

Embedded Systems

Lecture 11

Digital Signal Processing - Project

© Michele Magno

D-ITET Center for Project-Based Learning



Where We are

Hardware-
Software

- 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. No Lecture
- 5. Hardware-Software Interfaces - Timer, PWM and ADC

Real-Time

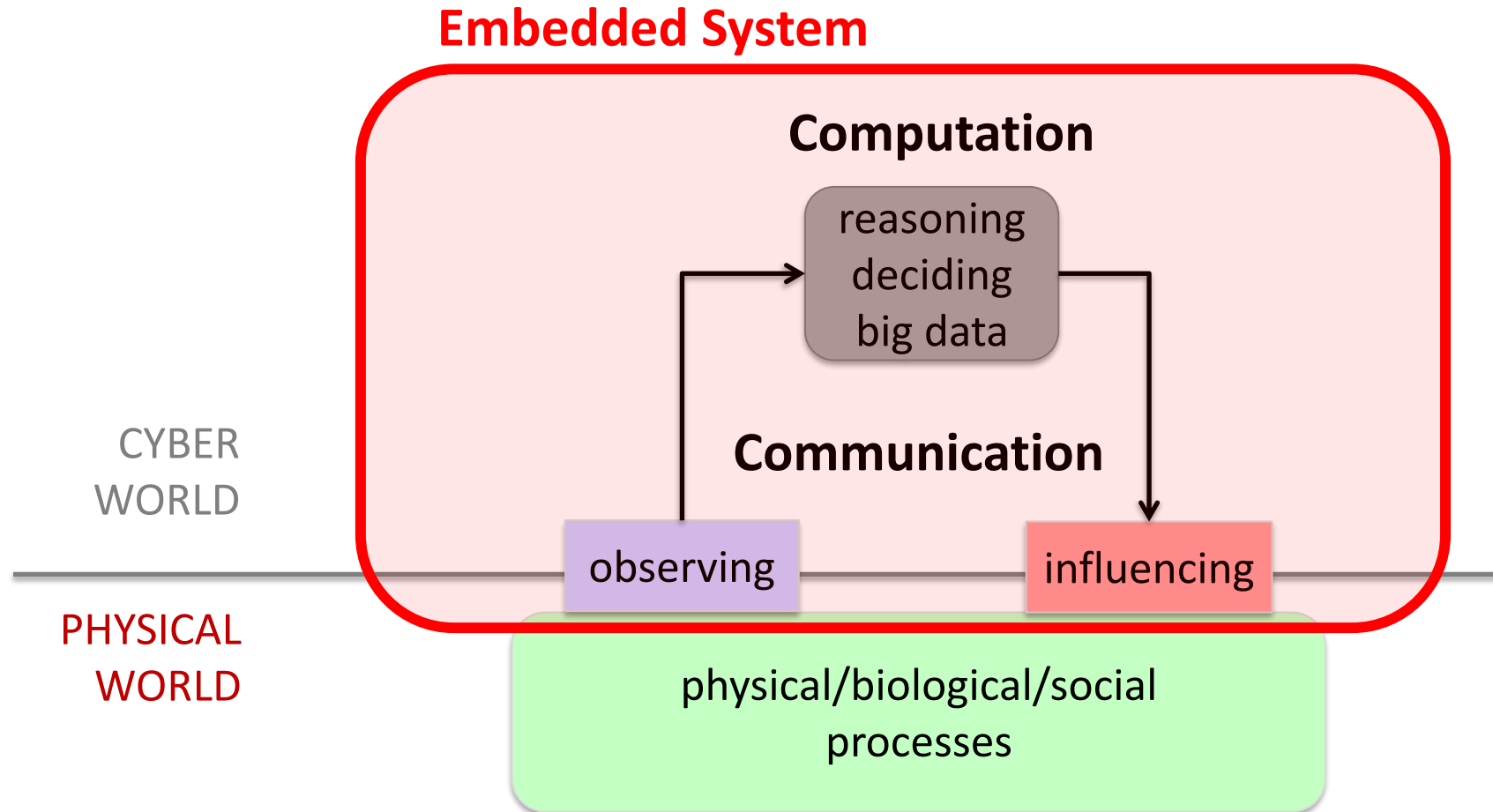
- 6. Real-Time Systems
- 7. Dynamic Scheduling and Real-Time Operating Systems
- 8. Deterministic Scheduling
- 9. Low Power Design

Special

- 10. Computational Units
- 11. Implementation Strategies & Project Kick-off
- 12. Project Q&A



Embedded Systems



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

Components and Requirements by Example

- Hardware – Software System Architecture -



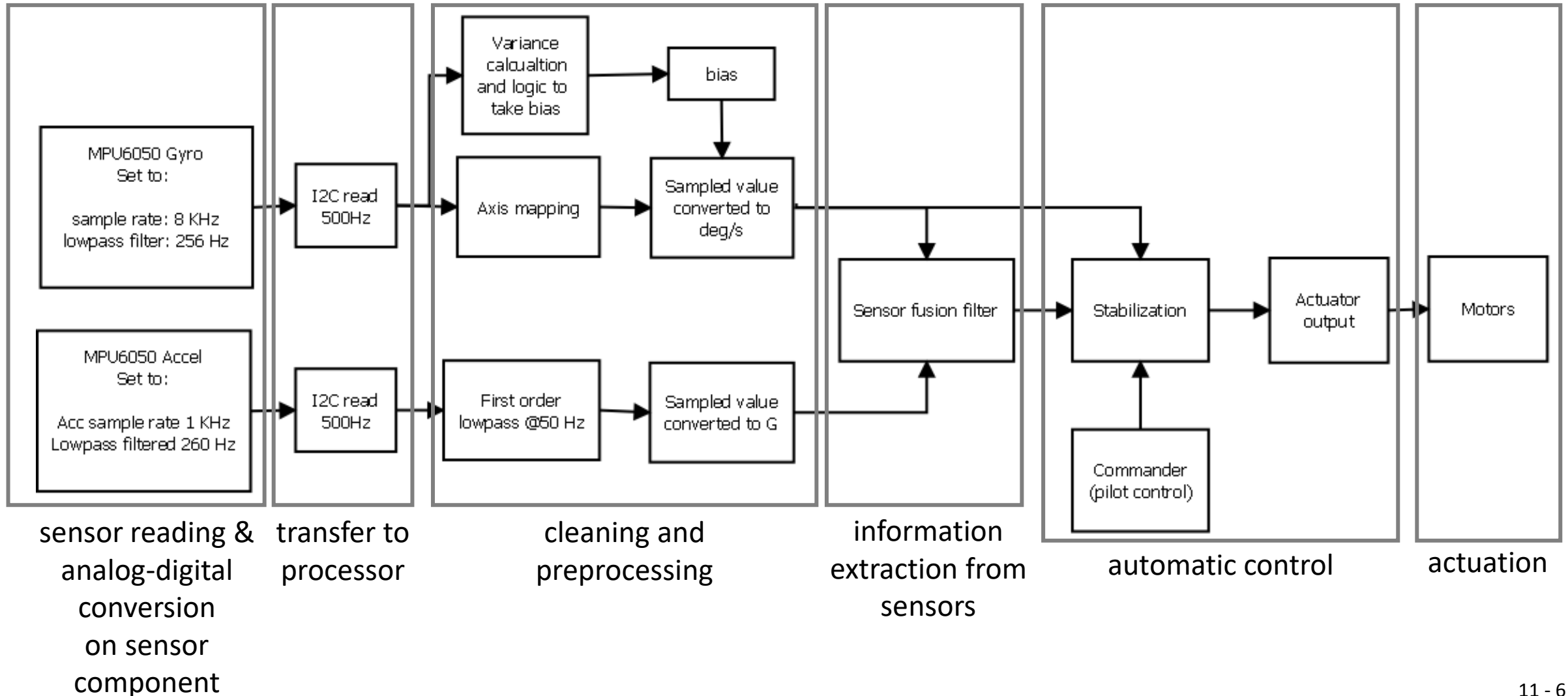
High-Level Software View

The *software architecture* supports

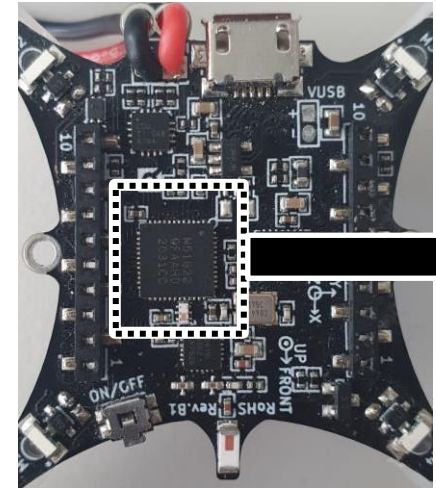
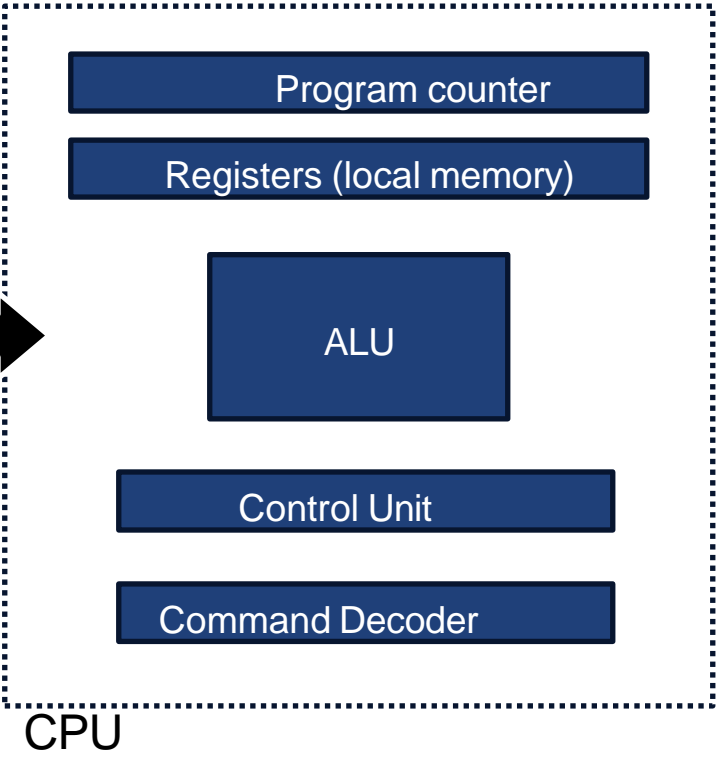
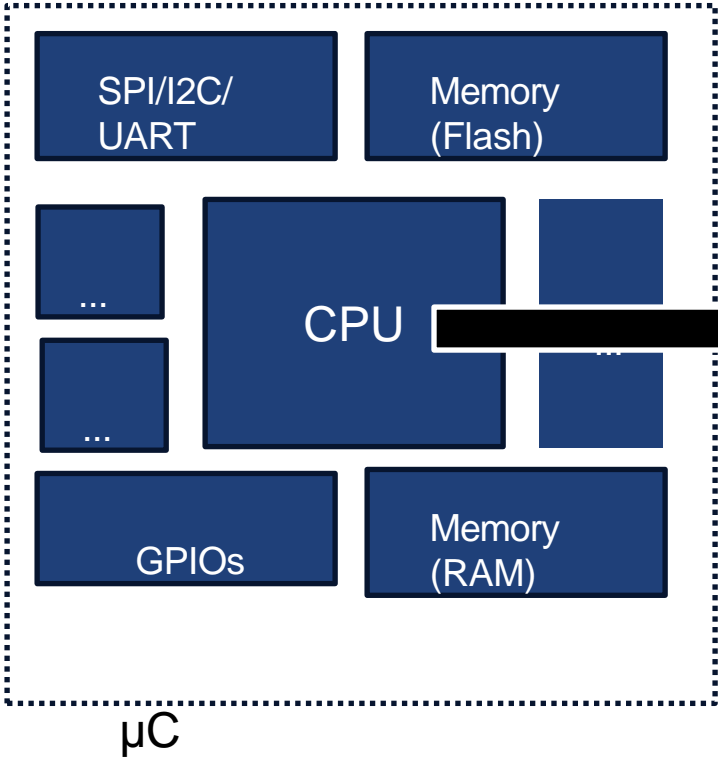
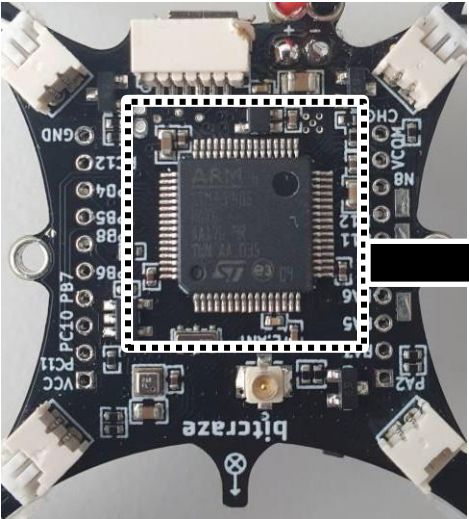
- *real-time tasks* for motor control (gathering sensor values and pilot commands, sensor fusion, automatic control, driving motors using PWM (pulse width modulation, ...) but also
- *non-real-time tasks* (maintenance and test, handling external events, pilot commands, ...).

High-Level Software View

Block diagram of the stabilization system:

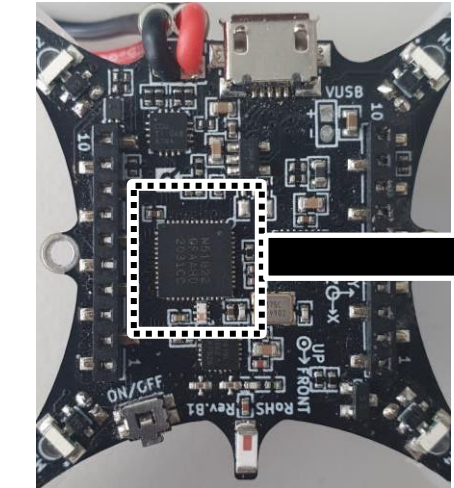
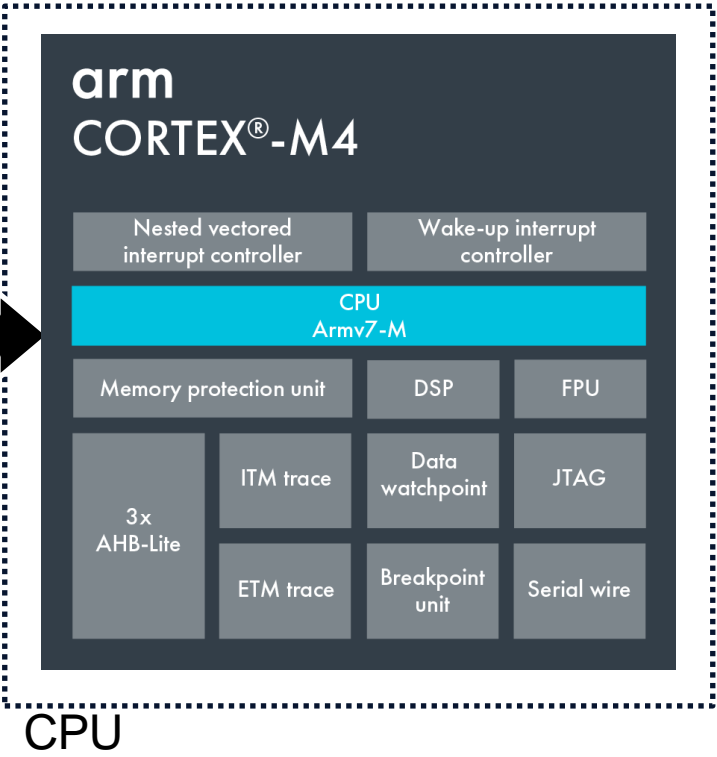
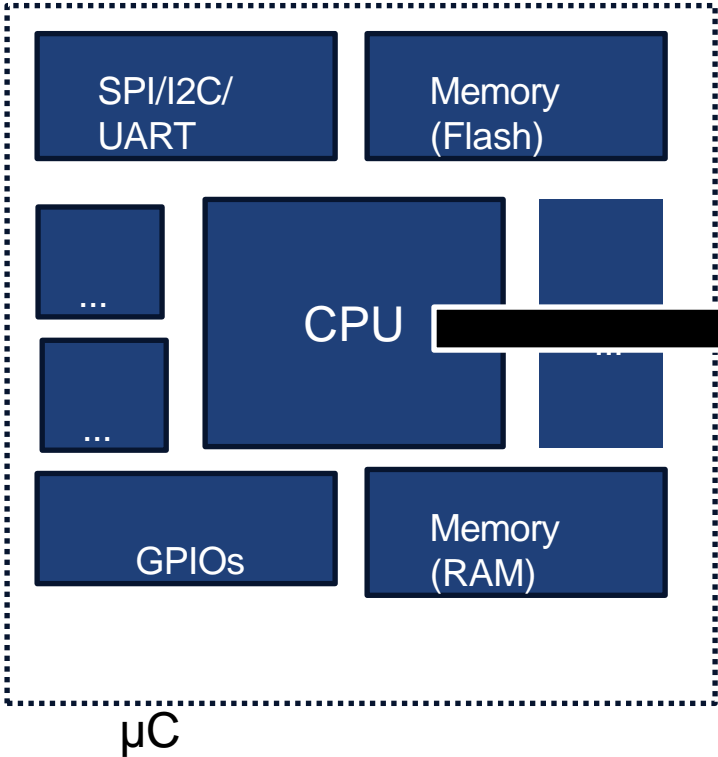
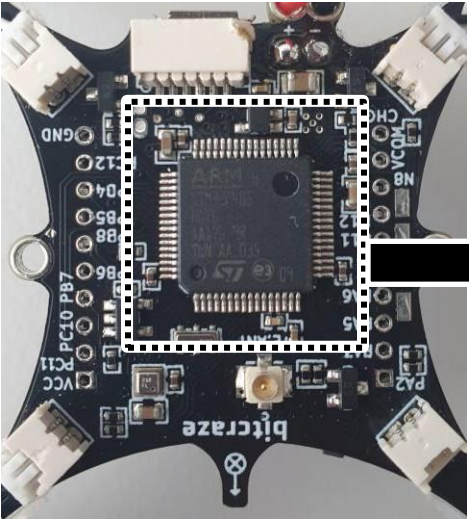


Where is the Software Processed?



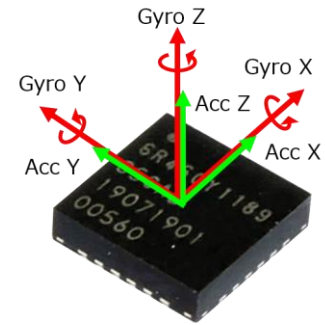
Embedded systems with drones

Where is the Software Processed?

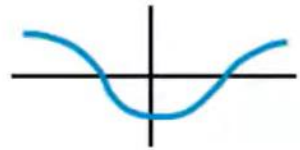


Embedded systems with drones

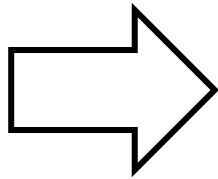
Digital Signal Processing



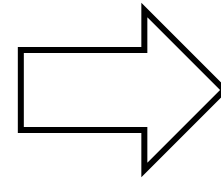
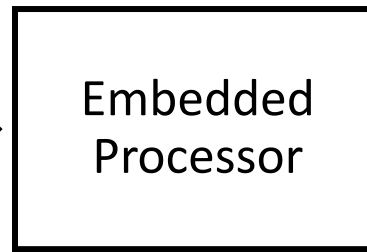
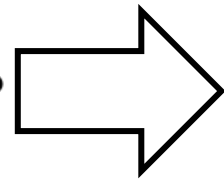
ARM Cortex-M4
with DSP Instruction



Analog
Input
Signal



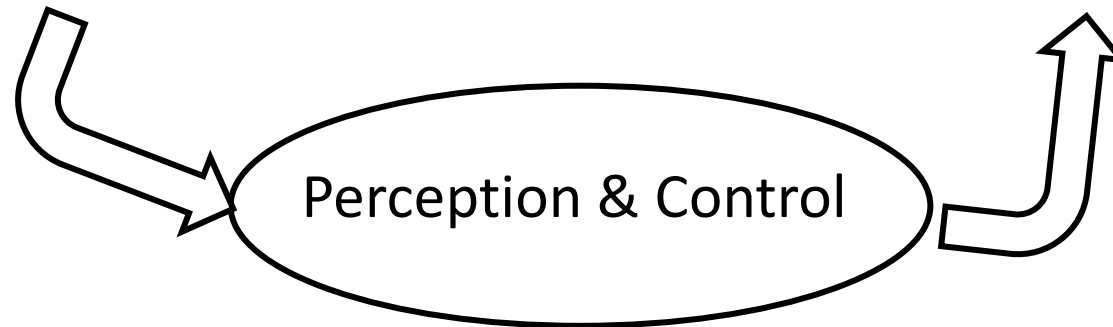
ADC



DAC -PWM

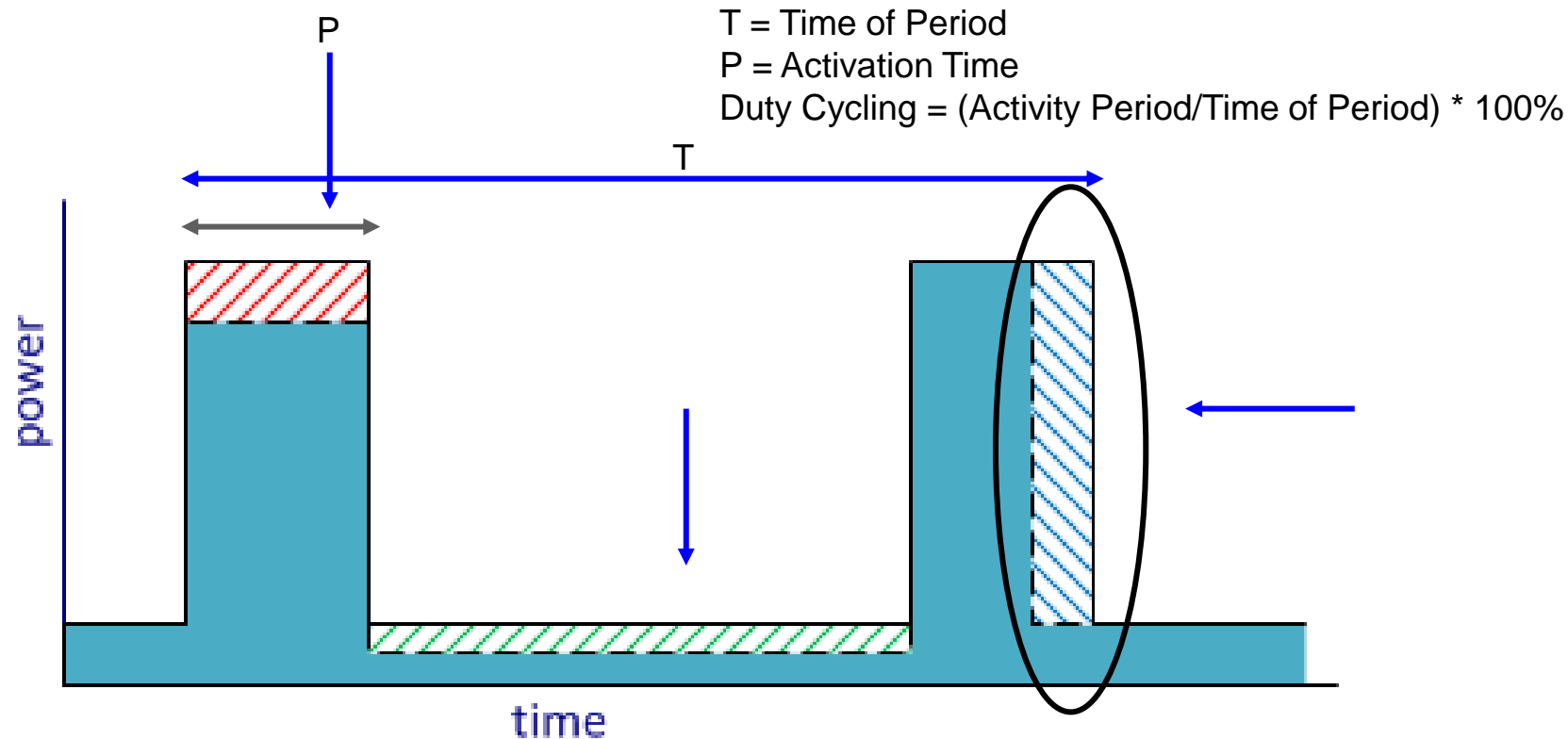


Analog
Output
Signal



Performance is Energy – But also Latency!

- The relation and balance between the performance and execution time needs to be carefully analyzed to find a best compromise which leads to lowest energy consumption.

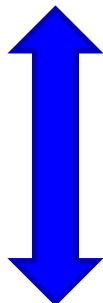


Less time spent in active → lower total energy consumed.

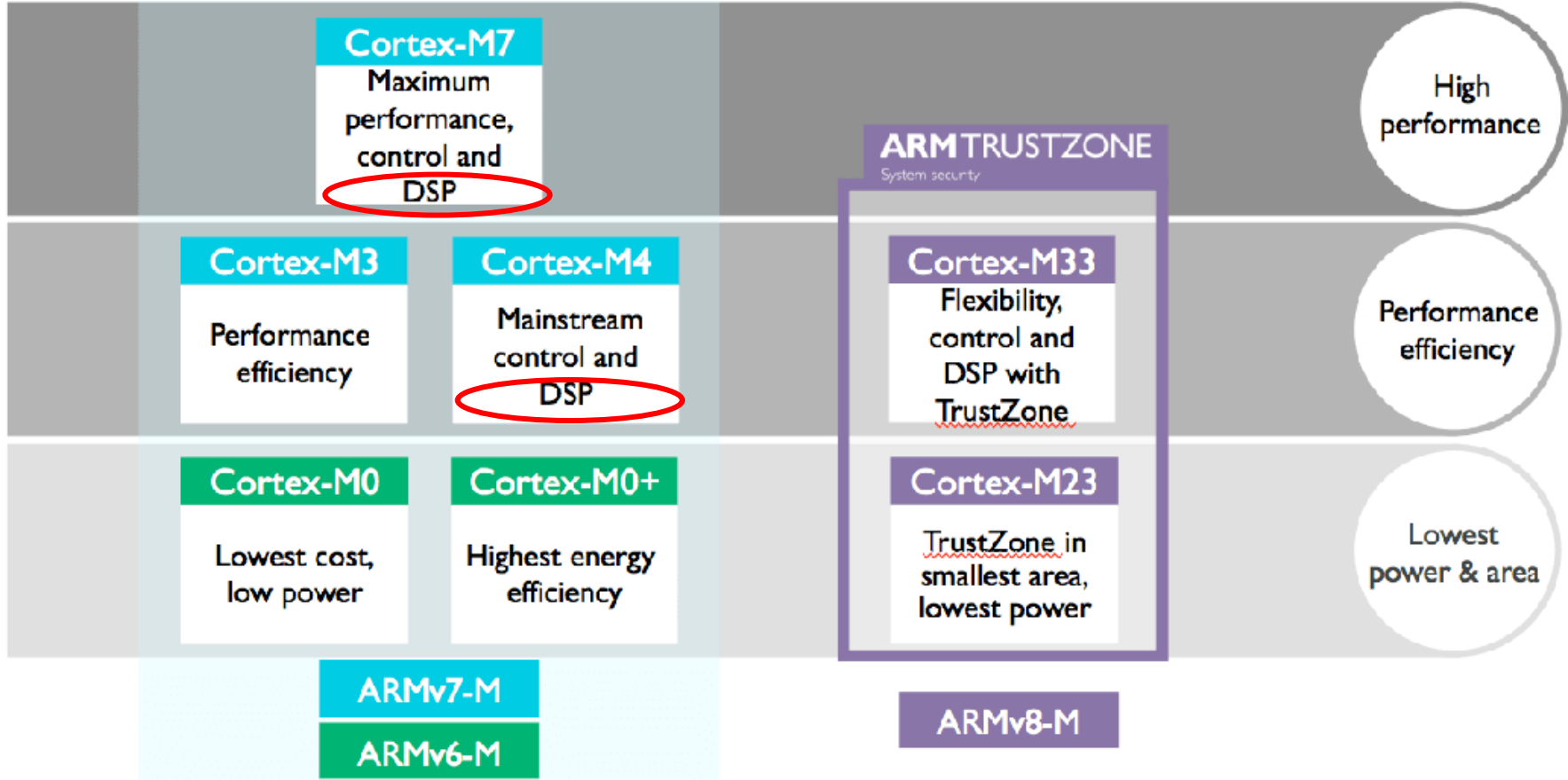
We can do so by using optimization techniques.

You need to know your Architecture!

FPU Double precision
SIMD Instructions



No FPU
No DSP



Cortex[®] DSP instructions

The Cortex[®]-Mx cores feature several instructions that result in efficient implementation of DSP algorithms.

Collection of 61 algorithms including

- Basic mathematical functions with vector operations
- Fast mathematical functions, like sine and cosine
- Complex mathematical functions like calculating magnitude
- Filtering functions like FIR or IIR
- Matrix computing functions
- Transform functions like FFT
- Controller functions like PID controller
- Statistical functions like calculating minimum or maximum
- Support functions like converting from one format to another
- Interpolation functions

CMSIS-DPS mandatory for M4 and M7 and ML

There are a number of arithmetic instructions for integer and fractional datatypes supported by the Cortex-M4 and Cortex-M7 which are most frequently used in DSP algorithms

- Accelerating the Multiply ACcumulate (MAC) operation
- Enabling the SIMD (Single Instruction Multiple Data) instructions

Fast Fourier Transforms (FFT) - ARM DSP.

Table Present the results for a running a Complex FFT for 32-bit input data (Q31) with block size varying from 32 to 1024

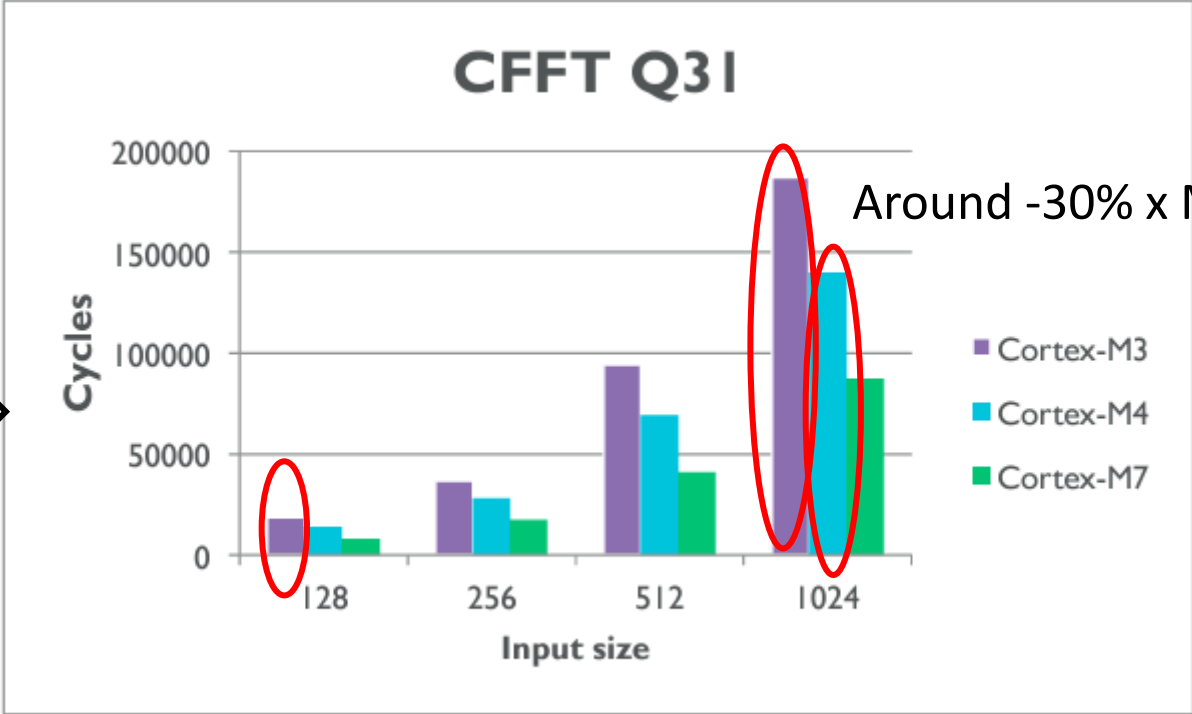
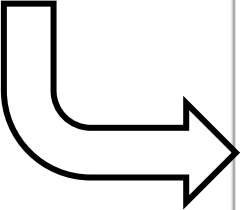
CFFT Q31	Block Size					
	32	64	128	256	512	1024
Cortex-M3	3374	6695	18549	36779	94267	187204
Cortex-M4	2577	5282	13823	28000	69253	139898
Cortex-M7	1497	3235	8050	17235	41076	87128

Cycle counts (hence small values are better).

Architecture is CRUCIAL

Cortex-M7 can execute up to two instruction per clock cycle (dual issue capability)

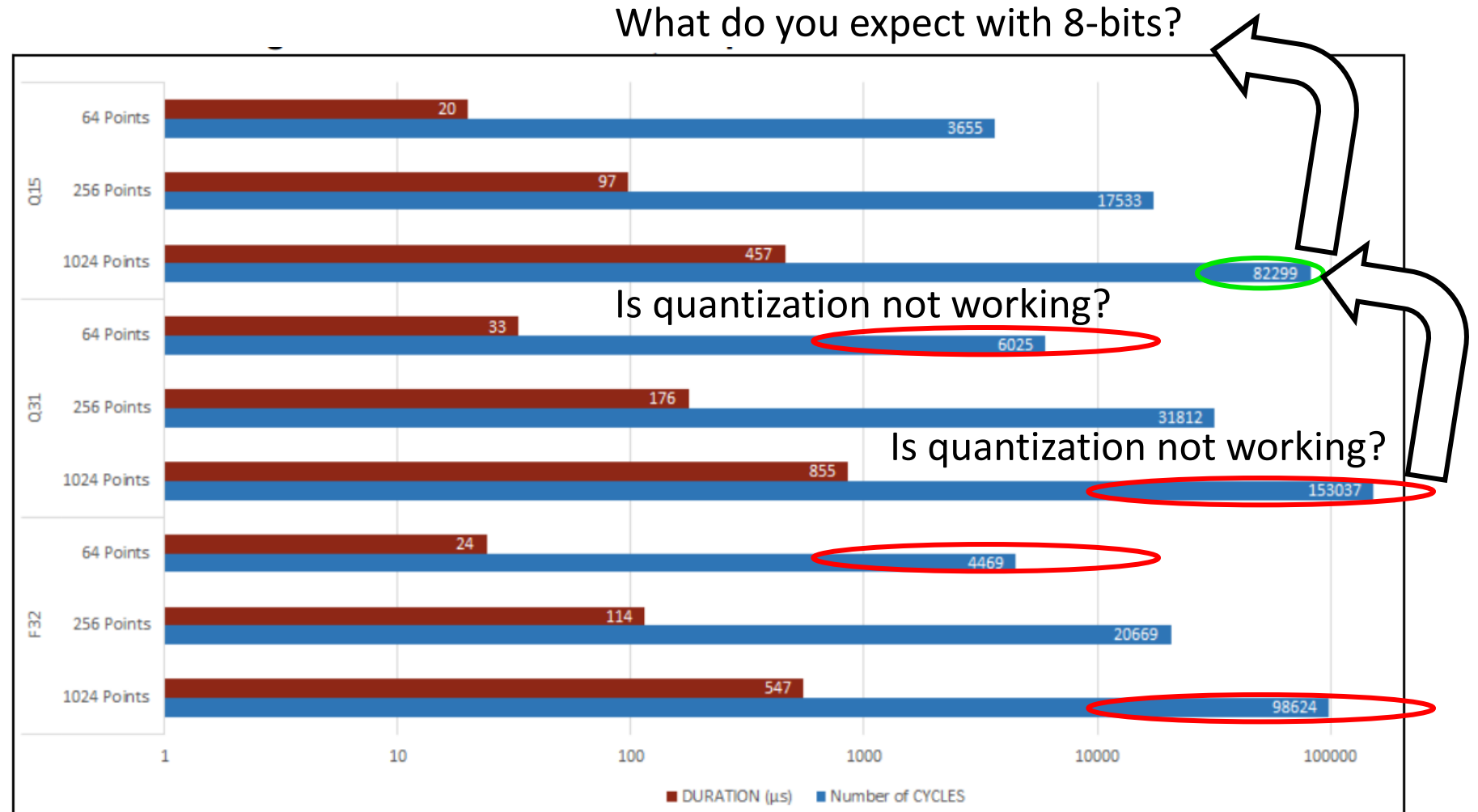
- Cortex-M7 has dynamic branch predication
- The floating-point unit in Cortex-M7 is designed to support higher floating point processing capability



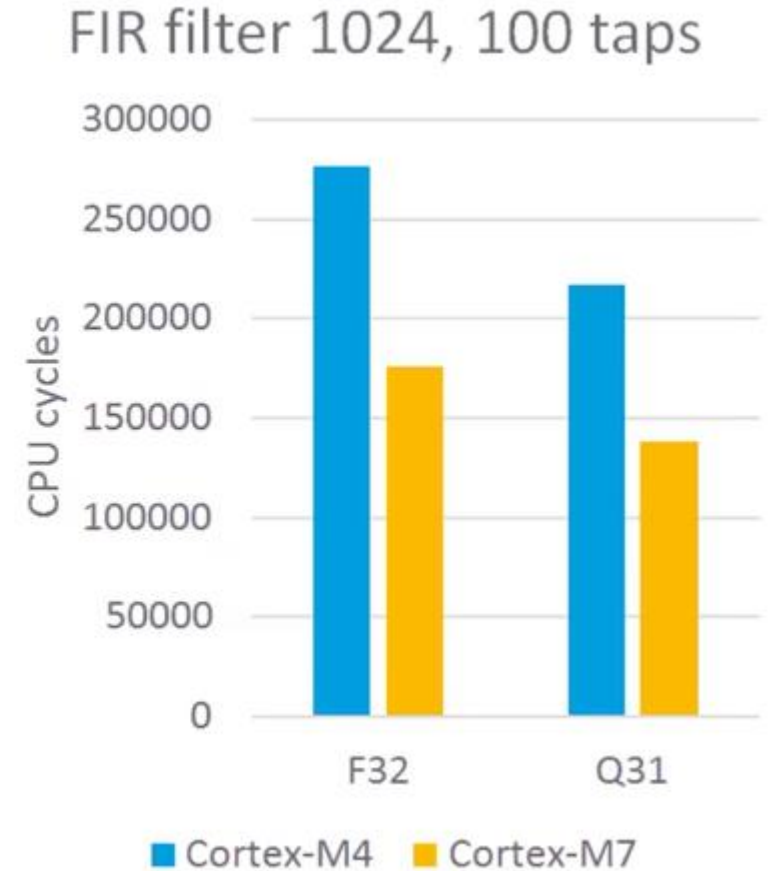
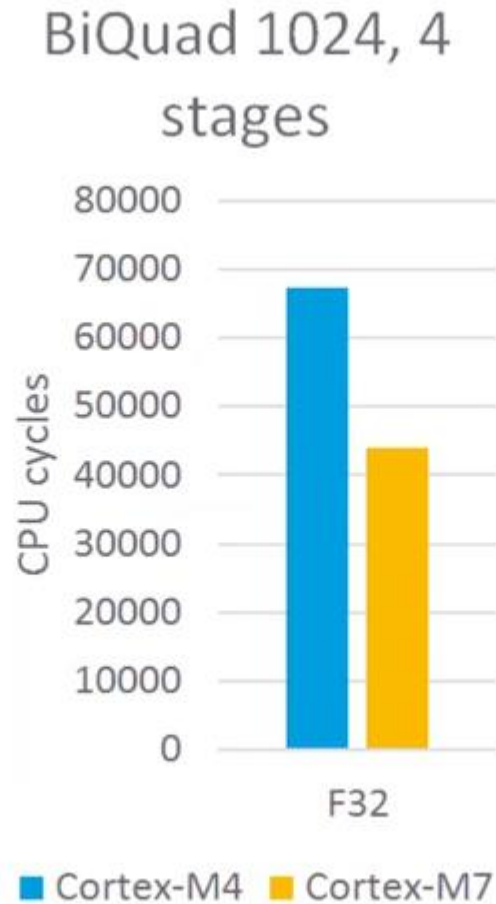
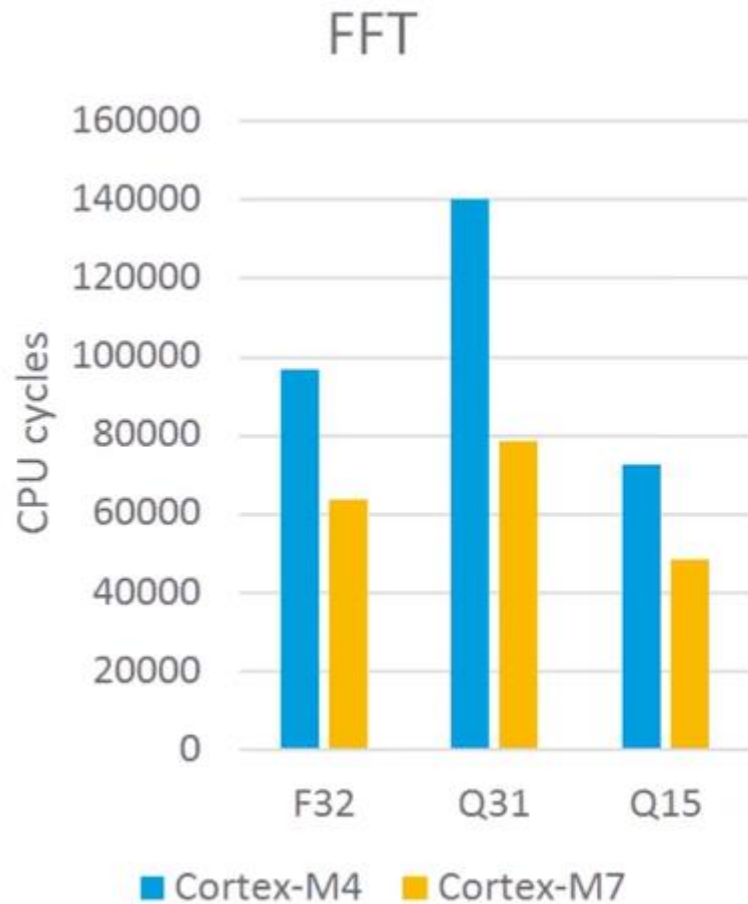
Source: The DSP capabilities of ARM® Cortex®-M4 and Cortex-M7 Processors

DSP and Fix vs Float

FFT size calculation performance ARM Cortex-M4F



Comparison M4 Vs M7 vs Operations and tasks



Quantifying the energy saving

▪ Comparison of Energy Consumption:

1. Cycle Count Reduction:

1. Without DSP (or with NOT optimized code): 200 instructions.
2. With DSP (or with optimized code): 50 instructions.
3. **Reduction:** 75% fewer instructions executed.

2. Energy Savings: Energy consumption is proportional to the number of instructions executed.

1. If the microcontroller consumes $E_{xClockPeriod} = P_{ActiveMode} N_{instructions}$
2. Energy Without DSP: $200 \times E_{xClockPeriod}$
3. Energy With DSP: $50 \times E_{xClockPeriod}$
4. **Assuming $E_{xClockPeriod}$ Doesn't change with the use of DSP instructions**
5. **Energy Saved: 75% reduction**

CMSIS-DSP (LAB)

The screenshot displays the CMSIS-DSP software library website. At the top left is the CMSIS logo. The main header reads "CMSIS-DSP Version 1.9.0" and "CMSIS DSP Software Library". Below this is a navigation bar with tabs for "General", "Core(A)", "Core(M)", "Driver", "DSP", "NN", "RTOS v1", "RTOS v2", "Pack", "Build", "SVD", "DAP", and "Zone". The "DSP" tab is selected. Below the navigation bar are sub-tabs for "Main Page", "Usage and Description", and "Reference". The "Reference" sub-tab is active, showing a list of functions: Support Vector Machine functions (SVM), Bayes classifier functions, Distance functions, and Quaternion functions. The main content area contains several paragraphs of text, including a section titled "Using the Library" which advises using `-Ofast` for performance and mentions the `arm_math.h` header file. Other sections include "Examples", "Toolchain Support", and "Preprocessor Macros", each with a brief description and a list of relevant macros.

CMSIS-DSP Version 1.9.0
CMSIS DSP Software Library

General Core(A) Core(M) Driver **DSP** NN RTOS v1 RTOS v2 Pack Build SVD DAP Zone

Main Page Usage and Description Reference

CMSIS-DSP

▼ CMSIS DSP Software Library

- Introduction
- Using the Library
- Examples
- Toolchain Support
- Preprocessor Macros
- CMSIS-DSP in ARM::CMSIS Pack
- Revision History of CMSIS-DSP
- Revision History of CMSIS-DSP
- Deprecated List

▼ Reference

- ▶ Examples
- ▶ Basic Math Functions
- ▶ Bayesian estimators
- ▶ Complex Math Functions
- ▶ Controller Functions
- ▶ Distance functions
- ▶ Fast Math Functions
- ▶ Filtering Functions
- ▶ Interpolation Functions
- ▶ Matrix Functions
- ▶ Quaternion Math Functions
- ▶ Statistics Functions
- ▶ Support Functions
- ▶ SVM Functions
- ▶ Transform Functions

- Support Vector Machine functions (SVM)
- Bayes classifier functions
- Distance functions
- Quaternion functions

The library has generally separate functions for operating on 8-bit integers, 16-bit integers, 32-bit integer and 32-bit floating-point values.

The library is providing vectorized versions of most algorithms for Helium and of most f32 algorithms for Neon.

When using a vectorized version, provide a little bit of padding after the end of a buffer (3 words) because the vectorized code may read a little bit after the end of

Using the Library

The library is released in source form. It is strongly advised to compile the library using `-Ofast` to have the best performances.

The library functions are declared in the public file `arm_math.h` which is placed in the `Include` folder. Simply include this file. If you don't want to include everything,

Examples

The library ships with a number of examples which demonstrate how to use the library functions.

Toolchain Support

The library is now tested on Fast Models building with `cmake`. Core M0, M4, M7, M33, M55, A32 are tested.

Preprocessor Macros

Each library project have different preprocessor macros.

- `ARM_MATH_BIG_ENDIAN`:

Define macro `ARM_MATH_BIG_ENDIAN` to build the library for big endian targets. By default library builds for little endian targets.

- `ARM_MATH_MATRIX_CHECK`:

Define macro `ARM_MATH_MATRIX_CHECK` for checking on the input and output sizes of matrices

- `ARM_MATH_ROUNDING`:

Example Real FFT function (LAB)

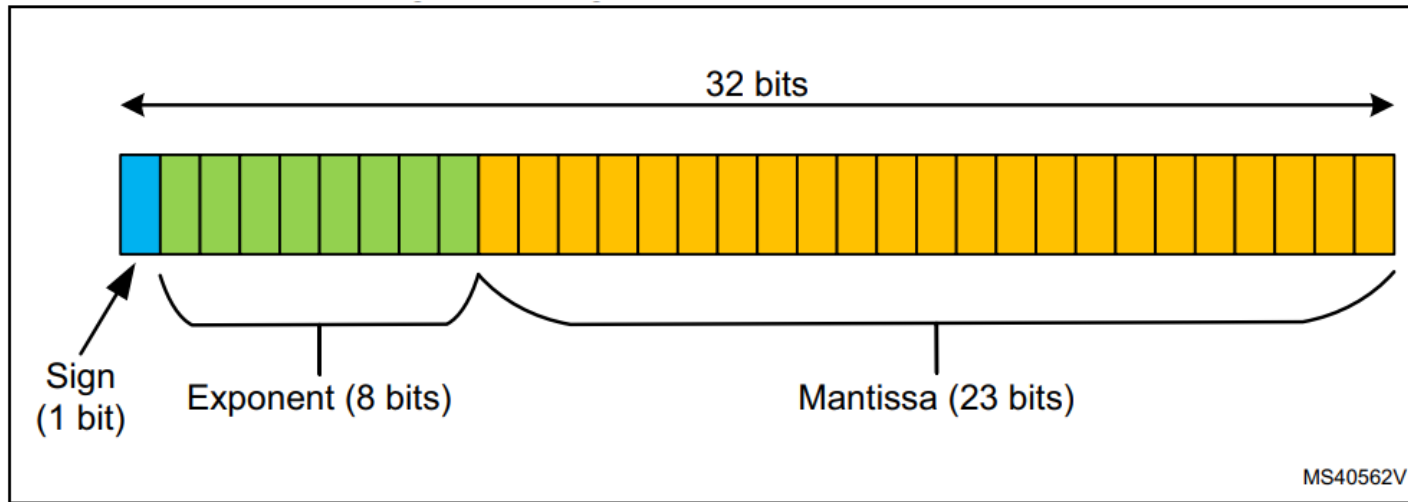
What is doing
arm_rfft
What is Q15???

The screenshot shows the CMSIS-DSP documentation page for Real FFT Functions. The left sidebar contains a navigation tree with 'Real FFT Functions' selected. The main content area is titled 'Real FFT Functions' and 'Transform Functions'. It lists several functions with their signatures and brief descriptions:

- void arm_rfft_f32** (const arm_rfft_instance_f32 *S, float32_t *pSrc, float32_t *pDst)
Processing function for the floating-point RFFT/RIFFT. Source buffer is modified by this function. More...
- void arm_rfft_fast_f16** (const arm_rfft_fast_instance_f16 *S, float16_t *p, float16_t *pOut, uint8_t ifftFlag)
Processing function for the floating-point real FFT. More...
- void arm_rfft_fast_f32** (const arm_rfft_fast_instance_f32 *S, float32_t *p, float32_t *pOut, uint8_t ifftFlag)
Processing function for the floating-point real FFT. More...
- void arm_rfft_fast_f64** (arm_rfft_fast_instance_f64 *S, float64_t *p, float64_t *pOut, uint8_t ifftFlag)
Processing function for the Double Precision floating-point real FFT. More...
- arm_status arm_rfft_fast_init_f16** (arm_rfft_fast_instance_f16 *S, uint16_t fftLen)
Initialization function for the floating-point real FFT. More...
- arm_status arm_rfft_fast_init_f32** (arm_rfft_fast_instance_f32 *S, uint16_t fftLen)
Initialization function for the floating-point real FFT. More...
- arm_status arm_rfft_fast_init_f64** (arm_rfft_fast_instance_f64 *S, uint16_t fftLen)
Initialization function for the Double Precision floating-point real FFT. More...
- arm_status arm_rfft_init_f32** (arm_rfft_instance_f32 *S, arm_cfft_radix4_instance_f32 *S_CFFT, uint32_t fftLenReal, uint32_t ifftFlagR, uint32_t bitReverseFlag)
Initialization function for the floating-point RFFT/RIFFT. More...
- arm_status arm_rfft_init_q15** (arm_rfft_instance_q15 *S, uint32_t fftLenReal, uint32_t ifftFlagR, uint32_t bitReverseFlag)
Initialization function for the Q15 RFFT/RIFFT. More...
- arm_status arm_rfft_init_q31** (arm_rfft_instance_q31 *S, uint32_t fftLenReal, uint32_t ifftFlagR, uint32_t bitReverseFlag)
Initialization function for the Q31 RFFT/RIFFT. More...
- void arm_rfft_q15** (const arm_rfft_instance_q15 *S, q15_t *pSrc, q15_t *pDst)
Processing function for the Q15 RFFT/RIFFT. More...
- void arm_rfft_q31** (const arm_rfft_instance_q31 *S, q31_t *pSrc, q31_t *pDst)
Processing function for the Q31 RFFT/RIFFT. More...

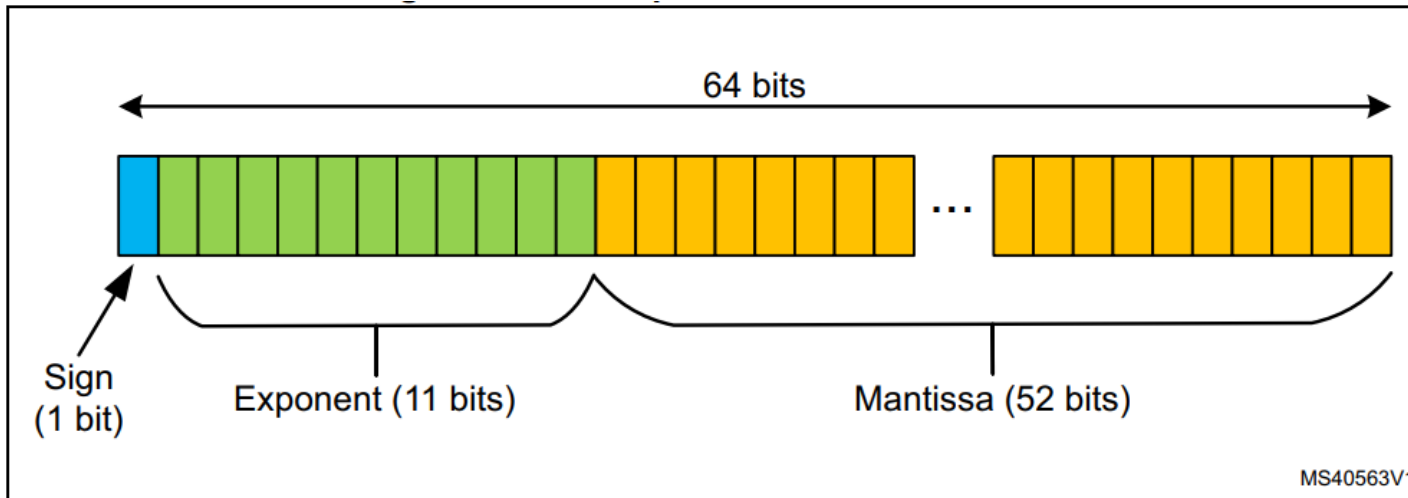
DSP operations can use either floating-point or fixed-point formats.

Single Precision



$$\text{Value} = (-1)^S \times M \times 2^{(E-127)}$$

Double Precision

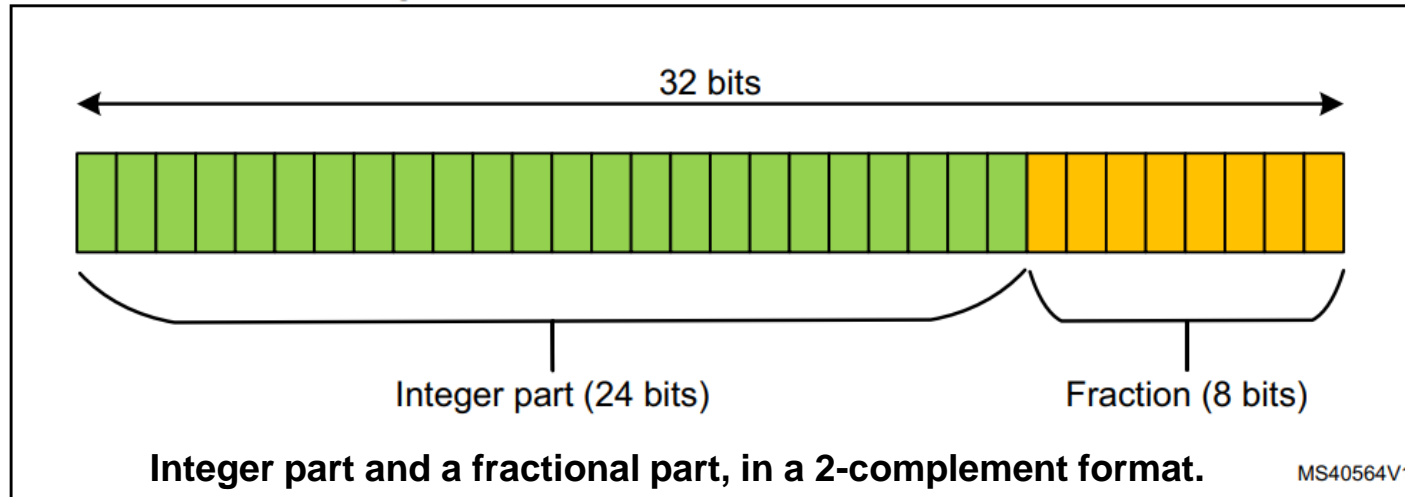


$$\text{Value} = (-1)^S \times M \times 2^{(E-1023)}$$

Fix Point Representation (LAB)

Most common format for DSP QX: Q7, Q15 and Q31

Where X is the number of fractional bits



32-bit fixed point representation.

$$\text{Value} = (-1)^{b_s} \times (b_{14} \times 2^{-1} + b_{13} \times 2^{-2} + \dots + b_1 \times 2^{-14} + b_0 \times 2^{-15})$$

The range of numbers supported in a Q15 number is comprised between -1.0 and 1.0, corresponding to the smallest and largest integers that can be represented, respectively

Avoiding Overflow

-32768 and 32767.

What Did You Learn?

- ✓ Embedded Systems relies on Hardware and Software co-design
- ✓ Implementation techniques
 - ✓ Understand the hardware architecture
 - ✓ DSP instruction on ARM Cortex-m
- ✓ Speed-up improvement with software programming



Time for Demo!! 😊

Robotics Today



From Edge to Extreme Edge Challenges

Advanced autonomous drone and vehicles

[1] A. Bachrach, "Skydio autonomy engine: Enabling the next generation of autonomous flight," IEEE Hot Chips 33 Symposium (HCS), 2021



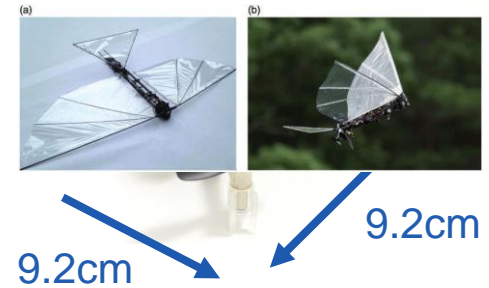
<https://www.skydio.com/skydio-2-plus>

Deployment in tight spaces?



Nano-drone

<https://www.bitcraze.io/products/crazyflie-2-1>



- 3D Mapping & Motion Planning
- Object recognition & Avoidance
- 0.06m² & **800g of weight**
- Energy Capacity (Battery) **5410mAh**



- Smaller form factor of 0.008m²
- Weight of **27g (30X lighter)**
- Battery capacity of **250mAh (20X smaller)**



Can we fit sufficient "intelligence" in a **30X** smaller payload and **20X** lower energy budget?

•Miniturized Drone

•WiFi module

•UWB centimeter precision ranging

•GAP9 SoC
•9+1 Cores
•1.5 MB RAM
•~ 100 mW

•total mass
•5.86 g

Perception

Mapping

Autonomous navigation

Localization

Onboard computation

•Quad ToF Deck

•4x VL53L5CX
•up to 60 Hz

•~300 mW

•4.2 g

•OmniVision
•OV5647 camera

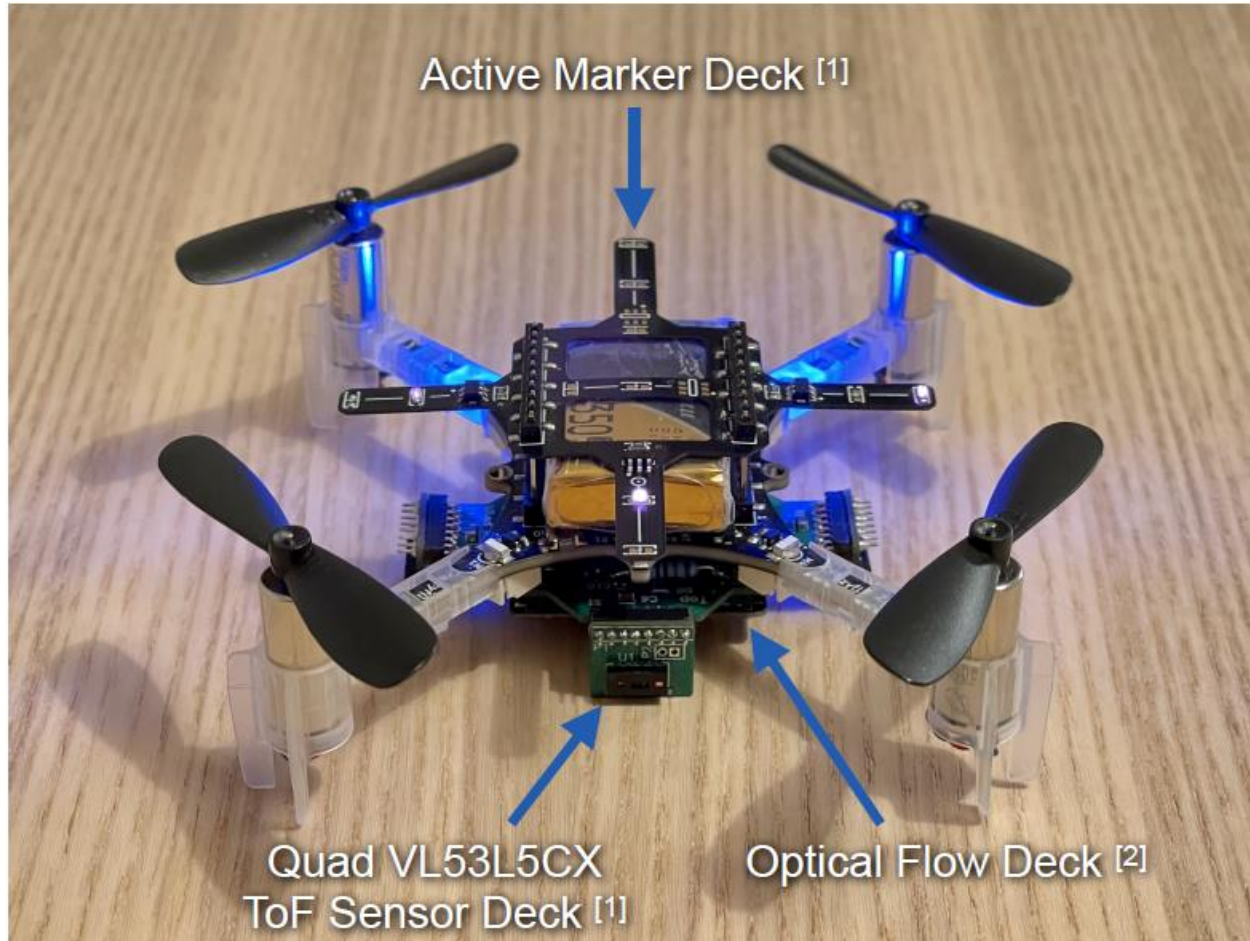
•RGB - 5 MPixel
•45 fps

•~90 mW

•search and rescue fast-mapping
•for emergency response



Fully Onboard Autonomous navigation, with mapping, planning , object avoidance and landing (LIVE DEMO!)



Each nano-UAV is equipped with:

1. 4x 64-pixels depth sensors
2. STM32 - controller
3. GAP9 - co-processor
4. NRF51 transceiver - intra-swarm communication
5. Active marker - ground truth

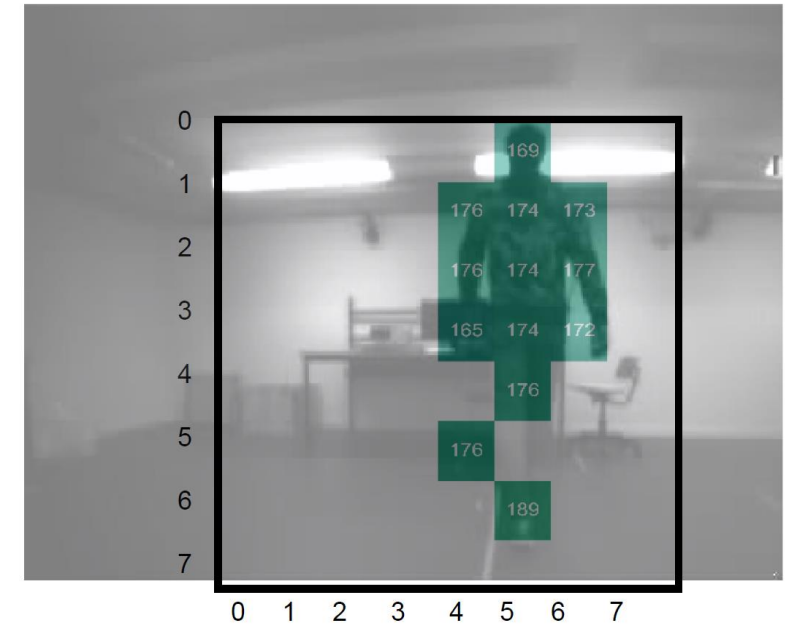
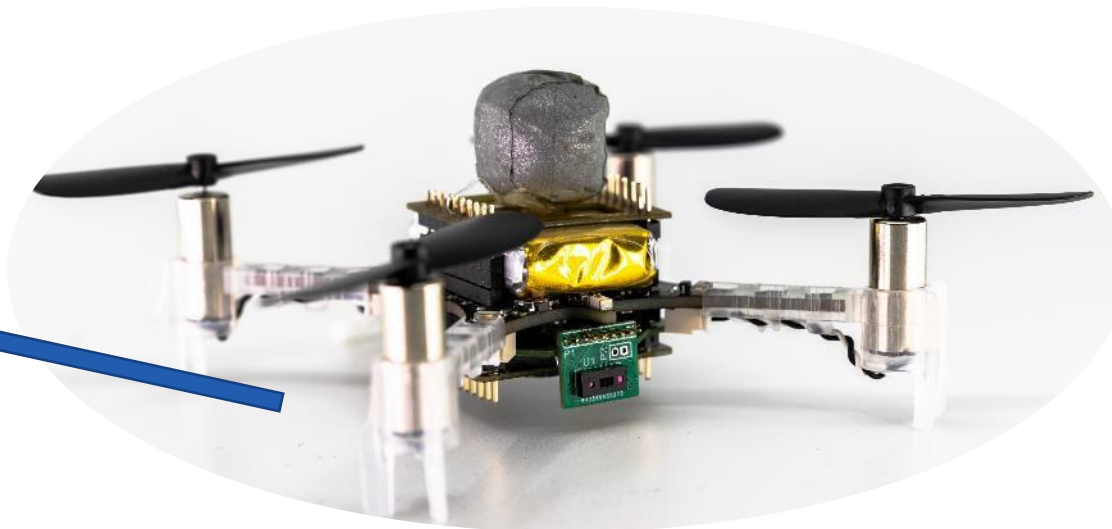
[1] Custom

[2] Commercially available

Sensor Fusion – RGB + Depth

8x8 VL53L8CX ToF sensor

- 8x8 or 4x4 multi-zone ranging
- weight 42 mg
- 2 cm – 4 m range
- automatic noise detection
- 215 mW @ 60 Hz

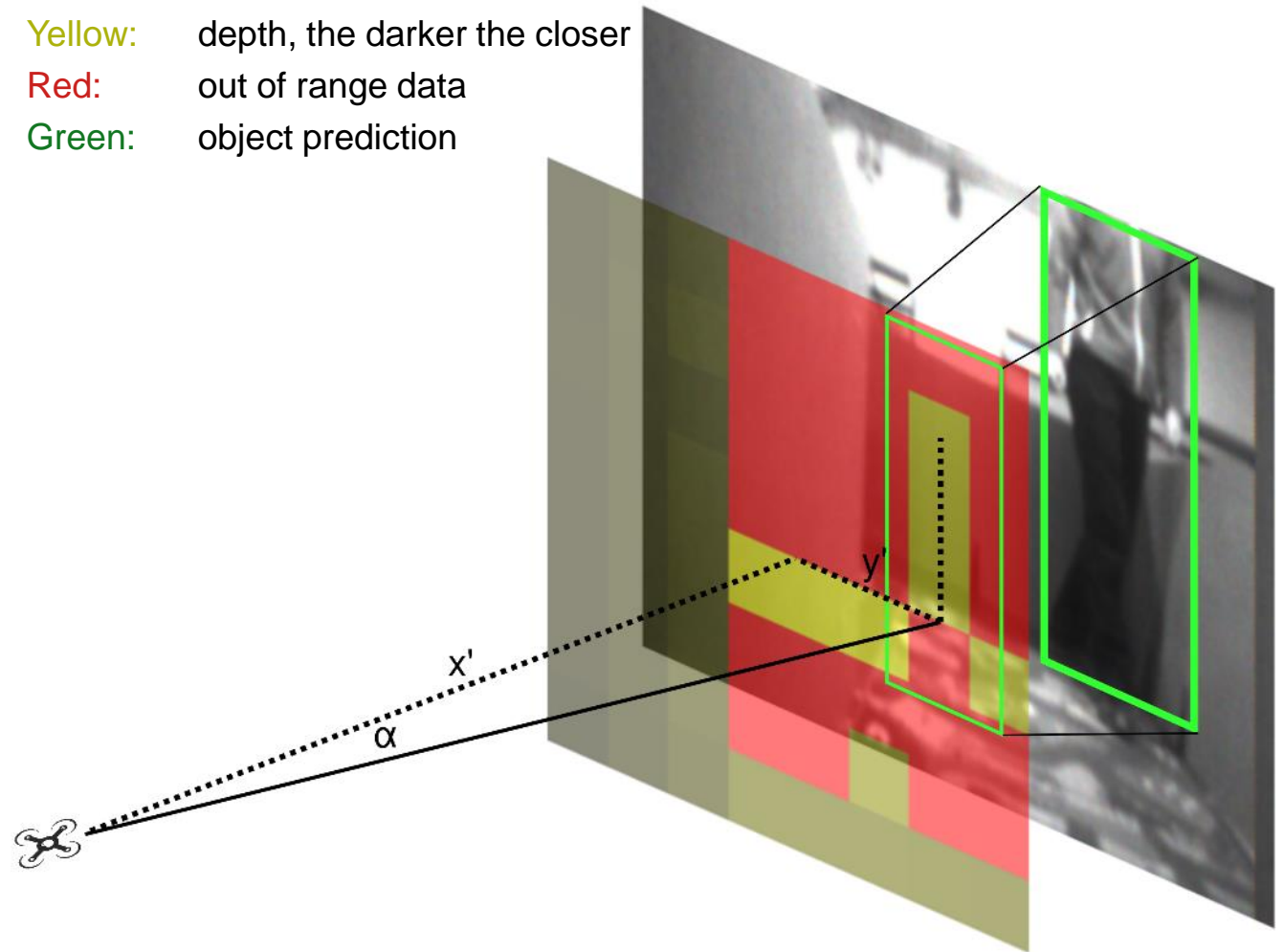


Sensor Fusion – RGB + Depth

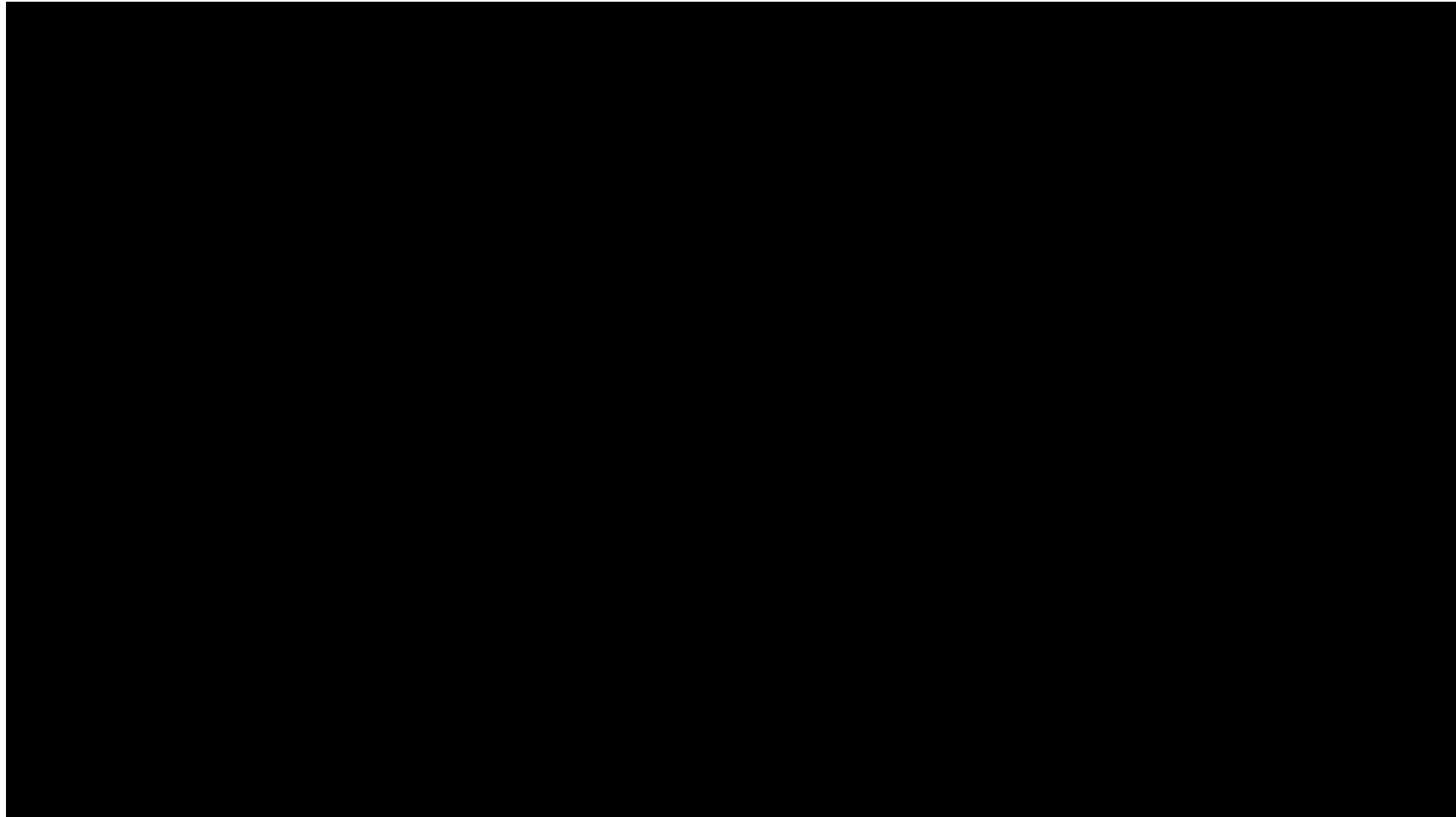
8x8 VL53L8CX ToF sensor

- 8x8 or 4x4 multi-zone ranging
- weight 42 mg
- 2 cm – 4 m range
- automatic noise detection
- 215 mW @ 60 Hz

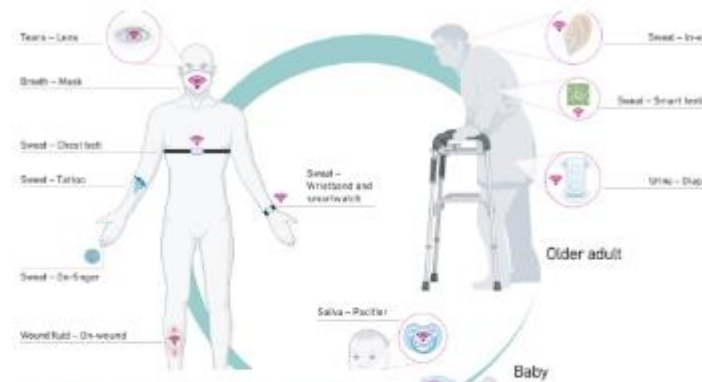
Yellow: depth, the darker the closer
Red: out of range data
Green: object prediction



If the demo did not work....



PBL Mini-Project

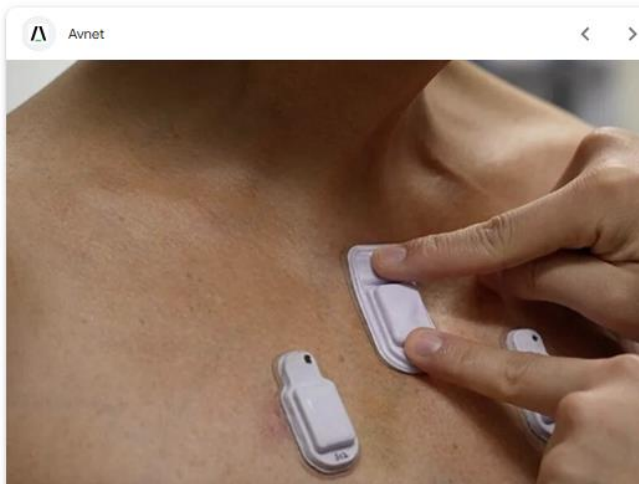
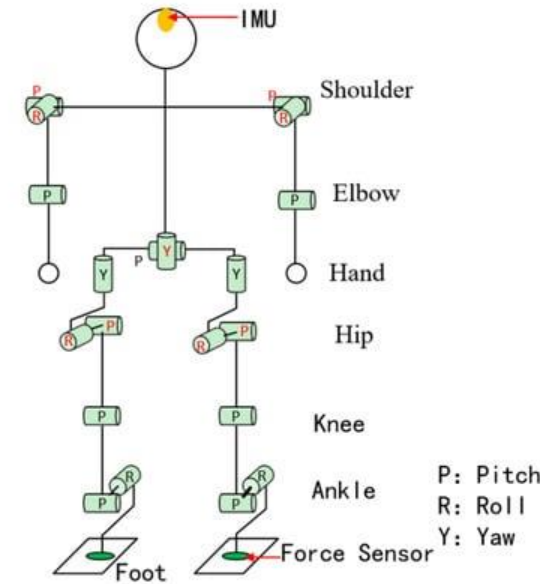


LIFE SCIENCES AND MEDICINE

How the latest sensors analyse body fluids

04.12.2024

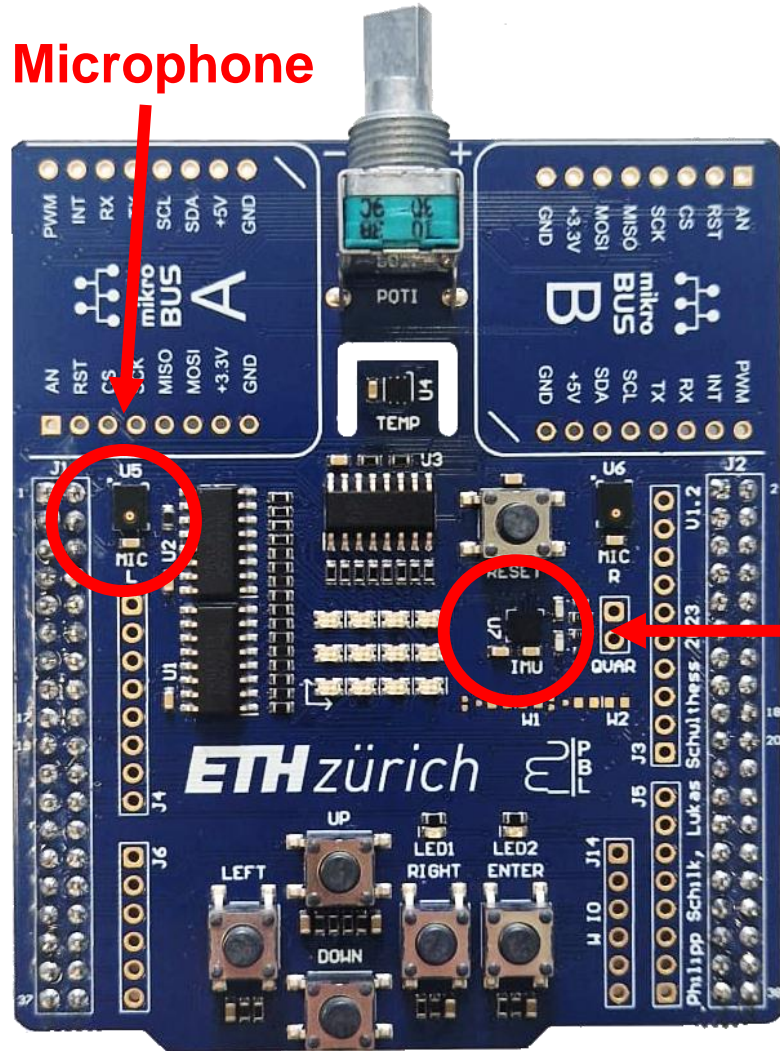
ETH zürich



Using sound to capture medical data boosts device innovation

Embedded Systems PBL Mini-Project

Microphone



IMU

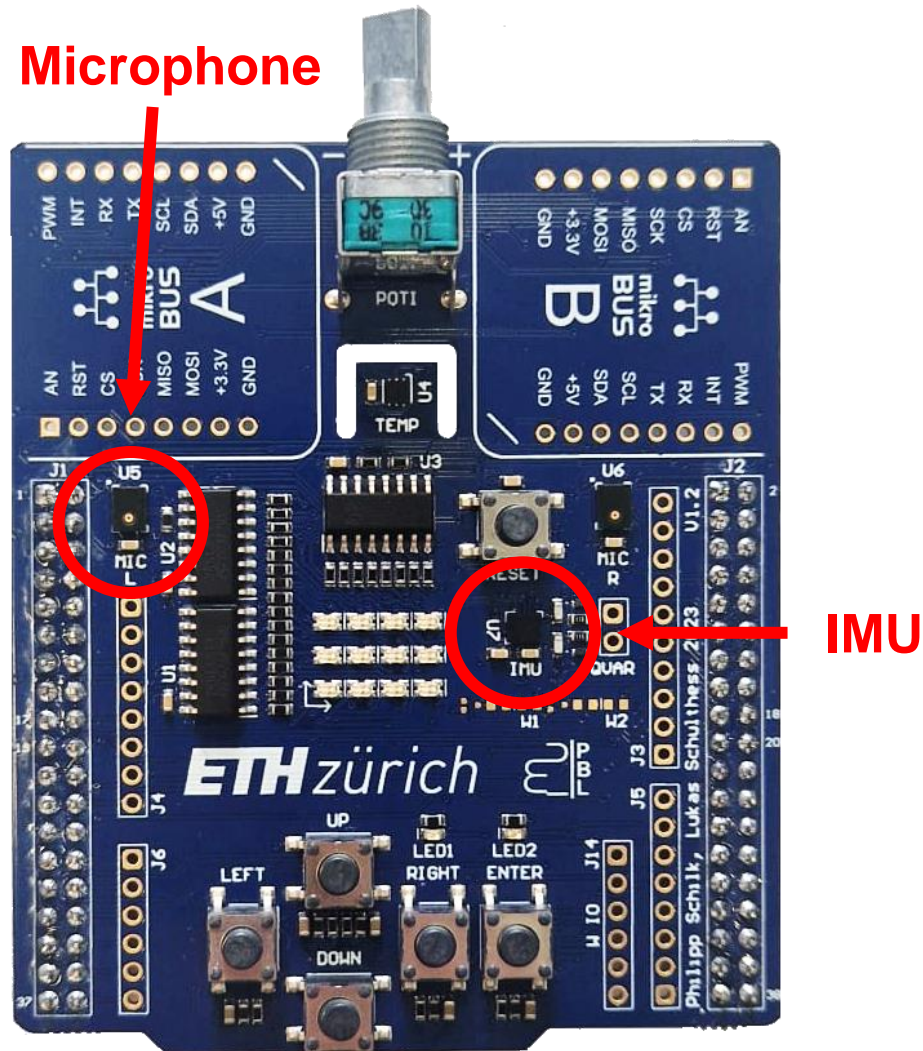
Goal: have a small real-world application running on your STM32 platform

- A few different ideas in **the task assignment**, but you may also realize an idea of your own, working with either the IMU or the microphone (**LAB helps here**)
- All applications will consist of **active mode** and **sleep mode**:
 - Collect sensor data (e.g. for 1 second)
 - Apply at least two signal processing algorithms on the data, so you can compare the performance
 - Giving the result to the user, then go to sleep mode

→ **How much energy will your application consume? And how fast will it process? To optimize, don't forget SIMD commands or the CMSIS-DSP library**

- But you can also play easy 😊 (i.e Time domain processing vs FFT)

Embedded Systems PBL Mini-Project



How to get the 0.25 bonus points?

- Collect data from at least one sensor and process it using a fixed sample rate and period.
- Provide a clear performance comparison of the two algorithms implemented in terms of speed and energy consumption.
- Include energy consumption calculations based on active and sleep phases.
- Submit a concise, half-page report summarizing the findings.
- Submit a 30sec to max. 60sec video of the developed application, showing its correct operation.
- Submit the code used to create the results

→ **Deadline for hand-in on Moodle:**
Dec. 20th 2024 23:59