reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino

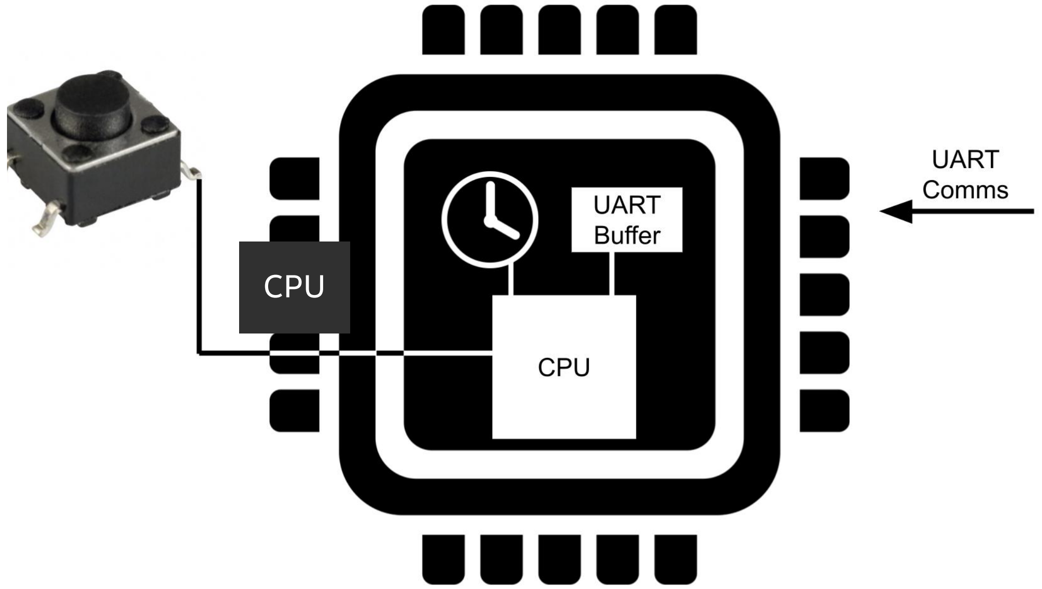
Department of Control and Computer Engineering

INTERRUPTS AND FREERTOS

STEFANO DI CARLO

HARDWARE INTERRUPTS

- Hardware interrupts are an important part of many embedded systems.
- They allow events to occur asynchronously (not as part of any executing program) and notify the CPU that it should take some action.
- These types of interrupts can cause the CPU to stop whatever it was doing and execute some other function, known as an "interrupt service routine" (ISR).
- Such hardware interrupts can include things like button presses (input pin voltage change), a hardware timer expiring, or a communication buffer being filled.

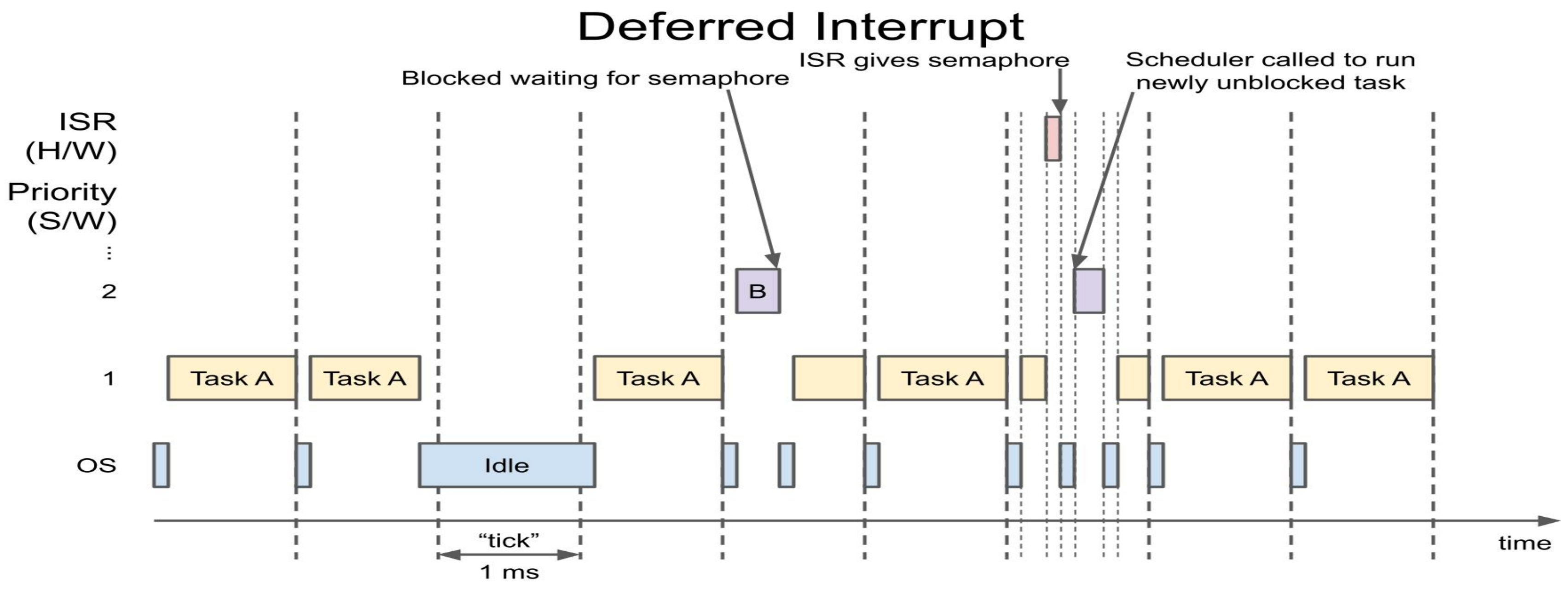


HARDWARE INTERRUPTS

- In FreeRTOS hardware interrupts have a higher priority than any task (unless we purposely disable hardware interrupts).
- When working with hardware interrupts, there are few things to keep in mind:
  1. An ISR should never block itself. ISRs are not executed as part of a task, and as a result, cannot be blocked. You must only use FreeRTOS function calls that end in *FromISR inside an ISR. You cannot wait for a queue, mutex, or semaphore.
  2. You must keep ISRs as short as possible to avoid delays in your waiting tasks!
  3. If a variable (such as a global) is updated inside an ISR, you likely need to declare it with the "volatile" qualifier. Marking a variable as "volatile" informs the compiler that it can change outside the current thread, preventing the compiler from optimizing it away due to assumed inactivity.
  4. One of the easiest ways to synchronize a task to an ISR is to use what's known as a "deferred interrupt." Here, we defer processing the data captured inside the ISR to another task. Whenever such data has been captured, we can give a semaphore (or send a task notification) to let some other task know that data is ready for processing.

DEFERRED INTERRUPTS

- Task B is blocked waiting for a semaphore. Only the ISR gives the semaphore. So, as soon as the ISR runs (and, say, collects some data from a sensor), it gives the semaphore. When the ISR has finished executing, Task B is immediately unblocked and runs to process the newly collected data.



TARGET PLATFORM (MPS2)

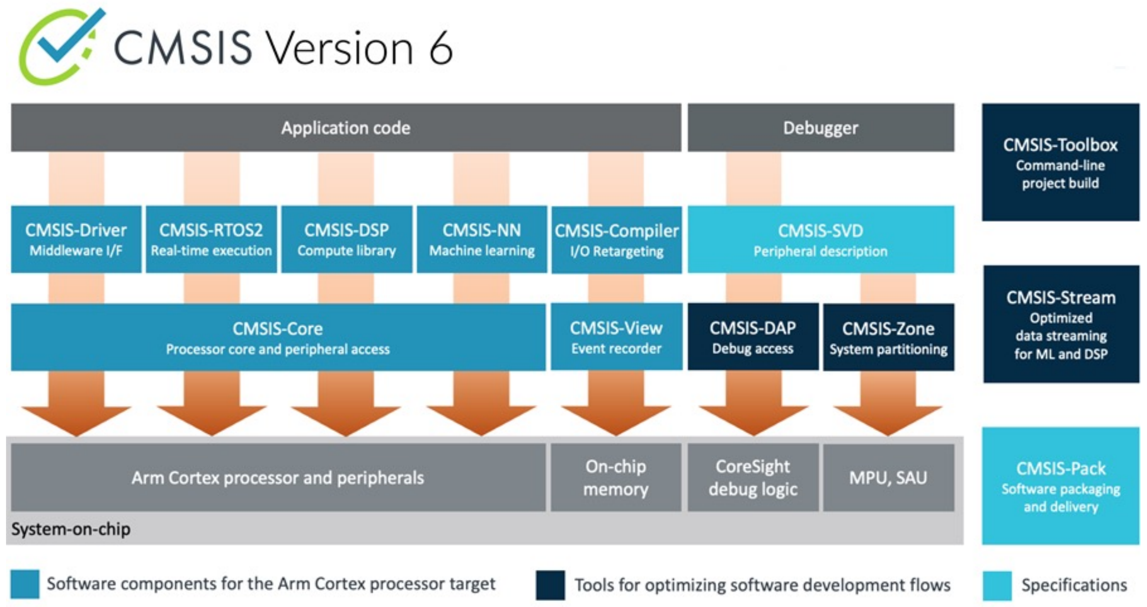
- The ARM MPS2 (Microprocessor System 2) platform is a development board designed by ARM to support the evaluation and prototyping of ARM-based systems.
- It serves as a flexible, scalable platform for testing ARM's Cortex-M and Cortex-A series processors, including their associated peripherals and features.
- The MPS2 platform typically provides hardware for software development, debugging, and performance evaluation.

LET’S WRITE SOME CODE

- https://baltig.polito.it/teaching-material/exercises-caos-and-os/interrupts

COMMON MICROCONTROLLER SOFTWARE INTERFACE STANDARD (CMSIS)

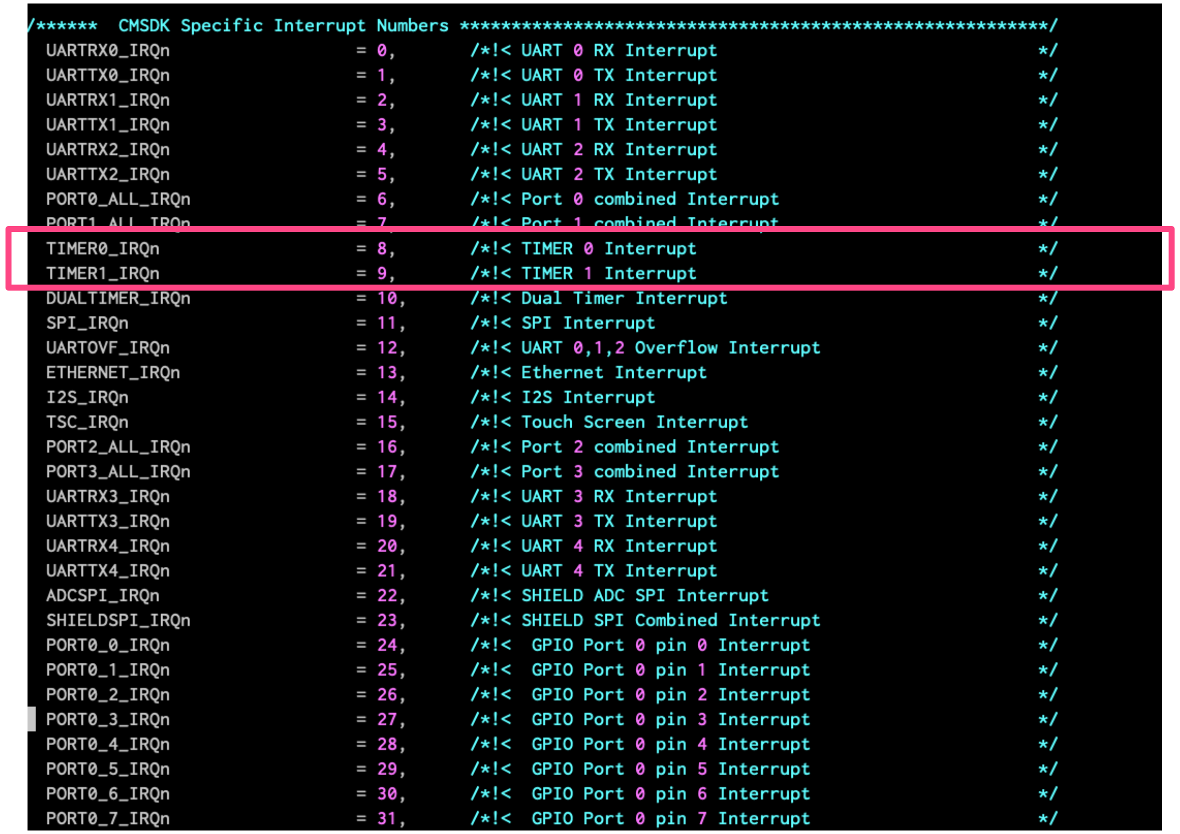
- CMSIS enables consistent device support and simple software interfaces to the processor and its peripherals, simplifying software reuse, reducing the learning curve for microcontroller developers, and reducing the time to market for new devices.



CMSIS Version 6
Application code
Debugger
CMSIS-Toolbox Command-line project build
CMSIS-Driver Middleware I/F
CMSIS-RTOS2 Real-time execution
CMSIS-DSP Compute library
CMSIS-NN Machine learning
CMSIS-Compiler I/O Retargeting
CMSIS-SVD Peripheral description
CMSIS-Stream Optimized data streaming for ML and DSP
CMSIS-Core Processor core and peripheral access
CMSIS-View Event recorder
CMSIS-DAP Debug access
CMSIS-Zone System partitioning
Arm Cortex processor and peripherals
On-chip memory
CoreSight debug logic
MPU, SAU
CMSIS-Pack Software packaging and delivery
System-on-chip
Software components for the Arm Cortex processor target
Tools for optimizing software development flows
Specifications

MPS2 AVAILABLE INTERRUPTS

- The list of available interrupts depends on the target board and is usually specified in the datasheets of the board
- The emulated MPS2 board in QEMU defines the following interrupts
- We will use the two timers TIMER0 and TIMER1 as an example

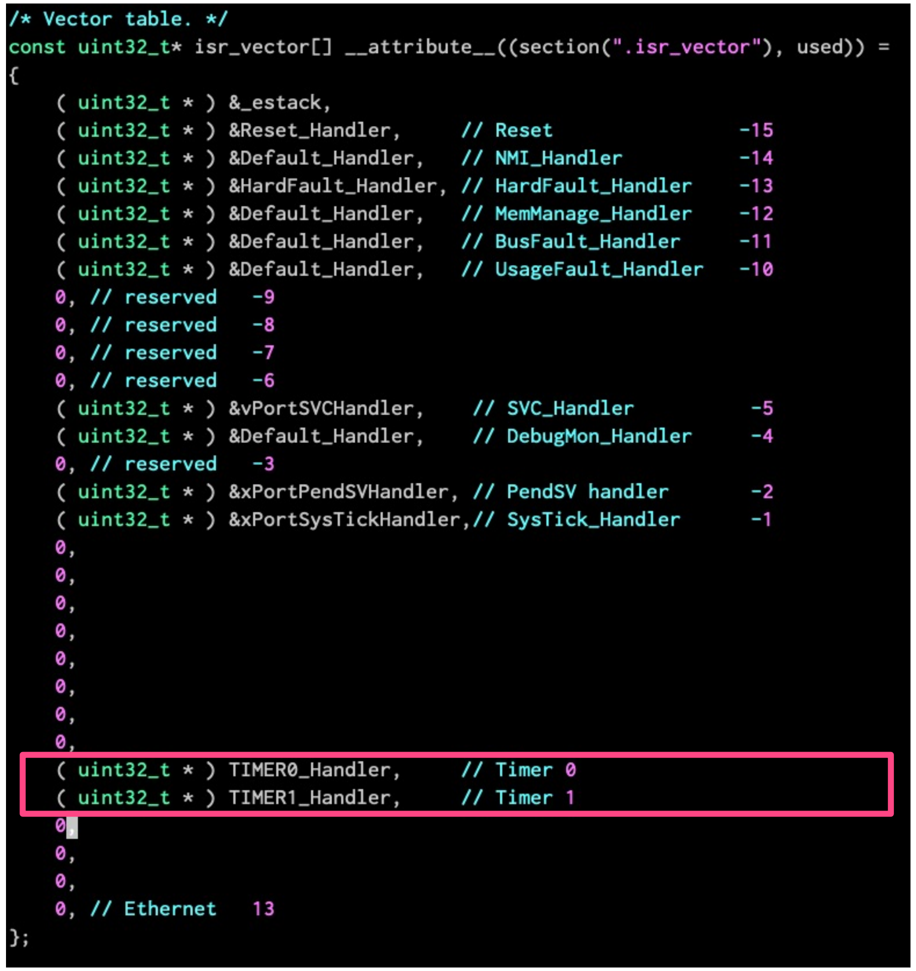


```
/******  CMSDK Specific Interrupt Numbers ************************************************************/
  UARTRX0_IRQn                = 0,      /*!< UART 0 RX Interrupt                               */
  UARTTX0_IRQn                = 1,      /*!< UART 0 TX Interrupt                               */
  UARTRX1_IRQn                = 2,      /*!< UART 1 RX Interrupt                               */
  UARTTX1_IRQn                = 3,      /*!< UART 1 TX Interrupt                               */
  UARTRX2_IRQn                = 4,      /*!< UART 2 RX Interrupt                               */
  UARTTX2_IRQn                = 5,      /*!< UART 2 TX Interrupt                               */
  PORT0_ALL_IRQn              = 6,      /*!< Port 0 combined Interrupt                         */
  PORT1_ALL_IRQn              = 7,      /*!< Port 1 combined Interrupt                         */
  TIMER0_IRQn                 = 8,      /*!< TIMER 0 Interrupt                                 */
  TIMER1_IRQn                 = 9,      /*!< TIMER 1 Interrupt                                 */
  DUALTIMER_IRQn              = 10,     /*!< Dual Timer Interrupt                              */
  SPI_IRQn                    = 11,     /*!< SPI Interrupt                                     */
  UARTOVF_IRQn                = 12,     /*!< UART 0,1,2 Overflow Interrupt                     */
  ETHERNET_IRQn               = 13,     /*!< Ethernet Interrupt                                */
  I2S_IRQn                    = 14,     /*!< I2S Interrupt                                     */
  TSC_IRQn                    = 15,     /*!< Touch Screen Interrupt                            */
  PORT2_ALL_IRQn              = 16,     /*!< Port 2 combined Interrupt                         */
  PORT3_ALL_IRQn              = 17,     /*!< Port 3 combined Interrupt                         */
  UARTRX3_IRQn                = 18,     /*!< UART 3 RX Interrupt                               */
  UARTTX3_IRQn                = 19,     /*!< UART 3 TX Interrupt                               */
  UARTRX4_IRQn                = 20,     /*!< UART 4 RX Interrupt                               */
  UARTTX4_IRQn                = 21,     /*!< UART 4 TX Interrupt                               */
  ADCSPI_IRQn                 = 22,     /*!< SHIELD ADC SPI Interrupt                          */
  SHIELDSPI_IRQn              = 23,     /*!< SHIELD SPI Combined Interrupt                     */
  PORT0_0_IRQn                = 24,     /*!<  GPIO Port 0 pin 0 Interrupt                      */
  PORT0_1_IRQn                = 25,     /*!<  GPIO Port 0 pin 1 Interrupt                      */
  PORT0_2_IRQn                = 26,     /*!<  GPIO Port 0 pin 2 Interrupt                      */
  PORT0_3_IRQn                = 27,     /*!<  GPIO Port 0 pin 3 Interrupt                      */
  PORT0_4_IRQn                = 28,     /*!<  GPIO Port 0 pin 4 Interrupt                      */
  PORT0_5_IRQn                = 29,     /*!<  GPIO Port 0 pin 5 Interrupt                      */
  PORT0_6_IRQn                = 30,     /*!<  GPIO Port 0 pin 6 Interrupt                      */
  PORT0_7_IRQn                = 31,     /*!<  GPIO Port 0 pin 7 Interrupt                      */
```

ISR HANDLERS

- In a FreeRTOS project, the **`isr_vector[]`** (or interrupt vector table) is not actually defined by FreeRTOS itself.
- Instead, it is typically part of the startup code provided by the compiler toolchain or device-specific libraries (such as CMSIS or a manufacturer's SDK).
- The interrupt vector table is set up by the microcontroller's startup file and is usually in assembly or C, depending on the toolchain and microcontroller.

startup_gcc.c



```
/* Vector table. */
const uint32_t* isr_vector[] __attribute__((section(".isr_vector"), used)) =
{
    ( uint32_t * ) &_estack,
    ( uint32_t * ) &Reset_Handler,      // Reset                -15
    ( uint32_t * ) &Default_Handler,    // NMI_Handler          -14
    ( uint32_t * ) &HardFault_Handler,  // HardFault_Handler    -13
    ( uint32_t * ) &Default_Handler,    // MemManage_Handler    -12
    ( uint32_t * ) &Default_Handler,    // BusFault_Handler     -11
    ( uint32_t * ) &Default_Handler,    // UsageFault_Handler   -10
    0, // reserved   -9
    0, // reserved   -8
    0, // reserved   -7
    0, // reserved   -6
    ( uint32_t * ) &vPortSVCHandler,     // SVC_Handler          -5
    ( uint32_t * ) &Default_Handler,     // DebugMon_Handler     -4
    0, // reserved   -3
    ( uint32_t * ) &xPortPendSVHandler, // PendSV handler       -2
    ( uint32_t * ) &xPortSysTickHandler,// SysTick_Handler      -1
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    ( uint32_t * ) TIMER0_Handler,      // Timer 0
    ( uint32_t * ) TIMER1_Handler,      // Timer 1
    0,
    0,
    0,
    0, // Ethernet   13
};
```

smilies

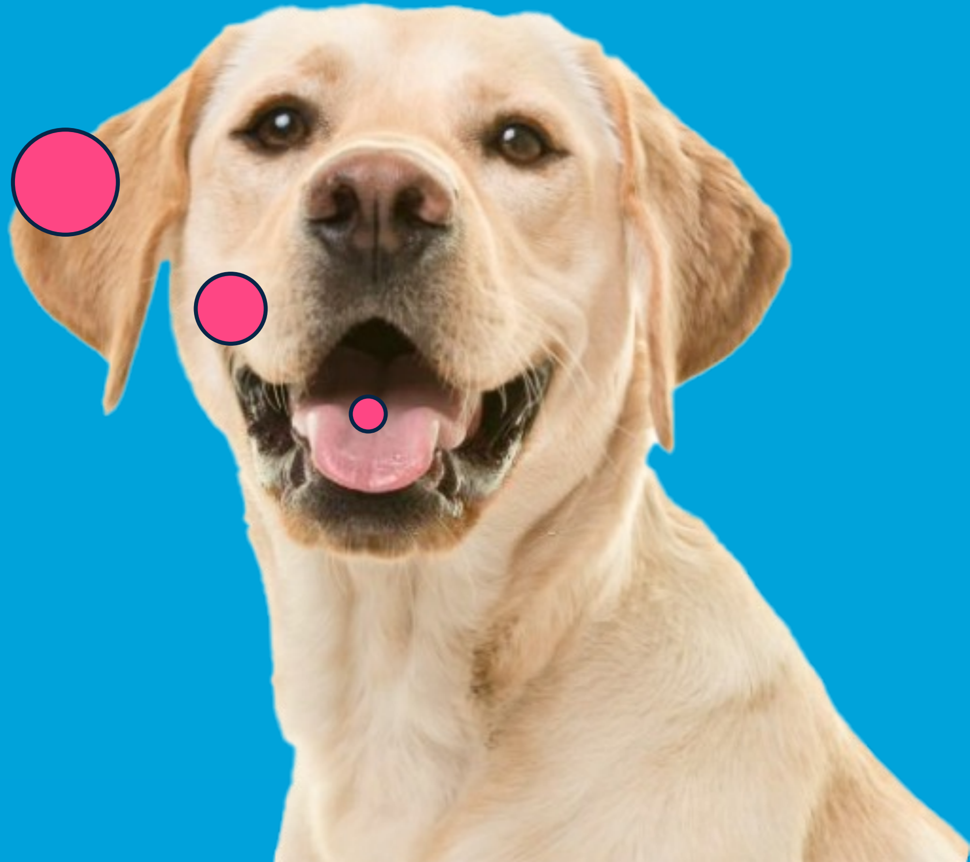
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino

Department of Control and Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!