µC/OS-III on the Freescale i.MX 6 SoloX using DS-5 Tutorial

Introduction

This document explains how to setup and start developing with µC/OS-III on the Freescale i.MX6 Solox. The instructions and example projects are compatible with the Sabre Development Board. Usage of both the A9 and M4 cores will be demonstrated. These instructions are also compatible with µC/OS-II when using the OS-II example project.

Requirements

- µC/OS-III source code. For evaluation purpose the full source code is distributed with the example projects below.
- Micrium demonstration project. The project archive can be downloaded freely from Micrium website. Login required.
- ARM DS-5
- i.MX6 SoloX SDB development board
- Suitable debugger. For the purpose of this tutorial an ARM DSTREAM debug probe is used.
- A pre-built boot image may be needed to first debug the board. See known issues section for details.

Importing the Target Configuration

These steps are needed if trying the example projects before the SoloX support is official part of DS-5. The support files are provided by FreescaleLocate your DS-5 support files for the iMX6SX-SDB

1. Locate the iMX6SX SDB support files for DS-5

The archive should be extracted to a suitable location. It contains among other thing the dtsl_config_script.py and iMX6SoloX.rvc files required for import. For this example we will assume "C:\iMX6 SoloX Sabre SDB"

2. Run the CDB Importer from a command prompt

The cdbimporter utility is located in the bin directory of the DS-5 install. If DS-5 is not part of your PATH environment variable under Windows it's recommended to cd into the bin directory.

From this directory run the cdbimporter utility with thw following arguments :

Location of the main DS-5 configuration database.

```
-c "C:\Program Files\DS-5\sw\debugger\configdb"
```

Target of the generated configuration database for the SoloX

```
-t c:\solox-cdb
```

Location of the SoloX support files to import

`"C:\iMX6 SoloX Sabre SDB"`

Paths with spaces should be enclosed in quotes.

```
C:\Program Files\DS-5\bin>cdbimporter -c "C:\Program
Files\DS-5\sw\debugger\configdb" -t c:\solox-cdb "C:\iMX6 SoloX Sabre
SDB\iMX6SoloX.rvc"
```

For more information on the cdbimporter utility see the ARM knowledge base article How do I add bare-metal support for a new target in DS-5?

3. Run through the import wizard

In the steps that follows please note that names should be typed as they appear, otherwise the example projects configurations will refer to non-existent target.

When prompted for the core press enter to continue.

```
DS-5 Config Database Import Utility v1.2
Copyright 2011-2014 ARM Ltd

Reading C:\iMX6 SoloX Sabre SDB\iMX6SoloX.rvc

Found 2 ARM cores
Import Summary -
ID  Name       Definition  Associated TCF files
--  ----       ----------  --------------------
10  Cortex-A9  Cortex-A9   <none>
14  Cortex-M4  Cortex-M4   <none>

Select a core to modify (enter its ID and hit return) or press enter to
continue
. []
```

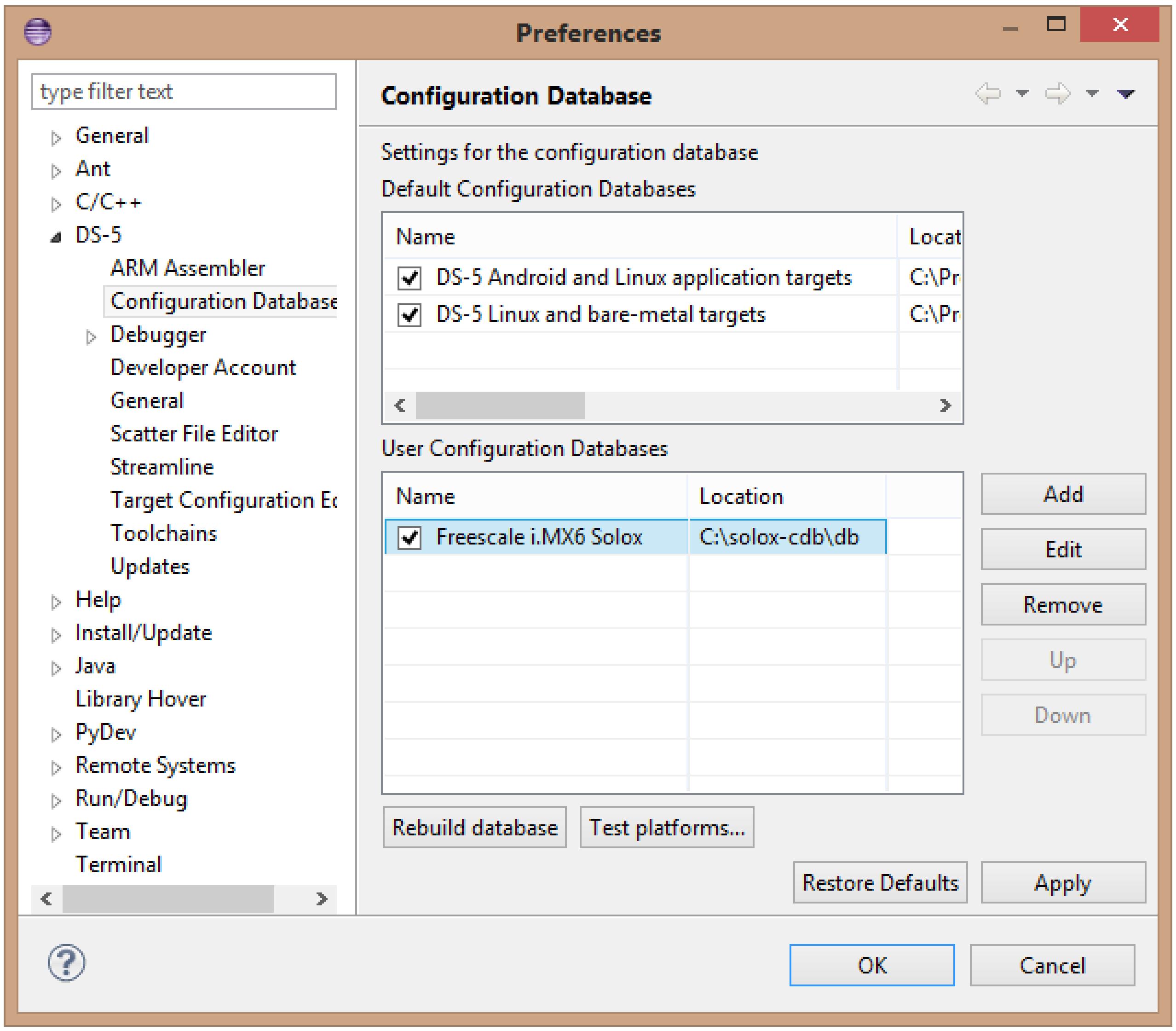
Type "Freescale" without the quote for the hardware manufacturer. Names are case sensitive.

And finally "i.MX6 Solox" without the quote for the platform name.

4. Add the configuration database to DS-5

From within the DS-5 preferences select the configuration database section. The preferences can be reached from the Windows->Preferences menu, then by selecting the DS-5 section.

From there click Add and browse or type the directory of the configuration database created at step 3.


Preferences

type filter text

- General
- Ant
- C/C++
- DS-5
  - ARM Assembler
  - Configuration Database
  - Debugger
  - Developer Account
  - General
  - Scatter File Editor
  - Streamline
  - Target Configuration E
  - Toolchains
  - Updates
- Help
- Install/Update
- Java
- Library Hover
- PyDev
- Remote Systems
- Run/Debug
- Team
- Terminal

Configuration Database

Settings for the configuration database

Default Configuration Databases

Name	Locat
✓ DS-5 Android and Linux application targets	C:\Pr
✓ DS-5 Linux and bare-metal targets	C:\Pr

User Configuration Databases

Name	Location
✓ Freescale i.MX6 Solox	C:\solox-cdb\db

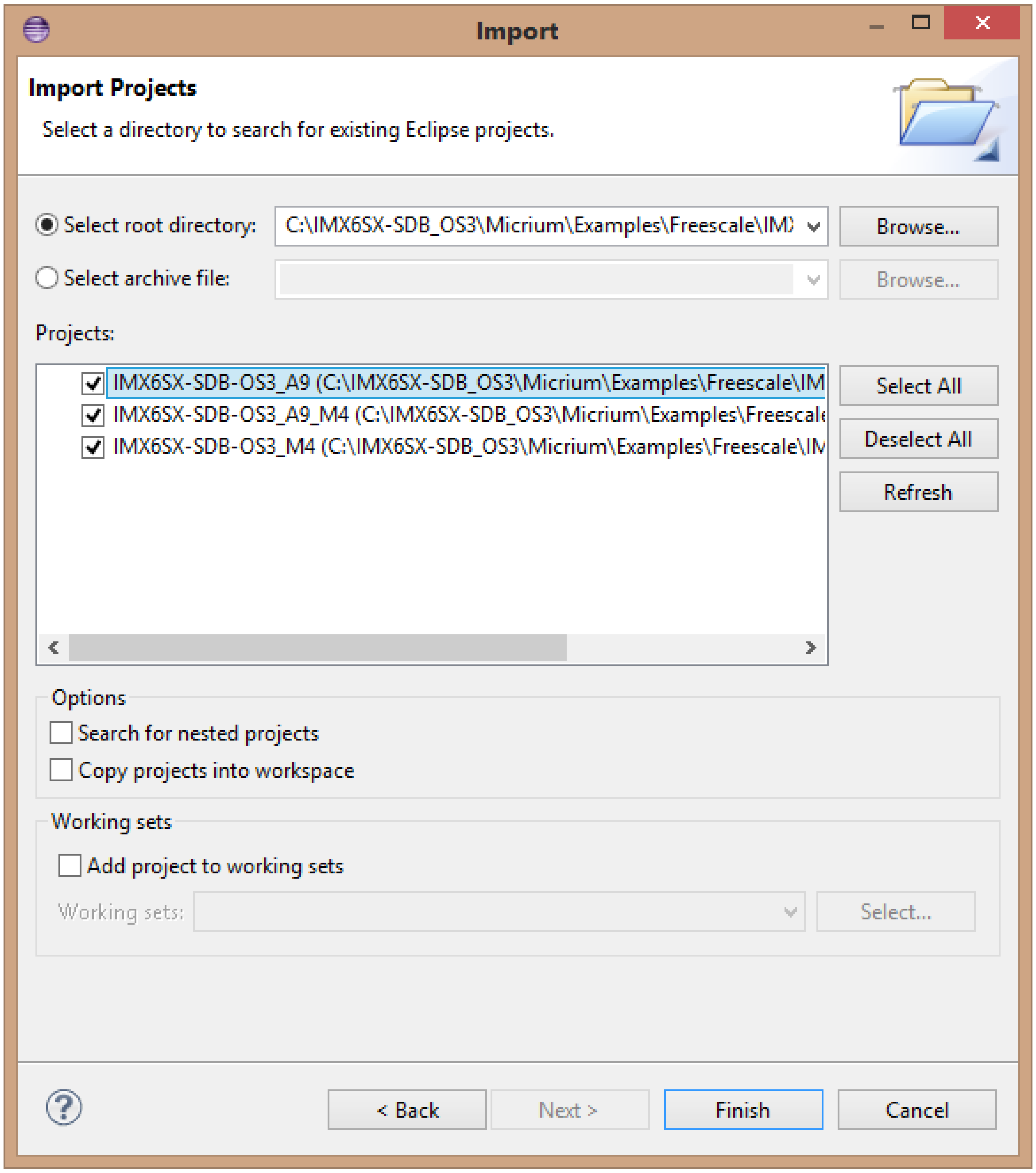
Figure - Configuration Database Import Dialog

Importing and Running the Example Project

For the purpose of this guide we will start from a set of fully working projects for the i.MX6 SoloX Sabre board.

1. Extracting and importing the DS-5 projects

The example projects ZIP archive should be extracted to a suitable location, in this example C:\. The projects can then be imported into a DS-5 workspace from the File->Import menu item. From there select the "Existing Projects into Workspace" option from the "General" category. On the next screen the root directory should be set to the Micrium\Examples\Freescale\IMX6SX-SDB folder location from the extracted archive. All three available projects should be imported and the "Copy projects into workspace" checkbox should be left unchecked as shown in the following image.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

(● Select root directory:) C:\IMX6SX-SDB_OS3\Micrium\Examples\Freescale\IM)

○ Select archive file:

Projects:

- [✓] IMX6SX-SDB-OS3_A9 (C:\IMX6SX-SDB_OS3\Micrium\Examples\Freescale\IM
- [✓] IMX6SX-SDB-OS3_A9_M4 (C:\IMX6SX-SDB_OS3\Micrium\Examples\Freescale
- [✓] IMX6SX-SDB-OS3_M4 (C:\IMX6SX-SDB_OS3\Micrium\Examples\Freescale\IM

Options

- [ ] Search for nested projects
- [ ] Copy projects into workspace

Working sets

- [ ] Add project to working sets

Working sets:

Figure - Eclipse Project Import Dialog

Three projects should now be imported into the workspace :

IMX6SX-SDB-OS3_A9 : Standalone project for the Cortex-A9

IMX6SX-SDB-OS3_A9_M4 : Combined project for the A9 and M4 simultaneously

IMX6SX-SDB-OS3_M4 : Standalone project for the M4

2. Building the projects

The projects can be individually built using the Project->Build Project menu item. The IMX6SX-SDB-OS3_A9_M4 requires the IMX6SX-SDB-OS3_M4 project to be built first.

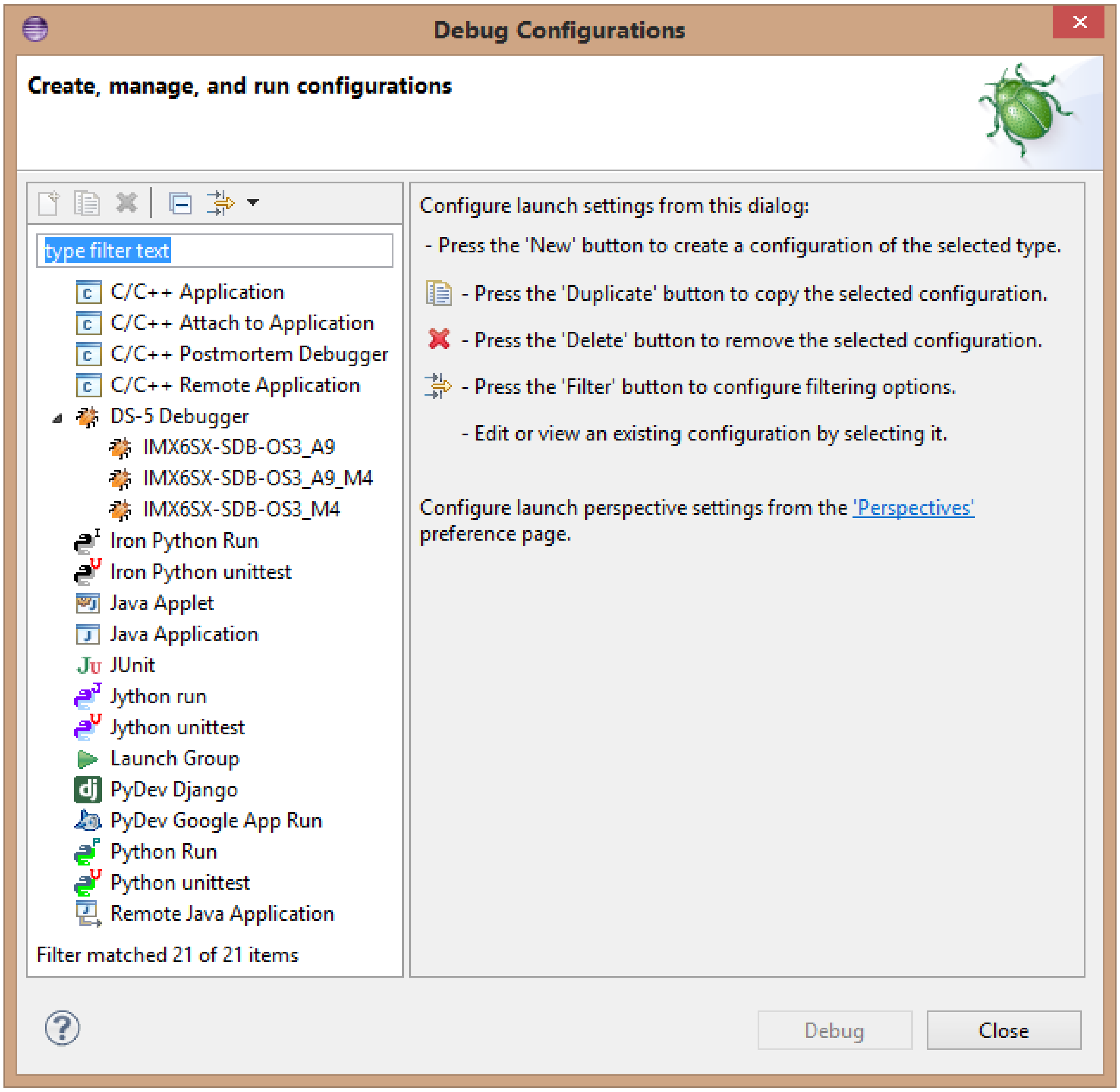
The projects can be individually built using the Project->Build Project menu item. The TWR-VF65GS10-A5_M4 requires the TWR-VF65GS10-M4 project to be built first.

3. Terminal Output Setup

The example projects contains a basic UART BSP to enable independent trace output from each cores. See the Freescale Getting Started guide for the board for details on installing and configuring the USB to UART converter.

4. Load and run the example projects

With all three project in the workspace opening the "Debug Configurations" panel should reveal four separate debug configuration provided with the examples.


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- DS-5 Debugger
  - IMX6SX-SDB-OS3_A9
  - IMX6SX-SDB-OS3_A9_M4
  - IMX6SX-SDB-OS3_M4
- Iron Python Run
- Iron Python unittest
- Java Applet
- Java Application
- JUnit
- Jython run
- Jython unittest
- Launch Group
- PyDev Django
- PyDev Google App Run
- Python Run
- Python unittest
- Remote Java Application

Filter matched 21 of 21 items

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.

- Press the 'Delete' button to remove the selected configuration.

- Press the 'Filter' button to configure filtering options.

- Edit or view an existing configuration by selecting it.

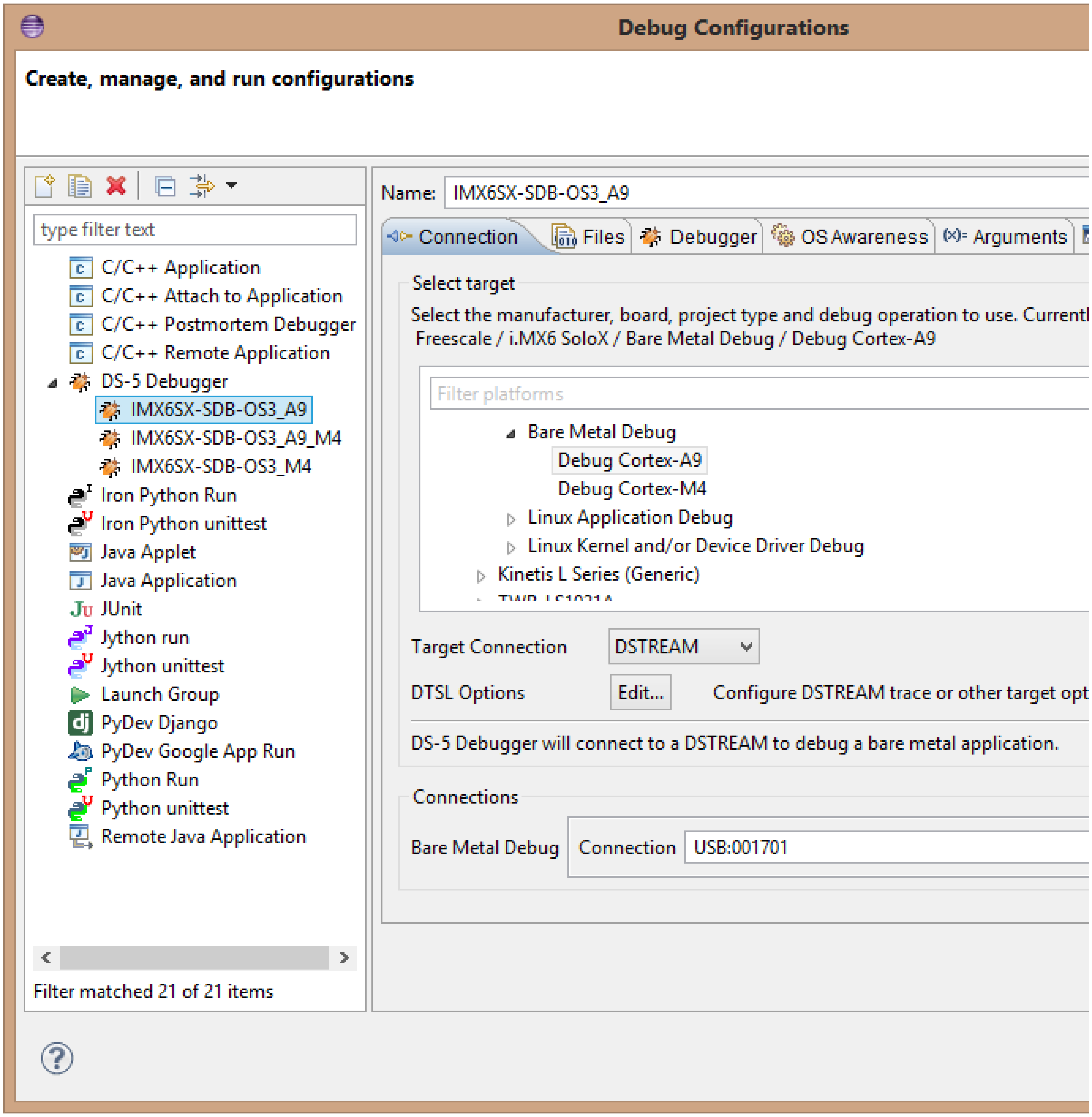
Configure launch perspective settings from the 'Perspectives' preference page.

Figure - Debug Configurations Panel

Loading and running the standalone A9 project

The standalone A9 example project runs µC/OS-III on the A9 exclusively with the M4 disabled. To run it select the IMX6SX-SDB-OS3_A9 debug configuration to open the target configuration panel. The following screen should appear :

(See the known issue section if an error message appears saying that the debugger can't connect to the target)


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- DS-5 Debugger
  - IMX6SX-SDB-OS3_A9
  - IMX6SX-SDB-OS3_A9_M4
  - IMX6SX-SDB-OS3_M4
- Iron Python Run
- Iron Python unittest
- Java Applet
- Java Application
- JUnit
- Jython run
- Jython unittest
- Launch Group
- PyDev Django
- PyDev Google App Run
- Python Run
- Python unittest
- Remote Java Application

Filter matched 21 of 21 items

Name: IMX6SX-SDB-OS3_A9

Connection | Files | Debugger | OS Awareness | Arguments

Select target

Select the manufacturer, board, project type and debug operation to use. Currentl
Freescale / i.MX6 SoloX / Bare Metal Debug / Debug Cortex-A9

Filter platforms

- Bare Metal Debug
  - Debug Cortex-A9
  - Debug Cortex-M4
- Linux Application Debug
- Linux Kernel and/or Device Driver Debug
- Kinetis L Series (Generic)

Target Connection: DSTREAM

DTSL Options: Edit... Configure DSTREAM trace or other target opt

DS-5 Debugger will connect to a DSTREAM to debug a bare metal application.

Connections

Bare Metal Debug: Connection USB:001701

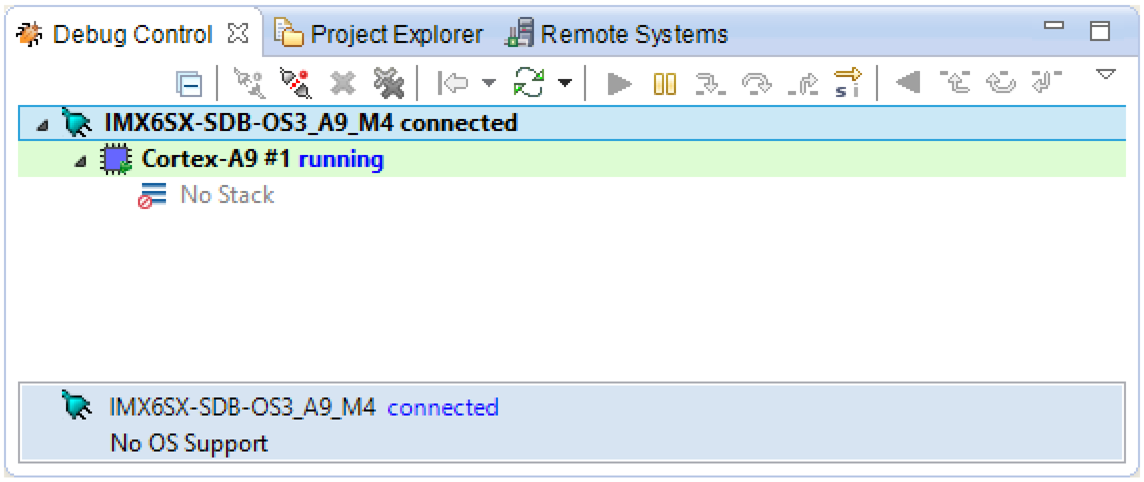
Figure - Debug Target Configuration Panel

From here the target connection should be updated to the correct debugger (DSTREAM in this example) and if application the Connections should be updated to point to the correct debugger by clicking the Browse button. Clicking Debug should open a debug session to the SoloX A9 core.

Loading and running the A9 and M4

The M4 on the SoloX cannot be debug in a standalone configuration and must first be booted from the Cortex-A9.

The details of creating a project that enables both cores is explained a little later in this tutorial. For now we will only describe the procedure to open two debug sessions for the A5 and M4. The first step is to load the IMX6SX-SDB-OS3_A9_M4 and then run it. This will enable and start execution of both cores. Before proceeding further the target should be running by pressing the "play" button. The debug control window should look like this :

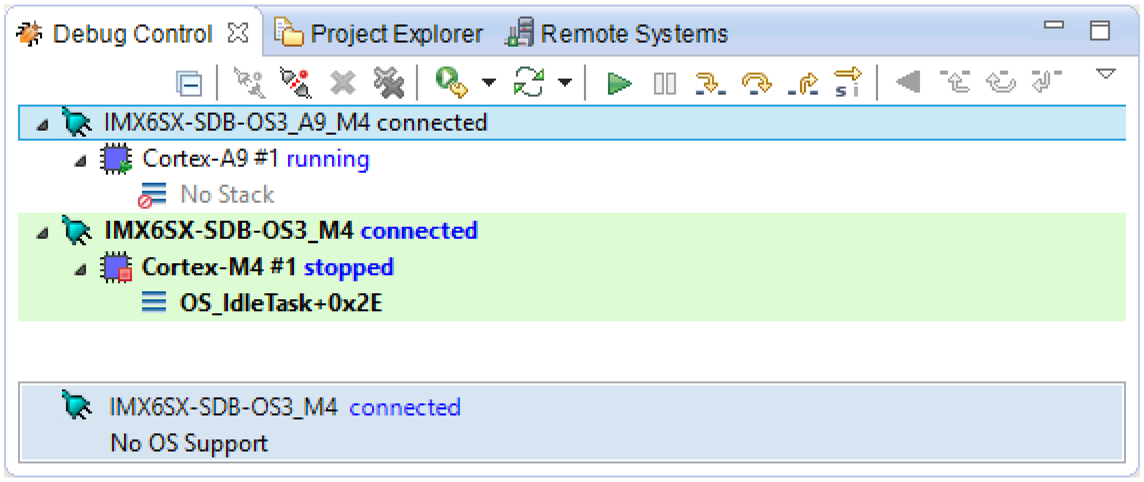


IMX6SX-SDB-OS3_A9_M4 connected
Cortex-A9 #1 running
No Stack

IMX6SX-SDB-OS3_A9_M4 connected
No OS Support

Figure - Debug Control window with one open connection

While the A9 core is running a second debug session can be opened in DS-5 by loading the IMX6SX-SDB-OS3_M4 debug configuration. Afterwards two debug sessions should be active and can be independently controlled by selecting the active session in the Debug Control windows.



IMX6SX-SDB-OS3_A9_M4 connected
Cortex-A9 #1 running
No Stack
IMX6SX-SDB-OS3_M4 connected
Cortex-M4 #1 stopped
OS_IdleTask+0x2E

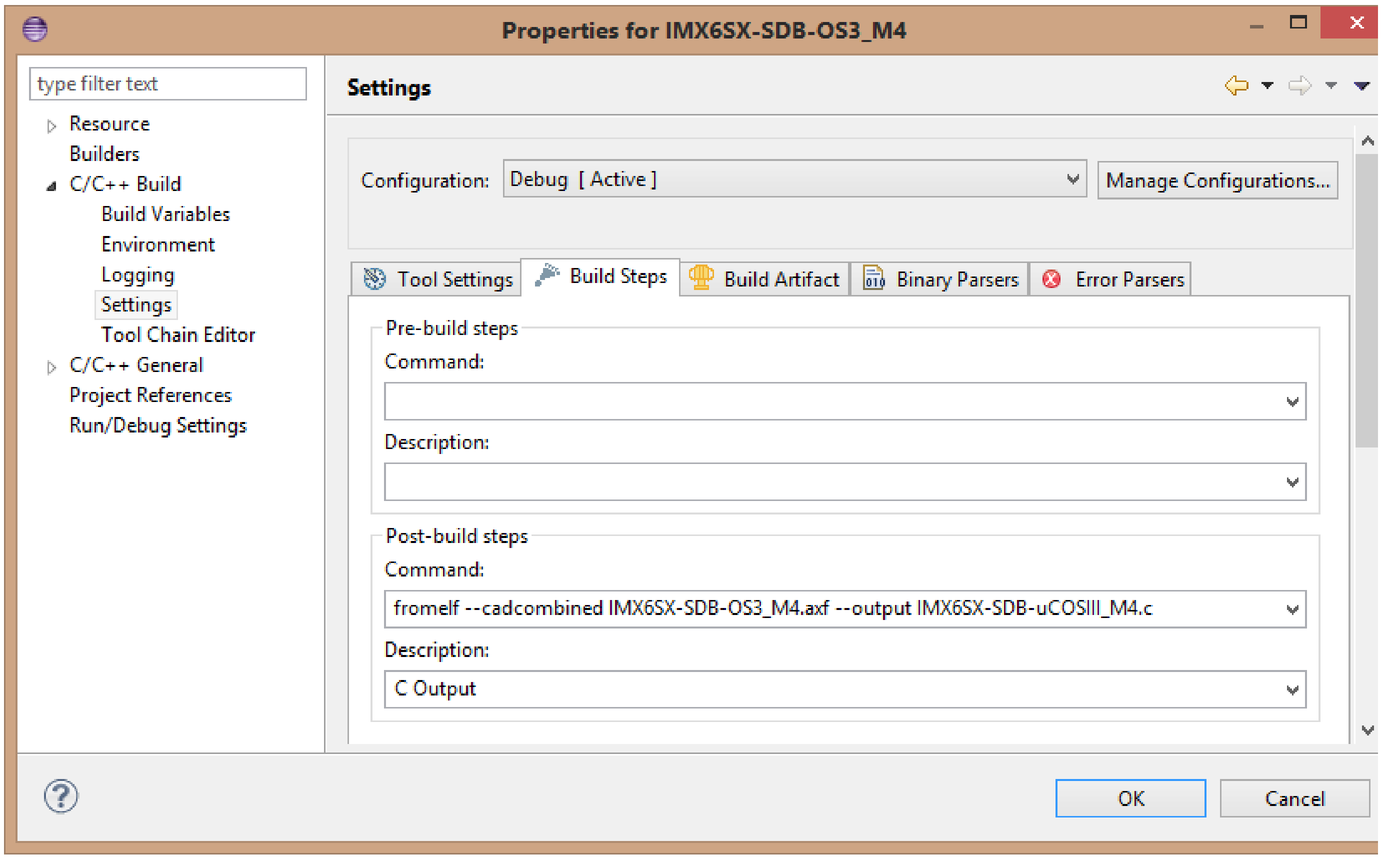
IMX6SX-SDB-OS3_M4 connected
No OS Support

Figure - Debug Control window with two open connections

Both the A9 and M4 can now be debugged at the same time.

µC/OS-III Dual Core AMP Configuration

Starting execution of the Cortex-M4 core from the A9 is relatively easy. The entry address of the M4 is taken from the starting vector configuration which is located at the start of the Cortex-M4 TCRAM. To simplify this demonstration project the M4 project in included in the A9 project as a C library. Conversion to a C array of a binary elf file can be done with the fromelf utility distributed with the ARM Compiler Toolchain. The utility can be invoked as a post build step from the DS-5 IDE as shown in the figure below :


Properties for IMX6SX-SDB-OS3_M4

Settings

Configuration: Debug [Active]

Pre-build steps

Command:

Description:

Post-build steps

Command:

```
fromelf --cadcombined IMX6SX-SDB-OS3_M4.axf --output IMX6SX-SDB-uCOSIII_M4.c
```

Description: C Output

Figure - M4 Post-build Step Configuration

The generated file IMX6SX-SDB-uCOSIII_M4.c is then imported into the A9_M4 project as a normal source file. The compiled source is fixed by the scatter file by adding a section for the internal SRAM. Note that the address of the M4 TCRAM is mapped differently in the A9.

```
M4_TCRAM 0x007f8000 0x10000
{
    M4_CODE +0
    {
        IMX6SX-SDB-uCOSIII_M4.o(+RW)
    }
}
```

Listing - A9_M4 Scatter File Addition

When configured this way it's important to wait until scatter loading is complete before activating the M4. In the example project this is done from the first user task, AppTaskStart.

```
#define  SRC  (*((CPU_REG32 *)0x20D8000))                       /* System reset
control.                                */

static  void  AppTaskStart (void *p_arg)
{
    OS_ERR       os_err;

    BSP_OS_TmrTickInit(1000u);

    BSP_Ser_Init();

    APP_TRACE_INFO(("\r\n"));
    APP_TRACE_INFO(("Application start on the Cortex-A9!\r\n"));

    /* Enable and start the Cortex-M4                       */
    APP_TRACE_INFO(("Starting the Cortex-M4\r\n"));
    DEF_BIT_SET(SRC, DEF_BIT_22);
    CPU_MB();
    DEF_BIT_CLR(SRC, DEF_BIT_04);
    CPU_MB();

...
...
```

Listing - M4 Startup Sequence

Troubleshooting and Known Issues

Can't connect to the A9

Debugging from DS-4 seems to be problematic from a cold boot. If this happens loading a valid boot image through the sdcard slot (Such as the provided Linux pre-built image) will alleviate this problem.