NXP Semiconductor

Debugging Bootloader and Application using KDS

By: Carlos Mendoza / Technical Information Center

About this document

This document explains how to use Kinetis Design Studio to debug both a bootloader and application at the same time, this will be done using GDB commands to specify an additional symbol file to be used in the debug session.

The bootloader used for this example is the project for the FRDM-K64F board provided in the KBOOT 1.2.0 named freedom_bootloader and the application is a bareboard led demo that was adapted to work with this bootloader by following the steps described in this document:

https://community.freescale.com/docs/DOC-256669

The document was created using the MK64FN1M0VLL12 MCU like the one in the FRDM-K64F board, but the same principles are applicable to any Kinetis MCU.

Software versions

The steps described in this document are valid for the following versions of the software tools:

- KDS v3.2.0
- KBOOT v1.2.0

Contents

1. Overview and concepts

1.1 Kinetis Bootloader

The Kinetis bootloader is a configurable flash programming utility that operates over a serial connection on Kinetis MCUs. It enables quick and easy programming of Kinetis MCUs through the entire product lifecycle from application development to final product manufacturing and beyond for updating applications in the field with confidence.

The bootloader is delivered in two ways: as full source code that is highly configurable; or pre-programmed by NXP into ROM or flash on select Kinetis devices. Host-side command line and GUI tools are available to communicate with the bootloader. Users can utilize host tools to upload/download application code via the bootloader.

The KBOOT 1.2.0 release contains the following PC-based host tools:

- **blhost** - command line debug tool to send individual commands to the bootloader .
- **Kinetis Updater** - GUI application to download and flash an application image.

And the following project types:

- **tower_bootloader** – bootloader designed to execute from target flash memory on the Tower platform.
- **freedom_bootloader** – bootloader designed to execute from target flash memory on the Freedom platform.
- **flashloader** – bootloader designed to execute from target RAM memory on either the Freedom or Tower platform.
- **flashloader_loader** – bootstrap loader designed to execute from flash memory on either the Freedom or Tower platform. This loader copies an image of the flashloader into RAM, then executes the flashloader from RAM.

1.2 GDB Server

The ‘GDB Server’ is a service or program which sits between the GDB inside Eclipse and the Debug Probe. GDB talks to the GDB server using TCP/IP and a port number, so that GDB Server can be either on your host machine, or on a remote machine. It translates and sends commands from the GDB in Eclipse to the Debug probe, things like setting breakpoints and so on. The communication between the GDB Server and the Debug Probe depends on the probe capabilities. It can be USB, TCP/IP, Serial or anything else.

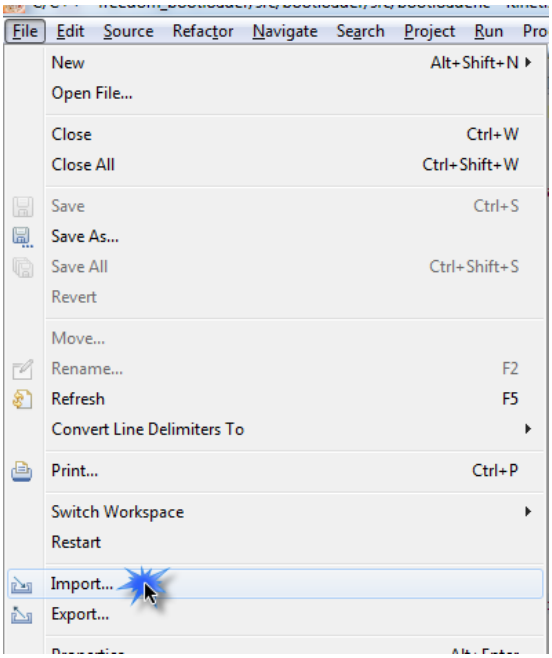
2. Flashing Bootloader and Application

2.1 Flashing freedom_bootloader project

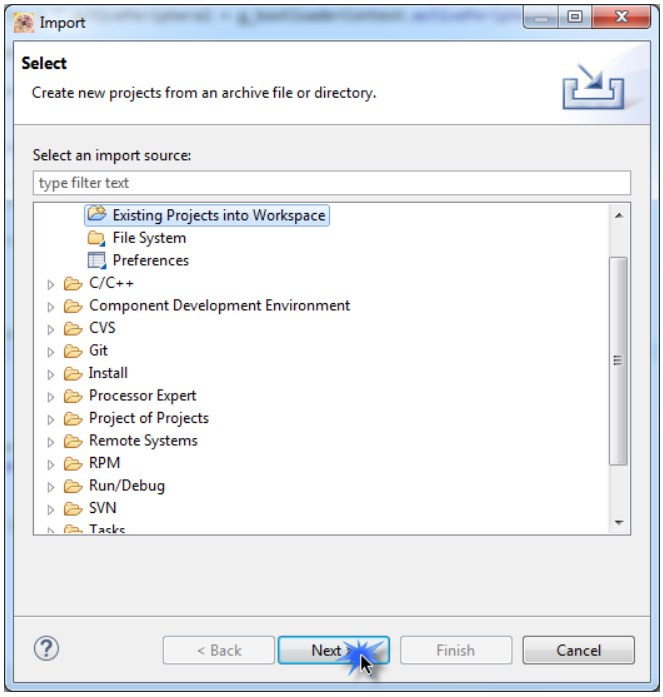
For this example we will use the **freedom_bootloader** project, this bootloader was designed to execute from target flash memory on the Freedom platform. The project can be found on the following path:

C:\Freescale\FSL_Kinetis_Bootloader_1_2_0\targets\MK64F12\kds\freedom_bootloader

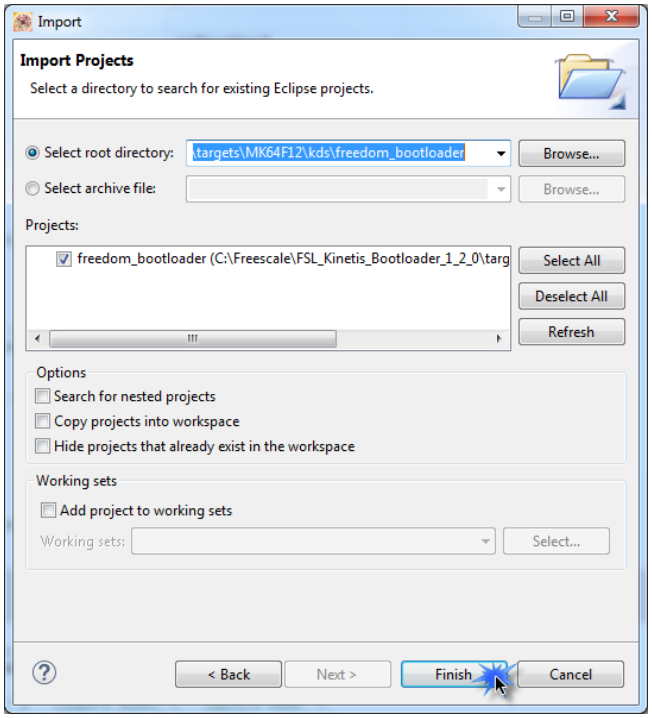
- Import the **freedom_bootloader** project to KDS by clicking on **Menu > File > Import**:



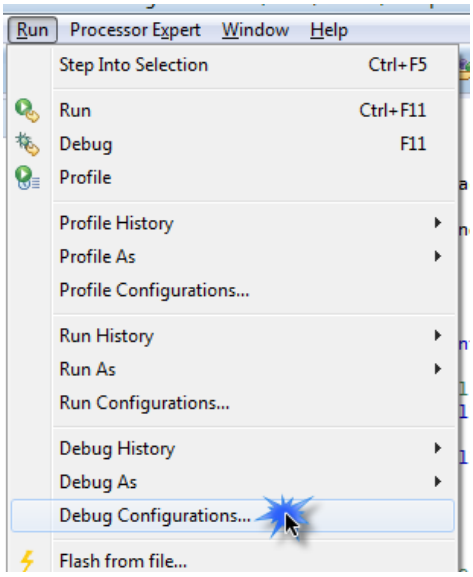
- Select **Existing Projects into Workspace** and click on **Next**:



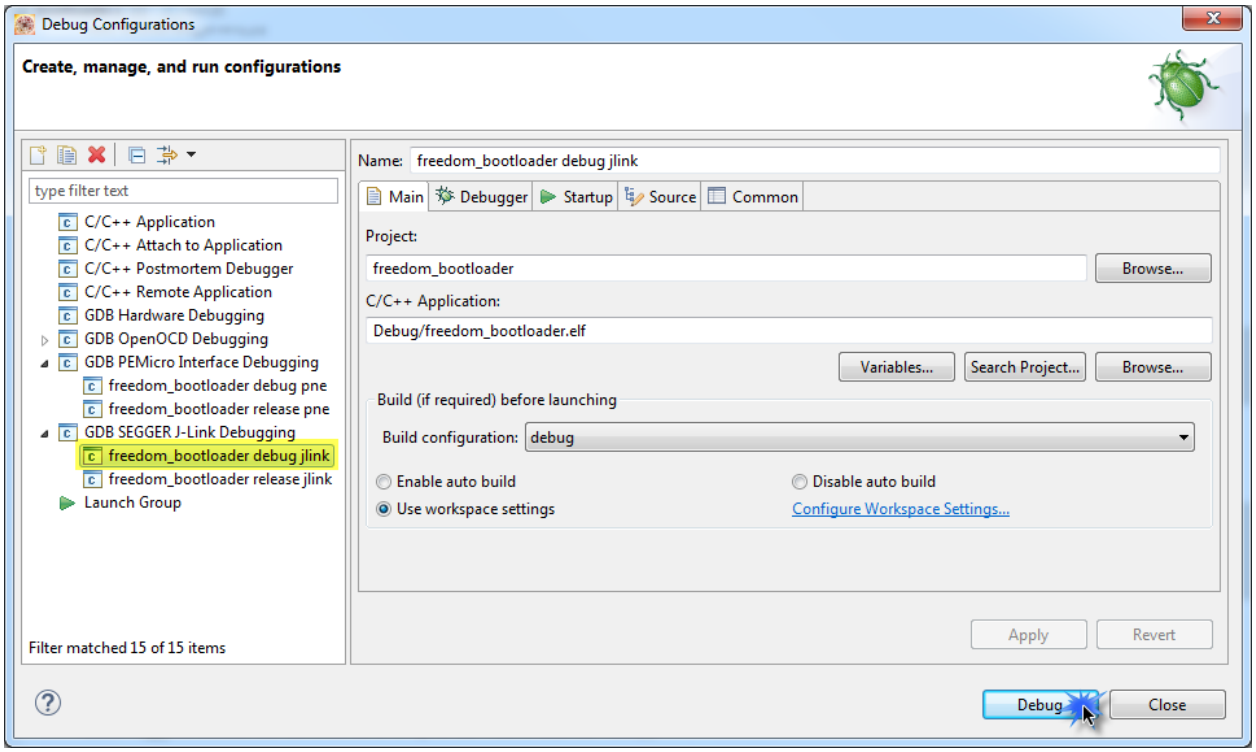
- Browse for the project and click on **Finish**:



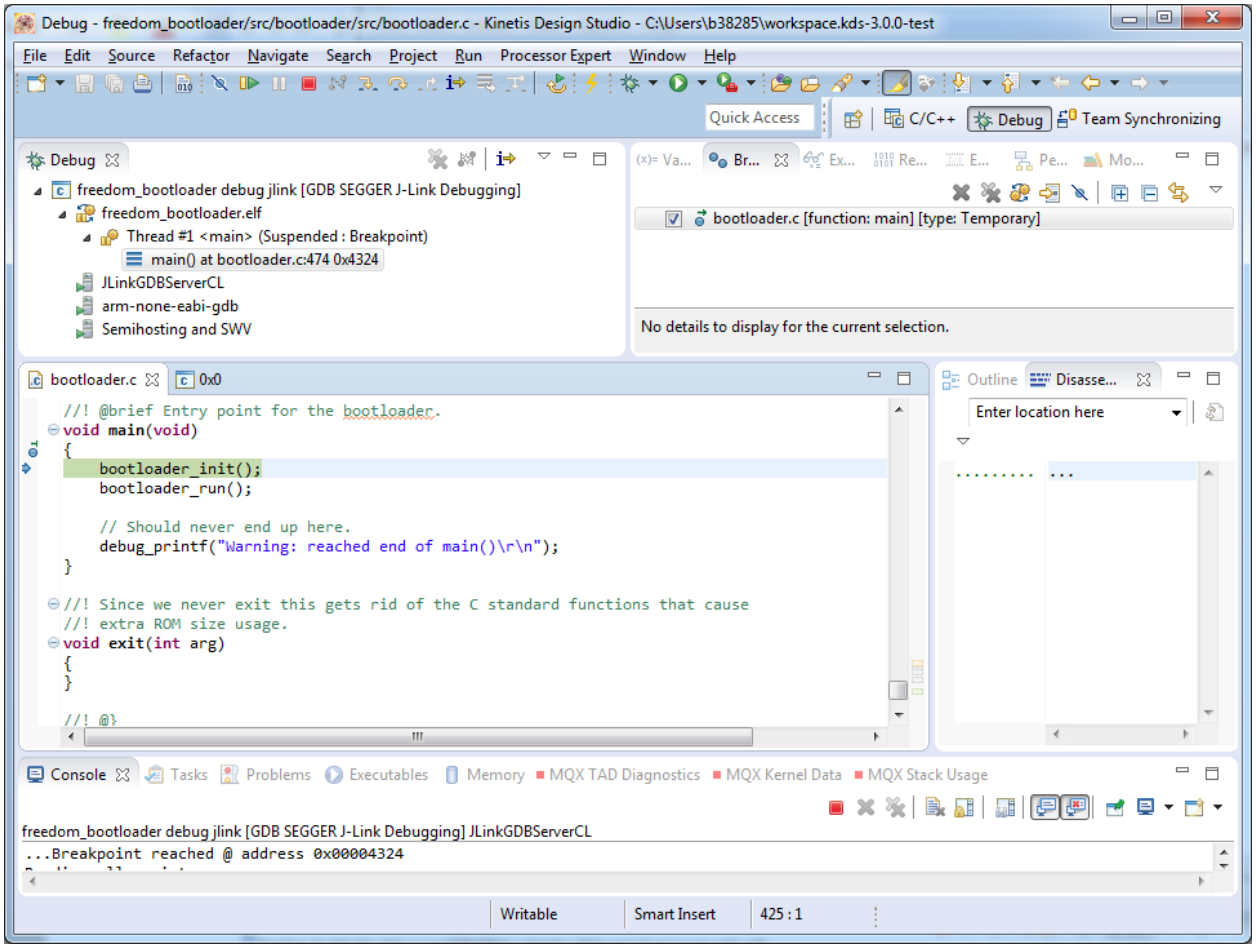
- Build the project and then click on **Menu > Run > Debug Configurations**:



- The debug configurations will open, choose the Segger interface and click on **Debug**. The same steps apply when using the P&E interface:



– The **Debug session** will start and the bootloader is now flashed on the board:



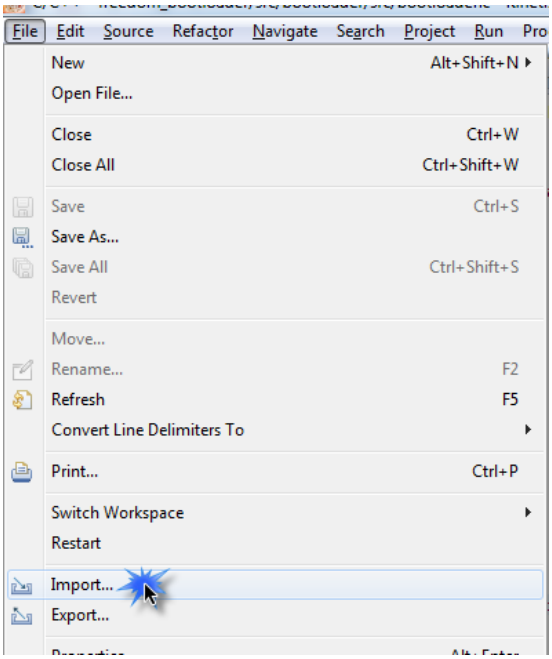
2.2 Loading demo application using the Kinetis Updater

For this section of the document the **Kinetis Updater** tool will be used to flash the application image to the MCU. The tool can be found on the following path:

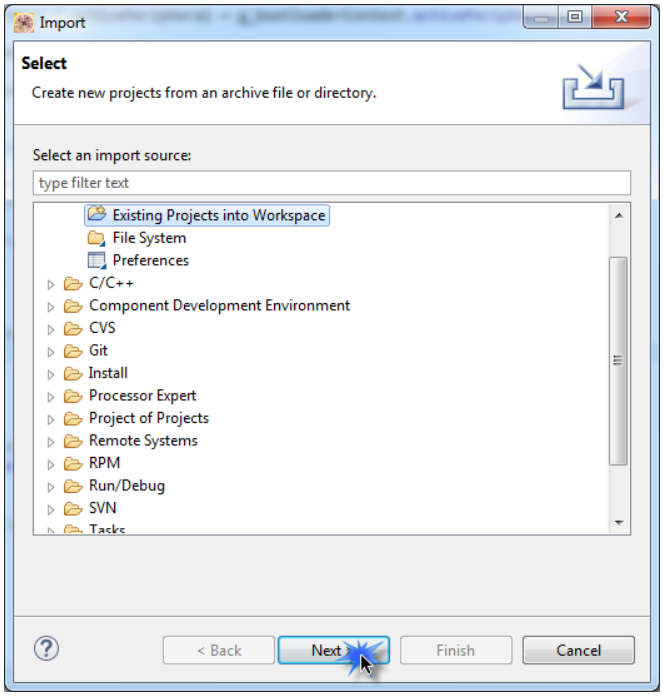
C:\Freescale\FSL_Kinetis_Bootloader_1_2_0\bin\win\KinetisUpdater

Before executing the Kinetis Updater tool we need to generate the **binary file** of the demo application that will be flashed to the MCU.

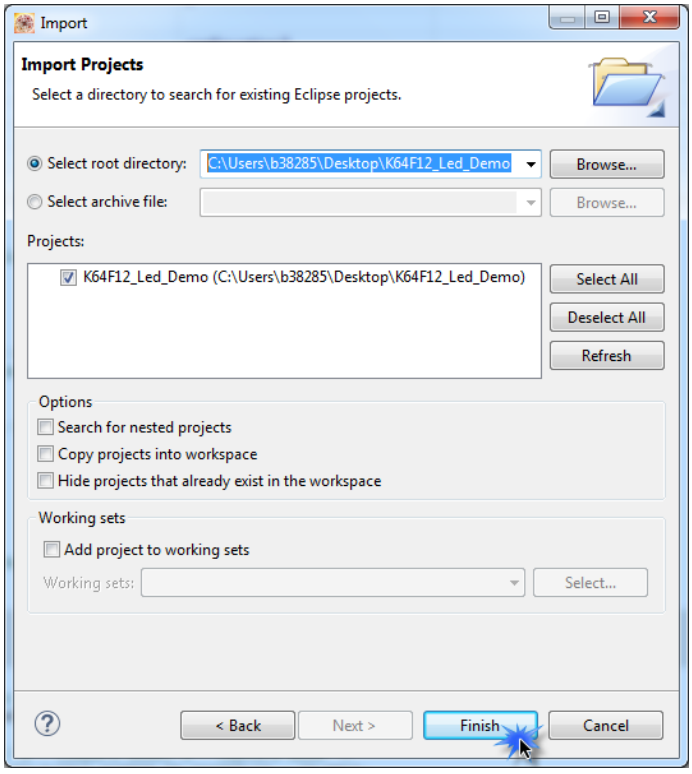
- Extract the **K64F12_Led_Demo** project and import it to KDS by clicking on **Menu > File > Import**:



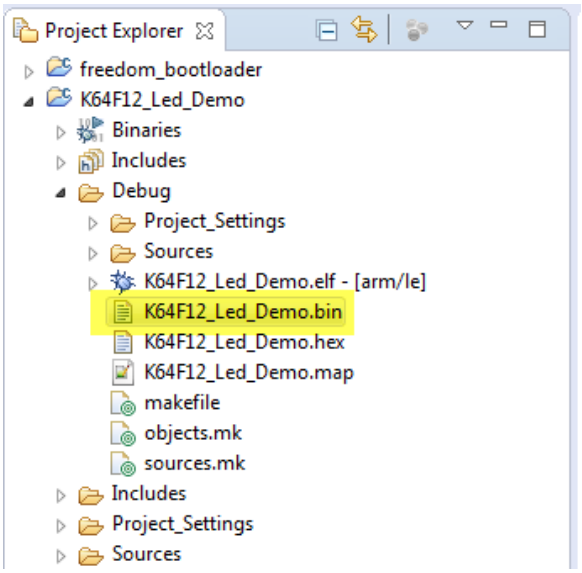
- Select **Existing Projects into Workspace** and click on **Next**:



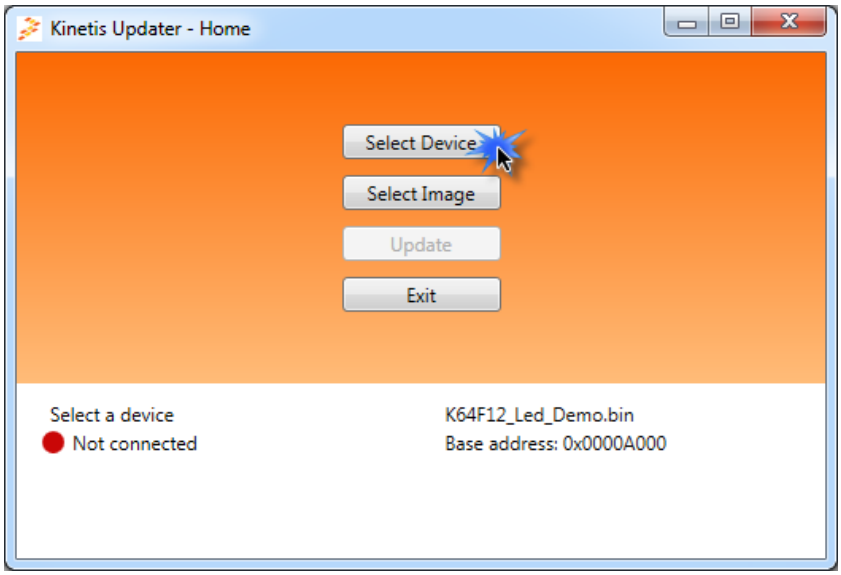
- Browse for the project and click on **Finish**:



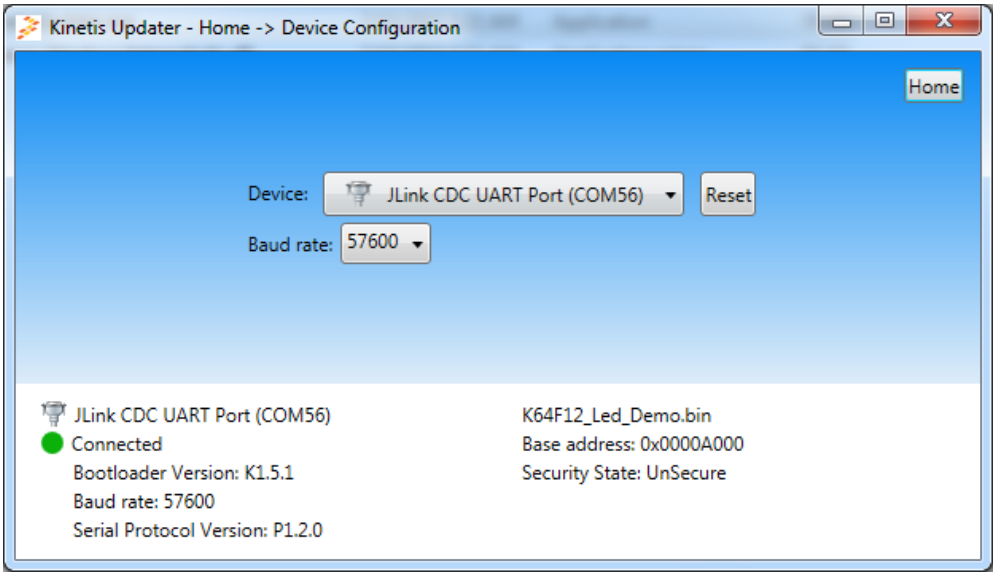
- Build the project and the binary file will be generated in the **Debug folder**:



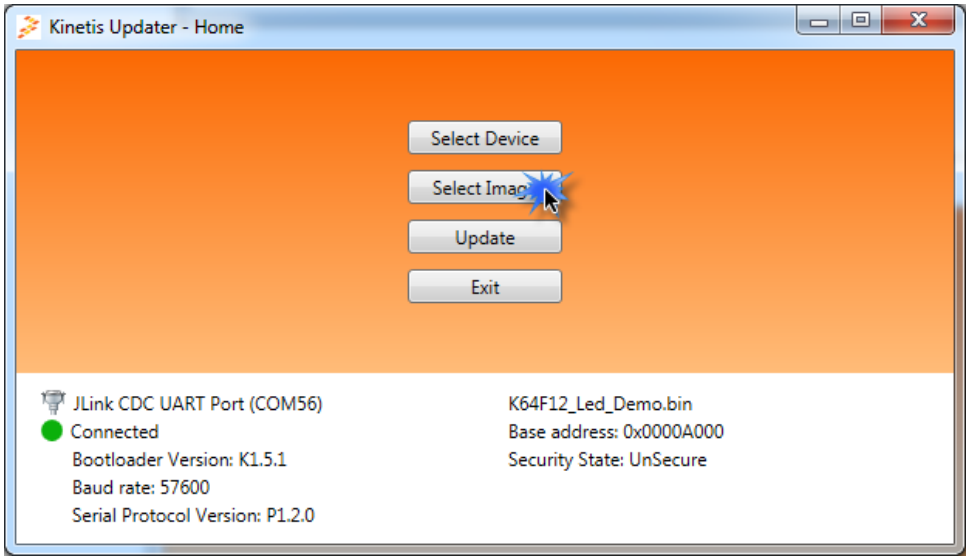
- Now open the Kinetis Updater tool and click on **Select Device**:



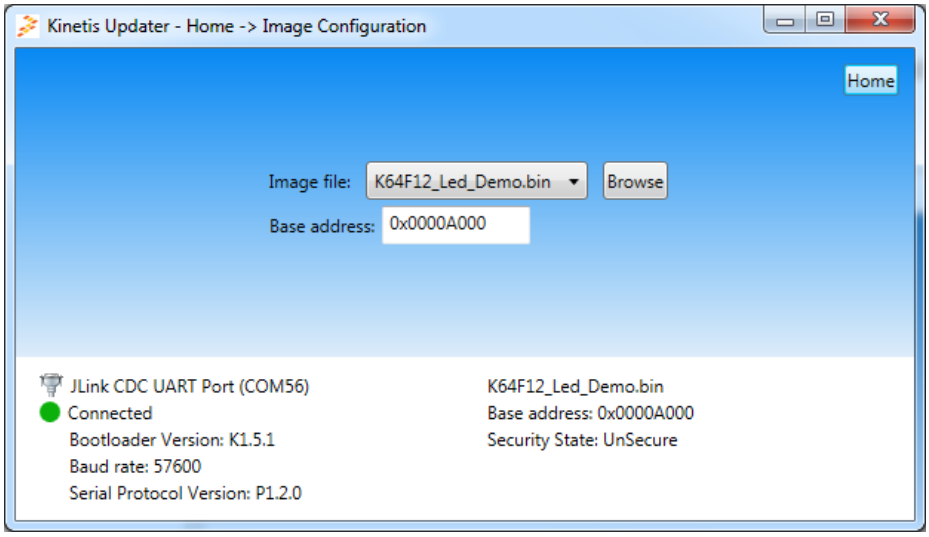
- Select your device and click on **Connect**, if the connection is not established perform a reset to the board:



- Click on **Home** and then on **Select Image**:

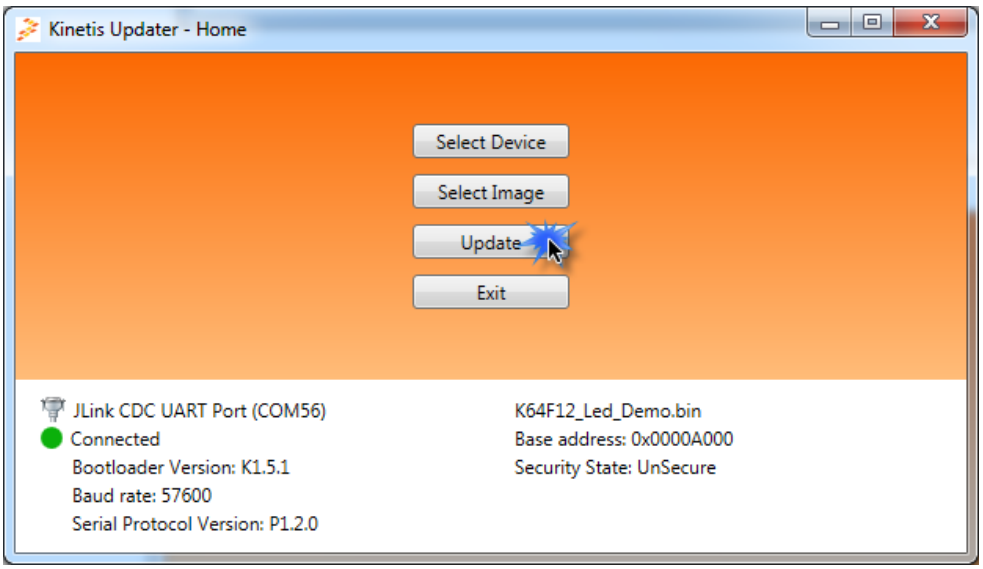


- Browse for the binary file that was generated by the demo project and update the Base address to 0x0000A000, this address is where the demo application starts:

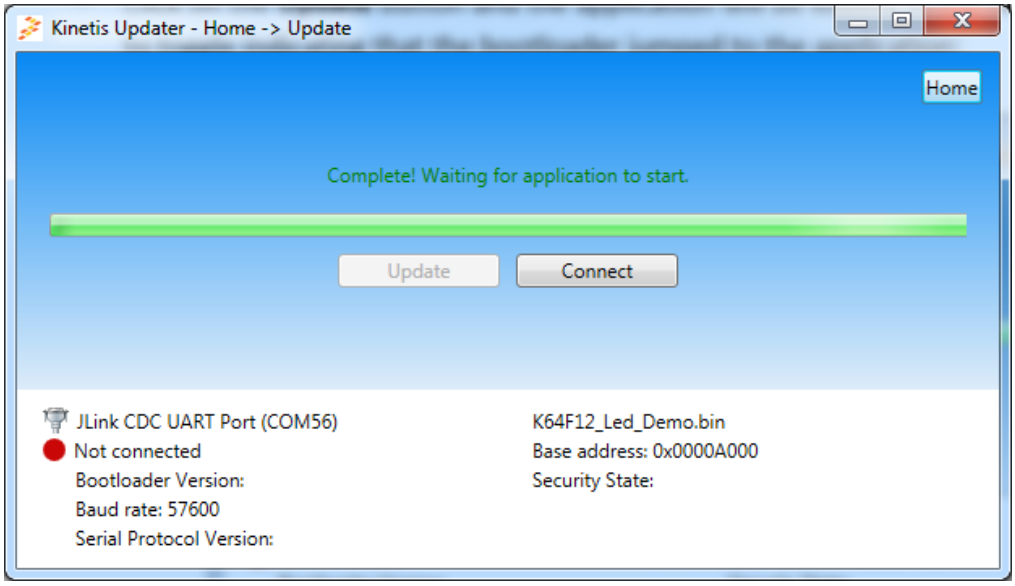



```
/* Specify the memory areas */
MEMORY
{
  m_interrupts          (RX)  : ORIGIN = 0x0000A000, LENGTH = 0x00000400
  m_text                (RX)  : ORIGIN = 0x0000A400, LENGTH = 0x00075C00
  m_data                (RW)  : ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
  m_data_2              (RW)  : ORIGIN = 0x20000000, LENGTH = 0x00030000
}
```

- Click on **Home** and then on **Update**:



- Click on the **Update** button and the application will be loaded to the MCU, the RGB led will start to toggle indicating that the bootloader jumped to the application:

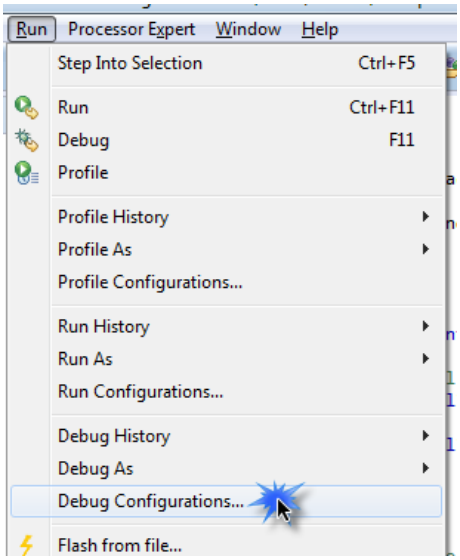


- At this point you can now proceed to the section 3.2 “Debugging bootloader and demo application projects using the Segger interface”.

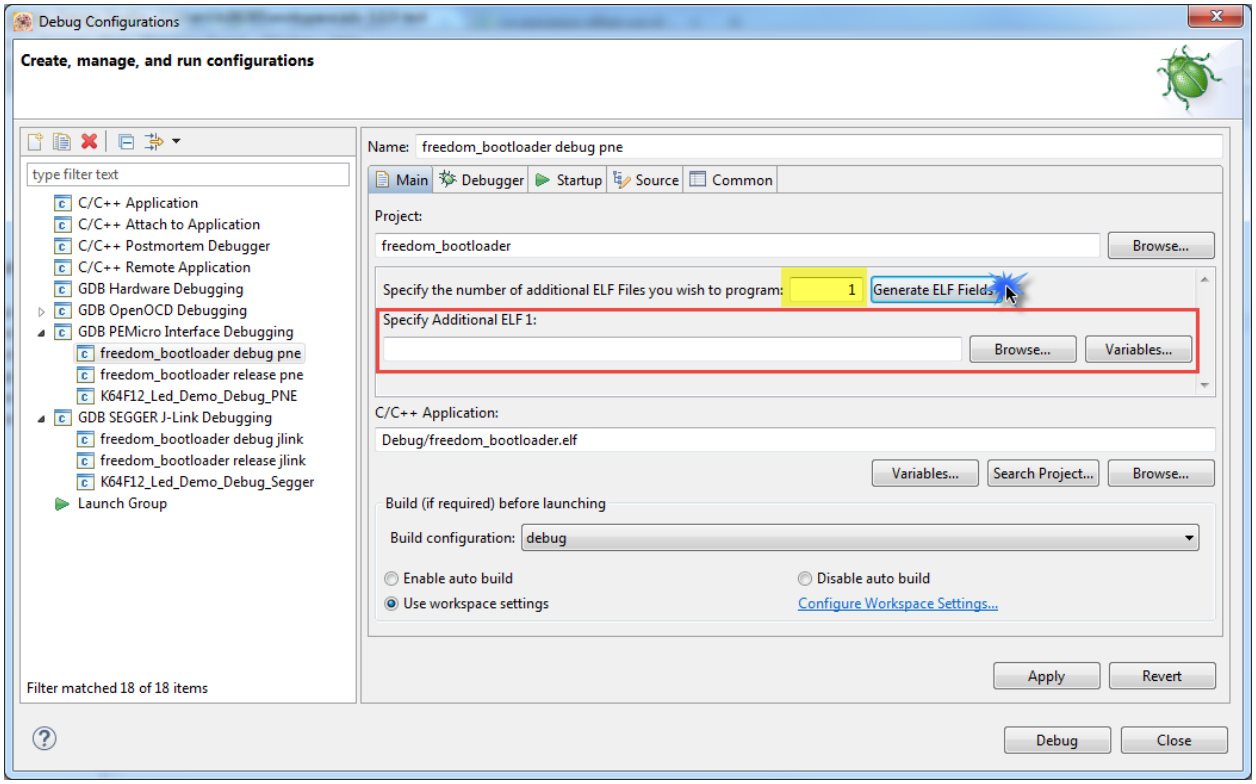
2.3 Flashing demo application and bootloader using the P&E advanced programming options

Refer to the sections 2.1 and 2.2 that explain how to import and build both the **freedom_bootloader** and **K64F12_Led_Demo** projects.

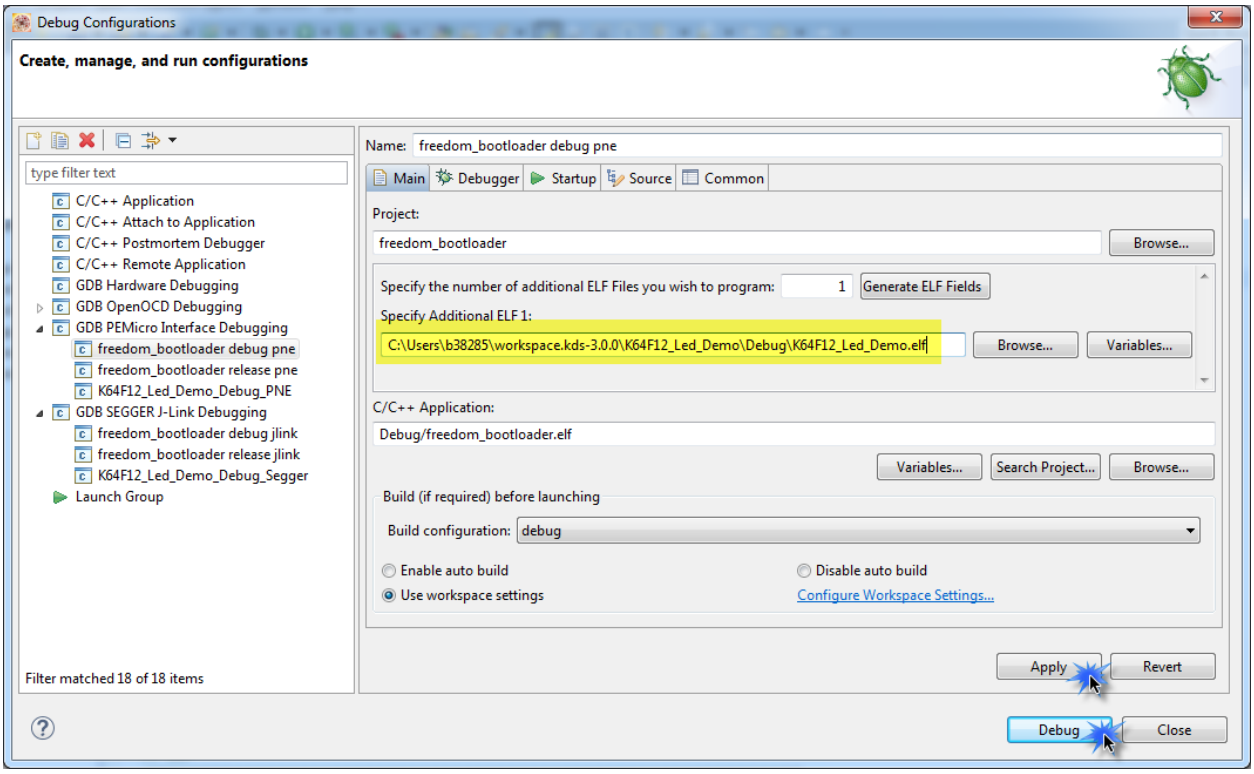
- Once that both projects are in you workspace and have been built click on **Menu > Run > Debug Configurations**:



- Choose the corresponding P&E connection for the bootloader project and generate **1 additional ELF field**:



- **Browse** for the *.elf file generated by the demo application, click on **Apply** then on **Debug**:

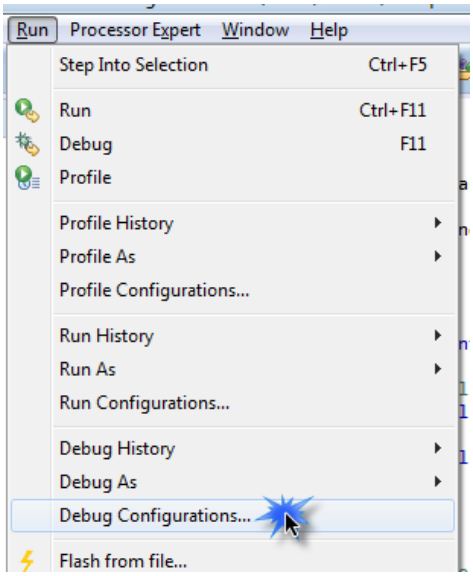


- The **Debug session** will start and both the bootloader and demo application are now flashed on the MCU, you can now proceed to the section 3.1 “Debugging bootloader and demo application projects using the P&E interface”.

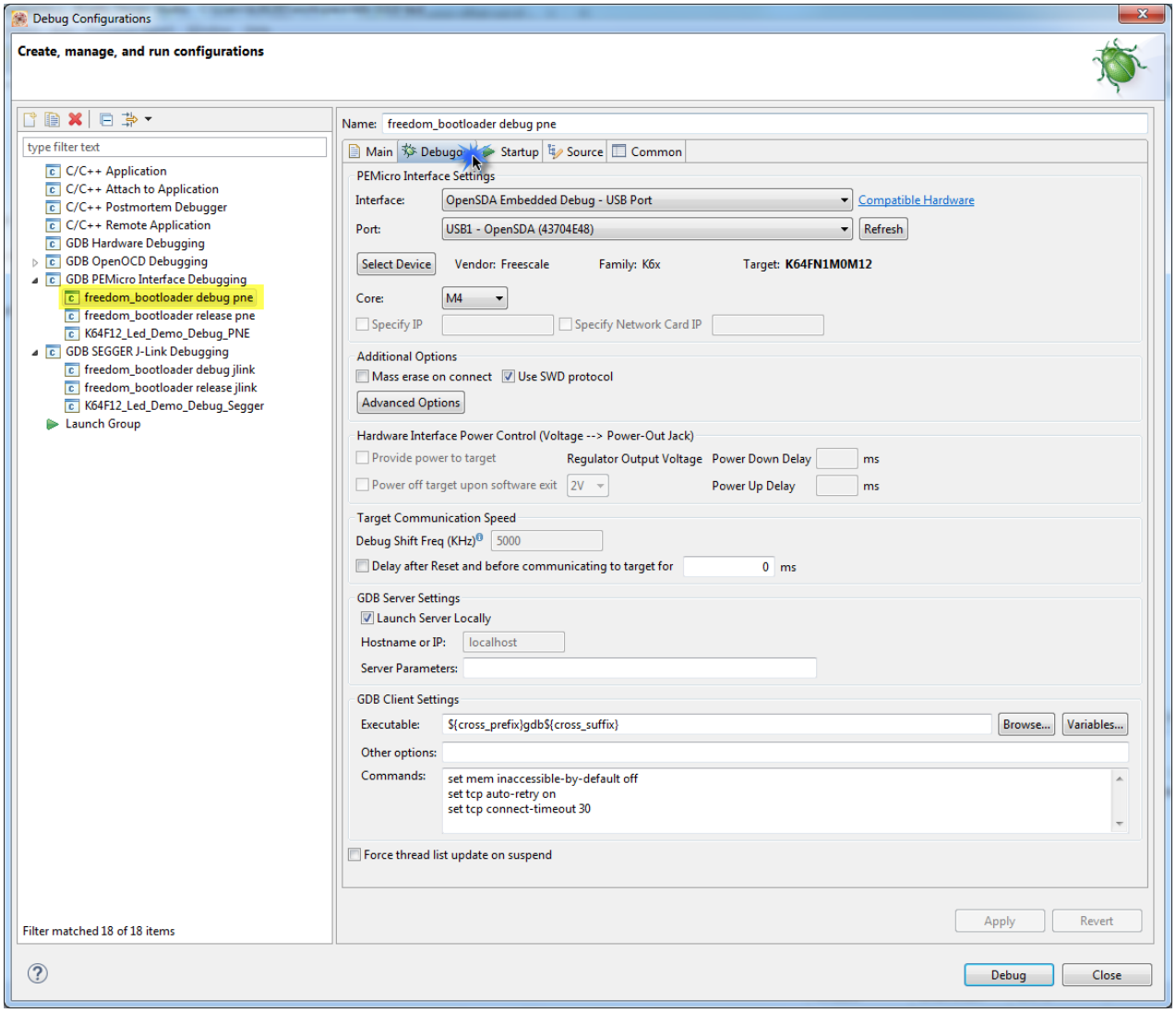
3. Debugging Bootloader and Application

3.1 Debugging bootloader and demo application projects using the P&E interface

- After the bootloader and the demo application have been flashed to the MCU using the P&E advanced programming options click on **Menu > Run > Debug Configurations**:



- Choose the corresponding P&E connection for the bootloader project and go to the **Debugger tab**:



- Now we need to specify an additional symbol file to be used in the debug session, this will be done by using the following GDB command:

  add-symbol-file filename address

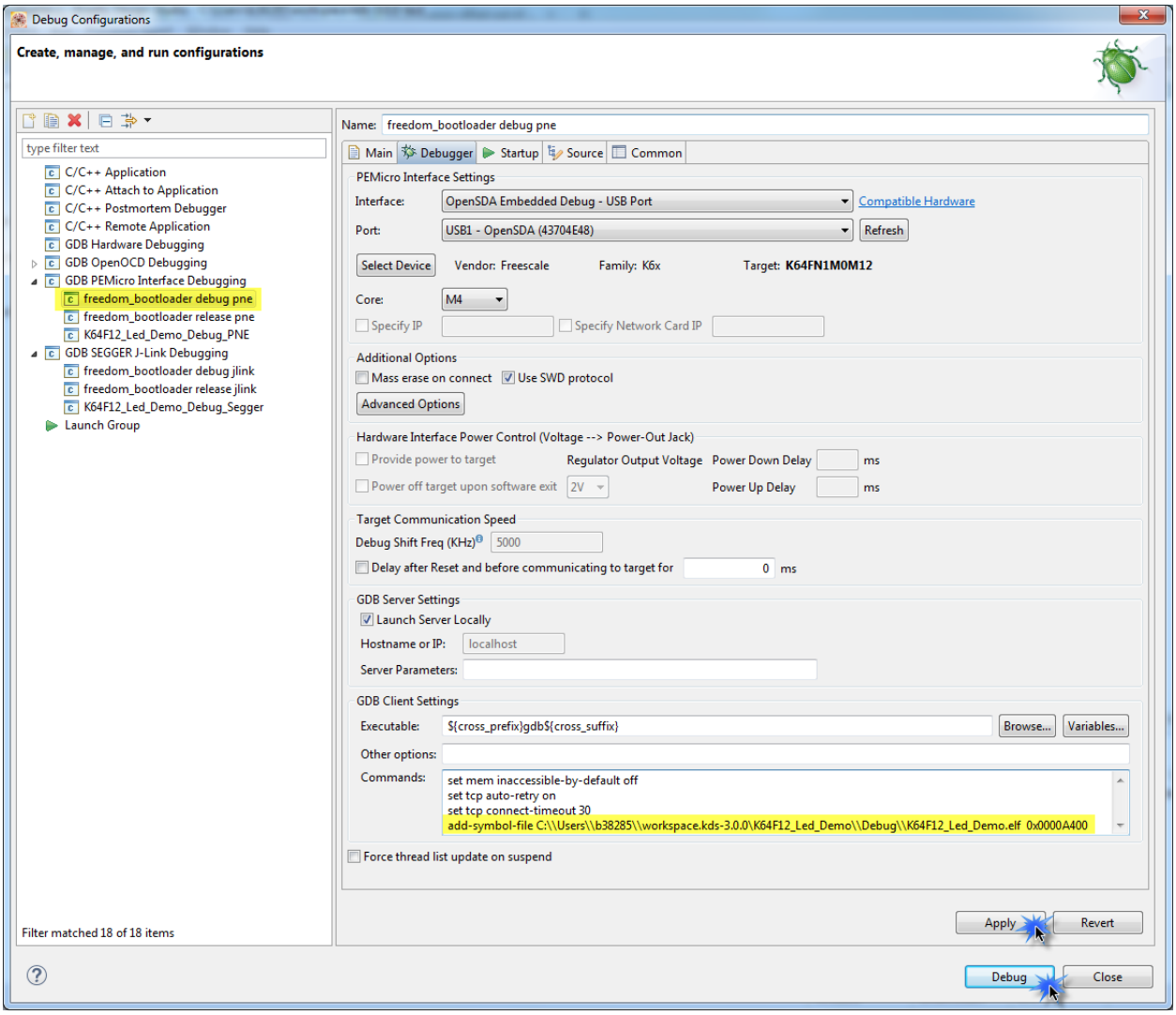
  Where *filename* is the path to the *.elf file generated by the demo application using double backslashes:

  C:\\Users\\b38285\\workspace.kds-3.0.0\\K64F12_Led_Demo\\Debug\\K64F12_Led_Demo.elf

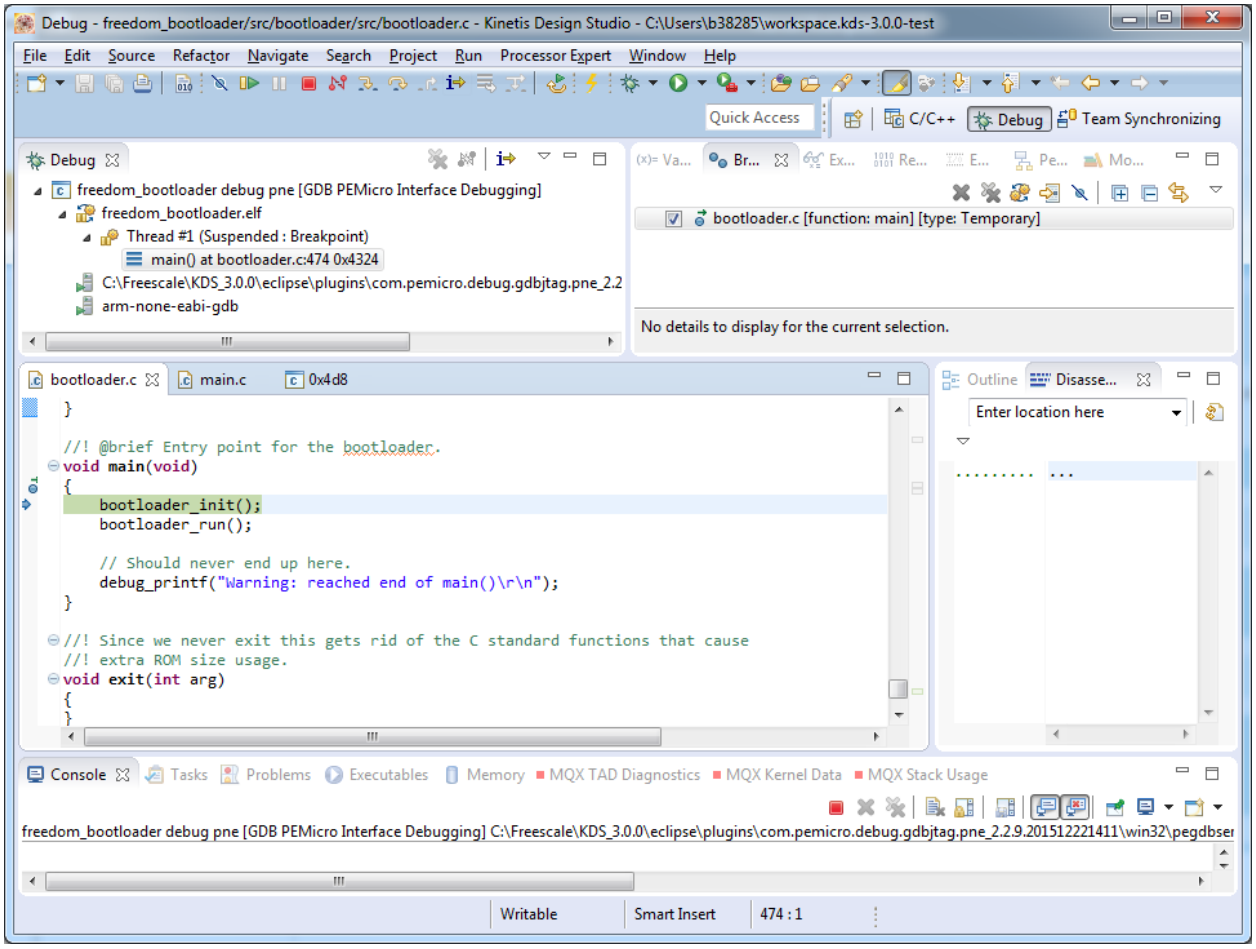
  And the *address* is the origin of the m_text section of my demo application, in this case *0x0000a410*:

```
/* Specify the memory areas */
MEMORY
{
  m_interrupts          (RX)  : ORIGIN = 0x0000A000, LENGTH = 0x00000400
  m_text                (RX)  : ORIGIN = 0x0000A400, LENGTH = 0x00075C00
  m_data                (RW)  : ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
  m_data_2              (RW)  : ORIGIN = 0x20000000, LENGTH = 0x00030000
}
```

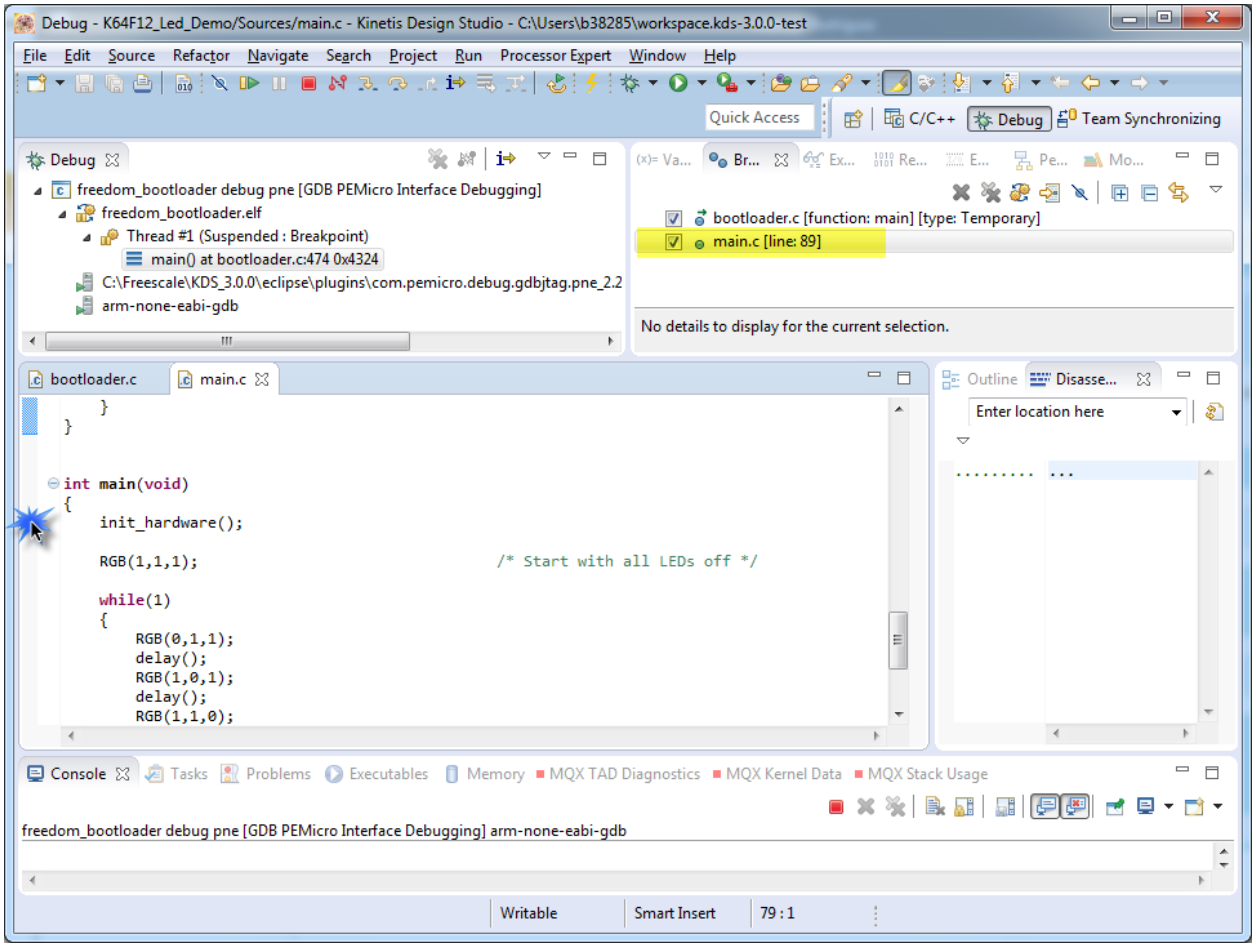
- Here are the updated debug configurations, after the GDB command is added, click on **Apply** then **Debug**:



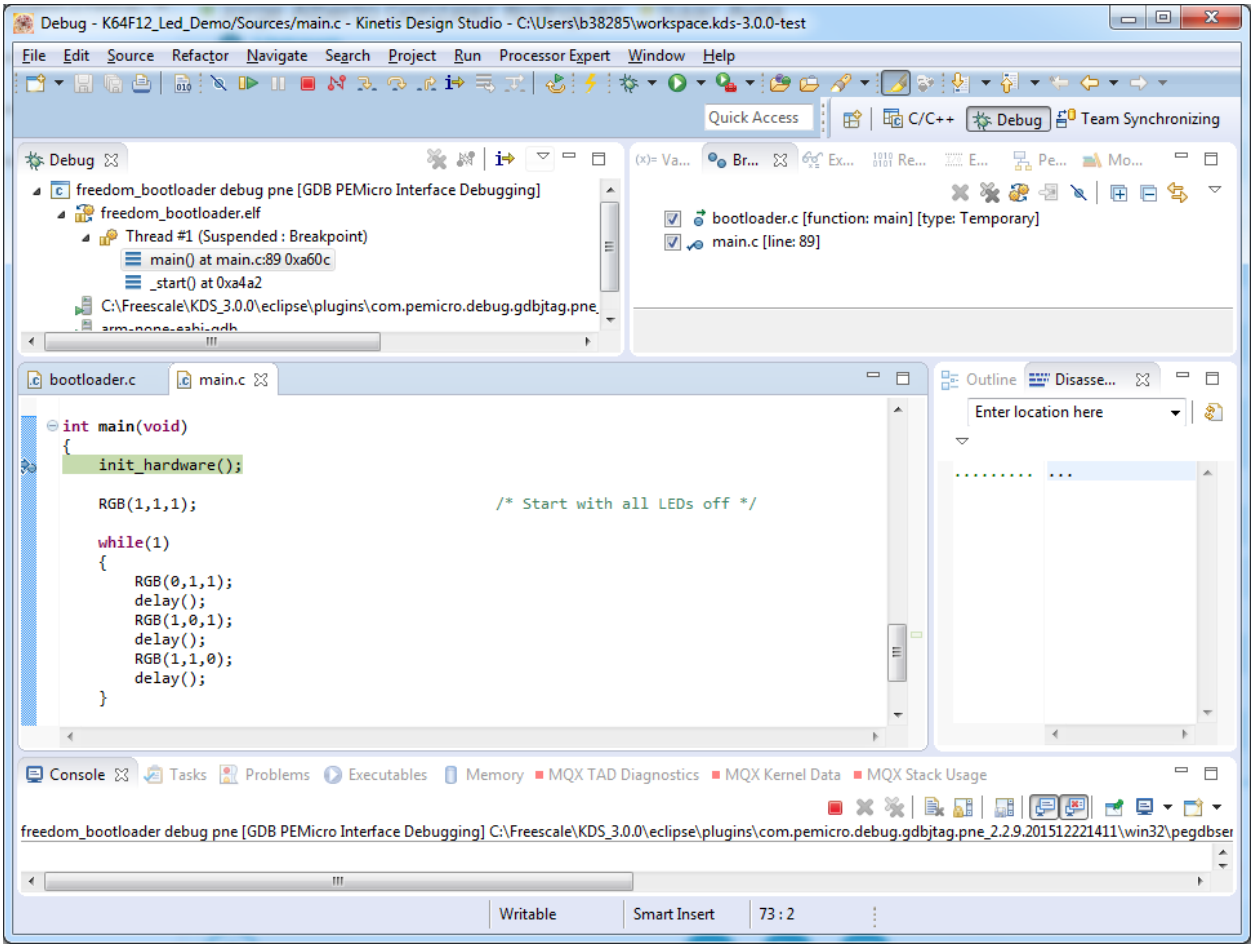
- The **Debug session** will start:



- Open the **main.c** file of the **K64F_Led_Demo** and set a breakpoint in the **init_hardware**() function:

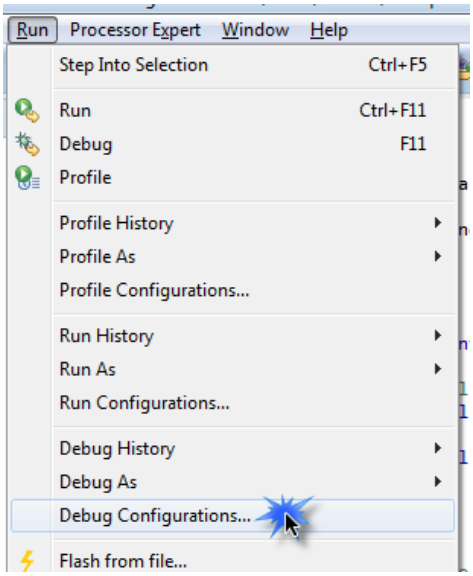


- At this point you can step through the bootloader program and then jump to the demo application and continue debugging it:

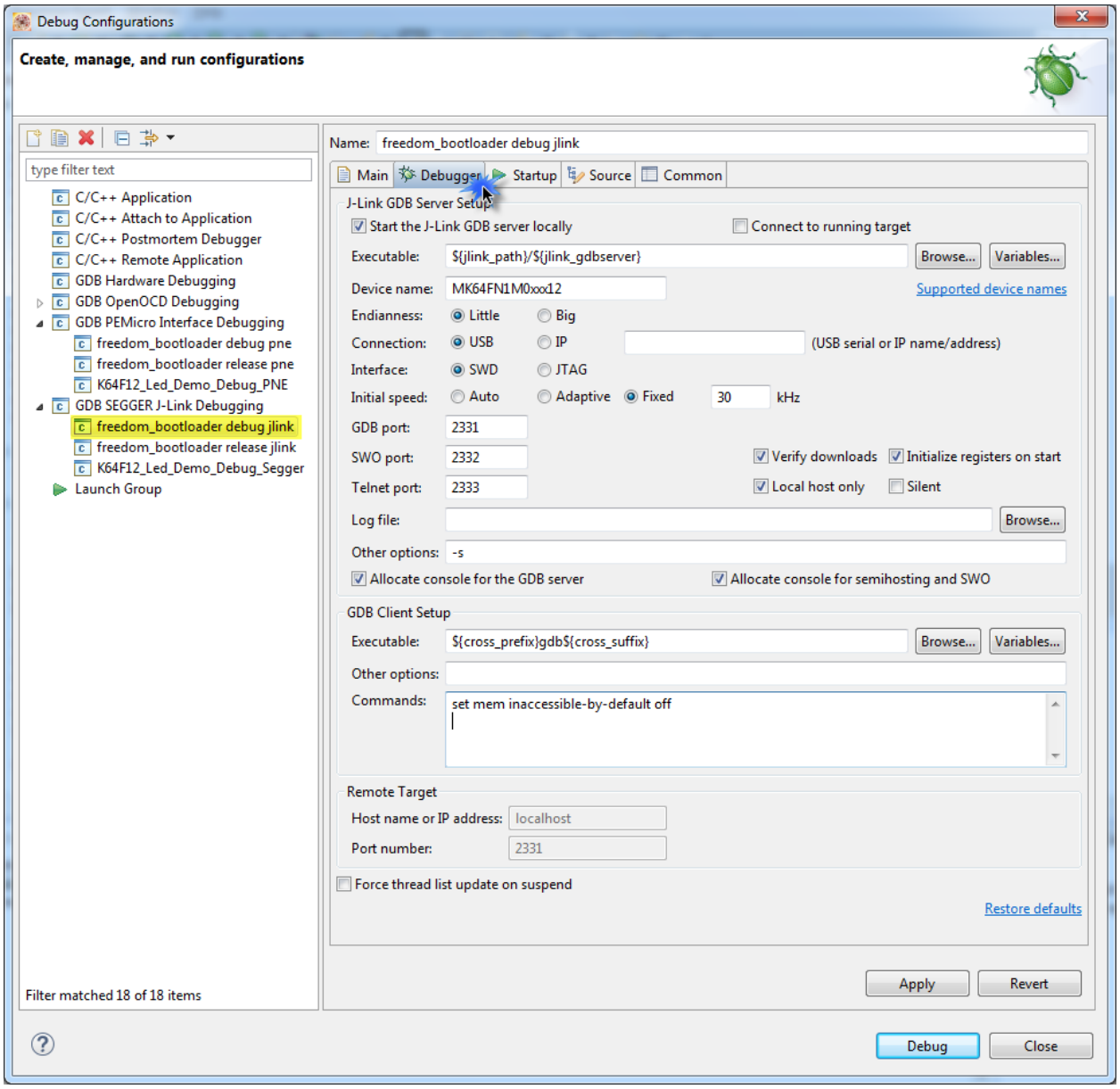


3.2 Debugging bootloader and demo application projects using the Segger interface

- After the bootloader has been flashed to the MCU and the demo application was loaded using the Kinetis Updater tool go to KDS, make sure both projects are opened and click on **Menu > Run > Debug Configurations**:



- Choose the corresponding Segger connection for the bootloader project and go to the **Debugger tab**:



- Now we need to specify an additional symbol file to be used in the debug session, this will be done by using the following GDB command:

  add-symbol-file filename address

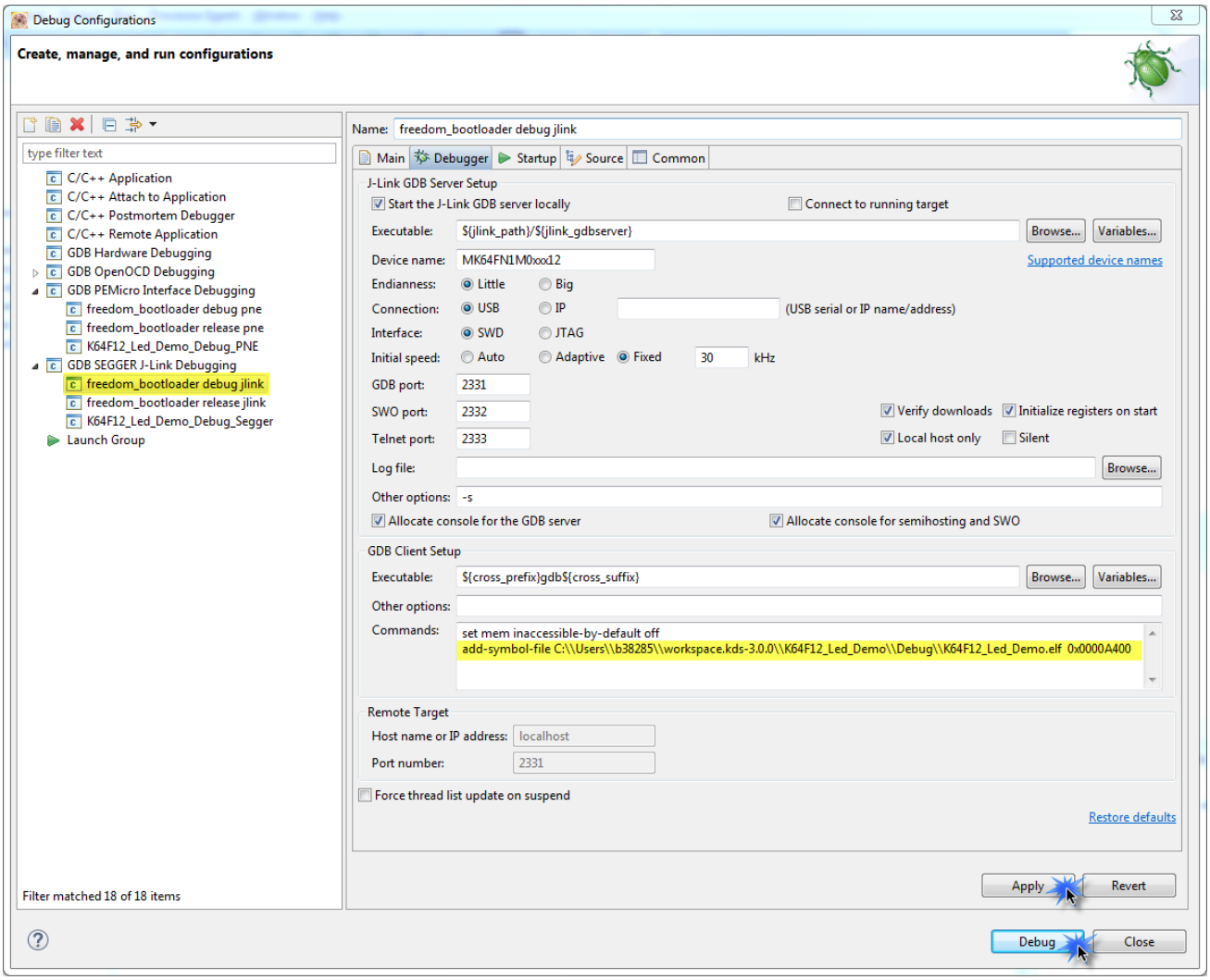
  Where *filename* is the path to the *.elf file generated by the demo application using double backslashes:

  C:\\Users\\b38285\\workspace.kds-3.0.0\\K64F12_Led_Demo\\Debug\\K64F12_Led_Demo.elf

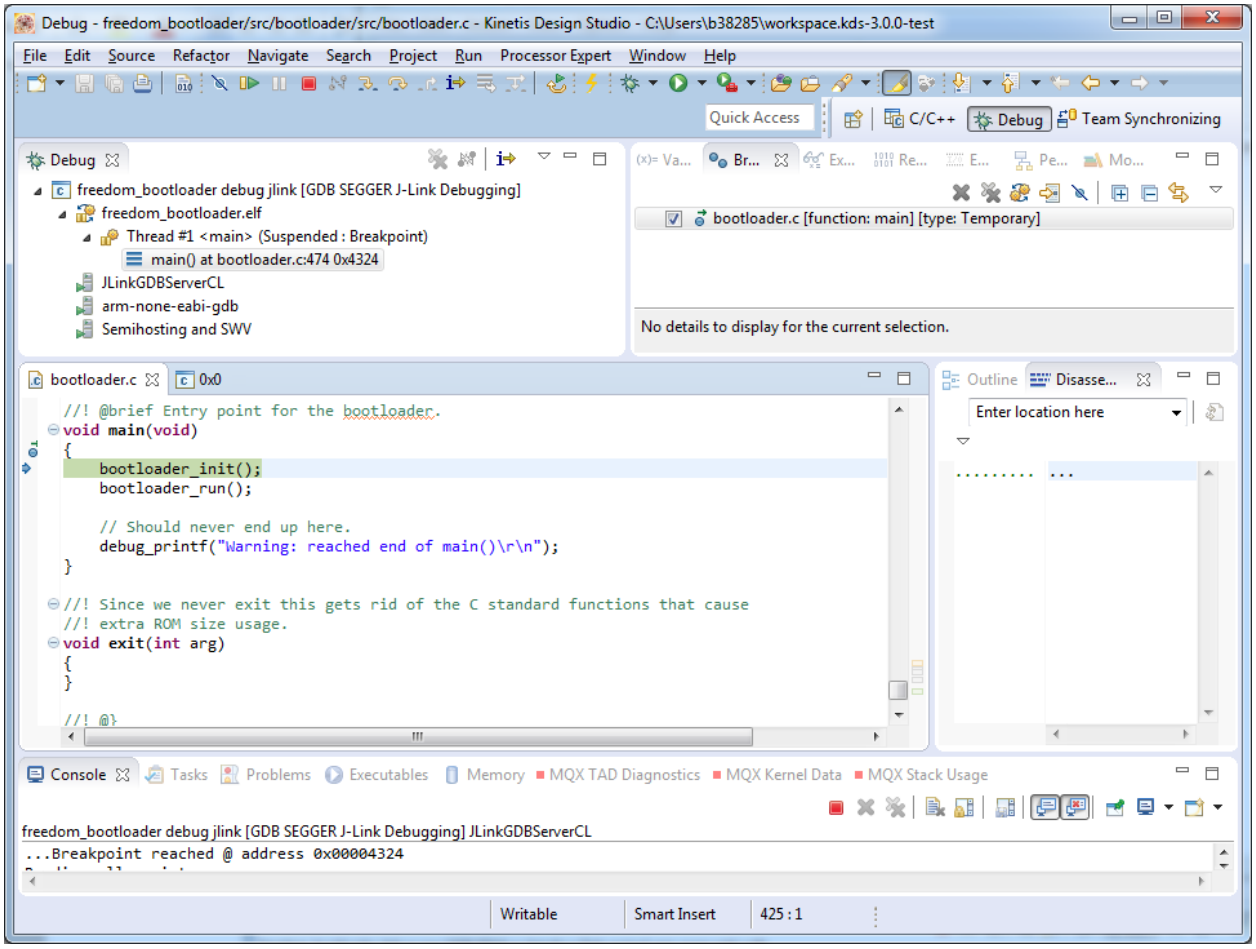
  And the *address* is the origin of the m_text section of my demo application, in this case *0x0000a410*:

```
/* Specify the memory areas */
MEMORY
{
  m_interrupts          (RX)  : ORIGIN = 0x0000A000, LENGTH = 0x00000400
  m_text                (RX)  : ORIGIN = 0x0000A400, LENGTH = 0x00075C00
  m_data                (RW)  : ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
  m_data_2              (RW)  : ORIGIN = 0x20000000, LENGTH = 0x00030000
}
```

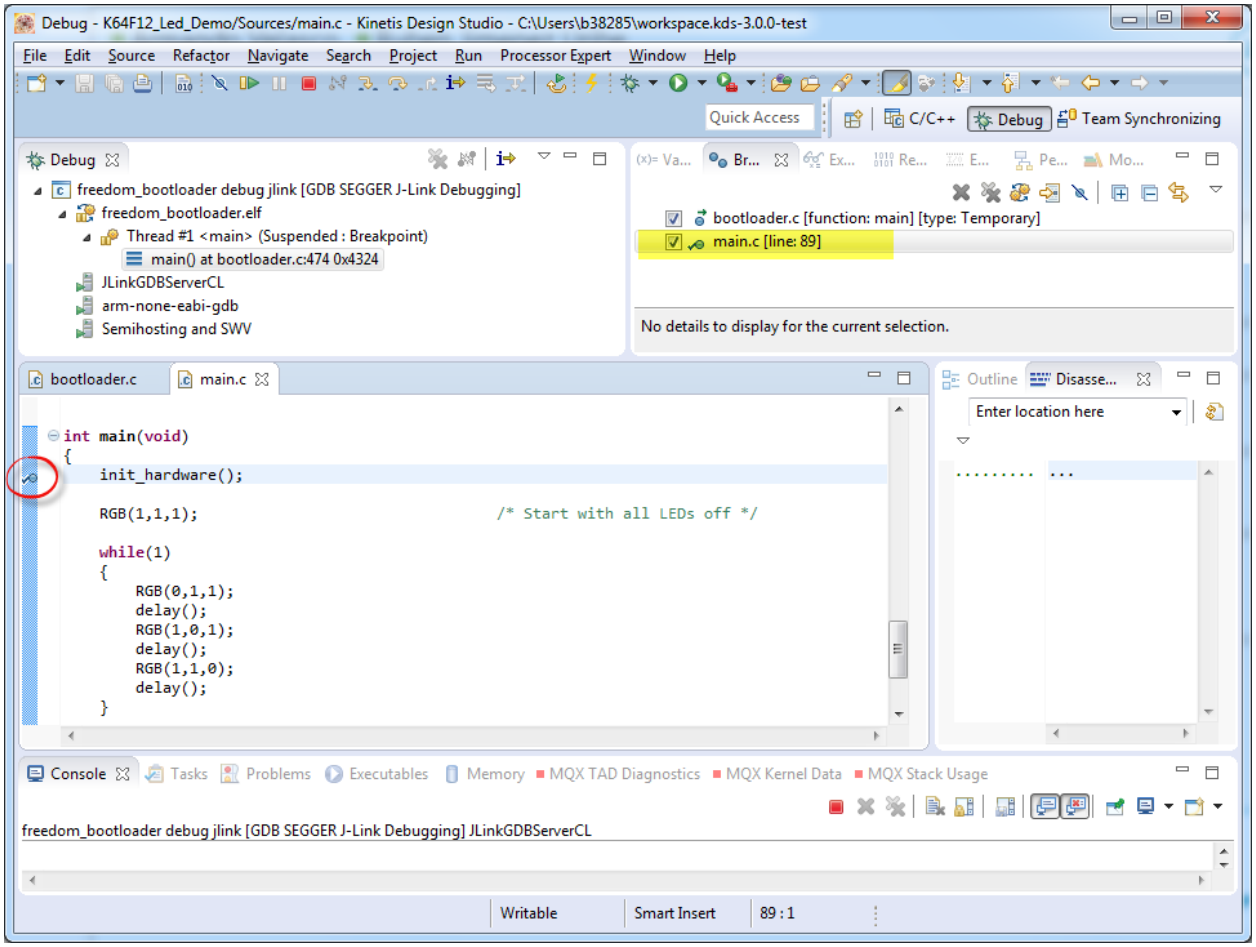
- Here are the updated debug configurations, after the GDB command is added click on **Apply** then **Debug**:



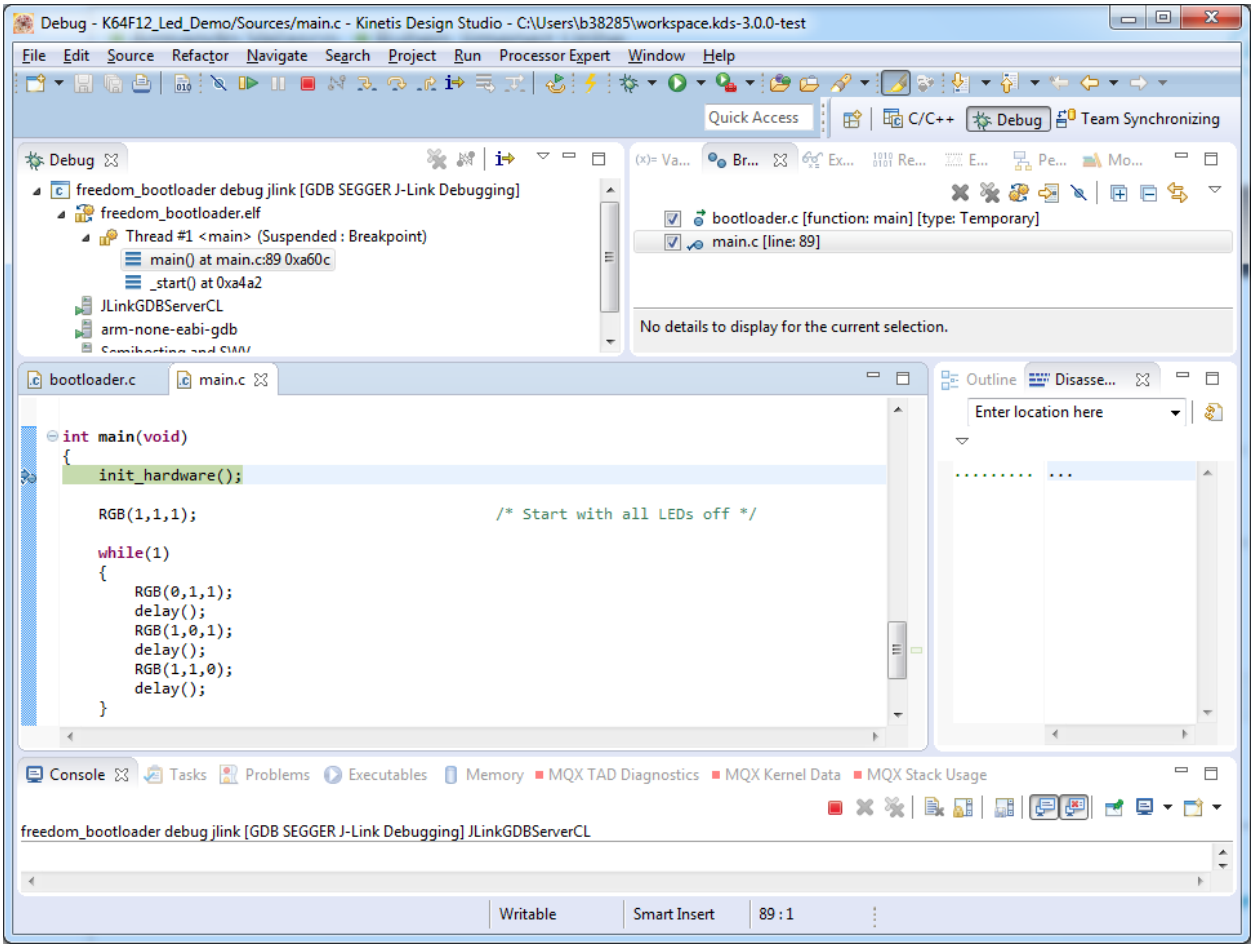
– The **Debug session** will start:



- Open the **main.c** file of the K64F_Led_Demo and set a breakpoint in the **init_hardware**() function:



- At this point you can step through the bootloader program and then jump to the demo application and continue debugging it:



4. Conclusion.

This document has demonstrated how to use Kinetis Design Studio to flash a bootloader and an application to an MCU and how to debug the bootloader and application at the same time using the GDB command *add-symbol-file filename address* that specifies an additional symbol file to be used in the debug session.

Appendix A - References

- KDS webpage:
  www.nxp.com/kds

- KBOOT webpage:
  www.nxp.com/kboot

- Debugging with gdb:
  https://sourceware.org/gdb/current/onlinedocs/gdb/

- Adapting KDS project for KBOOT flash resident bootloader:
  https://community.freescale.com/docs/DOC-256669

- MCU on Eclipse - Debugger (GDB Server with P&E and Segger):
  http://mcuoneclipse.com/2013/07/22/diy-free-toolchain-for-kinetis-part-3-debugger-gdb-server-with-pe-and-segger/