Author: MCUXpresso IDE Team

Version: 1.0.4 15th December 2017

Overview of using the MIMXRT1050-EVK with MCUXpresso IDE v10.1

Table of Contents

Overview

This document is intended to assist users who are new to using the MIMXRT1050-EVK board. It assumes some familiarity with creating and debugging projects with MCUXpresso IDE v10.1. It is also assumes an SDK for the MIMXRT1050-EVK board has been installed into MCUXpresso IDE, and the LinkServer/CMSIS-DAP debug connection will be used for all debug operations.

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

Read Me First

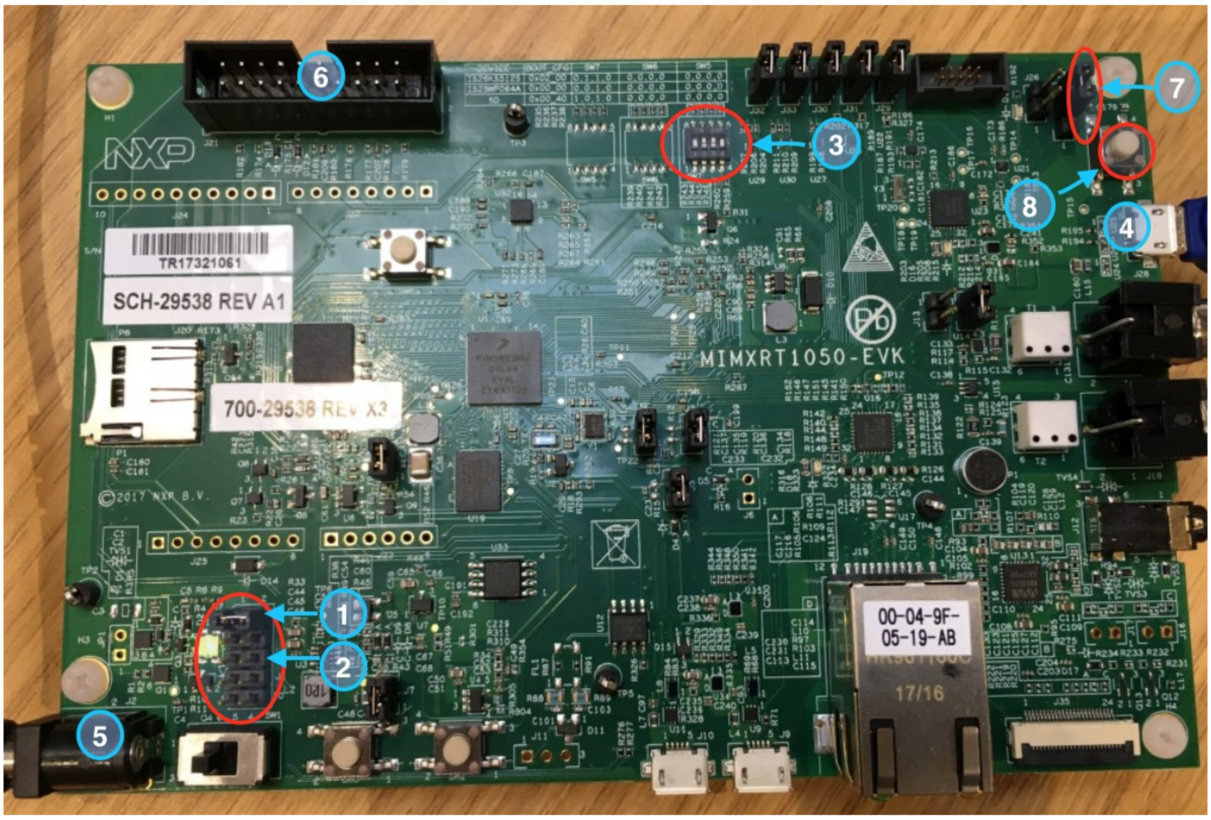
It has come to our attention that new boards are shipping with an image in flash (hyperflash) that when run can prevent a successful debug connection. Therefore, it is strongly recommended that the flash is erased.

To erase this flash, 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.

Overview of Board features related to Debug

Below is a photo of the MIMXRT1050-EVK board with key elements numbered and described:



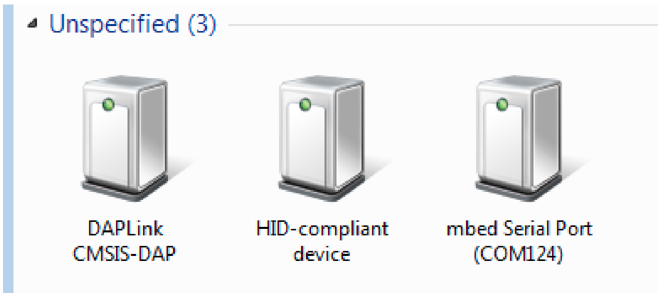
1. J1 - Link position for board powered externally (see 5). If the LED is not lit, the board is not powered.
2. J1 - Link position for board powered via the OpenSDA DAPLink debug probe connection (see 4). If the LED is not lit, the MCU is not powered.
3. SW7 - DIP switches to select boot options. Initially this should be set for boot from Hyperflash: so set as 1-off, 2-on, 3-on, 4-off.
4. J28 - OpenSDA DAPLink (CMSIS-DAP) connection (see 2). This is the recommended debug connection for initial use.
5. J2 - External power connection. If used, this should be 5v, centre +ve, and J1 set (see 1). This allows the power to the board to be controlled via switch SW1.
6. 10 way JTAG style connection for use with external debug probes such as the LPC-Link2. External debug probes will typically show significantly faster debug operations than the onboard OpenSDA debug probe.
7. J27 – OpenSDA Bootloader selection. Set jumper 1-2 for bootloader mode, 2-3 for OpenSDA debug.
8. SW4 – Reset for use with OpenSDA firmware programming when J27 is set 1-2.

Debug Connection

For initial use, we recommend using the OpenSDA USB connection (see 4 above) for the first debug operation(s).

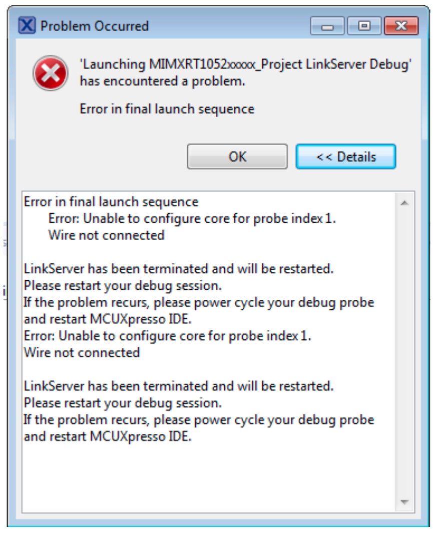
Boards are delivered with DAPLink debug probe firmware (CMSIS-DAP) pre-programmed into the OpenSDA hardware (please also see DAPLink Firmware version below).

Once an OpenSDA USB connection has been made (on Windows), 'Devices and Printers' will show the devices as below:



This connection can also power the board if the J1 link is set to the mid position (see point 2 in Board features above).

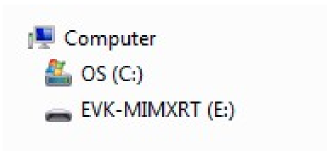
Note: if this link is set correctly, the LED next to J1 will light green. If the LED is not lit, the MCU will not be powered and debug will fail, resulting in an error similar to that below:



```
'Launching MIMXRT1052xxxxx_Project LinkServer Debug'
has encountered a problem.

Error in final launch sequence

Error in final launch sequence
    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.
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.
```

DAPLink Firmware version

Some of the initial shipments of boards contain a version of DAPLink firmware which has known issues. To check the version on your board, simply make a USB connection to the OpenSDA USB connection (as described above) and open a filer window on the EVK-MIMXRT drive.

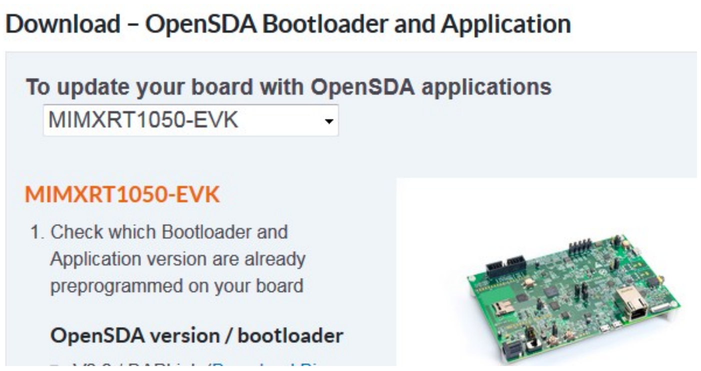


Open the file DETAILS.TXT on this drive and confirm the 'Interface Version:' is 0244 or greater.

If this is not the case, then the firmware should be updated following the procedure detailed below.

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 1050-EVT board.



Download – OpenSDA Bootloader and Application

To update your board with OpenSDA applications

MIMXRT1050-EVK

MIMXRT1050-EVK

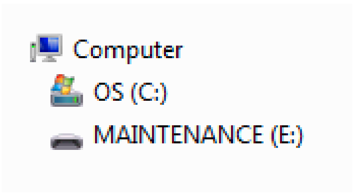
1. Check which Bootloader and Application version are already preprogrammed on your board

OpenSDA version / bootloader

Download the latest DAPLink binary – at the time this document was created this reports as DAPLink v0244.

To install the firmware, follow the procedure below:

1. Power off the board
2. Locate J27- identified in the board features as (7).
    a. Set the J27 Jumper between pins 1 and 2 (this is the lower position with the board oriented as the photograph above).
3. Press and hold SW4 – identified in the board features as (8).
4. Connect a USB cable to the OpenSDA connector
5. Release SW4
6. Open a filer window and observe a drive labelled MAINTENANCE appears:

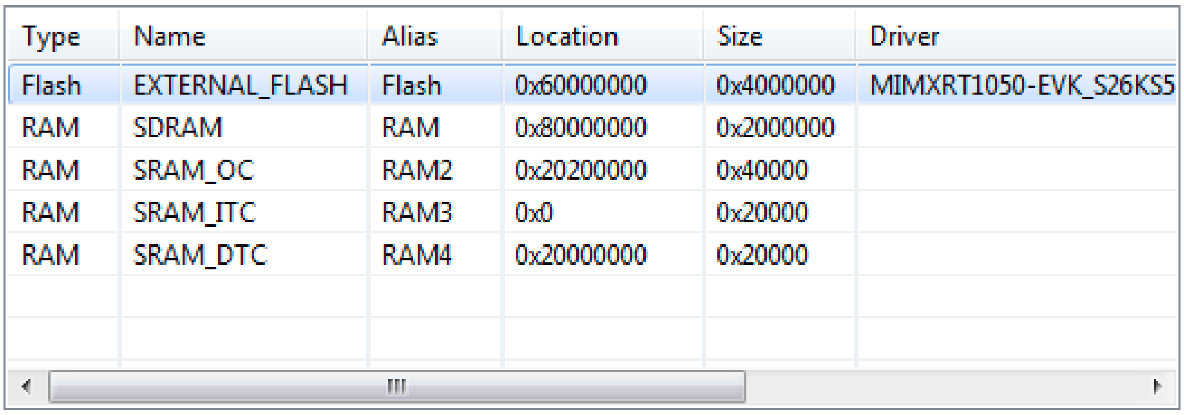

Computer
OS (C:)
MAINTENANCE (E:)

7. 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
8. Eject the device
9. Power off the board
10. Restore the J27 Jumper to between pins 2-3 (this is the upper position with the board oriented as the photograph above).

Now confirm the updated version number as described at the beginning of this section.

Memories

Below is a list of the 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	EXTERNAL_FLASH	Flash	0x60000000	0x4000000	MIMXRT1050-EVK_S26KS5
RAM	SDRAM	RAM	0x80000000	0x2000000	
RAM	SRAM_OC	RAM2	0x20200000	0x40000	
RAM	SRAM_ITC	RAM3	0x0	0x20000	
RAM	SRAM_DTC	RAM4	0x20000000	0x20000	

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).

External Flash (Hyperflash) at 0x60000000: This 64MB device is board memory external to the MCU. Programming of this device is provided by a flash driver called MIMXRT1050-EVK_S26KS512.cfx (for LinkServer CMSIS-DAP debug connections). On reset, (if SW7 is set for 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.

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

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. This mode of operation is not directly supported within MCUXpresso IDE v10.1.

SDRAM at 0x8000000: This 32MB device is board memory external to the MCU. This RAM block must be initialised before it can be used. An XIP Flash header contains the data for the BootROM to use to initialise this RAM memory. Alternatively, if a project is targeted to run from this RAM, a debug Script can be run to initialise this device. *Note: If this initialisation 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 MCU.

SRAM_OC at 0x20200000. This 256KB device is on chip SRAM. This memory should always be available.

SRAM_ITC at 0x0: This 128KB device is on chip SRAM, and tightly coupled to the MCU and will 'seen' by the CPU before the cache, therefore the contents of this RAM will not cached. This RAM will provide the best performance for program execution, data access from this device may see penalty cycles.

SRAM_DTC at 0x20000000.: This 128KB device is on chip SRAM, and tightly coupled to the MCU and will 'seen' by the CPU before the cache, therefore the contents of this RAM will not be cached. This RAM will provide the best performance for data accesses, program execution from this device may see penalty cycles.

Note: More accurately, 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).

For simplicity, the user may find creating projects targeting one of the SRAM regions simplifies initial development.

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 a cache designed to improve the system performance when accessing board memories (SDRAM and Flash). 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 this MCU to control these memory region properties.

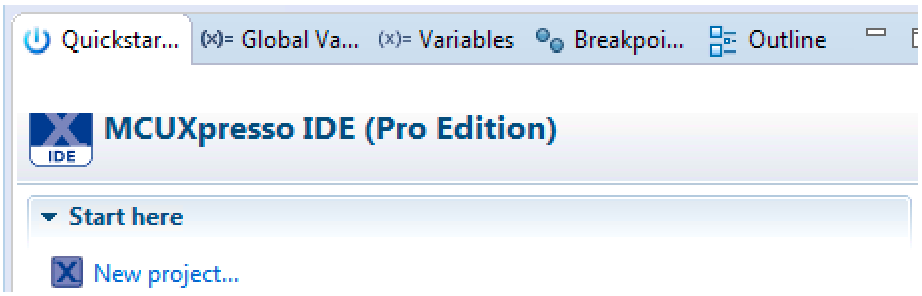
New and example (board) projects will contain a function *BOARD_ConfigMPU* within the file board.c . This function performs the MPU configuration for the various memory regions including cache setup and the exact behaviour of this function is controlled by a number of defined symbols.

It is strongly recommended that (if used) this function is examined and understood to ensure the memory system is configured as desired.

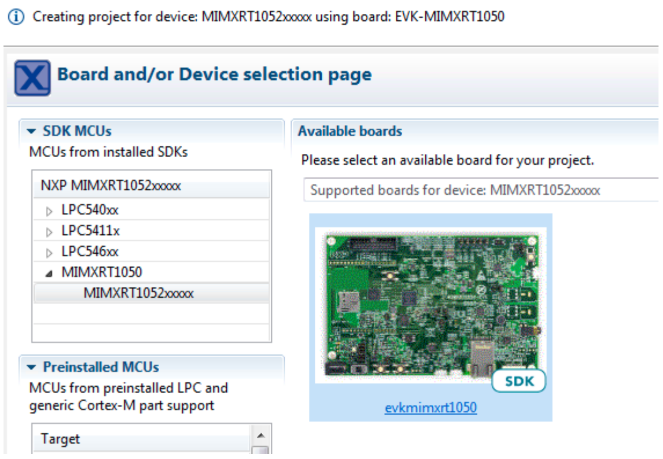
New Project Creation:

The MCUXpresso IDE New Project wizard defaults to creating projects to execute in place (XIP) from the board HyperFlash using the SDRAM for data.

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

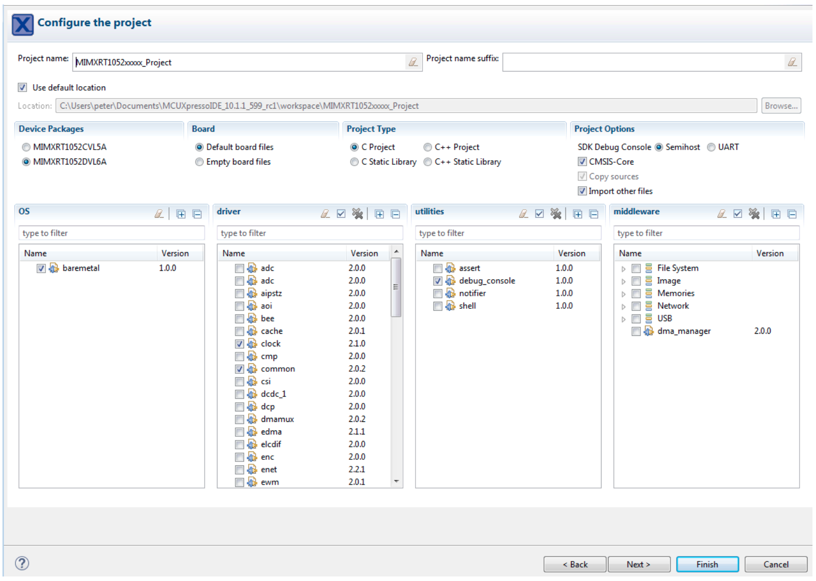


2 - Ensure the EVK board is selected (otherwise board features will not be available):


Creating project for device: MIMXRT1052xxxxx using board: EVK-MIMXRT1050

Board and/or Device selection page
SDK MCUs
MCUs from installed SDKs
NXP MIMXRT1052xxxxx
LPC540xx
LPC5411x
LPC546xx
MIMXRT1050
MIMXRT1052xxxxx
Preinstalled MCUs
MCUs from preinstalled LPC and generic Cortex-M part support
Target
Available boards
Please select an available board for your project.
Supported boards for device: MIMXRT1052xxxxx
SDK
evkmimxrt1050

3 - Click Next (accepting all the default options):


Configure the project

Project name: MIMXRT1052xxxxx_Project
Project name suffix:

[✓] Use default location

Location: C:\Users\peter\Documents\MCUXpressoIDE_10.1.1_599_rc1\workspace\MIMXRT1052xxxxx_Project

Device Packages: MIMXRT1052CVL5A [✓ MIMXRT1052DVL6A]

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

OS

Name	Version
✓ baremetal	1.0.0

driver

Name	Version
adc	2.0.0
adc	2.0.0
aipstz	2.0.0
aoi	2.0.0
bee	2.0.0
cache	2.0.1
✓ clock	2.1.0
cmp	2.0.0
✓ common	2.0.2
csi	2.0.0
dcdc_1	2.0.0
dcp	2.0.0
dmamux	2.0.2
edma	2.1.1
elcdif	2.0.0
enc	2.0.0
enet	2.2.1
ewm	2.0.1

utilities

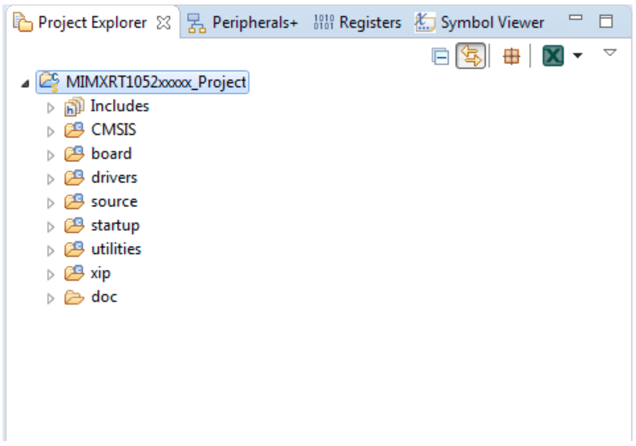
Name	Version
assert	1.0.0
✓ debug_console	1.0.0
notifier	1.0.0
shell	1.0.0

middleware

Name	Version
File System	
Image	
Memories	
Network	
USB	
dma_manager	2.0.0

4 - Then click Finish.

A new project (as below will be created).



This is a 'Hello World' project/application that will execute (XIP) from the Hyperflash memory using the first RAM region (SDRAM) for stack and global data. This RAM region is used because the SDRAM is marked by the SDK as the first RAM region for new projects (this can of course be changed). The SDRAM memory will be initialised by the BootROM (using the data from the XIP boot header) before being used by the application.

New Project issues

For a project configured to XIP from hyper flash, a define **XIP_EXTERNAL_FLASH** should also be created. This define is used to select some clock setup for the flash and change its MPU cacheable properties. Currently, the SDK does not specify this define for new projects.

Without this symbol, the projects performance will be reduced.

Also, for projects using the SDRAM, a define **SDRAM_MPU_INIT** should also be set since this is used to determine the MPU properties for the SDRAM region. Currently, the SDK does not specify this define for new projects.

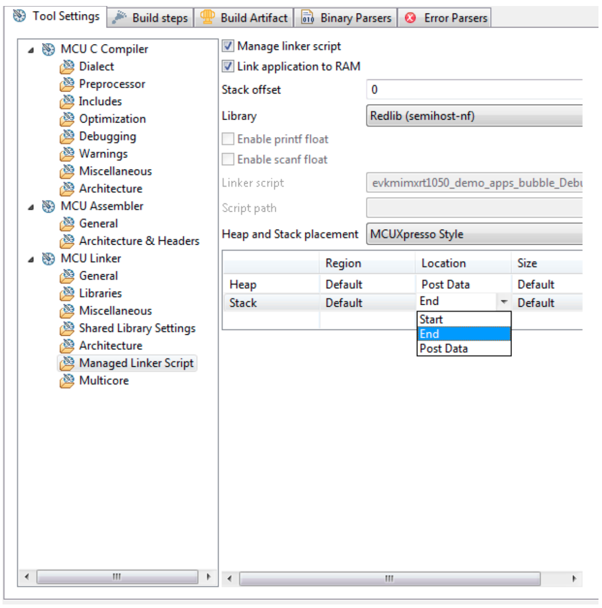
Without this symbol, the projects performance will be reduced.

Note: A new project define can be added in a number of ways, for example:

1. *Select the project in the Project Explorer.*
2. *From the QuickStart Panel -> Quick Settings -> Defined Symbols*
3. *Click +, enter the new symbol*
4. *Click OK, OK*

By default, the stack will be placed at the end of the first RAM region, therefore in our example project the stack will grow down from 0x82000000. However, the memory properties (as set in the MPU) for the last 2MB of the SDRAM are not optimal for stack operations. This problem can be corrected in a number of ways for example:

1. Edit the project properties and relocate the stack from the End of the memory region, to instead follow after the project's data – Post Data (as shown below).



[✓] Manage linker script
[✓] Link application to RAM
Stack offset: 0
Library: Redlib (semihost-nf)
[ ] Enable printf float
[ ] Enable scanf float
Linker script: evkmimxrt1050_demo_apps_bubble_Debu
Script path:
Heap and Stack placement: MCUXpresso Style

	Region	Location	Size
Heap	Default	Post Data	Default
Stack	Default	End	Default

Start
End
Post Data

2. Alternatively, edit the MPU description of the SDRAM region within the file board.c by simply deleting the Region 8 settings (in blue).

```
#if defined(SDRAM_MPU_INIT)
    /* Region 7 setting */
    MPU->RBAR = ARM_MPU_RBAR(7, 0x80000000U);
    MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 0, 0, 1, 1, 0,
ARM_MPU_REGION_SIZE_32MB);

    /* Region 8 setting */
    MPU->RBAR = ARM_MPU_RBAR(8, 0x81E00000U);
    MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 1, 0, 0, 0, 0,
ARM_MPU_REGION_SIZE_2MB);
#endif
```

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 users 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 RT1050, 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 initialisation and optimal configuration etc. The BootROM specification expects this configuration data to be located in an 8KB header at the start of the users 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.

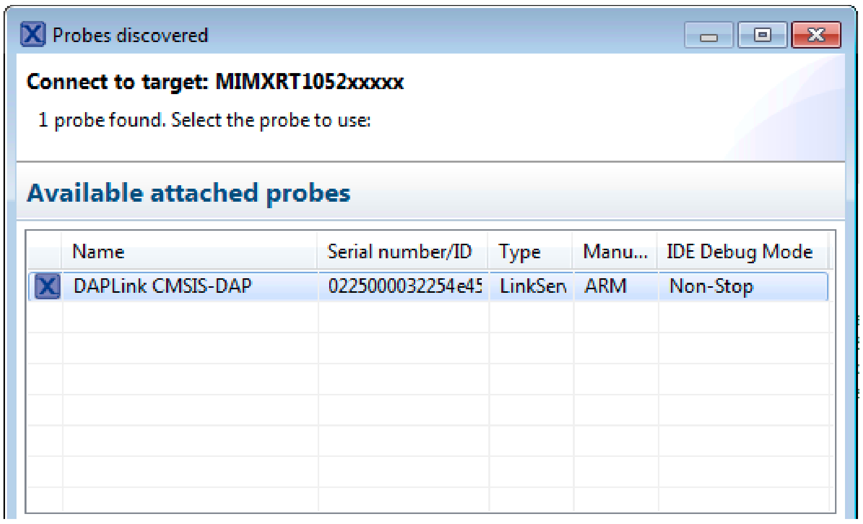
The New Project Wizard incorporates information to generate this header from template files located in the directory (xip). The files within this directory work in conjunction with our Managed Linker Script mechanism to create an appropriate header for this target flash device.

Note: this image header will only be created if the image is linked to the start of Hyperflash at 0x60000000 (the new project default).

Project Debug:

To debug this new 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

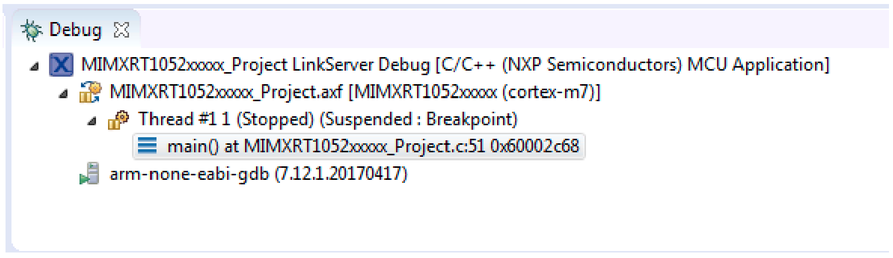
Connect to target: MIMXRT1052xxxxx

1 probe found. Select the probe to use:

Available attached probes

Name	Serial number/ID	Type	Manu...	IDE Debug Mode
DAPLink CMSIS-DAP	0225000032254e45	LinkServ	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
MIMXRT1052xxxxx_Project LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
  MIMXRT1052xxxxx_Project.axf [MIMXRT1052xxxxx (cortex-m7)]
    Thread #1 1 (Stopped) (Suspended : Breakpoint)
      main() at MIMXRT1052xxxxx_Project.c:51 0x60002c68
  arm-none-eabi-gdb (7.12.1.20170417)
```

The debug operations are logged (in the debug log) and will look as below:

```
MCUXpresso RedlinkMulti Driver v10.1 (Nov  9 2017 16:45:39 -
crt_emu_cm_redlink build 360)
Found chip XML file in
C:/Users/peter/Documents/MCUXpressoIDE_10.1.1_599_rc1/workspace/MIMXRT1052x
xxxx_Project/Debug\MIMXRT1052xxxxx.xml
Reconnected to existing redlink server (PID 4294967295)
Connecting to probe 1 core 0 (server PID unknown) gave 'OK'
Probe Firmware: DAPLink CMSIS-DAP (ARM)
Serial Number:  0225000032254e45003f800bb52900373f31000097969900
VID:PID:  0D28:0204
USB Path: \\?\hid#vid_0d28&pid_0204&mi_03#8&39758f8c&0&0000#{4d1e55b2-f16f-
11cf-88cb-001111000030}
debug interface type      = <unknown> (DAP DP ID 0BD11477) over SWD
processor type            = Cortex-M7 (CPU ID 410FC270)
number of h/w breakpoints = 8
number of flash patches   = 0
number of h/w watchpoints = 4
Probe(0): Connected&Reset. DpID: 0BD11477. CpuID: 410FC270. Info: <None>
Debug protocol: SWD. RTCK: Disabled. Vector catch: Disabled.
Content of CoreSight Debug ROM(s):
RBASE E00FD000: CID B105100D PID 000008E88C ROM dev (type 0x1)
ROM 1 E00FE000: CID B105100D PID 04000BB4C8 ROM dev (type 0x1)
ROM 2 E00FF000: CID B105100D PID 04000BB4C7 ROM dev (type 0x1)
ROM 3 E000E000: CID B105E00D PID 04000BB00C ChipIP dev SCS (type 0x0)
ROM 3 E0001000: CID B105E00D PID 04000BB002 ChipIP dev DWT (type 0x0)
ROM 3 E0002000: CID B105E00D PID 04000BB00E ChipIP dev (type 0x0)
ROM 3 E0000000: CID B105E00D PID 04000BB001 ChipIP dev ITM (type 0x0)
ROM 2 E0041000: CID B105900D PID 04001BB975 ARCH 23B:4A13r0 CoreSight dev
type 0x13 Trace Source - core
ROM 2 E0042000: CID B105900D PID 04004BB906 CoreSight dev type 0x14 Debug
Control - Trigger, e.g. ECT
ROM 1 E0040000: CID B105900D PID 04000BB9A9 CoreSight dev type 0x11 Trace
Sink - TPIU
```

```
ROM 1 E0043000: CID B105F00D PID 04001BB101 System dev (type 0x0)
CM7 Rev. 0.0  DTCM: 512KB  ITCM: 512KB
LoUU: Level 2: LoC: Level 2
Level 1 Cache Type: Instruction+Data
ICache 32K: WT: Y WB: Y RA: Y WA: Y NumSets:  512 Assoc:   2 LineSize:  8
DCache 32K: WT: Y WB: Y RA: Y WA: Y NumSets:  256 Assoc:   4 LineSize:  8
Inspected v.2 External Flash Device on SPI using SPIFI lib MIMXRT1050-
EVK_S26KS512S.cfx
Image 'MIMXRT1050-EVK_S26KS512SOct 19 2017 16:37:47'
Opening flash driver MIMXRT1050-EVK_S26KS512S.cfx
Sending VECTRESET to run flash driver
flash variant 'S26KS512S' detected (64MB = 256*256K at 0x60000000)
Closing flash driver MIMXRT1050-EVK_S26KS512S.cfx
NXP: MIMXRT1052xxxxx
Connected: was_reset=true. was_stopped=false
MCUXpressoPro Full License - Unlimited
Awaiting telnet connection to port 3331 ...
GDB nonstop mode enabled
Opening flash driver MIMXRT1050-EVK_S26KS512S.cfx (already resident)
Sending VECTRESET to run flash driver
Writing 31788 bytes to address 0x60000000 in Flash
Erased/Wrote page  0-0 with 31788 bytes in 1405msec
Closing flash driver MIMXRT1050-EVK_S26KS512S.cfx
Flash Write Done
Flash Program Summary: 31788 bytes in 1.40 seconds (22.09 KB/sec)
Starting execution using system reset and halt target
Stopped: Breakpoint #1
```

SDK Examples:

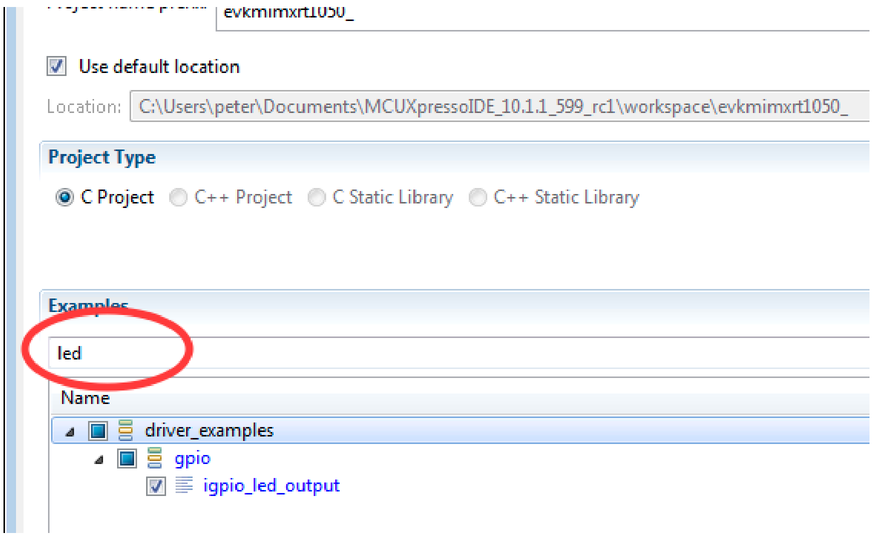
The SDK is supplied with a number of examples, however currently only one of these (hello_world_xip) has been created to execute in place (XIP) from Hyperflash. Since this example requires IO operations (via semihosting or UART), it isn't well suited to demonstrating XIP in action.

To see that an image is actually executing from flash, a simple LED blinky is a far better choice. The SDKs contains a LED blinky example called 'gpio_led_output' which by default runs from a RAM region; however, this can be converted to XIP operation (which will be discussed later in this document). First, the example must be imported.

Importing the 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. Type 'led' as shown below:


evkmimxrt1050_

☑ Use default location

Location: C:\Users\peter\Documents\MCUXpressoIDE_10.1.1_599_rc1\workspace\evkmimxrt1050_

Project Type

● C Project ○ C++ Project ○ C Static Library ○ C++ Static Library

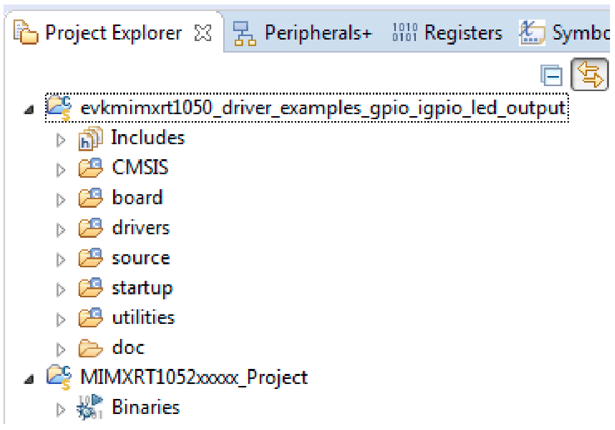
Examples

led

Name

- driver_examples
  - gpio
    - ☑ igpio_led_output

Click Finish. The Project Explorer will look as below:


Project Explorer | Peripherals+ | Registers | Symbo

- evkmimxrt1050_driver_examples_gpio_igpio_led_output
  - Includes
  - CMSIS
  - board
  - drivers
  - source
  - startup
  - utilities
  - doc
- MIMXRT1052xxxxx_Project
  - Binaries

This project can be debugged as is and you should see the board’s LED flash - however, since this project downloads to RAM, once the board is switched off, the RAM image will be lost.

Before moving onto XIP conversion etc. some discussion regarding RAM projects and debug is needed.

Resets:

When is a reset not a reset ...

The standard way a project is debugged is for the MCU to be reset (by the debug probe) and 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 Hyperflash with a correct header to describe the world and initialise the hyper flash.

However, it can also be useful to develop and debug applications running directly in a RAM region. For this to work the debugger must still gain control of the executing code i.e. via a breakpoint on main() again, but 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

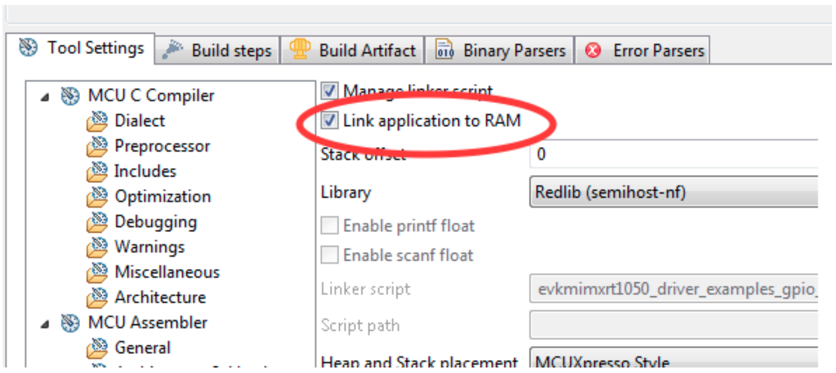
RAM projects, the debugger will issue a virtual reset using the type SOFT. A SOFT reset type, simulates some parts of a real reset including setting up the PC, SP, PSR etc. However since this is not a real reset, the MCU hardware can inherit some setup from its (pre reset) world, for example multiplexing, 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.

Building and Debugging Projects for RAM:

Most SDK example projects (for the iMX RT1050) are configured to link to RAM. This has been achieved using an MCUXpresso IDE feature that ensures any defined Flash region is ignored by the Linker resulting in an image being linked to the first defined RAM region.



☑ Manage linker script
☑ Link application to RAM
Stack offset 0
Library Redlib (semihost-nf)
☐ Enable printf float
☐ Enable scanf float
Linker script evkmimxrt1050_driver_examples_gpio_
Script path
Heap and Stack placement MCUXpresso Style

Note: As of the time of writing of this document, SDK examples that target RAM execution appear to target the DTCM RAM region at 0x20000000 using the SOFT reset type. They also make use of the above 'Link to RAM' feature while not actually including the definition of any Flash region - this feature is only intended for the case where flash is also present and is redundant if no flash is defined and may lead to user confusion (see the section below for more information).

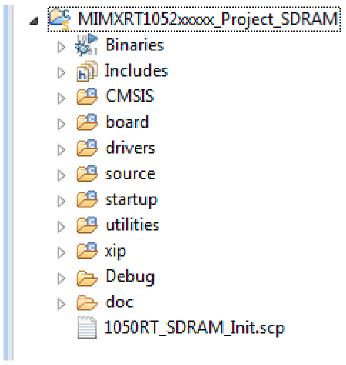
Using the SDRAM region

The TCM and OC memories should always be present when the part is powered on. However, the largest RAM on this board is the 32MB SDRAM. As mentioned above, this RAM is configured by the XIP header code for Flash projects - but for RAM projects to use this RAM area an alternative initialisation scheme is needed.

Note: If you wish to target the SDRAM region, this must be the first RAM region defined in the projects memory configuration. If a flash region is also defined, the Link application to RAM option must also be set.

LinkServer debug operations can run script files at the debug connect stage and also at the debug reset stage. In this situation, a connect script can be used to enable the SDRAM - therefore making this available for an image to download into the RAM (at 0x80000000). Such a script 1050RT_SDRAM_Init.scp.

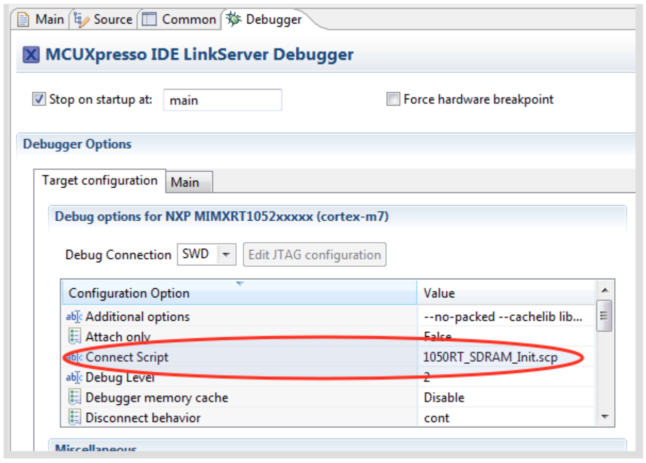
The appropriate script can be dropped directly into the project folder (as below) or copied into the product itself at <install dir>/ide/bin/Scripts.



```
MIMXRT1052xxxxx_Project_SDRAM
  Binaries
  Includes
  CMSIS
  board
  drivers
  source
  startup
  utilities
  xip
  Debug
  doc
  1050RT_SDRAM_Init.scp
```

To use the Script, edit the existing LinkServer Launch configuration (double click) and click inside the Connect Script Value field to browse for the Script. Select the Script and click OK.

Note: to create a new launch configuration either perform a debug operation via a LinkServer/CMSISDAP probe or right click on the project and select Launch Configurations -> Create new... -> MCUXpresso IDE LinkServer ...



Stop on startup at: main
Force hardware breakpoint

Debugger Options

Target configuration | Main

Debug options for NXP MIMXRT1052xxxxx (cortex-m7)

Debug Connection: SWD — Edit JTAG configuration

Configuration Option	Value
Additional options	--no-packed --cachelib lib...
Attach only	False
Connect Script	1050RT_SDRAM_Init.scp
Debug Level	2
Debugger memory cache	Disable
Disconnect behavior	cont

Miscellaneous

When run, the script will cause the following output in the connections Debug log.

```
============= SCRIPT: 1050RT_SDRAM_Init.scp =============
Setup SDRAM
Clock Init Done
SDRAM Init Done
============= END SCRIPT ================================
```

and of course SDRAM memory will be available for image download.

Note: For projects targeting the SDRAM, the define SDRAM_MPU_INIT should be set, otherwise this memory region will not be cached - so leading to reduced performance. This define is used within the file board.c.

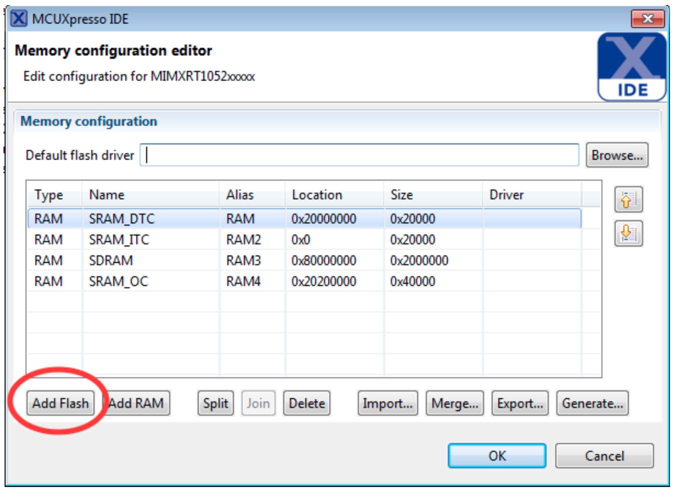
SDK Examples: Converting a RAM project to XIP from Flash:

Project built to run from RAM will of course be lost if power is removed however they can be converted to run from Flash (XIP) by following the procedure outlined below.

There are three (or four!) steps to convert an image for XIP from Flash.

1 - Change the projects memory configuration to include the HyperFlash region and add a flash driver.

Right click on the project and select Properties -> C/C++ Build -> MCU settings -> Edit
Click Add Flash



MCUXpresso IDE

Memory configuration editor

Edit configuration for MIMXRT1052xxxxx

Memory configuration

Default flash driver

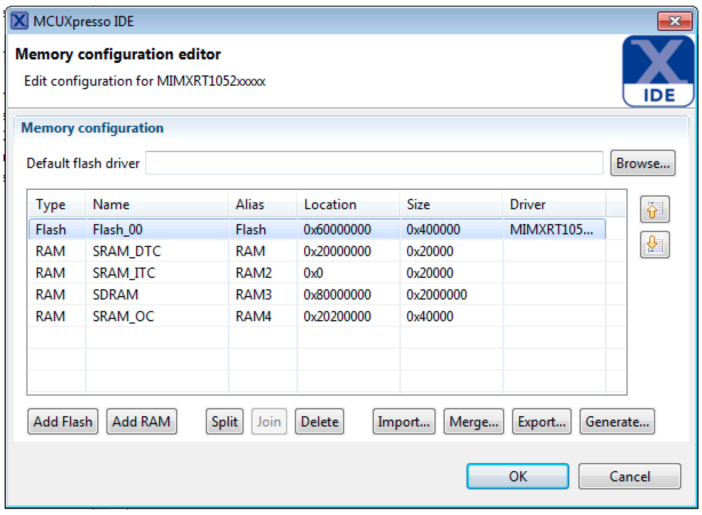
Type	Name	Alias	Location	Size	Driver
RAM	SRAM_DTC	RAM	0x20000000	0x20000	
RAM	SRAM_ITC	RAM2	0x0	0x20000	
RAM	SDRAM	RAM3	0x80000000	0x2000000	
RAM	SRAM_OC	RAM4	0x20200000	0x40000	

The hyper flash on this board is located at 0x60000000 and is 64MB in size.

Enter the new Location address at 0x60000000, and the Size as 0x400000 (or 64M), then click in the Driver field to select the flash driver 'MIMXRT1050-EVK_S26KS512S.cfx. **Note: Be sure to click outside of the edited field (i.e. inside a blank field) to ensure the edit completes.**

Your memory configuration will look like below:

Note: the first RAM (SRAM Data TCM in this case) will automatically be used for the project's stack and global memory usage. The order of the memory regions can be changed using the right hand (up/down) buttons if required.



MCUXpresso IDE

Memory configuration editor

Edit configuration for MIMXRT1052xxxxx

Memory configuration

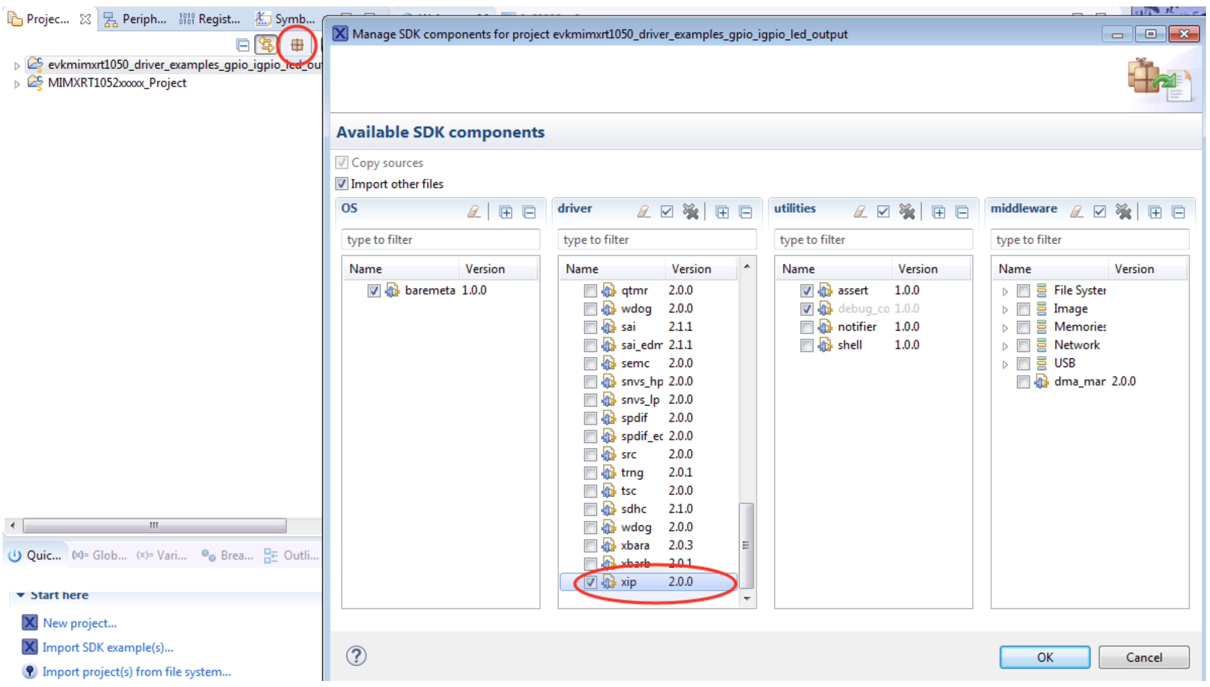
Default flash driver

Type	Name	Alias	Location	Size	Driver
Flash	Flash_00	Flash	0x60000000	0x400000	MIMXRT105...
RAM	SRAM_DTC	RAM	0x20000000	0x20000	
RAM	SRAM_ITC	RAM2	0x0	0x20000	
RAM	SDRAM	RAM3	0x80000000	0x2000000	
RAM	SRAM_OC	RAM4	0x20200000	0x40000	

Click OK.

2 - Add the XIP files.

To add XIP files to a project, select the Project and then click the Manage SDK Components as shown below:



Manage SDK components for project evkmimxrt1050_driver_examples_gpio_igpio_led_output

Available SDK components

Copy sources

[✓] Import other files

OS

Name	Version
[✓] baremeta	1.0.0

driver

Name	Version
[ ] qtmr	2.0.0
[ ] wdog	2.0.0
[ ] sai	2.1.1
[ ] sai_edm	2.1.1
[ ] semc	2.0.0
[ ] snvs_hp	2.0.0
[ ] snvs_lp	2.0.0
[ ] spdif	2.0.0
[ ] spdif_ec	2.0.0
[ ] src	2.0.0
[ ] trng	2.0.1
[ ] tsc	2.0.0
[ ] sdhc	2.1.0
[ ] wdog	2.0.0
[ ] xbara	2.0.3
[ ] xbarb	2.0.1
[✓] xip	2.0.0

utilities

Name	Version
[✓] assert	1.0.0
[✓] debug_co	1.0.0
[ ] notifier	1.0.0
[ ] shell	1.0.0

middleware

Name	Version
[ ] File Syster	
[ ] Image	
[ ] Memories	
[ ] Network	
[ ] USB	
[ ] dma_mar	2.0.0

3 - Add the symbol definition(s) as discussed above.

Right click on the project and select Properties -> C/C++ Build -> Settings -> Preprocessor -> click + and add the new define - **XIP_EXTERNAL_FLASH**

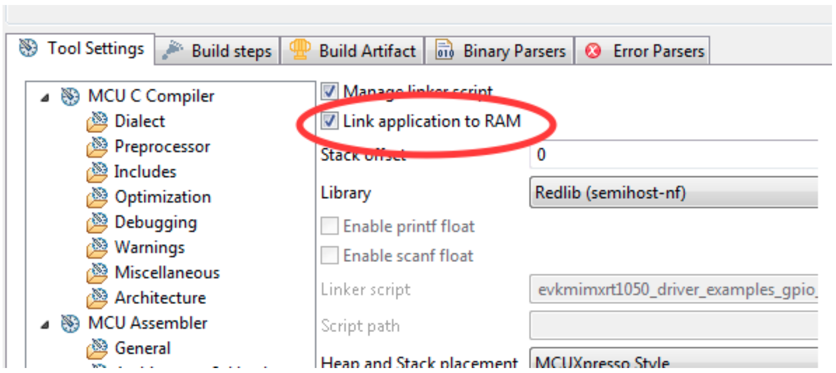
Note: If hyperflash is used, the define SDRAM_MPU_INIT should also be made

4 – Restore Linkage to Flash (if required).

For projects built to execute from Flash (the historically normal case) - there is a quick setting that can be used to force a project to instead link against the first RAM bank. The SDK is causing this feature to be set even when no Flash is configured. Hence when the project is rebuilt, it will link still to RAM even though we have just defined a Flash region.

To restore linkage to Flash:

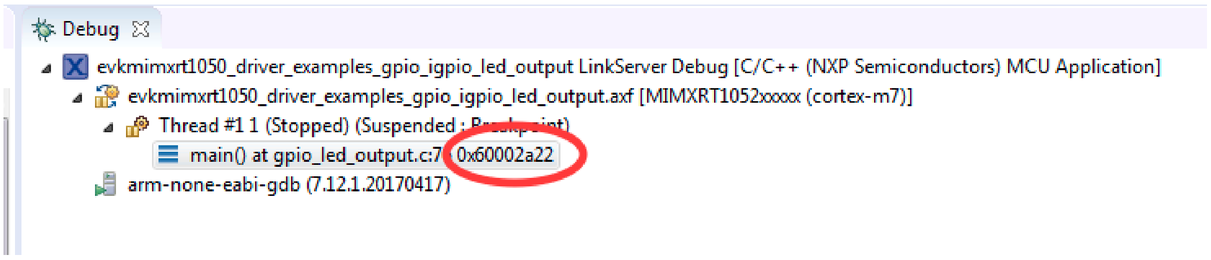
Right click on the project and select Properties -> C/C++ Build -> Settings -> Managed Linker Scripts -> Uncheck - Link application to RAM



- [✓] Manage linker script
- [✓] Link application to RAM
- Stack offset: 0
- Library: Redlib (semihost-nf)
- [ ] Enable printf float
- [ ] Enable scanf float
- Linker script: evkmimxrt1050_driver_examples_gpio_
- Script path:
- Heap and Stack placement: MCUXpresso Style

Now, debug this application as before:

You should see the PC stopped at a Hyperflash address i.e. in the 0x60000000 range as below:



```
evkmimxrt1050_driver_examples_gpio_igpio_led_output LinkServer Debug [C/C++ (NXP Semiconductors) MCU Application]
  evkmimxrt1050_driver_examples_gpio_igpio_led_output.axf [MIMXRT1052xxxxx (cortex-m7)]
    Thread #1 1 (Stopped) (Suspended : Breakpoint)
      main() at gpio_led_output.c:7 0x60002a22
  arm-none-eabi-gdb (7.12.1.20170417)
```

Note: as discussed above, projects targeting RAM will use a SOFT reset type. This reset type is assumed by the IDE to be the default for RAM projects however, some examples may explicitly set this reset type within the launch configuration. Therefore, if a RAM based project already has a launch configuration this SOFT reset type may remain after the conversion to XIP. The simplest way to fix this problem is to delete any existing launch configuration from the project and allow a new one to be automatically re-created. Alternatively, it can be edited and the SOFT option removed leaving no entry for reset type.

Finally, the real test…

Switch off power to the board (or unplug the OpenSDA debug connection). Remember to ensure the DIP switch (SW7) is set boot from Hyperflash, then reapply power. The board should boot the image and the LED should flash!

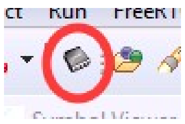
Note: although this example uses semihosting for printf, there is a handler included to ensure the example runs without debug support.

Troubleshooting

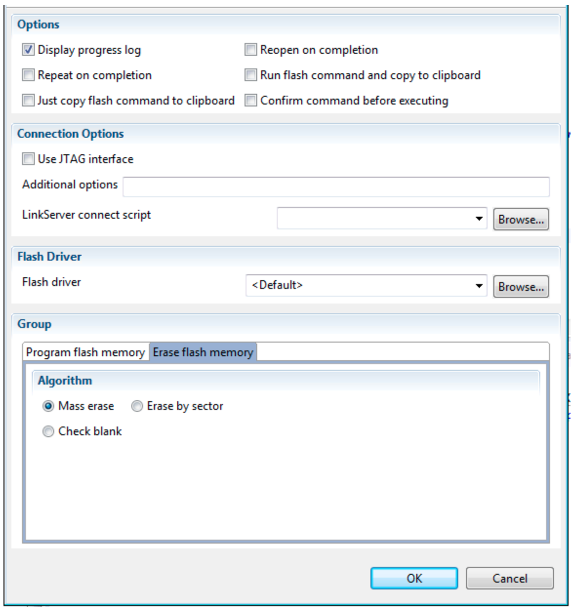
Erasing the Hyperflash

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 off the board.
2. Change the DIP switch (SW7) to prevent booting from hyperFlash e.g. set 1- on.
3. Use the GUI Flash Programmer to Erase the data in hyperFlash
   - ➔ to do this, select a project that is configured for XIP from flash (e.g. a New project as described above) and click the GUI flash programmer icon (the chip):



4. Select the probe as for a normal debug operation, then ensure the 'Erase flash memory' tab is selected. Click the 'Mass erase' radio button and then click OK



Options
- [✓] Display progress log
- [ ] Reopen on completion
- [ ] Repeat on completion
- [ ] Run flash command and copy to clipboard
- [ ] Just copy flash command to clipboard
- [ ] Confirm command before executing

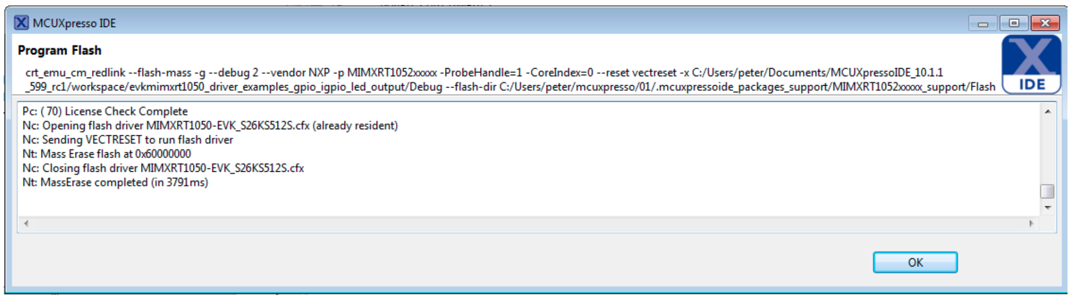
Connection Options
- [ ] Use JTAG interface
- Additional options:
- LinkServer connect script:

Flash Driver
- Flash driver: <Default>

Group
- Program flash memory | Erase flash memory

Algorithm
- [● Mass erase] ○ Erase by sector
- ○ Check blank

5. This will cause a mass erase of the hyper flash, leading to the following dialogue.



Program Flash

```
crt_emu_cm_redlink --flash-mass -g --debug 2 --vendor NXP -p MIMXRT1052xxxxx -ProbeHandle=1 -CoreIndex=0 --reset vectreset -x C:/Users/peter/Documents/MCUXpressoIDE_10.1.1
_599_rc1/workspace/evkmimxrt1050_driver_examples_gpio_igpio_led_output/Debug --flash-dir C:/Users/peter/mcuxpresso/01/.mcuxpressoide_packages_support/MIMXRT1052xxxxx_support/Flash
```

```
Pc: ( 70) License Check Complete
Nc: Opening flash driver MIMXRT1050-EVK_S26KS512S.cfx (already resident)
Nc: Sending VECTRESET to run flash driver
Nt: Mass Erase flash at 0x60000000
Nc: Closing flash driver MIMXRT1050-EVK_S26KS512S.cfx
Nt: MassErase completed (in 3791ms)
```

Remember to restore the DIP switch (SW7) to set boot from Hyperflash 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.

Debug performance

You may observe slower than expected debug performance (stepping etc.) particularly when using the DAPLink OpenSDA debug connection.

The delay you see is related to the conservative cache handling performed by the debugger to maintain coherency between it and the MCU. Since coherency issues rarely occur you can remove this handling altogether by editing a projects launch configuration.

Under "Addition options", you should see:

--no-packed --cachelib libm7_cache.so

Edit the "Additional options" to remove the cache plugin leaving only:

--no_packed

Furthermore, while the OpenSDA debug circuit provided on this board is convenient, faster debug solutions are available. NXP's LPC-Link2 debug probe will provide a significant increase in debug performance for a low cost.

SDRAM

A further final thought. Since the BootROM in conjunction with the XIP header - configures and enables the SDRAM... this area of memory should not be used unless a mechanism for enabling the SDRAM is provided (via the supplied LinkServer script).