Timers and FreeRTOS

Date	@November 4, 2024

Hardware timer basics

A timer, or a counter, is a special piece of hardware inside many microcontrollers. It can count up or down depending on the configuration and most of them roll over, generating an interrupt signal, once they reach their maximum value.

Applying different settings you can change the way they work, using special function registers inside the microcontroller. You can also connect other hardware or peripherals inside the microcontroller to the timer.

Some of the common hardware functions implemented with timers are:

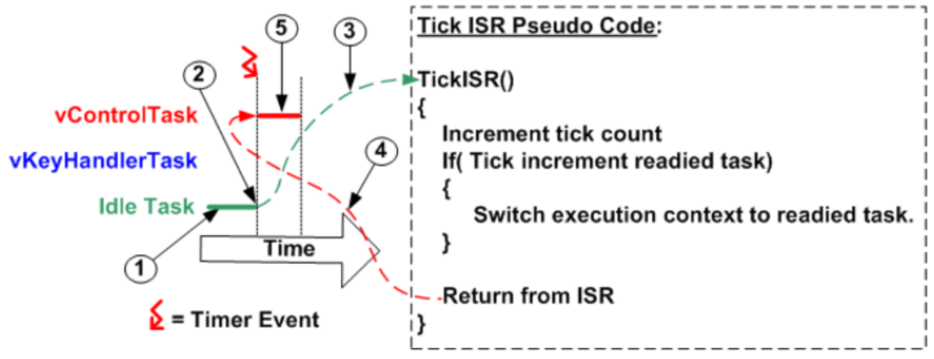
- output compare OC: toggle a pin when a timer reaches a certain value
- input capture IC: measure the number of count of a timer between events on a pin
- pulse width modulation PWM: toggle a pin when a timer reaches a certain value and on rollover

Prescaler

Timers runs as fast as how fast you tell them to run. A timer will tick each time it receives a clock pulse and to increase the time period, we need to use a prescaler, which is a piece of hardware that divides the clock source.

The FreeRTOS tick

the FreeRTOS real time kernel measures time using a tick count variable. A time 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.



Tick ISR Pseudo Code:

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

Looking at the diagram above:

- at the RTOS idle task is executing
- at 2 the RTOS tick occurs, and at 3 control transfers to the tick ISR
- the RTOS tick ISR makes vControlTask ready to run, and as vControlTaskhas a higher priority than the RTOS idle task, switches the context to that of vControlTask
- 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 valid to be preemptive, as the interrupted task is preempted without suspending itself voluntarily.

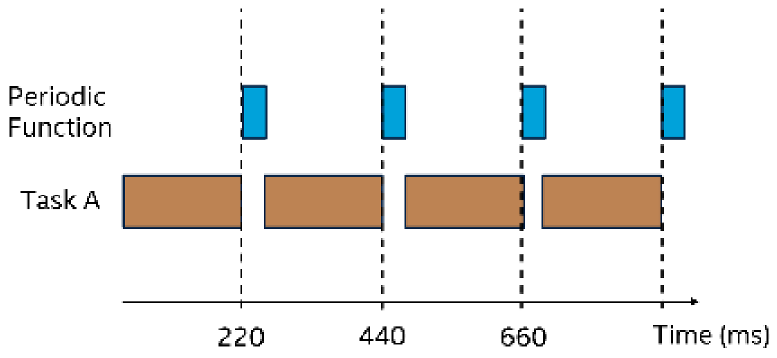
Performing periodic operations

- Approach 1: let the application measure time

We use an additional task with `vTaskDelay()` which allows us block the currently running task for a set amount of time. This solution is not efficient because a new task may require up to 1KB of additional memory.

- Approach 2:

We use an additional task A with `xTaskGetTickCount()` which returns the tick count since the scheduler was started. The tick count can be periodically compared with a given timestamp in task A.



- Approach 3:

Many microcontrollers (and microprocessors) include one or more hardware timers that 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 but the code will not be portable.

- Approach 4:

This solution uses FreeRTOS software timers which are built on top of the FreeRTOS tick timer and they are similar to hardware interrupts but operate at the task level. They enable to develop portable applications.

Software timer in FreeRTOS

FreeRTOS offers an API that makes managing these timers mush easier. When a timer is created, you assign a function, referred to as callback function, that is called whenever the timer expires. Timers are dependent on the tick timer, which defines their resolution, and they can be one-shot, if they execute the callback function once after the timer expires, or auto-reloaded, if they execute the callback function periodically every time the timer expires.