SMILIES

reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino

Department of Control and Computer Engineering

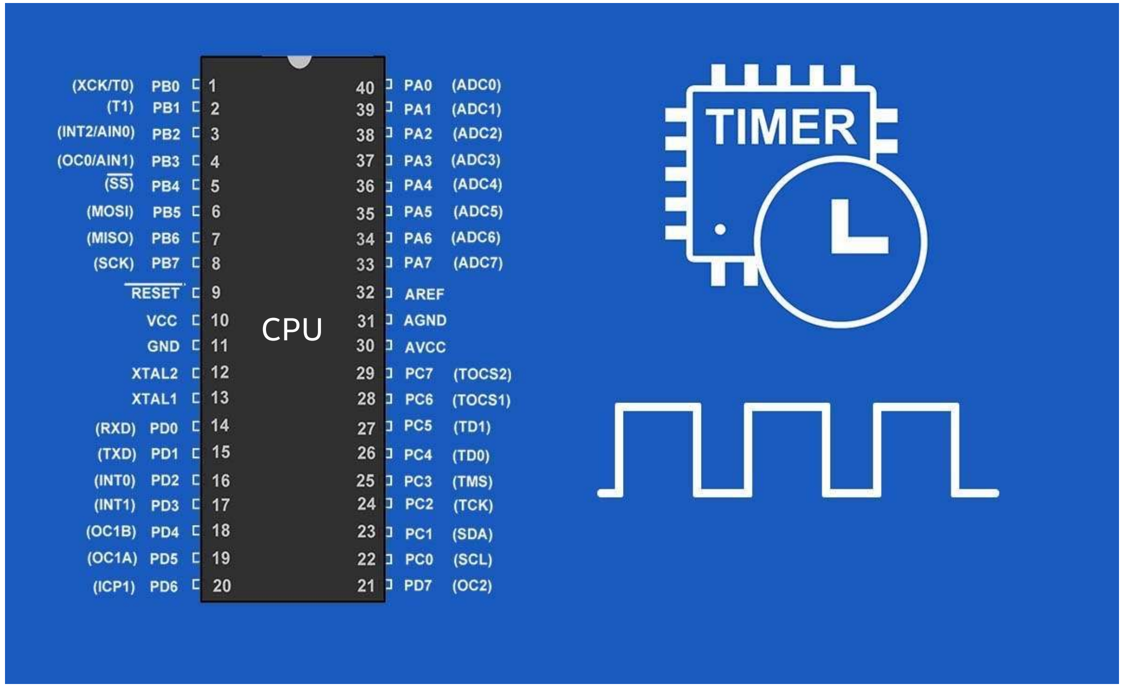
DAUIN

TIMERS AND FREERTOS

STEFANO DI CARLO

HARDWARE TIMER BASICS

- A timer (sometimes referred to as a counter) is a special piece of hardware inside many microcontrollers.
- It counts up or down, depending on the configuration
  - For example, an 8-bit timer will count from 0 to 255.
- Most timers “roll over” once they reach their max value.
  - So, our 8-bit timer would start over again from 0 once it reaches 255.

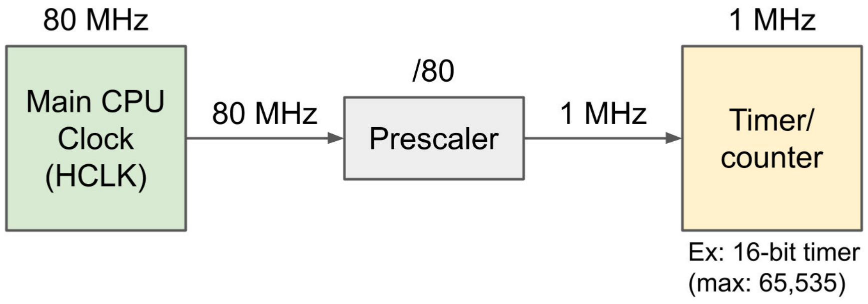


HARDWARE TIMER BASICS

- You can apply a variety of settings to most timers to change the way they function using special function registers inside the microcontroller:
  - Instead of counting to a maximum of 255, you might tell the timer that you want it to roll over at 100 instead.
  - You can often connect other hardware or peripherals inside the microcontroller to the timer, like toggling a specific pin automatically when the timer rolls over.
- Here are some of the common hardware functions you'll see with timers:
  - **Output compare (OC):** toggle a pin when a timer reaches a certain value
  - **Input capture (IC):** measure the number of counts of a timer between events on a pin
  - **Pulse width modulation (PWM):** toggle a pin when a timer reaches a certain value and on rollover. By adjusting the on versus off time (duty cycle), you can effectively control the amount of electrical power going to another device.

PRESCALER

- How fast do timers run?
- It depends on how fast you tell them to run.
- All timers require a clock of some sort.
  - Most will be connected to the microcontroller's main CPU clock (others, like real time clocks, have their own clock sources).
- A timer will tick (increment by one) each time it receives a clock pulse.
- To increase the timer period, we need to use a prescaler, which is a piece of hardware that divides the clock source.



CHOOSING A TIMER

- If you look at the datasheet for your microcontroller, you will often find a section talking about the various timers available.
- For example, here is a table from section 3.24 of our STM32L476 datasheet giving us information about the different general-purpose timers at our disposal.

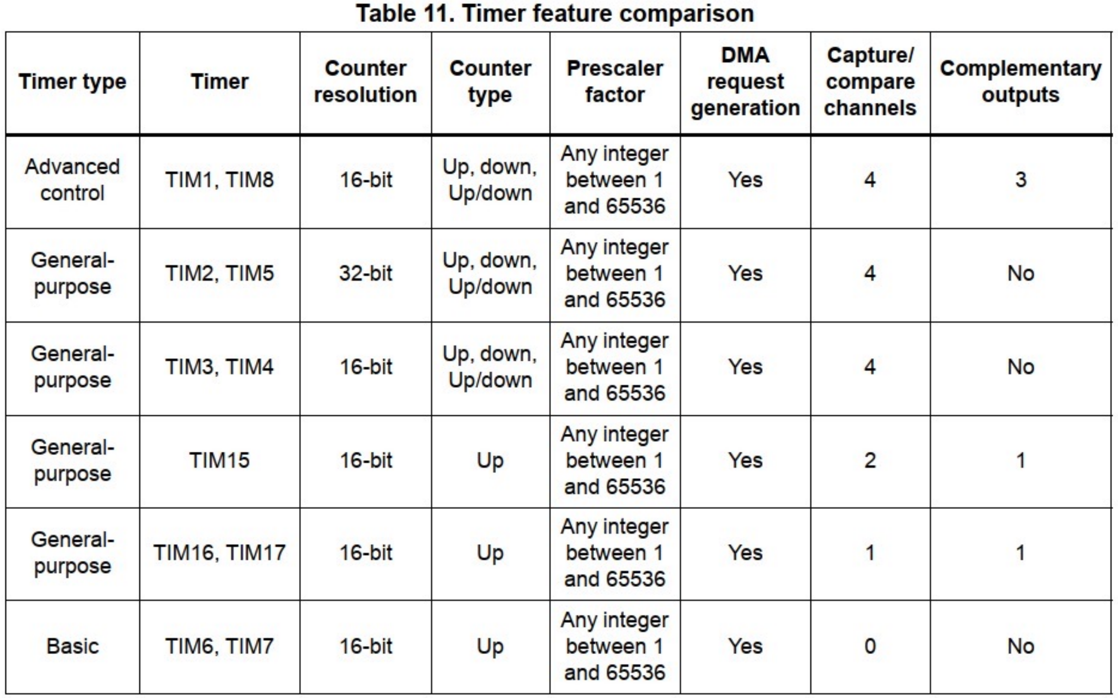
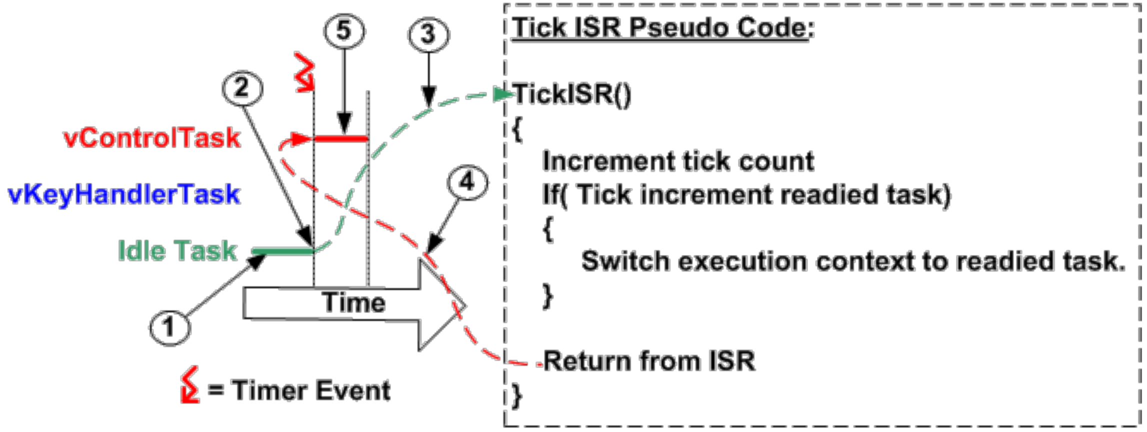


Table 11. Timer feature comparison

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary outputs
Advanced control	TIM1, TIM8	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	3
General-purpose	TIM2, TIM5	32-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM3, TIM4	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	1
General-purpose	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	1
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

THE FreeRTOS TICK

- The FreeRTOS real time kernel measures time using a tick count variable.
- A timer interrupt (the RTOS tick interrupt) increments the tick count with strict temporal accuracy - allowing the real time kernel to measure time to a resolution of the chosen timer interrupt frequency.
- Referring to the numbers in the diagram above:
  - At (1) the RTOS idle task is executing.
  - At (2) the RTOS tick occurs, and control transfers to the tick ISR (3).
  - The RTOS tick ISR makes vControlTask ready to run, and as vControlTask has a higher priority than the RTOS idle task, switches the context to that of vControlTask.

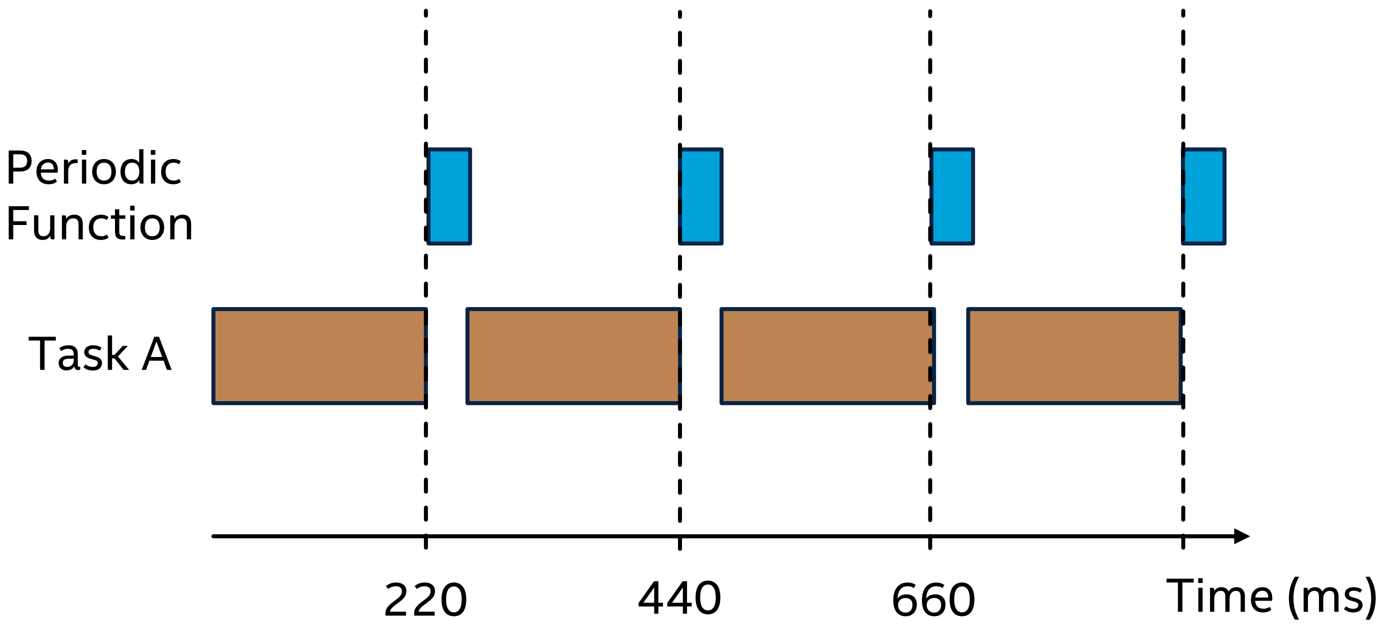


```
Tick ISR Pseudo Code:

TickISR()
{
    Increment tick count
    If( Tick increment readied task)
    {
        Switch execution context to readied task.
    }
    Return from ISR
}
```

  - As the execution context is now that of vControlTask, exiting the ISR (4) returns control to vControlTask, which starts executing (5)
- A context switch occurring in this way is said to be Preemptive, as the interrupted task is preempted without suspending itself voluntarily.

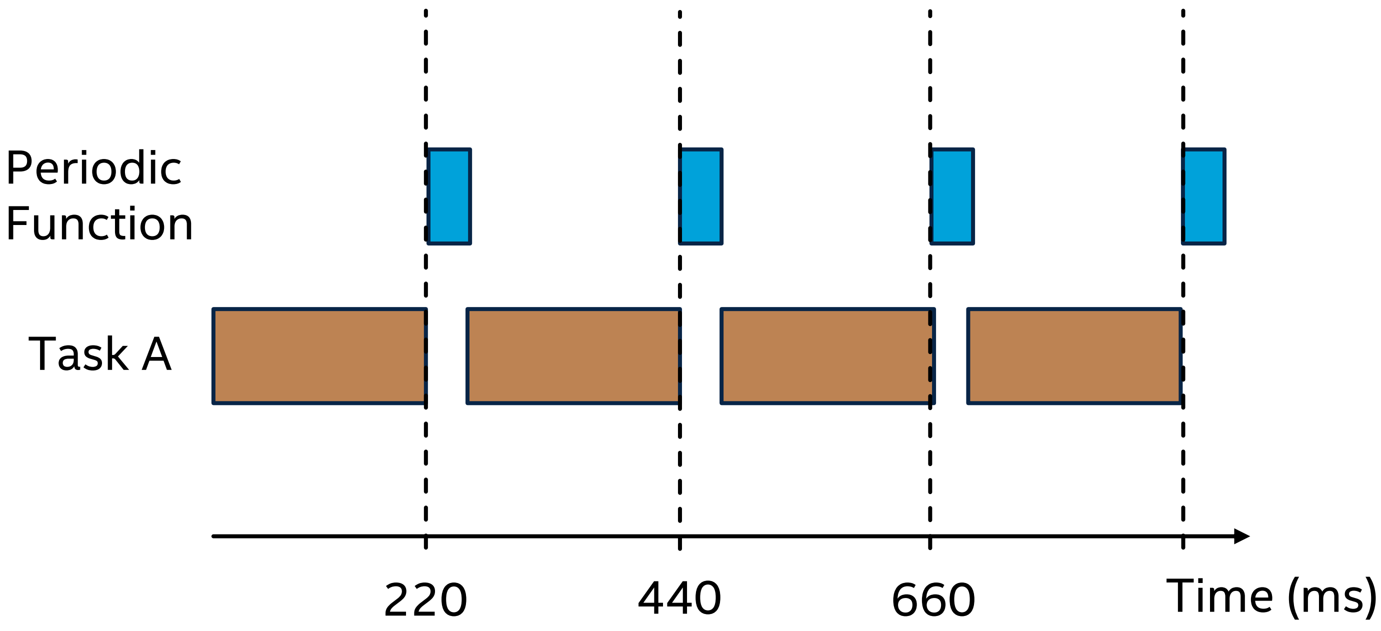
PERFORMING PERIODIC OPERATIONS



- Approach #1
  - New task with **`vTaskDelay()`**
  - `vTaskDelay()` allows us block the currently running task for a set amount of time (given in ticks).
  - Not efficient: a new task may require up to 1KB of additional memory

https://www.freertos.org/Documentation/02-Kernel/04-API-references/02-Task-control/01-vTaskDelay

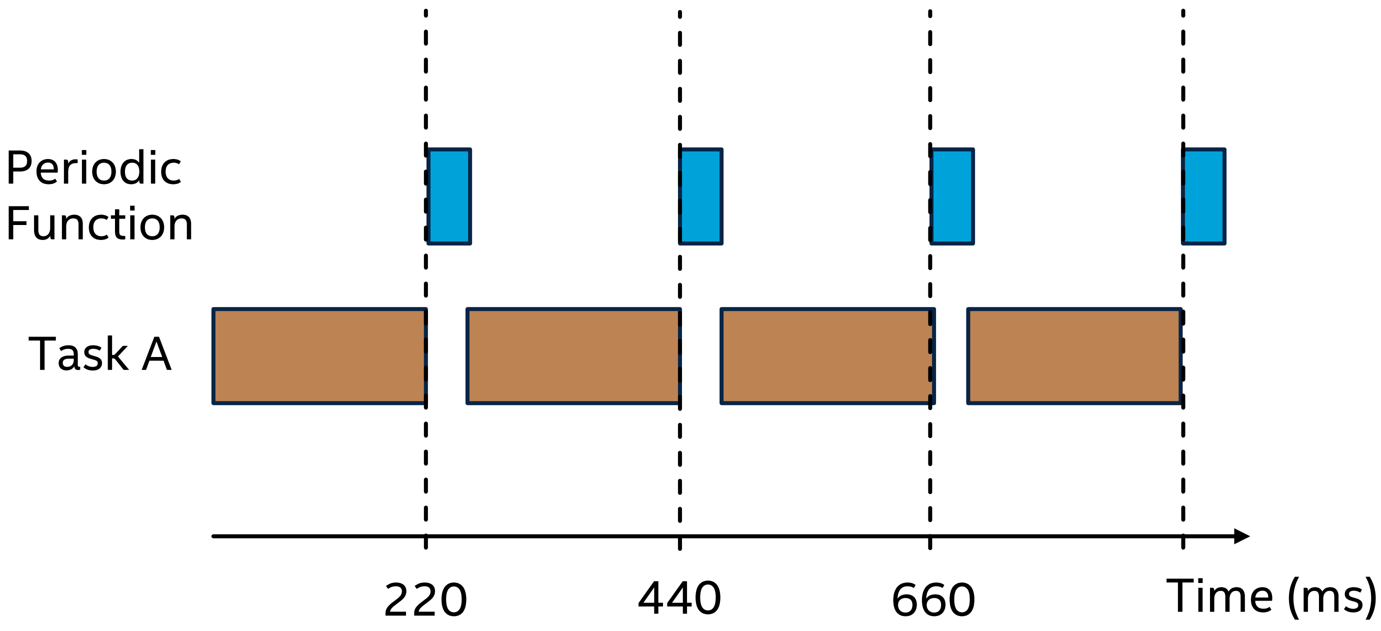
PERFORMING PERIODIC OPERATIONS



- Approach #2
  - Task A with **`xTaskGetTickCount()`**
  - `xTaskGetTickCount()` returns the tick count since the scheduler was started.
  - The tick count can be periodically compared with a given timestamp in task A

https://www.freertos.org/Documentation/02-Kernel/04-API-references/03-Task-utilities/00-Task-utilities#xtaskgettickcount

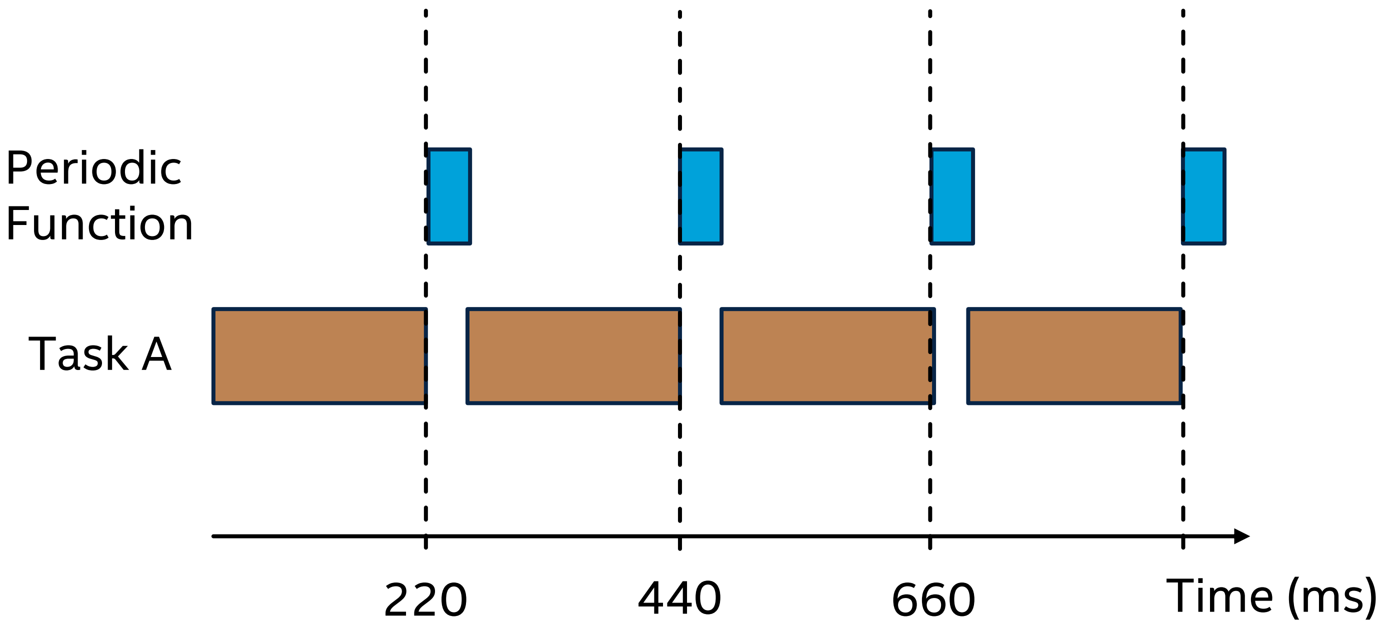
PERFORMING PERIODIC OPERATIONS



- Approach #3
  - Many microcontrollers (and microprocessors) include one or more hardware timers.
  - These can be configured to count up or down and trigger an interrupt service routine (ISR) when they expire (or reach a particular number).
  - They may have higher precision than the tick timer.
  - They are limited
  - Code will not be portable

They will be analyzed in the next lecture talking about interrupts

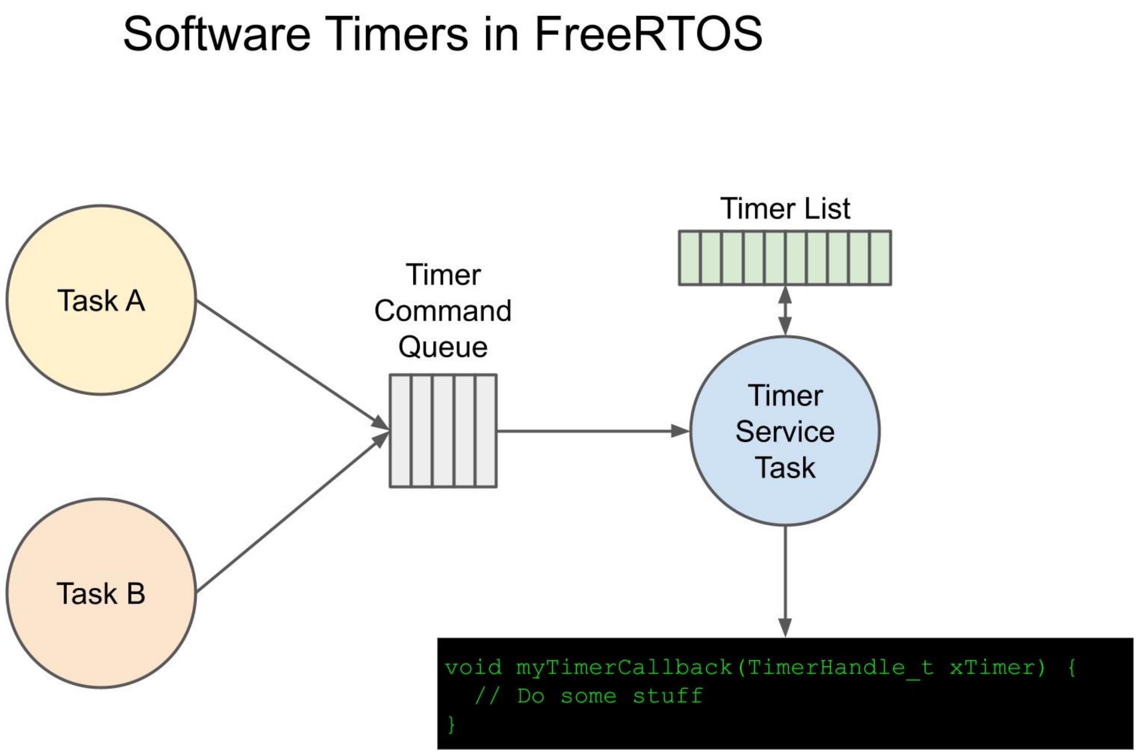
PERFORMING PERIODIC OPERATIONS



- Approach #3
  - FreeRTOS software timers.
  - Built on top of the FreeRTOS tick timer
  - Similar to hardware interrupts but operating at the task level
  - They enable to develop portable applications

SOFTWARE TIMERS IN FreeRTOS

- FreeRTOS offers an API that makes managing these timers much easier (you can read the API documentation here).
- When a timer is created, you assign a function (a "callback function") that is called whenever the timer expires.
- Timers are dependent on the tick timer, which defines their resolution
- Timers can be "one-shot" (executes the callback function once after the timer expires) or "auto-reload" (executes the callback function periodically every time the timer expires).


Software Timers in FreeRTOS

```
void myTimerCallback(TimerHandle_t xTimer) {
  // Do some stuff
}
```

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

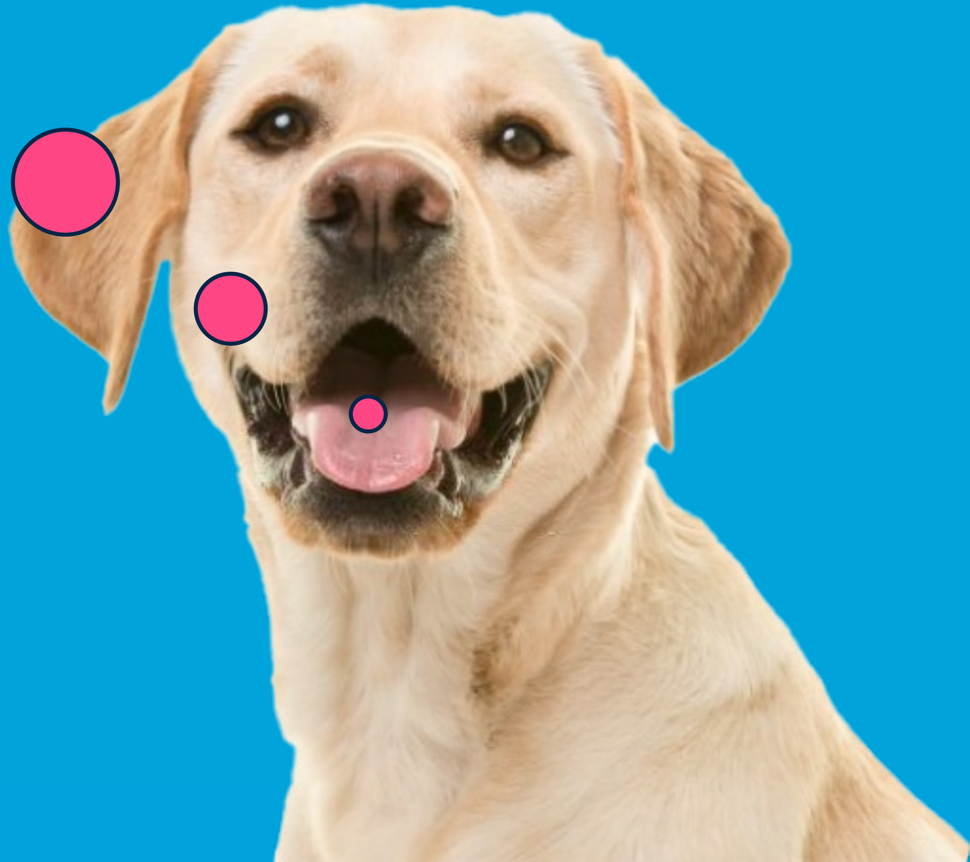
reSilient coMputer archItectures
and LIfE Sciences

Politecnico
di Torino

Department of Control and
Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!