Author: MCUXpresso IDE Team

Version: 1.2.0 - 15th September 2021

# Using the MIMXRT1170-EVK with MCUXpresso IDE v11.4.x

## Table of Contents

Revision History

| Date | Description |
|---|---|
| October 9th, 2020 | V1.0.0 – Initial Revision |
| February 26th, 2021 | V1.1.0 – Updates based on latest IDE & SDK releases |
| September 15th, 2021 | V1.2.0 – Updates based on IDE 11.4.x & SDK 2.10 releases. Added SWO Trace section |

## 1. Overview

This document is intended to assist users who are new to using the MIMXRT1170-EVK board. It assumes some familiarity with creating projects and debugging application with **MCUXpresso IDE v11.4.x**.

It is also assumes an SDK for the MIMXRT1170-EVK board has been installed into MCUXpresso IDE, and a LinkServer/CMSIS-DAP debug connection will be used for all debug operations.

This document applies to **MIMXRT1170-EVK REV C** or later boards which are fitted with MCU **i.MX RT1176 B0** silicon. The RT1170 (family) name is used interchangeably to refer to the MCU.

*Note: it is strongly recommended that the latest available SDK is installed – at the time of writing this is the MIMXRT1170-EVK version 2.10.0.*

For more information on using MCUXpresso IDE, please see the MCUXpresso IDE User Guide.

*Note: this is a guide only and not intended as a definitive document*

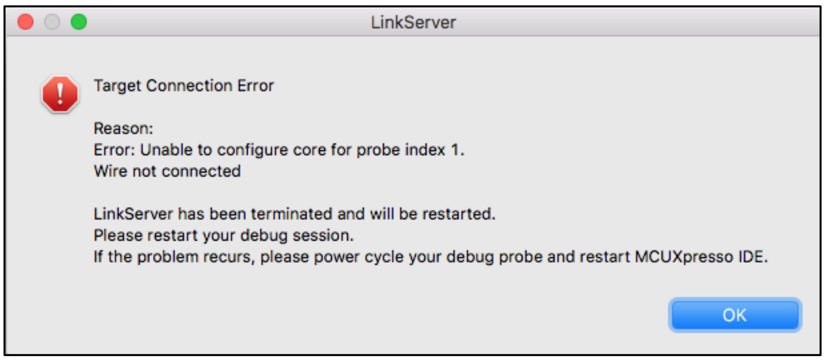
## 2. Read Me First

The MIMXRT1170-EVK board has a number of subtle design differences to the similar MIMXRT1060-EVK board, please refer to the section Overview of Board features related to Debug for details of board features.

The i.MX RT1170 MCU features a high-performance Arm® Cortex®-M7 core and a power-efficient Cortex®-M4 core.

The i.MX RT1170 uses the primary Cortex-M7 core for booting, while having the secondary Cortex-M4 core invisible to the debug world. For LinkServer debug connections, a connect script file is automatically selected to wake-up the M4 core such that it is visible for subsequent debug operations. Additionally, a reset script file is automatically selected to handle booting and debug visibility after a reset operation. Please refer to the section Debug Launch Configurations for more information.

Certain images when programmed into flash can prevent debug control on this part, leading to error messages of the form below when debug operations are attempted:



Target Connection Error

Reason:
Error: Unable to configure core for probe index 1.
Wire not connected

LinkServer has been terminated and will be restarted.
Please restart your debug session.
If the problem recurs, please power cycle your debug probe and restart MCUXpresso IDE.

For example, this can occur if an image is programmed onto flash that located the stack outside of accessible RAM, placed the device in a deep sleep state, or otherwise caused an exceptional condition which interferes with debug access.

To erase the flash (and potentially recover debug control), please follow the procedure described in the Troubleshooting section at the end of this document before any other debug operations are attempted.

Please also refer to the section DAPLink Firmware version to ensure your board is programmed with the correct debug probe firmware.

### MIMXRT1170-EVK Information

One key improvement is the use of new onboard debug probe hardware. This debug probe (sometimes referred to as “FreeLink”) is based on an LPC4322 MCU and is very similar to an LPC-Link2. As supplied it is pre-programmed with a version of DAPLink (CMSIS-DAP) firmware, however fitting a link J22 will allow it to operate as an LPC-Link2 debug probe and in this mode will deliver superior performance and features. Please see Onboard Debug Probe for more details.

At the time of writing there is only one variant of this board which ships fitted with an external 16MB QSPI Flash connected by default through FlexSPI1.

*Note: Even though there are additional flashes fitted on the EVK board, they cannot be used without board-level modifications, therefore the rest of this document as well as SDK example projects reference only the QSPI flash.*

For further information on the i.MX-RT1170 and the EVK development board, please see the following links:

https://www.nxp.com/pages/:i.MX-RT1170

https://www.nxp.com/pages/:MIMXRT1170-EVK

The document "MIMXRT1170 EVK Board Hardware User's Guide" may also prove useful https://www.nxp.com/webapp/Download?colCode=MIMXRT1170EVKHUG

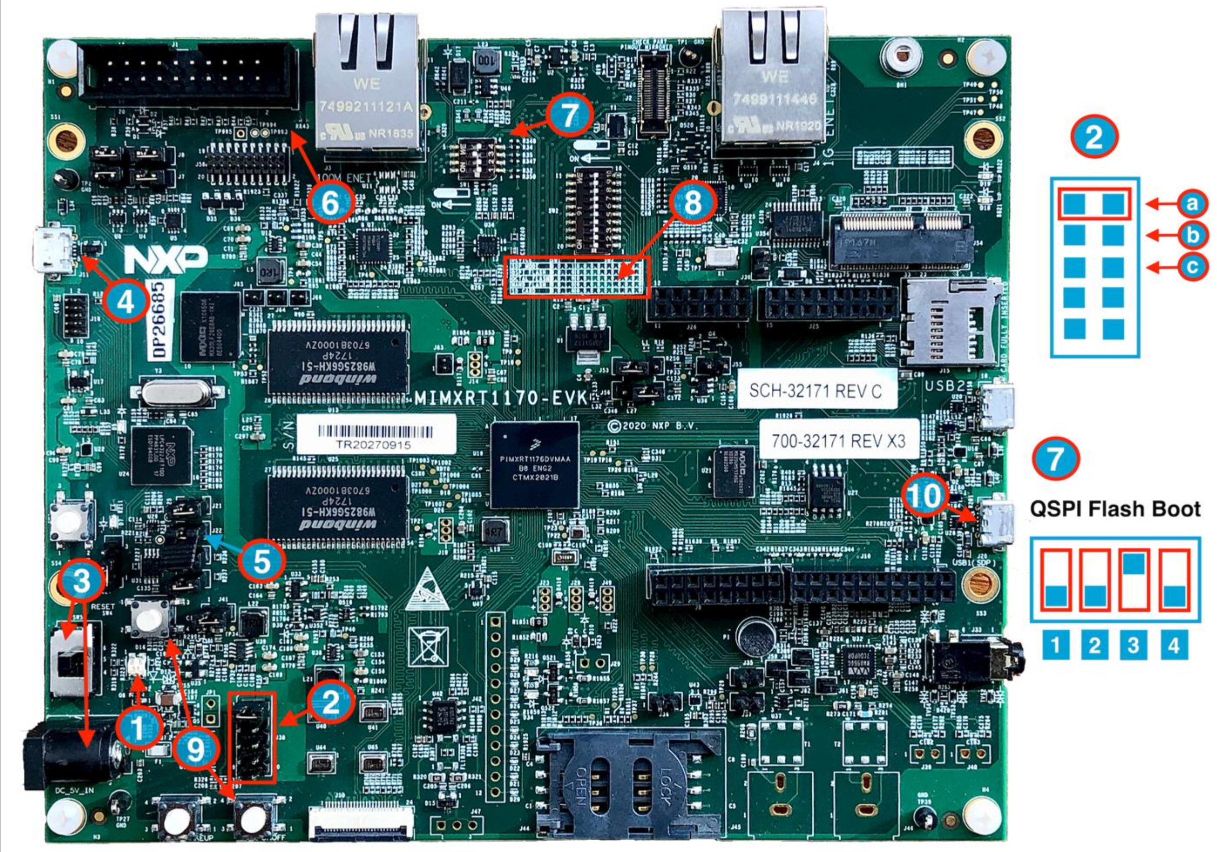
## Debug Restrictions

Despite the extensive feature set of this MCU/Board, some debug capabilities are not available.

ETB based instruction trace is not supported by this MCU.

## 3. Overview of Board features related to Debug

Below is a photograph of the MIMXRT1170-EVK board with key elements numbered and described:



*MIMXRT1170-EVK*

1. MCU Power LED (D16). **Note: if this LED is not lit, the MCU/board is not powered and cannot be debugged (see 2 and 9).**
2. Jumper block J38 – Controls MCU power source (see break out section link options)
    a. External power connection (see 3)
    b. Target USB1 provides power (see 10)
    c. Power via the USB debug connection (only available when using DAPLink OpenSDA firmware) (see 4, 5)
3. External power connection. If used, this should be 5V, centre +ve, and J38 set to position a (see 2a). This allows power to the board to be controlled via switch SW5 (above the barrel connector J43)
4. J11 – Onboard debug probe USB connection. This debug probe is pre-programmed with DAPLink OpenSDA firmware and in this mode the USB connection can also power the board (see 2c and 5)
5. J22 - When a link is fitted (as shown), the onboard debug probe will operate in LPC-Link2 compatibility mode. In this mode, MCUXpresso IDE will softload LPC-Link2 firmware automatically when a debug operation is performed. In this mode, performance will be considerably enhanced and SWO debug trace will become available. However, in this mode the board must be powered externally (see 2a, b)

6. J1 – 20-way JTAG style connection for use with external debug probes such as the standalone LPC-Link2
7. SW1 - DIP switches to select boot options (see break out section for details). Note: at the time of writing, the MIMXRT1170-EVK boards ship with QSPI flash
8. Board printed table showing boot switch settings
9. Reset switches, SW4 (above) and SW6 (below, right). SW4 is the power-on reset button. SW6 is the board ON/OFF button. A short pressing in OFF mode causes the internal power management state machine to change state to ON. In ON mode, a short pressing generates an interrupt as a software-controllable power-down (this can recover a WFI 'freeze'). An approximate 5 seconds or more pressing causes a forced OFF – but the Power LED can remain lit!
10. Target USB1 – can also be used to provide power to the target – (see 2b)

## 4. On board Debug Probe

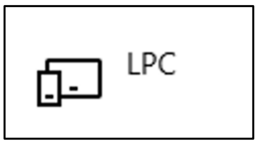
The onboard debug probe fitted to this board features enhanced hardware based on the LPC4322 MCU (as also used on the LPC-Link2 debug probe). As shipped, this debug probe runs DAPLink/CMSIS-DAP firmware in internal flash. This firmware features a mass storage device bootloader allowing programming via drag and drop to its filer window.

However, in this mode, debug performance is limited and SWO trace features are not available. Typical flash programming speed for small application – DAPLink firmware:

```
Flash Program Summary: 30764 bytes in 1.39 seconds (21.61 KB/sec)
```

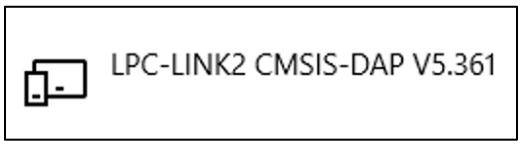
### LPC-Link2 mode

Fitting a jumper to link J22 and power cycling the board (and probe) will allow the debug probe to re-enumerate as a LPC device - visible in Windows Bluetooth and Devices as:


LPC

In this LPC-Link2 compatibility mode, MCUXpresso IDE will automatically recognise the device and softload LPC-Link2 firmware at the start of the next debug session.

At this point, the probe will re-enumerate and be visible in Windows as below:


LPC-LINK2 CMSIS-DAP V5.361

In addition to improved debug performance (which includes approximately 4x speed improvement for flash programming), SWO trace features are also available.

Typical flash programming speed for a small application – LPC-Link2 firmware:

```
Flash Program Summary: 30764 bytes in 0.39 seconds (76.84 KB/sec)
```

*Note: both versions of probe firmware are able to provide debug control and simultaneous serial communications via VCOM. Although there is no mass storage capability in this mode, binary file (or .axf) programming via the IDE's GUI Flash Tool will deliver much faster performance.*

### Permanently installing LPC-Link2 Firmware – LPCScrypt

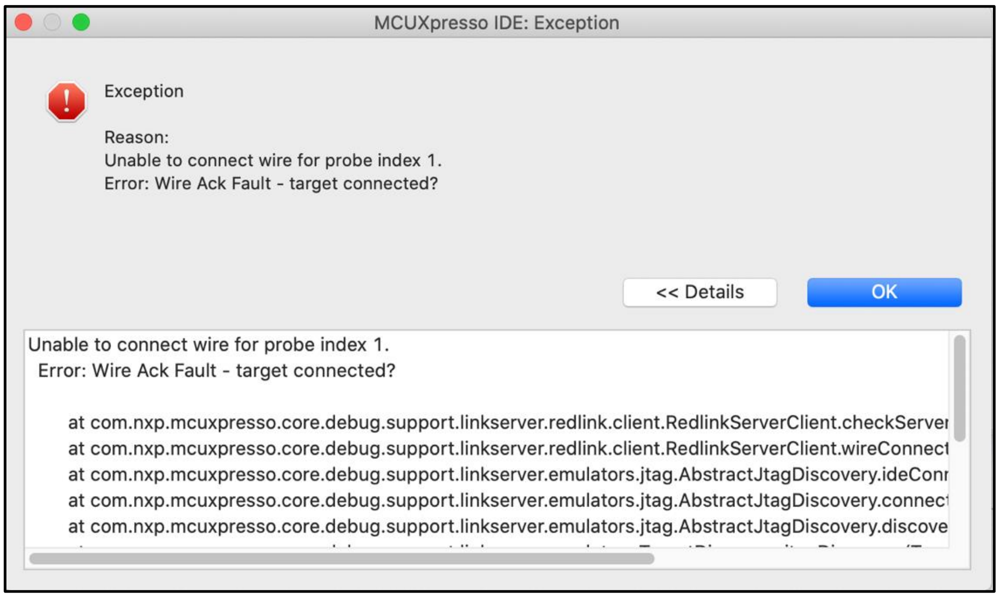
DAPLink firmware is factory programmed into the internal flash of the LPC4322 debug probe MCU. However, if desired, this can be permanently replaced using the LPCScrypt utility: https://www.nxp.com/LPCSCRYPT

If this is done, LPC-Link2 firmware will permanently replace DAP-Link when link J22 is not fitted.

## Debug Connection

For initial use, we recommend using the onboard debug probe via USB connection in DAPLink mode. See previous Overview of Board features… section above, option 2.c. **Before attempting a debug operation ensure that the Green LED D16 is lit – see 1 above.**

*Note: if the board has been configured correctly the LED next to SW5 switch will light green. If the LED is not lit, the MCU will not be powered and debug will fail, resulting in an error similar to that below (despite the DAPLink debug connection being available):*



```
Exception

Reason:
Unable to connect wire for probe index 1.
Error: Wire Ack Fault - target connected?
```

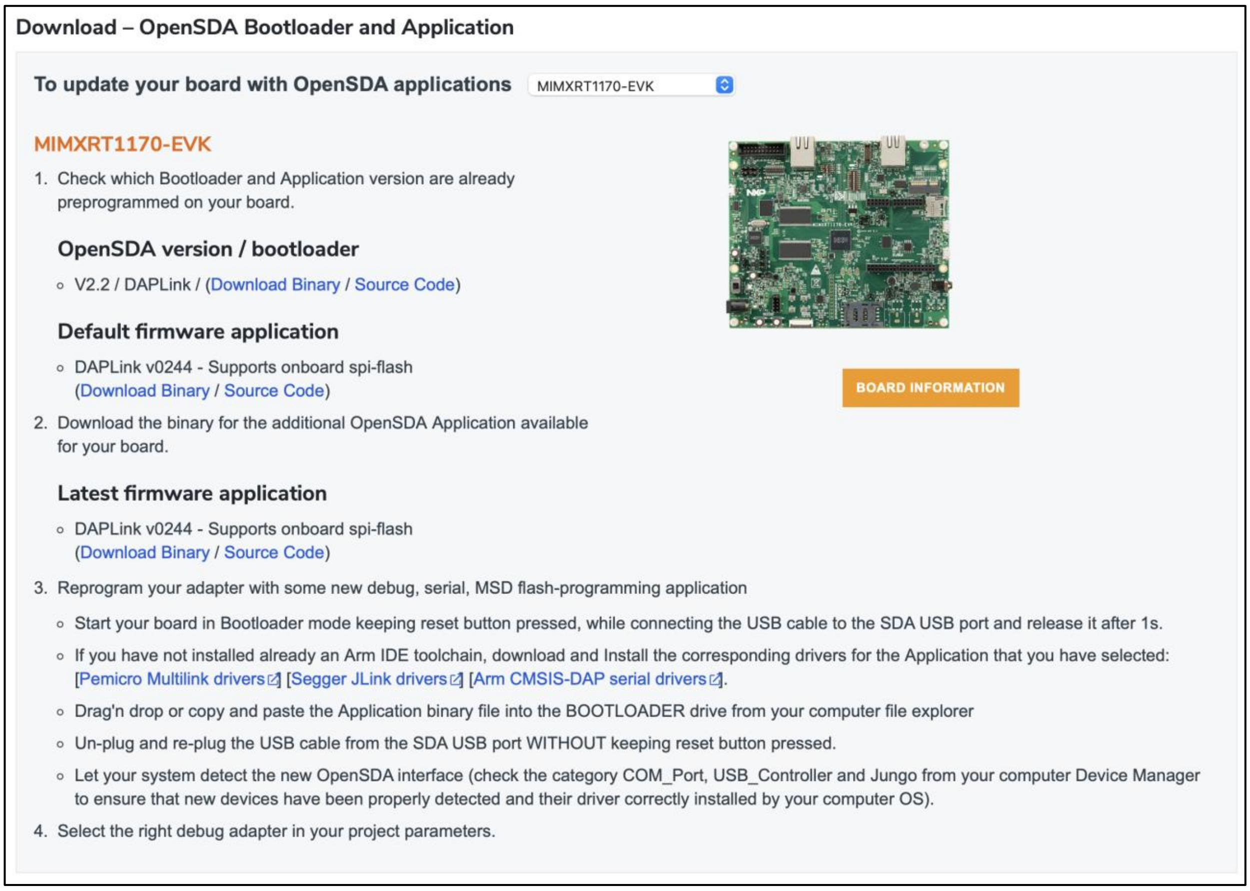
```
Unable to connect wire for probe index 1.
 Error: Wire Ack Fault - target connected?

    at com.nxp.mcuxpresso.core.debug.support.linkserver.redlink.client.RedlinkServerClient.checkServer
    at com.nxp.mcuxpresso.core.debug.support.linkserver.redlink.client.RedlinkServerClient.wireConnect
    at com.nxp.mcuxpresso.core.debug.support.linkserver.emulators.jtag.AbstractJtagDiscovery.ideConn
    at com.nxp.mcuxpresso.core.debug.support.linkserver.emulators.jtag.AbstractJtagDiscovery.connect
    at com.nxp.mcuxpresso.core.debug.support.linkserver.emulators.jtag.AbstractJtagDiscovery.discove
```

# 5. DAPLink Firmware version

At the time of writing there are no known issues with the supplied DAPLink firmware, and no new versions are available. However, the update mechanism is described below for reference.

## Updating the DAPLink firmware

From MCUXpresso IDE go to Help -> Additional resources -> OpenSDA Firmware Updates, this will open a web page 'OPENSDA: OpenSDA Serial and Debug Adapter'. From this page, locate the dropdown and select the MIMXRT 1170-EVK board.



**Download – OpenSDA Bootloader and Application**

**To update your board with OpenSDA applications** MIMXRT1170-EVK

**MIMXRT1170-EVK**

1. Check which Bootloader and Application version are already preprogrammed on your board.

   **OpenSDA version / bootloader**

   - V2.2 / DAPLink / (Download Binary / Source Code)

   **Default firmware application**

   - DAPLink v0244 - Supports onboard spi-flash (Download Binary / Source Code)
2. Download the binary for the additional OpenSDA Application available for your board.

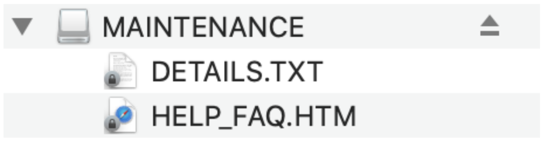
   **Latest firmware application**

   - DAPLink v0244 - Supports onboard spi-flash (Download Binary / Source Code)
3. Reprogram your adapter with some new debug, serial, MSD flash-programming application
   - Start your board in Bootloader mode keeping reset button pressed, while connecting the USB cable to the SDA USB port and release it after 1s.
   - If you have not installed already an Arm IDE toolchain, download and Install the corresponding drivers for the Application that you have selected: [Pemicro Multilink drivers] [Segger JLink drivers] [Arm CMSIS-DAP serial drivers].
   - Drag'n drop or copy and paste the Application binary file into the BOOTLOADER drive from your computer file explorer
   - Un-plug and re-plug the USB cable from the SDA USB port WITHOUT keeping reset button pressed.
   - Let your system detect the new OpenSDA interface (check the category COM_Port, USB_Controller and Jungo from your computer Device Manager to ensure that new devices have been properly detected and their driver correctly installed by your computer OS).
4. Select the right debug adapter in your project parameters.

Download the latest DAPLink binary

To install the firmware, follow the procedure below:

1. Power off the board
2. Press and hold SW3 (on the same side as the USB OpenSDA connector, below)
3. Connect a USB cable to the OpenSDA connector
4. Release SW3
5. Open a filer window and observe a drive labelled MAINTENANCE appears:

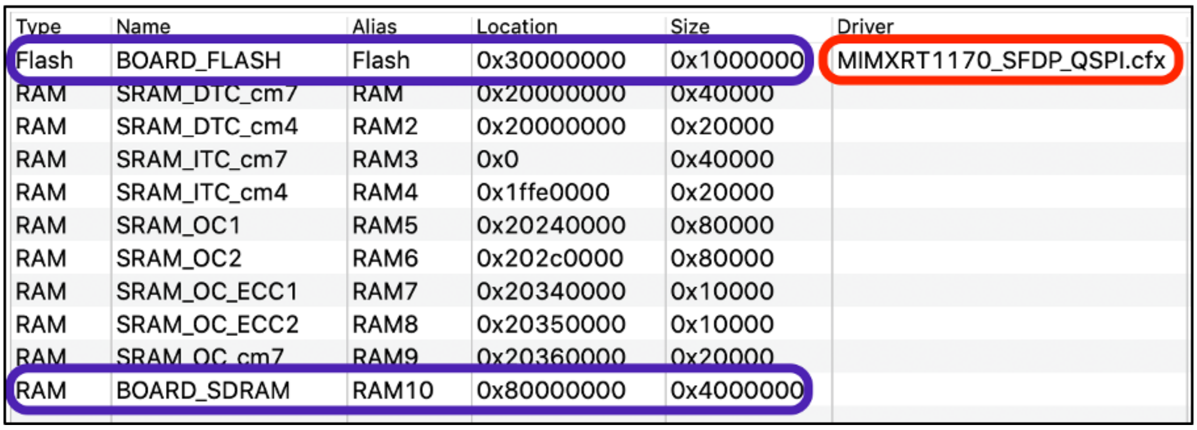


6. Open this drive and drag the previously downloaded firmware onto the filer window
    a. The filer window should close when the firmware update has completed
7. Eject the device
8. Power off the board

***Important Note:*** *This version of OpenSDA firmware supports drag and drop programming of the onboard QSPI device only. Please be aware that the maximum size that can be programmed in this manner is limited to the amount of available onchip RAM on this device.*

## 6. Memories

Below is the default arrangement of usable memories on this MCU and board. Some, or all of these regions will be visible within an MCUXpresso IDE project's Memory Configuration Editor (as below):



| Type | Name | Alias | Location | Size | Driver |
|---|---|---|---|---|---|
| Flash | BOARD_FLASH | Flash | 0x30000000 | 0x1000000 | MIMXRT1170_SFDP_QSPI.cfx |
| RAM | SRAM_DTC_cm7 | RAM | 0x20000000 | 0x40000 | |
| RAM | SRAM_DTC_cm4 | RAM2 | 0x20000000 | 0x20000 | |
| RAM | SRAM_ITC_cm7 | RAM3 | 0x0 | 0x40000 | |
| RAM | SRAM_ITC_cm4 | RAM4 | 0x1ffe0000 | 0x20000 | |
| RAM | SRAM_OC1 | RAM5 | 0x20240000 | 0x80000 | |
| RAM | SRAM_OC2 | RAM6 | 0x202c0000 | 0x80000 | |
| RAM | SRAM_OC_ECC1 | RAM7 | 0x20340000 | 0x10000 | |
| RAM | SRAM_OC_ECC2 | RAM8 | 0x20350000 | 0x10000 | |
| RAM | SRAM_OC_cm7 | RAM9 | 0x20360000 | 0x20000 | |
| RAM | BOARD_SDRAM | RAM10 | 0x80000000 | 0x4000000 | |

- *External board memories are highlighted in blue. LinkServer Flashdriver for the QSPI device highlighted in red.*
- *Note: Due to the multicore nature of this MCU, some memory regions have dedicated usage for the two cores. Some of the memories are visible to both cores at the same addresses, while for other memories each core uses a different address.*
- *Note: By default, projects will be linked to the first flash memory in this list and use the first RAM region for data, heap and stack. However, the SDK (projects and examples) may select a subset of these memories and/or change their order to control linkage (and also override default linkage using the LinktoRAM feature – see later)*

**BOARD_FLASH (QSPI) at 0x30000000:** This 16MB device is board memory external to the MCU. Programming of this device is provided by a flash driver called *MIMXRT1170_SFDP_QSPI.cfx* (for LinkServer CMSIS-DAP debug connections). On reset, (if board boot DIP switches SW1 are set for QSPI Flash Boot) the BootROM will interrogate this device and attempt to identify a specific image header. If found, the header data will be used to configure its operation (and also initialise the SDRAM). If a correct header is not found, this device will be unavailable.

*Note: MCUXpresso IDE will automatically generate and locate an appropriate header from information supplied by the SDK for new projects and as required for example projects.*

Code can be run directly from this Flash, this is known as Execute in Place (XIP). This Flash can be cached by the MCU.

*Note: The term XIP is used to differentiate from an alternative boot strategy, where the BootROM will relocate code (and data) from flash for RAM execution.*

*Note: Also see the section on Flash Drivers.*

**SRAM**

The i.MX RT1170 has 2MB total on-chip SRAM which includes:

- up to 512 KB of TCM for Cortex-M7
- 256 KB of TCM for Cortex-M4
- dedicated 1.25MB OCRAM

The configurable 512 KB shared with Cortex-M7 TCM is FlexRAM which can be allocated as a combination of I-TCM, D-TCM for Cortex-M7 and general OCRAM. The implied partitioning of this block is 256KB ITCM, 256KB DTCM, 0KB OCRAM

The SRAM includes some regions designated for ECC. When the ECC feature is enabled, the users cannot utilise the ECC memory regions in SRAM.

**SRAM_OC at 0x20240000:** This 1024KB is on chip SRAM accessed over AXI and is cacheable by the MPU. Code or data accessed from this memory will use space within the cache. If this memory is marked as not cacheable, performance will be significantly reduced.

Most example projects reserve the upper 256KB range as **NCACHE_REGION** to be used as non-cacheable area.

**SRAM_ITC_cm7 at 0x0:** This 256KB device (FlexRAM) is on chip SRAM, and tightly coupled to the Cortex-M7, and will be 'seen' by the CPU before the cache, therefore the contents of this RAM will not be cached. This RAM will provide the best deterministic performance for program execution.

**SRAM_DTC_cm7 at 0x20000000**: This 256KB (FlexRAM) is on chip SRAM, and tightly coupled to the Cortex-M7, and will be 'seen' by the CPU before the cache, therefore the contents of this RAM will not be cached. This RAM will provide the best deterministic performance for data accesses.

*Note: Tightly coupled memories may be described to the MPU with cacheable attributes, however their contents will not actually be cached. They are intended to be used for code (and data) requiring the maximum performance (and minimum power - this is a complex area and will not be discussed further in this document).*

**SRAM_ITC_cm4 at 0x1FFE0000:** This is 128KB Code TCM (LMEM RAM_L) for the Cortex-M4

**SRAM_DTC_cm4 at 0x20000000:** This is 128KB System TCM (LMEM RAM_U) for the Cortex-M4

*Note: The two TCM regions for Cortex-M4 are also available using the remapping address 0x20200000. Cortex-M7 can access Cortex-M4's TCM through this aliased address.*

**SDRAM at 0x80000000**: This 64MB device is board memory external to the MCU. This RAM block must be initialized before it can be used. An XIP Flash header contains the data for the BootROM to use to initialize this RAM memory.

*Note: If this initialization does not occur, then the RAM will not be available and a debug operation targeting this memory will fail!*

Code can be run directly from this RAM. This RAM can be cached by the MPU.

## MPU Memory attributes and Cache(s)

Complex memory systems present considerable flexibility to any system designer and much of this detail is beyond the scope of this document. However, it should be understood that this MCU contains two cores, each with its own set of caches designed to improve the system performance when accessing board memories (SDRAM, Flash and OC_RAM). Furthermore, memory regions can be assigned properties governing how memory access are treated inside that region by the CPU.

A Memory Protection Unit (MPU) is present on each core of this MCU to control these memory region properties.

New and example (board) projects will contain a function *BOARD_ConfigMPU* within the file board.c. This function performs the MPU configuration for the various memory regions including cache setup and the exact behavior of this function is controlled by a number of defined symbols. **It is strongly recommended that (if used) this function is examined and understood to ensure the memory system is configured as desired.**

*Note: While most examples in SDK version 2.10.0 use the last 256KB of SRAM_OC range as NCACHE_REGION mapped at 0x20300000, there may still be other projects that define NCACHE_REGION in the upper 16MB of SDRAM mapped as a non-cacheable region at 0x83000000 address. The SDRAM block must be initialized before it can be used. An XIP Flash header contains the data for the BootROM to use to initialize this RAM memory. If this initialization does not occur, then the SDRAM will not be available and a debug operation targeting this memory will fail!*

## 7. Flash Drivers

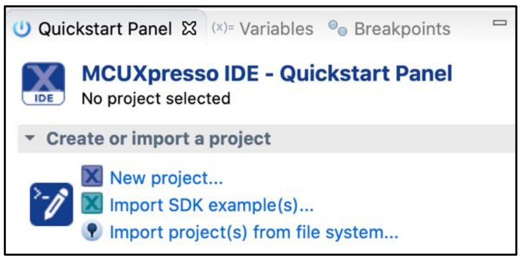
Supplied with MCUXpresso IDE v11.4.x is a QSPI flash driver for CMSIS-DAP debug connections. This driver ***MIMXRT1170_SFDP_QSPI.cfx*** will program a range of QSPI devices connected through FlexSPI1 that provide identification via the JEDEC Serial Flash Discovery Protocol.

Please see the MCUXpresso IDE v11.4.x User Guide 15.2.4 “Flash Drivers using SFDP protocol”) for more information.

# 8. New Project Creation

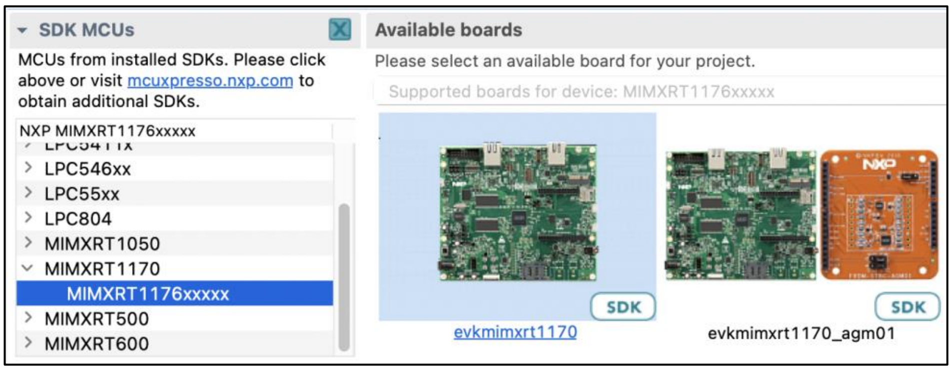
The MCUXpresso IDE New Project wizard defaults to creating projects suitable for Cortex-M7 core to execute in place (XIP) from the board QSPI using the SRAM_DTC memory for data.

1 – To create a New Project, Click 'New Project' to launch the New Project Wizard:



Quickstart Panel
Variables
Breakpoints
MCUXpresso IDE - Quickstart Panel
No project selected
Create or import a project
New project...
Import SDK example(s)...
Import project(s) from file system...

2 - Ensure the EVK board is selected (otherwise board features including flash memory will not be available):



SDK MCUs

MCUs from installed SDKs. Please click above or visit mcuxpresso.nxp.com to obtain additional SDKs.

NXP MIMXRT1176xxxxx

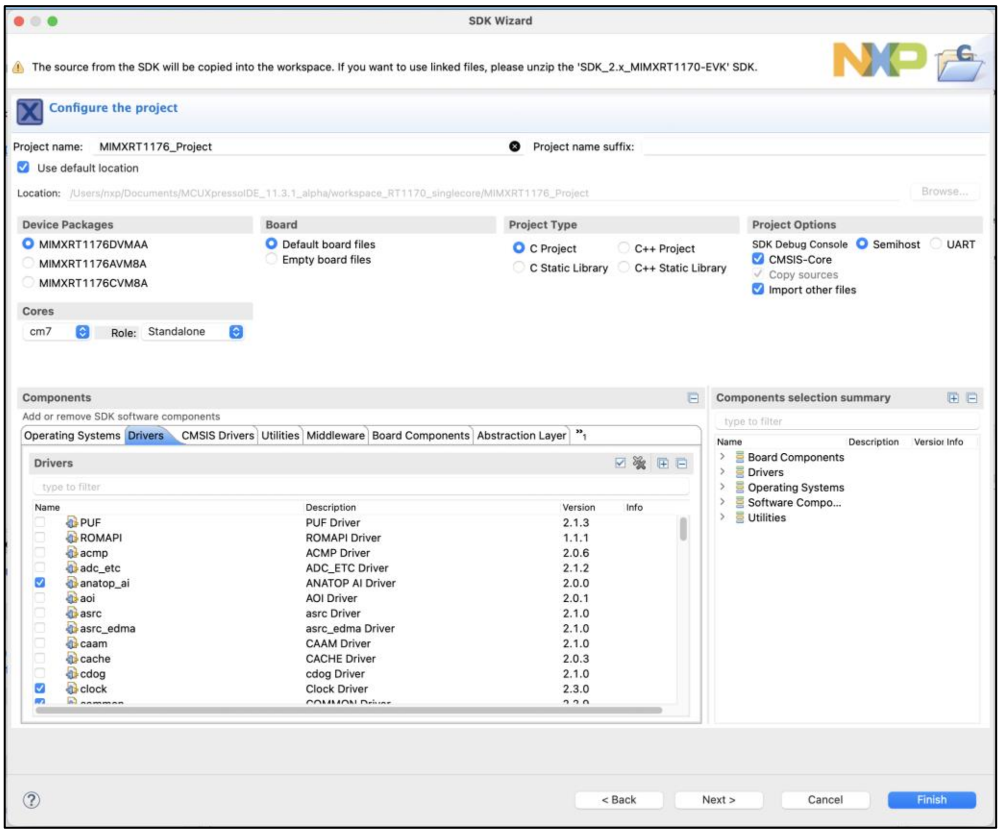
- LPC546xx
- LPC55xx
- LPC804
- MIMXRT1050
- MIMXRT1170
  - MIMXRT1176xxxxx
- MIMXRT500
- MIMXRT600

Available boards

Please select an available board for your project.

Supported boards for device: MIMXRT1176xxxxx

3 - Click Next:



SDK Wizard

The source from the SDK will be copied into the workspace. If you want to use linked files, please unzip the 'SDK_2.x_MIMXRT1170-EVK' SDK.

Configure the project

Project name: MIMXRT1176_Project    Project name suffix:

Use default location ✓

Location: /Users/nxp/Documents/MCUXpressoIDE_11.3.1_alpha/workspace_RT1170_singlecore/MIMXRT1176_Project

Device Packages: [✓ MIMXRT1176DVMAA] MIMXRT1176AVM8A MIMXRT1176CVM8A

Board: [✓ Default board files] Empty board files

Project Type: [✓ C Project] C++ Project C Static Library C++ Static Library

Project Options: SDK Debug Console [✓ Semihost] UART; CMSIS-Core ✓; Copy sources ✓; Import other files ✓

Cores: cm7    Role: Standalone

Components

Add or remove SDK software components

Operating Systems | Drivers | CMSIS Drivers | Utilities | Middleware | Board Components | Abstraction Layer

Drivers

| | Name | Description | Version | Info |
|---|---|---|---|---|
| | PUF | PUF Driver | 2.1.3 | |
| | ROMAPI | ROMAPI Driver | 1.1.1 | |
| | acmp | ACMP Driver | 2.0.6 | |
| | adc_etc | ADC_ETC Driver | 2.1.2 | |
| ✓ | anatop_ai | ANATOP AI Driver | 2.0.0 | |
| | aoi | AOI Driver | 2.0.1 | |
| | asrc | asrc Driver | 2.1.0 | |
| | asrc_edma | asrc_edma Driver | 2.1.0 | |
| | caam | CAAM Driver | 2.1.0 | |
| | cache | CACHE Driver | 2.0.3 | |
| | cdog | cdog Driver | 2.1.0 | |
| ✓ | clock | Clock Driver | 2.3.0 | |

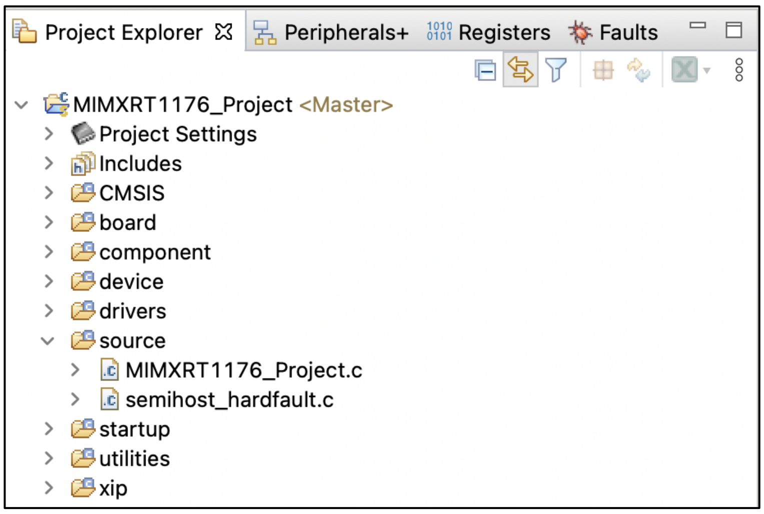
Components selection summary

| Name | Description | Version | Info |
|---|---|---|---|
| Board Components | | | |
| Drivers | | | |
| Operating Systems | | | |
| Software Compo... | | | |
| Utilities | | | |

4 - In the **Cores** section select **cm7** as core and **Standalone** as role. Click Next (accepting all the default options)

5 - Accept the default options in Advanced project settings and click Finish.

A new project (as below will be created).



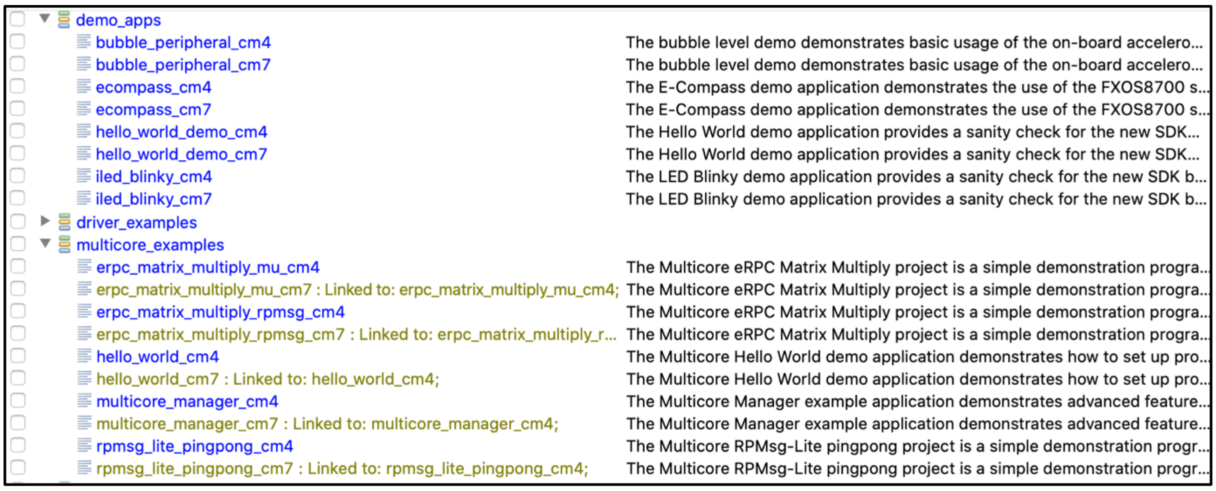
```
MIMXRT1176_Project <Master>
  Project Settings
  Includes
  CMSIS
  board
  component
  device
  drivers
  source
    MIMXRT1176_Project.c
    semihost_hardfault.c
  startup
  utilities
  xip
```

This is a 'Hello World' project/application that will execute (XIP) from the QSPI memory using the first RAM region (SRAM_DTC_cm7) for stack and global data. This RAM region is used because the SRAM_DTC_cm7 is marked by the SDK as the first RAM region for new projects (the order of the memory regions can be changed in the Advanced project settings in step 5 above).

## 9. SDK Examples

The *MIMXRT1170-EVK SDK version 2.10.0* contains many examples that can be used to demonstrate various functionalities of the dual-core i.MX RT1176 MCU.

Currently in SDK there are three types of example projects:



| | |
|---|---|
| demo_apps | |
| bubble_peripheral_cm4 | The bubble level demo demonstrates basic usage of the on-board accelero... |
| bubble_peripheral_cm7 | The bubble level demo demonstrates basic usage of the on-board accelero... |
| ecompass_cm4 | The E-Compass demo application demonstrates the use of the FXOS8700 s... |
| ecompass_cm7 | The E-Compass demo application demonstrates the use of the FXOS8700 s... |
| hello_world_demo_cm4 | The Hello World demo application provides a sanity check for the new SDK... |
| hello_world_demo_cm7 | The Hello World demo application provides a sanity check for the new SDK... |
| iled_blinky_cm4 | The LED Blinky demo application provides a sanity check for the new SDK b... |
| iled_blinky_cm7 | The LED Blinky demo application provides a sanity check for the new SDK b... |
| driver_examples | |
| multicore_examples | |
| erpc_matrix_multiply_mu_cm4 | The Multicore eRPC Matrix Multiply project is a simple demonstration progra... |
| erpc_matrix_multiply_mu_cm7 : Linked to: erpc_matrix_multiply_mu_cm4; | The Multicore eRPC Matrix Multiply project is a simple demonstration progra... |
| erpc_matrix_multiply_rpmsg_cm4 | The Multicore eRPC Matrix Multiply project is a simple demonstration progra... |
| erpc_matrix_multiply_rpmsg_cm7 : Linked to: erpc_matrix_multiply_r... | The Multicore eRPC Matrix Multiply project is a simple demonstration progra... |
| hello_world_cm4 | The Multicore Hello World demo application demonstrates how to set up pro... |
| hello_world_cm7 : Linked to: hello_world_cm4; | The Multicore Hello World demo application demonstrates how to set up pro... |
| multicore_manager_cm4 | The Multicore Manager example application demonstrates advanced feature... |
| multicore_manager_cm7 : Linked to: multicore_manager_cm4; | The Multicore Manager example application demonstrates advanced feature... |
| rpmsg_lite_pingpong_cm4 | The Multicore RPMsg-Lite pingpong project is a simple demonstration progr... |
| rpmsg_lite_pingpong_cm7 : Linked to: rpmsg_lite_pingpong_cm4; | The Multicore RPMsg-Lite pingpong project is a simple demonstration progr... |

Single-core CM7 projects:

- named with *_cm7* suffix, shown in blue color
- intended to demonstrate a functionality executing on M7 core
- target execution (XIP) from QSPI flash - they produce an image that can be programmed in flash and booted by the platform (CM7 is the default boot core in i.MX RT1170)

Single-core CM4 projects:

- named with *_cm4* suffix, shown in blue color, outside of a linked projects pair
- intended to demonstrate a functionality executing on the secondary M4 core
- linked to RAM, without a boot header – they produce an image that cannot be flashed and booted by the platform (M7 is the default boot core in i.MX RT1170)
- rely on a debugger to download the code in RAM and start executing the application

  **For more information see** Debugging a stand-alone CM4 example on the secondary core **section**

Multicore projects:

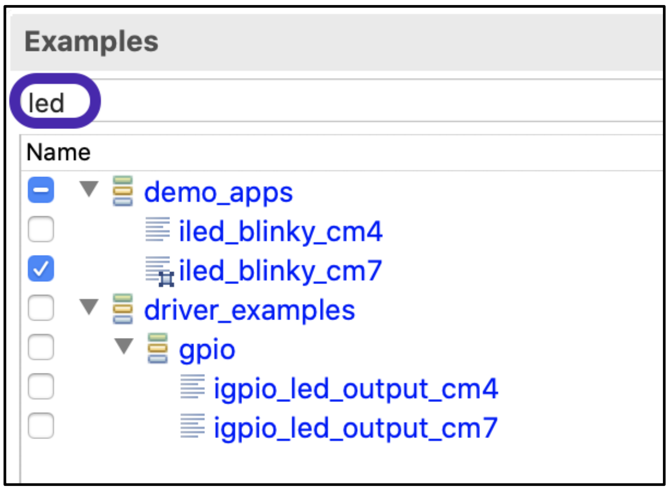
- consisting of a pair of linked projects, one for CM7 (primary core), and one for CM4 (secondary core)
- project linking is represented by the yellow color and 'Linked to:' reference
- intended to demonstrate multicore execution / interaction between CM7 and CM4 cores
- the code produced by the secondary *_cm4* project gets included in the master's image (*_cm7*) at link-time
- the generated (combined) flash image contains both the code for M7 and for M4 cores, and the resulting image can be booted by the MCU
- during the execution, the M7 code prepares and releases the M4 core to execute its dedicated code
- the cores can be debugged simultaneously (when a debug session for the master core is launched, it automatically starts a second debug session for the secondary core)
  **For more detailed information on the principles and practicalities of using multicore projects, please see the MCUXpresso IDE User Guide, chapter 18. Multicore Projects.**

## Importing an example

From the QuickStart panel select the Import SDK example wizard. Ensure the Board is selected as in the previous wizard and click Next.

Use the Filter to quickly locate the required example.

For example: type 'led' as shown below:



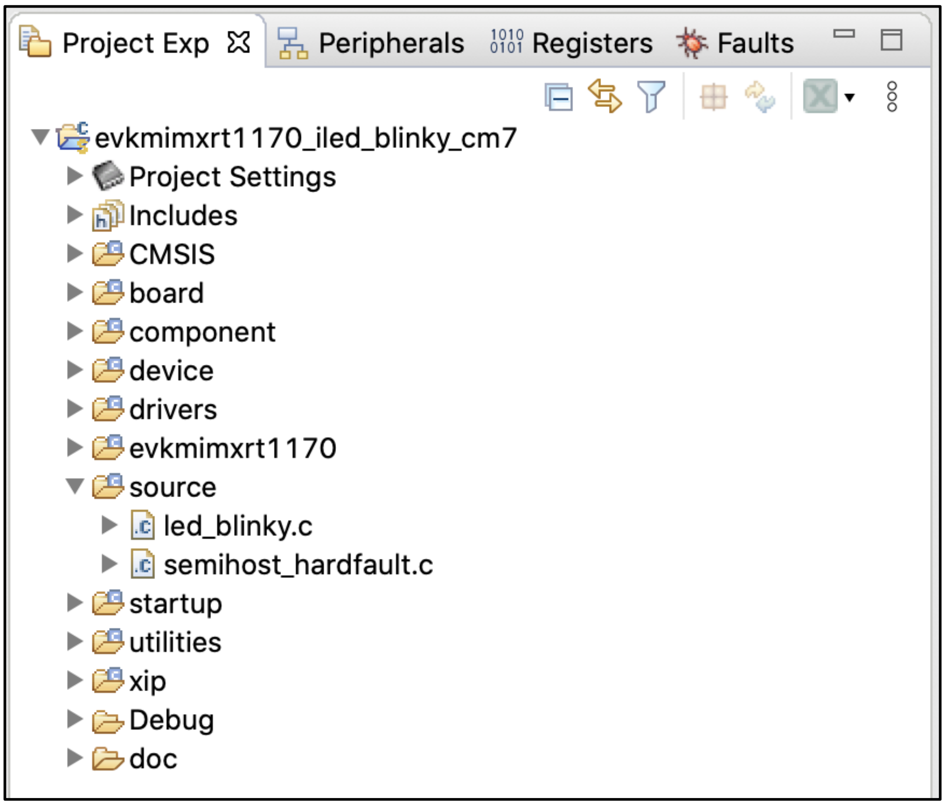
Examples

led

Name

- [–] demo_apps
  - [ ] iled_blinky_cm4
  - [✓] iled_blinky_cm7
- [ ] driver_examples
  - [ ] gpio
    - [ ] igpio_led_output_cm4
    - [ ] igpio_led_output_cm7

Select the required project and Click Finish. The Project Explorer will look as below:



```
evkmimxrt1170_iled_blinky_cm7
    Project Settings
    Includes
    CMSIS
    board
    component
    device
    drivers
    evkmimxrt1170
    source
        led_blinky.c
        semihost_hardfault.c
    startup
    utilities
    xip
    Debug
    doc
```

*Note: the project demonstrates a simple delay based blinky. It imports the required XIP header and is executed from Flash.*

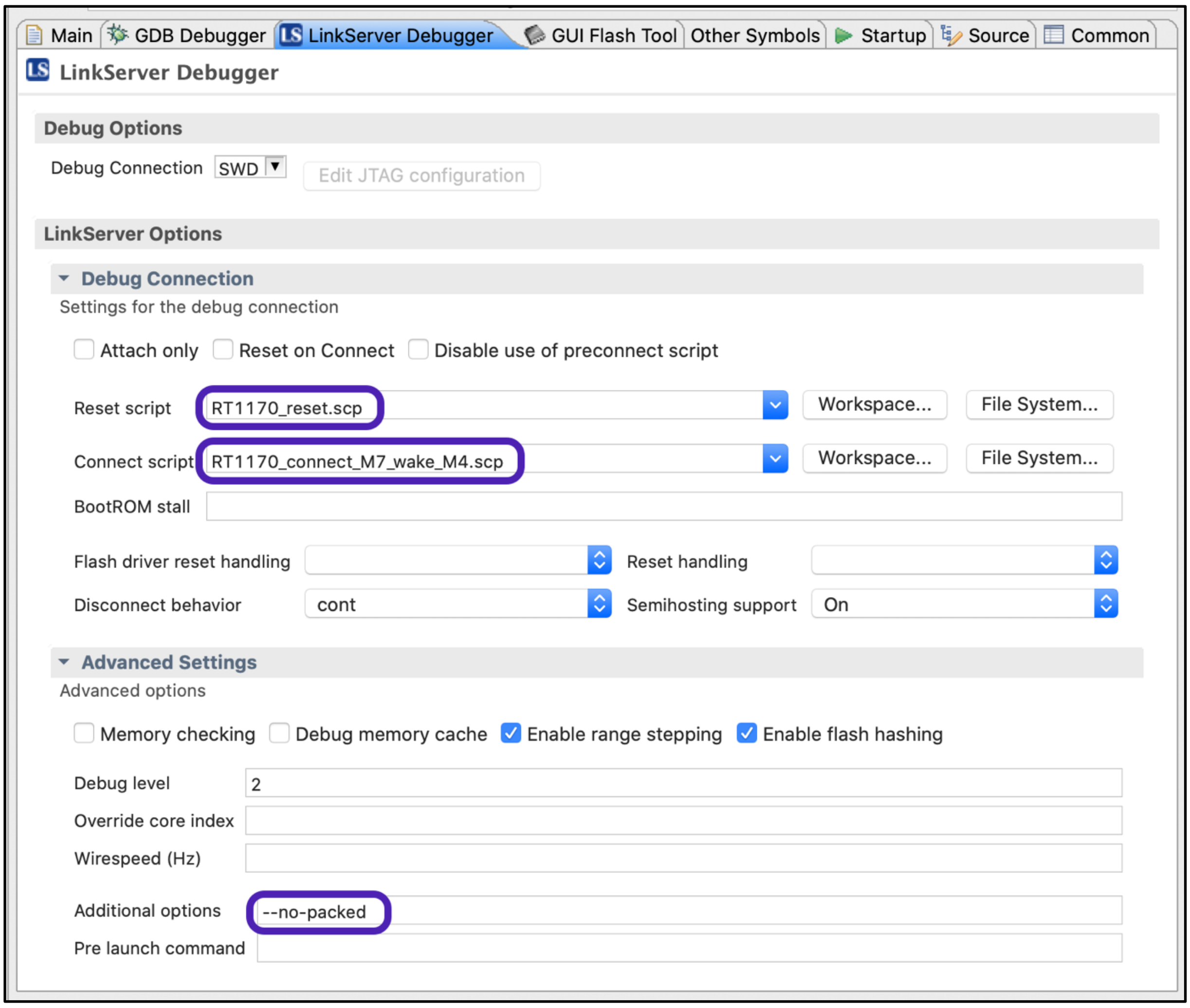
This project can be debugged directly, and you should see the board's LED toggling. If the board is power cycled, you should see the LED blinky program re-run from Flash.

## 10. Debug Launch Configurations

LinkServer Launch Configurations are automatically generated when a project is first debugged. A few important non-standard features are automatically added (as highlighted below).

The i.MX RT1170 MCU has two cores: a high-performance Cortex-M7 core and a power-efficient Cortex-M4 core. It is assumed that the MCU has the default configuration in which the CM7 is the boot core to begin execution starting from the on-chip boot ROM.

*Note: The RT1170 features a configurable fuse setting that may be blown to select the CM4 core as boot core. This configuration is briefly discussed in a subsection later on.*



Debug Connection: SWD

[ ] Attach only [ ] Reset on Connect [ ] Disable use of preconnect script

Reset script: RT1170_reset.scp

Connect script: RT1170_connect_M7_wake_M4.scp

BootROM stall:

Flash driver reset handling:

Reset handling:

Disconnect behavior: cont

Semihosting support: On

[ ] Memory checking [ ] Debug memory cache [✓] Enable range stepping [✓] Enable flash hashing

Debug level: 2

Override core index:

Wirespeed (Hz):

Additional options: --no-packed

Pre launch command:

The connect script *RT1170_connect_M7_wake_M4.scp* is required to ensure the secondary Cotex-M4 core is also visible when a debug connection is made to the primary Cortex-M7 boot core. This is required because the secondary (M4) AP is not visible while the core is held in reset. In order to make the M4 core visible to the debugger (such that it can be used for a multi-core debugging session), the connect script releases the M4 core in case it has not been released already and re-enumerates the available cores.

The reset script *RT1170_reset.scp* is required to ensure correct functionality of debugger-initiated resets while accounting for some particularities of i.MX RT1170:

1. Unlike previous MCUs, when i.MX RT1170 is being reset via SYSRESET_REQ while under debugger control, the boot ROM code will not pass control to a user's application residing in flash, instead it will enter a debug loop at a fixed PC address (0x00223104). In this situation, the debugger needs to initialize the execution context (by explicitly setting PC, SP to point to the application in flash) in order to continue the interrupted boot flow.

   The script initializes the execution context based on the application's vector table in flash (assumed to reside at 0x30002000 address)

2. When the MCU is reset, the boot core (M7) is allowed to execute, while the secondary core (M4) is held in reset. In this state, the M4 core is not visible on the debug bus. Similar to the actions performed by the connect script, the reset script also needs to ensure debug visibility of the secondary core after a reset operation.

   Unlike the connect script which needs to be less intrusive, the reset script takes greater control of the sequence used to wake up the M4 core:

   - a spin loop code is assembled in D-TCM region (0x2021FF00)
   - the M4 core clock is being set to a safe setting
   - the M4 core is reset and released to execute the spin loop prepared in advance

   This sequence serves two purposes:

   - it enables the M4 AP, required for debug visibility
   - it keeps the M4 core in a well-defined state, preventing it from executing any other code which can interfere with the platform until it will be properly initialized by an application at a later stage.

   *Note: since the M4 core is reset and released by the reset script at an early stage, an application that intends to use the M4 (secondary) core needs to follow the recommended sequence which involves resetting and releasing the M4 core. This is already taken care of when using SDK's mcmgr functions (MCMGR_StartCore). However, if an application uses its own implementation of booting the secondary core, it must respect the above principle as outlined by the reference implementation provided in mcmgr driver in SDK. At the time of writing, there are some SDK example projects (mu_* driver examples) which do not properly implement the sequence to boot the secondary core, resulting in the M4 core continuing to execute the spin loop code (0x2021FF00) instead of the intended code when debugged in MCUXpresso IDE.*

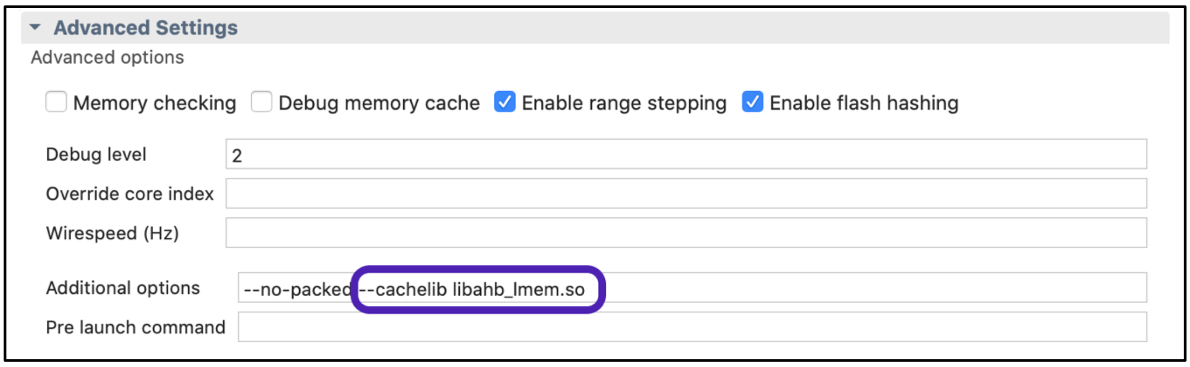
When a reset script is used, it overrides any other reset configuration in the IDE. Therefore, the reset script needs to handle the most common scenarios (see also Resets section):

- when used post flash programming - in this case it performs a SYSTEM reset
- when debugging RAM-based applications - in this case it performs a SOFT reset

The option *--no-packed* is required because the Debug Access Port on this part does not support packed transfers. If this option is not present, then certain debug operations such as semihosting may fail to operate correctly.

The above options will be added automatically to project launch configurations when importing or creating projects using the MIMXRT1170-EVK SDK.

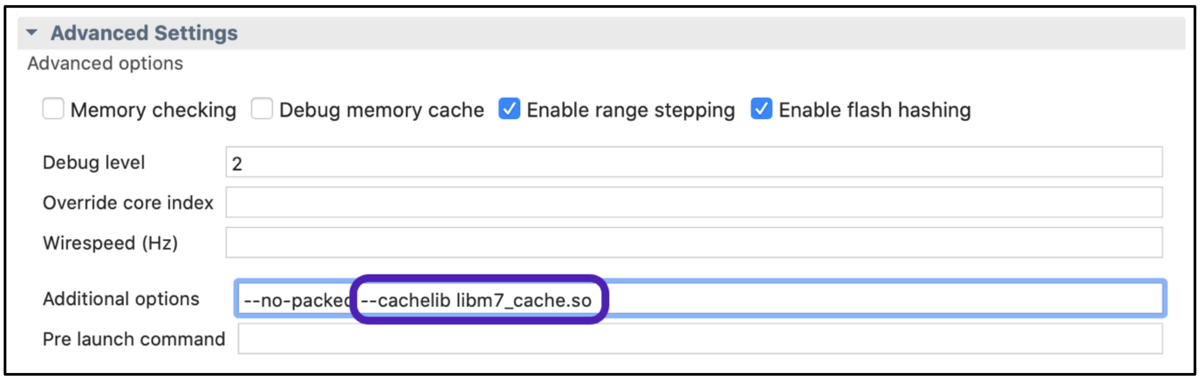
When debugging Cortex-M4 images, the following option is automatically set within the launch configuration as shown below:


Advanced Settings
Advanced options
[ ] Memory checking [ ] Debug memory cache [✓] Enable range stepping [✓] Enable flash hashing
Debug level: 2
Override core index:
Wirespeed (Hz):
Additional options: --no-packed --cachelib libahb_lmem.so
Pre launch command:

This --*cachelib libahb_lmem.s*o module ensures that cache coherence is maintained during debug operations.

Failing to specify this module when debugging Cortex-M4 with caches enabled may lead to erroneous debug information and operation, for example inability to hit or resume from software breakpoints.

When debugging Cortex-M7 images that make use of SDRAM or OC_RAM for storage of variable data (globals, stack, heap etc.) then the following option should be **manually** set within the launch configuration as shown below:


Advanced Settings
Advanced options
[ ] Memory checking [ ] Debug memory cache [✓] Enable range stepping [✓] Enable flash hashing
Debug level: 2
Override core index:
Wirespeed (Hz):
Additional options: --no-packed --cachelib libm7_cache.so
Pre launch command:

This --*cachelib libm7_cache.s*o module ensures that cache coherence is maintained during debug operations.

*Note: this module is not specified by default as its use incurs a debug performance penalty. However, failing to specify this module when debugging cached RAMs may lead to erroneous debug information and operation. This module is not required if the SDRAM or OC_RAM only contains constant or uncached data or are not used within the project.*

## Single-core RT117x derivatives

While this document mostly covers the dual-core RT1176 device, the i.MX RT1170 processor family includes single-core derivatives which feature only a Cortex-M7 core, such as RT1171 and RT1172. These derivatives are supported by a dedicated device SDK package.

When creating a new project for a single-core derivative, the set of options automatically selected to project launch configurations will be slightly different to the options selected for dual-core derivatives described previously.

Specifically, the debugger scripts are different, since they don’t need to handle the M4 core: *RT1170_connect_M7.scp* is used as connect script, while *RT1170_reset_M7.scp* is the reset script.

## Using Cortex-M4 as boot core

Dual-core RT117x processors can be configured via a fuse setting to use the Cortex-M4 core for booting (instead of the default Cortex-M7).

The boot order is controlled via a BT_CORE_SEL fuse that needs to be changed. This operation is outside the scope of this document. **Please keep in mind that such an operation is not reversible!**

Note also that due to change in the boot process this configuration cannot be supported in the regular SDK package for MIMXRT1170-EVK. A special software support package with multi-core example projects for this use-case will be provided in a separate Application Note, which is also outside the scope of this document.

When importing the example projects specially designed for using Cortex-M4 as primary boot core, the set of options automatically selected to project launch configurations will be different to the options selected for the regular use-cases described in the previous sections.

Specifically, the debugger scripts will be different, since they need to handle the ‘mirrored’ boot flow: *RT1170_connect_M4_wake_M7.scp* is used as connect script, while *RT1170_reset_M4.scp* is the reset script.

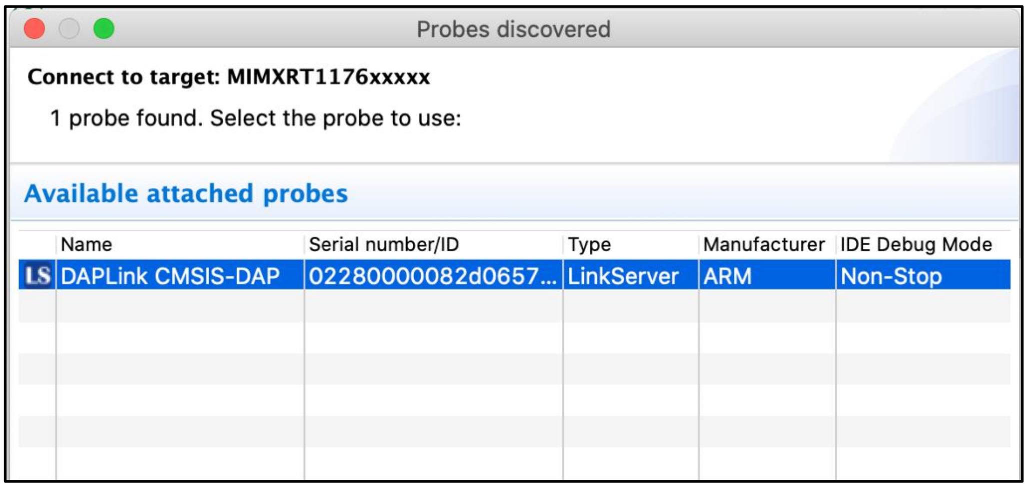
Also, the *Flash driver reset handling* option will be changed to *SOFT* for the Cortex-M4 core.

The debug configuration for Cortex-M7 will automatically select --cachelib libahb_lmem.so option to ensures that cache coherence is maintained during multicore debug operations.

## 11. Project Debug

To debug a new project or an imported example project, check the section at the start of this document and ensure the board is correctly configured and powered. Then simply select the Project and from the Quickstart panel, click Debug.

A probe discovery operation will be performed, which should locate the on board DAPLink debug probe. Select this and click OK.


Probes discovered

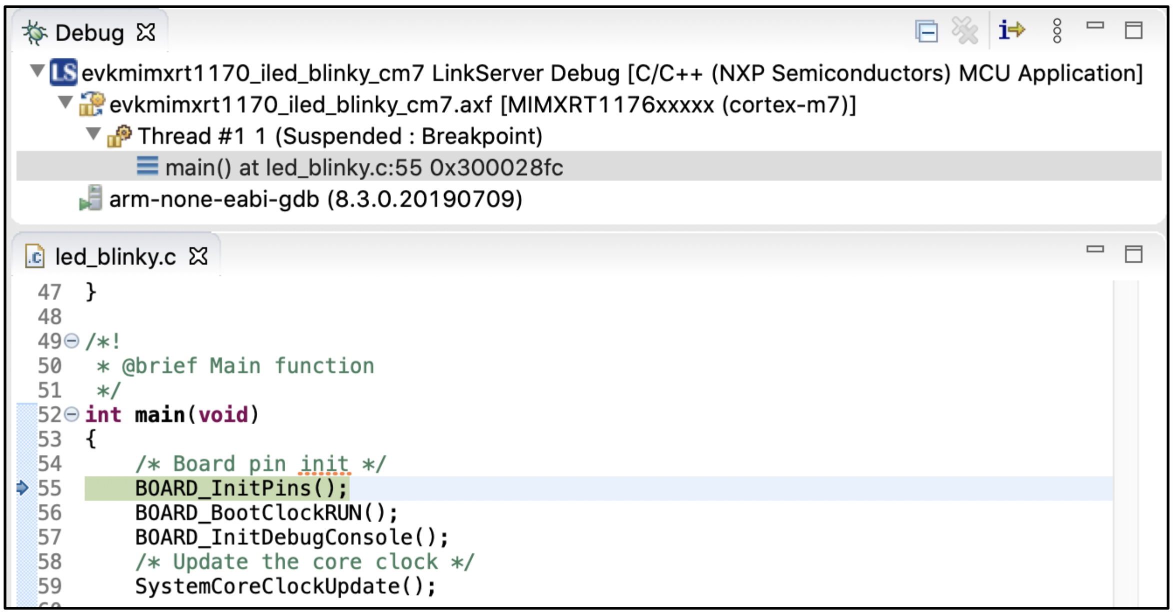
**Connect to target: MIMXRT1176xxxxx**

1 probe found. Select the probe to use:

**Available attached probes**

| Name | Serial number/ID | Type | Manufacturer | IDE Debug Mode |
|---|---|---|---|---|
| DAPLink CMSIS-DAP | 02280000082d0657... | LinkServer | ARM | Non-Stop |

The project will build and a debug operation will commence. If all goes well, you should see the following Debug stack and the application halted on main().


Debug

```
evkmimxrt1170_iled_blinky_cm7 LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
  evkmimxrt1170_iled_blinky_cm7.axf [MIMXRT1176xxxxx (cortex-m7)]
    Thread #1 1 (Suspended : Breakpoint)
      main() at led_blinky.c:55 0x300028fc
  arm-none-eabi-gdb (8.3.0.20190709)
```

led_blinky.c

```
}

/*!
 * @brief Main function
 */
int main(void)
{
    /* Board pin init */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    /* Update the core clock */
    SystemCoreClockUpdate();
```

The debug operations are logged (within the Console -> Debug Messages) and will look as below. This log contains a lot of data, the flash programming section is highlighted in bold; the connect and reset scripts (see Debug Launch Configurations) are shown in a different colour.

```
MCUXpresso IDE RedlinkMulti Driver v11.4 (Sep  6 2021 11:06:09 -
crt_emu_cm_redlink build 11)
Found chip XML file in
/Users/nxp/Documents/MCUXpressoIDE_11.4.1/workspace/evkmimxrt1170_iled_blin
ky_cm7/Debug/MIMXRT1176xxxxx.xml
Reconnected to existing LinkServer process.
============= SCRIPT: RT1170_connect_M7_wake_M4.scp =============
RT1170 Connect M7 and Wake M4 Script
DpID = 6BA02477
APID = 0x84770001
View cores on the DAP AP
DpID = 6BA02477
TAP 0: 6BA02477 Core 0: M7  APID: 84770001 ROM Table: E00FD003*
TAP 0: 6BA02477 Core 1: M4  APID: 24770011 ROM Table: E00FF003
============= END SCRIPT ========================================
Probe Firmware: DAPLink CMSIS-DAP (ARM)
Serial Number:  02280000082d06570000000000000000000000097969905
VID:PID:  0D28:0204
USB Path: USB_0d28_0204_14423000_ff00
Using memory from core 0 after searching for a good core
debug interface type      = CoreSight DP (DAP DP ID 6BA02477) over SWD TAP 0
processor type            = Cortex-M7 (CPU ID 00000C27) on DAP AP 0
number of h/w breakpoints = 8
number of flash patches   = 0
number of h/w watchpoints = 4
Probe(0): Connected&Reset. DpID: 6BA02477. CpuID: 00000C27. Info: <None>
Debug protocol: SWD. RTCK: Disabled. Vector catch: Disabled.
Content of CoreSight Debug ROM(s):
RBASE E00FD000: CID B105100D PID 000008E88C ROM (type 0x1)
ROM 1 E00FE000: CID B105100D PID 04000BB4C8 ROM (type 0x1)
ROM 2 E00FF000: CID B105100D PID 04000BB4C7 ROM (type 0x1)
ROM 3 E000E000: CID B105E00D PID 04000BB00C Gen SCS (type 0x0)
ROM 3 E0001000: CID B105E00D PID 04000BB002 Gen DWT (type 0x0)
ROM 3 E0002000: CID B105E00D PID 04000BB00E Gen (type 0x0)
ROM 3 E0000000: CID B105E00D PID 04000BB001 Gen ITM (type 0x0)
ROM 2 E0041000: CID B105900D PID 04001BB975 CSt ARM ETMv4.0 type 0x13 Trace
Source - Core
ROM 2 E0042000: CID B105900D PID 04004BB906 CSt type 0x14 Debug Control -
Trigger, e.g. ECT
ROM 1 E0043000: CID B105900D PID 04001BB908 CSt CSTF type 0x12 Trace Link -
Trace funnel/router
NXP: MIMXRT1176xxxxx
DAP stride is 1024 bytes (256 words)
Inspected v.2 External Flash Device on SPI using SFDP JEDEC ID
MIMXRT1170_SFDP_QSPI.cfx
Image 'iMXRT1170_FlexSPI_SFDP_QSPI Feb 10 2021 11:12:20'
Opening flash driver MIMXRT1170_SFDP_QSPI.cfx
Sending VECTRESET to run flash driver
Flash variant 'JEDEC_FlexSPI_Device' detected (16MB = 256*64K at
0x30000000)
Closing flash driver MIMXRT1170_SFDP_QSPI.cfx
Connected: was_reset=false. was_stopped=false
Awaiting telnet connection to port 3334 ...
GDB nonstop mode enabled
Opening flash driver MIMXRT1170_SFDP_QSPI.cfx (already resident)
Sending VECTRESET to run flash driver
Flash variant 'JEDEC_FlexSPI_Device' detected (16MB = 256*64K at
0x30000000)
Writing 30764 bytes to address 0x30000000 in Flash
30004000 done  53% (16384 out of 30764)
30008000 done 100% (32768 out of 30764)
Sectors written: 1, unchanged: 0, total: 1
Erased/Wrote sector  0-0 with 30764 bytes in 1410msec
Closing flash driver MIMXRT1170_SFDP_QSPI.cfx
Flash Write Done
Flash Program Summary: 30764 bytes in 1.41 seconds (21.31 KB/sec)
```

```
============= SCRIPT: RT1170_reset.scp =============
SYSTEM Reset
DpID = 6BA02477
TAP 0: 6BA02477 Core 0: M7  APID: 84770001 ROM Table: E00FD003*
TAP 0: 6BA02477 AP   1:     APID: 24770011 ROM Table: E00FF003
TAP 0: 6BA02477 AP   2:     APID: 54770002 ROM Table: 00000002
APID = 0x84770001
Setting M4 spin code
Setting M4 clock
Resetting M4 core
Releasing M4
View cores on the DAP AP
DpID = 6BA02477
TAP 0: 6BA02477 Core 0: M7  APID: 84770001 ROM Table: E00FD003*
TAP 0: 6BA02477 Core 1: M4  APID: 24770011 ROM Table: E00FF003
R15 = 0x00223104
Vector table SP/PC is the reset context.
PC = 0x300024E1
SP = 0x20040000
XPSR = 0x01000000
VTOR = 0x30002000
============= END SCRIPT ===========================
Stopped: Breakpoint #1
```

## 12. Project Modifications required to enable SWO Trace

At the time of writing, the latest SDK for the MIMXRT1170-EVK is version 2.10.0. This SDK does describe to the IDE that SWO Trace features are available on this board, however neither the examples nor new projects completely enable this feature.

Specifically, although some SDK examples configure the trace clock, they assign the wrong pin for trace in the pin mux settings – GPIO_DISP_B2_07, when in fact GPIO_LPSR_11 should have been used.

It is expected that a later version of the SDK will enable SWO trace out of the box for example projects.

The necessary changes described below can be manually applied in the specific files, or the Pins and Clocks Config tools can be used to add the configuration interactively and re-generate the files.

**Note: SWO trace features are not supported by DAP-Link firmware; LPC-Link2 firmware (or an external debug probe) is required.**

### Modifications to pin_mux.c

Within a project, select the *board* subfolder and within this, locate the file pin_mux.c and modify the function *BOARD_InitPins* as shown below:

```
void BOARD_InitPins(void) {
  [. . .]
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_AD_04_GPIO9_IO03,           /* GPIO_AD_04 is configured
as GPIO9_IO03 */
      0U);                                    /* Software Input On Field:
Input Path is determined by functionality */
~~  IOMUXC_SetPinMux(~~                          /* delete these lines */
~~      IOMUXC_GPIO_DISP_B2_07_ARM_TRACE_SWO,   /* GPIO_DISP_B2_07 is
configured as ARM_TRACE_SWO */~~
~~      0U);                                    /* Software Input On Field:
Input Path is determined by functionality */~~
~~  IOMUXC_SetPinConfig(~~
~~      IOMUXC_GPIO_DISP_B2_07_ARM_TRACE_SWO,   /* GPIO_DISP_B2_07 PAD
functional properties : */~~
~~      0x02U);~~
  IOMUXC_SetPinMux(                    /* add these lines */
      IOMUXC_GPIO_LPSR_11_ARM_TRACE_SWO,
      0U);
}
```

### Modifications to clock_config.c

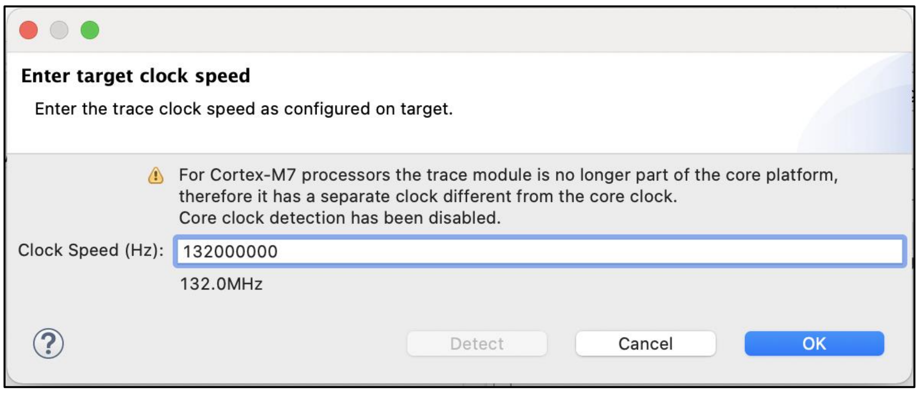
Within a project, select the *board* subfolder and within this, locate the file clock_config.c and add the lines if not already present in *BOARD_BootClockRUN* as shown below:

```
    /* Configure CSTRACE using SYS_PLL2_CLK */
    rootCfg.mux = kCLOCK_CSTRACE_ClockRoot_MuxSysPll2Out;
    rootCfg.div = 4;
    CLOCK_SetRootClock(kCLOCK_Root_Cstrace, &rootCfg);
```

CSTRACE clock must be configured such that it generates a frequency no larger than the maximum supported frequency (132MHz).

The above snippet uses SYS_PLL2_CLK (528MHz) divided by 4 to yield 132MHz trace clock.

The trace clock value configured in the application must be specified when prompted as part of the SWO Trace Config dialog.



**Enter target clock speed**

Enter the trace clock speed as configured on target.

For Cortex-M7 processors the trace module is no longer part of the core platform, therefore it has a separate clock different from the core clock.
Core clock detection has been disabled.

Clock Speed (Hz): 132000000

132.0MHz

With these changes applied, SWO trace can be captured.

Please refer to the MCUXpresso IDE SWO Trace Guide for information on the use of SWO trace features.

## 13. XIP How and Why

Traditionally a standard Cortex-M application image is programmed into an internal flash memory (of an MCU), this image is automatically executed once the MCU is reset (a bootable flash). Although essentially hidden from the user; when an MCU is reset, the first code to run is (usually) an internal BootROM, which is responsible for internal hardware setup and passing control to the user's application in Flash. From the perspective of the user however, it appears as though their application is run immediately on reset.

In the case of the RT1170, all flash memory is external to the MCU and therefore unknown to the BootROM. For the BootROM to boot an image from this flash, some additional information must be supplied to allow flash initialization and optimal configuration etc. The BootROM specification expects this configuration data to be located in an 8KB header at the start of the user's image (application). An XIP image supplies this information in an 8KB header at the start image itself. Once programmed into flash, this information can be read by the BootROM using basic subset of flash operations.

When the New Project Wizard is used and a board (evkimxrt1170) is selected, board components will automatically be pre-selected - including an xip driver. This driver component will 'pull in' the required header files into a project folder (xip). The files within this folder work in conjunction with our Managed Linker Script mechanism to create and locate an appropriate header for this target flash device.

*Note: this image header will only be created if the image is linked to the start of flash at 0x30000000 (the New Project default).*

## 14. Resets

When is a reset not a reset ...

The standard way a project is debugged is (after flash programming) for the MCU to be reset (by the debug probe) and user debug control made via an automatic breakpoint set on main(). This scheme though will only work if the application being debugged can be launched via the BootROM. In our case above, that would be an XIP image in QSPI with a correct header to describe the world and initialize the QSPI flash.

However, it can also be useful to develop and debug applications running directly in a RAM region. For this to work the user must still gain control of the executing code i.e. via a breakpoint on main() again. However, a real reset cannot be used since this will run the BootROM and this will control the boot process and not lead us to our RAM location ...

Instead for RAM projects, the debugger will issue a virtual reset using the type SOFT. A SOFT reset type, simulates some parts of a real reset including setting up the PC, SP, PSR etc. Since this is not a real reset, the MCU hardware can inherit some setup from its (pre reset) world, for example multiplexing, RAM and/or FLASH configurations. *Note: This may be beneficial, but may also cause problems or confusion if not well understood*.

SDK projects that target RAM use this SOFT reset mechanism.

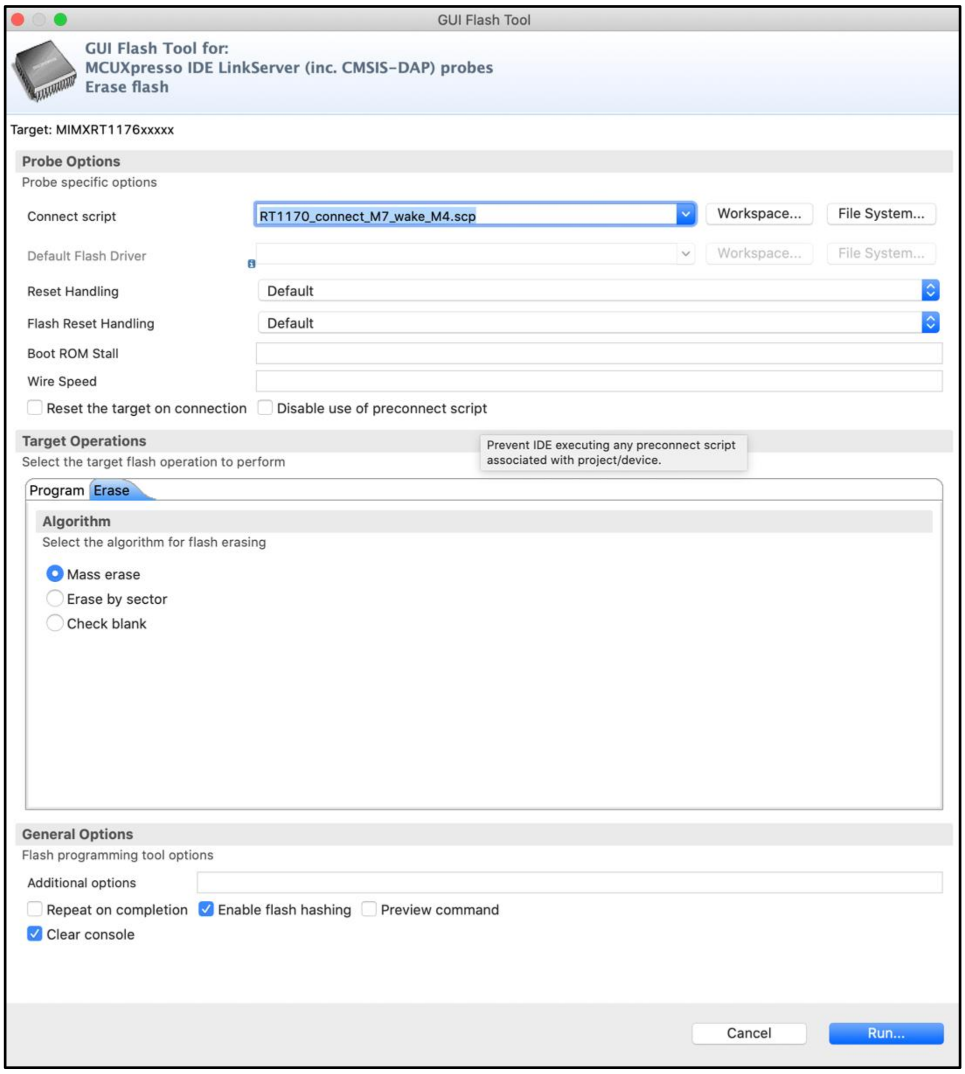
*Note: A project running from RAM may not restart successfully since Global data may only have the intended initial values on the first execution. This is a consequence of the mechanism and not a fault as such. The expected result can be achieved by using the QuickStart 'Terminate, Build and Debug' feature. All information will of course be lost if the target board is power cycled.*

# 15. Troubleshooting

## Erasing the QSPI Flash

If for any reason a 'bad' application is programmed into flash and the BootROM boots this image, the resulting executed code may affect the part in such a way that prevents new debug operations succeeding. Should this occur, the following operation should recover the situation.

1. Power of the board.
2. Change the DIP switch (SW1) to prevent booting from QSPI Flash.
   Set it to SDP mode – OFF-OFF-OFF-ON (see Overview of Board features... section)
3. As a precaution, kill any active debug components
   ➔ to do this, click the 'Clean Up Debug' toolbar icon:
4. Use the GUI Flash Tool to Erase the data in QSPI Flash.
   ➔ to do this, select a project that is configured for XIP from flash (e.g. a New project or Example project as described above) and click the GUI Flash Tool icon (the chip):
5. Select the probe as for a normal debug operation
6. Ensure the 'Erase flash memory' tab is selected. Click the 'Mass erase' radio button and then click OK.



GUI Flash Tool

GUI Flash Tool for:
MCUXpresso IDE LinkServer (inc. CMSIS-DAP) probes
Erase flash

Target: MIMXRT1176xxxxx

**Probe Options**
Probe specific options

Connect script: RT1170_connect_M7_wake_M4.scp
Default Flash Driver:
Reset Handling: Default
Flash Reset Handling: Default
Boot ROM Stall:
Wire Speed:
☐ Reset the target on connection ☐ Disable use of preconnect script

Prevent IDE executing any preconnect script associated with project/device.

**Target Operations**
Select the target flash operation to perform

Program | Erase

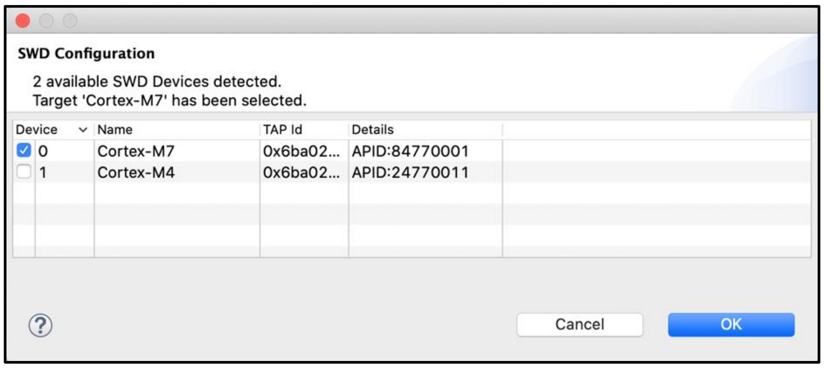
**Algorithm**
Select the algorithm for flash erasing

[✓ Mass erase]
Erase by sector
Check blank

**General Options**
Flash programming tool options

Additional options:
☐ Repeat on completion ☑ Enable flash hashing ☐ Preview command
☑ Clear console

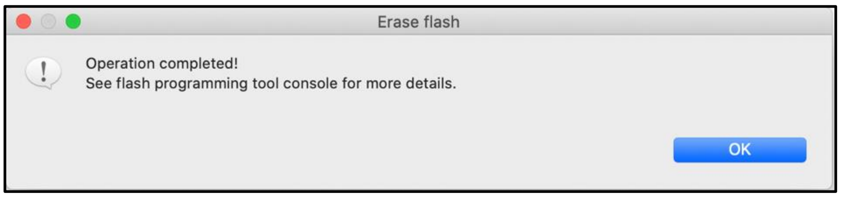
7. If asked, select the Cortex-M7 core


SWD Configuration

2 available SWD Devices detected.
Target 'Cortex-M7' has been selected.

| Device | Name | TAP Id | Details |
|---|---|---|---|
| ☑ 0 | Cortex-M7 | 0x6ba02... | APID:84770001 |
| ☐ 1 | Cortex-M4 | 0x6ba02... | APID:24770011 |

8. This will cause a mass erase of the QSPI flash, leading to the following dialogue.


Erase flash

Operation completed!
See flash programming tool console for more details.

Remember to restore the DIP switch (SW1) to set boot from QSPI and also power cycle the board. Next time the board boots, the BootROM will identify the Flash as erased and avoid running any flash image.

*Note: The time taken to erase this flash is proportional to the size of the flash and may take some time to complete in case of large flash sizes.*

## Use of __WFI() or low-power modes

A common embedded program layout will consist of a top level loop containing a __WFI() function call. This call will cause the CPU to enter a low power state pending the arrival of an interrupt. Typically, there are no negative consequences for application and/or debug operations when using WFI.

However, at the time of writing, examples and new projects for this MCU may lose debug control if __WFI() or other low-power modes like deep sleep are used. Furthermore, a WFI operation may suspend certain interrupt clocks such as used by SysTick, resulting in an application effectively stopping. If an application executes a WFI operation directly after reset, it may be difficult to recover the part.

If this occurs, holding down switch SW7 (WAKEUP) for 2 seconds should restore normal debug (and interrupt) behaviour until the next time the board is power cycled.

If debug control cannot be achieved, it is recommended to erase the flash to recover the board as explained in the section above.

## Using hardware breakpoints

The Cortex-M4 core can use hardware breakpoints only at addresses lower than 0x20000000. This means that HW breakpoints will not work when debugging code in some areas like SRAM_OC or SRAM_DTC_cm4. The Cortex-M7 core does not have this restriction.

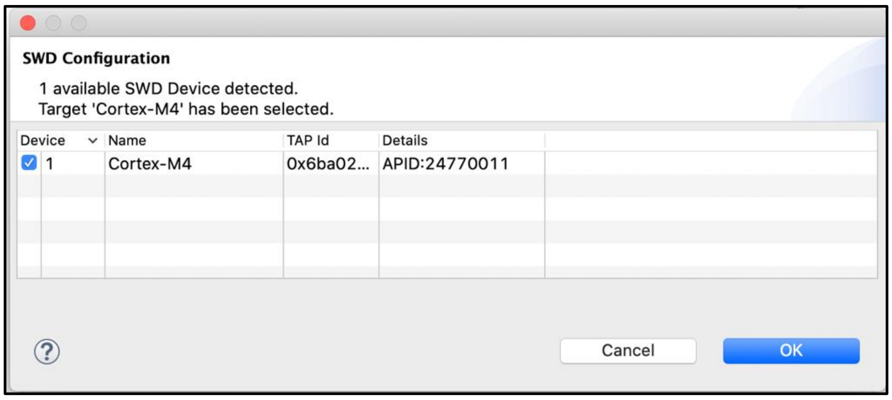
## Identifying unexpected debug failures due to unavailable M7 core

If for any reason a 'bad' application in flash crashes the booting core or affects it in such a way that prevents debugging operations, it may result in the primary M7 core not being available on the debug bus anymore.

This situation can cause the IDE to fail in unexpected ways.

Should this situation occur, any of the following manifestations are indicative of such problems:

- A SWD configuration dialog showing only the M4 core popping up at the beginning of a debug session:

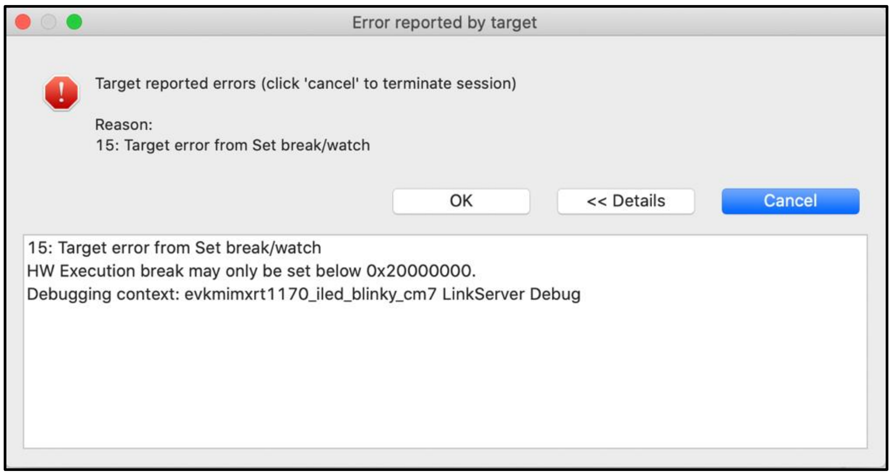


**SWD Configuration**

1 available SWD Device detected.
Target 'Cortex-M4' has been selected.

| Device | Name | TAP Id | Details |
|---|---|---|---|
| ☑ 1 | Cortex-M4 | 0x6ba02... | APID:24770011 |

- Error messages reporting HW breaks can only be set below 0x20000000



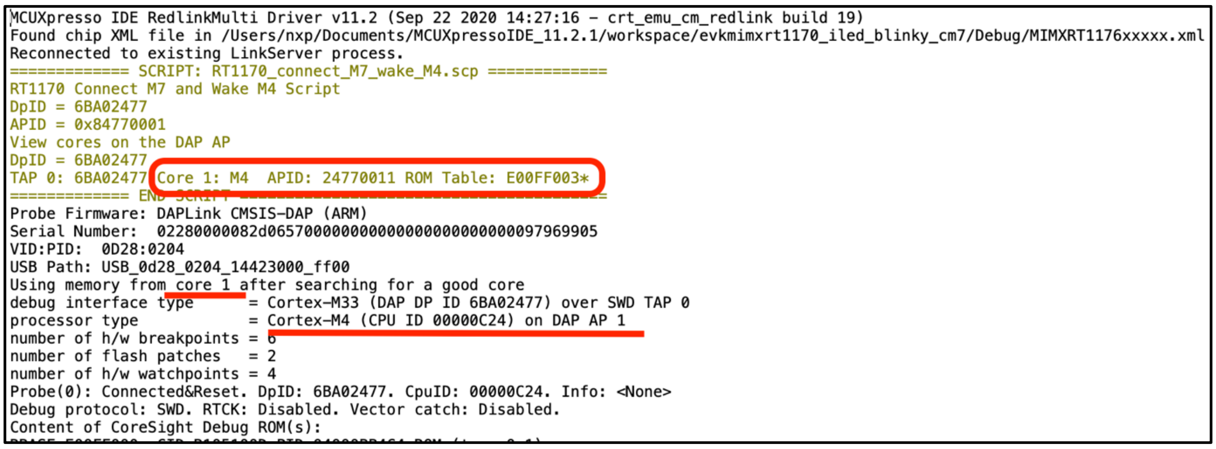
Error reported by target

Target reported errors (click 'cancel' to terminate session)

Reason:
15: Target error from Set break/watch

15: Target error from Set break/watch
HW Execution break may only be set below 0x20000000.
Debugging context: evkmimxrt1170_iled_blinky_cm7 LinkServer Debug

- Debug Messages console indicating M4 core being used (and M7 core missing from the list of available cores)



```
MCUXpresso IDE RedlinkMulti Driver v11.2 (Sep 22 2020 14:27:16 - crt_emu_cm_redlink build 19)
Found chip XML file in /Users/nxp/Documents/MCUXpressoIDE_11.2.1/workspace/evkmimxrt1170_iled_blinky_cm7/Debug/MIMXRT1176xxxxx.xml
Reconnected to existing LinkServer process.
============= SCRIPT: RT1170_connect_M7_wake_M4.scp =============
RT1170 Connect M7 and Wake M4 Script
DpID = 6BA02477
APID = 0x84770001
View cores on the DAP AP
DpID = 6BA02477
TAP 0: 6BA02477 Core 1: M4  APID: 24770011 ROM Table: E00FF003*
============= END SCRIPT ========================================
Probe Firmware: DAPLink CMSIS-DAP (ARM)
Serial Number:  02280000082d065700000000000000000000000097969905
VID:PID:  0D28:0204
USB Path: USB_0d28_0204_14423000_ff00
Using memory from core 1 after searching for a good core
debug interface type      = Cortex-M33 (DAP DP ID 6BA02477) over SWD TAP 0
processor type            = Cortex-M4 (CPU ID 00000C24) on DAP AP 1
number of h/w breakpoints = 6
number of flash patches   = 2
number of h/w watchpoints = 4
Probe(0): Connected&Reset. DpID: 6BA02477. CpuID: 00000C24. Info: <None>
Debug protocol: SWD. RTCK: Disabled. Vector catch: Disabled.
Content of CoreSight Debug ROM(s):
```

## Debugging a stand-alone CM4 example on the secondary core

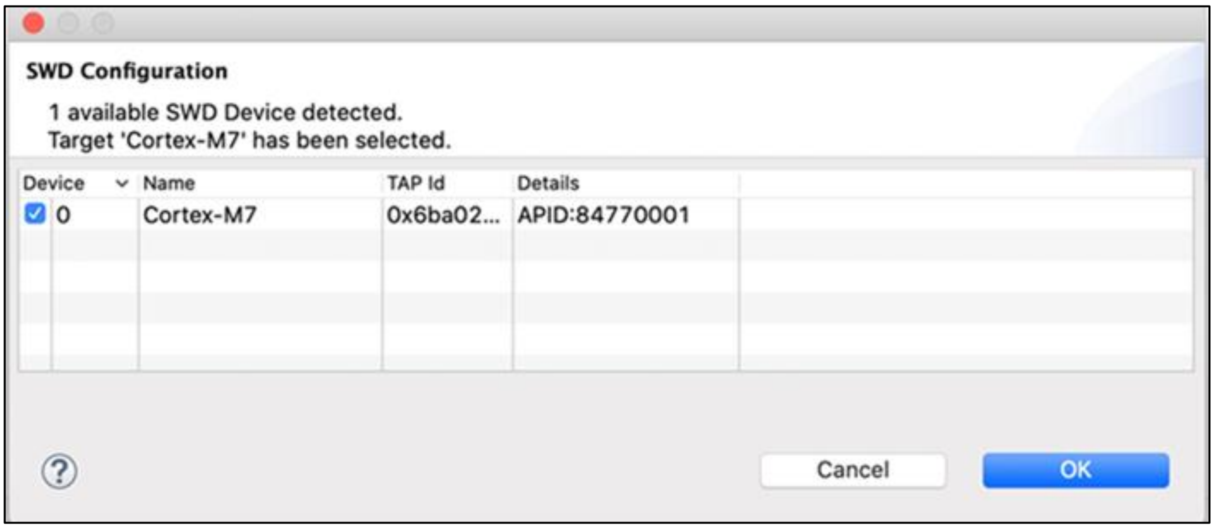
The stand-alone single-core CM4 example projects are RAM-based projects which rely on a debugger connection to download to the designated RAM area and start executing. In order to be able to execute on the M4 core, the secondary core need to be available on the debug bus.

A particularity of the RT1170 MCU is that it boots with the secondary M4 core held in reset and not available on the debug bus (thus having only the primary M7 core available). To make the secondary core available on the debug bus, software must release the M4 core. In case of multi-core projects, application code running on the M7 core prepares and releases the M4 core. Also, debugger scripts implement the sequence to make the M4 core available for connection. For example, when starting a debug session for a flash-based application (multi-core or single-core M7 projects) the debugger uses scripts to reset the MCU via SYSRESET_REQ and make the M4 core available on the debug bus for a possible subsequent debug connection to the M4 core.

But given the CM4 SDK examples are RAM-based applications, no system reset is performed when starting the debug session, therefore the M4 core may be unavailable if it hasn't yet been released by either an application or a previous debugger connection to the M7 core.

One simple way to make sure the M4 core is available is to launch a debug session using one of the CM7 projects - this will make the M4 core available, even after terminating the M7 debug session. Another option is to have an application in flash that releases the M4 core. This way the M4 core will be available after POR.

When the M4 core is not available when launching a debug session with an example project targeting the secondary core, you will get a dialog like below:



**SWD Configuration**

1 available SWD Device detected.
Target 'Cortex-M7' has been selected.

| Device | Name | TAP Id | Details |
|---|---|---|---|
| ☑ 0 | Cortex-M7 | 0x6ba02... | APID:84770001 |

This is an indication that the Cortex-M4 is not available; it offers you the choice to continue the debug connection using the Cortex-M7 core instead.

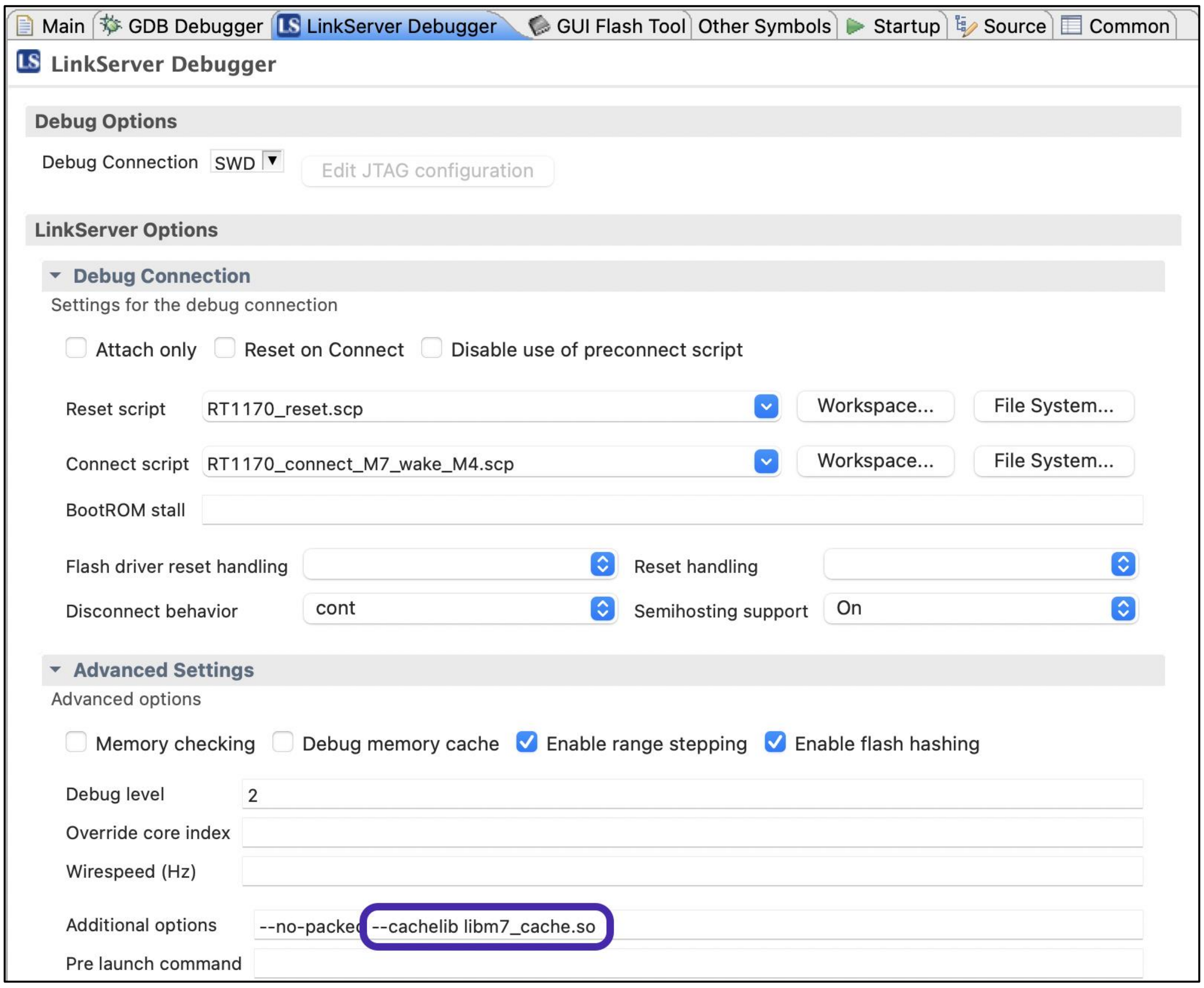
## Using external Debug Probes

The onboard DAPLink OpenSDA debug interface does not deliver particularly high debug performance. The standard 20-way 'JTAG' header may be used with external debug probes such as the LPC-Link2 for faster debug operation.

If an external LPC-Link2 is used it is recommended that the board be both powered externally and the Link JP2 should be fitted to the LPC-Link2 probe.

## Debug performance and the Data Cache

When debugging images on the Cortex-M7 core that make use of SDRAM or OC_RAM for storage of variable data (globals, stack, heap etc.) then the following option should be set within the LinkServer debug launch configuration as shown below:



Debug Connection: SWD

[ ] Attach only [ ] Reset on Connect [ ] Disable use of preconnect script

Reset script: RT1170_reset.scp

Connect script: RT1170_connect_M7_wake_M4.scp

BootROM stall:

Flash driver reset handling:

Reset handling:

Disconnect behavior: cont

Semihosting support: On

[ ] Memory checking [ ] Debug memory cache [✓] Enable range stepping [✓] Enable flash hashing

Debug level: 2

Override core index:

Wirespeed (Hz):

Additional options: --no-packed --cachelib libm7_cache.so

Pre launch command:

This module ensures that debug cache coherence is maintained, and correct debug operations may fail if this module is not specified. However, there will be a debug performance penalty when this module is used.

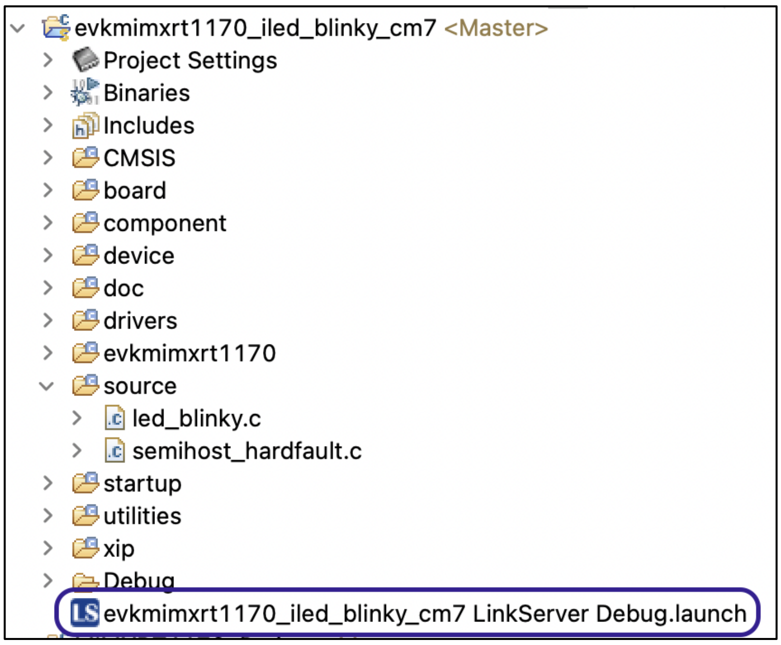
*Note: this module is not required if the SDRAM (or OC_RAM) only contains constant or uncached data.*

## Using different debug probe types

Although this document only describes using LinkServer debug connections, the IDE supports using other families of debug probes (SEGGER, PEmicro). Please see the MCUXpresso IDE User Guide for more information.

Debugging a project with a probe will cause the selected probe to be stored in the launch configuration for the current configuration (typically Debug or Release) of the current project.

If you wish to debug the same project using a different family of debug probe(s), then the simplest option is to delete the launch configuration files associated with the project and start a debug operation.



```
evkmimxrt1170_iled_blinky_cm7 <Master>
  Project Settings
  Binaries
  Includes
  CMSIS
  board
  component
  device
  doc
  drivers
  evkmimxrt1170
  source
    led_blinky.c
    semihost_hardfault.c
  startup
  utilities
  xip
  Debug
  evkmimxrt1170_iled_blinky_cm7 LinkServer Debug.launch
```

Failing to delete the launch configuration file associated with the previously used probe type will prevent probe discovery from finding debug probes of the new type.