Author: MCUXpresso IDE Team
Version: 1.0.0 - 25th February 2019

LPC55xx Multicore Applications with MCUXpresso IDE

Note:

- This article assumes the use of MCUXpresso IDE v10.3.1 (or later).
- An MCUXpresso SDK for LPC55xx will also be required.
  - This must be "SDK Version: 2.5.0 (released 2019-02-25)" or later – which can be obtained from https://mcuxpresso.nxp.com
- This article also assumes that you are using an LPCXpresso55S69 EVK board. For more details of this board, see:
  - https://www.nxp.com/LPC55S69-EVK

Introduction

The LPC55xx series of MCUs contain a main or 'Master' Cortex M33 core, plus an optional 'Slave' Cortex M33 core (with reduced functionality).

On devices with both cores implemented, after a power-on or Reset, the Master core starts executing, but the Slave core is held in reset (i.e its code does not start to execute). Code running on the Master core is then responsible for releasing the Slave core from reset; hence the names Master and Slave. In reality Master and Slave only applies to the booting process; after boot, your application may treat either of the cores as the Master or Slave.

MCUXpresso IDE provides highly flexible support for creating and debugging LPC55xx applications, for both single core and multicore systems. This article details the process of creating and debugging such multicore applications. Note that it is assumed that you are familiar with standard MCUXpresso IDE behavior, such as downloading and installing an appropriate MCUXpresso SDK for an MCU, importing examples from the SDK, creating new projects and editing memory configuration in project properties. For more details of these and other operations, please consult the MCUXpresso IDE User Guide provided in the product. The User Guide also provides more general information on multicore projects and debugging.

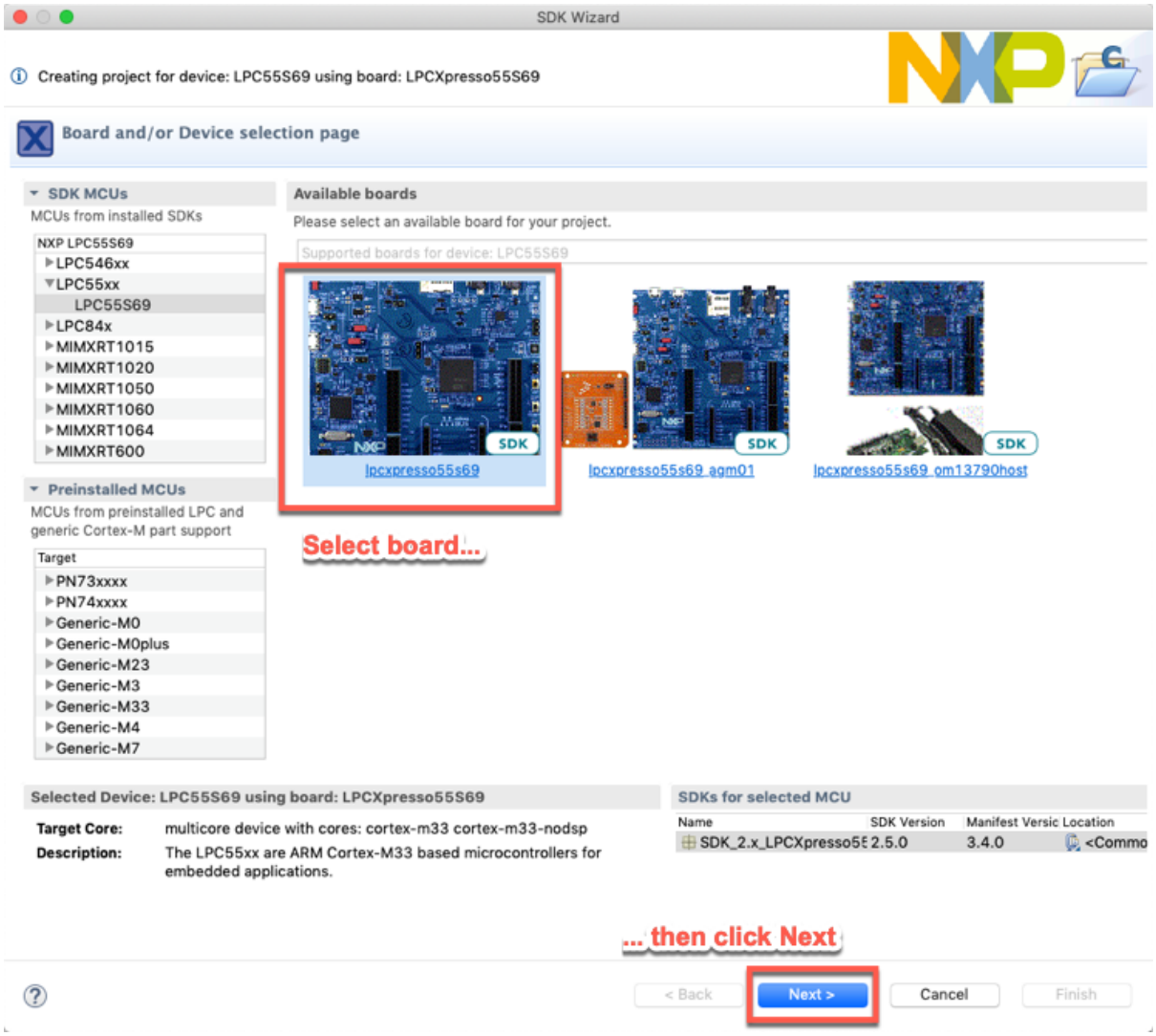
Creating a simple multicore application

A multicore application in MCUXpresso IDE consists of two linked projects - one project containing the slave code, and the other project containing the master code. The 'Master' project will also contain a link to the 'Slave' project which will cause the output image from the 'Slave' to be included into the 'Master' image. The following steps detail how such a pair of projects can be created and linked.

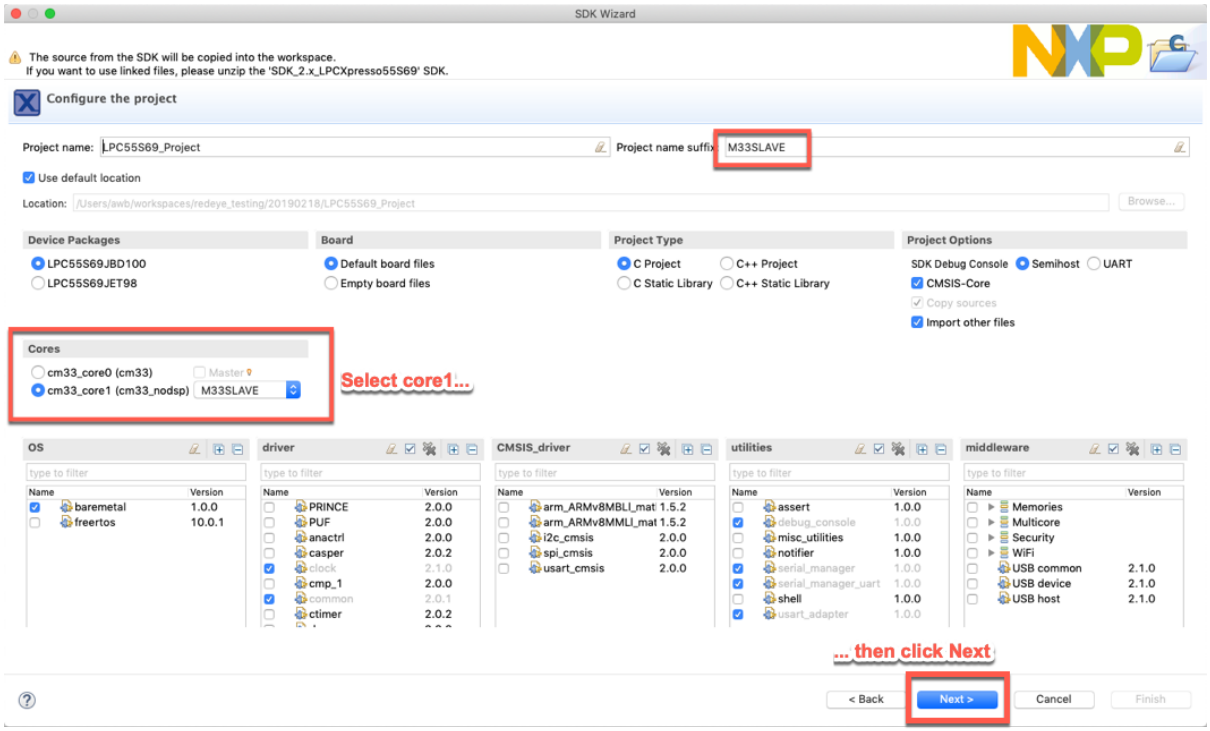
Step 1 - Create the slave project

The recommended way to create an new multicore application is to start by creating the project for the slave core first, then create the project for the master core afterwards (as you can then include the reference to the slave project as part of the master project creation process.

To begin, go to Quickstart -> New project and select “lpcxpresso55s69” in the Available Boards lists. Then click Next.

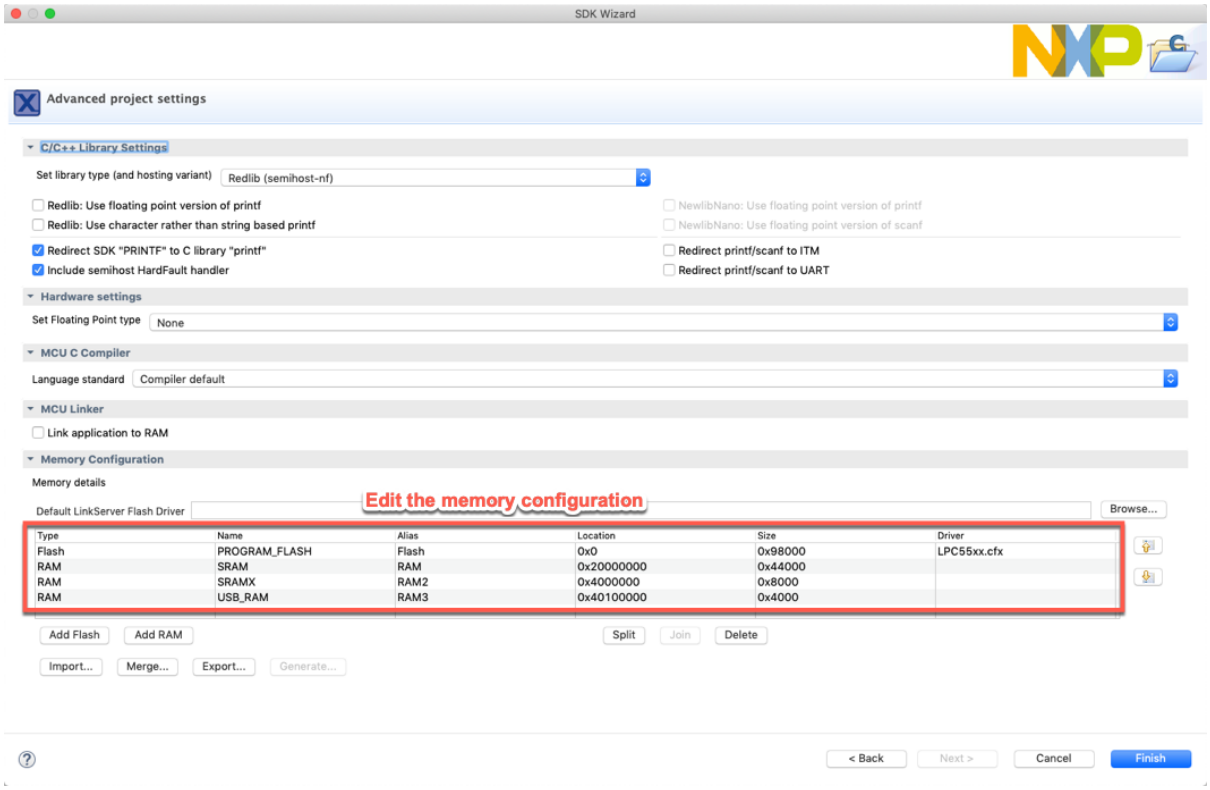


Now select core1 – which is the slave core in the MCU. Then, if required, you can change the default name of the project. Note that the wizard will automatically add a suffix “M33SLAVE” to the project name, though this can be deleted if you wish.



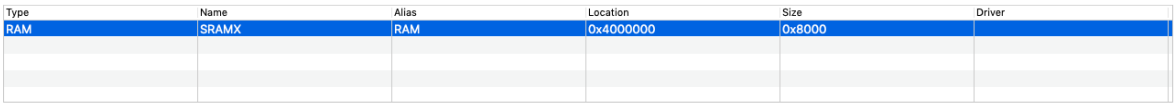
Now click “Next” to move onto the Advanced Project settings page. Here you need to modify the default memory map to specify memory that will only be used by the Slave core.

Note: the Slave code will run from RAM and must not conflict with memory used by the Master project.



- Highlight the “PROGRAM_FLASH / Flash” line in the table and click on the 'Delete' button.
- Highlight the “SRAM / RAM” line in the table and click on the 'Delete' button.
- Highlight the "USB_RAM/RAM3" line in the table and click on the 'Delete' button.

This should leave just the “SRAMX/RAM2” line (note that “RAM2” alias will update to “RAM” as you delete the other memory regions). This sets the memory block that the Slave image will be executed from:



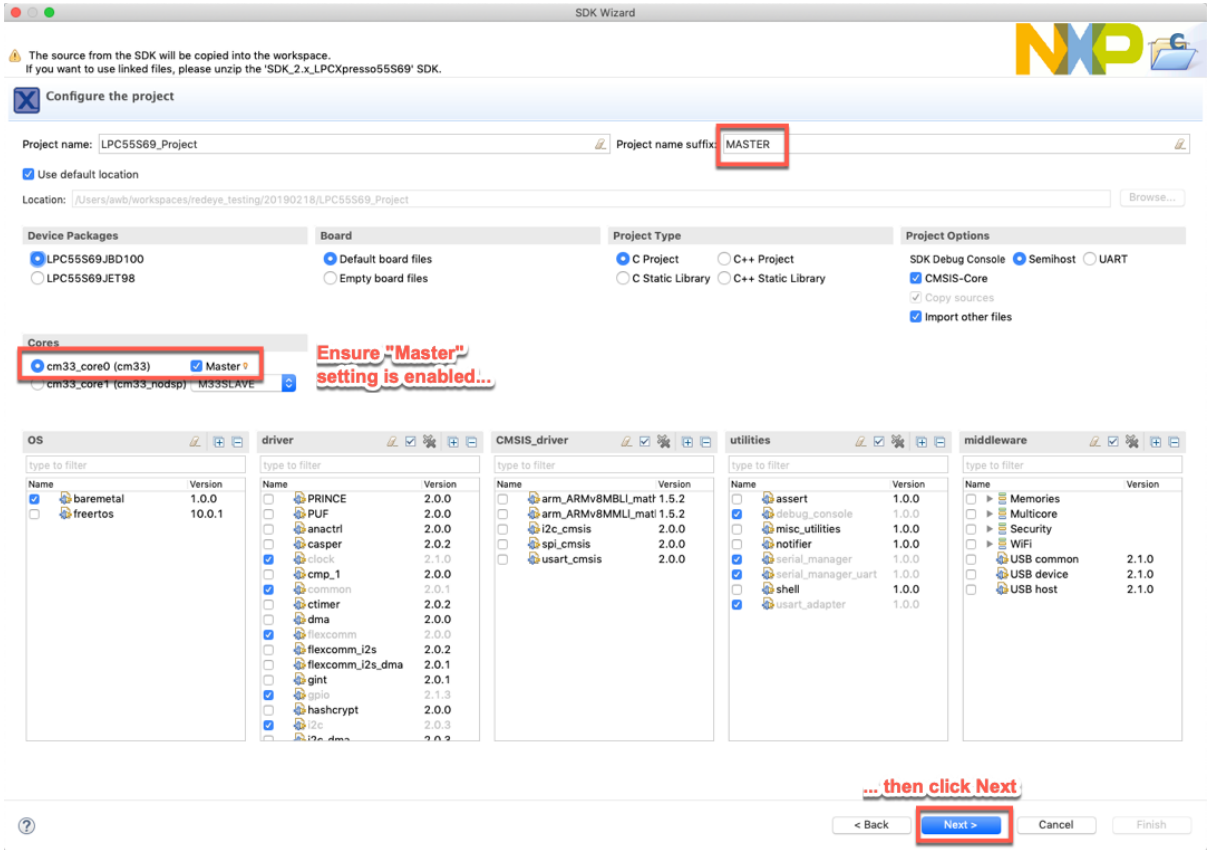
Now click “Finish” and your Slave project will be created.

Step 2 - Create the master project

Having created the Slave project, we can now create the Master project

Go to Quickstart -> New project again, and ensure that “lpcxpresso55s69” is selected in the Available Boards lists. Then click Next.

Now in the core selection, leave the setting as core0 – but ensure that the “Master” setting is enabled (ticked).

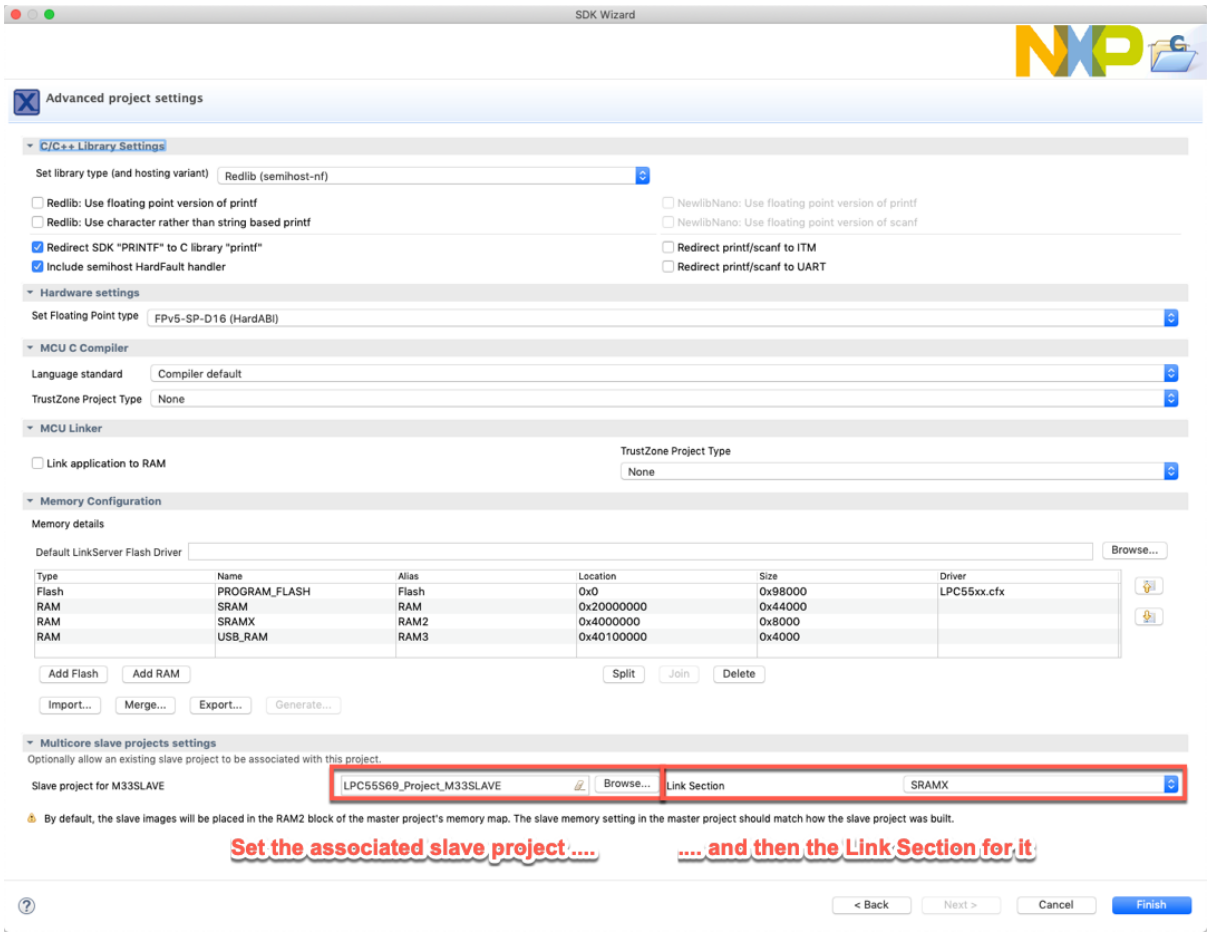


Then, if required, change the default name of the project. Note that the wizard will automatically add a suffix “MASTER” to the project name, though this can be deleted if you wish.

Now click “Next” to move onto the Advanced Project settings page. Here you need to select the Slave project that will be associated with this Master project.

Scroll down the page if necessary, and use the Browse button to select the Slave project to be used from the set of projects already in the workspace. Then choose the RAM bank that the Slave project is configured to use in the “Link Section” – in our case it was “SRAMX” (though this is the same as the alias “RAM2” in the Master project memory configuration.)

Note that it is important to make sure the Link Section is correctly set so that the Master project “sees” the Slave project at exactly the same memory address that the Slave project was configured to use.



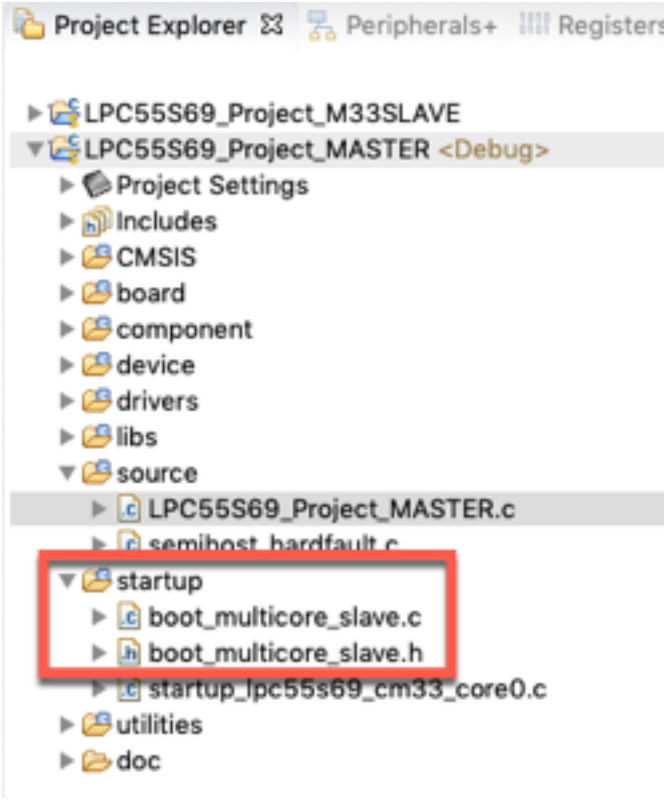
Now click “Finish” and your Master project will be created.

Notes about Multicore project contents and configuration

There are a few points to note about the projects that have been created by the above steps. First of all, the main() routine in the Master project contains a call to a function, boot_multicore_slave() which will release the Slave core from reset when executed:

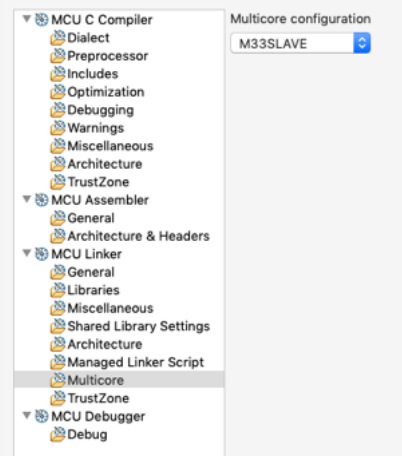
```
50 int main(void) {
51
52     /* Init board hardware. */
53     BOARD_InitBootPins();
54     BOARD_InitBootClocks();
55     BOARD_InitBootPeripherals();
56     /* Init FSL debug console. */
57     BOARD_InitDebugConsole();
58
59     /* Start slave CPU. */
60     boot_multicore_slave();
61
62     PRINTF("Hello World from MASTER\n");
63
```

This function is implemented in the corresponding header and source file that are to be found in the “startup” folder of the Master project:

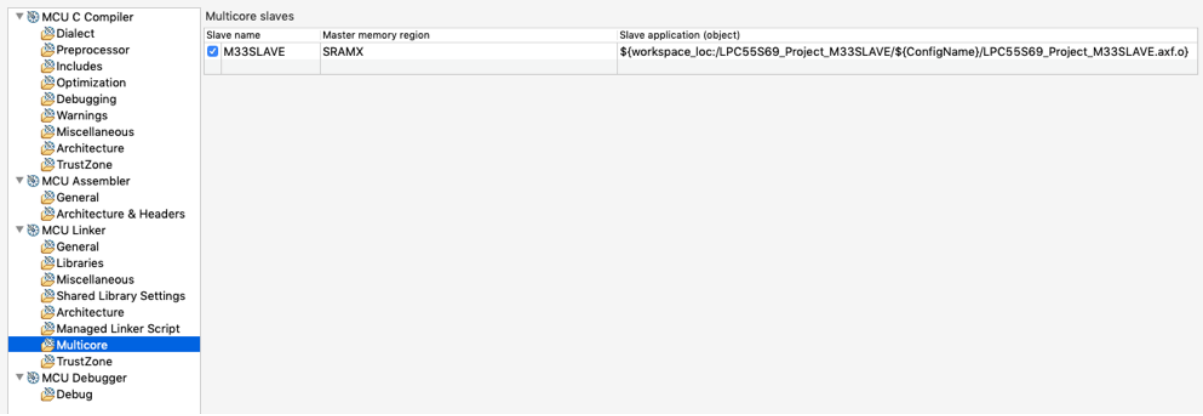


Note that these two files will actually be found in all LPC55xx projects. However the boot code will be conditionally compiled only for MASTER projects (using the symbol “__MULTICORE_MASTER”). And even when compiled in, unused section elimination will remove them from the final image if the boot_multicore_slave() function is not actually called.

For projects compiled as multicore ones, there are also additional configurations to be found in the Project Settings. For a slave project these simply sets the project to be a slave of the appropriate type:



For the Master project, the project settings will actual contain the linkage to the corresponding Slave project:



Building a multicore application

The simplest way to build a multicore project is to trigger a build of the Master project, typically using the "Build" option in the Quickstart Panel. As the Master and Slave projects are linked, this will trigger a build of the Slave project first, followed by the build of the Master project – which will embed a copy of the image from the Slave project into the image generated by the build of the Master project.

If, when building the Master project, you see a linker error of the form:

ld: M33SLAVE execute address differs from address provided in source image

this indicates that the memory configuration for the Slave image is incorrectly set in either the Slave project settings or the Master project settings. Check the settings in the Memory Configuration Editor for both projects, and also ensure that the "Master memory region" setting in the Master project's "Multicore" setting specifies the correct memory region.

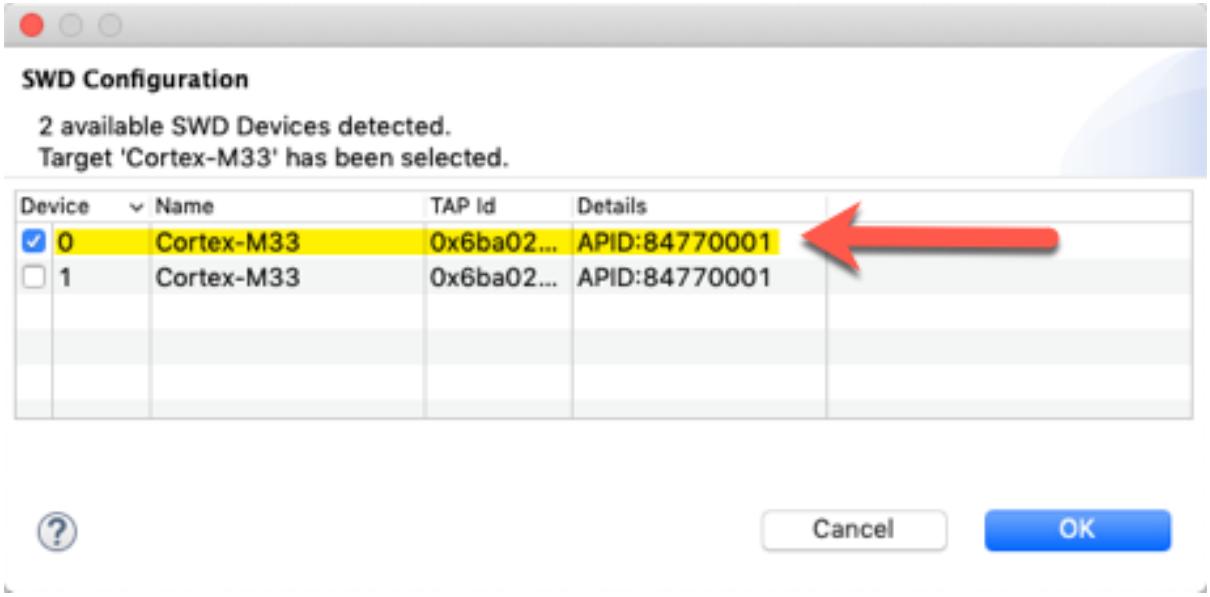
Debugging a multicore application

Having built your simple multicore application, you can now debug it.

Note : In the debug example below we are going to use the on-board "LPC-Link2" debug probe on the LPCXpresso55S69 board. However, the details are basically the same if using a standalone "LPC-Link2" debug probe. We are not going to cover the use of a P&E Micro or SEGGER J-Link probe in this article. General information on using these probes for multicore debugging can be found in the MCUXpresso IDE v10.3 User Guide, but you may also need to update the version of the P&E Micro plugin or SEGGER J-Link firmware that you have installed.

First of all, start a debug session for Master project using the Debug option in the Quickstart Panel. If this is the first time you have debugged the project, assuming your board's USB "Debug link" port is connected to your computer, a dialog should be displayed giving the details of your LPC-Link2. Ensure this is selected, then click OK.

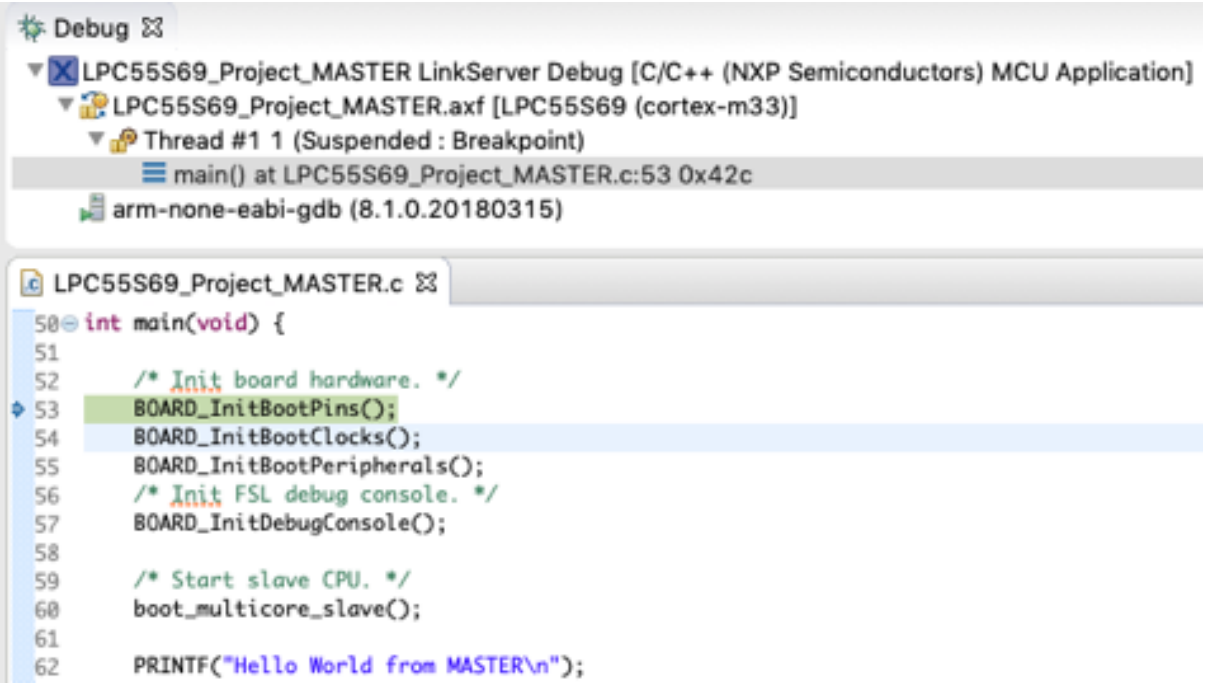
As the LPC55xx is a multicore system, we need to ensure that each project is debugged on the correct core. This is done via the "SWD Configuration" dialog. When this is displayed, select the Device (core) to connect to, which in the case of the Master project is "0", then click OK:



Note that this information is then stored within the project for use in future debug sessions, meaning that the next time you debug, you will normally not need to make this selection again. However you can reset things if necessary by deleting the ".swd" file found in the build configuration directory of your project (so normally "Debug" or "Release").

Having selected the appropriate core, MCUXpresso IDE should download the executable containing both the Master and Slave images into the target MCU flash memory.

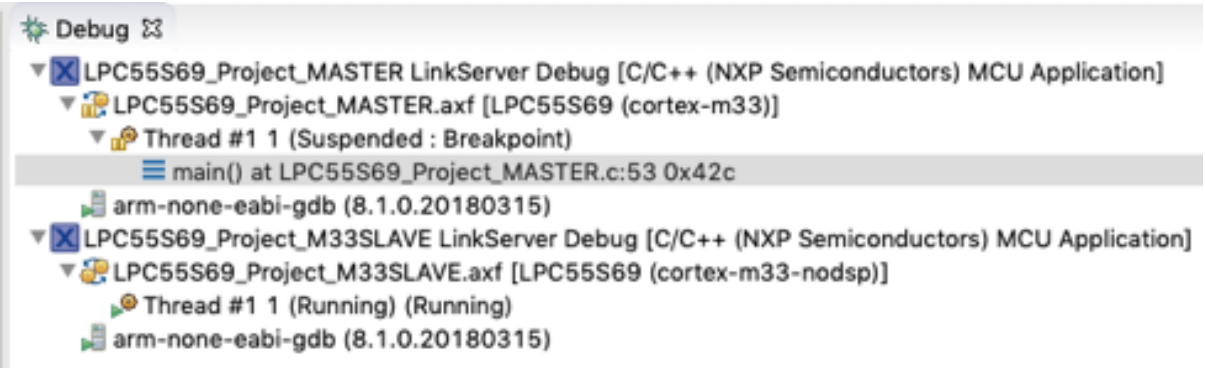
Once the connection has been made, you should hit the default breakpoint at the start of main() in the Master executable :



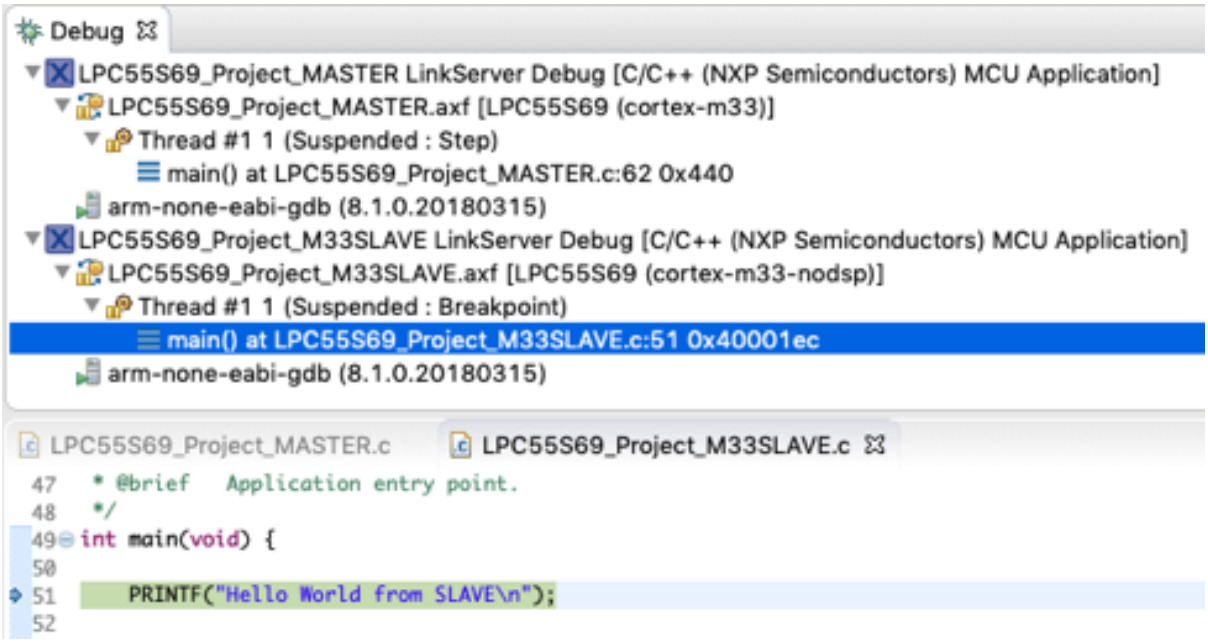
At this point, select the Slave project in the Project Explorer view, then use the Quickstart Panel's Debug option to start a second debug connection.

If prompted, select SWD target to connect to – which for the Slave core should be Device "1". However, as long as you have already started the Master debug session, then the Slave device should be selected automatically.

MCUXpresso IDE should detect that this is a Slave project and make an 'attach only' connection (and hence it will not download code – since this was done by the Master project – and leave the slave image 'executing'):



Now step through the code in the Master project until you reach the boot_multicore_slave() function. After stepping over this call, the Slave core will begin to execute, hitting the default breakpoint on it's main() function:



You can now debug both cores in parallel, selecting which core to step, resume, suspend, etc. by switching between them in the Debug View.

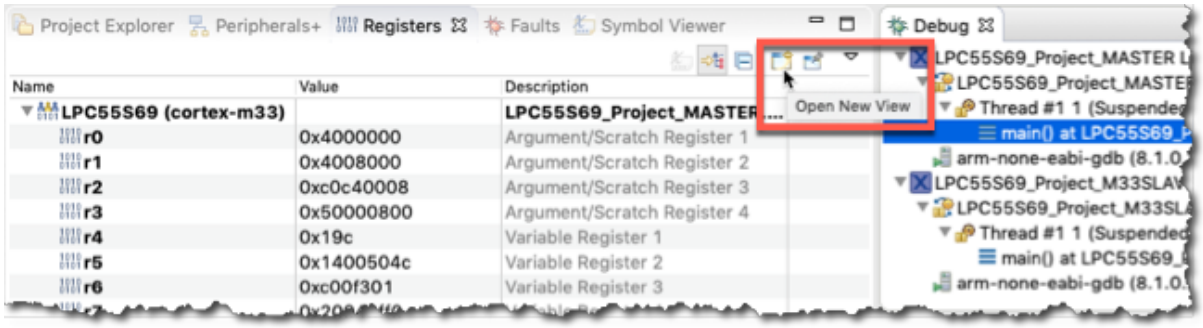
You can select both applications at the same time in the Debug View (typically by using CTRL-Click) and operations like step, resume, suspend, etc will then be carried out on both cores by the debugger in parallel.

You can also more easily carry out debug operations on both cores at once by using the "Resume All", "Pause All", "Step into All", etc buttons in the main IDE toolbar:

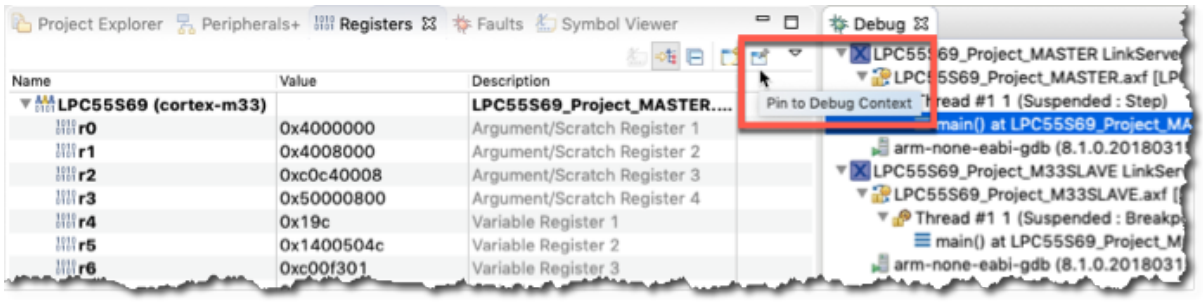
Note that these "All" operations are implemented via software, not by using any hardware debug logic synchronising the debug operations on the two cores.

The currently selected core in the Debug View will be the one used for displaying many of the debug related views, such as Registers and Locals etc. It is also possible to create copies of many of the debug related views and lock each copy to a particular core.

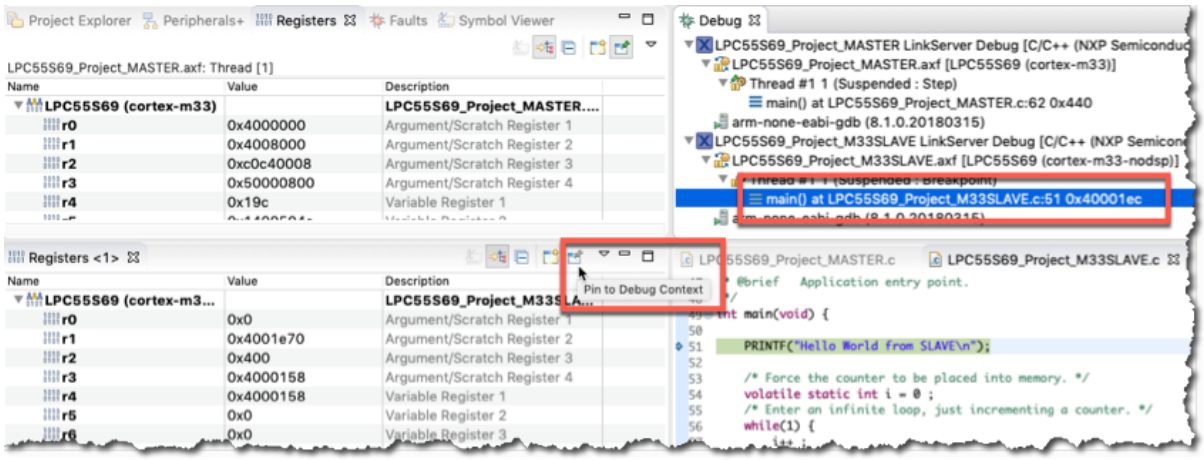
For example, to create two register views, one for the Master CM33 and one for the Slave CM33 …First of all, use the "Open New view" button in the Registers view to create a second Registers view:



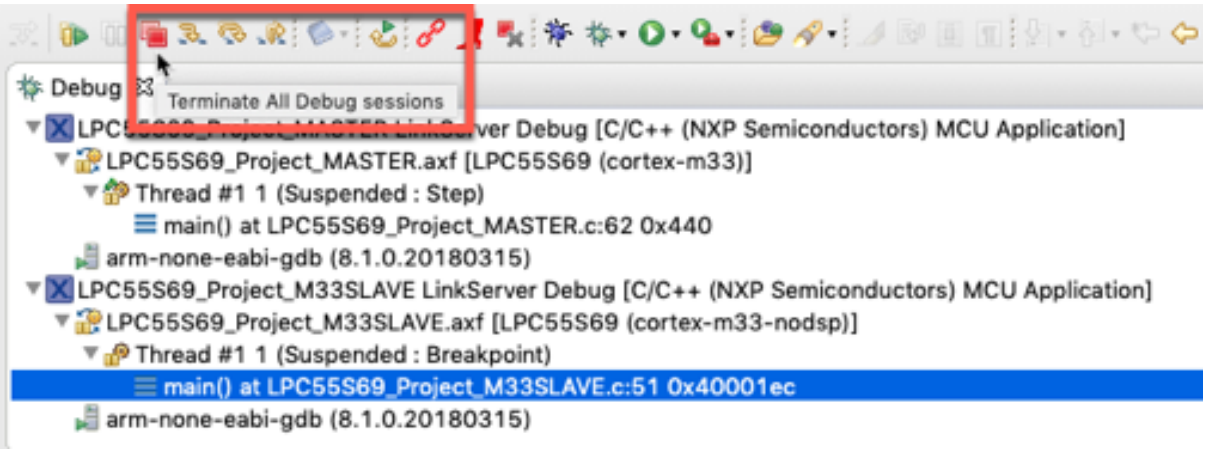
Now pin the original view to the core currently selected in the Debug, using the “Pin to Debug context” button :



Now select the other core in the Debug view, and go to the second Register view. Use this view’s “Pin to Debug Context” button to lock this second Registers view to the currently selected core:



Note that when you have finished debugging, you can use “Terminate/Disconnect All” button on the icon bar to disconnect the debug connections to both the Master and Slave cores at the same time:



SDK Example multicore applications

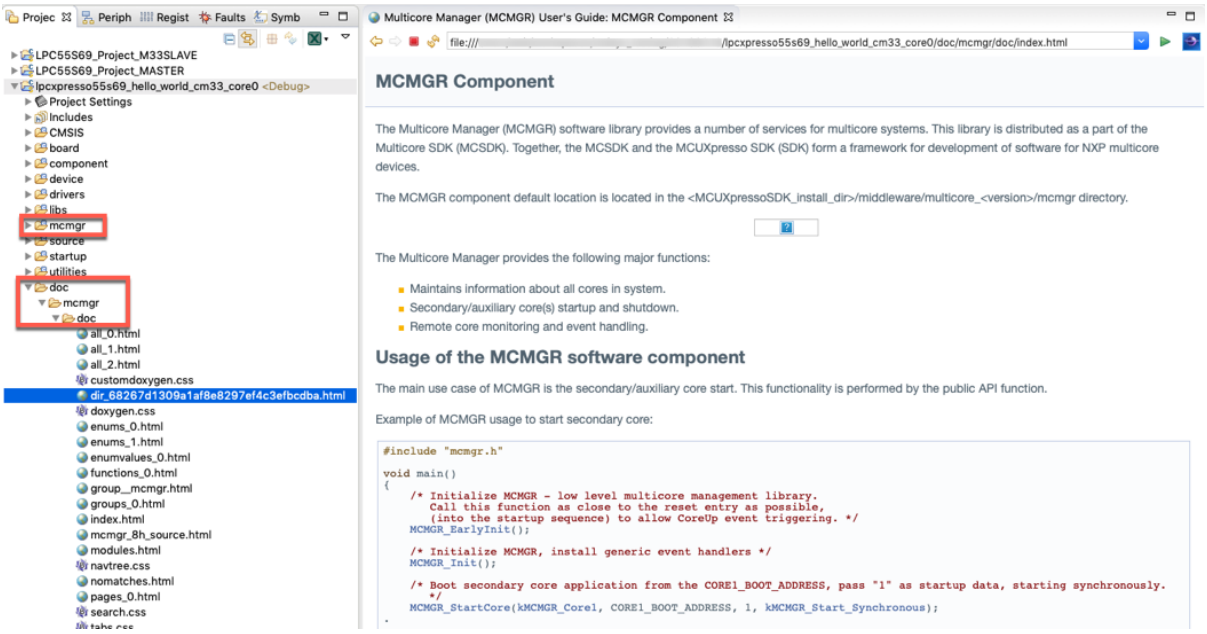
The MCUXpresso SDK for LPC55xx contains a set of example multicore examples, each consisting of two linked projects – with "core0" indicating the Master project and "core1" the slave project.

These can be imported into your IDE workspace via the QuickStart Panel's "Import SDK example(s)…" option :

Once imported, these can be built and debugged as per the simple Master and Slave example created using the IDE's New Project Wizard.

One important difference to note though is the SDK supplied multicore example use a much more sophisticated way of booting the slave core – using the "MCMGR" or "Multicore Manager" component. This component also provides additional functionality, such as

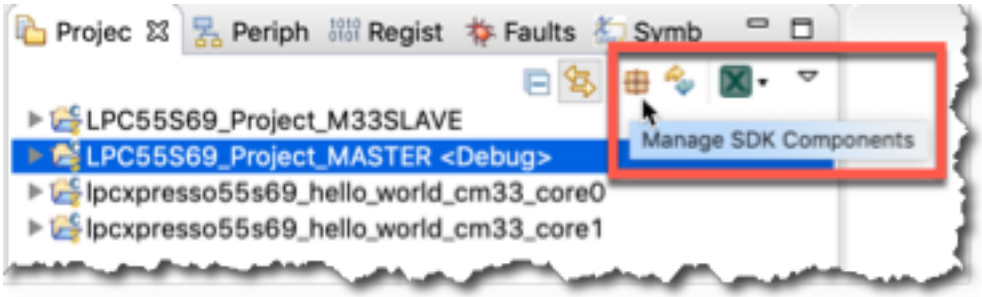
providing synchronization between the Master and Slave cores. For more information, please see the documentation provided by the SDK on the MCMGR component:



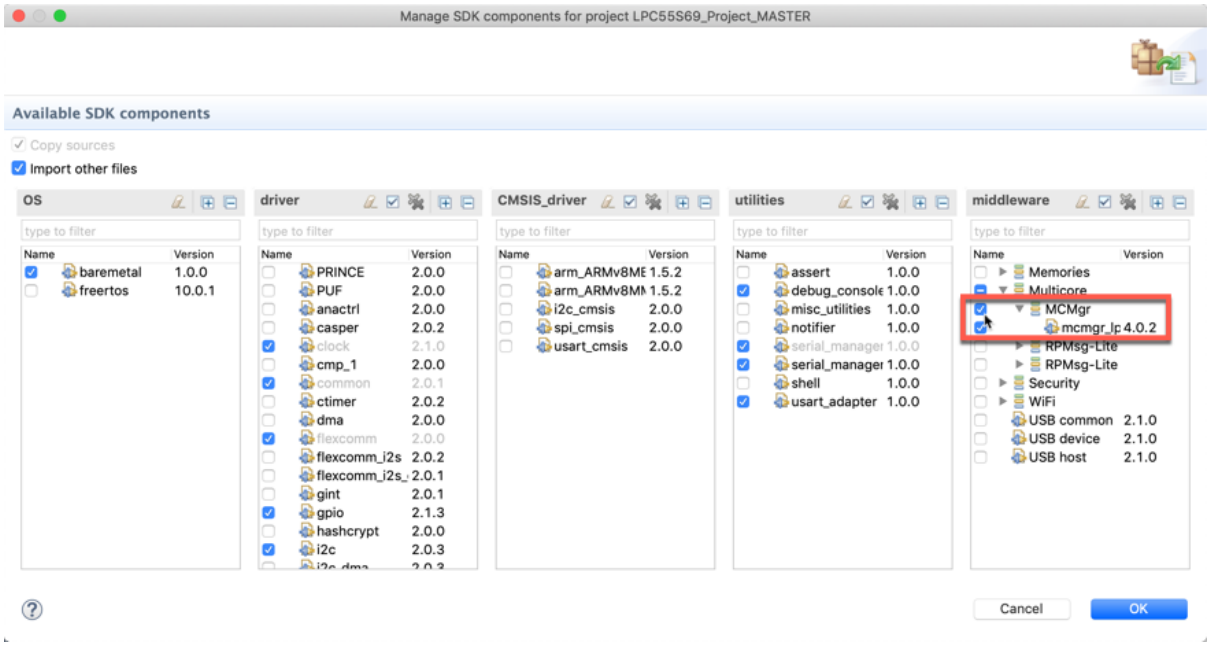
Converting a simple multicore application to use MCMGR

If required, it is relatively straightforward to convert a multicore application created using the IDE's New Project Wizard to boot the slave using the MCMGR rather than the simple boot_multicore_slave() function. This also makes available to your application the additional functionality that MCMGR can provide beyond slave booting.

First select the Master project and click on the "Manage SDK Components" button to launch the wizard.

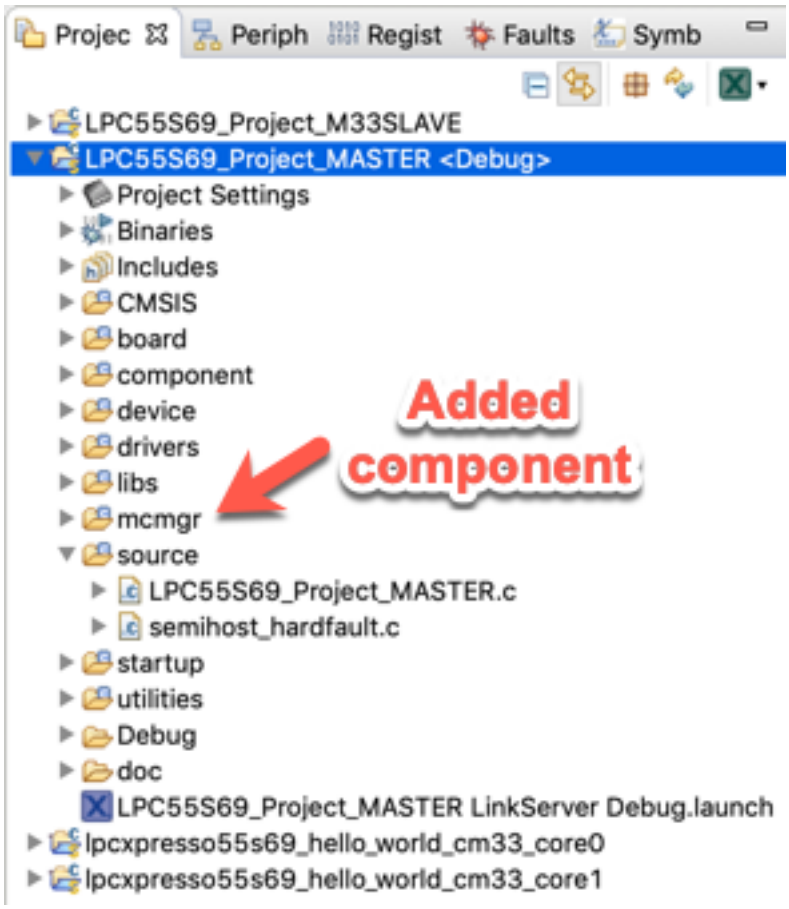


Within the middleware section of the wizard, select the Multicore->MCMgr component.



Click on OK, then on “Yes” to confirm the updating of the components inside your project.

If you now open up your project in the Project Explorer, you will see the MCMGR component has been added:



Now open the main C file in your project, in this case LPC55S69_Project_Master.c

Now replace the include of the simple slave boot code:

```
#include "boot_multicore_slave.h"
```

with the inclusion of the MCMGR component header:

```
#include "mcmgr.h"
```

Then delete the call to the slave boot function:

```
    boot_multicore_slave();
```

You then need to add appropriate calls to the MCMGR functions. A simple conversion can be seen below:

Relevant code from the above screenshot for copy / pasting into your own application can be found below:

```
#include "mcmgr.h"
```

```
void SystemInitHook(void)
{
    MCMGR_EarlyInit();
}


    MCMGR_Init();


    MCMGR_StartCore(kMCMGR_Core1, (void *)0x4000000, 5,
                    kMCMGR_Start_Asynchronous);
```

For more details on using MCMGR, please see the LPC55xx SDK multicore examples, along with the SDK documentation.