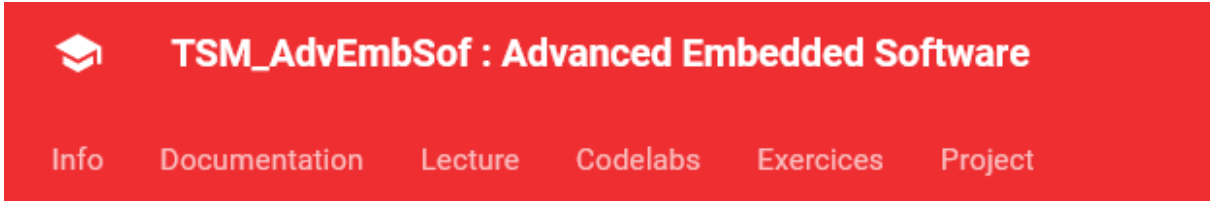
MSE | MASTER OF SCIENCE IN ENGINEERING

TSM_AdvEmbSof
Introduction

Some administrative matters

- Proposed schedule
  - 14h45-15h30 + 15h35-16h20 + 16h30-17h15
- Resources
  - Site: https://advembsof.isc.heia-fr.ch
  - Development kit, software
  - Moodle is not used for this lecture
- Project
  - At the end of the semester, you have to deliver the source code of a project.
  - Working in team of 2 students.
  - The project is evaluated and students may get a bonus to their grade (written exam).

Course content



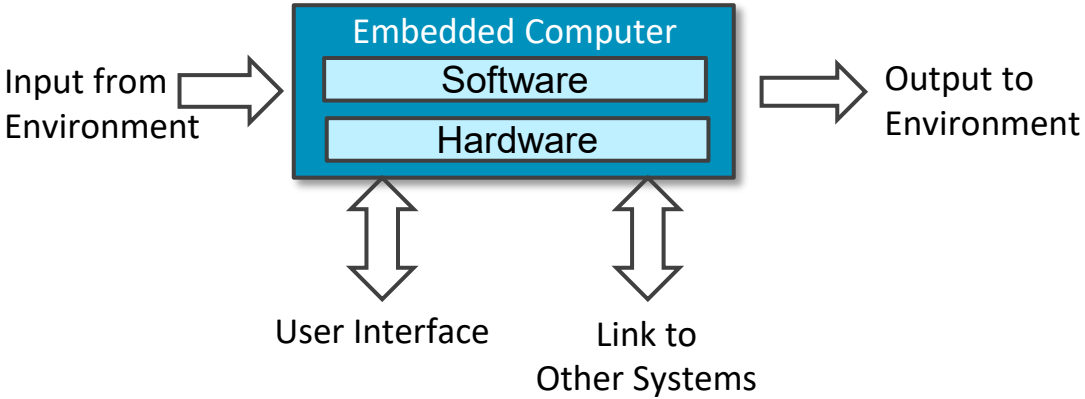
- Entire content available on the lecture website
- Lecture
  - Content delivered on slides
- Codelabs
  - Guided, hands-on coding
  - Some parts may be hidden at first, with solution made available after two-three weeks
- Exercises
  - Addressing specific problems
  - Solutions made available after two-three weeks
- Project
  - To be implemented based on codelabs and exercises
  - Implemented in 2-3 phases, delivered on GitHub with possible issues to be fixed in each phase

Introduction to Embedded Systems

- What is an embedded system?
  - Application-specific computer system
  - Interacting with its environment
  - Built into a larger system
  - Often with real-time computing constraints
- What is the motivation for building an embedded system?
  - Better performance
  - More functions and features
  - Lower cost e.g. through automation
  - More dependable
  - Lower power

Introduction to Embedded Systems

Macroscopic view on a device level



Microscopic view on a functionality level

Applications for Embedded Systems

- Closed-loop control system
  - Monitor and (pre)process a system state, adjust an output to maintain a desired set point (temperature, speed, direction, etc.)
  - Edge (pre-)processing: remove noise, select desired signal features
- Sequencing
  - Step through different stages based on environment and system
- Communications and networking
  - Exchange information reliably and quickly
- Part of a larger system
  - Taking over very specialized functions as part of a larger system, e.g. fault handling, handling networking

Example of Embedded System: Bike Computer

Functions:
- Speed measurement
- Distance measurement

Constraints:
- Size
- Cost
- Power and energy
- Weight

Inputs:
- Wheel rotation indicator
- Mode key

Output:
- Liquid crystal display

Use low-performance microcontroller:
- 9-bit, 10 MIPS

Input:
Wheel rotation
Mode key

Output:
Display speed and distance

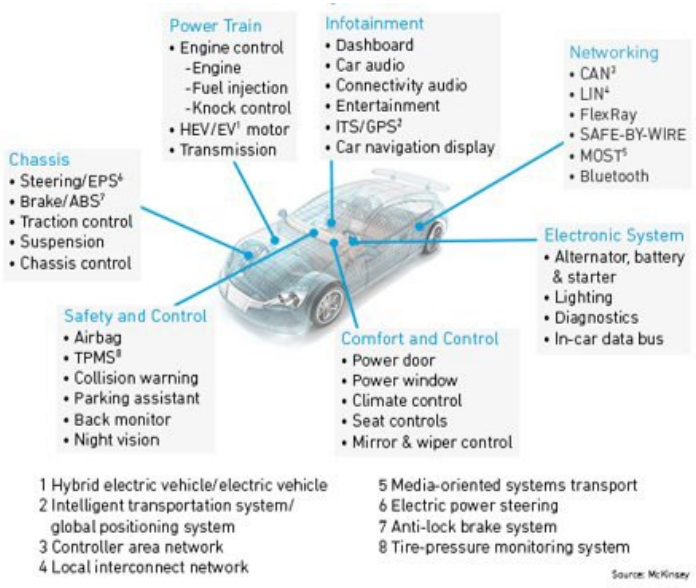
Another Example: Gasoline Automobile Engine Control Unit

Functions:

- Fuel injection
- Air intake setting
- Spark timing
- Exhaust gas circulation
- Electronic throttle control

Many inputs and outputs:

- Discrete sensors and actuators
- Network interface to rest of car



Constraints:

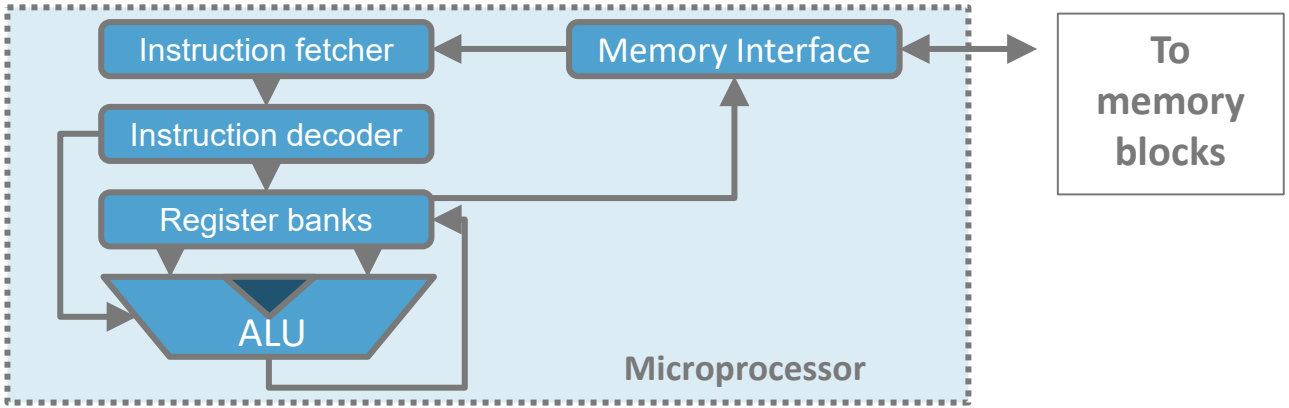
- Reliability in harsh environment
- Cost
- Size

Use high-performance microcontroller:

- E.g. 32-bit, 3 MB flash memory, 50-300 MHz

An Embedded System is more than a Processor

- A microprocessor (CPU) is defined as a processor core that supports instruction fetching, decoding and executing.
  - It can be used for general-purpose computing
  - But it needs to be supported with memory and inputs/outputs (I/O) !



Attributes of Embedded Systems

- Interfacing with larger systems and environments
  - Analog signals for reading sensors
    - Typically use a voltage to represent a physical value
  - Power electronics for driving motors, solenoids
  - Digital interfaces for communicating with other digital devices
    - Simple – switches
    - Complex – displays
- Concurrent and reactive behaviors
  - Must respond to sequences and combinations of events
  - Real-time systems have deadlines on responses
  - Typically must perform multiple separate activities concurrently

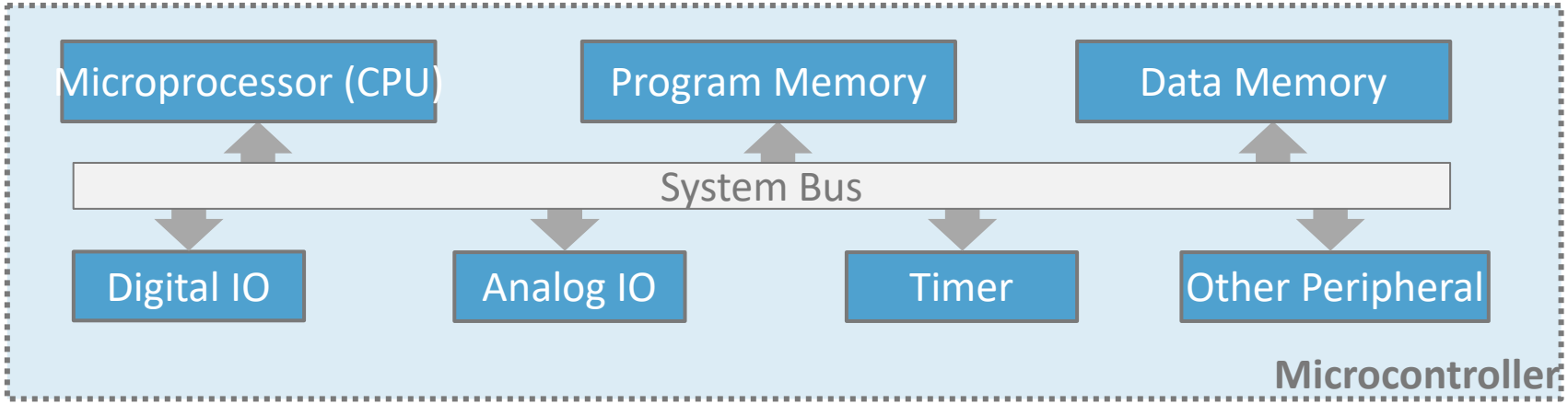
Attributes of Embedded Systems

- Fault handling
  - Many systems must operate independently for long periods of time
  - Requires them to handle faults without crashing
  - Often, fault-handling code is larger and more complex than the normal-case code

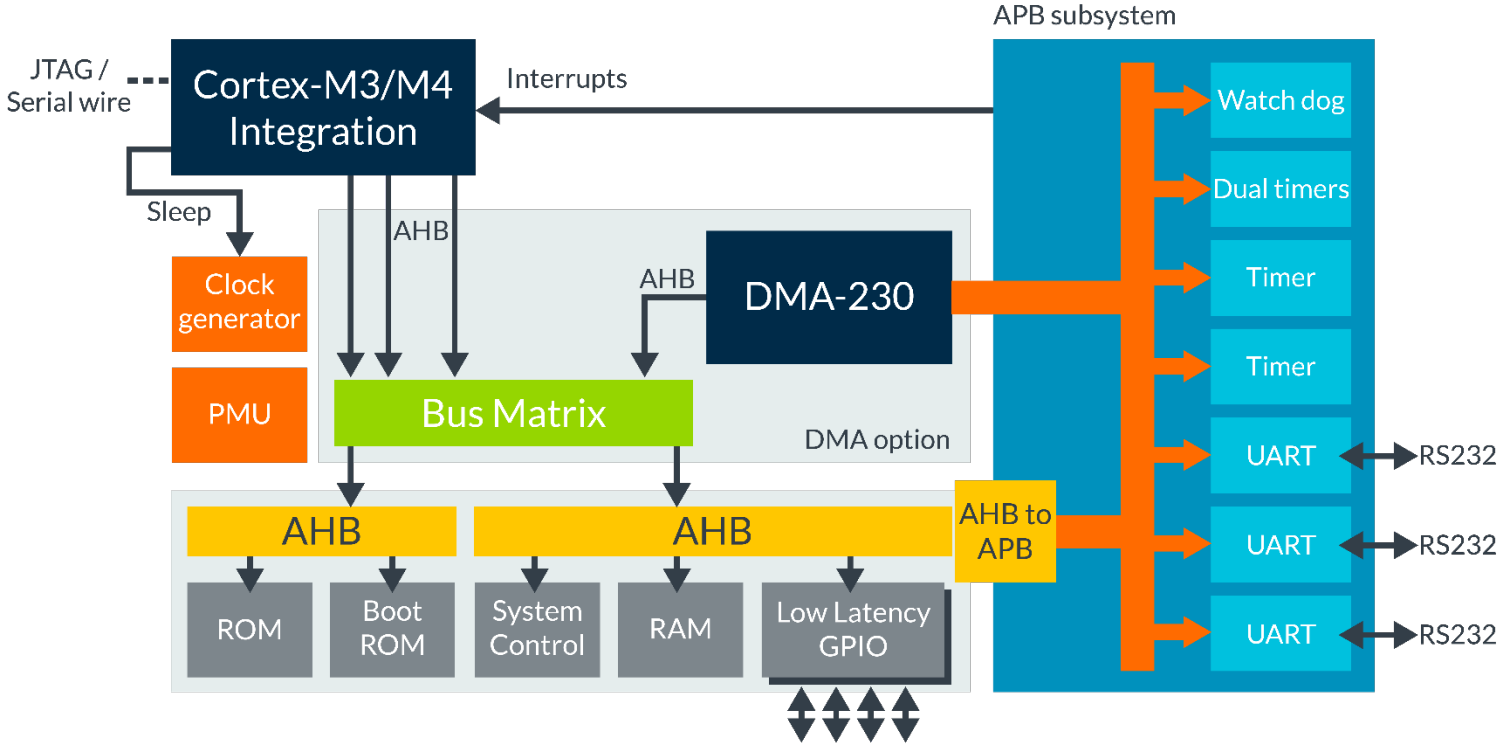
- Diagnostics
  - Help service personnel determine problems quickly

From a Processor to an Embedded System

- Embedded systems are often built using microcontrollers (MCU)
  - Typically has a single processor core
  - Has memory blocks, digital and analog IOs, other peripherals



Example of an Arm M4-MCU Architecture



Source: https://developer.arm.com/ip-products/subsystem/corstone-foundation-ip/cortex-m-system-design-kit

Options for Building Embedded Systems

	Implementation	Design Cost	Unit Cost	Upgrades & Bug Fixes	Size	Weight	Power	System Speed
Dedicated Hardware	Discrete logic	low	mid	hard	large	high	?	very fast
	ASIC	high ($500K/ mask set)	very low	hard	tiny – 1 die	very low	low	extremely fast
	Programmable logic – FPGA, PLD	low to mid	mid	easy	small	low	medium to high	very fast
Software Running on Generic Hardware	Microprocessor + memory + peripherals	low to mid	mid	easy	small to medium	low to moderate	medium	moderate
	Microcontroller (int. memory & peripherals)	low	mid to low	easy	small	low	medium	slow to moderate
	Embedded PC	low	high	easy	medium	moderate to high	medium to high	fast

Microcontroller based embedded system

Benefits of Microcontroller-based Embedded Systems

- Greater performance and efficiency
  - Software makes it possible to provide sophisticated control
- Lower costs for mixed signal-processing systems
  - Less expensive components can be used
  - Overall costs reduced (manufacturing, operating and maintenance)
- More features
  - May not be possible or practical with other approaches
- Better dependability
  - Adaptive system which can compensate for failures
  - Better diagnostics to improve repair time

Constraints of Microcontroller-based Embedded Systems

- Microcontrollers used (rather than microprocessors)
  - Include peripherals to interface with other devices – is done in a specific way by each manufacturer
  - On-chip RAM, ROM reduces circuit board complexity and cost
- Programming language
  - Programmed in the C language rather than Java (resulting in smaller and faster code – less expensive MCU)
  - Some performance-critical code may be in assembly language
- Operating system
  - Typically no OS used, but instead a simple scheduler
  - If OS is used, it is likely to be a lean RTOS

As a summary, why Microcontroller-based Embedded Systems?

- In most embedded systems, MCUs are the best solution as they offer:
  - Low development and manufacturing cost
  - Easy porting and updating
  - Light footprint
  - Relatively low power consumption
  - Satisfactory performance for low-end products
- In our lab sessions, we will learn some fundamentals of developing for embedded systems with a MCU-based prototyping platform, using the Mbed platform that contains a RTOS

Internet of Things (IoT) and Embedded Systems

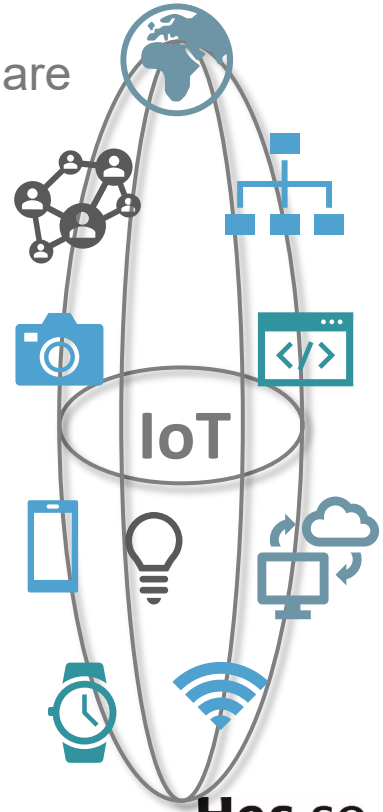
- IoT generally refers to a world in which a large range of objects are addressable via a network

Why IoT?

- Items can have more functionality and become more intelligent
- Items can be managed more easily
- More information becomes available

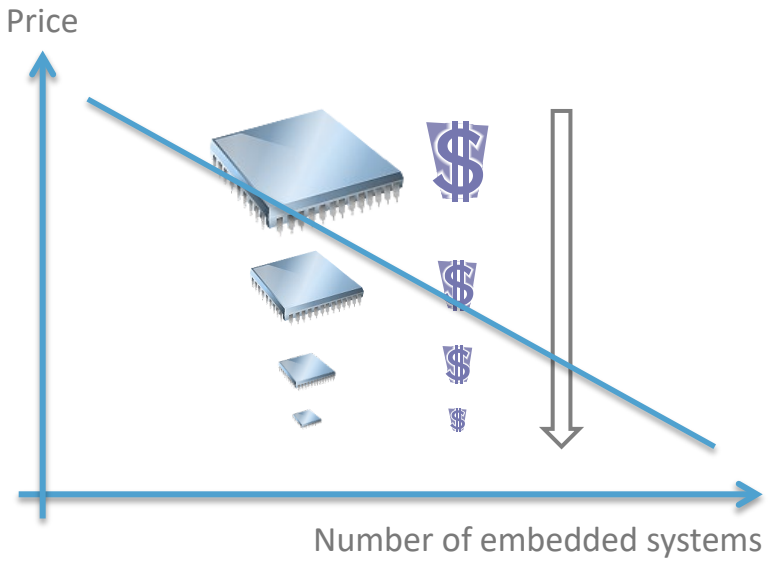
Objects can include:

- Smart buildings and home appliances
  - Fridges, TVs, cookers
- Civil engineering structures
  - Bridges, railways
- Wearable devices
  - Smart watches, glasses
- Medical devices
  - Smart inhaler, embedded pills



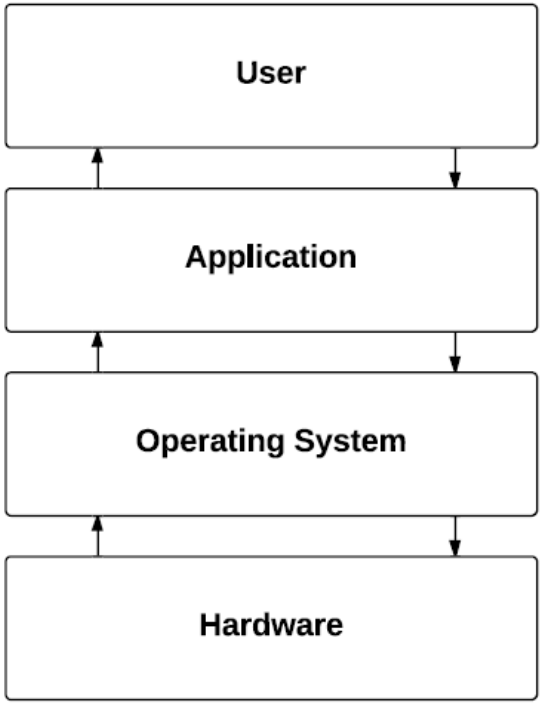
Internet of Things: Why Now?

- Embedded chips are becoming:
  - Cheaper
  - Smaller
  - Lower power
- Energy harvesting
- Communication is becoming faster and more efficient



Embedded Systems and OS

- Should we use an OS for programming embedded systems?
- An OS provides an abstraction of the Hardware
  - Hardware is detailed and specific to every manufacturer, e.g. for MCUs.
  - Manipulating hardware requires not only programming knowledge, but also understanding of the hardware.
  - Should the programmer have to care about the detailed of each hardware?
  - She/he can be more productive by using an abstraction layer



Program structures of Embedded Systems

- It is possible to implement everything in a long sequential infinite loop (super-loop model)
  - Implies a lot of shortcomings
- Improving the structure with an event-driven model
  - Instead of continuously checking for inputs, take actions in response to events
  - Since many inputs are unpredictable, this model allows the main program to wait for any event to occur and take action when it occurs.
  - But events can have different priorities and the system needs to provide a solution for handling priorities.
- For dealing with event priorities, implement event handlers as independent execution entities called processes or tasks.

Embedded systems and OS components

- Process/Task/Thread management
  - How to run a program?
  - How to allocate resources?
  - How to schedule and synchronize tasks?
- Memory management
  - Memory allocation
  - Protection
  - Virtual memory
- File systems
  - Secondary storage
- I/O
  - Device Drivers
- Network
- Security

Embedded Systems and RTOS

- Embedded systems must often satisfy timing constraints.
- Two types:
  - Hard real-time: ensures the critical tasks are to be completed on time.
  - Soft real-time: if the deadline is not met, it is still worth finishing the task.
- Key design requirements for OS in embedded systems:
  - Predictability and determinism
  - Speed
  - Responsiveness
  - Fail-safety
- RTOS and EOS are not exactly the same, but most EOSs are RTOSs

RTOS capabilities

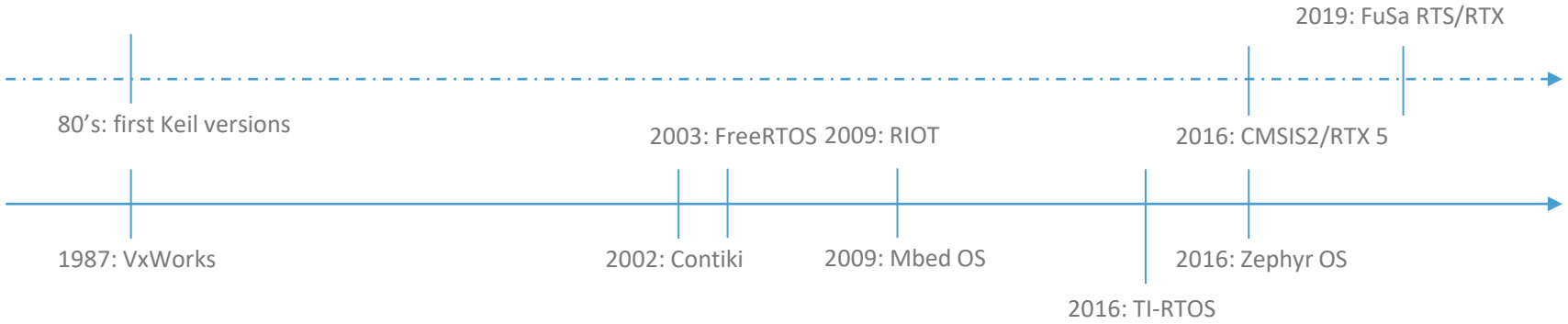
- For meeting the timing requirements, RTOS are usually designed with the following capabilities
  - Minimum interrupt latency
  - Short critical regions
  - Preemptive task scheduling
  - Advanced task scheduling algorithm

RTOS overview

- RTOS are designed to provide only limited functionalities intended for specialized environments:
  - Much simpler than general purpose OS
  - Usually all tasks run in the same address space
  - No separate kernel and user modes
  - Only limited file systems, UI or other functionalities
  - Due to the simplicity, easier to develop
- Large number of existing RTOS
- RTOS development is accelerated by the development of the IoT
  - Many OS are targeting connected resource-constrained devices for IoT applications

A Short RTOS History

- Wikipedia lists over 50 different RTOSes !
- First differentiate themselves on license models and supported platforms
  - Differences in the ecosystem (from kernel only to OS with many middleware components)



Mbed OS overview

- What is Mbed OS?
  - A platform used for the easy prototyping and development of applications and systems based on Arm Cortex-M-based microcontrollers, typically for use in the world of the Internet of Things
- The Mbed OS platform provides:
  - Open software libraries
  - Open hardware designs
  - Open offline/online tools for professional rapid prototyping of products based on Arm-based microcontrollers

Mbed OS overview

- The Mbed platform includes:
  - Mbed Operating System (Mbed OS)
  - Libraries, RTOS core, HAL, API, and more
  - A microcontroller Hardware Development Kit (HDK) and supported development boards
  - Integrated Development Environment (IDE), including an online compiler and online developer collaboration tools

Mbed OS overview

Mbed OS - Security

- Mbed provides two security-focused embedded building blocks:
  - Arm Mbed TLS
  - Secure Partition Manager (SPM)
- Mbed TLS is a protocol for securing communication channels between devices and servers or gateways
- The secure partition manager is responsible for:
  - Isolating software within partitions
  - Managing the execution of software within partitions
  - Providing Inter-Process Communication (IPC) between partitions

Mbed OS - Connectivity

- Mbed OS supports a number of connectivity protocols
  - Paired with Pelion Device Management to provide full support for a range of communication options
- Connectivity technologies include:
  - NarrowBand-IoT (NB-IoT)
  - Bluetooth Low Energy (BLE)
  - 6LoWPAN
  - Thread

Mbed OS – Development tools

Mbed Studio

Mbed Online Compiler

Mbed CLI

Mbed OS – Mbed Studio

- Integrated development environment (IDE) for Mbed OS 5/6 applications
  - Includes everything required to create, compile and debug Mbed programs
  - Automatically detects connected Mbed enabled boards
  - Quick development for specific targets
  - Flashes code directly to connected platform
  - Provides debug session for debugging and profiling the target board
- Mbed Studio is also available as an online environment

Mbed OS – Mbed CLI

- Arm Mbed CLI is a command-line tool packaged as ‘mbed-cli’ and based on Python.
- Enables Git and Mercurial-based version control, along with dependency management, code publishing, support for remotely hosted repositories, and use of the Arm Mbed OS build system.
- Can be used in combination with Mbed Studio

Mbed OS - Testing

- The Mbed platform offers a number of tools that support testing of your Mbed code
- Greentea
  - Automated testing tool for Arm Mbed OS development
  - Pair with 'UNITY' and 'utest' frameworks
  - Greentea
- Icetea
  - Automated testing tool for Arm Mbed OS development
  - Typically used for local development and automation in a continuous integration environment
  - Icetea
- Process of flashing boards, running the tests, and generating reports is automated by the test system

Mbed OS – Mbed Enabled Platforms

- The Arm® Mbed Enabled™ program outlines a set of functionality and requirements that must be met in order to become “Mbed Enabled”. This can cover development boards, modules, components, and interfaces
  - This benefits developers as they are assured that the platforms they choose to work with can perform certain functions/provide certain performance
  - It is also beneficial to the vendors as it allows their products more exposure when certified, and enables their product to become more familiar with developers in the Mbed eco-system
- We will use a STM Mbed Enabled™ platform

Mbed OS – Why C+?

- **Advantages:**
  - Higher productivity (less development time)
  - Portability across devices
  - Resulting code that is easier code to read and maintain
  - Allows reuse of code
  - Rapid prototyping of applications
- **Disadvantages:**
  - Less optimized code
  - Additional translation time for source to machine code
  - Another level of abstraction to deal with

- **Advantages:**
  - More optimized code and memory efficient
  - Less translation time for source to machine code
  - Directly talk to hardware
- **Disadvantages:**
  - Less portability from one device to another
  - Resulting code is more difficult for others to read, reuse, and maintain
  - Low productivity

Codelabs

- Link to all codelabs for this lecture

  Codelabs for TSM_AdvEmbSof -> follow the codelabs tab

- Start developing with Mbed OS and improve your C++ skills

  Getting started with Mbed OS

  C++ basics

- Blinky using low-level vs. high-level programming

  High-level vs. Low-level programming