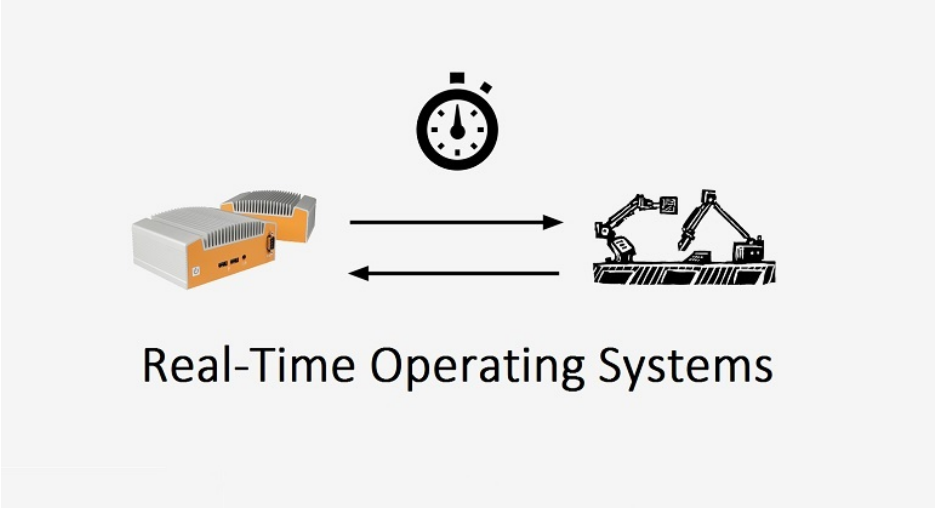

Real-time Operating Systems

Tutorial: FreeRTOS introduction
2021/2022

Paulo Pedreiras
DETI/UA/IT
pbrp@ua.pt

November 12, 2021

1 Objectives

Become familiar with a real-time operative system (RTOS) for microcontrollers and resource-constrained microprocessors.

- Study the FreeRTOS API, the tasks' structure, the initialization an termination functions and some of the IPC mechanisms.
- Develop simple applications using the FreeRTOS real-time kernel.

2 Procedure

2.1 Installation of toolchain and FreeRTOS

- Setup and install the PIC32 compiler toolchain: MPLABX IDE and XC32
- Download and unpack the FreeRTOS source code, available at: https://www.freertos.org/
  - To simplify the installation it is advisable to download the package with examples (e.g. "FreeRTOS 202107.00") instead of the LTS package.
  - Supplied demo code was adapted and tested with FreeRTOS V202107 and XC32 2.50. It is advisable to use these versions.
- Download the demo project available on the course website ("FreeRTOS Demo Code").
- Unpack and copy the supplied demo project to the FreeRTOS demo folder:
  "FREERTOS_FOLDER/FreeRTOSv202xxx.yy/FreeRTOS/Demo/"
  where xxxx.yy match the FreeRTOS downloaded version.

2.2 Code analysis

- Open the "ChipKitLedBlink" MPLABX Project.
- Browse the source code. Pay special attention to the system calls used to define, create and activate the tasks. Study in detail the structure of the tasks.

- Compile and execute the supplied code. Observe the application behavior and explain it. Detailed instructions about the process of compiling and executing applications on the Digilent chipKIT Max32 board are supplied on the course site.
- Repeat the above procedure for the two other supplied demo applications.

2.3 Using FreeRTOS

- [**A1**] Open the "ChipKitLedBlink" MPLABX Project. Invert the tasks' priorities. Observe the activation pattern of the LED. Explain why its behavior changes.
- [**A2**] There are different API calls for defining the task's periods. Find one that is more accurate than the one used in the example provided and modify the code to use it.
- [**A3**] Develop a data acquisition and processing application structured in three tasks: data acquisition (**Acq**), data processing (**Proc**) and data output (**Out**).
  - Task **Acq** is executed every $100\,ms$ and should acquire a sample of ADC Channel 0. You shall use a potentiometer ($0\,V\ to\ 3.3\,V$) that simulates a temperature sensor. Assume a linear sensor in the range ($0\,^oC = 0\,V ... 100\,^oC = 3.3\,V$).
  - Task **Proc** takes as input the output of task **Acq** and outputs the average of the last five samples received (integer value, sliding window).
  - Task **Out** takes as input the output of task **Proc**, converts the numeric result to a string and prints it.
  - The Inter-Process Communication (IPC) should be carried out via shared memory. Investigate several options to synchronize the tasks and choose one for implementation, providing a proper justification.
- [**A4**] Repeat Assignment [**A3**], but considering now that are employed Queues for IPC.

3 Deliverables

The following elements should be uploaded for evaluation:

- Two separated compressed files with the project code, corresponding to Assignments [**A3**] and [**A4**].
  Projects should be standalone and referenced (i.e. level 1 child) to the "Demo" folder. A user should be able to compile the projects just by decompressing the uploaded files to the Demo folder of a clean FreeRTOS installation. Failing to comply with this requirement implies a 0 (zero) grade.
- A **one page** report, in "pdf" format, with your analysis of:
  - The call used for Assignment [**A2**] and a short justification why it performs better than the original one.
  - A brief description of the IPC mechanism considered in Assignment [**A3**] and a justification for the choice made with respect to the other possibilities.