Operating Systems for Embedded Systems

Student: Piergiovanni Ferrara - ID: 242549

4th December 2016

Target of Assignment #1
Development of an alarm system to monitor the voltage of a given analog input, and to inform the user using visual indications through an RGB led.

Implementation

Two ways were chosen based on different time management to reach the goal, however having a common part. Firstly, *main*() function is called in order to setup hardware, GPIO pins and other services.

The ADC module is initialized for the knowing of the voltage of an analog pin. A resolution of 16 bits is set. In both ways the module is configured for generating an interrupt when digital conversion is complete.

Subsequently, a starter task is also created. This particular task is able to create a semaphore to manage critical sections during ADC computation and to generate other tasks.

First way

In this way the starter task creates two tasks. After performing its job, it can be deleted becoming a dormant task and therefore it will not be no longer managed by Micrium OS.

The first task, *ADCTask*, is able to convert a value from the analog PTB2 pin into a digital one. Inside it, a function is called for starting AD conversion and enabling interrupt notification only setting two bits in two registers. When a digital value is ready, interrupt occurs and the handler *ADC_InterruptHandler* starts to work. Thanks to it, the new value is stored into a global variable. The operations of reading and storing are regulated by a semaphore that enables next conversion just after the new digital value is saved.

The other, *LEDTask*, controls RGB LED blinking according to the different digital values converted using a frequency rate of 10Hz or 20Hz, only in one case it will be fixed. For doing that, it is used an OS function called *OSTimeDlyHMSM*() that allows to delay for 50 or 25 milliseconds, depending of corresponding frequency wanted.

Second way

Another way consists in creating only one task by the starter task, which is $\underline{ADCTask}$ and the difference stands also on the time management.

In fact, in this case it is employed a timer module called PIT (*Periodic Interrupt Timer*) on channel 0, which is an array of timers that can be used to generate interrupts. To use that, you only need to define a macro called $PERIODIC\_TIMER$ that enables some code lines.

A function, $\underline{getFLED()}$, called inside an infinite loop by the starter task, reads a digital value just after the $\underline{ADC\_InterruptHandler}$ stored it inside the global variable on the previous step and by a semaphore.

After reading, $getFLED()$ sets the correct period into the module register and starts the timer. It is important to mark that Core clock is set by default to 120MHz while Peripherals clock to 60MHz, so the period is calculated through the equation shown below:

$$Period = Peripherals\_Clock \cdot \frac{Blink\_Rate}{2}$$

When the counter starting from the set period, reaches a zero value, an interrupt is raised and handled by the function that toggles RGB LED according to its blink rate, thanks to the prior settings. Thus, you don't need another task unlike the first way (LEDTask).