MASTER OF SCIENCE
IN ENGINEERING

MA_EmbReal

Introduction

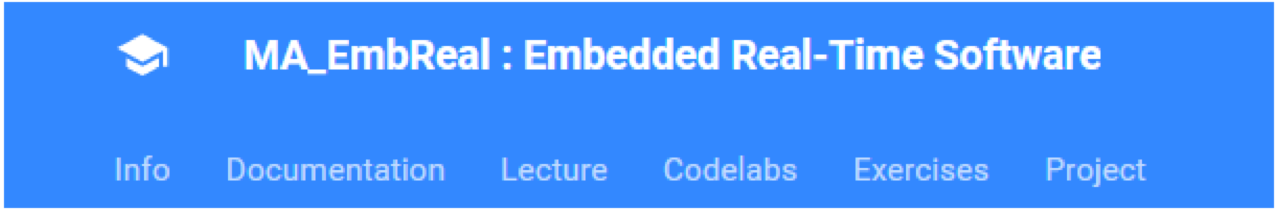
Version: 1.2

Serge Ayer – Luca Haab | 18.02.2025 | Cours MSE

Some administrative matters

- Lecture schedule
  - 11h15-12h00 + 12h05-12h50 + 12h55-13h40
- Resources
  - Site: https://embreal.isc.heia-fr.ch
  - Development kit + various software
- Project
  - Along the semester, you must deliver the source code of a project in 3 runs.
  - Working in team of 2 students.
  - The project is evaluated after each phase.
  - The total number of points for the project is 100 pts (30/30/40)
  - The grade is calculated as (points/100 * 5 + 1)
- Course grade
  - The project/oral exam grade is 30%/70% of the course grade.

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 weeks
- Exercises
  - Addressing specific problems
  - Solutions made available after a few weeks
- Project
  - To be implemented based on codelabs and exercises
  - Implemented in 3 phases, delivered on GitHub with possible issues to be fixed in each phase

Course Teams’ instance

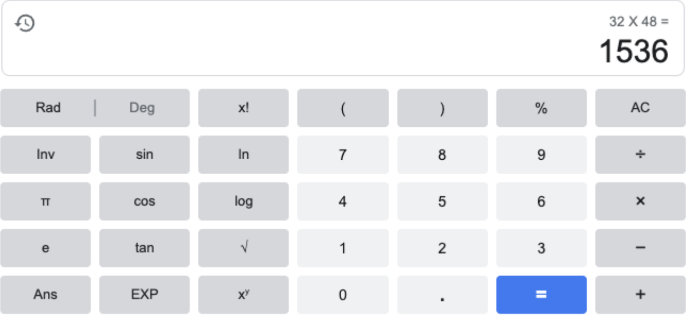
ab1x3yw

A few Questions to You

https://app.wooclap.com/EMBREAL0

What does real-time mean?

Correct behavior



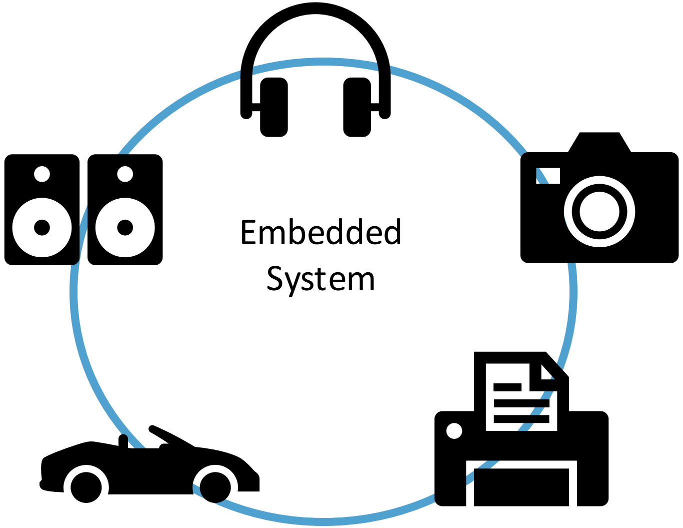
+

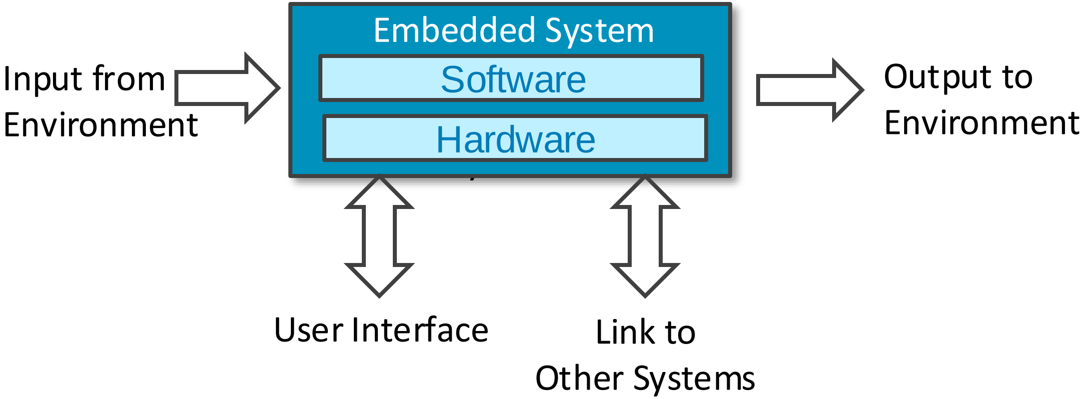
Within expected time

Real-time System Applications

Typically
- run on systems embedded into the system to be controlled
- are mostly implemented on embedded systems

Embedded Systems – Different Views



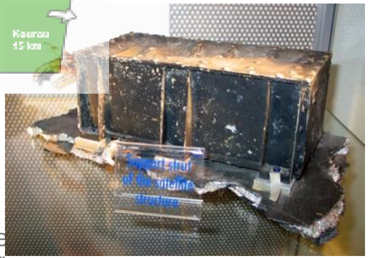


Device: macroscopic

Functionality: microscopic

How to develop real-time systems

- Developing systems with time constraints != developing fast systems
  - Assembly
  - Low-level drivers
  - Manipulating task and interrupt priorities
- Very empirical and not the proper way
  - Tedious coding and difficult to understand
  - Costly and difficult
  - Challenging verification
  - Unpredictable !
- Cause of a high percentage of accidents
  - Famous example: [Ariane 5 Disaster](#) (as silly as an overflow causing for $370m damage)
  - Pictures taken by Phrd - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=6376804 and By De Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=6383059 and https://news.mit.edu/2015/integer-overflow-debugger-outperforms-predecessors-0324



Some famous laws and rules on real-time

- Murphy's General Law

  "If something can go wrong, it will go wrong."
- Naeser’s Law

  "One can make something bomb-proof, not jinx-proof."
- Green’s Law

  "If a system is designed to be tolerant to a set of faults, there will always exist an idiot so skilled to cause a non-tolerated fault."
- Johnson’s First Law

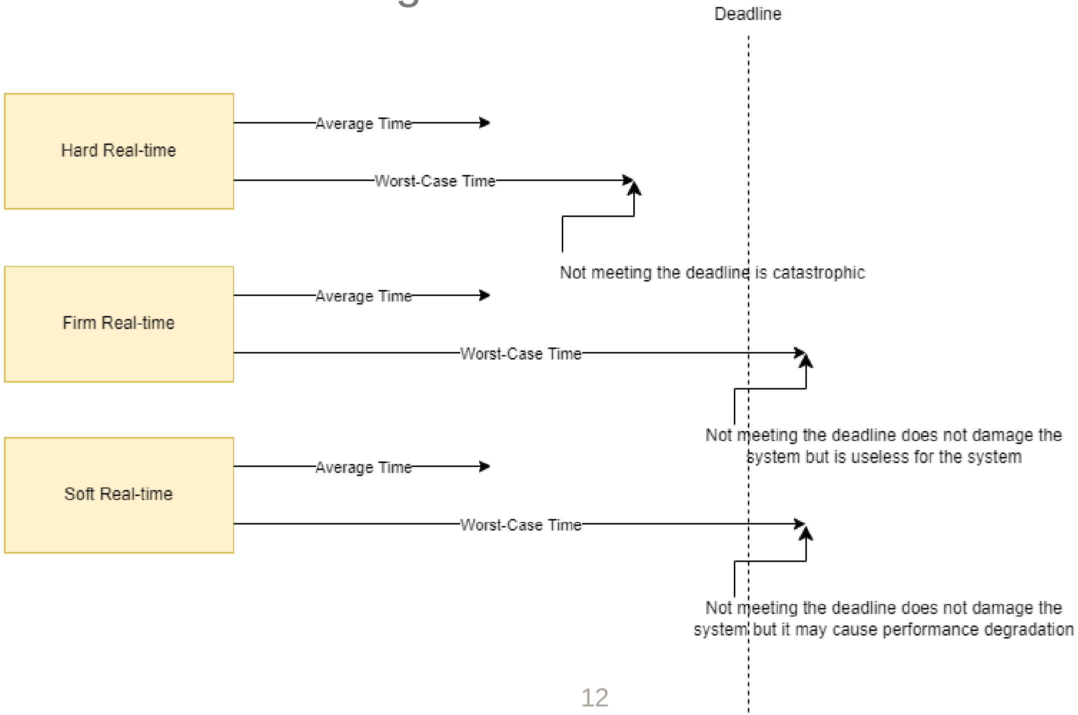
  "If a system stops working, it will do it at the worst possible time."

How to develop real-time systems

- Fast computing systems == minimize the average response time or maximize throughput
- Real-time computing systems == meet timing requirements of each task
  - Even the shortest average response time cannot guarantee the individual timing requirements
  - A methodology is required for making sure that time requirements are met
  - The keyword is: ***Predictability***
- In real-time systems tasks are characterized by a ***deadline***
  - A deadline is the latest time at which the execution of a task shall be completed
  - Producing a correct computation result after the deadline is ***wrong !***

Classification of real-time tasks

- A real-time system may consist of tasks with different timing constraints
- Timing constraints are categorized as:



Tasks with different timing constraints

- Hard real-time:
  - Detection of critical conditions
  - Control of critical system components
  - Action that tightly interact with the environment
  - Sensing of critical data
- Firm real-time:
  - Multi-media/video/audio
  - Sensory data transmission
- Soft real-time:
  - Often related to interaction with user

Needs for programming real-time systems

- Timeliness
  - Kernel mechanisms for time management including a real-time clock
- Predictability
  - Timing requirements must be analyzed and guaranteed
  - Make sure that possible delays are known and bounded
- Efficiency
  - Efficient management of the limited available resources
- Robustness
  - Load / overload must also be considered and managed
- Fault tolerance
  - Behavior shall also be predictable in case of faults
  - Consider also hardware redundancy
- Maintainability
  - Built with modularity and certified components

Achieving predictability

- Guarantee that timing constraints will be met
- Shall be analyzed and guaranteed offline
- But also depends on many factors such as
  - Hardware: CPU and access to memory
  - Kernel (scheduling, synchronization mechanisms, interrupt handling, etc.)
- The above influences
  - Worst-case execution times (WCETs) of tasks
  - Possible delays in the scheduling of tasks
- Hardware, kernels, programming languages to be designed for predictability

Options for Building Real-time 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	sy	medium	moderate to high	medium to high	fast

Microcontroller based embedded system

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

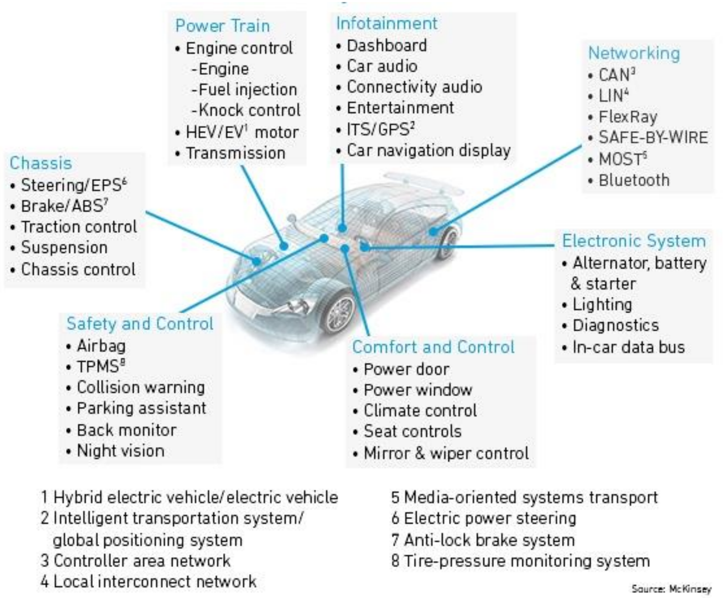
Example of Embedded System (II): Car Combustion Engine Control Unit

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

Inputs and outputs:
- Discrete sensors and actuators
- Network interface to rest of car
- Injectors

Constraints:
- Reliability in harsh environment
- Cost
- Size

Use high-performance microcontroller:
- E.g. 32-bit, 3 MB flash memory, 50-300 MHz

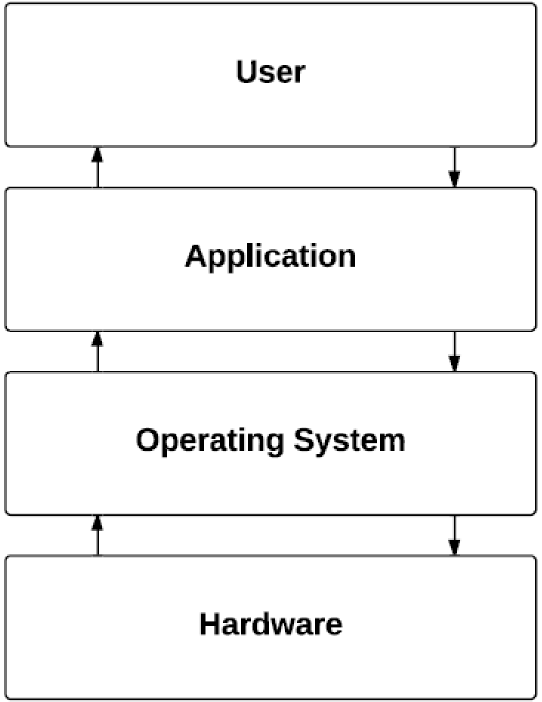


Benefits of Microcontroller-based Embedded Systems

- 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 (aka extensibility)
- Better dependability
  - Adaptive system which can compensate for failures
  - Better diagnostics to improve repair time

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
  - Manipulating hardware requires not only programming knowledge, but also understanding of the hardware.
  - Should a programmer have to care about the details of each hardware?
  - She/he can be more productive by using an abstraction layer



ARM FuSa RTS

- Arm® FuSa RTS is a set of software components qualified for use in safety-critical applications
  - Includes Arm® FuSa RTX OS
  - Includes processor abstraction layers
  - Includes verified C library and compiler for Cortex-M processors
- Arm® FuSa RTS is certified for different safety standards (automotive, industrial, railway, medical)
- It supports and utilizes features of the Cortex-M0/M3/M4/M7 cores
  - We will use a Cortex-M4 based STM platform, enabled for use with Arm® FuSa RTS

ARM FuSa RTS

- Arm® FuSa RTS includes the following components
  - FuSa RTX RTOS: deterministic real-time operating system
  - FuSa Event Recorder: for collecting execution statistics
  - FuSa CMSIS-Core: independent software interface to processor
  - FuSa C library: subset of the C library, certified for safety-critical applications
- FuSa RTX RTOS
  - Qualified for safety-critical applications
  - Based on RTX RTOS
  - Multi-tasking, priority-based, preemptive scheduling
  - Written in C99 with MISRA C:2012 guidelines applied
  - Small memory footprint
  - Includes a tick-less operation mode for low power mode

ARM FuSa RTX RTOS

- Reliability
  - Time-deterministic interrupt execution
- Safety
  - Separate stacks for RTOS and threads
  - Stack overflow checking
  - Runtime check of kernel objects
- Memory management
  - Memory pools for avoiding memory fragmentation
  - Static memory allocation for kernel objects
- RTOS-aware debugging
  - RTOS events recording
  - Stack usage
  - Memory usage of RTX objects
  - Thread statistics