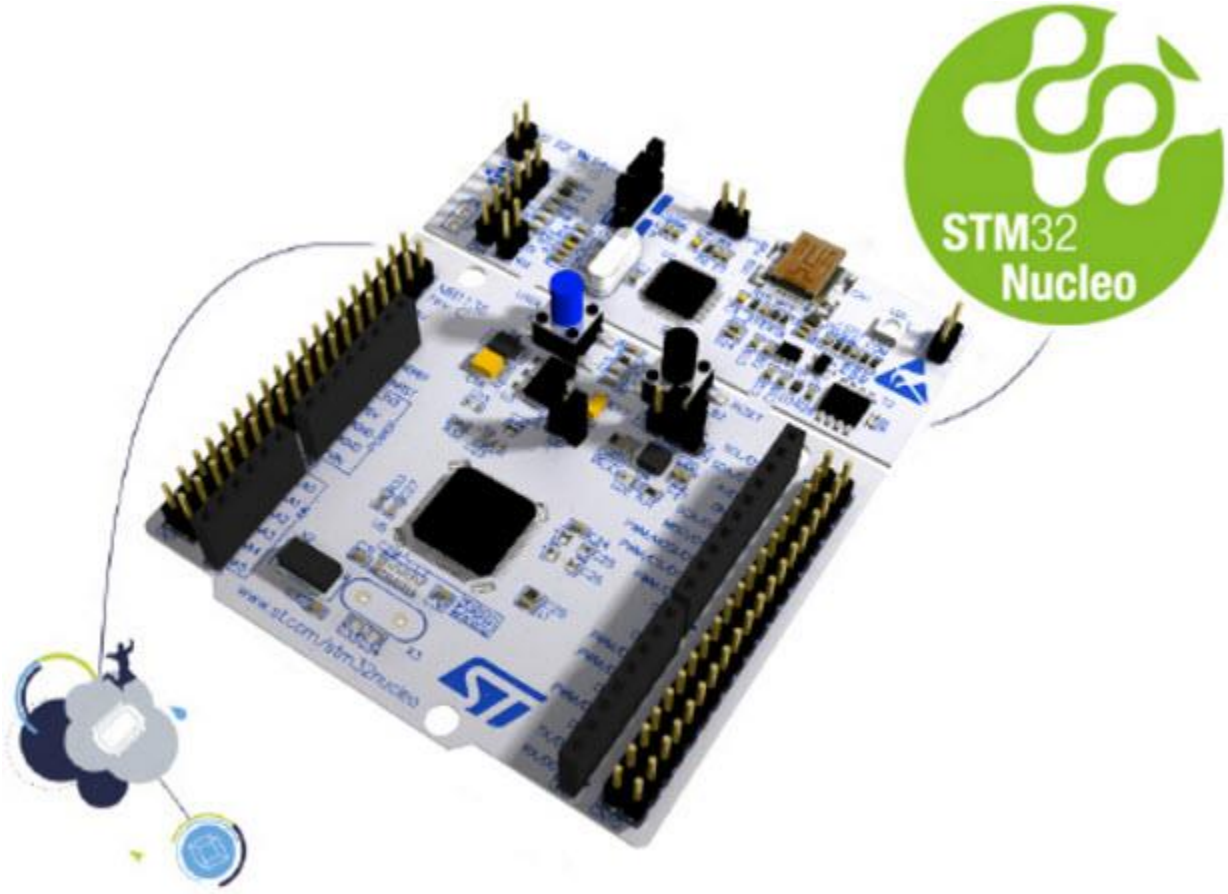
Technical report

Assignment 3:SysTick

By Veselin Atanasov



Teachers: Rene Bakx, Robert Verhijen

Date:02-10-2024

Contents

Introduction

This assignment delves deeper into the Nucleo F303RE STM32 board. It focuses on SysTick and interrupt events. The goal is to create a millis function which tracks how much time has passed since the system is up in milliseconds and also to create a blocking delay function.

Description

The first step, to making any application which requires accurate timing is to set the system clock to the desired settings. In the “Hello world!” example there is a function called SystemClock_Config(). This function sets all the required register settings for the system to run at the maximal possible clock rate which is 72Mhz. The next step is to calculate the reload value. The reload value is the variable which dictates after how many processor clocks, an interrupt event will occur. It is calculated with the following formula: (DesiredTickFrequency*ClockFrequency)-1. In this case, an interrupt has to occur each millisecond meaning that the formula with the actual values will look like this:(1ms*72Mhz)-1. The result of this calculation is 71 999 to achieve an interrupt each millisecond. This value needs to be set in the LOAD(RVR) register. This value serves as a counter. Each processor cycle one is subtracted from the original LOAD value which is at this point loaded in the VAL(CVR) register until 0 is reached, when 0 is reached, an interrupt event occurs. When the interrupt happens, VAL cycle retrieves its original LOAD value and the process starts again. Before assigning the reload value, it is important to first clear the CTRL(CSR) register to avoid unexpected behaviour. When that is done the LOAD can be set and the VAL value can be reset. The next step is to set the priority in the interrupt vector table in the NVIC register.

Now the CTRL(CSR) register should be configured. First an impulse source should be stated. The source could be external, but for this purpose the internal clock generator on the Cortex M4 chip is used. This is specified in the third bit of the CTRL register. The following step is to enable the SysTick interrupts by setting the second bit in the CTRL register. After this is done the clock counter should be enabled. The last and final step to configure the SysTick interrupts is to enable the clock itself. That setting is done in the first bit in the CTRL register.

After all the registers are configured, it is time to program the necessary functions and create the interrupt handler. To save the value of the millis, a global volatile uint32 variable is created. This value is volatile, because it will be changed by the interrupt handler and also usable for other purposes. The function of the handler is to just increment this variable, as the interrupt tick happens every millisecond because of the SysTick settings. The only purpose of the GetMillis() function, is to return the value of the volatile variable with its value at the moment the function is called. The last function that is required is the Delay() function. It takes as an argument the amount of time in milliseconds for the program to be blocked. This function also make use of the volatile variable to get the current time at the point where

the function is called. Afterwards a while loop is needed. The program will be stuck in this while loop until the user given argument is greater than the difference of the current value of the volatile variable and the value recorded when the function is called.

To check if the program works as expected, the internal led is set to blink at 1Hz. This is showcased by both functions, which are switched depending on the state of the internal button.

Conclusion

Timers are important for a lot of tasks. Time critical tasks like airplane black box systems, car engine ECUs and much more, or some less important as getting the air temperature reading every two minutes or so. To make reliable and fast time critical programs its of great importance to understand how timers work in depth.