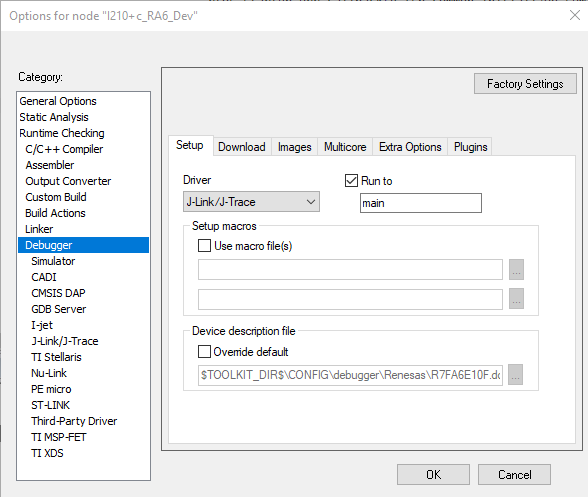
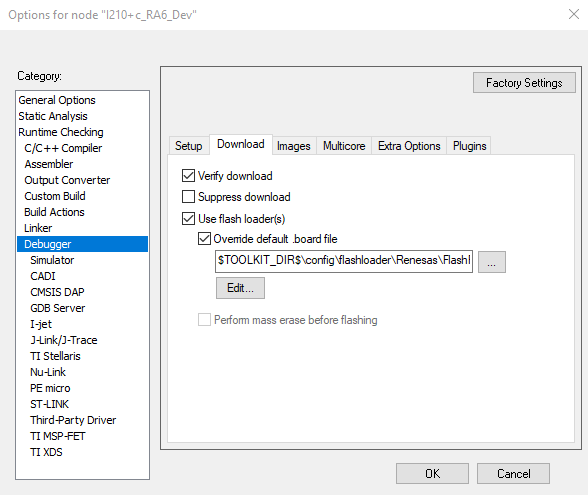
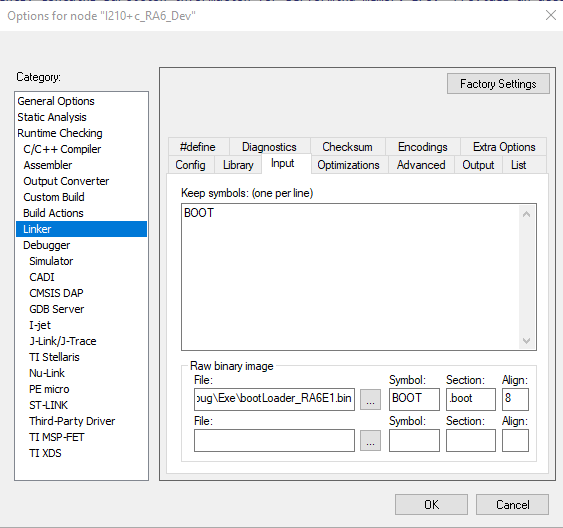
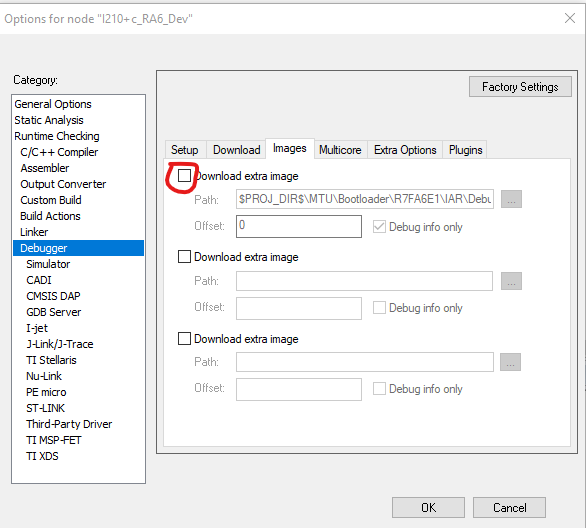
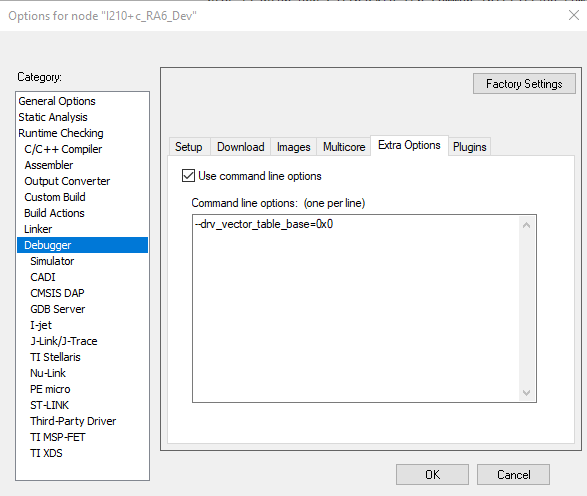
# Bootloader Instructions:

## General:

* A new RA6E1 bootloader has been added to the Samwise workspace and must be compiled before any application that uses it.
* The NXP\_K24 and RA6E1 bootloaders have been changed to use source files in the form filename\_BL.c/h to allow changes to the application to have minimal effect on the bootloader image.
* bootloader\_K24 compiles without error but is UNTESTED
* bootloader\_RA6E1 compiles without error and is being tested
* A batch script has been create to aid in determining when changes to the filename\_BL.c/h files have been made from the base file, filename.c/h
  + Requires WinMerge (or change script to support your diffing tool)
  + Looks for files with filename\_BL.c/h pattern and copies to the Bootloader/compare/bl folder as filename.c/h
  + Looks for a base file, filename.c/h, in the same folder as the filename\_BL.c/h file and, if found, copies that file to Bootloader/compare/base
  + Opens WinMerge on the compare/bl and compare/base folders to identify changes
  + The user is then required to manually modify the filename\_BL.c/h as needed.
  + Run the MTU\Bootloader\check\_BL\_files.bat to make the comparison
* The RA6E1 bootloader now creates a bootloader\_RA6E1.bin file that is included in the linker step of the application build. Add this file with the Linker 🡪 Input instructions below.
* The Debugger 🡪 Images extra image file, bootloader\_RA6E1.hex, no longer needs to be added. Remove this from your debugger configuration if previously included.
* The copy-from-external-flash-to-code-flash code has been ported to the RA6E1 bootloader
* See the bootloader test command section for new commands added to perform basic testing of the bootloader.

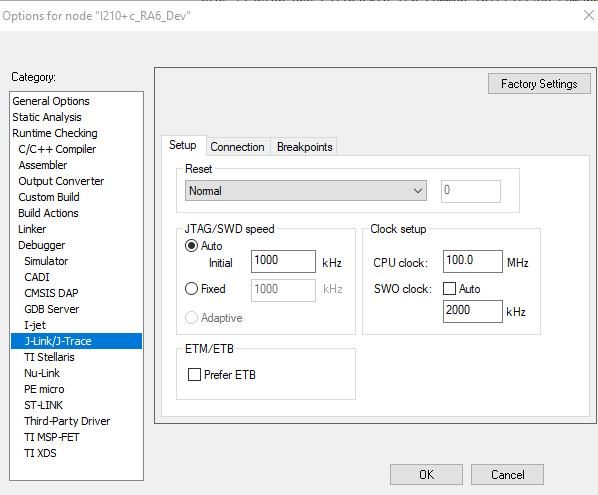
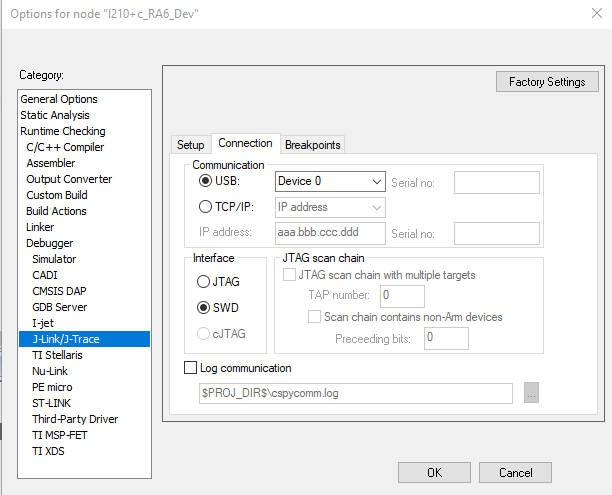
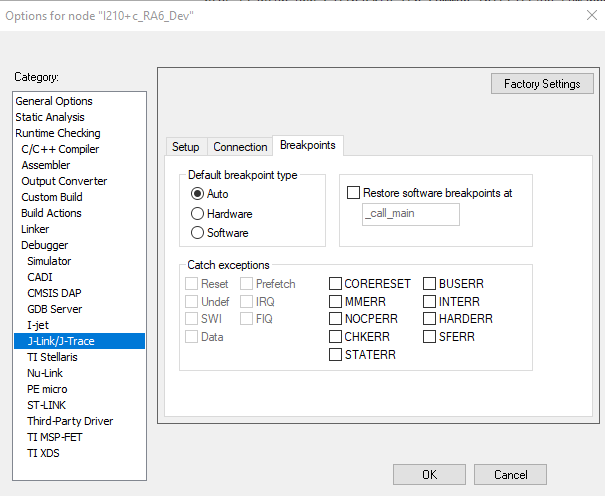
## IAR-EWARM Configuration:

To configure the IDE for J-Link, follow the steps below:

1. Open the Options for I210+c\_RA6\_Dev project
2. Set the Debugger 🡪 Setup options:  
   
3. Set the Debugger 🡪 Download options:  
   
4. Set the Linker 🡪 Input to the settings below:  
     
   File text: **$PROJ\_DIR$\..\Bootloader\R7FA6E1\IAR\Debug\Exe\bootLoader\_RA6E1.bin**Note: this creates a .boot section that can then be placed at address 0x0 by the (modified) fsp.icf linker script.
5. Set the Debugger 🡪 Image options:  
     
   text: **$PROJ\_DIR$\MTU\Bootloader\R7FA6E1\IAR\Debug\Exe\bootLoader\_RA6E1.out**Note: to debug the bootloader while running the application, check the Download extra image box and the debug info will be included. This will cause the debugger to not be able to stop at main (because now two main symbols exist), but you will be able to debug.
6. Set the Debugger 🡪 Extra options:  
     
   text: **--drv\_vector\_table\_base=0x0**
7. Set the Debugger 🡪 Plugins options – no change: FreeRTOS (with task local context), FreeRTOS and OpenRTOS)

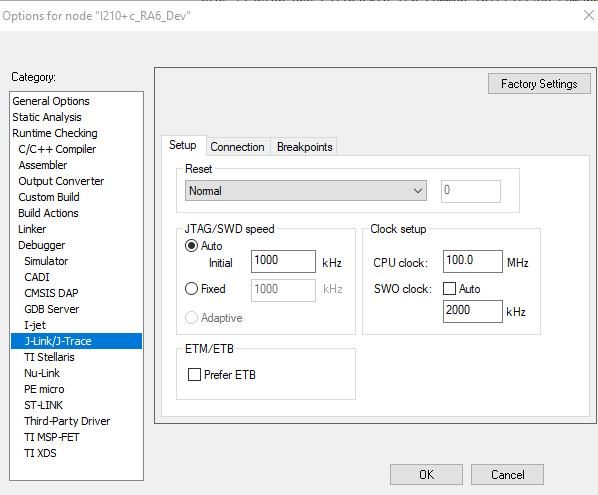
### J-Link Debugger:

For J-Link debugger, follow the steps below:

1. Set J-Link 🡪 Setup  
   
2. Set J-Link 🡪 Connection  
     
   Note: SWD is required (JTAG causes an unknown error when downloading)
3. Set J-Link 🡪 Breakpoints (use default):  
   

### I-Jet Debugger:

For I-Jet debugger, follow the steps below:

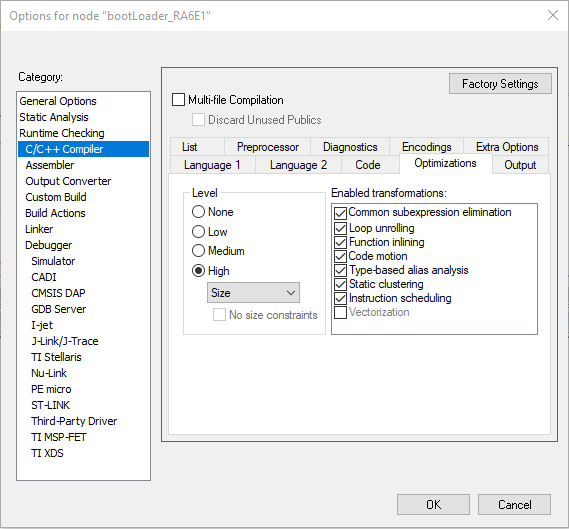
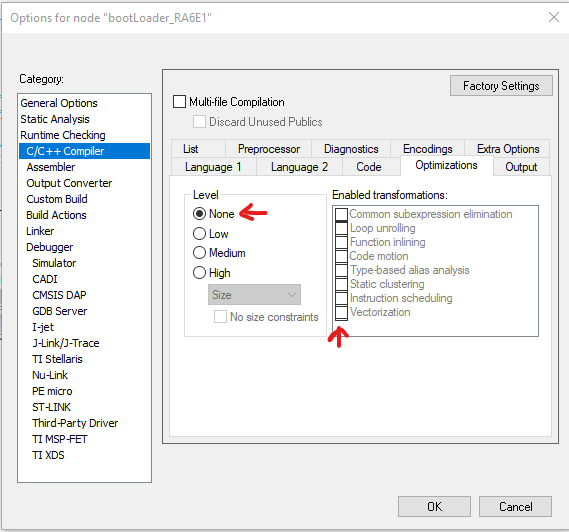
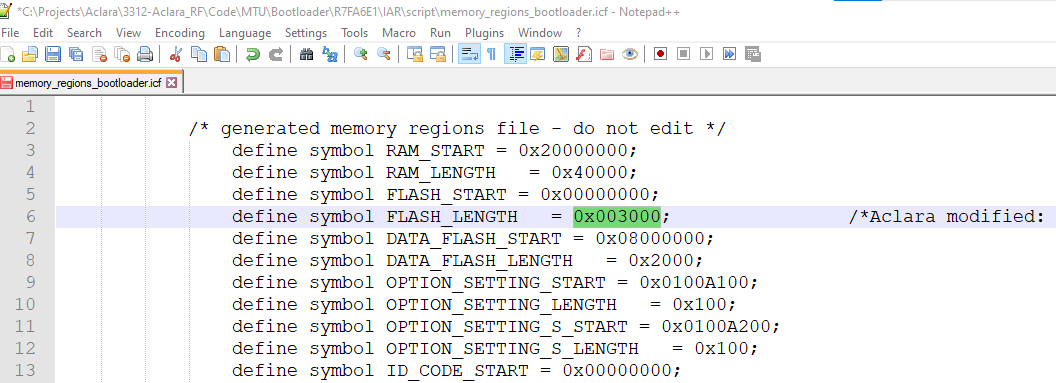
1. TBD
2. Set J-Link 🡪 Setup  
   

## Bootloader Debugging:

In order to fit into the 8KB bootloader space (0x0-0x1FFF), optimization has been set to the maximum optimization for size. Step-by-step debugging is very difficult when optimization is turned on. To debug the bootloader, optimization can be turned off and the bootloader memory can be extended to fit into the back-up bootloader area (0x2000-0x3FFF) by following the steps below

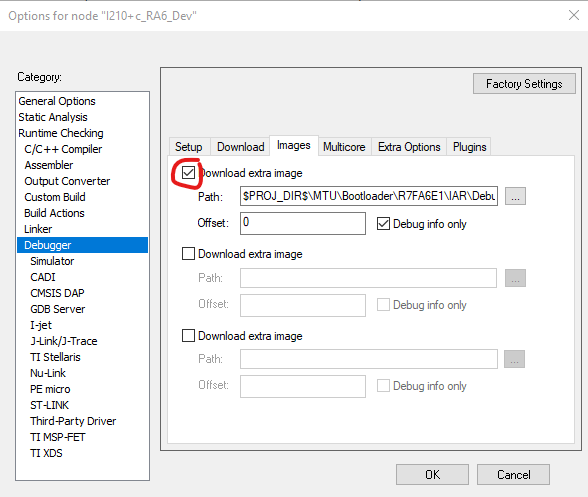
### Bootloader Debugging:

To change the bootloader optimization for easier debugging, follow the steps below:

1. Open the Options for BootLoader\_RA6E1 and select C/C++ Compiler🡪Optimizations and change to select None  
   🡪   
    Current Not optimized
2. Click OK to save the settings and exit Options
3. Open the {PROJ\_DIR} \MTU\Bootloader\R7FA6E1\IAR\script\memory\_regions\_bootloader.icf and change the FLASH\_LENGTH to 0x003000  
   
4. Rebuild All on the BootLoader\_RA6E1 project  
   Note: you can observe in the .map file that the size is now > 0x2000
5. Debug as needed.

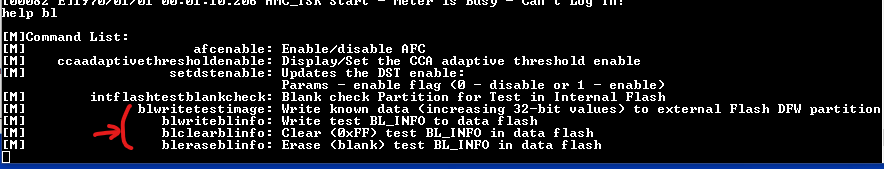
### Bootloader Debugging – Alternate Method:

You can debug the (optimized) bootloader from within the application with the following steps:

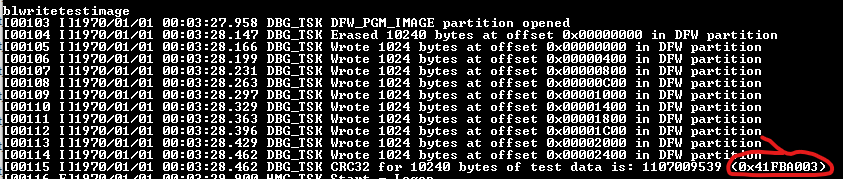
1. Check the Debugger🡪Images🡪Download extra images box to include (optimized) bootloader symbols  
     
   text: **$PROJ\_DIR$\MTU\Bootloader\R7FA6E1\IAR\Debug\Exe\bootLoader\_RA6E1.out**
2. Debug the I210+c\_RA6\_Dev project. The debugger will fail to set a breakpoint at main but should stop at the Bootloader Reset\_Handler function and debugging can then occur for both bootloader (optimized) and Application.

## Bootloader Test Commands (in I210+c App):

The bootloader has been tested using the new debug port commands below. These commands allow test data (10KB of increasing count uint32 values) to be programmed into the DFW\_PGM\_IMAGE partition (with CRC32 calculation printed), setup of the DFW\_BL\_INFO data into the DFW\_BL\_INFO partition (with manual CRC32 value entry), clearing of the BL\_INFO (all 0xFF), and erasing of the BL\_INFO (blank state).

In addition to the new commands, the help command has been improved to allow a string matching look-up (e.g. enter “help bl” to find all bootloader commands with the string “bl” as part of the command name).  


Commands:

**blwritetestimage**: writes test data to the DFW\_PGM\_IMAGE partition.  
  
Note: use the CRC32 in the blwriteblinfo command

**blwriteblinfo**: writes the 1st segment of the DFW BL INFO structure to store 10KB at offset 0xD0000 in the APP CODE partition. Use the CRC32 from the blwritetestimage for a valid CRC32 check or substitute an incorrect CRC32 to verify repeated retries of the programming. CRC32 parameter can be entered as hex or decimal  


**blclearblinfo**: writes the 1st segment of the DFW BL INFO structure as all 0xFFs.   


**bleraseblinfo**: erases the DFW BL INFO partition (making all data “blank”).   
