



Optimizing FreeRTOS

Lab Hand Out

Contents

1 Lab Overview	3
2 Prerequisites.....	3
3 Using MCUXpresso IDE and Task Aware Debugging	3
3.1 Basic Debugging	3
3.2 FreeRTOS Task Aware Debugging.....	8
4 FreeRTOS Advanced	9
4.1 FreeRTOS Optimization	9
4.2 FreeRTOS with SystemView.....	11
4.3 FreeRTOS Compiler Optimization	14

1 Lab Overview

This lab will use a FRDM-K64F board to demonstrate FreeRTOS debugging techniques with MCUXpresso IDE.

2 Prerequisites

The following items are needed to complete this hands-on lab. This has already been installed on your computer:

- Hardware
 - One FRDM-K64F
- Software
 - MCUXpresso SDK for FRDM-K64F
 - MCUXpresso IDE: <http://nxp.com/mcuxpresso/ide>
 - Segger SystemView: <https://www.segger.com/products/development-tools/systemview/>

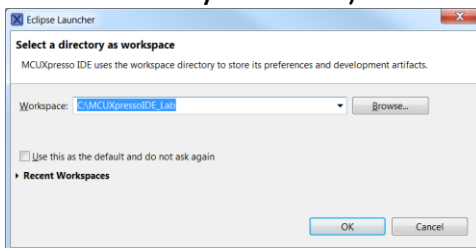
3 Using MCUXpresso IDE and Task Aware Debugging

3.1 Basic Debugging

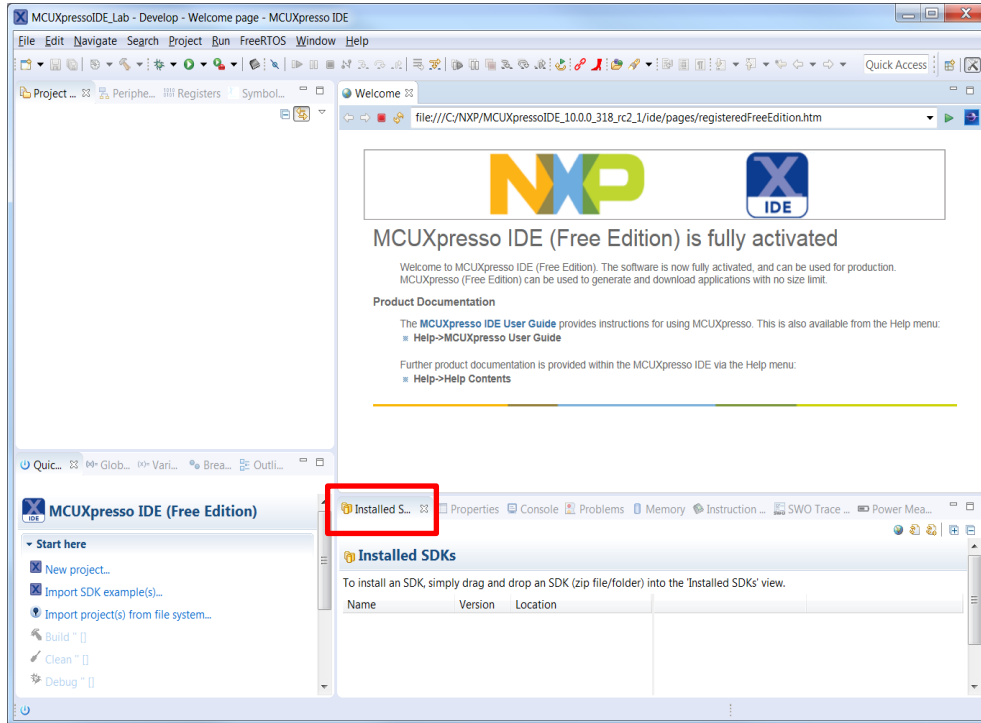
1. Open MCUXpresso IDE



2. Set the workspace directory to **C:\MCUXpressoIDE_Lab** (or your choice of other new directory location) and click on OK.

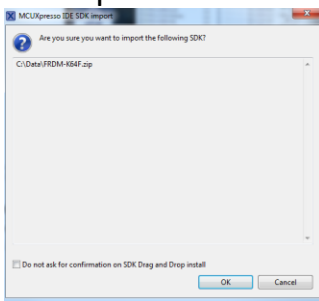


3. Open the **Installed SDKs** view within the MCUXpresso IDE

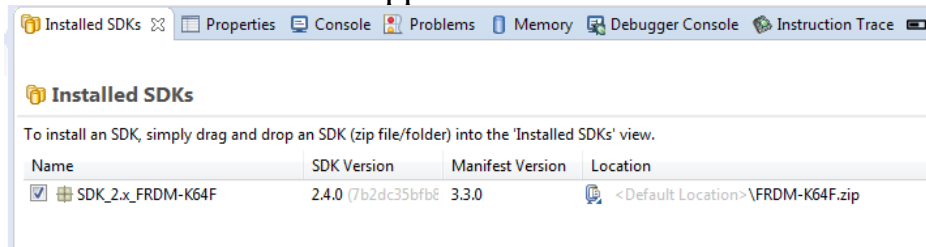


4. If the FRDM-K64F SDK is not in that view already, open the MCUXpresso SDK folder on your desktop, and drag and drop the FRDM-K64F SDK .zip file into the **Installed SDKs** view.

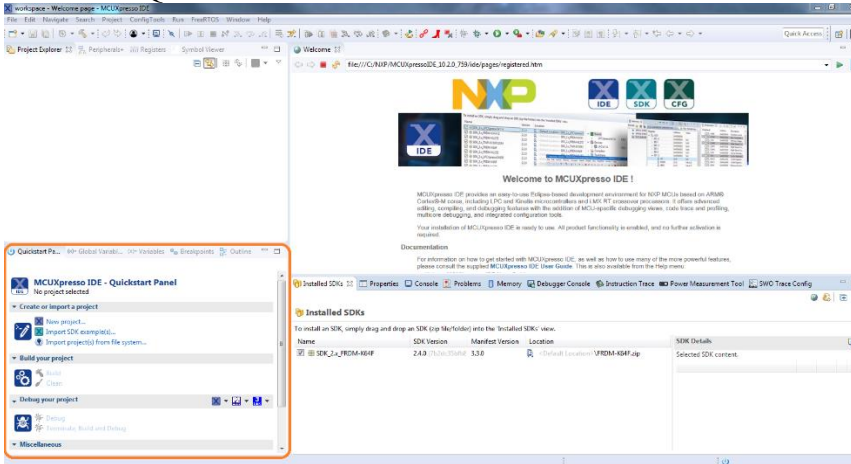
5. You will get a pop-up dialog that looks similar to below. Click on **OK** to continue the import:



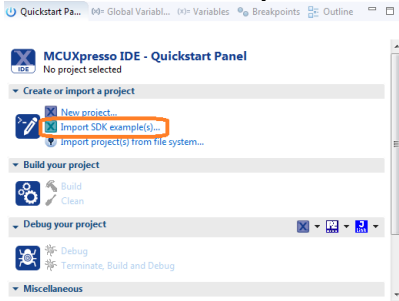
6. The installed SDK will appear in the Installed SDKs view as shown below:



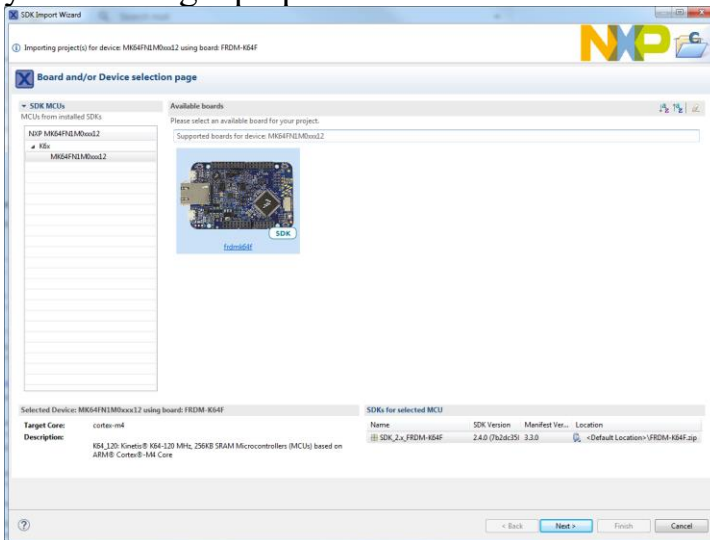
7. Find the Quickstart Panel in the lower left hand corner



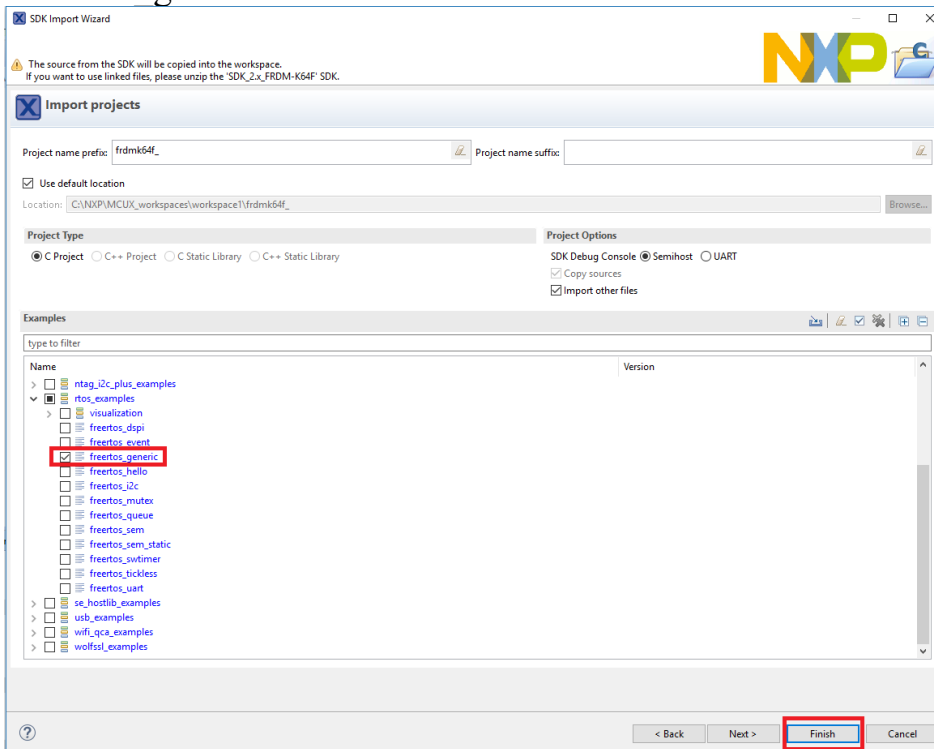
8. Then click on Import SDK example(s)...



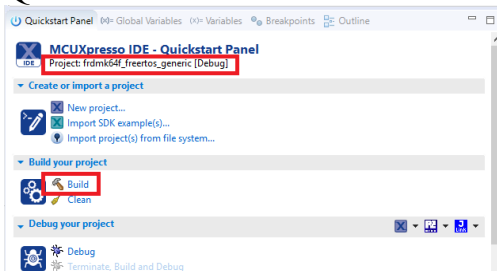
9. Click on the **frdmk64f** board to select that you want to import an example that can run on that board, and then click on Next. Note there may be more boards listed on your training laptop.



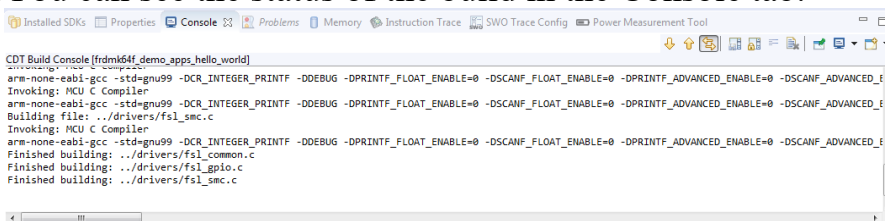
10. Select the “rtos_examples” category and then put a check next to “freertos_generic” and click on Finish.



11. Click on the **frdmk64f_freertos_generic** name in the Project Explorer panel. Verify it's selected in the Quickstart panel and then click on “**Build**” in the Quickstart Panel window to build the project.

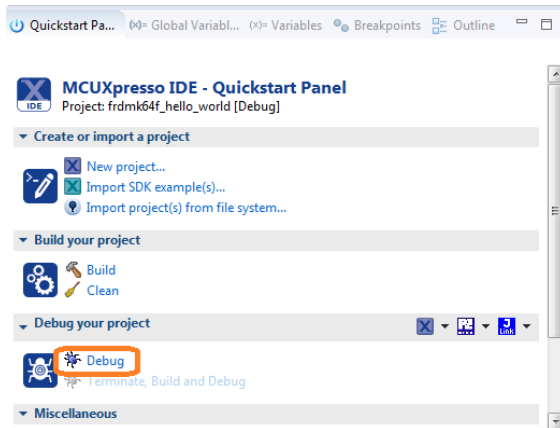


12. You can see the status of the build in the Console tab.

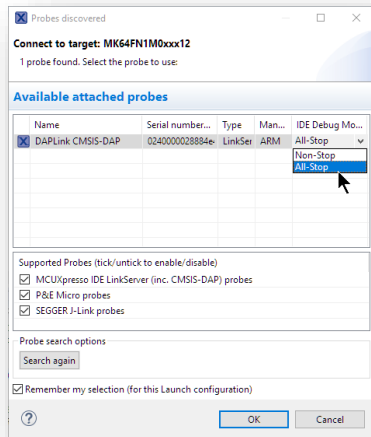


13. Make sure the FRDM-K64F board is connected from your laptop to the micro USB connection labeled “SDA USB”

14. Click on **Debug** in the Quickstart Panel.

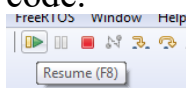


15. MCUXpresso IDE will probe for connected boards and should find the MBED CMSIS-DAP debug probe that is part of the integrated OpenSDA circuit on the FRDM-K64F. In the dialog box that comes up, change the IDE Debug Mode to “All-Stop” which allows for a thread aware debug view.

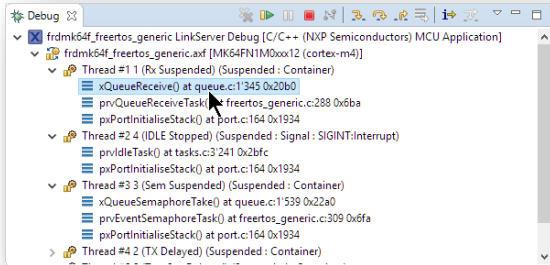



16. The firmware will be downloaded to the board and the debugger started.


17. Start the application by clicking the "Resume" button and see the messages in the console. Use the other debug buttons to Pause, Step In, Step Out, and Step Over code.



18. Press the “Pause” button and explore all the FreeRTOS tasks. Click on a function (top of the thread) and debug it by stepping out or stepping over.

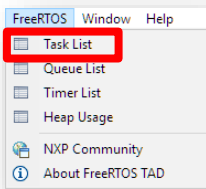


19. When finished, press the Terminate button to stop the debug session. 

20. If needed, you can get back to the Develop view by clicking on the icon in the upper right corner. 

3.2 FreeRTOS Task Aware Debugging

1. Using the same `freertos_generic` project as the last section, start a debug session, run the code, and then hit pause in the debugger.
2. Open the FreeRTOS Task List by going to FreeRTOS in the menu bar and select Task List. Review the information that shows all the tasks currently running.

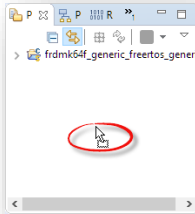


3. You can save the task data as a .csv file if desired by clicking on the floppy disk icon.
4. View the FreeRTOS Queue List by going to **FreeRTOS->Queue List**
5. View the FreeRTOS Timer List to see all the RTOS timers being used by going to **FreeRTOS->Timer List**
6. View the FreeRTOS Heap Usage to see the status of Heap and Memory allocation by going to **FreeRTOS->Heap Usage**
7. When finished, press the terminate icon and wait for instructor. Do not proceed to next section yet.

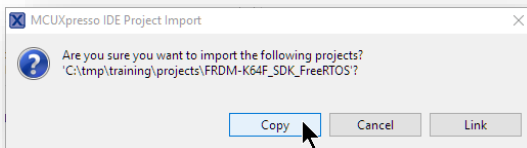
4 FreeRTOS Advanced

4.1 FreeRTOS Optimization

1. Import the **FRDM-K64F_SDK_FreeRTOS** project found on the desktop by dragging the folder into the empty space in the Project Explorer view. This project is designed to show off FreeRTOS features.



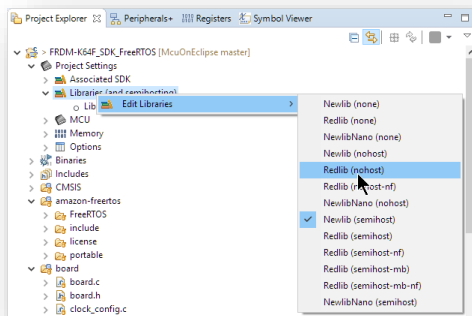
2. In the dialog box that comes up, select “Copy”



3. With the project now imported, build the project and make note of the initial code and data size needed.

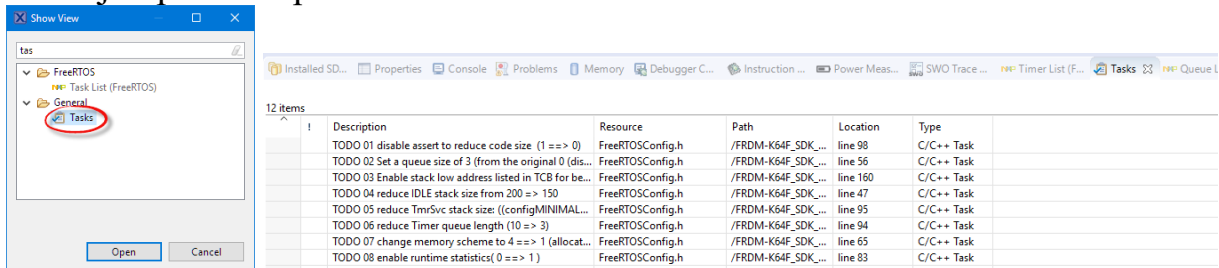
Memory region	Used Size	Region Size	%age Used
PROGRAM_FLASH:	56840 B	1 MB	5.42%
SRAM_UPPER:	46800 B	192 KB	23.80%
SRAM_LOWER:	0 GB	64 KB	0.00%
FLEX_RAM:	0 GB	4 KB	0.00%

4. Change the libraries and semihosting/printf to disable and save code space
 - a. Use the RedLib library by opening the **Project Settings** category in the project, and right clicking on the **Libraries** folder and selecting **RedLib (nohost)**.



- b. Rebuild and check code size impact

- For the rest of this lab, each specific item to change is associated with a TODO. You can view a list of the TODOs by opening the Tasks view: **Windows->Show View->Other** and then **General->Tasks**. You can double click on the items in the list to jump to the specific source location.



- These changes will all be done in the `\source\FreeRTOSConfig.h` file
- TODO01:** Disable ASSERTs for release version by changing the `#if 1` on line 98 to `#if 0`. Recompile the project and make note of the code size change.
- TODO02:** Enable the Queue Registry by setting `configQUEUE_REGISTRY_SIZE` on line 56 to 1 and verify by debugging the project and using TAD to view the FreeRTOS Queues.

The next few steps will be used to optimize the stack size

- TODO03:** Enable the recording of the stack address in order to debug stack size usage. On line 160 change `configRECORD_STACK_HIGH_ADDRESS` to **1**
- TODO04:** Decrease the stack size for the IDLE task to save RAM. On line 47 change `configMINIMAL_STACK_SIZE` to **150**.
- TODO05:** Reduce the stack for the TmrSvc task which controls the FreeRTOS timers to save RAM. On line 95 change `configTIMER_TASK_STACK_DEPTH` to **150**
- TODO06:** Reduce the Timer queue length to save RAM. On line 94 change `configTIMER_QUEUE_LENGTH` to **3**.
- Compile and Debug the project. Run for a while and then hit pause in the debugger.
- Look at the FreeRTOS->Task List view and note the stack usage. Remember that the values put into the code are in 32-bit units.

TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
> 1	Timers	0x20000cd0	Blocked	0 (0)	276 B / 496 B		
> 2	Master	0x20000f48	Blocked	0 (1)	324 B / 496 B		
> 3	first_task	0x200011c0	Blocked	0 (4)	336 B / 496 B		
> 4	IDLE	0x20001498	Running	0 (0)	88 B / 592 B		
> 5	Tmr Svc	0x200017f8	Blocked	0 (9)	365 B / 592 B	TmrQ (Rx)	
> 6	second_task	0x20001a70	Blocked	0 (3)	136 B / 496 B		
> 7	Slave	0x20001d40	Suspended	0 (1)	220 B / 496 B	IPC_Sem (Rx)	

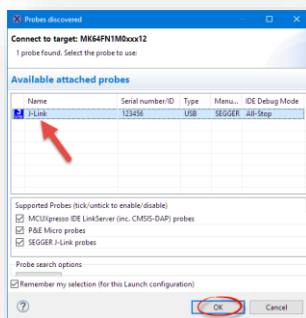
15. **TODO07:** Now change the heap allocation scheme to “Allocate Only”. On line 64 change `configFRTOS_MEMORY_SCHEME` to 1.
16. **TODO08:** Enable Runtime stats by modifying line 83 to change `configGENERATE_RUN_TIME_STATS` to 1.
17. Compile and run the project. View the Task List again to see which tasks are running the most often.

TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
> 1	Timers	0x20000cc8	Blocked	0 (0)	276 B / 496 B		0x5 (0.0%)
> 2	Master	0x20000f38	Blocked	0 (1)	324 B / 496 B		0x1 (0.0%)
> 3	first_task	0x200011a8	Blocked	0 (4)	324 B / 496 B		0xb (0.0%)
> 4	IDLE	0x20001478	Running	0 (0)	88 B / 592 B		0x1771 (0x1771)
> 5	TmrQ	0x200017c8	Blocked	0 (9)	365 B / 592 B	TmrQ (R)	0x2 (0.0%)
> 6	second_task	0x20001a38	Blocked	0 (3)	136 B / 496 B		0x9 (0.0%)
> 7	Slave	0x20001c18	Suspended	0 (1)	220 B / 496 B	IPC_Sem (R)	0x0 (0.0%)

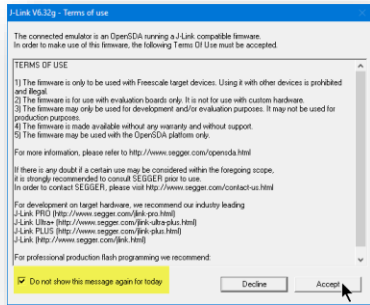
18. There’s a performance issue in this code. Can you spot it? While the code is running, hold down SW3, and then look at the Runtime statistics again.
19. This can be fixed. Add the following line in `FreeRTOS_Timers.c` on line 84 to add a delay inside the `while()` loop that waits for SW3 to be unpressed. Run again and checks the Runtime statistics.
`vTaskDelay(pdMS_TO_TICKS(50));`
20. When finished, hit the terminate icon to stop the debug session.

4.2 FreeRTOS with SystemView

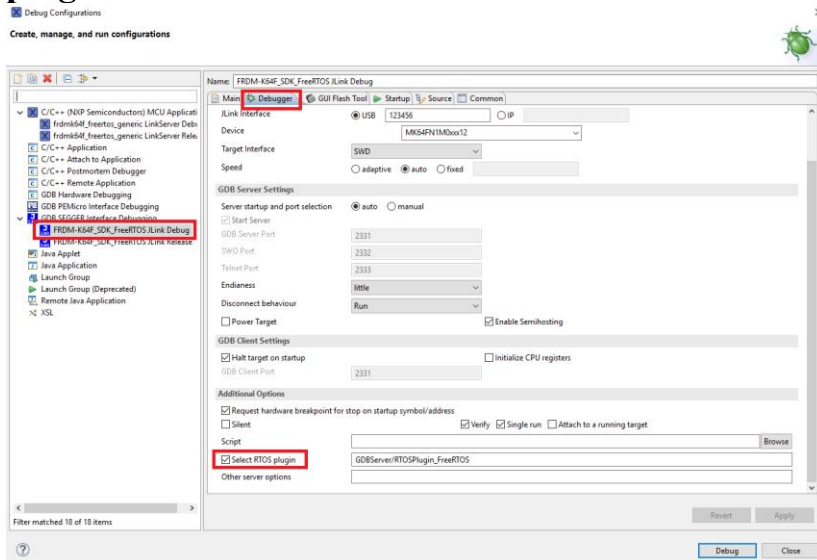
1. SystemView requires the Segger debug interface. We can change the debug interface on the OpenSDA circuit on the FRDM-K64F with the following steps:
 - a. Unplug the board
 - b. Press and hold the reset button on the board while plugging the board back into the computer
 - c. The board will enumerate as MAINTENANCE on your computer
 - d. Copy `OpenSDA\SEGGER_02_OpenSDA_FRDM-K64F.bin` to drive
 - e. Unplug the USB cable and replug back in.
2. Inside of MCUXpresso IDE, hold down the shift key and then click on Debug from the Quickstart menu. This will rescan for debug probes to find the new JLink firmware.
3. It should find the JLink software. Click on OK.



- You may get the following dialog box. Click on the box to not show the message again and then click on Accept.

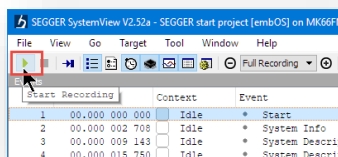


- Terminate the debug session.
- Go to Run->Debug Configurations from the menu bar at the top
- Make sure to select the “**FRDM-K64F_SDK_FreeRTOS JLink Debug**” configuration and then on the **Debugger** tab, scroll down to check “**Select RTOS plugin**” so it is enabled. Hit **Close** to save.

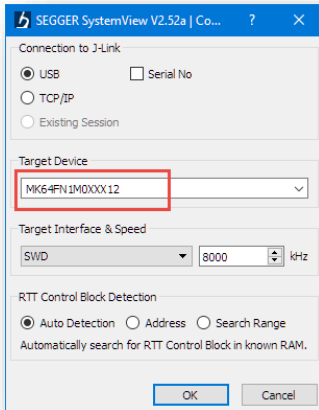


- Click on “Debug” from the Quickstart Window and click on the “Resume” button to begin running the code in the debug session.

- Launch SystemView
- Press Start Recording



11. Specify the target device: MK64FN1M0XXX12 and hit OK.



12. It will record data. After a few seconds, press **Stop Recording**.



13. Explore the SystemView interface. Use the mouse wheel to zoom in and out of the timeline, and the mouse to move the time.

14. You can see the list of events in the **Events** window. Click on an event to see it in the Timeline view.

15. View the messages sent to the terminal in the **Terminal** window. These are generated by a Segger API used in the code.

```
static void first_task(void *pvParameters) {
    if (!taskCreate(&second_task, "second_task", 500/sizeof(StackType_t), NULL, 3, NULL) != pdPASS) {
        PRINTF("Task creation failed!\r\n");
        vTaskSuspend(NULL);
    }
    /* dummy code, print counter and delay */
    for (int counter = 0; counter++; ) {
        #if API_CONFIG_USE_SEGGER_SYSTEMVIEW
            SEGGER_SYSVIEW_PrintTarget("first task counter: %d ", counter++);
        #endif
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
```

16. View the run count, statck size, task frequency, and more in the **Contexts** window.

17. **TODO09:** Reduce the tick frequency so don't wakeup so often. On line 45 change **configTICK_RATE_HZ** to **200**.

18. Compile and record the system again like done previously. You should now see reduced number of ticks.

19. **TODO10:** Use tickless mode. On line 43 change **configUSE_TICKLESS_IDLE** to **1**.

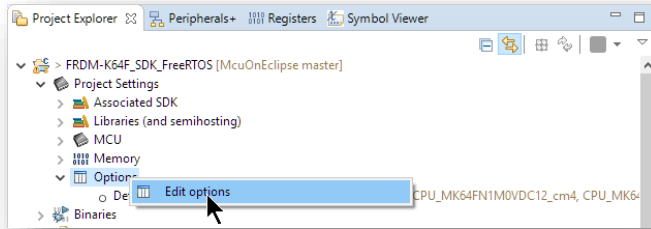
20. Compile and run the code again and see a greatly reduced number of interrupts.

21. When finished experimenting with SystemViewer, put OpenSDA back into bootloader mode (holding down reset) and load **OpenSDA\DAPLINK_k20dx_frdmk64f_if_crc_legacy_0x5000.bin** on it.

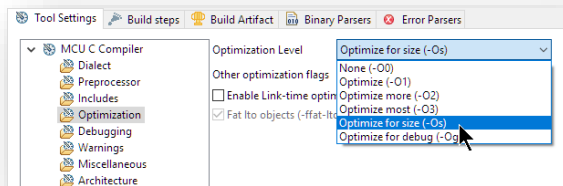
4.3 FreeRTOS Compiler Optimization

This section is last because these compiler optimizations can impact debugging.

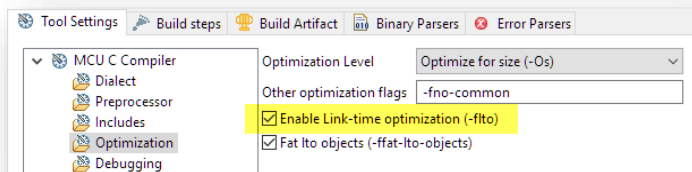
22. Increase the optimization level used by the compiler by opening Project Settings->Options and right click to select Edit Options.



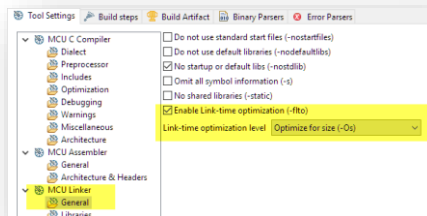
23. Inside the Optimization category, change the Optimization level to -Os and then recompile. Note the code size changes.



24. Enable Link-Time optimization by enabling it for both the compiler and linker.
 - a. Under **MCU C Compiler->Optimization** select “**Enable Link-Time optimization (-flto)**”.



- b. Under **MCU Linker->General**, select “**Enable Link-time Optimization (-flto)**” and set the optimization level to -Os.



25. Compile and compare code sizes.