

Lab 8: Introduction to Real Time Operating Systems

1 Overview

The purpose of this exercise is to understand the basics of a real time operating system (RTOS). You will be using Freescale's OSEKturbo RTOS and will be able to see how Tasks, Alarms, and Semaphores, defined as Resources by Freescale, interact with each other and prevent issues such as dead-lock on the microprocessor.

NOTE: This part of the lab in its current condition will not require any programming by the student. This is merely a tool to reinforce concepts discussed in lecture.

2 OSEK Tasks

As stated in the OSEK User's Manual:

"Complex control software can be conveniently subdivided into parts executed according to their real-time requirements. These framework for execution of functions. The Operating System provides parallel and asynchronous task execution organization by the scheduler.

The Basic Tasks release the processor only if:

- they are being terminated
- the OSEK OS is execution higher-priority tasks
- interrupt occurred"

2.1 Task Priority

For the OSEKturbo RTOS the Task Priorities determine whether a task can preempt another Task that is already in operation. Priorities are defined by the user with the highest numeric value giving the highest priority. Therefore a task with a priority of 5 would preempt a task of 4 while a task of 3 could not preempt a task with priority 4.

2.2 Sample code for Task Initialization:

```
TASK(Task Priority)
{
  Task Code Goes here much like a function
}
```

Please refer to User's Manual for more detail.

3 OSEK Alarms

As stated in the OSEK User's Manual:

"The alarm management is built on top of the counter management. The alarm management allows the user to link task activation or even setting or a call to callback function to a certain counter value. These alarms can be defined as either single (one-shoot) or cyclic alarms. Examples of possible alarm usage are:

- 'Activate a certain task, after the counter has been advanced XX times (Where XX is a number the user picks).'
- 'Set a certain even, after teh counter has reach a value of XX.'

An example of an initialization of an Alarm:

```
SetRelAlarm(&OsAlarms[0], 3, 3);
```

The first input defines which task will be called when the alarm goes off. The second value initializes the alarm, and the third value is what the alarm is reset to after it has gone off. *Please refer to User's Manual for more detail.*

4 OSEK Resources

As stated in the OSEK User's Manual:

"The resource management is used to coordinate concurrent access of several tasks or, and ISR's to shared resources, e.g. management entities (scheduler), program sequences (critical section), memeory or hardware areas.

The resource management ensures that:

- Two modules (tasks or ISRs) cannot occupy the same resource at the same time.
- Priority inversion cannot arise while resources are used.
- Deadlocks do not occur due to the use of these resources.
- Access to resources never results in waiting state.

The functionality of the resource management is required only in the following cases:

- Full-preemptable tasks.
- Non-preemptable tasks, if the user intends to have the application code executed under other scheduling policies too.
- Resource sharing between tasks and, or ISRs

Resources cannot be occupied by more than one task or ISR at a time. The resource which is now occupied by a task or ISR must be released before another task or ISR can get it. The OSEK operating system ensures that tasks are only switched from ready to running state if all the resources which might be occupied by that task during its execution have been release."

Example code to Get and Release a resource:

```
GetResource(Resource Name);
```

Your Code

```
ReleaseResource(Resource Name);
```

Please refer to User's Manual for more detail.

5 Lab Demo

Given the above reference you will be able to see how these different features of the OS are integrated to provide dead-lock protection as well as significant robustness on task operation. Unzip the required OSEK files and go to `OSEK_RESOURCE/Test_1_rappid_rtw/lab8` In this directory there should be a `Test_1.elf` and a C-file `rappid_osek_main.c`. Open `rappid_osek_main.c` and familiarize yourself with the code inside both the BaseRate Task and the SubRate Task. A list of the functionality of each LED can be seen below:

LED Functionality:

[28]:Oscillates at the end of the baserate inside of RateTransition Resource

[29]:Oscillates at the beginning of the baserate outside of RateTransition Resource

[30]:Oscillates inside a while loop triggered by dip switch [125]=1 outside of RateTransition Resource in baserate task

[35]:Turns on at the Beginning of SubRate task, turns off at the end of SubRate task

[36]:Turns on at the Beginning of BaseRate task, turns off at the end of BaseRate task

[40]:Oscillates inside a while loop triggered by dip switch [124]=1 outside of RateTransition Resource in SubRate task

[41]:Oscillates inside a while loop triggered by dip switch [123]=1 inside RateTransition Resource inside SubRate task

[42]:Oscillates inside RateTransition resource inside SubRate task

[43]:Is turned off by dipswitch [126]=1 in BaseRate task and is turned on by SubRate task

NOTE: You can go to `OSEK_RESOURCE/Test_1_rappid_rtw/osek` and open `cfg.oil` to see the priority of the Base Task and the SubTask as well as the Task declarations, Alarm initializations and other set-up features.

You should notice that there are both LED's as well as Dipswitches being used. Their functionality will be explained once the code is running. Also take note of where the `GetResource` and `ReleaseResource` function calls are.

Once you have familiarized yourself with the code go back and make sure `Test_1.elf` exists. If it does not just double click on `Test_1.bat` and the .elf should show up, otherwise consult with your GSI. Load the .elf file using the P&E debugger. Go ahead and run the code. You will notice six LED's flashing at different rates. There are three other LED's that will be used when triggered by the correct dip switches for a total of nine LED's. While you can spend a significant amount of time digging through all the different combinations of Dipswitches and LEDs, There are two combinations of particular interest.

5.1 Dip Switch[123]

Flip the Dip Switch up. You will notice that LED [41] is now the only LED blinking. Consider LED[41]'s location in code where it is inside the `GetResource` and `ReleaseResource` inside a while loop and it is obvious that no other task can preempt it since they are both link by the RateTransition Resource. Consider the advantages this provides for data protection. Once you have satisfied your interest flip the Dip Switch down to zero and continue.

5.2 Dip Switch[124]

Flip the Dip Switch up. You will notice that LED [40] begins flashing at an intermittent rate. While this LED is inside a while loop it is *not* inside a resource scope. Thus the other task, which has higher priority, can in fact preempt the SubRate task which is why you see LED[28] and LED[29] continuing to flash.