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`.

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. The configuration of the ADC allows to store these values in the shared memory. Currently we store 11 elements (Number of samples + 1) because the first reading is always out of range (probably a hardware bug). 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 be required to blink the 4 available LEDS for 5 seconds. However, this is a bit buggy and the button can bounce when pressed. Reset the board if necessary.

2.5 Time Diagram of

After running the program for a while we chose to operate with the reference values displayed in the table below. Note that T_i references period/minimum inter-arrival time.

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

With this information we shall present figure 1 as our time diagram that should describe the execution of the program in its normal behaviour.

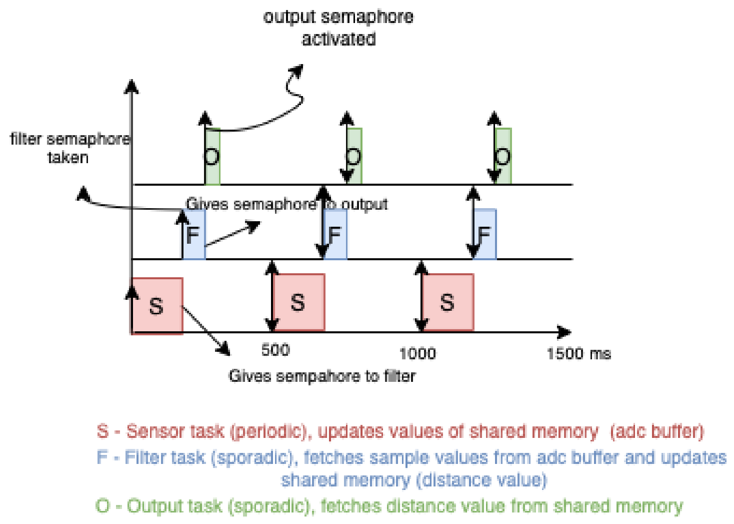
output semaphore activated
filter semaphore taken
Gives semaphore to output
O
O
O
F
F
F
S
S
S
500
1000
1500 ms
Gives sempahore to filter
S - Sensor task (periodic), updates values of shared memory (adc buffer)
F - Filter task (sporadic), fetches sample values from adc buffer and updates shared memory (distance value)
O - Output task (sporadic), fetches distance value from shared memory

Fig. 1: Multi-task application time diagram