

# 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. The things to put in the report are marked with red in the exercises.**

**The second USB will show up as another COM-port. All the input and output on the second USB can viewed using MCUXpresso serial console or PuTTY.**

**Important!**

**Keep the second USB cable disconnected when you start debugging. When the code hits break point at the beginning of main() then connect the second USB cable.**

## 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. Use the variant that uses SCTimer for statistics collection

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 changed with a minor change in code. Which option is better?

Then test both versions using Cortex-M3 cycle counter as the statistics collection timer. Report results.

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

The example in FreeRTOSConfig.h shows two ways of measuring the runtime statistics: processor core cycle counter and 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 statistics collection overflows.

Report SCT tick rate and the estimated overflow time.

Study Cortex-M3 cycle counter.

Report the tick rate of cycle counter and the estimated overflow time.

## Exercise 2 – Stack space monitoring and heap monitoring

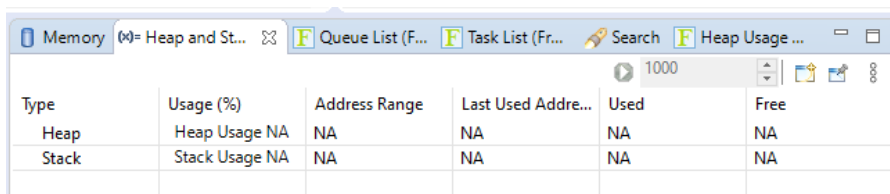
Report stack usage of each task in the project (See FreeRTOS task list).

Change the task that sends the increasing counter values (send\_task) so that it always sends a fixed string and does not call `sprintf()` at all. Minimize the stack size of each task including idle task so that each task uses only minimum amount of stack that is needed to run the task. You need to edit task create calls and to study FreeRTOS documentation to learn how to change idle task stack size.

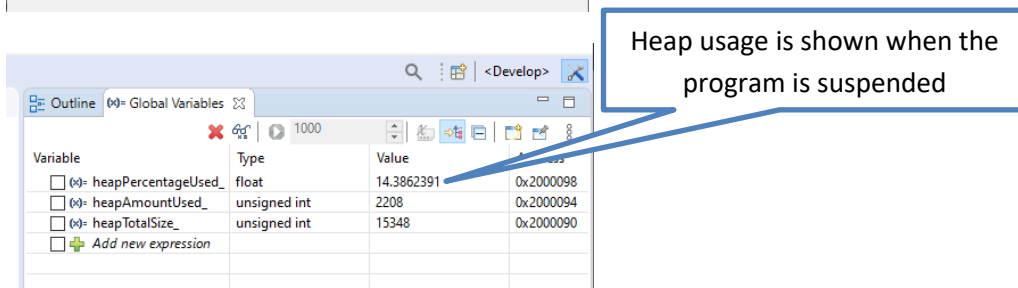
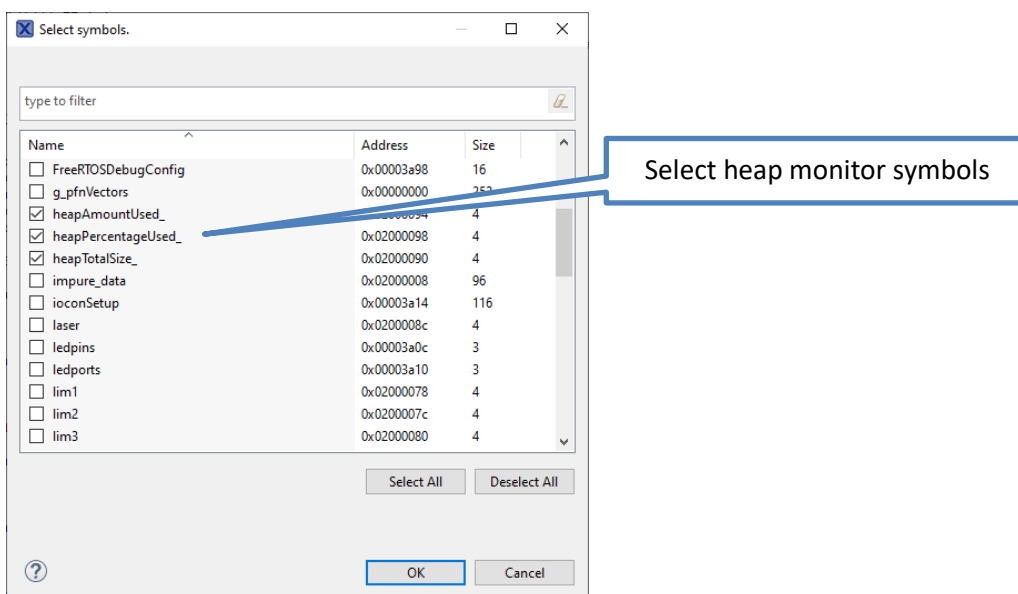
Report stack usage of each task and the values used for stack sizes after you have reduced stack sizes to minimum.

Add files from heap\_lock.zip to your project and call `heap_monitor_setup()` after hardware init. In addition to heap locking there is heap usage monitoring that can be used to monitor C/C++ heap usage when memory model **heap3** is used (FreeRTOS uses `malloc` for OS objects).

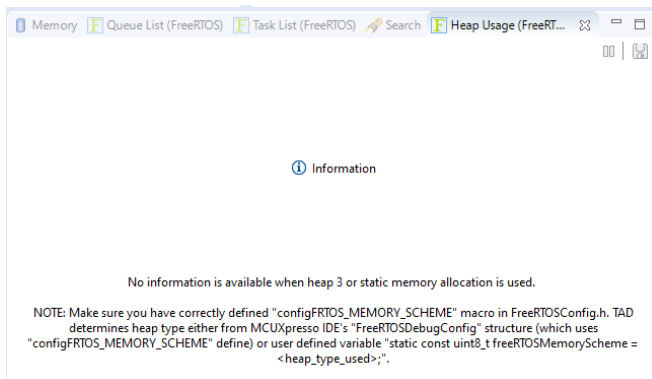
MCUXpresso heap monitoring is disabled when FreeRTOS is used (thank you NXP!). You can view heap usage by inspecting global variables.



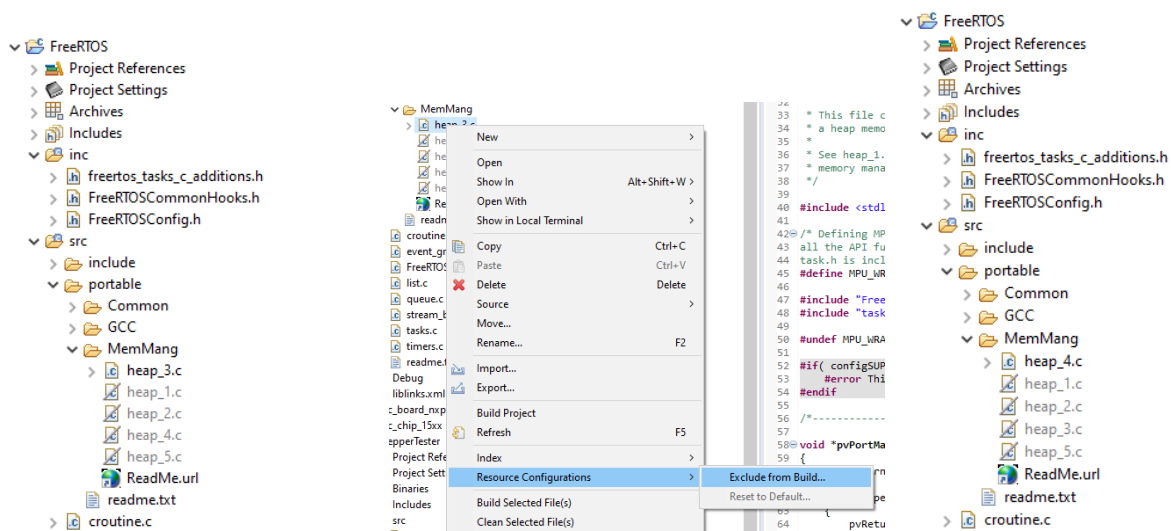
Type	Usage (%)	Address Range	Last Used Address	Used	Free
Heap	Heap Usage NA	NA	NA	NA	NA
Stack	Stack Usage NA	NA	NA	NA	NA



## Heap4 memory model



First, we need to switch memory model. Exclude heap\_3.c from build and enable heap\_4.c



```
#define configFRTOS_MEMORY_SCHEME 4
#define configUSE_PREEMPTION 1
#define configUSE_IDLE_HOOK 0
#define configMAX_PRIORITIES ( 8 )
#define configUSE_TICK_HOOK 0
#define configCPU_CLOCK_HZ ( (uint32_t) SystemCoreClock )
#define configTICK_RATE_HZ ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE ( ( unsigned short ) 100 )
#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 6 * 1024 ) )
```

Add this line to specify memory model. If not present the memory model defaults to 3. Task aware debugger uses this information to determine if heap usage can be displayed.

Specify heap size. This will create a special heap dedicated to FreeRTOS. C++ will keep on using the standard heap.

Type	Usage (%)	Used	Free	Address Range
Heap #4	3.07% Used	1,98 kB / 6 kB	66,93% (4,02 kB)	0x020001dc - 0x02...
#	Details	Block Start	Block End	Size
1	Allocated	0x020001dc	0x020001e7	0xc (12 B)
2	stepper (Task Stack)	0x020001e8	0x0200068f	0x4a8 (1,16 kB)
3	Allocated	0x02000690	0x0200069f	0x10 (16 B)
4	stepper (Task TCB)	0x020006a0	0x0200075b	0xbc (188 B)
5	Allocated	0x0200075c	0x02000767	0xc (12 B)
6	IDLE (Task Stack)	0x02000768	0x020008ef	0x188 (392 B)
7	Allocated	0x020008f0	0x020008ff	0x10 (16 B)
8	IDLE (Task TCB)	0x02000900	0x020009bb	0xbc (188 B)
9	Allocated	0x020009bc	0x020009bf	0x4 (4 B)
10	Free	0x020009c0	0x020019cf	0x1010 (4,02 kB)

When dedicated heap is used TAD is able to display detailed FreeRTOS heap usage.

Measure how much heap FreeRTOS uses and how much standard heap is used. FreeRTOS dedicated heap will be allocated at compile time so it will not be part of standard heap statistics.

### Exercise 3- The empire strikes back

When creating a new task, it is usually easiest to start with a large stack for the task and then check the runtime statistics to see the real stack size needed.

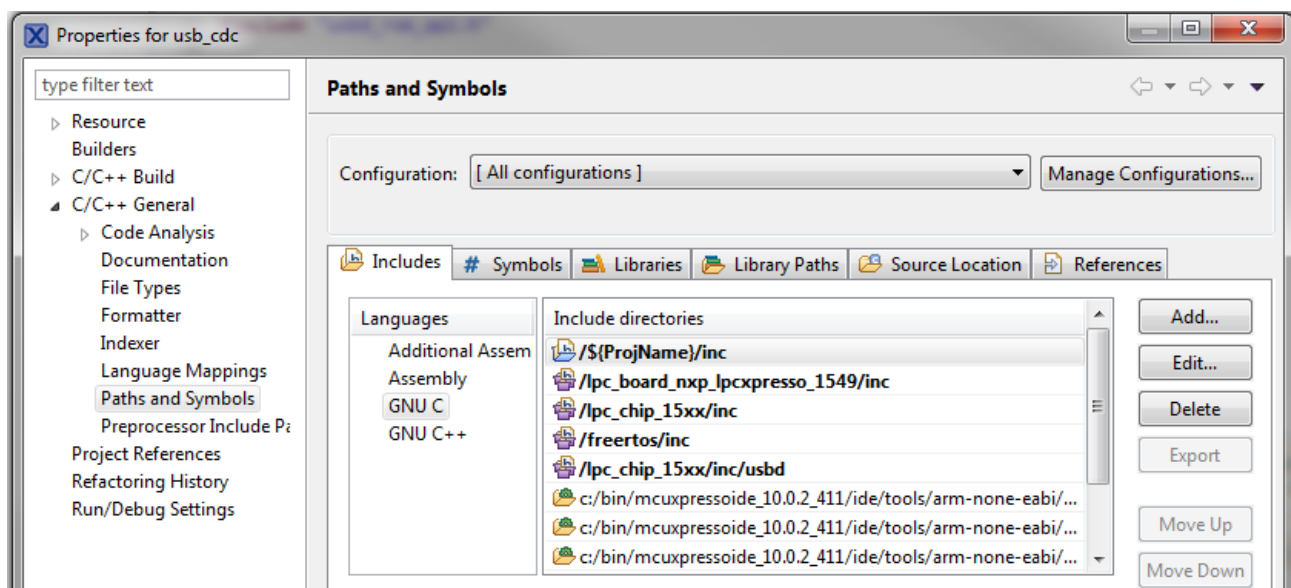
Implement Lab2-Ex3 (the oracle) using USB. Make the oracle say: “I find your lack of faith disturbing” instead of “Hmmm...” when oracle is thinking.

Note that when you read data from the USB port you need to use a buffer of at least RCV\_BUFSIZE bytes. Using a smaller buffer may cause received data to be discarded. See user\_vcom.h for more details.

Add a task that reads commands from debug serial port. When a command “stat” is received the debug task prints run time statistics to the debug serial port. See: <https://freertos.org/rto-run-time-stats.html> for information about seeing runtime statistics without debug probe. When command “list” is entered the task prints task list. See: <https://freertos.org/a00021.html#vTaskList> for task list.

**Note:** Using USB requires that you add /lpc\_chip\_15xx/inc/usbd directory to your project paths.

Switch to **Includes** tab and select **All configurations**. The IDE allows us to set both Debug and Release configurations on one go.



Select one language at a time and add the directory to each language (there are 4 languages listed). Remember to check **Is a workspace path**.

