Chapter 14: bugs and clarifications

1/29/2022

The issues described here are minor. The more interesting ones are bold and underlined, e.g., **Clarification**.

Chapter 14: Choosing an RTOS API (pg 363)

- **Typo** (pg 370)
  - `osKernelGetSysTimerCount` is stated twice:

The kernel interfaces are similar, although some timer implementations that STM has provided aren't all that intuitive, specifically `osKernelGetSysTimerCount` and `osKernelGetSysTimerCount`.

- **Clarification**, threads vs tasks (pg 375)
  - It appears that CMSIS-RTOS's *threads* are the same as FreeRTOS's *tasks*.
  - What threads and tasks are varies among operating systems.
  - From *Mastering the FreeRTOS Real Time Kernel*,

In FreeRTOS, each thread of execution is called a 'task'. There is no consensus on terminology within the embedded community, but I prefer 'task' to 'thread,' as thread can have a more specific meaning in some fields of application.

  - Linux threads are different than CMSIS-RTOS's threads:
    - https://www.thegeekstuff.com/2013/11/linux-process-and-threads/
  - In the IBM mainframe's OS, its tasks are different than FreeRTOS's *tasks*
    - https://www.ibm.com/docs/en/zos/2.2.0?topic=subtask-attach-attachx-description

- **Typo** (pg 379)
  - The step "*5. Start the scheduler:*" should be after Step 10. Steps 2-4 are concerned with setting-up the GreenTask, and Steps 6-10 are concerned with setting-up the RedTask.
  - Also, the paragraph shown below is after Step 5. It's not part of Step 5, but the indentation incorrectly implies it is part of Step 5:

"*main_taskCreation_CMSIS_RTOSV2.c also contains an example of starting a task with statically allocated memory...*".

  - That paragraph is concerned with Step 6. The same applies to the two paragraphs following that paragraph. It seems those three paragraphs should be aligned on the left margin to indicate they are not part of Step 5, or they should be made Step 6.

- **Clarification**, CMSIS-RTOS and the underlying RTOS (pg 379)
  - **Problem:**
    - The following two paragraphs are under Step 5 (though, they should not be, as just described). The paragraphs describe how the TCB is configured, however, some rewording and additional info would help clarify.

`main_taskCreation_CMSIS_RTOSV2.c` also contains an example of starting a task with statically allocated memory for the task control block and task stack.

Static allocation requires computing the sizes of RTOS control blocks (such as `StaticTask_t`) that are specific to the underlying RTOS. To reduce the coupling of code to the underlying RTOS, an additional header file should be used to encapsulate all RTOS-specific sizes. In this example, this file is named `RTOS_Dependencies.h`.

  - **Attempted clarification**:

RedTask is configured next. It shows how a task can use statically-allocated memory for the task control block (TCB) and the task stack. It also shows an example of a dependency between CMSIS-RTOS and the underlying RTOS, and how to manage such dependencies.

For the TCB, the underlying RTOS's TCB will be stored in statically-allocated memory. So, the static allocation must specify the size of the underlying RTOS's TCB. In `main_taskCreation_CMSIS_RTOSV2.c`, the declaration for the TCB storage is:

```
uint8_t RedTask_TCB[TCB_SIZE];
```

(In the code, that declaration's comments are incorrect, as described below.)

The storage size is specified by the macro `TCB_SIZE`. By using a macro, the program does not have a direct dependency on FreeRTOS.

The definition of `TCB_SIZE` is shown below. `StaticTask_t` is the FreeRTOS TCB, and it's defined in