MIC5-LAB-V – RTOS

The MIC5 workshops are part of the microcontroller courses MIC1 – MIC6. The main subject of MIC5 is Real Time Operating Systems (RTOS). The workshops have a hands-on approach, meaning you will gain both knowledge and skills related to using an RTOS on a microcontroller.

There are six workshops. Each workshops starts with a presentation that introduces that week's topic. The remaining time in a workshop will be used for working on the lab exercises. Unfinished exercises must be finished at home.

1. FreeRTOS

There are numerous RTOS solutions available that can be used to add operating system features to embedded applications, such as µCOS, Keil RTX, Zephyr, and embOS. For these workshops FreeRTOS was selected for several reasons:

1. FreeRTOS is platform independent and there are ports available for many different microcontrollers, such as ARM Cortex-M and Atmel AVR.
2. FreeRTOS is well documented and maintained by professional developers.
3. FreeRTOS supports kernel aware debugging, which allows to see detailed information of kernel state.
4. FreeRTOS has been successfully deployed in projects at HAN University of Applied Sciences, such as HANcoder.
5. FreeRTOS is open source and can be used in commercial applications.

2. Teaching material

This chapter describes the teaching material used for these workshops.

2.1 Books

The course is largely based on the information from the following free ebook:

Barry, R. (2016). Mastering the freertos real time kernel. Pre-release 161204 Edition. *Real Time Engineers Ltd*.

This book, amongst other material, such as the FreeRTOS reference manual, are available as a download from the FreeRTOS website:

https://www.freertos.org/Documentation/RTOS_book.html

2.2 Hardware

FreeRTOS runs on many different microcontrollers. These workshops reuse the FRDM-KL25Z development board, depicted in the adjacent image, that was used in previous MIC workshops.

2.3 Software

In the previous MIC workshops, the FRDM-KL25Z development board was programmed in the Keil MDK-ARM integrated development environment (IDE). For these workshops a different IDE has been selected:

MCUXpresso IDE

The most important reason for using a different IDE this time, is that MCUXpresso IDE supports FreeRTOS kernel aware debugging. Other reasons are that it will allow you to get familiar with another IDE, while using the same microcontroller development board and that MCUXpresso IDE doesn't have a 32kB linker restriction.

MCUXpresso IDE has the option to add FreeRTOS to your project by selecting this as a configuration option when creating a new project. We will, however, not use this option, because one of the learning goals of these workshops is the creation and configuration of an RTOS project in any IDE. This means you must understand the files that are part of the FreeRTOS distribution and how to configure them yourself.

3. Topics

The following list summarizes the topics that will be discussed:

1. Preface
   Understanding the FreeRTOS distribution
   Task management
2. Creating a FreeRTOS project
   Heap memory management
3. Queue management
   Software timer management
4. Interrupt Management
5. Resource Management
6. Event Groups (TODO: optioneel? Eventueel laten vervallen als voorgaande niet binnen de tijd af te ronden is.)
   Task Notifications (TODO: optioneel? Eventueel laten vervallen als voorgaande niet binnen de tijd af te ronden is.))

   A detailed API description is provided in each PowerPoint presentation, however, these are available for reference and will be skipped during the presentations of the PowerPoint in class.

4. Examination

4.1 Learning goals

The learning goals are examined by means of an assignment. FYI, the learning goals are as follows:

The student

1. *knows different scheduling techniques and can name the advantages and disadvantages.*
2. *knows different methods for inter-task communication and can name the advantages and disadvantages.*
3. *can integrate an existing RTOS into a microcontroller development environment.*
4. *can optimally configure a RTOS, e.g. memory usage, for a given functional description.*
5. *can use a RTOS to run given tasks.*
6. *can combine a RTOS with interrupt handling.*
7. *can use the debugger to detect functional errors related to the RTOS.*

Basically, it is your assignment to use an RTOS, preferably FreeRTOS, on an ARM microcontroller. The RTOS must have been added and configured manually for a project. In other words, without using a tool (i.e. MCUXpresso, STM32CubeMX, etc.) that does the adding and configuration more or less automatically. If you are using an RTOS in the semester 6 project, it is allowed to hand in the relevant work from that project for MIC5 as well. Otherwise, a (mini) project must be handed in. Either way, at least the following properties must be part of the project in order to fulfil all assessment criteria:

- A description of scheduling techniques and why a scheduling technique was chosen for the project. The description is provided in a readme file together with project related source code files.
- The same readme contains a description of methods for inter-task communication and which methods were used in the project and why.
- Source code for an ARM microcontroller of your own choice and an IDE of your own choice. The project includes all source files, so also the RTOS source files.
- Documented RTOS configuration files (i.e. FreeRTOSConfig.h), by adding or updating comment relevant to hardware and software used in your project (i.e. related to memory usage).
- A high level design showing the system tasks, interrupt handlers and inter-task communication. The system uses at least five different tasks and two different interrupt handlers.
- A document describing at least three test scenarios. Each test scenario describes the expected behaviour by means of a timing diagram and the actual behaviour by means of RTOS aware debugger information (i.e. a visualisation of the tasks and their states at specific timestamps in the timing diagram).

4.2 Assessment model

To assess the learning goals, the examiner uses the following assessment model. To score at a level *"Good"*, the underlying level *"Sufficient"* must also be met.

1. Description of scheduling techniques			
Bad/Missing	0		There is no description.
Insufficient	1		The description mentions what scheduling technique is chosen.
Sufficient	2		The description mentions why the selected scheduling technique is chosen.
Good	3		The description mentions why other scheduling techniques are not chosen and why.
Feedback			
2. Description of inter-task communication			
Bad/Missing	0		There is no description.
Insufficient	1		The description mentions what inter-task communication methods are chosen.
Sufficient	2		The description mentions why the selected inter-task communication methods are chosen.
Good	3		The description mentions why other inter-task communication methods are not chosen.
Feedback			
3. Source code			
Bad/Missing	0		There is no source code or it is incomplete.
Insufficient	1		Source code does not build without errors.

Sufficient	2		Source code is complete and the project builds and runs successfully.
Good	3		The source code is well organized (i.e. using folders for distinct project parts) and doesn't contain unnecessary files, such as build output files.
Feedback			
4. Documented RTOS configuration files			
Bad/Missing	0		There is no documentation or the file is missing.
Insufficient	1		Standard documentation is still there, but not updated with project specific configuration descriptions.
Sufficient	2		The standard documentation is enhanced with project specific configuration description, so it is clear what was changed with respect to the standard configuration.
Good	3		The descriptions (of the most important configurations) make clear why they have been configured in that way.
Feedback			
5. A high level design			
Bad/Missing	0		The design is missing.
Insufficient	1		The design is incomplete, does not match the implemented source code or is too simple (contains less than five tasks and less than two ISRs).
Sufficient	2		The design matches the implemented code, but contains minor differences (such as different task names, or different inter-task communication methods).
Good	3		The design matches the implemented code exactly and the design makes sense for the problem at hand.
Feedback			
6. Test scenarios			
Bad/Missing	0		Test scenarios are missing.
Insufficient	1		The expected behaviour by means of a timing diagram is missing or the actual behaviour by means of an RTOS aware debugger is missing.
Sufficient	2		The expected behaviour by means of a timing diagram and the actual behaviour by means of an RTOS aware debugger are described. The relation between expected behaviour and actual behaviour is described or visualized clearly.
Good	3		At least one test scenario describes a comprehensive situation (of at least 10 task switches and the use of inter-task communication) and demonstrates a mismatch between expected and actual behaviour. The right conclusions are drawn, for example the tasks that were involved in the error and how this might be or has been solved.
Feedback			

Based on the completed assessment form, the examiner holistically determines the final mark on a scale of 1 .. 10. In doing so, the following calculation serves as the basis for the final mark:

$$grade = \frac{\sum(score\ per\ assessment\ criterium)}{3 \times number\ of\ assessment\ criteria} \times 10$$