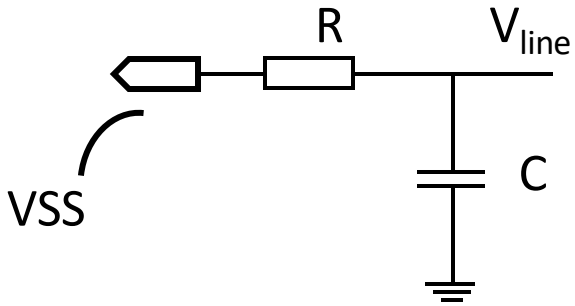


The capacitor gets charged with:

$$V_{line} = V_{DD}(1 - e^{-\frac{t}{RC}})$$

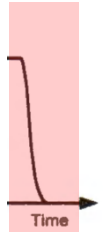


Discharging (initial $V_{line} = V_{DD}$)



The capacitor gets discharged with:

$$V_{line} = V_{DD}e^{-\frac{t}{RC}}$$



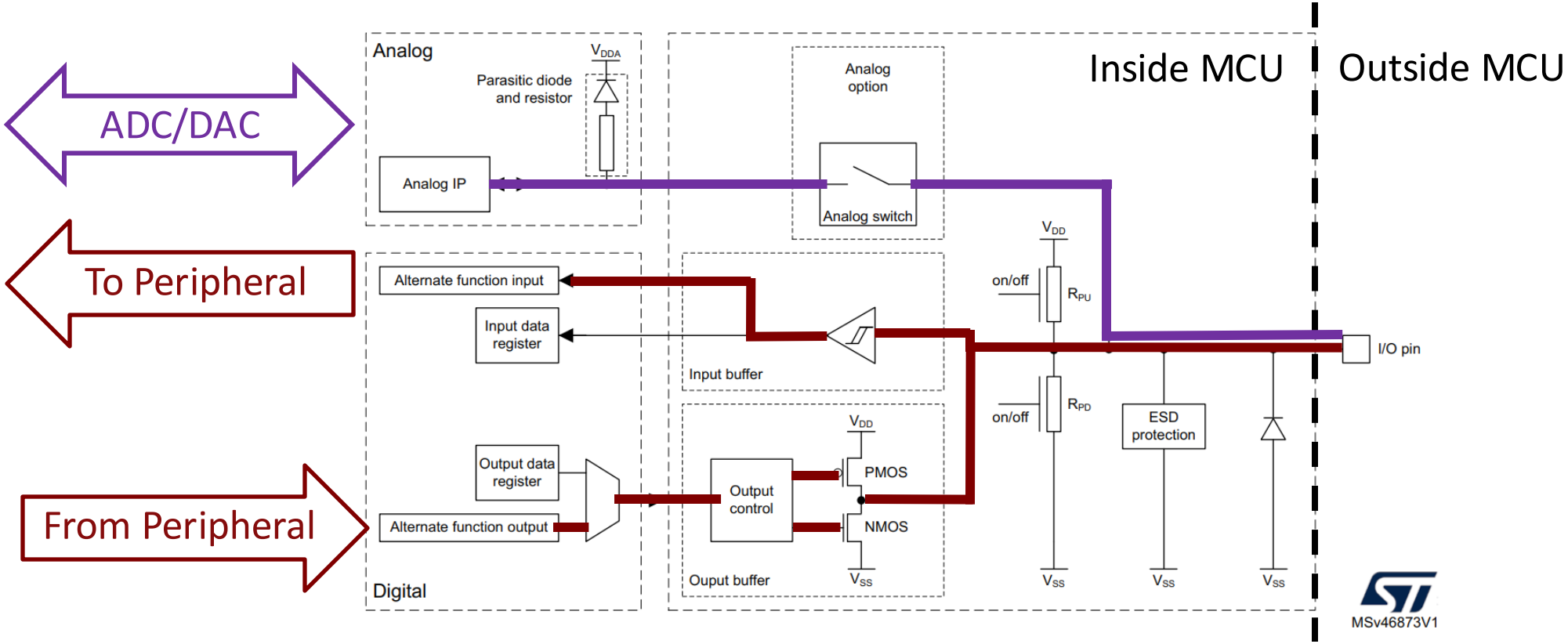
GPIO Operating Modes – Digital Output Speed

- The slew rate is configurable by setting the GPIO output speed

- For STM32L4 (the MCU we will use in the lab):
  - low speed (<400 kHz)
  - Medium speed (<3 MHz)
  - Fast speed (<10 MHz)
  - High speed (<40 MHz)

- This changes the drive strength of the output transistor, changing the effective R value of the RC equivalent.

GPIO: Alternate Function & Analog Pins



- *Analog pins* connect to digital-to-analog or analog-to-digital converters.
- *Alternate function pins* connect to dedicated peripherals, such as communication interfaces, timers, PWM outputs, …

Embedded Systems in a Picture – An Overview

Hardware Perspective

- Defining the physical Connection (Schema)
- Designing the actual electronics

System Overview

Firmware/Software Perspective

- programming functionality
- interfacing HW w. drivers

Microcontroller (MCU)

Processor

Debug/Program interface

Flash Memory

SRAM

Ports

...

Battery

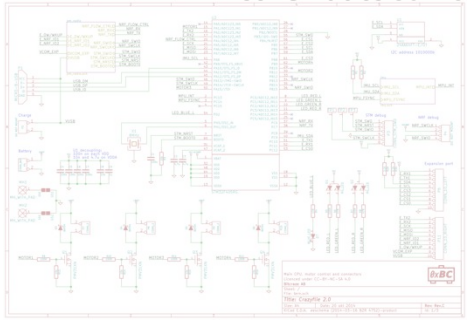
Sensors

Motors

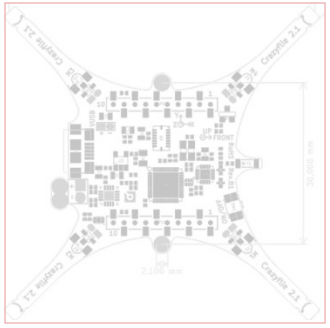
Ext. Memory

This Week

schematics.SchDoc



pcb.PcbDoc



code.c

```
// High-level language
// C-source code
int main (void){
    int total = 0;
    for (int i = 0;
        total += i;
    }
    while(1);
}
```

Compiler

code.asm

```
// Low-level Language
// Target Assembly Code
        MOVS r1, #0
        MOVS r2, #0
        B    check
loop:   ADD  r1, r1, r0
        ADDS r0, r0, #1
check:  CMP  r0, #10
        BLT  loop
self:   B    self
```

Next Week

Assembler

code.hex

```
0x2100
0x2000
0xE001
0x4401
0x1C40
0x280A
0xDBFB
0x8F00
0xE7FE
```

Target Machine Code

code.bin

```
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
```

Learning objective of Today Class

- ✓ Where we are what we will do
- ✓ What are Embedded Systems
- ✓ Both, hardware and software are important
- ✓ Block Diagram and Schematics
- ✓ GPIOs
- ✓ Trends

PBL Open Day

Tue 24 Sept.
16:00-18:30
Gloriastrasse 35, E floor

Open to all students and D-ITET staff!

CENTER FOR PROJECT-BASED LEARNING

TINY ML

D-ITET Center for Project-Based Learning

FUTURE LEARNING INITIATIVE

16:00 - 18:30

Projects exhibition

Live demo of the center projects

TinyML Swiss Demos

16:30 - 17:00

Ceremony

Best BSc Thesis Award ***by Peoplefone***

Best PBL Theses

Check our projects here

SCAN ME

See You Tomorrow or Next Week!

CENTER FOR PROJECT-BASED LEARNING