Title:	Flashtec NVMe3016 NVM Controller Firmware		
Abstract:	NVM Controller Firmware Quick Start Guide		
Marketing No.:	PM8627	**Revision Date:**	November 2018
Document No.:	ESC-2171536		

Detailed Revision History

Revision	Revision Date	Prepared By	Details of Change
5.0	October 2018	John Lloyd	Updated the firmware compilation description.
4.0	July 2018	John Lloyd	Updated with details about new firmware make procedure.
3.0	March 2018	Frederick Adi	Updated with dual port usage description.
2.01	March 2018	John Lloyd	Updated firmware compilation description.
2.0	December 2017	Frederick Adi	Added Microcode debug. Change toolset to DS5.27.1 and GCC 4.8
1.0	September 2017	Balan Rajendran and Vinoj Soundararajan	Initial Revision

PM8627 Quick Start Guide Flashtec NVMe3016 NVM Controller Firmware

Revision History : November 2018

Revision	Revision Date	Details of Change
5.0	October 2018	Updated the firmware compilation description.
4.0	July 2018	Updated with details about new firmware make procedure.
3.0	March 2018	Updated with dual port usage description.
2.01	March 2018	Updated firmware compilation description.
2.0	December 2017	Added Microcode debug. Change toolset to DS5.27.1 and GCC 4.8
1.0	September 2017	Initial Revision

Table of Contents

Figures

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision 6.0

Revision 5.0

Revision 5.0 of this document was published in October 2018. The following is a summary of the changes.

Updated the firmware compilation description.

Revision 4.0

Revision 4.0 of this document was published in July 2018. The following is a summary of the changes.

Updated with details about new firmware make procedure.

Revision 3.0

Revision 3.0 of this document was published in March 2018. The following is a summary of the changes.

Updated firmware compilation description.

Updated with dual port usage description.

Revision 2.0

Revision 2.0 of this document was published in December 2017. The following is a summary of the changes.

Added Microcode debug. Change toolset to DS5.27.1 and GCC 4.8

Revision 1.0

Revision 1.0 is a preliminary release of this document published in July 2017.

2 Introduction

This document serves as a Quick Start Guide for the NVMe based Enterprise Flash Controller (EFC) firmware. Before you begin, obtain the EFC firmware, development tools, and architectural simulator packages listed below:

- "PM71_85_270_sdk_x.y.z.tgz" – Package from *myPMC* which includes:
  - Quick Start Guide
  - Release Notes
  - Firmware Source Code
  - Microcode Source Code
- "PM71_85_270_simulator_x.y.z.tgz" – Package from *myPMC* which includes:
  - Bsim Users Guide
  - Quick Start Guide
  - ArchSim package
  - HostSim package which includes the Block Layer Simulator(also known as bsim)
  - Uartsim
  - Release notes
- "RG-2017.8_revA.zip" package from

  "RG-2017.8" package from

  myPMC which includes:
  - Xplorer-7.0.8-linux-installer.bin
  - Belmar_fc_rev6_tie11_PROD_linux_redist.tgz
  - Xplorer-7.0.8-linux-installer.bin
  - Morristown_rev0_tie11_redist.tgz
- DS-5 and FastModel – Package from ARM which includes:
  - DS-5 Development Studio version 5.28.1
  - FastModel version 10.2
  - FastModel Third Party IP 10.2 for SystemC
- Argtable2 – Package from SourceForge *argtable2-13*

3 Installation

This section describes hints and tips related to developing under Linux, in addition to the development tools, Architecture Simulator and firmware installation procedures.

Linux Tips

The development tools are supported on CentOS (RHEL) 6.9 64-bit. It is recommended to install CentOS as **"Software Development Workstation"** and this guide was written with the assumption that the CentOS was installed as **"Software Development Workstation"**. After installation of CentOS 6.9, do not perform any updates to the OS, otherwise the attempt to install additional packages required by the firmware build utilities and build environment mentioned in the following will not succeed.

> **Note:** For any other CentOS version installation, additional packages may need to be installed.

ARM development tools require gcc 4.8, but the default gcc version is 4.4 for CentOS 6.

The following additional packages are required by the firmware build utilities and build environment:

- `yum install glibc-2.12-1.212.el6.i686`
- `yum install libgcc-4.4.7-18.el6.i686`
- `yum install libstdc++-4.4.7-18.el6.i686`
- `yum install zlib-1.2.3-29.el6.i686`
- `yum install glibc-devel.i686`
- `yum install pciutils-devel.i686`
- `yum install gtk2.i686`
- `yum install gtk2-devel.i686`
- `yum install libXtst.i686`

> **Note:** Root user access privileges may be required for installation of the above packages.

Centos.org has built the "*redhat developer toolset 2*" that contains gcc 4.8.2. Use this repository to install gcc 4.8 as shown in the following example.

- `wget http://people.centos.org/tru/devtools-2/devtools-2.repo -O /etc/yum.repos.d/devtools-2.repo`
- `yum install devtoolset-2-gcc devtoolset-2-binutils devtoolset-2-gcc-c++`

> **Note:** In order for `yum` to work correctly the proxy settings in `/etc/yum.conf` must be correct.

Development Tools

The version numbers of the tools mentioned in the following are tested and installation of any other version of the tools may lead to undesirable results.

- DS-5 Ultimate Edition

  Download and install the DS-5 development studio v5.28.1 from *https://developer.arm.com/products/software-development-tools/ds-5-development-studio/downloads*

  > **Note:** Different versions of DS-5 other than v5.28.1 are not compatible with the required versions of FastModels 10.2 with GCC 4.8.

  > **Note:** Warnings on missing 32-bit library dependencies are ignored during the installation, as they don't affect DS-5 normal operation. For more information, see *https://static.docs.arm.com/100950/0527/getting_started_guide_100950_0527_00_en.pdf*.

- Xtensa tools

  Download the latest version of the development tools package from Cadence, that is Xplorer-7.0.8-linux-installer.bin

  Download the latest version of the development tools package from Cadence, that is Xplorer-7.0.8-linux-installer.bin

  Installing the development tools is a multi-step process, described below:

  1. Xtensa Xplorer
     - To install Xtensa Xplorer and the Xtensa development tools, execute `Xplorer-<version>-linux-installer.bin`.
  2. XOCD
     - XOCD is the Xtensa On-chip Debug Daemon that is required for hardware debugging. The latest version of XOCD should be downloaded from Cadence.
  3. Configuring your environment
     - Xtensa Xplorer licenses are checked automatically as needed. If no licenses are found, Xplorer prompts you to configure licenses. See section 4.5 of the *Xtensa Development Tools Installation Guide* (dev_tools_install_guide.pdf) for more information.
  4. Once installed and started Xplorer will automatically check for updates. Install any updates that are automatically detected.
  5. There is a core configuration needed to be installed for microcode. These values will be used by the current archsim makefile
     - Belmar_fc_rev6_tie11_PROD_linux_redist.tgz
     - Morristown_rev0_tie11

- Xtensa core configuration install

  Belmar_fc_rev6_tie11_PROD_linux_redist.tgz**Morristown_rev0_tie11_redist.tgz** is the file containing the Flash channel core configuration.

  The following steps describe how to install the core configuration using **xplorer**.

1. Look for **System Overview** window. If it does not exist, open **Window** in the menu tab. Then open **Show View**, and lastly, select **System Overview**.

Figure 3-1 • System Overview Menu

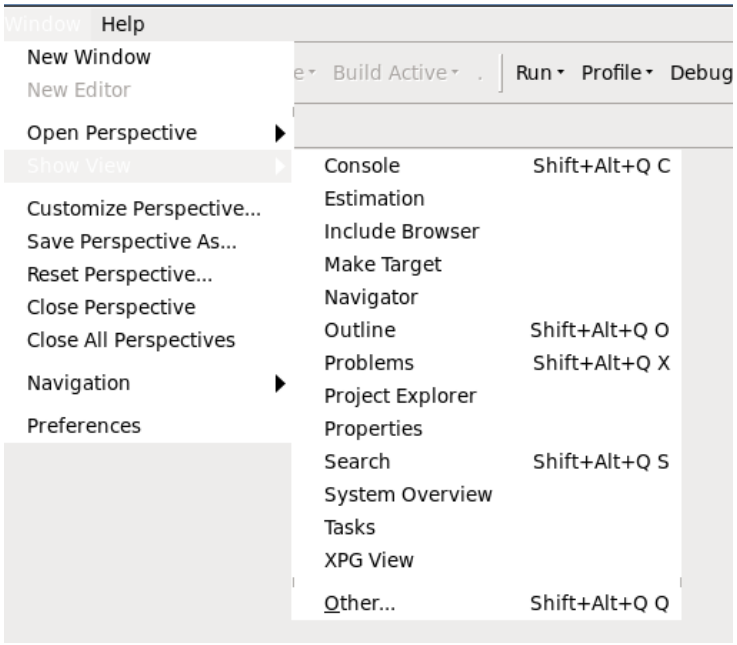
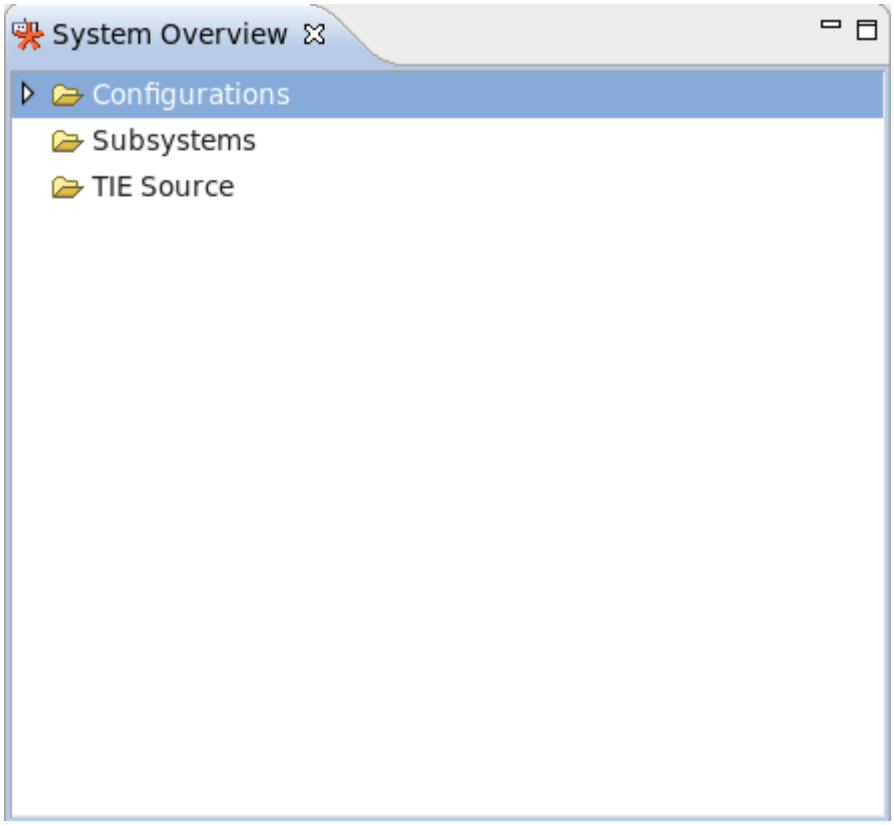
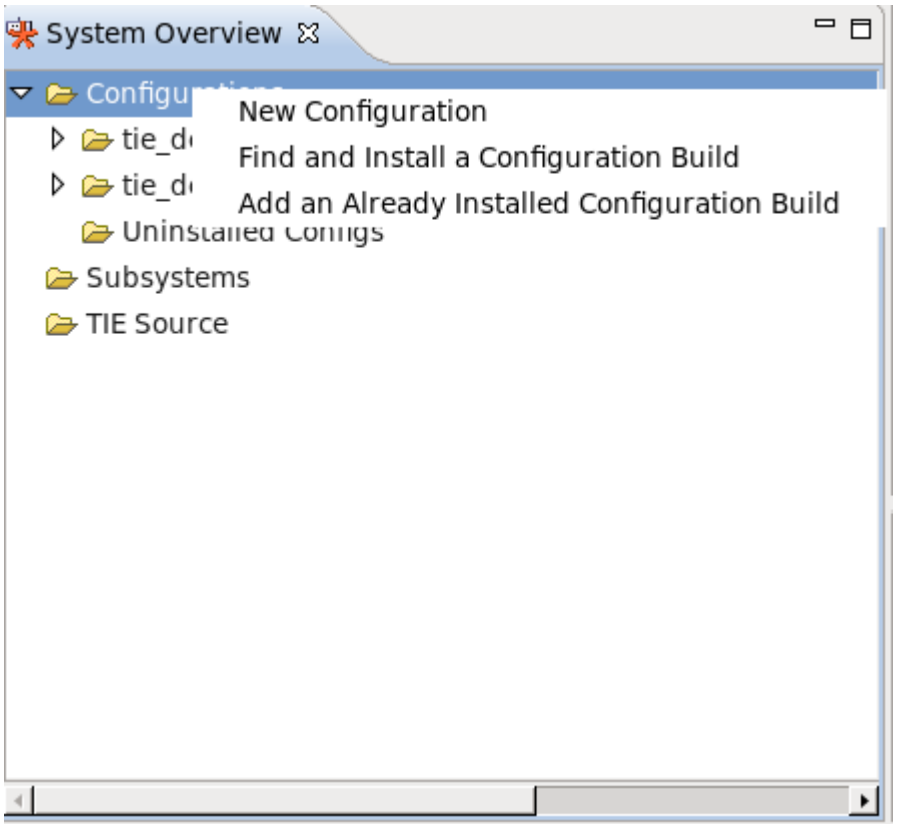


Figure 3-2 • System Overview Window



2. In the **System Overview** window, select **Configurations** and right-click. From the new optiond displayed, select **Find and Install a Configuration Build**.

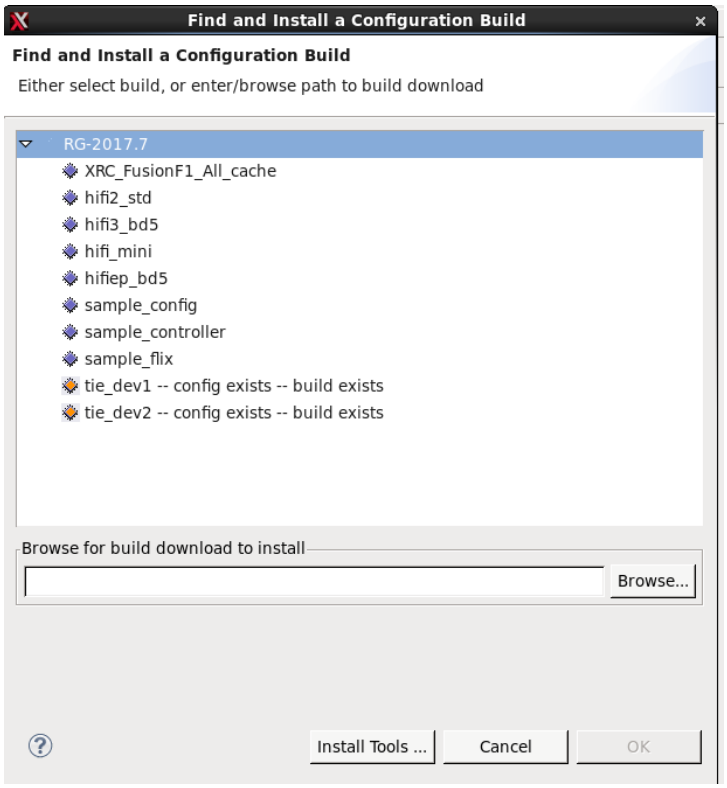
Figure 3-3 • System Overview Configuration Option



3. The following figure shows the currently installed configurations under the **RG-2017.8RG-2017.8** tool that came installed with **Xplorer-7.0.8Xplorer-7.0.8**. Select **Browse** to look for flash channel

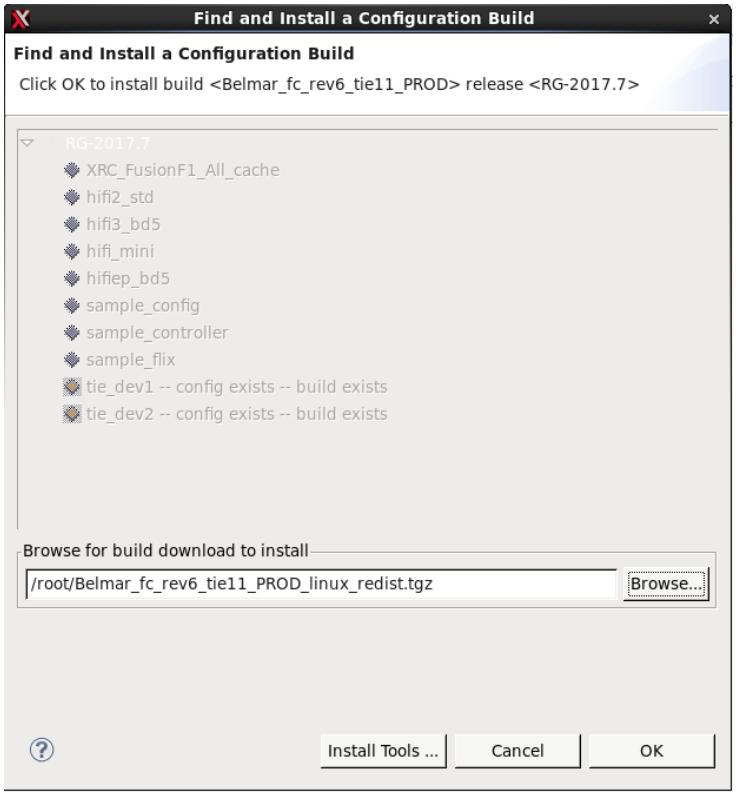
core configuration, for example, **Belmar_fc_rev6_tie11_PROD_linux_redist.tgzMorristown_rev0_tie11_redist.tgz**.

Figure 3-4 • Find and Install a Configuration Build



After finding the correct core configuration file, click **OK** to install.

Figure 3-5 • Install core configuration



4. The following figure shows the System Overview window with the newly installed flash channel core configuration.

Figure 3-6 • Core Configuration Installation Complete

- FastModel

  Download and install the FastModels 10.2 and the third party package. The download link for the FastModels package is available upon request from ARM (*ARM support*), and requires a license to build and run.

  Note: It is recommended to install FastModels from ARM and the third party package in the same path.

- Licensing

  The following licenses are required from ARM for running an instance of the architectural simulator:

  - Required to Build and Debug Firmware

ARM DS-5 Ultimate Edition License (Number of Licenses Required: 1)

Refer to "ARM DS-5 License Management Guide" at *https://developer.arm.com/products/software-development-tools/ds-5-development-studio/docs* to get information on how to Request, Configure and Install the license.

Note: Licenses for the DS-5 tools should be obtained directly from ARM.

- Required to Build the FastModels

  MaxCore_SystemGen (Number of Licenses Required: 1)

- Required to Run the Simulation

  FM_Simulator (Number of Licenses Required: 1)

  SG_v7SystemIP_CT (Number of Licenses Required: 1)

  SG_ARM_Cortex-A53_CT (Number of Licenses Required: 4 [1 per A53 cluster])

- Required to build/run microcode (FCC)

  The Xtensa development tools are licensed through FLEXlm. FLEXlm operates in two different modes: node-locked and floating (networked). The license installation procedure varies depending on whether you are using node-locked or floating licenses. To install and manage licenses for the Xtensa software, follow the instructions in section 4 of the *Xtensa Development Tools Installation Guide* (dev_tools_install_guide.pdf). The following licenses may be required:

  - XTENSA_SDK - required for compilation and debugging. Recommended to have one license per development engineer.
  - XT_ISS_XTMP - required to run the Architectural Simulator (ArchSim)
  - XT_ISS_TURBO - required to improve the speed of the Architectural Simulator (ArchSim)
  - No additional license required for on-chip debugging

  Note: Obtain licenses for the Xtensa tools directly from Cadence.

Simulator

Download the latest version of the simulator package from myPMC, that is PM71_85_270_simulator_x.y.z.tgz, and extract its contents.

The architectural simulator (ArchSim) and the host simulator (HostSim) are distributed as tar-balls, PM71_85_270_archsim_x.y.z.tgz and PM71_85_270_hostsim_x.y.z.tgz respectively, within the simulator package downloaded from myPMC. To install ArchSim and HostSim, extract the contents of the tar-balls.

Architectural Simulator (ArchSim) on page 13 explains the required environment variable settings for the simulator.

Argtable

The Block Layer Simulator (BSim) relies on the argtable2 third-party shared library. This library must be downloaded, compiled, and installed before compiling BSim.

Firmware

To install the firmware for DS-5 Development studio, open the DS-5 Development studio from `<DS-5 Directory>/bin`. For example, `/DS-5_v5.28.1/bin/eclipse`.

The firmware sources are packaged as a DS-5 workspace project. The project is included in the PM71_85_270_sdk_x.y.z.tgz package that is downloaded from myPMC.

Installation of Firmware Package for DS-5 development studio is performed in two steps:

1. Untar the PM71_85_270_sdk_x.y.z.tgz file.

2. Import the project into Workspace.

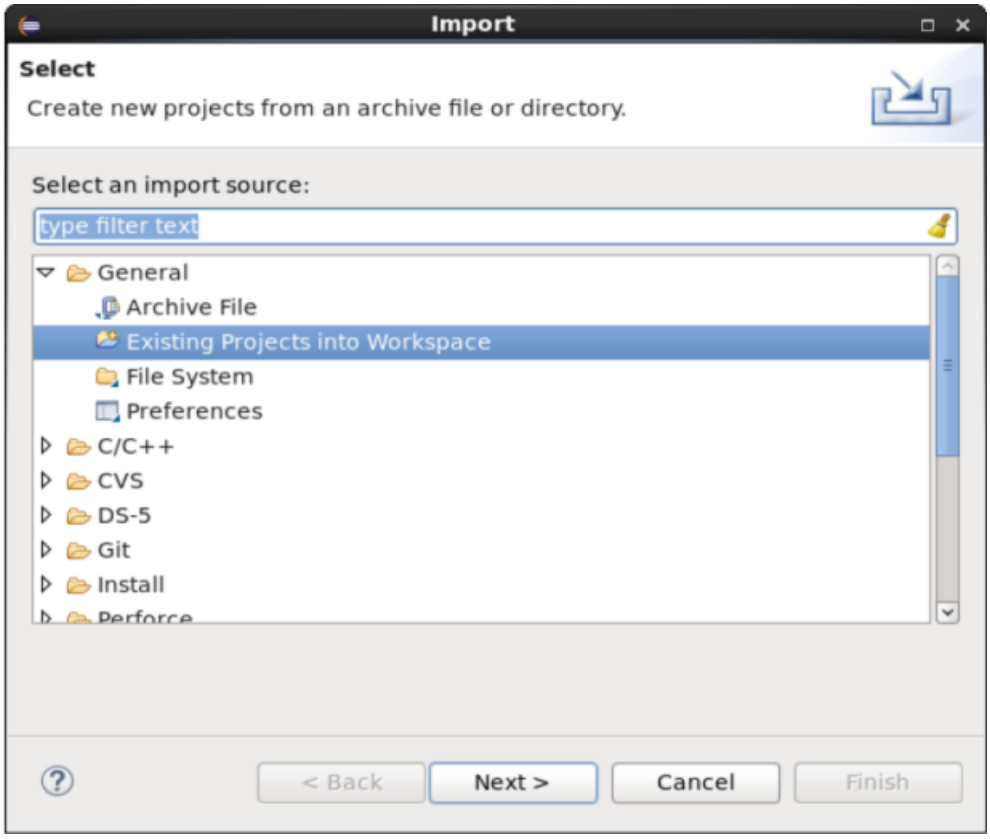
Untar the PM71_85_270_sdk_x.y.z.tgz file

Extract the PM71_85_270_sdk_x.y.z.tgz to the desired location. The contents of the firmware package are under the PM71_85_270_sdk_x.y.z folder, which serves as the firmware root directory.

Import the Project into Workspace

Importing a project will install the firmware sources. To import the project into the DS-5 workspace, select **File** and from the pull down menu click **Import**. Import Dialog appears.

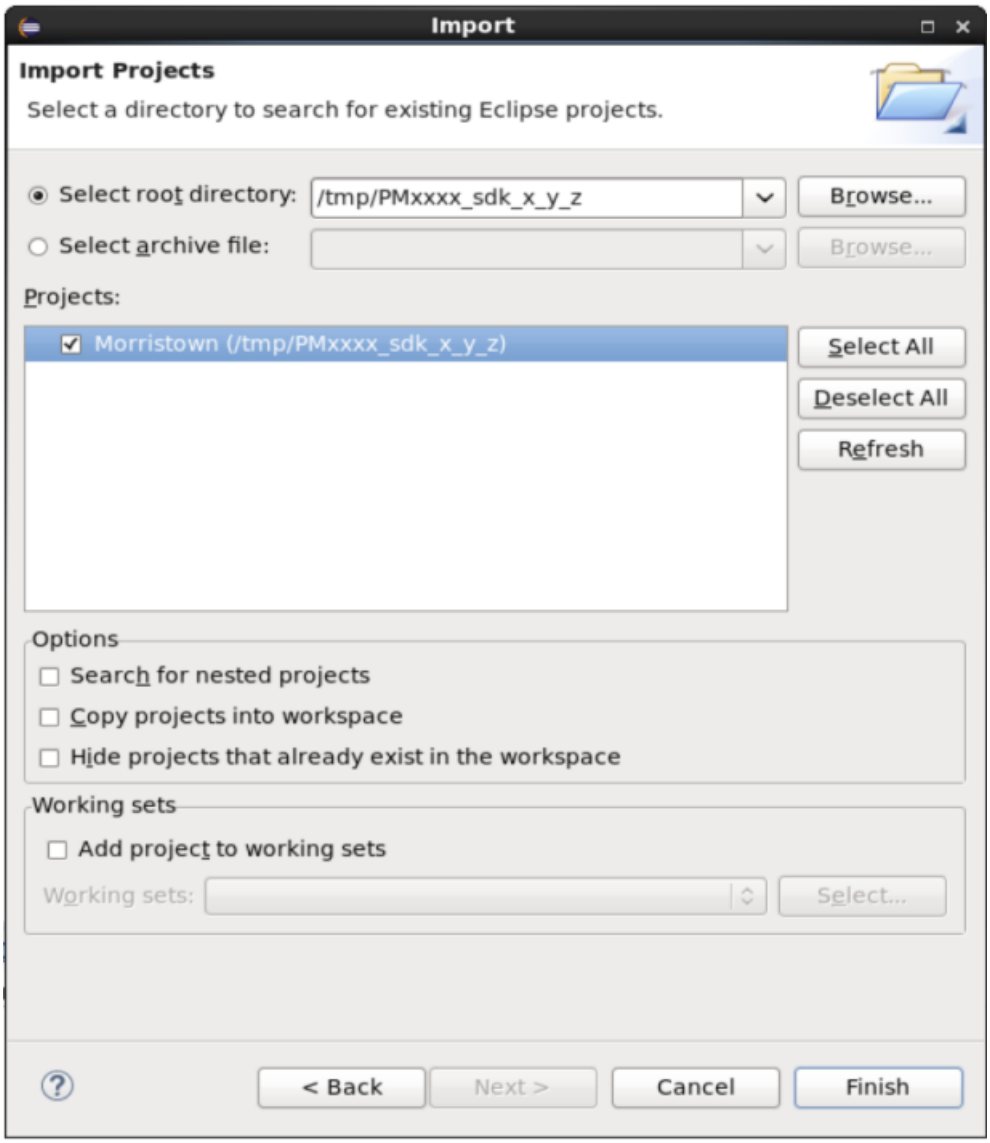
Figure 3-7 • Import Dialog



Select **General and** choose **Existing Projects into Workspace** and click **Next**.

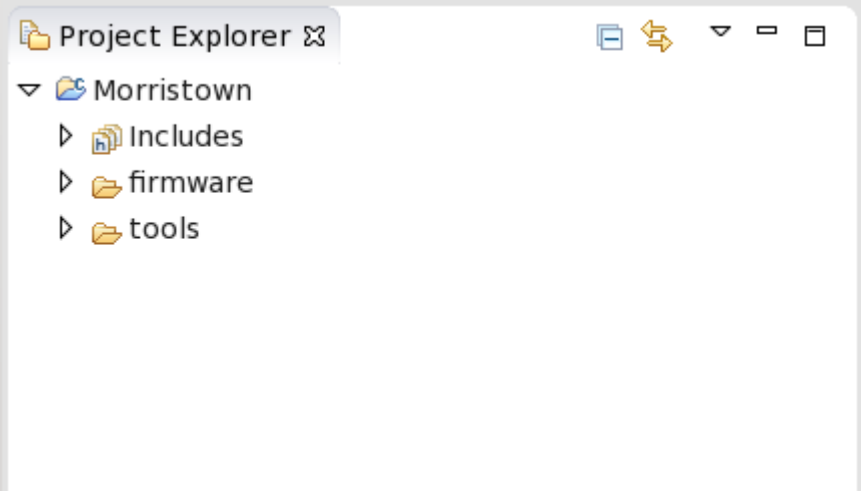
From the Import Dialog under **Select root directory**, choose the location of PM71_85_270_sdk_x.y.z folder. **Morristown** is selected by default under Projects (if not, select the **Morristown** project by clicking on the check box).

Figure 3-8 • Import Projects Dialog



Now click **Finish**.

Figure 3-9 • Imported Project



Congratulations, you have successfully imported the project into the workspace.

4 Compilation

This section describes how to build the architectural simulator (ArchSim), the host simulator library (HostSim), the block layer simulator (BSim), and the firmware.

Before you begin, set the required environment variables.

Common Environment Variables

The archsim, hostsim, and bsim build relies on header files that are located in the firmware directory structure. The EFC_FW_PATH environment variable is used by ArchSim to locate the firmware EFC directory. For example, if the firmware is located in `/home/username/eclipse/workspace/firmware`, the environment variable should be set to `/home/username/eclipse/workspace/firmware/efc`. See the example below of setting this environment variable in a C-shell.

```
setenv EFC_FW_PATH <firmware root directory>/<workspace>/firmware/efc

setenv ARGTABLE_LIB_DIR <argtable-root>/lib

setenv ARGTABLE_INC_DIR <argtable-root>/include
```

Architectural Simulator (ArchSim)

Archsim Environment Setup

From the command window, extract the ArchSim package, and go to the ArchSim installation directory (that is, `cd simulator/archsim`), then set the following environment:

> **Note:** The file path may be different based on where the tools are installed. This example shows environment setting using C shell (csh). If bash is used, then the user needs to use **export**, `<Thirdparty FastModels path>/Accellera/source_me.sh` and `<ARM FastModels path>/source_all.sh`

1. Set the path to redhat developer toolset 2.

   ```
   setenv PATH /opt/rh/devtoolset-2/root/usr/bin:${PATH}
   ```

2. Setup the license file.

   ```
   setenv ARMLMD_LICENSE_FILE <arm license file>
   ```

3. Configure Xtensa environment for RG-2017.8.

   ```
   setenv XTINSTALL <Tensilica tools directory>/xtensa/XtDevTools

   setenv XTVERSION RG-2017.8

   setenv PATH $XTINSTALL/install/tools/$XTVERSION-linux/XtensaTools/bin:$PATH

   setenv LM_LICENSE_FILE port@<xtensa license server>
   ```

4. Setup the **libstdc++.so.6** library for gcc 4.8 and higher.

   Copy libstdc++.so.6 from `<arm directory path>/DS-5_v5.28.1/sw/models/bin` to `/usr/lib64`

   This will override the currently installed **libstdc++.so.6** due to the requirement for a newer version of the library, which is backward-compatible with the original.

   It is recommended to save the original **libstdc++.so.6** by moving to a new name, for example, **libstdc++.so.6.ORIG**. Rename the file in case you need to revert to the earlier version, for example.

   The next step is to add the location where **libstdc++.so.6** is copied to; to the front of LD_LIBRARY_PATH. In our example, it is /usr/lib64.

```
setenv LD_LIBRARY_PATH /usr/lib64:${LD_LIBRARY_PATH}
```

5. Configure the FastModel environment.

   ```
   source <Thirdparty FastModels path>/Accellera/source_me.csh
   ```

   ```
   source <ARM FastModels path>/source_all.csh
   ```

   The paths to these scripts are established when you install the FastModel software. The scripts are not part of the ArchSim software package.

Build Architectural Simulator

From the `simulator/archsim` directory, execute the following commands:

1. `make build <Enter>` — To build the FastModel
2. `make cxx_cmd <Enter>` — To build the systemC

This will build the architectural simulator executable image (`A53x4_clusts4_CCN502.x`) and place it in the current directory.

Host Simulation Library (HostSim)

From the command window, go to the HostSim installation directory (i.e. `cd simulator/hostsim`) and type `make clean` and then `make all`. This will build the HostSim library and place it in the *lib* directory.

Block Layer Simulator (BSim)

From the command window, go to the BSim installation directory (that is, `cd simulator/hostsim/applications/bsim`) and type `make clean` and then `make all`. This will build the BSim application executable image (`bsim`) and place it in the current directory.

Firmware

Firmware Environment Setup

Set the path to the ARM compiler.

```
setenv PATH <arm directory path>/DS-5_v5.28.1/sw/ARMCompiler6.9/bin:${PATH}
```

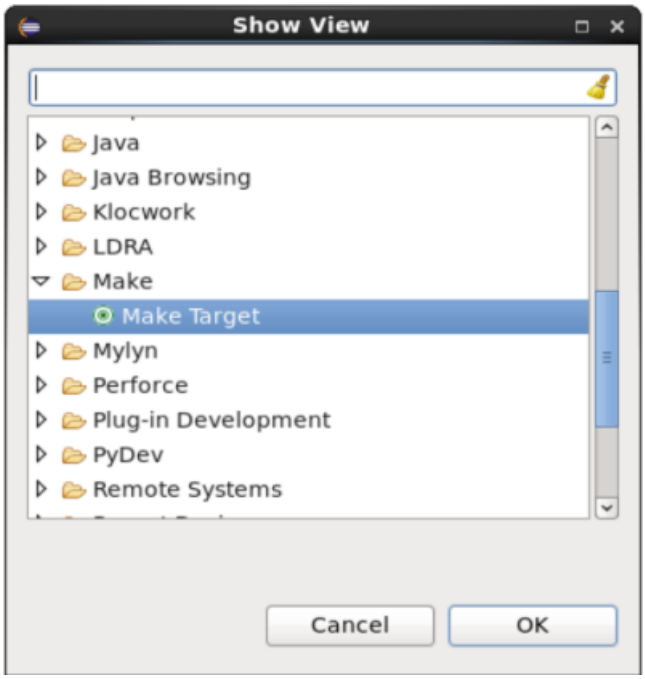
Complete the SDK Build

To compile the firmware in DS-5 Development studio, open the DS-5 Development studio from `<DS-5 Directory>/bin`. For example, `/DS-5_v5.28.1/bin/eclipse`.

The firmware source code is compiled through the DS-5 Make Targets dialog box. The Make Targets view provides an overview of make targets that are present in a project and provides a convenient way

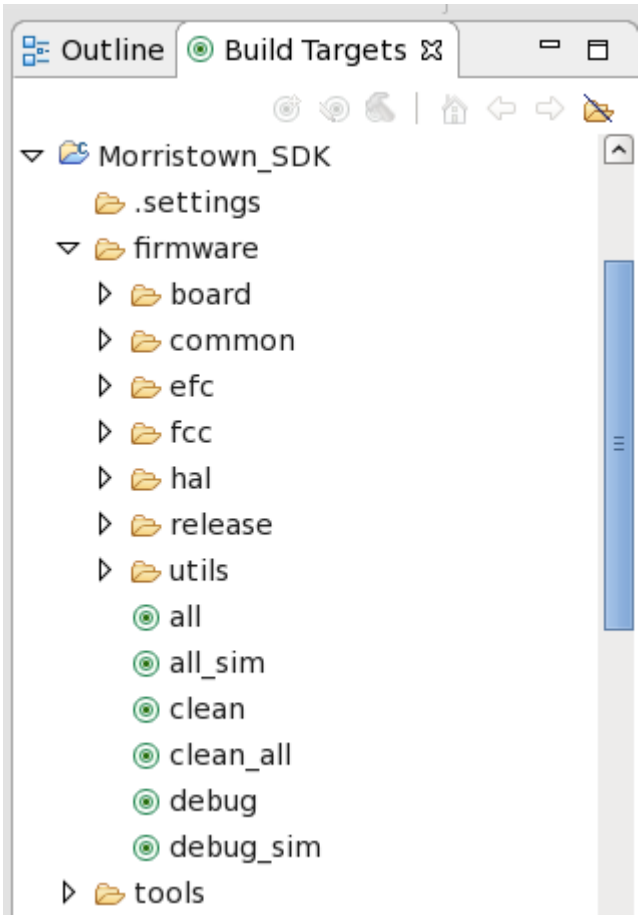
to execute them. To access the Make Targets view, from the **Window** pull-down menu select **Show View -> Other**. Select **Make Target** from the **Make** folder as seen in the following figure and click **OK**.

Figure 4-10 • Show View Dialog



The **Make Target** view, shown in following figure will appear in DS-5. Expand **Morristown**, then expand **firmware** under Morristown to see the make targets. Using these targets will build both the SDK firmware and the flash firmware.

Figure 4-11 • Firmware make targets



The following make targets are supported for generating executables for use with the hardware:

- **all** — Compiles firmware for hardware (-O3).
- **debug** — Compiles firmware for hardware with compiler optimizations disabled (-O0).

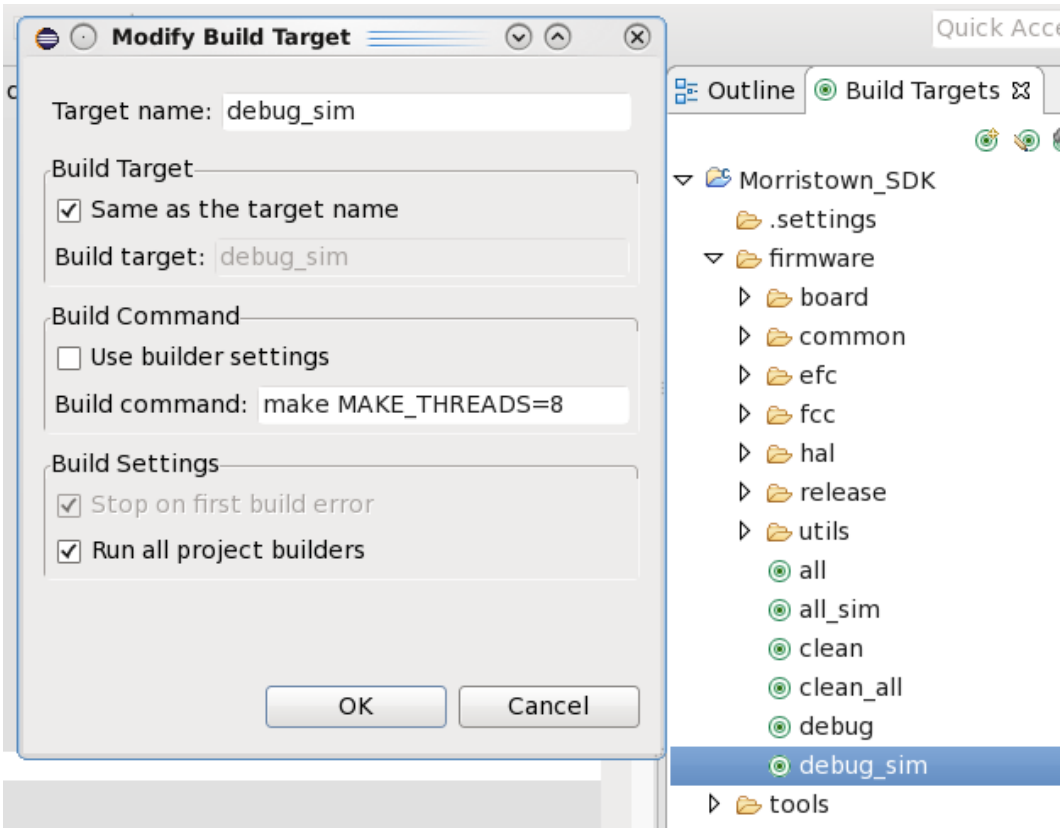
In addition to the above targets, the following targets are also supported. These targets are used for generating executables for use with the architectural simulator:

- **all_sim** — Same as the all target but compiles firmware for Architectural Simulator.
- **debug_sim** — Same as the debug target but compiles firmware for Architectural Simulator.
- **clean_all** — Clean_all will remove all the object files and all the files in the /firmware/release directories.

After building the target firmware and microcode, create EEPROM images and other data files that will be programmed into the hardware or loaded into the simulator.

The build process can take several minutes to complete. If you have purchased multiple user licenses for the ARM tools, it is possible to perform multiple operations in parallel by using the MAKE_THREADS parameter.

Figure 4-12 • Multi-threaded make release



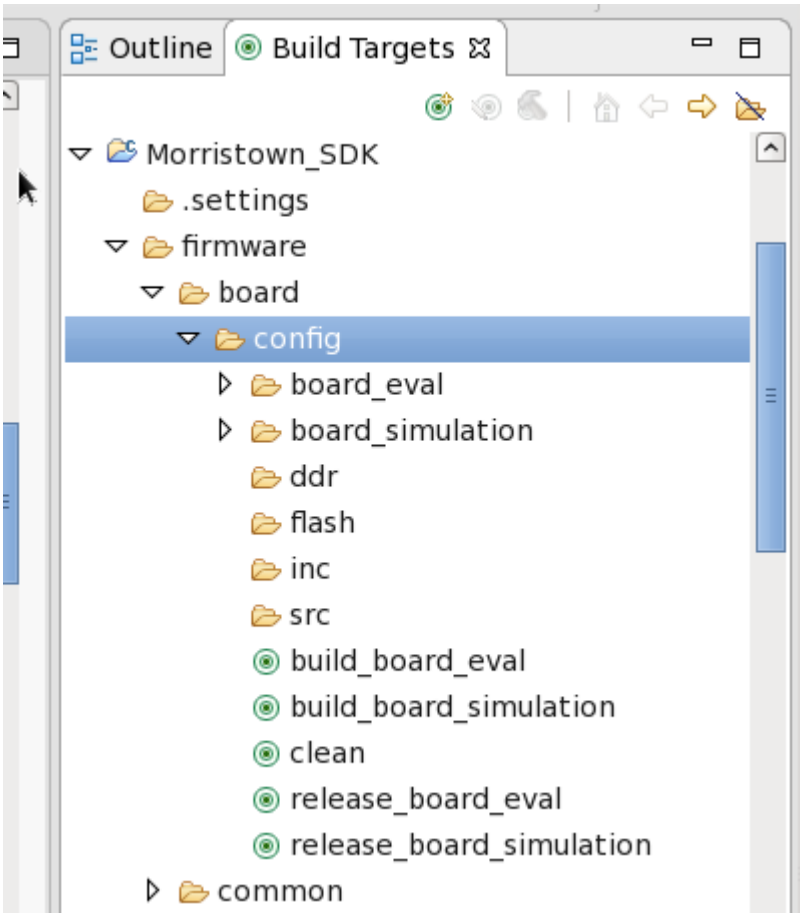
Building a Single Firmware Image or Board Configuration

During firmware development and debugging you may want to just recompile one build of your firmware for simulator debugging. To just rebuild the SDK firmware for ArchSim simulator in DS-5, double-click on the **debug_sim** make target. The updated SDK code image file sdk.axf will appear in the **PM71_85_270_sdk.x.y.z/firmware/release/archsim** directory as before.

To rebuild just the board-specific files, double-click on the **build_board_simulation** make target. This can be found when you expand the make targets in **Morristown->firmware->board->config**. When the

build is complete, use **release_board_simulation** to copy the files into the **firmware/release/board_simulation** directory.

Figure 4-13 • Board Configuration make targets



The updated files will be placed in the **PM71_85_270_sdk.x.y.z/firmware/board/config/board_simulation** directory, where you will also find the files that provide the board configuration details. Taking the additional step of executing the **release_board_simulation** make target will copy the updated files to their usual place under the release directory.

- **build_board_simulation** — Build only the board specific files for use with the Architectural Simulator.
- **build_board_eval** — Build only the board specific files for use with the hardware evaluation board.
- **clean** — Removes compiled files from all the board configuration directories.
- **release_board_simulation** — Copies all simulation board files to the firmware/release directory.
- **release_board_eval** — Copies all evaluation board files to the firmware/release directory.

Files Created by a Firmware Build

When the release build is done, a number of files are created in the release directories board-speciific directories with names beginning with "board_". Directory **PM71_85_270_sdk_x.y.z/firmware/release/board_simulation** receives the board-specific files for use with the ArchSim simulator. The following files will be found in this path: These files include:

- <install path>/firmware/release/board_simulation/spi.*.part — Each file contains all the data to be written to a SPI device partition.
- <install path>/firmware/release/board_simulation/spi*.sh — Shell scripts to program the individual partitions.

- <install path>/firmware/release/board_simulation/eeprom*.bin — Binary EEPROM images.

The partition file (extension ".part") aggregates all of the data that will reside in a partition of the EEPROM. See the Firmware Design Guide for more information on partitions.

Running SDK via PBL with ArchSim From a Shell Prompt

To test the result of your build you can the complete the firmware boot sequence starting with the PBL (primary boot loader) and ending with the SDK loaded and running. The results of this test can be seen in the archsim console output and via the UART. For more information on using the UART, see *How To Use UART Terminal with Archsim* on page 46 .

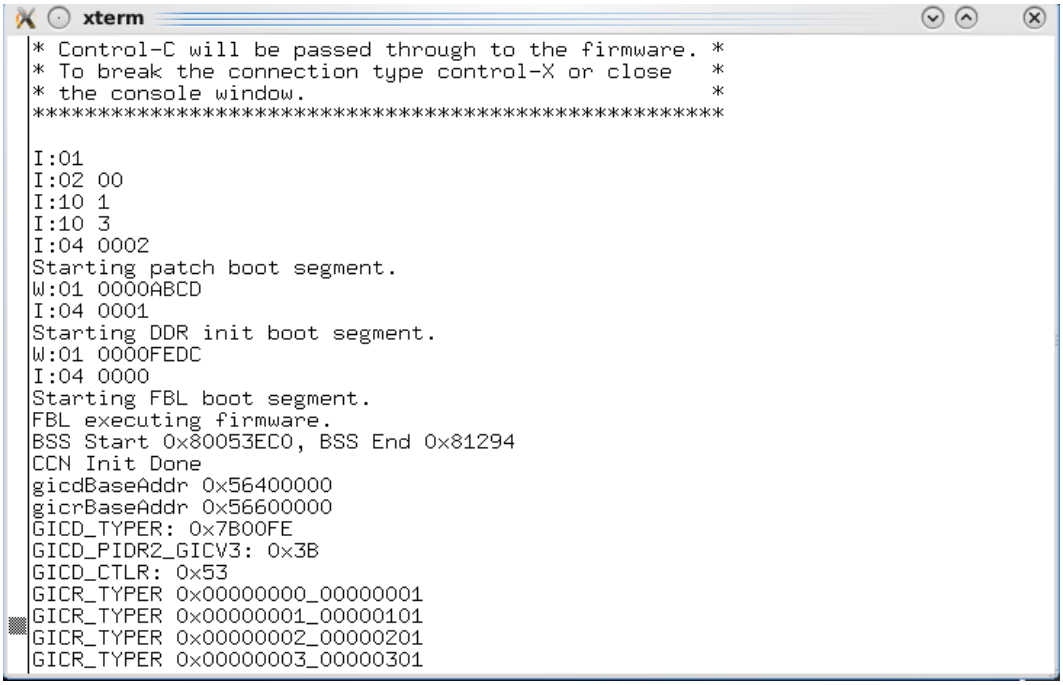
Use the following steps:

1. `cd <install path>/simulator/archsim`
2. `source <Thirdparty FastModels path>/Accellera/source_me.csh`
3. `source <ARM FastModels path>/source_all.csh`
4. `cp ../../PM71_85_270_sdk_x.y.z/firmware/board/config/board_simulation/eeprom?.bin .`
5. `./A53x4_clusts4_CCN502.x -b 0x00 --tcpIpPortBase <local-TCP-port> -a "pm8627_SC_subsys_i.pm8627_SC_subsys_internal_i.coreSystem.cluster*"=<path to SDK IMAGES>/release/pbl/pbl_sim.axf --run`

The following lines should appear in the console output as the simulator runs:

- "I:01" shows that the PBL has started to execute. Similar looking messages in the console text (<letter>:<number>) also come from PBL. See the *Microsemi Flashtec NVMe3016 Flash Controller Boot ROM Functional Specification* for details on these messages.
- The rest of the console output comes from the SDK firmware.

Figure 4-14 • UART Output Booting SDK



Adding debug stubs to a build

It is sometimes helpful to be able to stop boot path code execution before moving on to the next stage of the firmware boot process. The SDK boot process consists of a series of boot segments that execute serially. Replacing one of the boot segments with code that enters an infinite loop is a good way to stop the process until a JTAG or CADI debugging aid can take control of the processor.

The debugStub boot segment does exactly this. The SDK provides a number of boot images that incorporates the debugStub. For complete details refer to the firmware design guide section titled "Firmware Debug Stubs". Boot images created when a board configuration is built include:

boot.image — This is the normal image used to load and execute the firmware.

debug_all.image — This image configures the hardware and initializes the DDR, but loads a tight spin loop into the Flash controllers. This allows code to be loaded to debug the Flash microcode.

To replace the standard boot image in a SPI partition, edit the partitionDef.text file. Look for the lines beginning with "bootImage path=" and replace the path to the boot image with the path to one of the boot stubs listed above. You will need to rebuild the board completely and copy the new eeprom images to your ArchSim directory for the change to take effect.

- Go to the board configuration directory of the board you want to use for debug (e.g., board_simulation).
- Open partitionDef.text with a text editor and scroll down to the SPI partition you want to change..
- In the line beginning with "bootImage path=" replace the path to the standard boot.image file with the path to one of the debug images listed above.
- From the "config" directory "make clean" and then rebuild the board configuration files. .
- Replace ArchSim's eeprom?.bin files with the updated EEPROM files.

5 Debugging

The following sections describe how to run and debug the firmware. It is assumed that the procedures described in the previous chapters of this document have already been completed.

5.1 Archsim Debugging

There are two ways of debugging archsim:

- Attaching to a running archsim process
- Running archsim from debugger

Attach to archsim

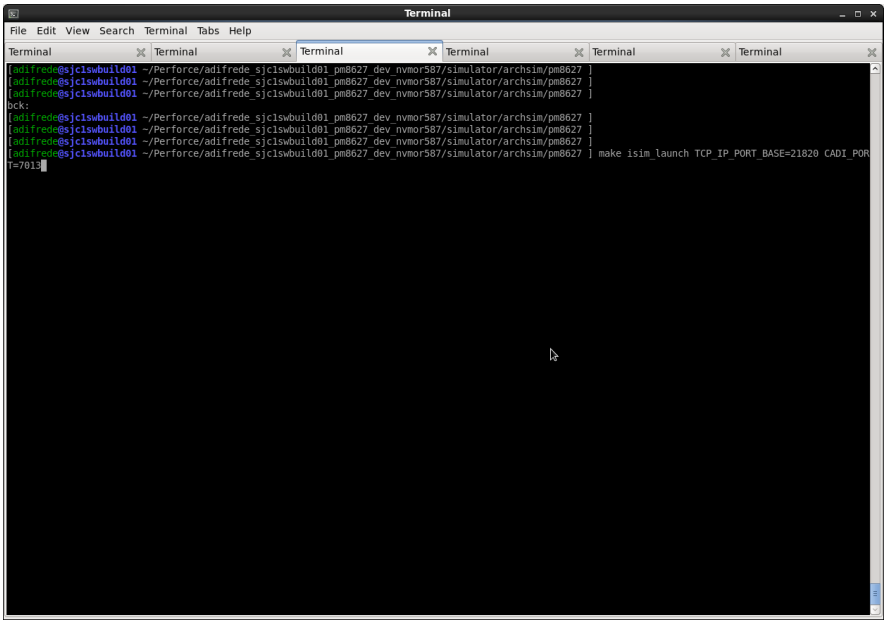
This section will explain how to attach to an archsim process

- The image below shows how to run the archsim. Use the make command line as specified in the Archsim guide. For example:

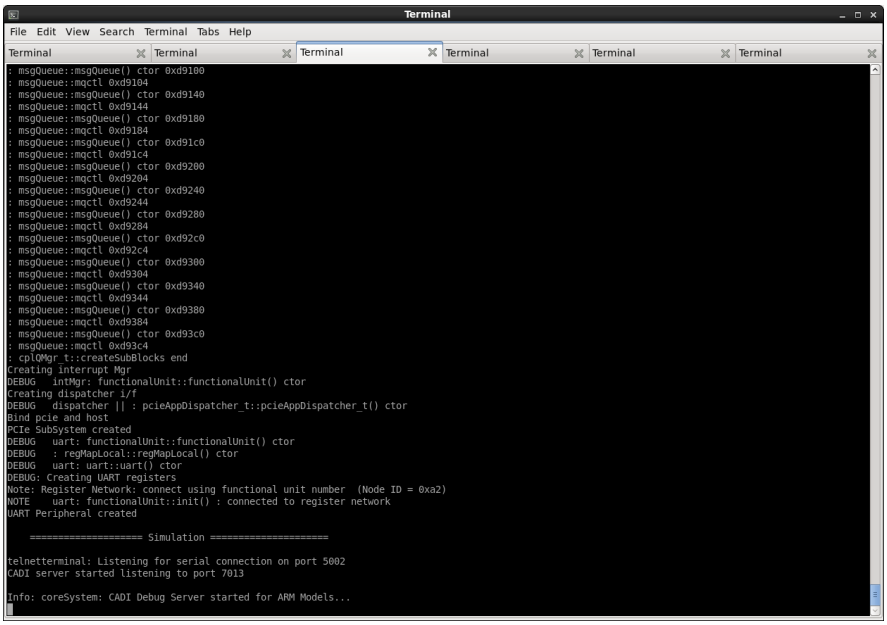
  ```
  make isim_launch TCP_IP_PORT_BASE=<TCP_IP_Port> CADI_PORT=70xx.
  ```

  If you wish to start Archsim with dual port support, specify this simCfg file as an argument:

  ```
  make isim_launch TCP_IP_PORT_BASE=<TCP_IP_Port> CADI_PORT=70xx
  SIMCFGFILE=simCfg_dualport.txt
  ```



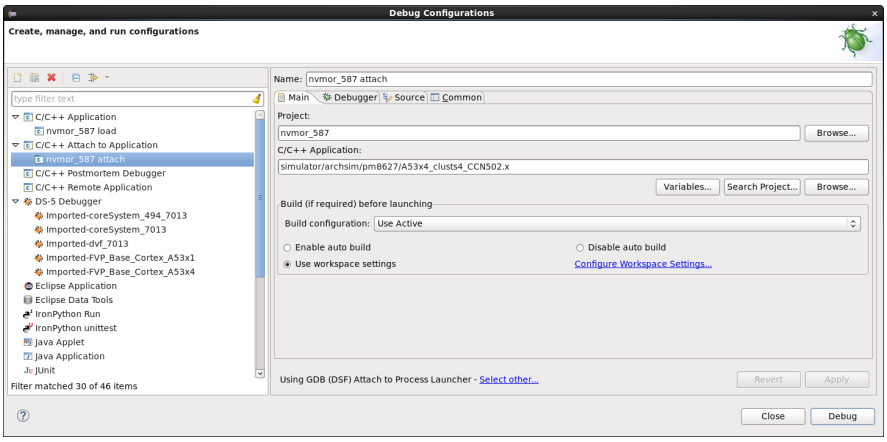
- The image below shows that the archsim is currently running.



- Create a new C/C++ Attach to Application debug configuration in DS-5.

  In the Main tab, fill out the Project entry as your current project where the archsim executable resides. Also, fill C/C++ Application entry to point to ***A53x4_clusts4_CCN502.x*** which is the archsim executable.

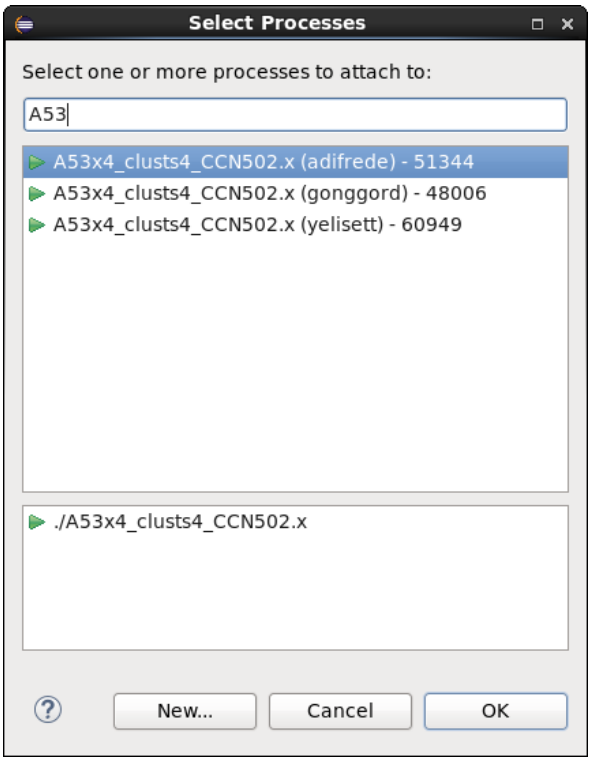
  Click Apply.



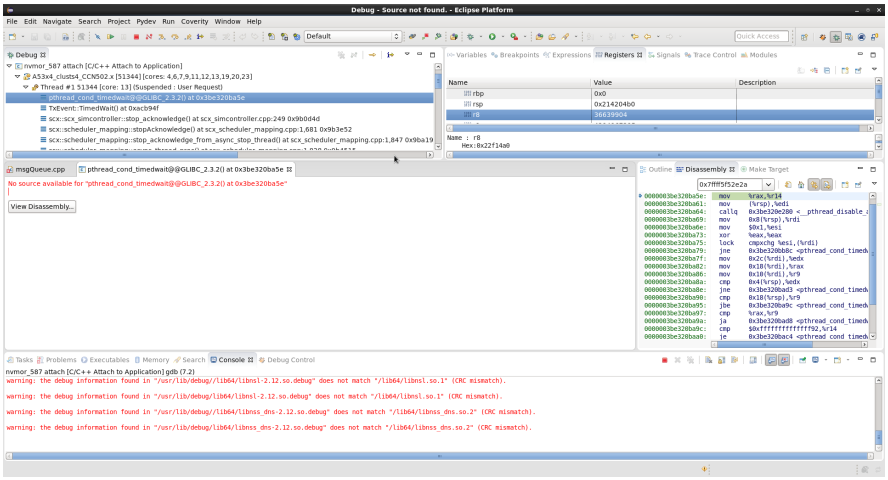
- Once Debug button is pressed. It will show the below window.

  Start typing ***A53*** to show all the ***A53x4_clusts4_CCN502.x*** process that are currently running. In this case, there are 3 running archsim processes.

  Pick your process and click OK.



- Once the debugger is done loading. We will see the window below. This shows the debugger has been attached to the process



All of the steps above are done using DS-5, however, it is also possible to use gdb command line to attach to archsim process.

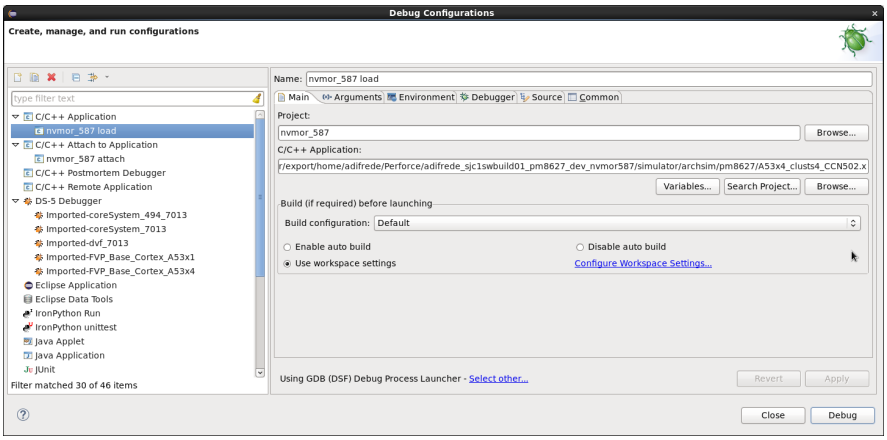
Running archsim with debug

This section will explain how to run archsim with debugger attached

- Create a new C/C++ Application debug configuration in ds-5.

  In the Main tab, fill out the Project entry as your current project where the archsim executable resides. Also, fill C/C++ Application entry to point to ***A53x4_clusts4_CCN502.x*** which is the archsim executable.

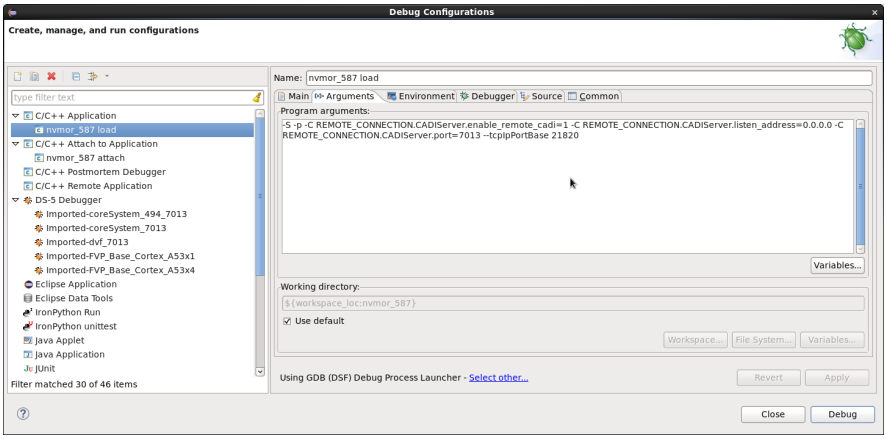
  Click Apply.



- There are two settings for the Arguments tab. The first image below shows the setting for the Arguments tab. The following is the argument example on how to run the archsim with the CADI server to attach the firmware to.

  -S is the option to start the CADI server. REMOTE_CONNECTION.CADIServer.port is the port where the firmware debug run will connect to. --tcpIpPortBase is to set the port to connect bsim. See the below figure for example:

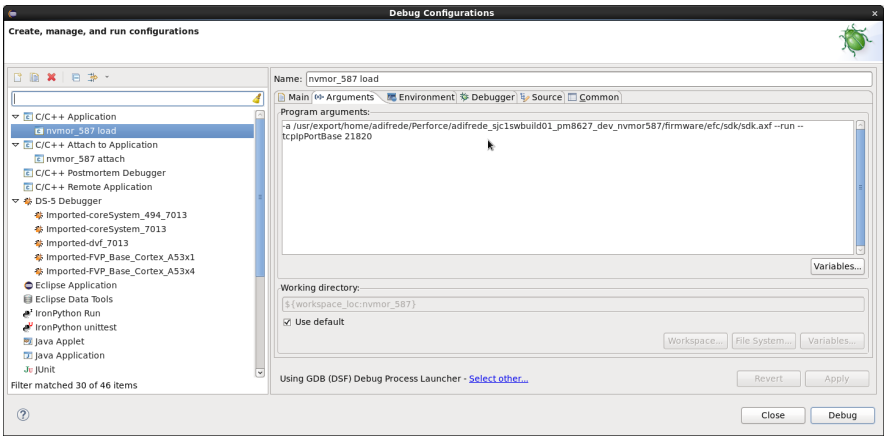
  -S -p -C REMOTE_CONNECTION.CADIServer.enable_remote_cadi=1 -C REMOTE_CONNECTION.CADIServer.listen_address=0.0.0.0 -C REMOTE_CONNECTION.CADIServer.port=7013 --tcpIpPortBase <TCP_IP_Port>



- This is the second setting for the Arguments tab. The following is the argument example on how to run the firmware and archsim altogether.

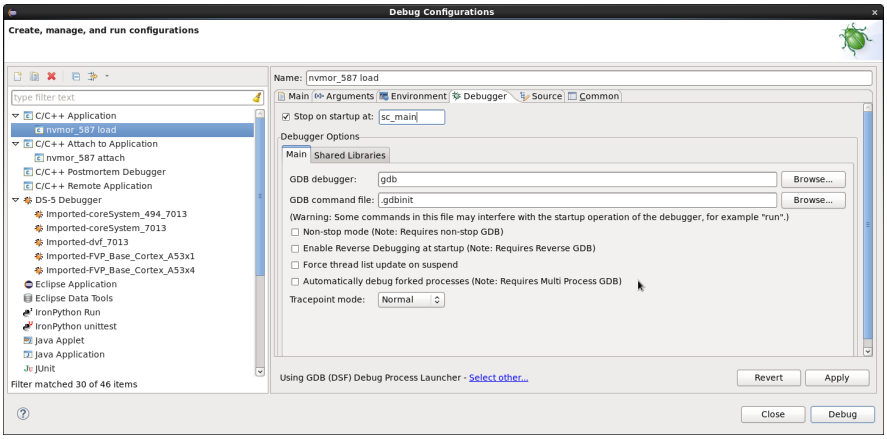
  -a is the option to set the firmware (*.axf) image. --run is to start the firmware. --tcpIpPortBase is to set the port to connect bsim. See the below figure for example:

  -a "pm8627_SC_subsys_i.pm8627_SC_subsys_internal_i.coreSystem.cluster*"=/Morristown/firmware/release/archsim/sdk.axf --run --tcpIpPortBase <TCP_IP_Port>

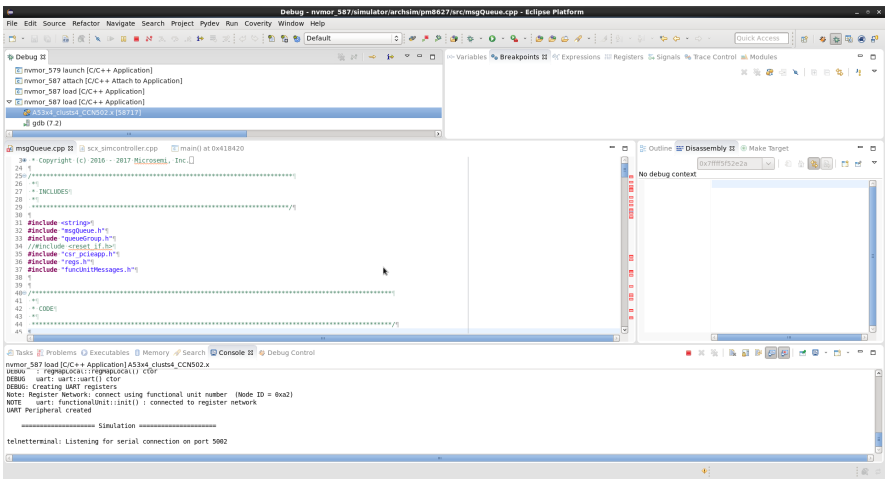


Note: Since the system utilize the same firmware image for all the cores, it is enough to load the image to only one core which will speed up the loading time.

- On the Debugger tab, set the Stop on startup at sc_main if you want the execution to stop at sc_main.



- Once Debug button is pressed. It will show the below window after the debugger fully loads. This shows the debugger has been attached to the process



All of the steps above are done using ds-5, however, it is also possible to use gdb command line to load and run the archsim process.

5.2 Firmware Debugging in Simulation

This section describes the procedure to initiate a firmware debugging session using the simulator (Archsim), once the firmware has been compiled for simulation (all_sim or debug_sim make targets).

Launch Archsim

Steps to run Archsim from a terminal once the compilation is complete:

1. From the **Command terminal window**, go to the folder where the Archsim is unpacked "`simulator/archsim`" and issue the following command:

   ```
   make isim_launch TCP_IP_PORT_BASE=<TCP_IP_Port> CADI_PORT=<70xx>
   ```

   If you wish to start Archsim with dual port support, specify this simCfg file as an argument:

   ```
   make isim_launch TCP_IP_PORT_BASE=<TCP_IP_Port> CADI_PORT=<70xx>
   SIMCFGFILE=simCfg_dualport.txt
   ```

   Port numbers 1024 to 65535 are valid TCP IP Ports. Port Numbers 7000 to 7063 are valid CADI ports.

   Figure 5-15 • Command Terminal Window

   There is another way to launch Archsim and the firmware together if firmware debugging is not required. In the same folder, after building the Archsim and the firmware, issue the following command to run with the SDK firmware image:

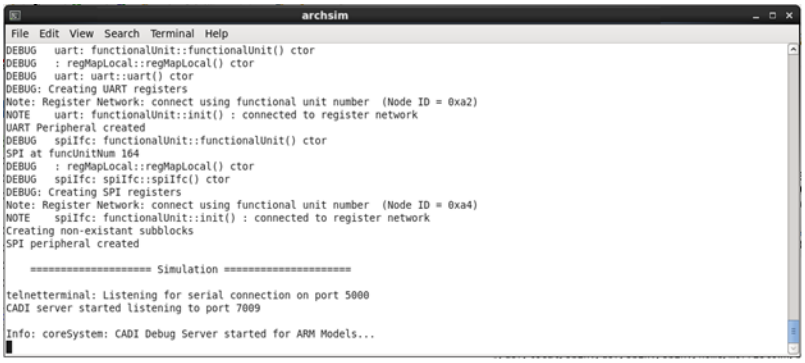
   ```
   ./A53x4_clusts4_CCN502.x -a
   "pm8627_SC_subsys_i.pm8627_SC_subsys_internal_i.coreSystem.cluster*"=<Path to
   SDK>/firmware/release/archsim/sdk.axf --run --tcpIpPortBase <TCP_IP_Port>
   ```

   -a is the option to set the firmware (*.axf) image. --run is the option to start the firmware. --tcpIpPortBase is the option to set the port to connect bsim.

   The FCC microcode will be loaded automatically by Archsim if the simCfg file is using the fcc parameter, which includes the path to the microcode file. By default the simCfg file uses the directory firmware/fcc/fcc_images as the location of fccNode.elf.

2. The following image shows that Archsim is currently running. It may take a few minutes for the SDK to start running.

Figure 5-16 • Archsim Running

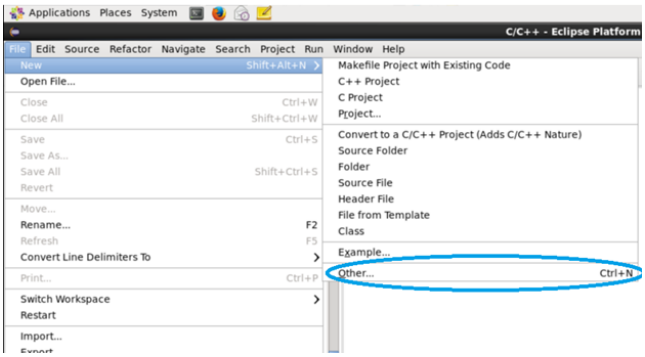


Run Firmware

Steps to run the firmware using DS-5 debugger:

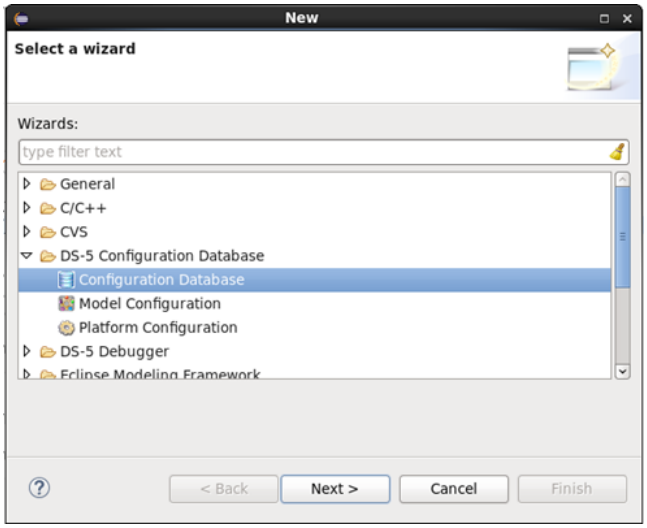
1. In the **DS-5 development studio**, select the following menu "**File->New->Other...**" as shown in the following image.

Figure 5-17 • DS-5 Development Studio



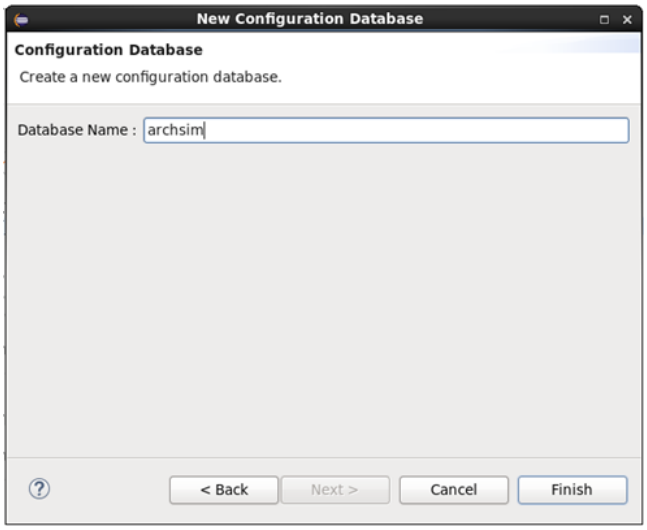
2. In the wizard, select **Configuration Database** and press **Next** as shown in the following image.

Figure 5-18 • DS-5 Configuration Database



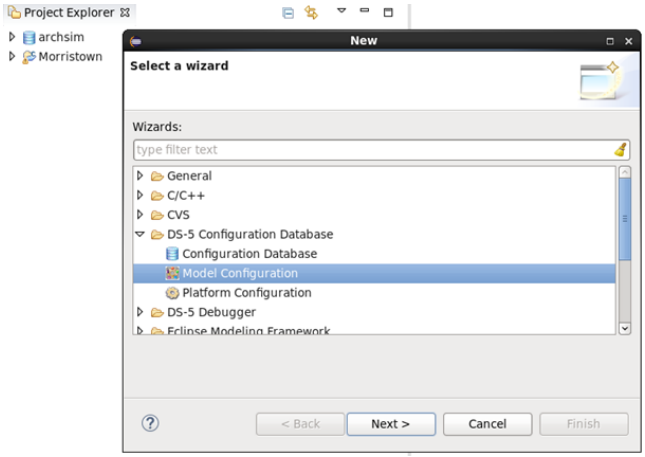
3. Provide a name to this new configuration, for example, "*archsim*" in **Database Name** and select **Finish.**

Figure 5-19 • New Configuration Database



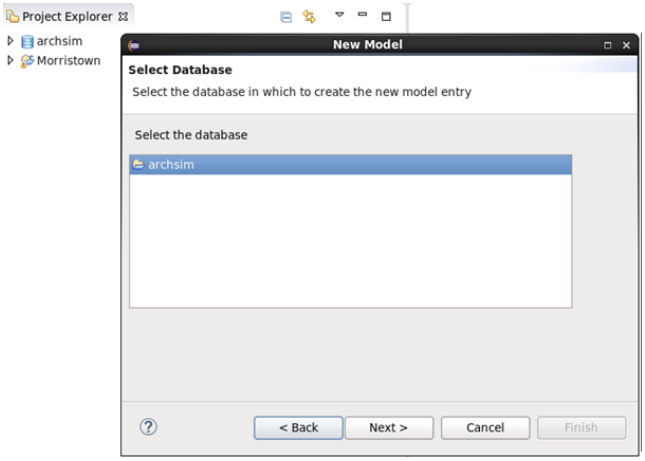
4. Again from the DS-5 studio, go to "**File->New->Other...**" and select "**Model Configuration**".

Figure 5-20 • Model Configuration



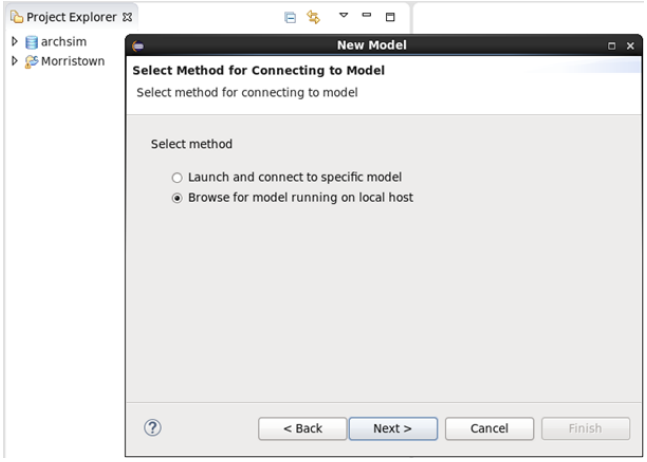
5. Select the database created earlier and press **Next.**

Figure 5-21 • New Model



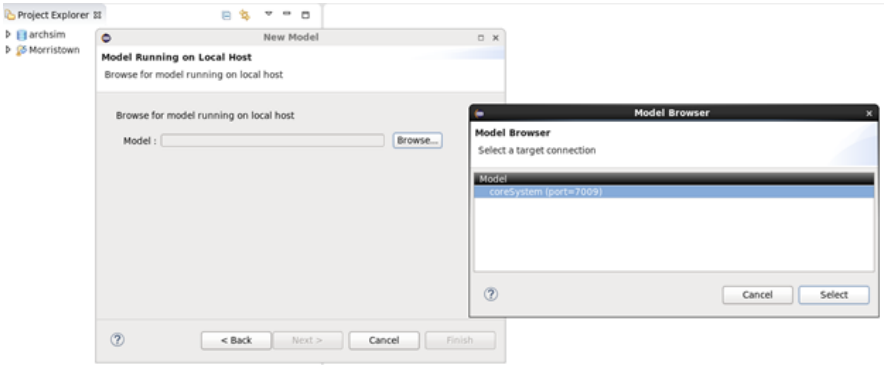
6. Select the **Browse for model running on local host** option and press **Next.**

Figure 5-22 • New Model Connection



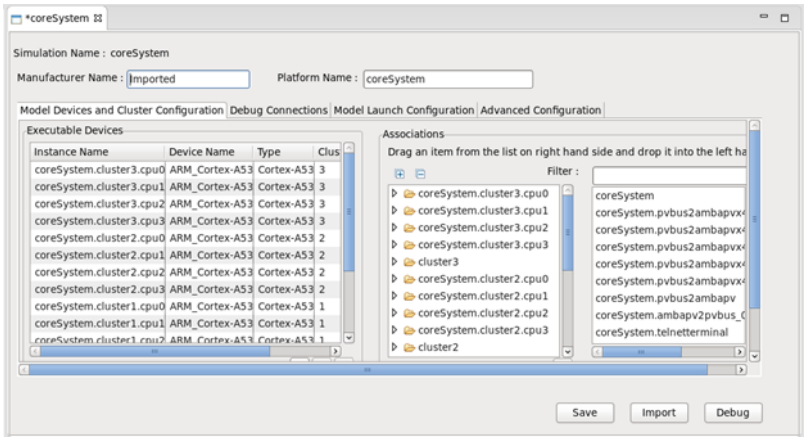
7. From the list of models, select the Archsim model run earlier (see *Launch Archsim* on page 26). In the following example it is "7009", select and click **Finish.**

Figure 5-23 • Model Browser



8. Once the Model is selected, this opens the coreSystem window as shown in the following image. Press the following buttons in the following order **Save** then **Import** and then **Debug**.

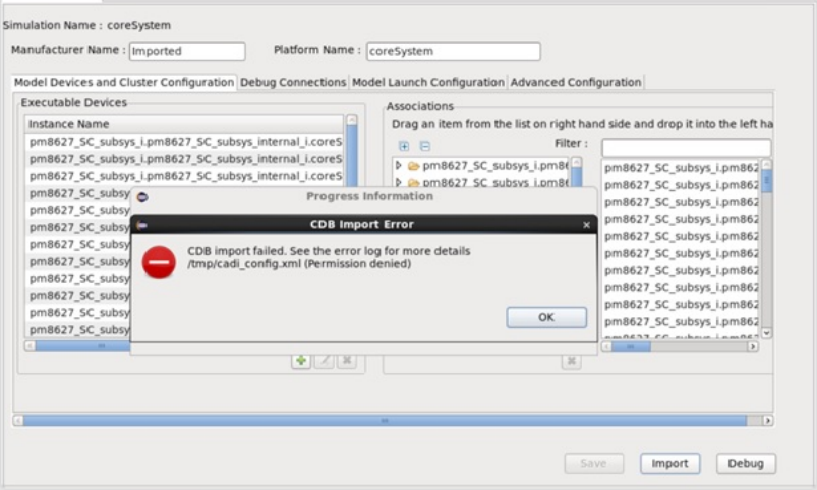
Figure 5-24 • CoreSystem



In DS-5 version 5.26.2, the **Import** creates a temporary file "`/tmp/cadi_config.xml`" that needs to be deleted manually so that another model can be run on the same server. This issue is fixed in version 5.27.

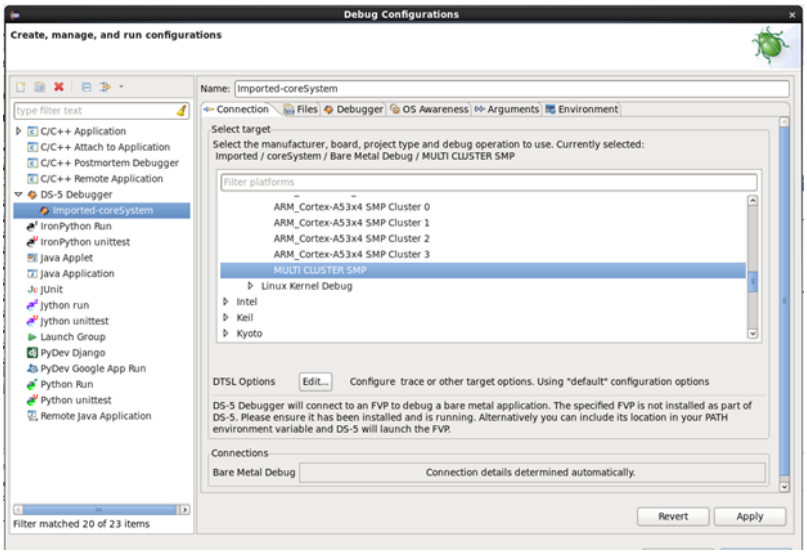
The message **CDB Import Error** is shown when the "`/tmp/cadi_config.xml`" file is not deleted and you attempt to **Import** a new model configuration. To proceed to **Debug** in this case, the file should be deleted by its owner.

Figure 5-25 • CDB Import Error



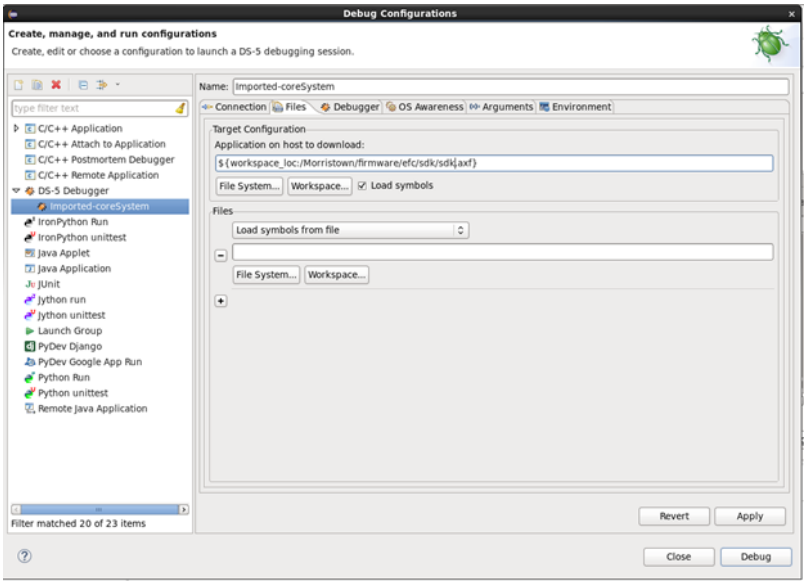
9. Selecting the **Debug** button above, opens the **Debug configurations** window as shown in the following image. In the **Connection** tab, select the **Multi cluster SMP** target which is under **Imported -> coreSystem -> Bare Metal Debug.**

Figure 5-26 • Debug Configurations Connection



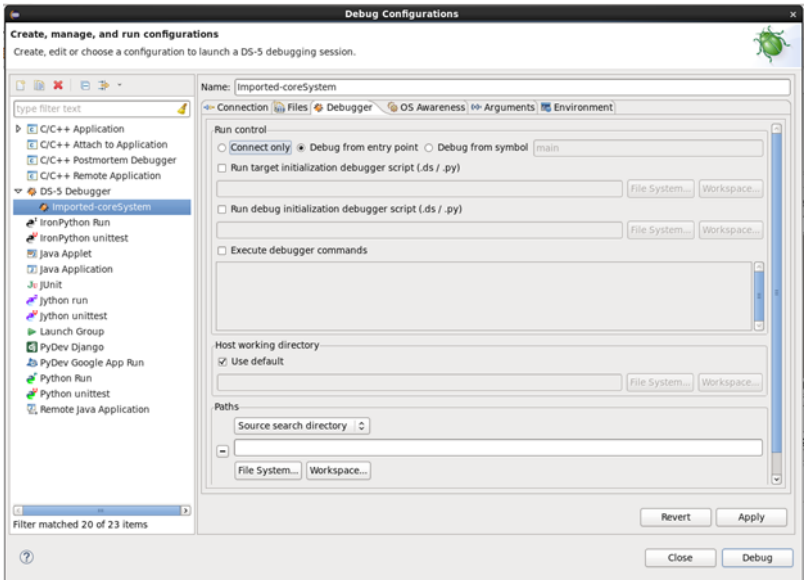
10. In the **Files** tab, click on the **Workspace** and select the firmware executable image as shown in the following image, which in this case points to:
`PM71_85_270_sdk_x.y.z/firmware/release/archsim/sdk.axf`

Figure 5-27 • Debug Configurations Files



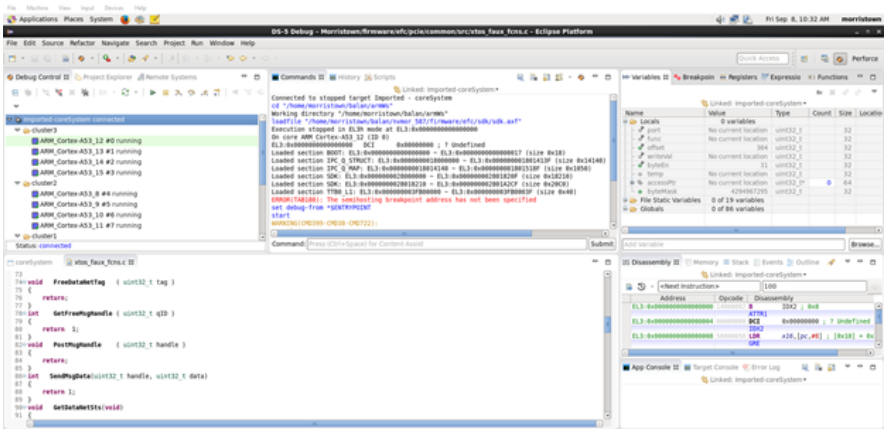
11. In the **Debugger** tab, select **Debug from entry point**, as shown in the following image. Now press **Debug** to launch the debug session.

Figure 5-28 • Debug Configurations Debugger



12. If the debug launch is successful, DS-5 will switch to the **Debug Perspective**. The following image shows the DS-5 debugger connected to the simulator. In the Debug Control window, press the icon ▶ to run the firmware.

Figure 5-29 • Debug Perspective



At this point, the firmware is up and running in the simulator. Now the simulator is ready to accept a connection from Hostsim.

Note: Once the debug configuration is created, you can go to the Debug Configuration option and select the previously created Debug Configuration.

Run Hostsim

Steps to run Hostsim from a terminal once the firmware has started running in the simulator (Archsim):

1. From the command terminal window, go to the folder "`simulator/hostsim/application/bsim`" and issue the following command

```
./bsim -p <TCP_IP_Port>
```

The port number "62100" is based on the archsim run command parameter TCP_IP_PORT_BASE (See *Launch Archsim* on page 26).

Figure 5-30 • Hostsim Terminal

2. The following image shows the hostsim has completed the PCIe and NVMe enumeration.

Figure 5-31 • Bsim Command Prompt

Exercising Firmware with BSim

When the PCIe and NVMe initialization shown in the preceding **Bsim Command Prompt** has been completed, BSim will issue an Identify Controller followed by one or more Identify Namespace NVMe administration commands to the firmware running on the architectural simulator to determine the Solid State Drive (SSD) configuration. When the configuration is known, the user is presented with the "BSim prompt" as seen below.

```
bsim >
```

This prompt allows users to specify commands that will be issued to the device. Typing *help* at the BSim prompt will display a list of supported commands. The following example commands, if entered in order, can be used to create an I/O completion/submission queue pair and then issue a write followed by a read to the device.

Examples:

```
bsim > createcplq -q 1
```

Creates a completion queue with the given queue number and default queue size.

```
bsim > createsubq -q 1 -c1
```

Creates a submission queue with the given queue number and associated completion queue.

```
bsim > write -q 1 -l 0 -b 8 -s
```

Issues a write command to write 8 host LBA's (-b8) starting from LBA 0 (-l0) on submission queue 1 with a seeded data pattern (-s).

```
bsim > read -q 1 -l 0 -b 8 -s
```

Issues read command to read 8 LBAs (-b8) starting from LBA 0 (-l0) on submission queue 1 that will verify the seeded data (-s).

5.3 Microcode Debugging in Simulation

This section describes the procedure to initiate a microcode debugging session using the simulator (archsim), once the firmware has been compiled for simulation (debug_sim make targets). Microcode debugging utilizes **xplorer**, which is used to debug the Flash channel cores. This document refers to the other section on running the firmware. To run and debug the microcode, the user will need to do the following:

1. Makefile changes.

In the archsim Makefile, add the **fccdebug** option under **isim_launch**. Adding **fccdebug** option will keep FCC cores stalled until the debugger is attached to them.

Figure 5-32 • Makefile fccdebug

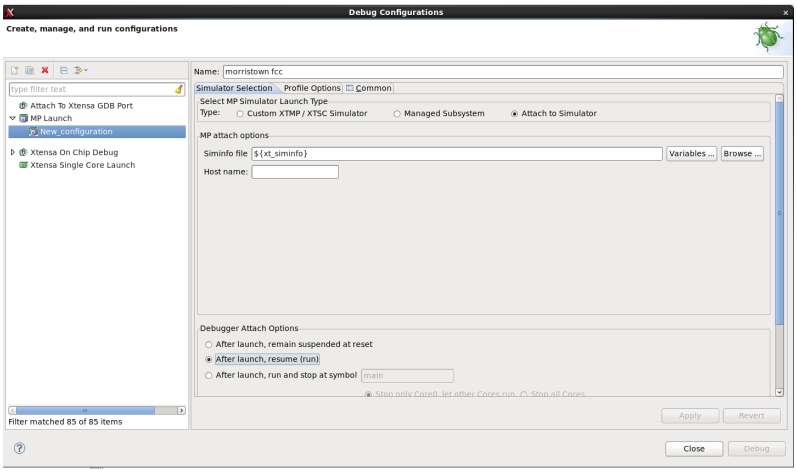
```
isim_launch:
    $(ISIM_DIR)/$(ISIM_NAME) \
    -S \
    -p \
    -C REMOTE_CONNECTION.CADIServer.enable_remote_cadi=1 \
    -C REMOTE_CONNECTION.CADIServer.listen_address=0.0.0.0 \
    -C REMOTE_CONNECTION.CADIServer.port=$(CADI_PORT) \
    --new-flash \
    --config-file $(ISIM_DIR)/simCfg.txt \
    --tcpIpPortBase $(TCP_IP_PORT_BASE) \
    --fccdebug
```

2. Launch archsim and load firmware into Morristown cores. This can be done as specified in the *Microcode Debugging in Simulation* section.

3. Set up the Xplorer debug configuration. The Xplorer debug configuration setup is similar to the one in DS-5, since they are both Eclipse-based.

   Selecting the **Debug** button from along the top of the main Xplorer window opens the **Debug Configurations** window as shown in the following image. In the left column, select **MP Launch** and create a **new launch configuration**. Rename this new debug configuration if desired. Select **Attach to Simulator** for the **Launch Type**. Scroll down if needed and select **After launch, run and stop at symbol <main>** for the **Debugger Attach Options.** This will pause the debugger after attaching and allow you to set breakpoints before running the microcode. If you do not set any breakpoints before running, then later, once the microcode has started running and the flash channel controllers are waiting for scheduler events, you will not be able to break into their execution using the debugger.

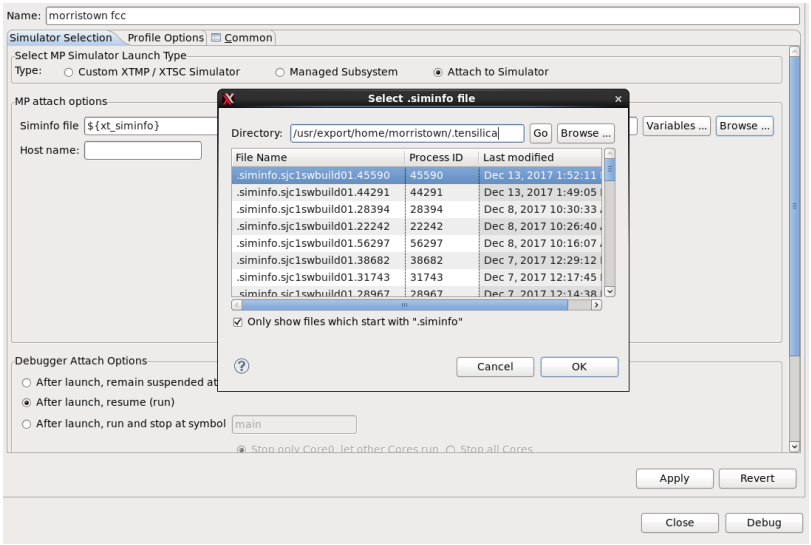
Figure 5-33 • Xplorer Debug Configuration



Under **MP attach options**, select **Browse** to find the **Siminfo file** for your currently running simulation. There could be problems if the tool cannot automatically find the correct **Siminfo file**, which essentially finds the running FCC processes based on the Tensilica metadata. If so, browse to find the correct location of the **.tensilica** directory. This is normally created in the user's home directory. Browse to

your **.tensilica** directory and select the latest process (Last modified) as shown in the following image. Click OK and continue to debug.

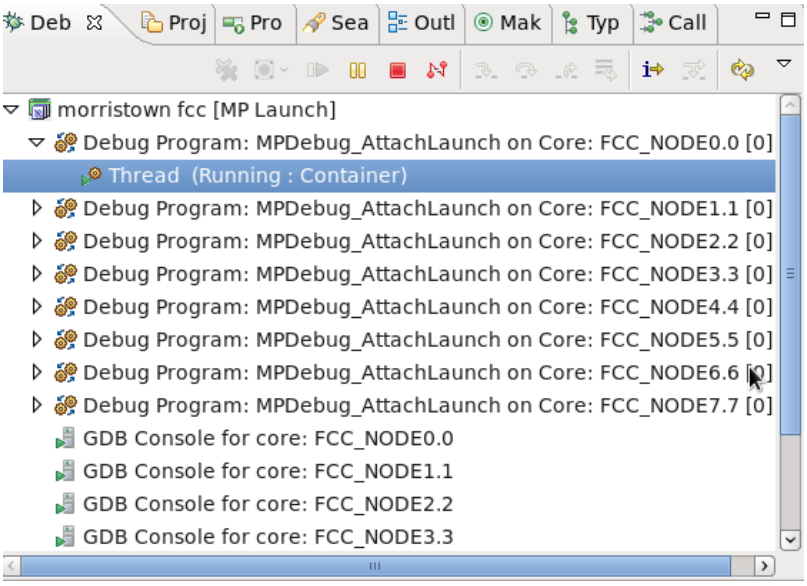
Figure 5-34 • Xplorer Find Running FCC Process



4. Xplorer debug window

Once the current running process is found, the debugger will attach to the process and show the following debug window. The following image shows eight FCC cores running. At this point, the user can set the breakpoint and start debugging.

Figure 5-35 • Xplorer Debug Window



5.4 Firmware Debugging in Hardware

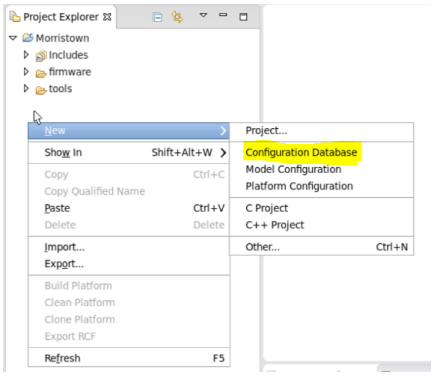
This section describes the procedure to create the Database Configuration, Hardware Debug Configuration and Trace Configuration

Database Configuration

Steps to create Database Configuration

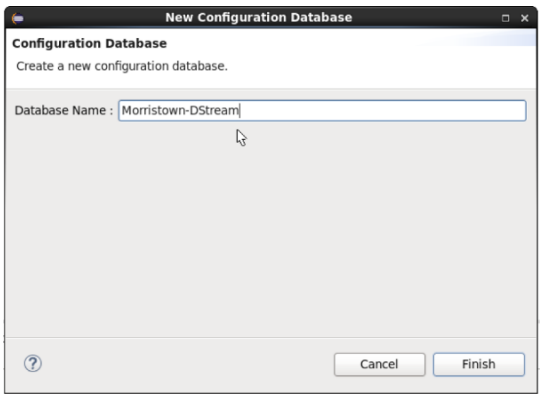
1. In DS-5, right-click in the project explorer window and select "Configuration Database as shown below

Figure 5-36 • Database Configuration



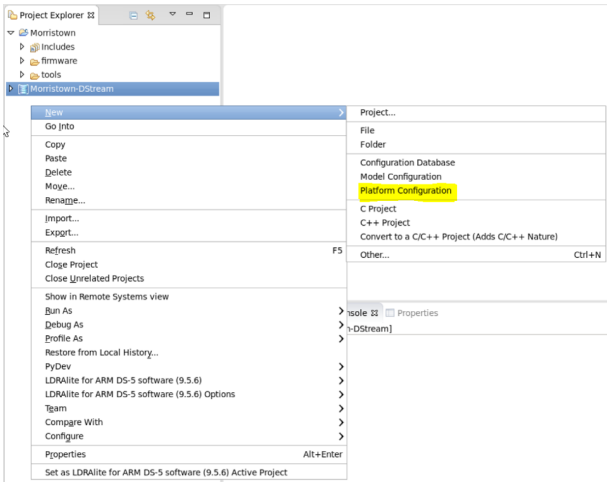
2. Enter the database name of your choice in the text box and click "Finish"

Figure 5-37 • Database name



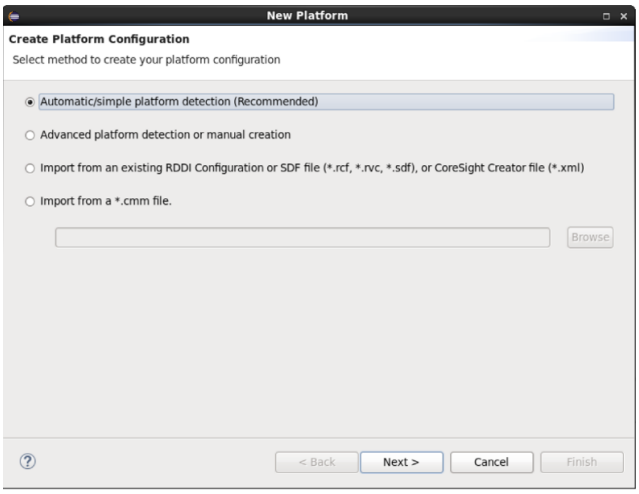
3. Again right-click in the project explorer, and select "Platform Configuration" as shown below

Figure 5-38 • Platform configuration



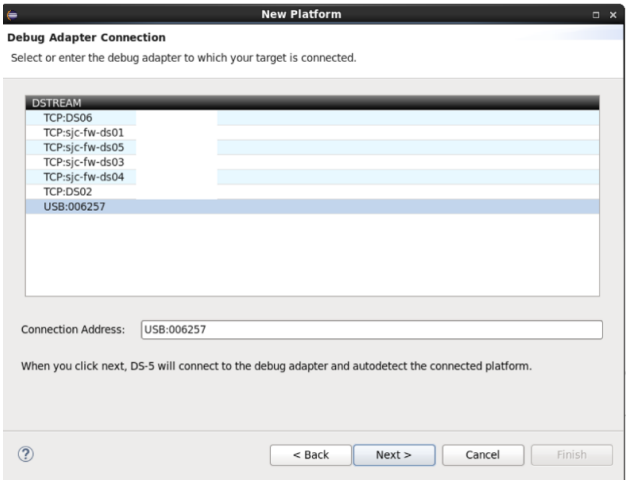
4. Select "Automatic/Simple platform detection (Recommended)" in "Create Platform Configuration" and click "Next"

Figure 5-39 • Platform configuration options



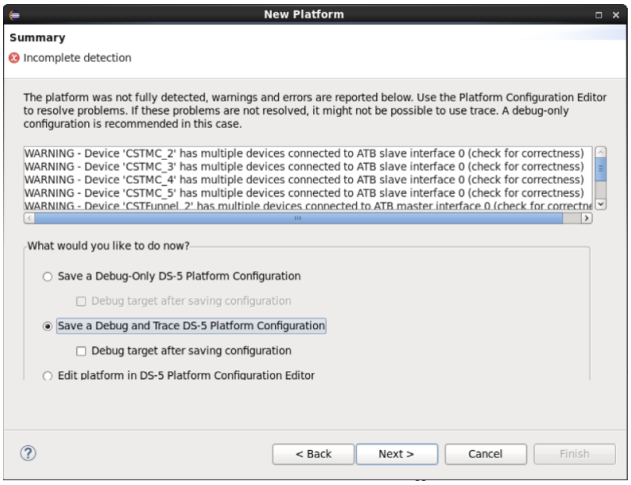
5. Select DStream device connected to your Linux host machine, and click "Next" to connect to the device

Figure 5-40 • DStream selection



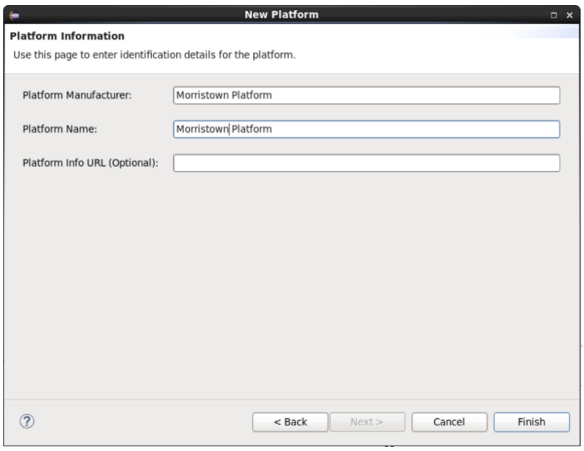
6. Select "Save a Debug and Trace DS-5 Platform Configuration" and click "Next"

Figure 5-41 • Platform configuration save options



7. Enter the platform manufacturer name and platform name in the text box as shown below

Figure 5-42 • Platform information



8. To remove the duplicate connections generated by the auto-detect,

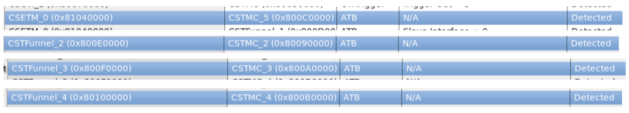
   Open the *.sdf file for the current platform

   Open the component connections tab under devices

   Remove the following device types

   Save the platform file by selecting "Save" from "File" menu

Figure 5-43 • Remove devices

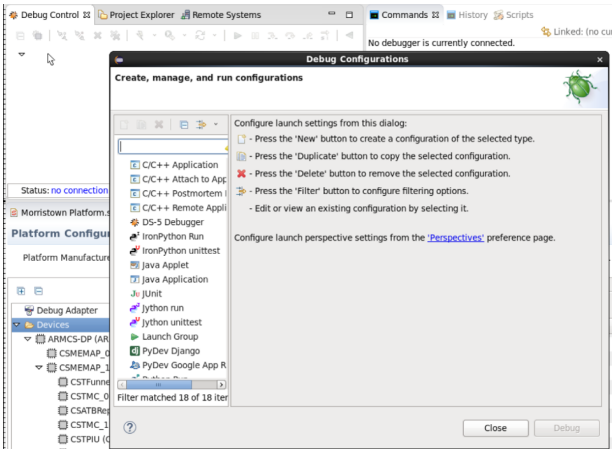


Hardware Debug Configuration

Steps to configure the hardware debug configuration

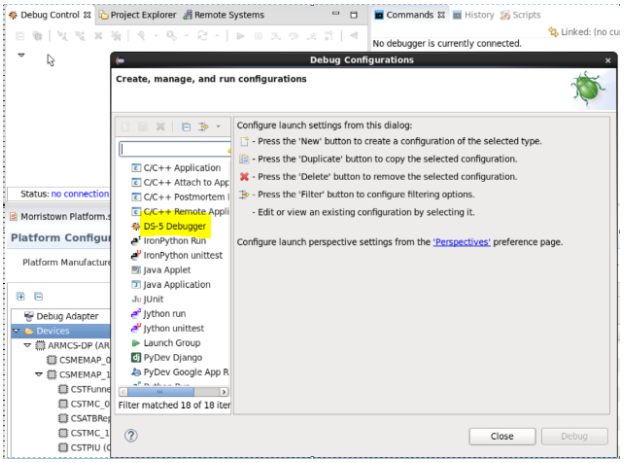
- In "Debug Perspective", right-click in "Debug Control" and select "Debug Configuration"

Figure 5-44 • Debug Configuration



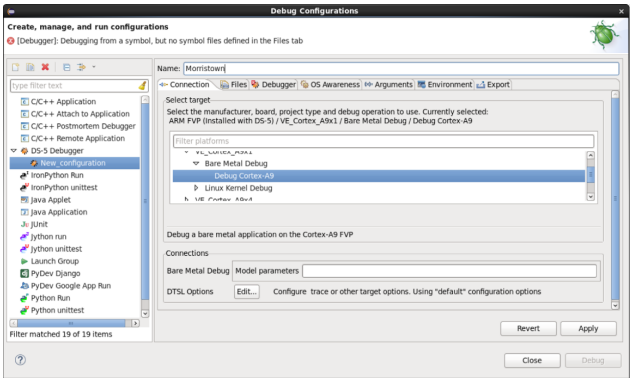
- Right-click on "DS-5 Debugger" and select "New"

Figure 5-45 • DS-5 debugger



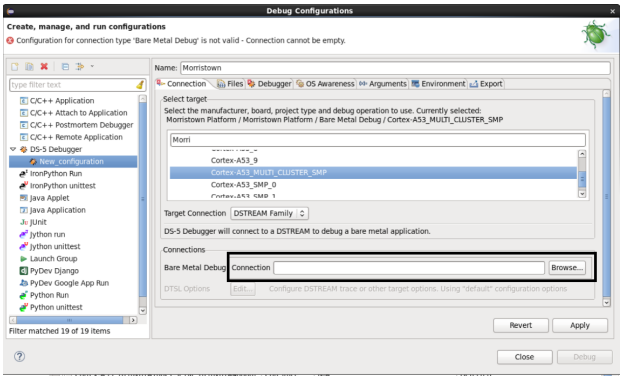
- Add name to the "Debug Configuration" in the "Name" field as shown below

Figure 5-46 • Debug configuration name



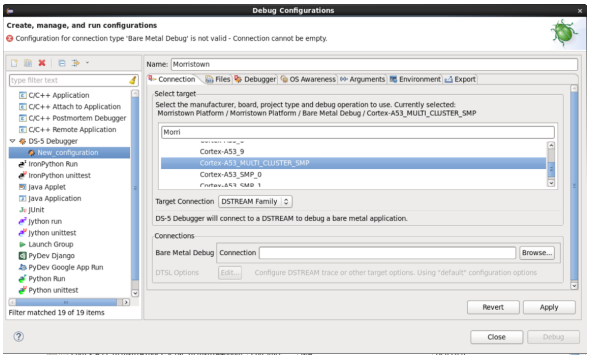
- In the "Connection" tab enter the name of the database created (eg., Morristown Platform), and select the target database

Figure 5-47 • Debugger connection



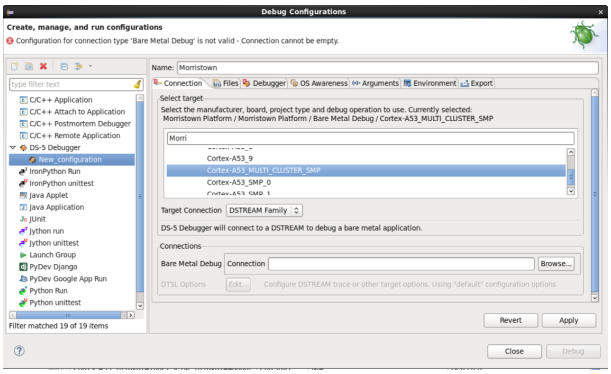
- After selecting the database, select "Corex-A53 Multi Cluster SMP" under "Bare Metal Debug"

Figure 5-48 • Processor configuration



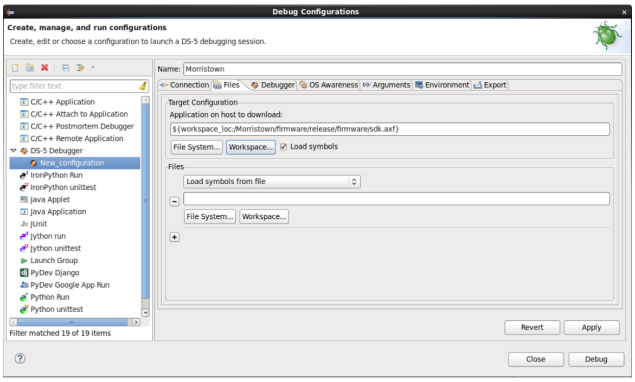
- Enter DStream ip-address in the connection field

Figure 5-49 • Debugger ip-address



- In the "Files" tab, select "sdk.axf" file in the workspace as shown below

Figure 5-50 • Target configuration



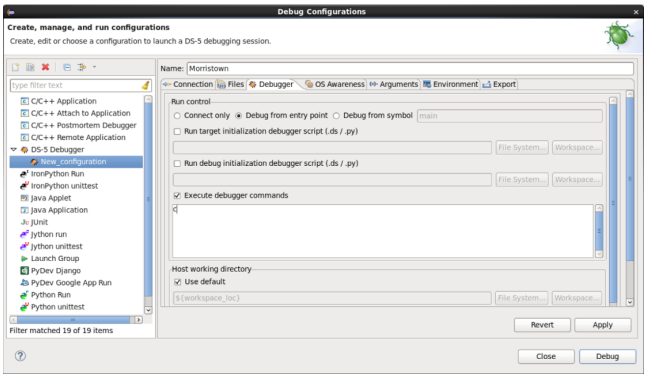
- In the "Debugger" tab, do the following

  Select "Debug from entry point"

  select "Execute debugger commands"

  In the text area, type "c" (this is to continue after launch)

Figure 5-51 • Debugger configuration settings



Click "Apply" and "Debug" to launch the debug configuration

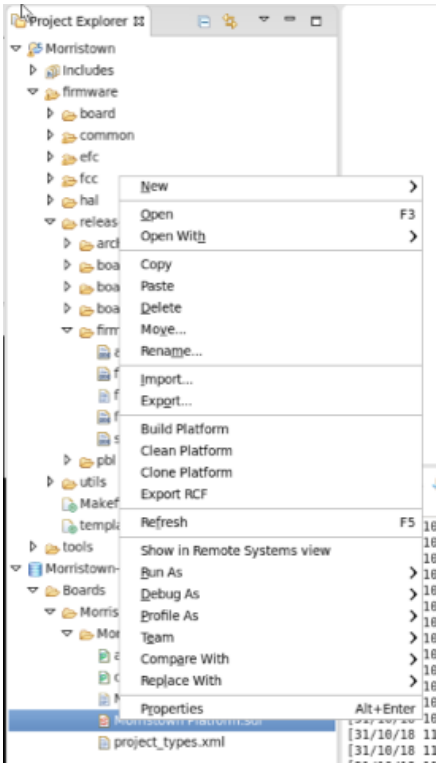
Note: Once the debug configuration is created, user can go to the Debug Configuration option and select the previously created Debug Configuration.

Trace Configuration

Steps for trace configuration

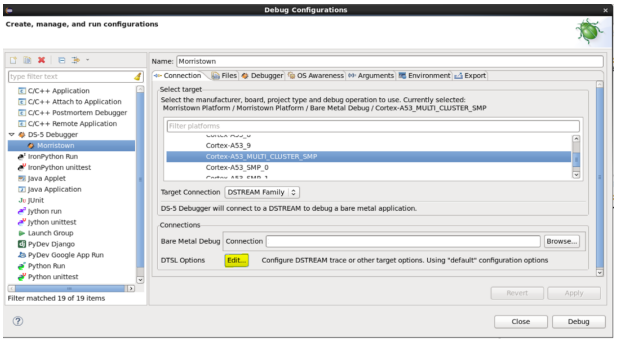
- Right-click on the board folder and select "Build Platform"

Figure 5-52 • Build platform



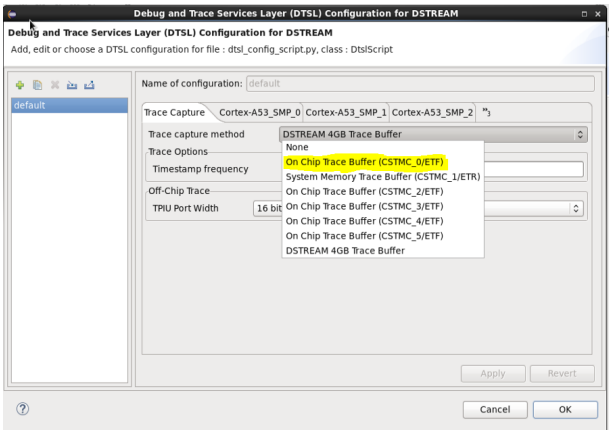
- In the "Debug Configuration", select "Edit" under "DST:L Options" as shown below

Figure 5-53 • DSTL options



- In the "Trace Capture" tab, select "On Chip Trace Buffer (CSTMC_0/ETF)"

Figure 5-54 • Trace capture options

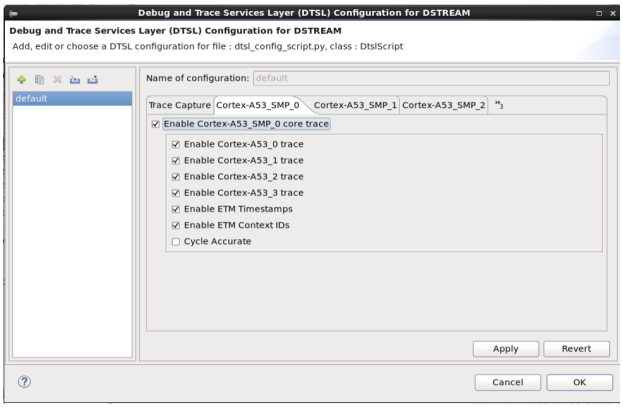


- In the "Cortex-A53_SMP_0" tab, select "Enable Cortex-A53_SMP_0 core trace" as shown below

  Select the appropriate cores to be traced.

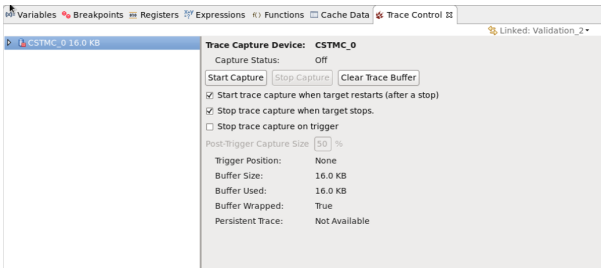
  Repeat the process for the other clusters

Figure 5-55 • Processor core trace options



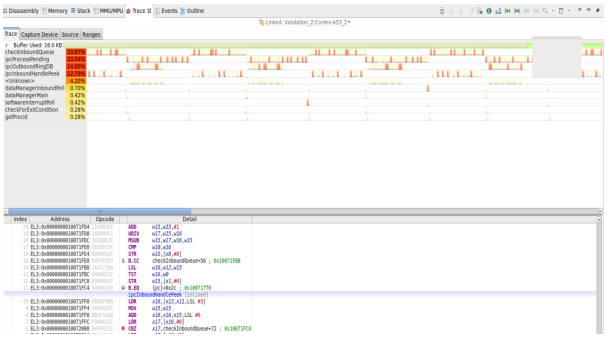
- "Trace Control" view can be opened to start trace capture, stop trace capture and clear trace buffer

Figure 5-56 • Trace control view



- "Trace Window" view will show trace captures as shown below

Figure 5-57 • Trace window view



6 How To Use UART Terminal with Archsim

The simulator tries to open named pipes "uart_console_in" and "uart_console_out" to mimic an external terminal's receive and transmit byte streams. If these files are not found, then the simulator assumes that the UART is being used for logging rather than interaction with a user terminal. In this case, UART output (the log text) goes to ArchSim stdout and there is no input stream.

If you want to quickly simulate terminal interaction with an application like DiagMgr or Hello World with ArchSim:

1. Compile UART terminal simulator.

   ```
   > make -C simulator/hostsim/applications/uartsim clean
   > make -C simulator/hostsim/applications/uartsim all
   ```

2. Create named pipes in the working directory where you will launch the simulator.

   ```
   > cd simulator/archsim
   > mknod uart_console_in p
   > mknod uart_console_out p
   ```

3. Launch ArchSim, which will look in the present working directory for the pipes.
4. In a separate Unix window, go to the same directory and run `"uartsim"` to start a simple terminal simulation. The terminal simulator will will open the other end of the pipes.

   ```
   > cd simulator/archsim
   > cp ../hostsim/applications/uartsim/uartsim .
   > ./uartsim
   ```

5. Use `control-X` to exit from the terminal emulator when you are done. `Control-C` is trapped and passed through to the UART.

If it is not convenient to run the terminal emulation from the ArchSim runtime directory, you can tell `uartsim` where to look for the pipe files. The formal command syntax is:

`uartsim [<display-input-pipe> [<keyboard-output-pipe>]]`

Default for `<display-input-pipe>` is "uart_console_in"

Default for `<keyboard-output-pipe>` is "uart_console_out"

7 References

Microsemi

1. *NVMe3016 BSim Users Guide*, ESC-2172032.
2. NVMe3016 SSD Development Platform Design Specification.
3. NVMe3016 Archsim Functional Specification.

ARM

1. *https://developer.arm.com/products/software-development-tools/ds-5-development-studio/docs*
2. *https://developer.arm.com/products/software-development-tools/ds-5-development-studio/downloads*
3. *https://developer.arm.com/support*