Tutorial 3: Zephyr RTOS Introduction

Bernardo Falé[1][93331] and Diogo Maduro[2][80233]

University of Aveiro, Aveiro, Portugal

1 Introduction

In this work we learn how to implement a real-time application using Zephyr RTOS within a micro-controller environment. This application is composed of three tasks and follows the typical structure of embedded software. All the available work and code is available in the team's repository at https://github.com/bernardofalle/SOTR, so the sustained development of the work can be verified.

2 Analysis

2.1 Sensor Task

The Sensor task is responsible for taking the values of the samples with a period defined previously in the *SAMP_PERIOD_MS()* macro, and we can define this job as the "leader" of the three defined tasks; This means that this task also works as a state machine, that checks if a test flag is true or not; if the flag is true we are in the test state, that allows the LED toggling until the defined timer has expired.

The configuration of the ADC allows to store the samples in the shared memory. We also define options to read read extra samples, in this case *N_SAMPLES-1* . This task then calls *k_sem_give()* , which gives a "flag" to initiate a sporadic task described in the next topic.

2.2 Filter Task

After this task leaves the suspended state and executes it computes an average value from the ten samples that were read in the sensor task. This reduces the noise from the retrieved ADC readings assuming that the values represent a distance sensor; Similar to the previous task, this job calls *k_sem_give()* , which in return sends a signal to the output task, and flags its execution.

2.3 Output Task

Once the output semaphore is taken, this task check what's the stored distance and makes a decision based on its value. It displays 4 different configurations of LED combinations.

2.4 ISR

To implement the test button, it was chosen to configure an interrupt that should stop the normal flow of the program, that who'd turn on the test flag. We made this decision based on the fact that the the interrupt takes an infinitely small time to compute.

2.5 Time Diagram

After running the program for a while we chose to operate with the reference values displayed in the table below, obtained through some implementation in our program, and to add some context to the diagram. Note that T_i references period for the sensor periodic task and minimum inter-arrival time for the remaining sporadic tasks.

τ_i	C_i(ms)	T_i(ms)
Sensor	167	500
Filter	20	515
Output	5	600

With this information we designed our time diagram and shall present it below in figure 1; This should describe the execution of the program in its normal behaviour.

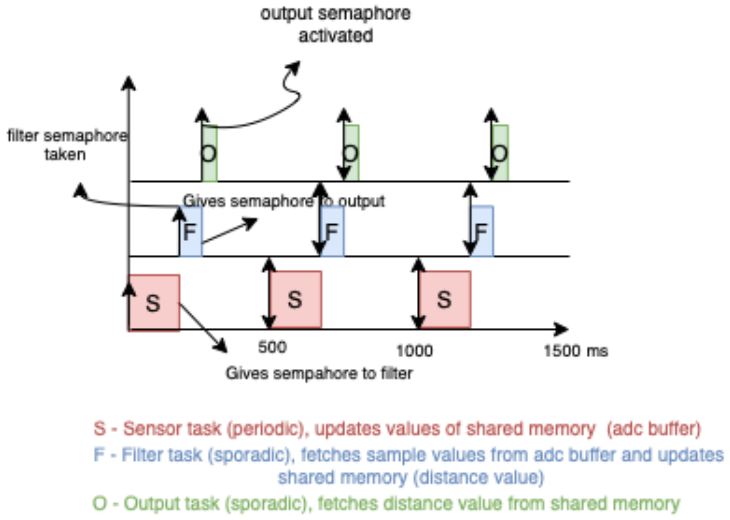

Fig. 1: Multi-task application time diagram