Heterogeneous multi-processing with Linux and the CMSIS-DSP library

DS-MDK Tutorial

AN290, September 2016, V 1.1

feedback@keil.com

Abstract

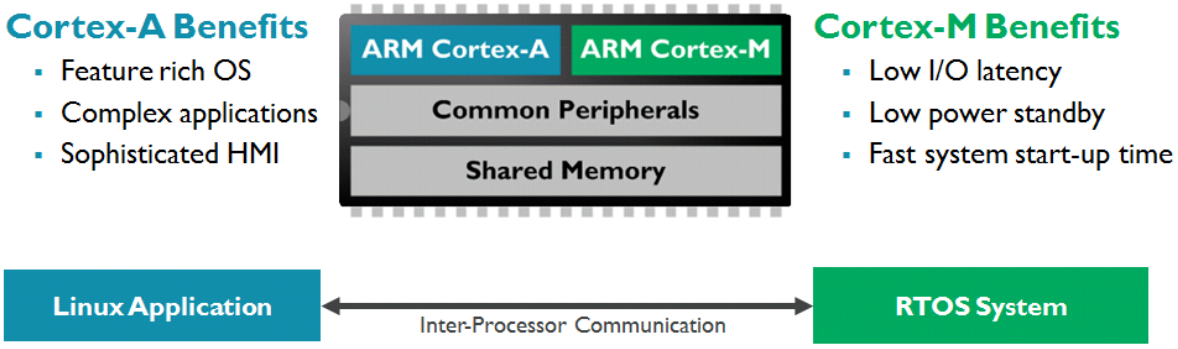
This Application note shows how to use DS-MDK to debug a typical application running on heterogeneous device: a bare-metal DSP algorithm running on an ARM® Cortex®-M core together with a Linux application running on one or more ARM Cortex-A cores. This document describes the steps necessary to build and debug the application on the NXP i.MX7 SABRE board.

Contents

Abstract 1
Introduction 1
Software Stack 3
  Prerequisites 3
Connecting the board 4
Copying the CMSIS project 5
  Stop Linux boot 6
  Configure CMSIS DS-5 Debugger 6
Copy the Linux application example 8
Set up Remote System Connection 9
Debug Linux application 9
Running a Python Web Server 10
Appendix 10

Introduction

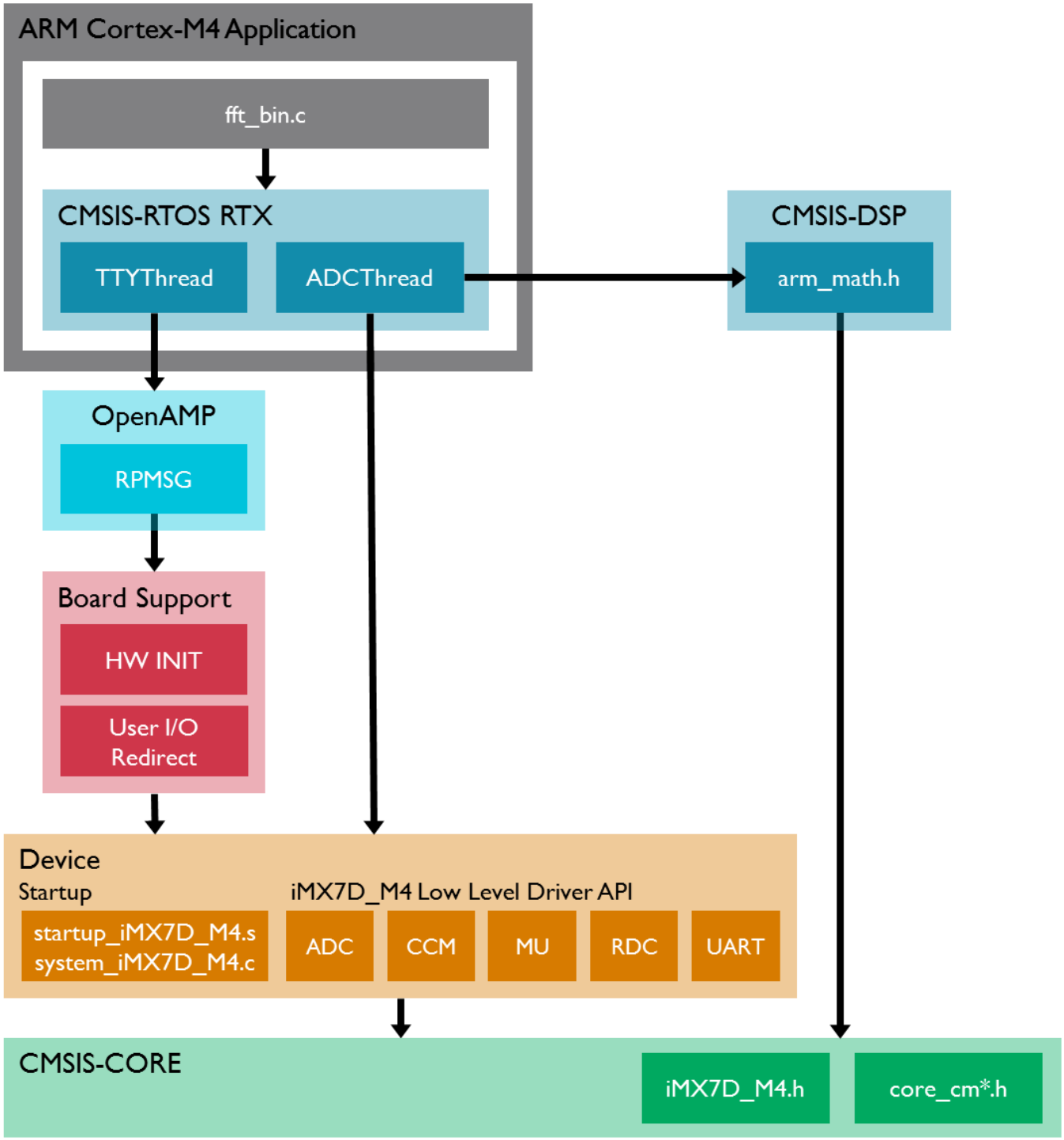
Heterogeneous systems combine the best of both worlds: one or more ARM Cortex-A based processors able to run a feature rich OS and complex applications together with an ARM Cortex-M based processor for fast start-up time and low I/O latency.



Copyright © 2016 ARM Ltd. All rights reserved

NXP provides several heterogeneous devices, such as the i.MX 6SoloX (ARM Cortex-A8/Cortex-M4) or the i.MX 7 (ARM Cortex-A7/Cortex-M4). This application note explains how to use DS-MDK to create and debug applications making use of both processors on an NXP i.MX 7 device.

The application acquires an analog signal from an external source and uses a DSP algorithm to compute the frequency of the signal. Because of the real time nature of the task, the calculation is done on the Cortex-M core.



Frequency calculation application running on the ARM Cortex-M4

Copyright © 2016 ARM Ltd. All rights reserved

Software Stack

The application uses CMSIS-RTOS RTX and the CMSIS-DSP library.

CMSIS-RTOS RTX is a real-time operating system that is part of MDK and adheres to the CMSIS specification. It is used to control the application in real-time.

The **CMSIS-DSP** library provides a set of functions that are specifically designed to support DSP algorithms. The *ADCThread* in the fft_bin.c files uses some of these functions for analyzing the input signal to detect its frequency.

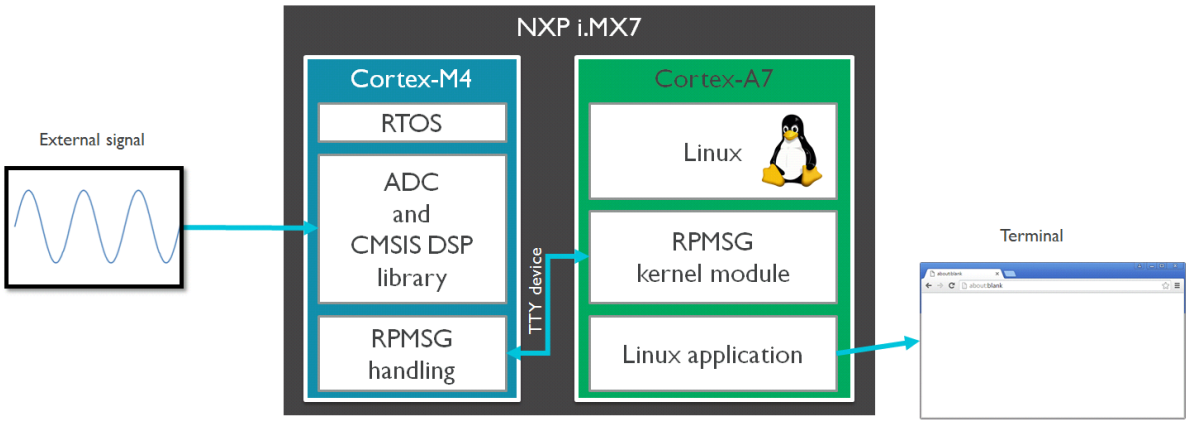
OpenAMP provides an open source framework that allows asymmetric multiprocessing applications to leverage parallelism offered by the multicore configuration. RPMSG is used to send messages from one core to the other.

The **board support** files enable the user to quickly develop code for the hardware that is used here. Here, it is used to setup the USARTs of the device to enable printf-style output on the serial terminal.

The **device family pack (DFP)** contains the support for the device in terms of startup and system code, and a device family specific software framework with hardware abstraction layer (HAL).

The basis for the software framework is **CMSIS-Core** that implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals. The device header files adhere to the CMSIS-Core standard and help the user to access the underlying hardware.

The application shows the calculated value of the frequency on the ARM Cortex-M4 terminal and through a user-space Linux application running a web server.



Prerequisites

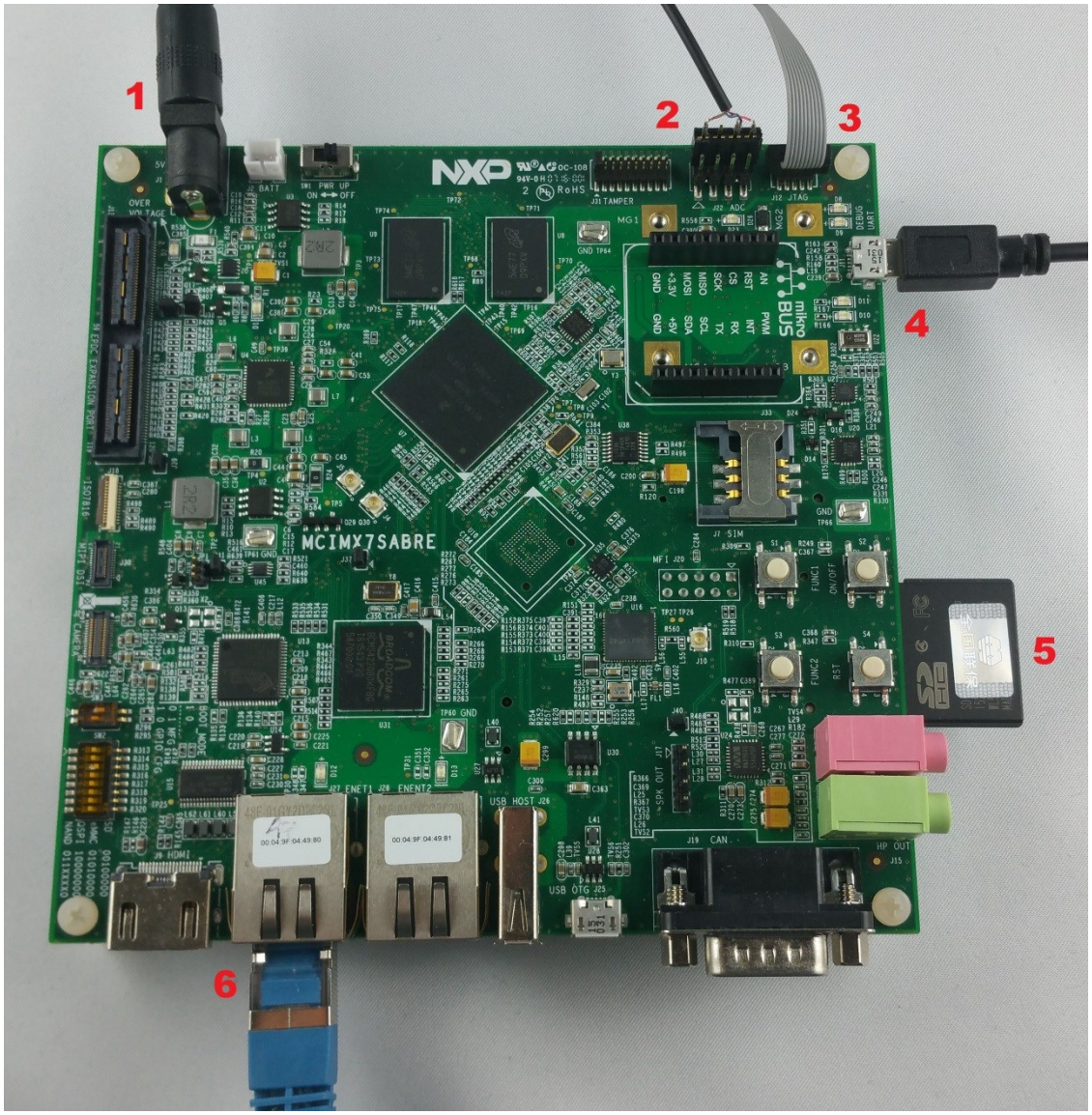
The Getting Started with DS-MDK user's guide available at www.keil.com/mdk5/ds-mdk/install teaches the basic concepts of working with DS-MDK. This Application Note assumes the SD-Card contains already a working Linux image. A working image for the NXP i.MX7 SABRE board is available on www.keil.com/mdk5/ds-mdk/imx7reference/

In order to generate the input signal use a professional signal generator or any other source of sine signal.

Copyright © 2016 ARM Ltd. All rights reserved

Connecting the board

This application note is using the NXP i.MX7 SABRE board. The following picture illustrates how the various elements of the board need to be connected in order for the application to run correctly.

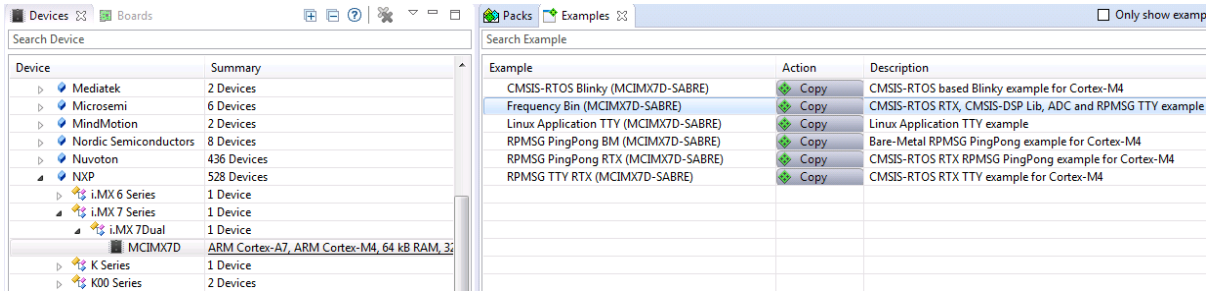


1. **Power supply**: connect the power supplier included in the package.
2. **Analog Input to the ADC**: the **ADC** header provides input to the i.MX 7
3. **JTAG debug port**: connect the JTAG connector of the Keil debug adapter ULINK2 Pro.
4. **Debug UART**: connect the USB port to your computer for the virtual serial ports used for debug purpose.
5. **SD-Card slot**: insert the SD-Card with the Linux image you previously prepared or downloaded from www.keil.com/mdk5/ds-mdk/imx7reference/
6. **Ethernet port**: Linux application debug via gdbserver uses a TCP connection. The Ethernet port needs to be connected to our host machine via a cross Ethernet cable or via a Switch device.

Copyright © 2016 ARM Ltd. All rights reserved

Copying the CMSIS project

1. Launch **DS-MDK**
2. Click on **CMSIS Pack Manager** icon on the top-right side of the window.
   a. If not visible, click on the **Window – Open perspective – Other**
   b. Select **CMSIS Pack Manager** and click **OK**.
3. Select **NXP – i.MX7 Series – i.MX7 Dual – MCIMX7**
4. Click on the **Examples** tab and select "**Frequency Bin (MCIMX7-SABRE)**"



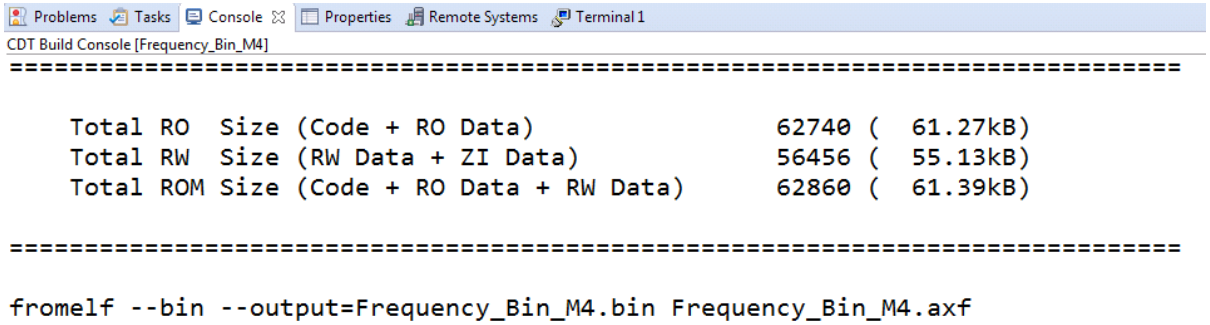
5. Click on the **Copy** button and confirm with **Copy**.

The project should be now available in your workspace. The *fft_bin.c* file contains all the code necessary to acquire the analog signal through the ADC and calculate the frequency of the signal. In particular, in the function *ADCThread()* is possible to find the calls to the **CMSIS-DSP** functions necessary to process the signal:

```
arm_cfft_f32(&arm_cfft_sR_f32_len2048, input, 0, 1);

/* Process the data through the Complex Magnitude Module for
                    calculating the magnitude at each bin */
arm_cmplx_mag_f32(input, output, LENGTH_SAMPLES/2);

/* Remove DC component */
output[0]=0;
output[1]=0;

/* Calculates maxValue and returns corresponding BIN value up to half of sample rate */
arm_max_f32(output, LENGTH_SAMPLES/2, &maxValue, &maxIndex);
```

The project is ready to be built with **ARM Compiler 5**, just right click on the project and select **Build Project.**

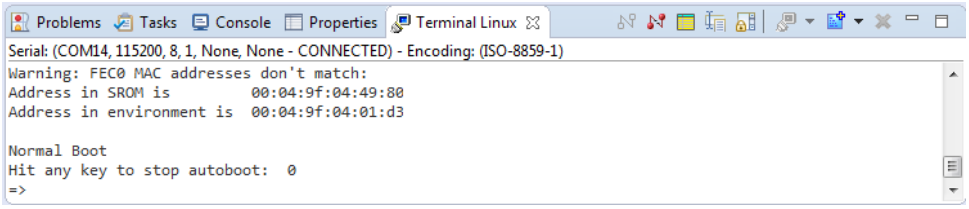
At the end of the build, the Console View should show the size of the binaries as below.



Copyright © 2016 ARM Ltd. All rights reserved

Stop Linux boot

1. Go to **Window → Show View → Other…** to open a *Terminal* view.
2. Select **Terminal → Terminal** and click **OK**.
3. Open the settings dialog from the toolbar of the **Terminal 1** window, set the following and click OK:
   - View Title: Terminal Linux
   - Connection Type: Serial
   - Port: Use the first of the virtual COM ports associated with the NXP SABRE board
   - Baud Rate: 115200
4. Press the reset button on the development board to observe the boot process in the *Terminal* window
5. Press any keyboard key to interrupt the boot process:

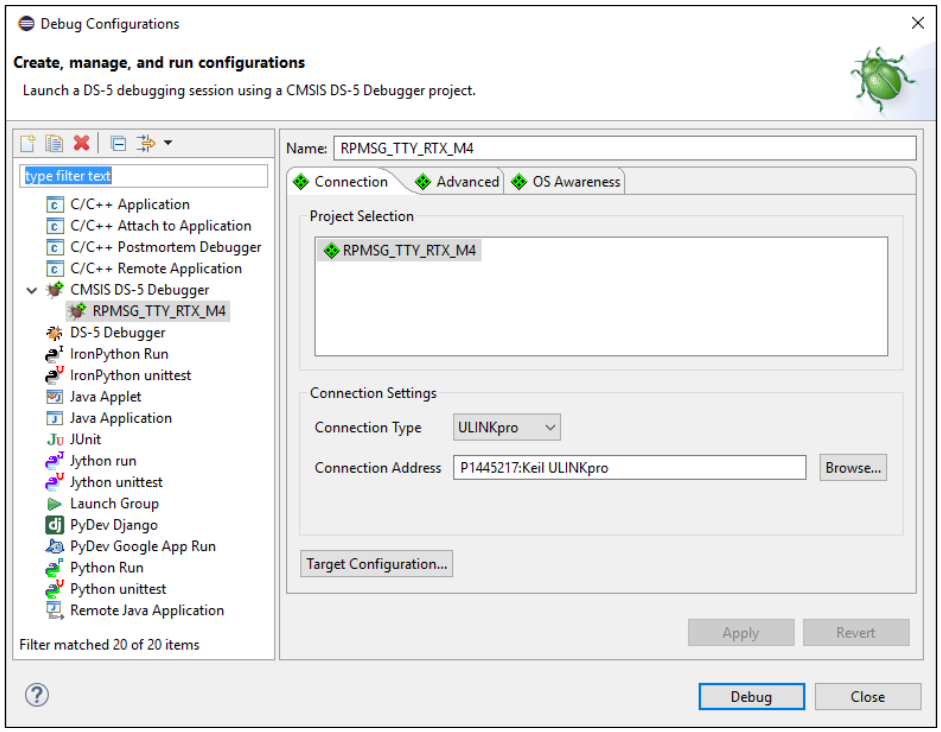


NOTE

You must halt the boot loader at this point to be able to connect the ULINK*pro debug adapter to the ARM Cortex-M processor.*

Configure CMSIS DS-5 Debugger

Right-click the **RPMSG_TTY_RTX_M4** project and select **Debug As → CMSIS DS-5 Debugger** to launch the debug configurations dialog:



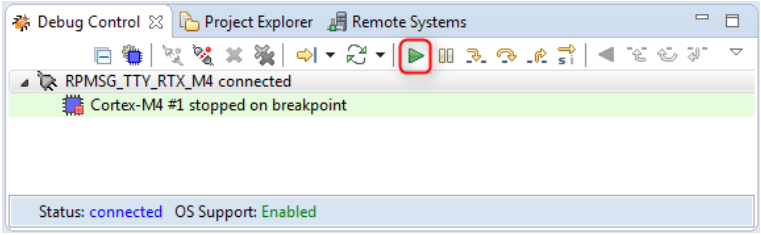
Verify the **Connection Settings** and ensure that ULINK*pro* is correctly detected. If in doubt, use **Browse…** to list available debug adapters.

Click on the **Debug** button to start the debugging session

DS-MDK switches to the DS-5 Debug perspective. The application loads and runs until main.

Copyright © 2016 ARM Ltd. All rights reserved

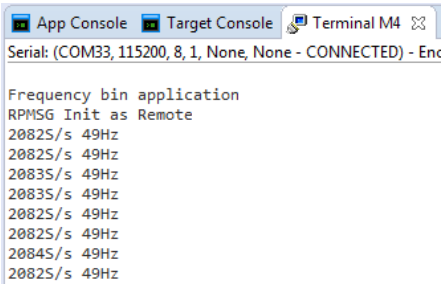
Click on the Run button to start the Cortex-M4 application click **Run** in the *Debug Control* view.



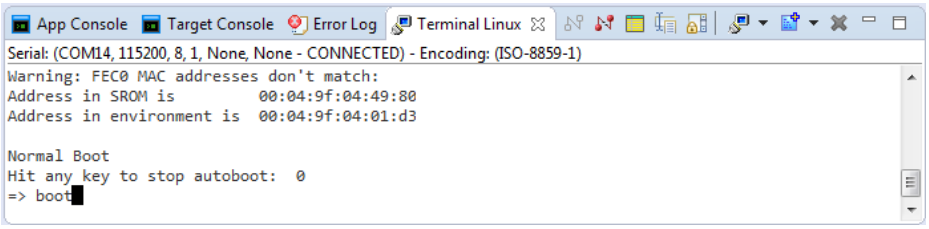
NOTE

To observe the output of the ARM Cortex-M4, add another Terminal view to the debug perspective using the same serial port setting on the next available USB COM port by using ***Window → Show View → Terminal****.*

Observe the output of the application in the *Terminal M4* window.



In the *Terminal Linux* enter "boot" to start the Linux system:



When the boot process has finished, log in as **root** (no password required).

In the *Terminal Linux*, load the kernel module that communicates with the Cortex-M4 application with this command:

```
root@imv7dsabresd:~# modprobe -v imx_rpmsg_tty
```

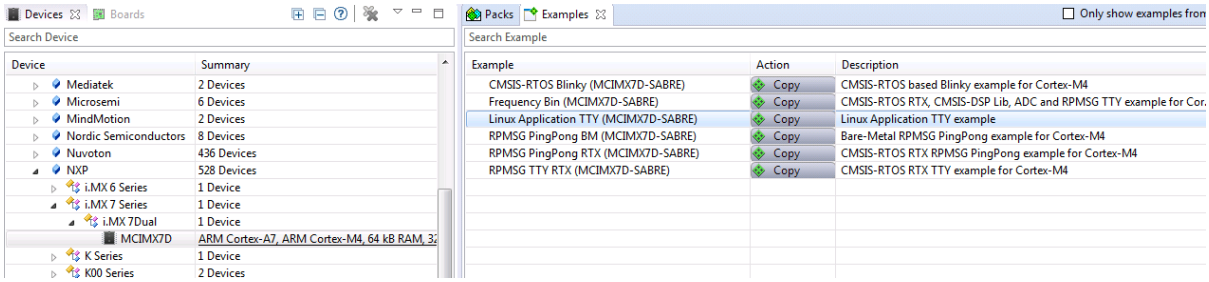
The kernel module will be loaded:

```
insmod /lib/modules/4.1.15-1.1.0+ga4d2a08/kernel/drivers/rpmsg/imx_rpmsg_tty.ko
imx_rpmsg_tty rpmsg0: new channel: 0x400 -> 0x0!
Install rpmsg tty driver!
```


Copyright © 2016 ARM Ltd. All rights reserved

Copy the Linux application example

1. Click on **CMSIS Pack Manager** icon on the top-right side of the window.
    a. If not visible, click on the **Window – Open perspective – Other**
    b. Select **CMSIS Pack Manager** and click **OK**.
2. Select **NXP – i.MX7 Series – i.MX7 Dual – MCIMX7**
3. Click on the **Examples** tab and select "**Linux Application TTY** (**MCIMX7-SABRE)**



4. Click on the **Copy** button and confirm with **Copy**.

Similarly with the Frequency Bin project, the Linux application project is now copied in the workspace and can be built in the same way with Right Click on the project and then **Build Project**. This time the compiler used is GNU GCC for Linux.

Let's take a look at the code of the Linux application. The main function contains all the necessary steps to read the message sent by the Cortex-M counterpart. The RPMsg kernel module creates a virtual serial port in /dev/ttyRPMSG. The code would just need to open the serial port, write a string to it and just read the message sent back from the Cortex-M firmware.

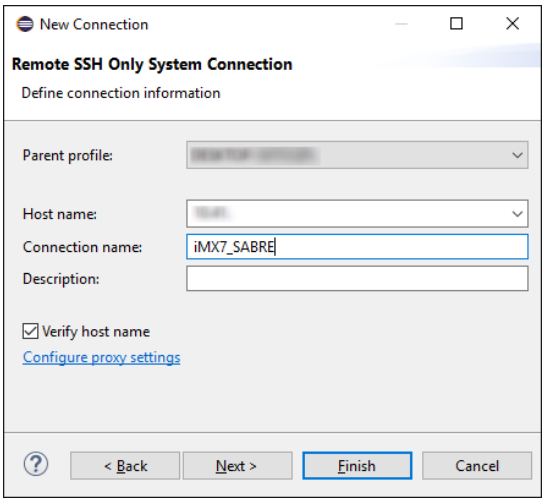
```
char *portname = "/dev/ttyRPMSG";

int fd = open (portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
    printf ("Error %d opening %s: %s", errno, portname, strerror (errno));
    return -1;
}

set_interface_attribs (fd, B115200, 0);

write (fd, "Hello from A7!", 14);
usleep (10000);

char buf[14];
read (fd, buf, sizeof buf);
printf ("Get Message From Remote Side: %s", buf);
```

The message is then finally printed on the terminal console: within DS-MDK, this can be read within the App Console.

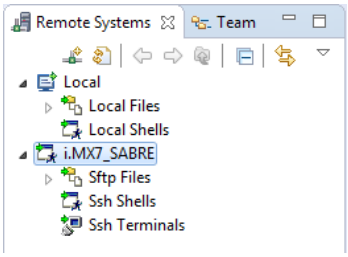
Copyright © 2016 ARM Ltd. All rights reserved

Set up Remote System Connection

1. Go to **Window → Open Perspective → Other...**, then select **Remote System Explorer**
2. Use the button to create a new connection.
3. Select **SSH Only** and click **Next**.
4. RSE communicates with the target using TCP/IP. Enter the target's IP address into the **Host Name** field. Enter a meaningful name in the **Connection name** box:

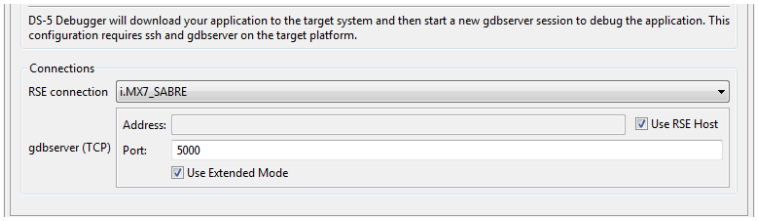


1. Click **Finish** to show your connection in the **Remote Systems** window:



Debug Linux application

1. Right-click on the project **Linux Application TTY** and select **Debug As → Debug Configurations…**
2. In the *Debug Configurations* window, select DS-5 Debugger and then press the icon to create a new debug configuration.
3. Name it `GDB Debug` and select in the *Connection* tab **Linux Application Debug → Application Debug → Connections via gdbserver → Download and debug application**.
4. The RSE connection from the previous step shows up:



5. Click **Debug**.
6. Use the **Continue** button to run the Linux application.

Copyright © 2016 ARM Ltd. All rights reserved

Running a Python Web Server

One of the advantages of using a Linux system is the wide variety of open source software available for this platform. For example, if we wanted to extend our application with a Web Server to show the frequency of the signal on a browser, we could either write our own web server in C/C++ or we could use a higher level framework. For our application, we used Flask (http://flask.pocoo.org/), a minimal web server written in Python which fits perfectly with our requirements.

Appendix

Getting Started with DS-MDK user's guide: www.keil.com/mdk5/ds-mdk/install
Linux kernel and source for the NXP i.MX 7 SABRE: www.keil.com/mdk5/ds-mdk/imx7reference/
Kernel.org: http://www.kernel.org/doc/Documentation/arm/Booting
Debugging with scripts: https://developer.arm.com/docs/dui0446/latest/debugging-with-scripts
Debug configurations: https://developer.arm.com/docs/dui0446/latest/ds-5-debug-perspectives-and-views/debugconfigurations-debugger-tab

Copyright © 2016 ARM Ltd. All rights reserved