Run time statistics, stack usage and USB

Study the chapters about generating run time statistics and stack usage monitoring in the FreeRTOS tutorial. Study *MCUXpresso IDE FreeRTOS Debug Guide* in the IDE installation directory.

LPC1500 processor family has variants with variable amount of on chip RAM. The RAM is divided into banks that can enabled/disabled to save power if some of the regions are not used for example in sleep mode. Probably for this reason the default MCU settings divide RAM into separate sections. The default linker script uses the first RAM region on the list for stack and heap and leaves allocating the rest of the memory for the user. Placing variables/object in other areas requires use of special compiler specific directives.

C++ uses a lot of heap and our default configuration uses standard heap for FreeRTOS objects. Depending on task stack sizes the default 16 kB heap may be exhausted after starting just a few tasks. A typical indication of running out of heap space is getting a hard fault when the scheduler is started. However it is possible to edit the memory map in a way that allows linker to use more RAM. You have learned how join RAM regions to get 32 kB of RAM for you application. The top 4 kB (the third RAM region) is reserved ROM USB stack and can be used only if USB is not going to be used.

In the following exercises you will learn how to use FreeRTOS kernel aware debugging of MCUXpresso IDE and how to use the built in USB stack in your projects.

This set of exercises requires two micro-USB cables and that you make a small report of your findings.

Exercise 1 – Run time statistics

Import usb_cdc project from the course workspace into a workspace that contains chip, board, and freertos libraries. Check that the project compiles without errors.

Enable collection of run time statistics in FreeRTOSConfig.h.

Debug usb_cdc project and test that you can send and receive data. Red led toggles every time when your board sends data to USB and green when data is received from USB.

Suspend the project and take a screenshot of the runtime statistics. Add screen shot to your report and report CPU usage per task.

Edit user_vcom.h: comment out line 12 (#define POLLING_CDC). Clean and build your project.

Debug your project. Test that input and output work the same way as before. Suspend the project and take a screenshot of the runtime statistics. Add screen shot to your report and report CPU usage per task. You should see different CPU usage. Explain why the values are change with a minor change in code.

Exercise 2 – Stack space monitoring and SCTimer

Report stack usage of each task in the project.

Change the task that sends the increasing counter values so that it always sends a fixed string and does not call sprint at all. Minimize the stack size of each task including idle task (edit task create calls) so that each task uses only minimum amount of stack that is needed to run the task.

Report stack usage of each task and the values used for stack sizes.

Study chapters 1 and 6 of SCTimer PWM cookbook and FreeRTOS Run Time Statistics chapter in the FreeRTOS tutorial.

The example code uses small SCTimer in unified mode (= single 32 bit counter) to replicate the functionality from the FreeRTOS tutorial example. Find the SCTimer tick rate and estimate how long it takes before an unsigned 32 bit counter used for statictics collection overflows. Report both tick rate and the estimated overflow time.

Exercise 3- The empire strikes back

Implement Lab2-Ex3 (the oracle) using USB. Make the oracle say: “I find your lack of faith disturbing” instead of “Hmmm…” when oracle is thinking.