

MCUExpresso Install, Setup, and Notes for use with FRDM K64F

March 14, 2018

1. Setup K64F

(May not be needed for some boards or if developing on mac)

- Plug board into USB
- If the drive is named MBED (MBED (K:), MBED (E:), etc.) then follow the directions to update the bootloader
- <https://os.mbed.com/blog/entry/DAPLink-bootloader-update/>
 - Download the binary file 0244.k20dx_bootloader_update_0x5000.bin
- Then install DAPLINK: <https://armmbed.github.io/DAPLink/>
 - Search K64 (select FRDM K64F).
 - The board should appear as
 - Download the binary file 0244.k20dx_frdmk64f_0x5000.bin

2. Setup serial port

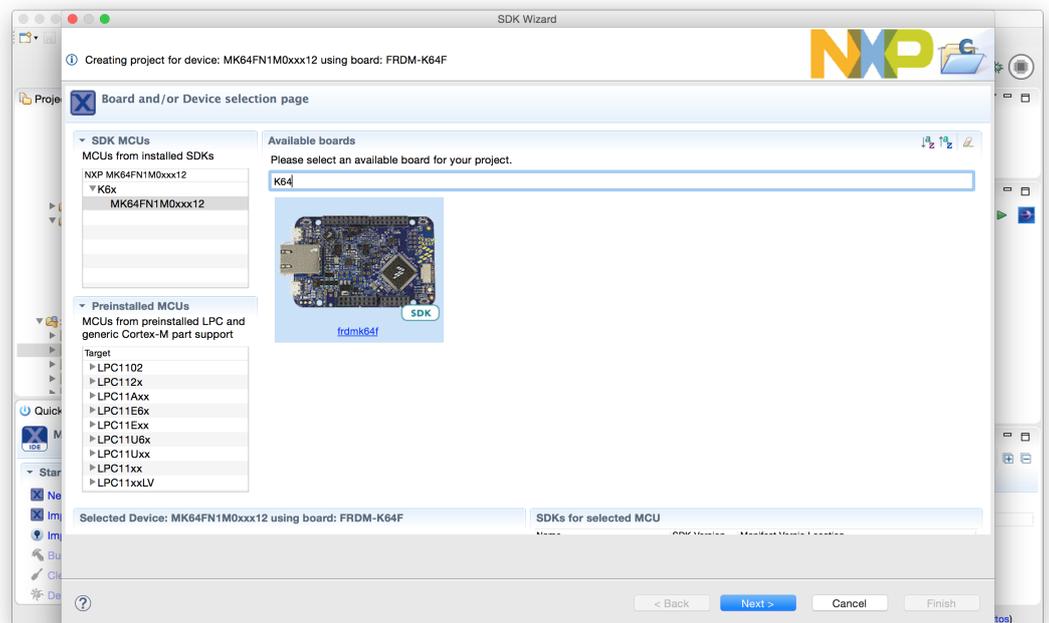
- Plug board into USB. It should appear as DAPLINK (K:)
- Windows
 - Use device manager to find Ports COM#
 - Use Putty to connect to COM port from device manager with baud rate 115200
- Mac
 - Open terminal and type: “ls /dev/tty.usbmodem*” you should see the name of the USB port which the K64F is plugged into (i.e. /dev/tty.usbmodem1412).
 - Type “screen /dev/tty.usbmodem1412 115200”

3. Setup Git or SmartGit (a GUI for git)

- Setup Git (Command Line) or Smart Git (GUI Client)
 - For Smart Git: choose github as hosting provider, set master password if needed, make github acct if you don't have one, authenticate and enable synteo to access your github
- Clone the repo <https://github.com/ucb-ee192/SkeletonMCUX>

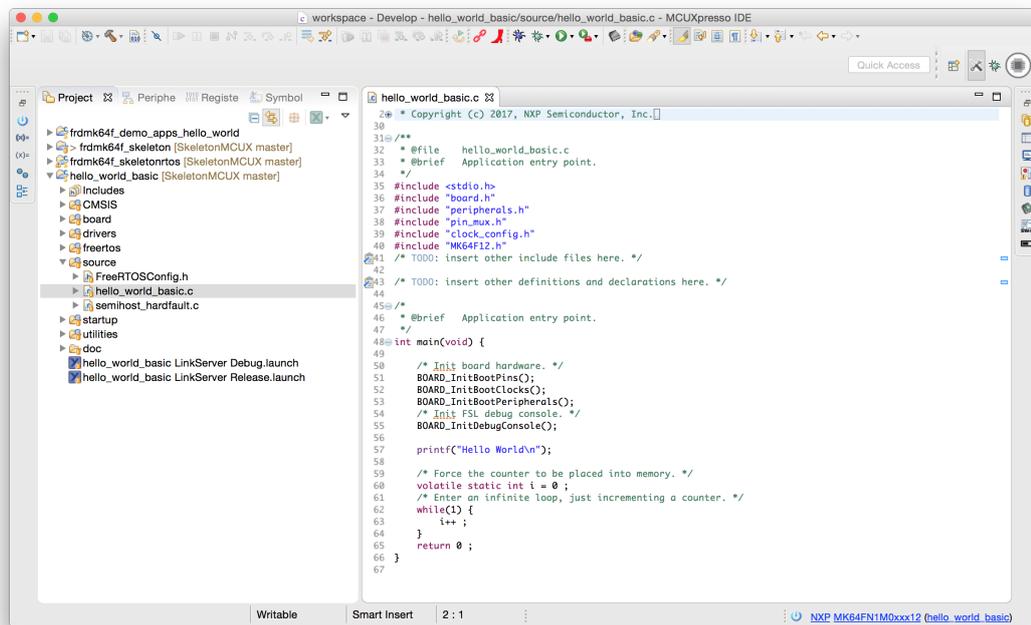
4. Install and setup MCUExpresso

- Install MCUExpresso Version 10.1.0 **here**
 - Version 10.1.0 is not the latest build. Navigate to the "previous" tab and download 10.1.0
- Download The K64F SDK zip file **here** (use your Berkeley email address for access)
- Start MCUExpresso
- Drag the K64F zip file into the "Installed SDK's" subwindow and click okay. This will install the K64F SDK
- Check that the K64F SDK installed correctly. Try File→ New→ Project→ MCUExpresso IDE→ New C/C++ Project. Search K64- you should see the K64F as an option.

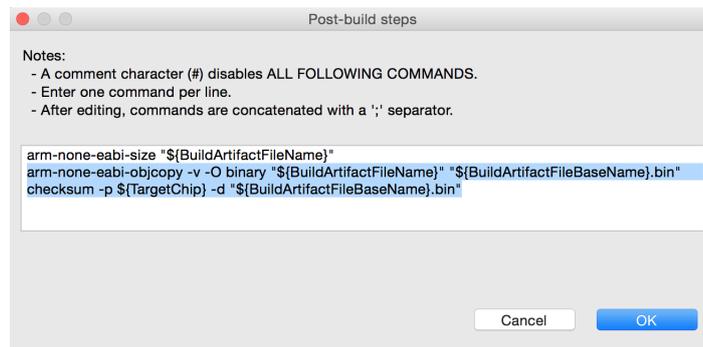


5. Import HelloWorld and FreeRTOS examples

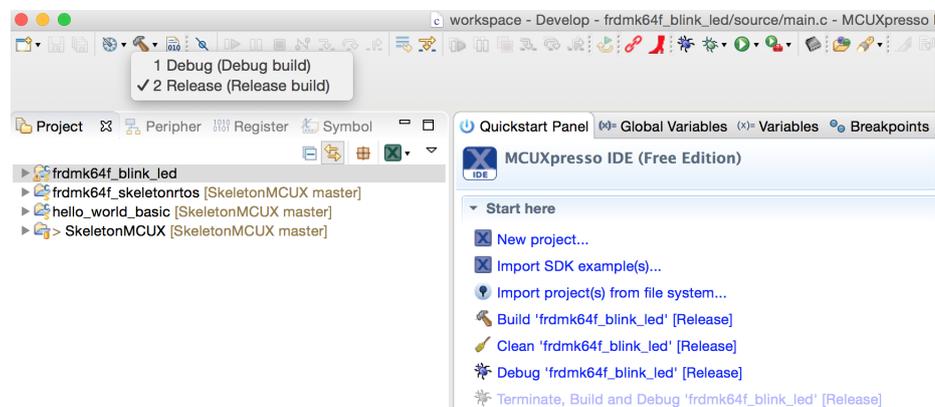
- File → Open Projects from File System
- Select SkeletonMCUX folder you cloned from github
- projects: frdmk64_skeletonrtos, hello_world_basic
- Hello World
 - Open the hello_world_basic folder in the left project explorer
 - The main file is in hello_world_basic → source → hello_world_basic.c



- To build select Debug ‘hello_world_basic’ [Debug] (in the quick start menu or the blue bug on the top toolbar).
- Ignore any firewall requests that might occur (windows machines)
- Probes Discovered window: should show CMSIS-DAP. → OK
- Hello World should print. The program will then increment a counter in an infinite loop.
- You can exit by clicking the red square on the top tool bar
- FreeRTOS Example
 - The main file is in frdm64f_skeletonrtos → source → main.c
 - To build select Debug ‘frdm64f_skeletonrtos’ [Debug] (in the quick start menu or the blue bug on the top toolbar).
 - It should print EE192 Spring 2018 16 Dec 2017 v0.0 then blink the onboard LED
- Release Build in MCUXpresso (Note: use PRINTF not printf)
 - Configure MCUXpresso to build a .bin file for drag & drop flash programming
 - * Select the project.
 - * Navigate to Project → Properties → C/C++ Build → Settings → Build Steps
 - * Select edit on “Post-Build steps”. Uncomment the lines with commands “arm-none-eabi-objcopy” and “checksum”.



- Change the build settings to “Release” by clicking the hammer on the top toolbar and selecting “Release”



- Build the release project (from quickstart panel or by clicking the hammer)
- Plug in your K64f board. It should appear as DAPLINK
- Copy the file “Project_Location/Project_Name/Release/project_name.bin” to DAPLINK drive
- A green light will blink rapidly as the program is flashed to the MCU
- After the green light stops blinking the program has been successfully flashed. Hit the reset button to run it

Random Notes

1. There is already a running debug session for the launch.
 - Use task manager (windows) or "kill" command line utility on Mac/Linux to kill any tasks called:
 - redlinkserv
 - arm-none-eabi-gdb
 - crt_emu_*
 - Close and reopen project
 - Stop on console view?
 - Reopen MCUExpresso
 - Unplug and Replug in K64F board
2. To see queue in FreeRTOS View Queues
 - `vQueueAddToRegistry(log_queue, "PrintQueue");`
3. Watch out for printf, fragmenting memory
 - Section 12.4 Inappropriate Use of printf() and sprintf(). printf() and sprintf() may call malloc(), which might be invalid if a memory allocation scheme other than heap_3 is in use. See section 2.2, Example Memory Allocation Schemes, for more information.
 - Add printf-stdarg.c instead. Note: use
 - Printf-stdarg.c does not handle floating point. Need to use malloc intensive floating point printf (builtin).
4. FreeRTOS notes: use FreeRTOS → TaskList to show memory usage
 - If running out of stack in FreeRTOSConfig.h change:
 - `#define configMINIMAL_STACK_SIZE ((unsigned short)128)`
 - `// changed for larger idle task- watch heap size...`
 - `#define configTOTAL_HEAP_SIZE ((size_t)(16 * 1024))`
 - Task runtime with percentage value. Both `configUSE_TRACE_FACILITY` and `configGENERATE_RUN_TIME_STATS` need to be set to 1, except requires bunch of other enables...
 - How to add watch to stack pointer?
5. Memory view: Use debug view instead of develop view. Memory will appear in console

Advanced Setup

1. Change pins if needed to enable LEDs: (See Lab 3 Getting started with MCUXpresso [here](#))
 - right click project name → MCUXpresso config tools → Open pins
 - Go to pins and turn on all LEDs. Add 33/PTE26, PTB21, PTB22
 - export to board directory, refresh and recompile.
2. change FreeRTOSConfig.h to be 1 KHz instead of 200
3. Quick settings → Set Floating point type → Enable hardware floating point
4. Quick Settings → Set Library Header type → NewlibNano (semihost)
5. NewlibNano: If your codebase does require floating point variants of printf/scanf, then these can be enabled by going to:
 - Project → Properties → C/C++ Build → Settings → MCU Linker → Managed Linker Script and selecting the " Enable printf/scanf float" tick box.
 - (Library selection can be reset by quick settings..., so do NewlibNano last)
 - Note: Quick settings: → SDK Debug Console → UART (changes to redlib...)
 - A further alternative is to put an explicit reference to the required support function into your project codebase itself. One way to do this is to add a statement such as: `asm (".global_printf_float");`
 - Unfortunately, as redlib does not support C++, we must use newlib for modern C++ projects.
 - PRINTF uses UART, printf uses semihost console.
6. Add timer component. Manage SDK Components, add driver: pit.c
 - (Note in general should be able to right click project name → MCUXpresso config tools → peripherals. However PIT module is missing, so needed to be added manually.)
7. Add explicit idle top level loop (non-real time)
 - `configUSE_IDLE_HOOK` must be set to 1 in FreeRTOSConfig.h for the idle hook function to get called.
 - `/* Idle hook functions MUST be called vApplicationIdleHook(), take no parameters, and return void. */`
8. To check stack usage etc, use FreeRTOS → Task List or FreeRTOS → Heap Usage

CAUTION NOTES TO BE ADDED

For reliability, smaller embedded systems typically don't use dynamic memory, thus avoiding associated problems of leakage, fragmentation, and resulting out-of-memory crashes. In Nadler & Associates smaller embedded projects, we've historically avoided any runtime use of dynamic memory (enforced by deleting malloc-family routine's object files from the runtime libraries) printf-stdarg.c distributed in the FreeRTOS Lab TCPIP example; this one only uses stack storage but does not implement floating point.

<https://community.nxp.com/thread/441637>