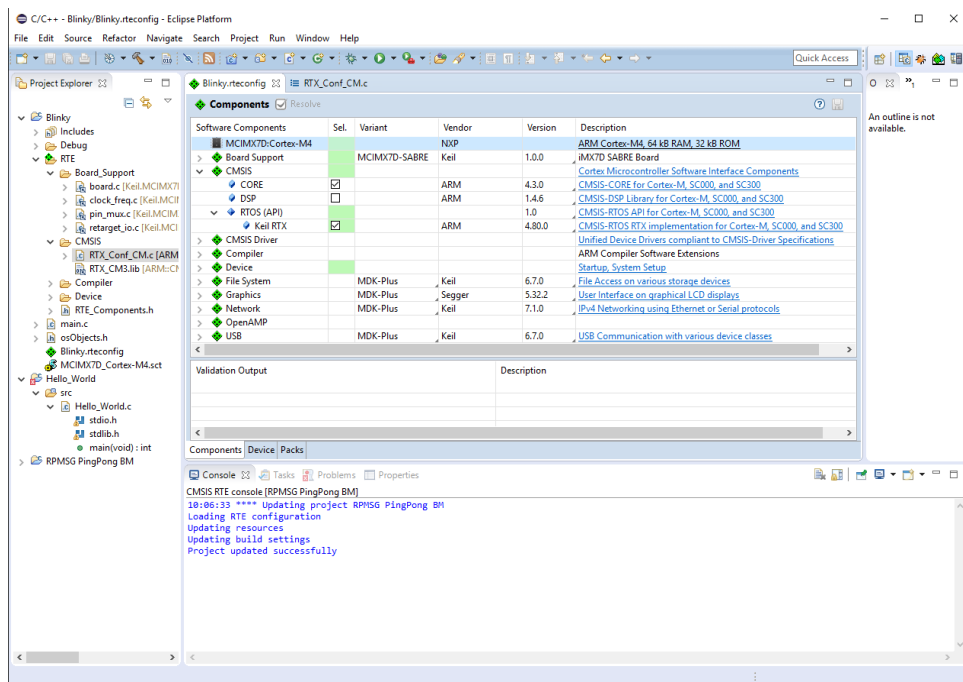


Getting Started with DS-MDK

Create Applications for Heterogeneous
ARM[®] Cortex[®]-A/Cortex-M Devices



Information in this document is subject to change without notice and does not represent a commitment on the part of the manufacturer. The software described in this document is furnished under license agreement or nondisclosure agreement and may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than for the purchaser's personal use, without written permission.

Copyright © 1997-2016 ARM Germany GmbH
All rights reserved.

ARM, Keil, μ Vision, Cortex, and ULINK are trademarks or registered trademarks of ARM Germany GmbH and ARM Ltd.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Eclipse is a registered trademark of the Eclipse Foundation, Inc.

NOTE

We assume you are familiar with Microsoft Windows, the hardware, and the instruction set of the ARM® Cortex®-A and Cortex-M processors.

Every effort was made to ensure accuracy in this manual and to give appropriate credit to persons, companies, and trademarks referenced herein.

Preface

Thank you for using the DS-MDK Development Studio available from ARM. To provide you with the very best software tools for developing ARM based embedded applications we design our tools to make software engineering easy and productive. ARM also offers therefore complementary products such as the ULINK™ debug and trace adapters and a range of evaluation boards. DS-MDK is expandable with various third party tools, starter kits, and debug adapters.

Chapter Overview

The book starts with the installation of DS-MDK and describes the software components along with complete workflow from starting a project up to debugging on hardware. It contains the following chapters:

DS-MDK Introduction provides an overview about the DS-MDK, the software packs, and describes the product installation along with the use of example projects.

Eclipse IDE explains the basic concepts of the IDE and the most frequently used perspectives.

Create Cortex-M Applications guides you through the process of creating and modifying projects using CMSIS and device-related software components for the Cortex-M microcontroller.

Create Linux Applications shows you how to create and modify applications for the Cortex-A processor running Linux.

Debug Applications describes the process of how to connect to the target hardware and explains debugging applications on the target.

Store Cortex-M Image gives further details on how to store the application image on the target and how to run it at start up time.

Contents

Preface.....	3
DS-MDK Introduction.....	7
Solution for Heterogeneous Systems.....	7
DS-MDK Licensing.....	8
License Types.....	8
Installation.....	9
Software and Hardware Requirements.....	9
Install DS-MDK.....	9
Manage Software Packs.....	11
Install the Linux Image.....	12
Hardware Connection.....	13
Verify Installation with Example Projects.....	14
Documentation and Support.....	17
Eclipse IDE.....	18
Perspectives.....	18
C/C++ Perspective.....	19
CMSIS Pack Manager Perspective.....	22
Remote System Explorer Perspective.....	23
DS-5 Debug Perspective.....	24
Create Cortex-M Applications.....	25
Blinky with CMSIS-RTOS RTX.....	25
Setup the Project.....	26
Select Software Components.....	28
Configure CMSIS-RTOS RTX Kernel.....	29
Create the Source Code Files.....	30
Adapt the Scatter File.....	32
Build the Cortex-M Image.....	33
Create Linux Applications.....	34
Setup the Project.....	34
Build the Application Image.....	35
Debug Applications.....	36
Prepare Terminal Views.....	37
Debug Cortex-M Application.....	39
Stop in U-Boot.....	39
Configure CMSIS DS-5 Debugger.....	40
Run Cortex-M Application.....	42
Debug Linux Application.....	42

Setup RSE Connection	43
Boot Linux	43
Configure DS-5 Debugger	44
Run the Linux Application	46
Store Cortex-M Image	47
Create a Cortex-M Binary Image (BIN)	47
Store Cortex-M BIN File on SD Card	48
Run Cortex-M BIN File from U-Boot	49
Index	50

NOTE

This user's guide describes how to create applications with the Eclipse-based DS-MDK IDE and Debugger for ARM Cortex-A/Cortex-M based NXP i.MX 6 and 7 series.

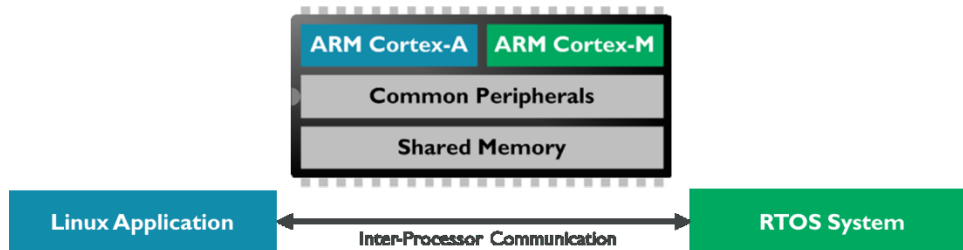
*Refer to the **Getting Started with MDK** user's guide for information how to create projects for ARM Cortex-M microcontrollers with the μ Vision® IDE/Debugger.*

DS-MDK Introduction

DS-MDK combines the Eclipse-based DS-5 IDE and Debugger with CMSIS-Pack technology and uses software packs to extend device support for devices based on 32-bit ARM Cortex-A processors or heterogeneous systems based on 32-bit ARM Cortex-A and ARM Cortex-M processors.

Initially, only NXP i.MX 6 and 7 series devices are supported that combine computing power for application-rich systems with real-time responsiveness. For such embedded systems, the DS-5 Debugger gives visibility to multi-processor execution and allows optimization of the overall software architecture.

Solution for Heterogeneous Systems



Heterogeneous systems usually consist of a powerful ARM Cortex-A class application processor and a deterministic ARM Cortex-M based microcontroller. These systems combine the best of both worlds: the Cortex-A class processor can run a feature-rich operating system such as Linux and enables the user to program complex applications with sophisticated human-machine interfaces (HMI). The Cortex-M class controller offers low I/O latency, superior power efficiency and a fast system start-up time for embedded systems.

Usually, both processors have access to a set of communication peripherals and shared memory. The biggest challenge with heterogeneous systems is the synchronization and inter-processor communication.

DS-MDK offers a complete software development solution for such systems:

- Manage Cortex-A Linux and Cortex-M RTOS projects in the same development environment.
- Use the Cortex Microcontroller Software Interface Standard ([CMSIS](#)) development flow for efficient Cortex-M programming. Add software packs any time to DS-MDK to make new device support and middleware updates independent from the toolchain. The IDE manages the provided software components that are available for the application as building blocks.
- Debug multicore software development projects with the full visibility offered by the DS-5 Debugger.

DS-MDK Licensing

DS-MDK is part of the [Keil® MDK-Professional Edition](#) and the product requires a valid license for MDK-Professional Edition.

License Types

The following licenses types are available:

Single-User License (Node-Locked) grants the right to use the product by one developer on two computers at the same time.

Floating-User License or FlexLM License grants the right to use the product on several computers by a number of developers at the same time.

For further details, refer to the *Licensing User's Guide* at www.keil.com/support/man/docs/license.

Installation

Software and Hardware Requirements

DS-MDK has the following minimum hardware and software requirements:

- *A workstation running Microsoft Windows 64-bit*
- *Dual-Core Processor with > 2 GHz*
- *4 GB RAM and 8 GB hard-disk space*
- *1280 x 800 or higher screen resolution*

Install MDK

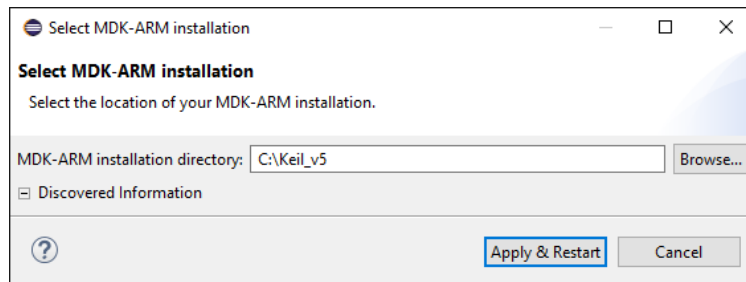
Download MDK from www.keil.com/download - Product Downloads and run the installer. It also adds the software packs for ARM CMSIS and MDK Middleware.

Follow the instructions on www.keil.com/support/man/docs/license/license_sul_install.htm to activate a MDK-Professional license, which is required for DS-MDK.

Install DS-MDK

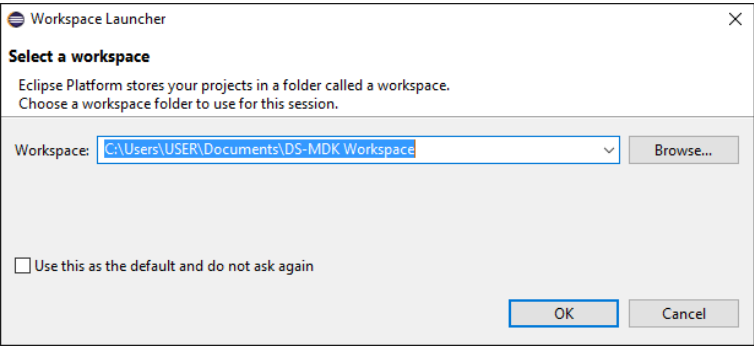
Download DS-MDK from www.keil.com/mdk5/ds-mdk/install and run the installer. To start DS-MDK, use **Eclipse for DS-MDK** from the Start menu (Windows 10: **All apps** → **ARM DS-MDK** → **Eclipse for DS-MDK**).

Initially, select your MDK installation for license purposes:

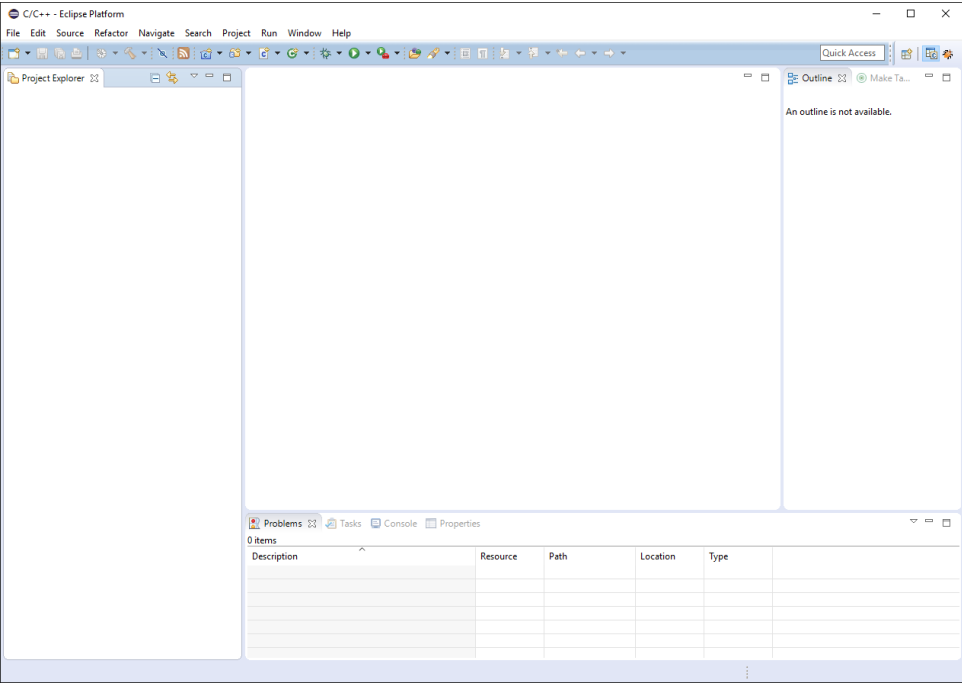


If required, change the installation destination.

Specify a directory for your workspace (the area where your projects will be stored). For most users, the default suggested directory is the best option.



The Eclipse-based IDE opens in the **C/C++ Perspective**:




NOTE

Refer to chapter *Eclipse IDE* on page 18 for more information on Eclipse workbench concepts.

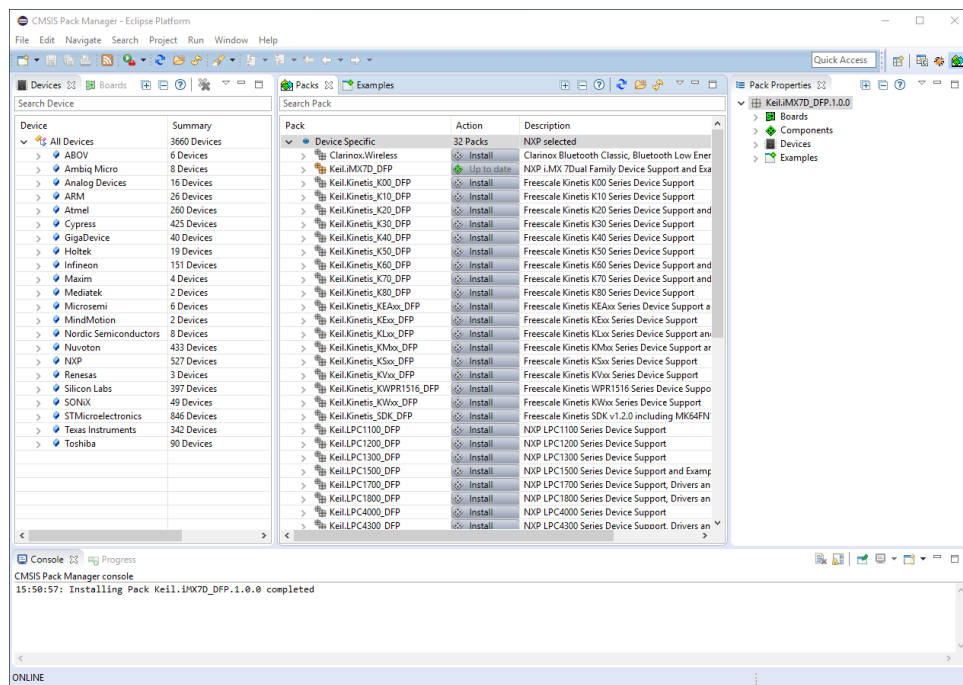
Manage Software Packs

Use the **CMSIS Pack Manager** perspective to manage software packs on the local computer.


Use  **Window → Open Perspective → CMSIS Pack Manager** to open this perspective. Install the software packs related to your target device or evaluation board.

NOTE

Currently, only software packs for the NXP i.MX 6 and 7 series are qualified for DS-MDK.



The **Console** window shows information about the Internet connection and the installation progress.

TIP: The device database at www.keil.com/dd2 lists all available devices and provides download access to the related software packs. If the Pack Manager cannot access the Internet, you use the **Import existing packs** icon  or double-click on *.PACK files to manually install software packs.

Install the Linux Image

Currently, DS-MDK supports the following development board:

- NXP i.MX 7 SABRE development board: MCIMX7SABRE

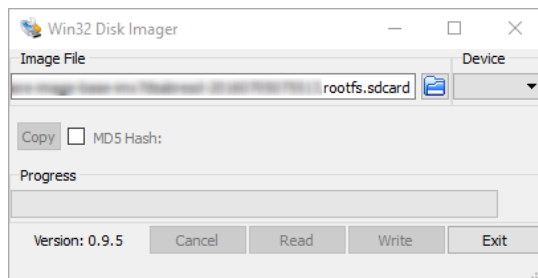
For this development board, a pre-configured Linux image with DS-MDK specific debug settings is available. Please download the zipped image file here: www.keil.com/mdk5/ds-mdk/imx7reference

This website contains documentation that explains all steps to create a Linux image for the MX7DSABRESB board to be able to debug applications with *ULINKpro*.

Copy the Linux Image to an SD-Card

Once you have downloaded the zipped Linux Kernel image, unzip it before you can flash it onto an SD-Card. Use the open source tool **Win32 Disk Imager** from <http://win32diskimager.sourceforge.net/>.

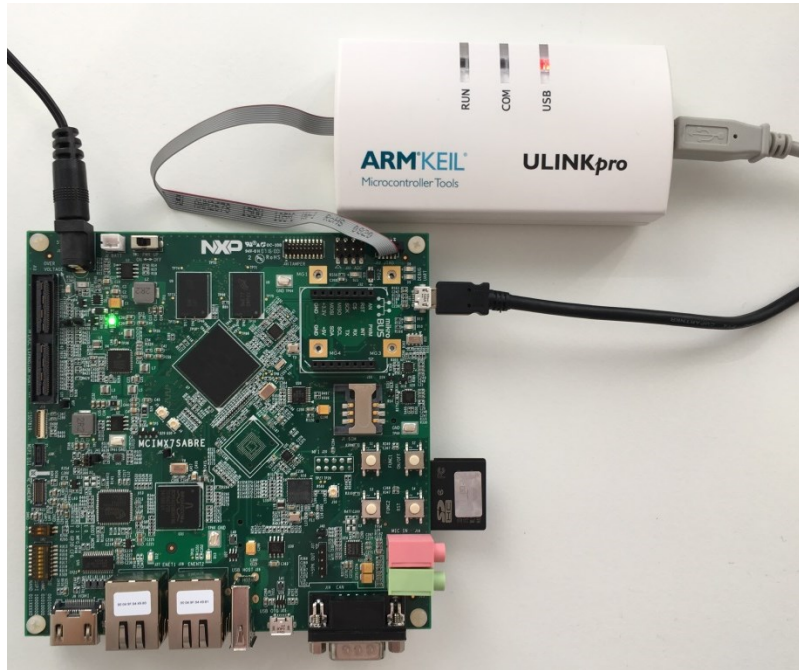
Install and run the tool. To write the image to the memory card, specify the location of the image file, select the **Device** letter of the SD card and press the **Write** button:



Hardware Connection

i.MX 7 SABRE Board

- Insert the SD-Card with the Linux image into the slot labelled **SD1 BOOT**.
- Use the 10-pin ribbon cable to connect the ULINK_{pro} debug adapter to **J12 JTAG**.
- Connect your computer to the USB connector labelled **DEBUG UART**. Your Windows workstation automatically detects a dual USB serial port component and installs the required drivers.
- Connect the 5V power supply to **J1**.

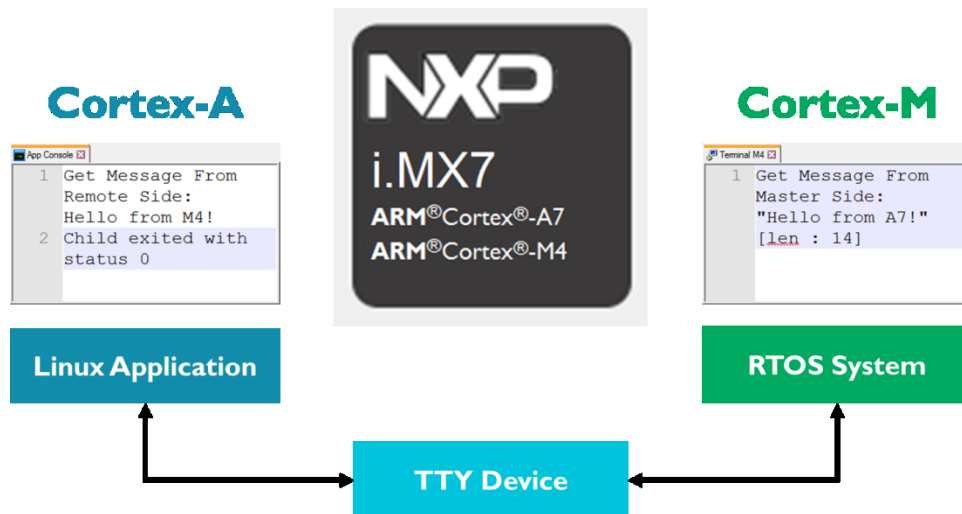


Verify Installation with Example Projects

Once you have selected, downloaded, and installed a software pack for your device, you can verify your installation using one of the examples provided in the software pack.


Remote Processor Messaging Protocol Example

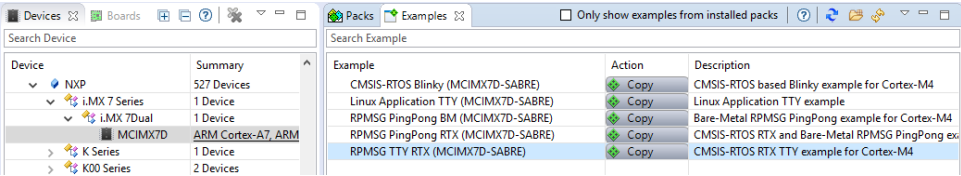
The i.MX 7 Device Family Pack contains two example projects that show how the two processors communicate with each other using the remote processor messaging protocol (RPMSG) via a TTY serial device.



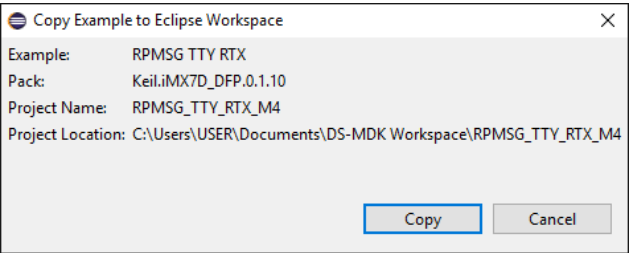
The *Linux Application TTY* runs on the Cortex-A7 processor and writes a message to the TTY device. The terminal of the *RPMSG TTY RTX* application running on the Cortex-M4 processor shows this message. The application responds on the TTY device. The Linux application reads this message and shows it in its App console.

Copy the RPMSG TTY RTX Example Project

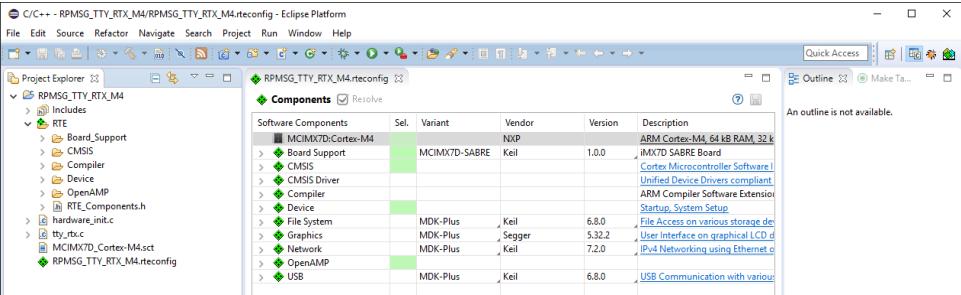
 In the **CMSIS Pack Manager** perspective, select the **Examples** tab. Use filters in the toolbar to narrow the list of examples.



Click **Copy** next to the **RPMSG TTY RTX** example. Confirm your selection:

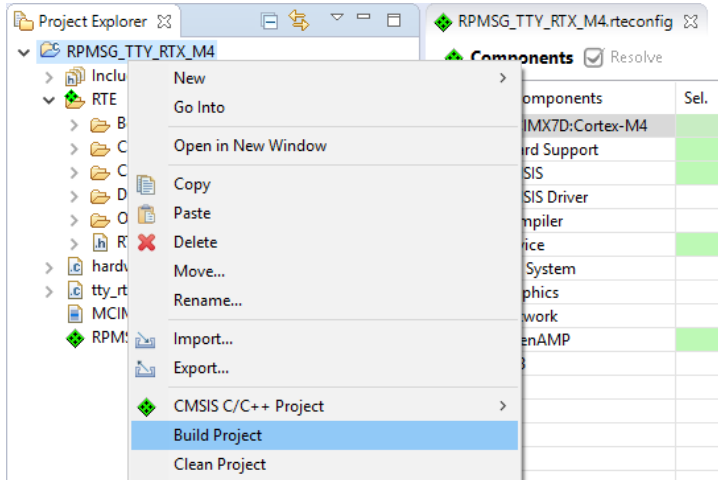


CMSIS Pack Manager copies the example into your workspace and switches to the **C/C++** perspective:

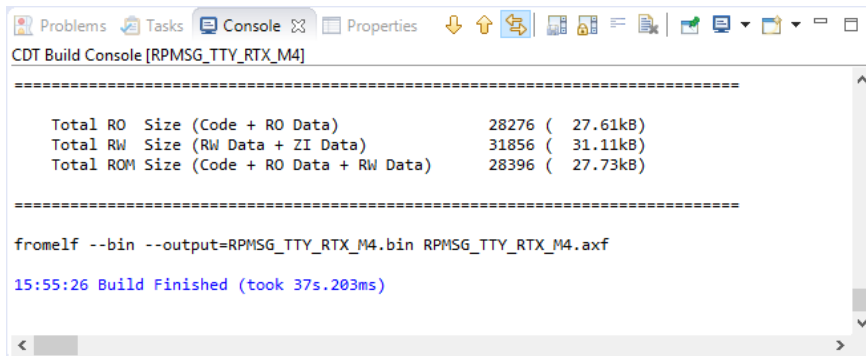


Build the Application

Build the project from the context menu in the **Project Explorer**:



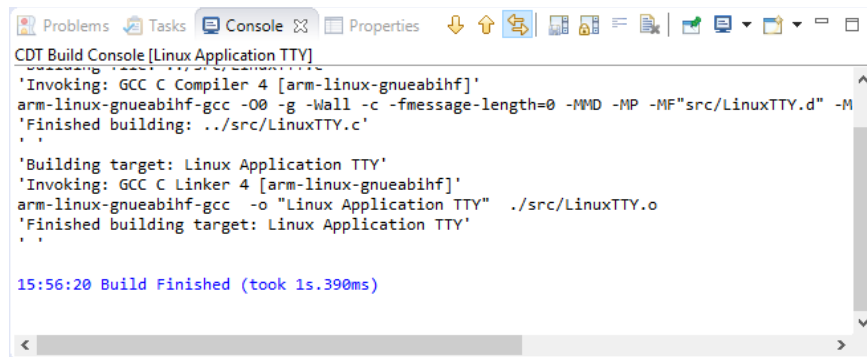
The **Console** window shows information about the build process:



Copy and Build the Linux Application TTY

Switch back to the **CMSIS Pack Manager** perspective and copy the **Linux Application TTY** example project to your workspace.

Build the project from the context menu in the **Project Explorer**. The **Console** should show an error-free build:



```
CDT Build Console [Linux Application TTY]
'Invoking: GCC C Compiler 4 [arm-linux-gnueabi]'
arm-linux-gnueabi-gcc -O0 -g -Wall -c -fmessage-length=0 -MMD -MP -MF"src/LinuxTTY.d" -M
'Finished building: ../src/LinuxTTY.c'
'Building target: Linux Application TTY'
'Invoking: GCC C Linker 4 [arm-linux-gnueabi]'
arm-linux-gnueabi-gcc -o "Linux Application TTY" ./src/LinuxTTY.o
'Finished building target: Linux Application TTY'

15:56:20 Build Finished (took 1s.390ms)
```

Continue with the chapter **Debug Applications** on page 36 that explains how to debug both applications with the DS-5 Debugger.

Documentation and Support

DS-MDK provides online manuals and context-sensitive help. The **Help** menu opens the main help system that includes the *CMSIS C/C++ Development User's Guide*, the *ARM DS-MDK Documentation*, the *RSE User Guide*, and other reference guides.

Many dialogs have context-sensitive **Help** buttons that access the documentation and explain dialog options and settings.

If you have suggestions or you have discovered an issue with the software, please report them to us. Support and information channels are accessible at www.keil.com/support.

Eclipse IDE

DS-MDK is an Integrated Development Environment (IDE) that combines the Eclipse IDE with the compilation and debug technology of ARM.

Use DS-MDK as a project manager to create, build, debug, monitor, and manage projects for ARM targets. It uses a single folder called a workspace to store files and folders related to specific projects.

Users can extend its abilities by installing plug-ins written for the Eclipse platform, such as the **CMSIS Pack Manager** and **Remote System Explorer**, included in DS-MDK.

Perspectives

DS-MDK contains multiple perspectives. Each perspective contains an initial set and layout of views that help you to create, build and debug projects. While working with DS-MDK, you will switch perspectives frequently. It is always possible to change a perspective layout and to add new views to it.

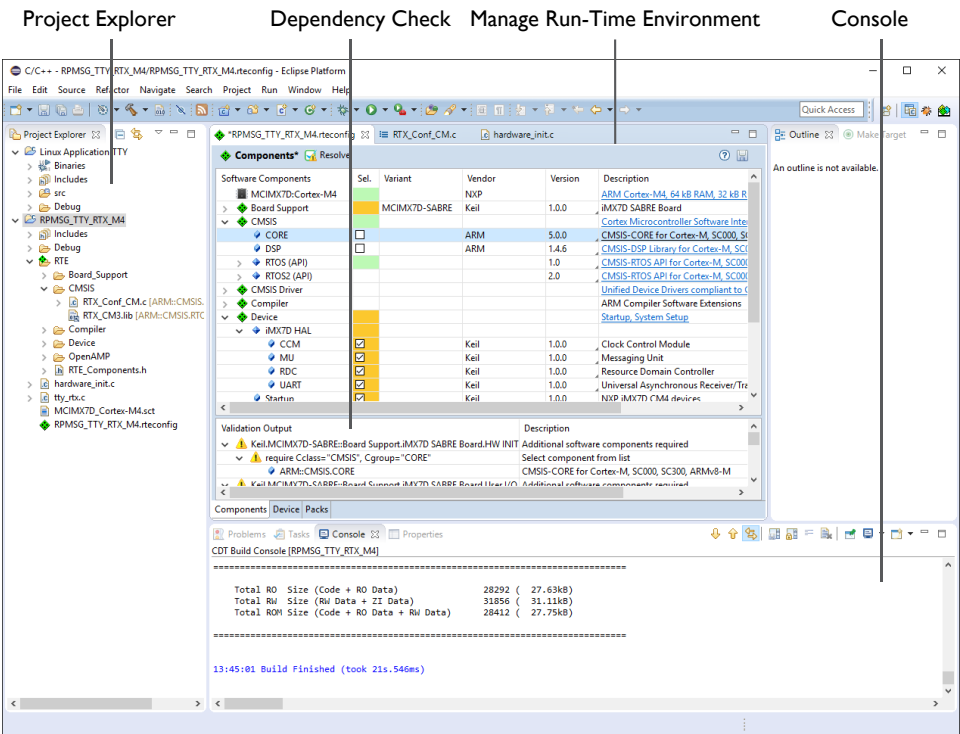
DS-MDK uses mainly these perspectives:

- **C/C++ Perspective**
- **CMSIS Pack Manager Perspective**
- **Remote System Explorer Perspective**
- **DS-5 Debug Perspective**

C/C++ Perspective

By default, this perspective consists of the Project Explorer, an editor area and views for tasks, properties, and a message console.

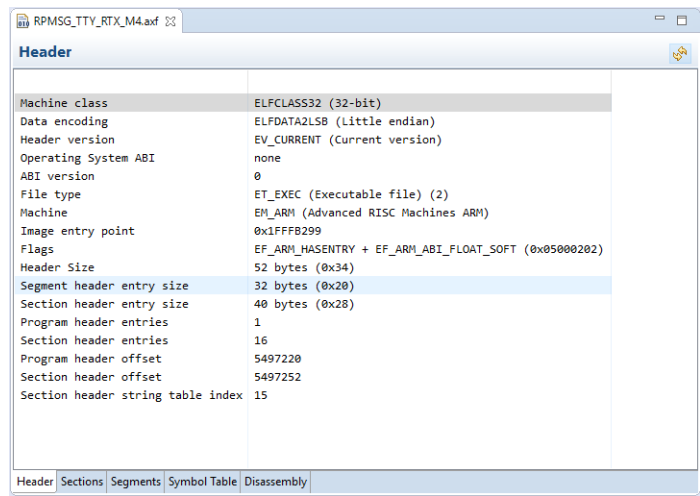
The editor area shows C/C++ source code as well as graphical representations of various configuration files such as the Run-Time Environment configuration file, the AXF file, the scatter file, and files with CMSIS configuration wizard annotations.



For more information, refer to the *C/C++ Development User's Guide* and the *CMSIS C/C++ Development User's Guide* available from the Eclipse help system (**Help** → **Help Contents**).

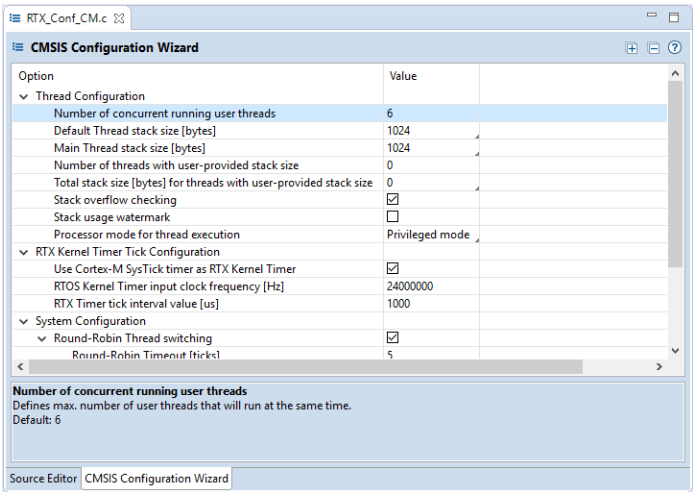
AXF File Viewer

An AXF file is the executable image generated by the ARM linker that contains object code and debug information. Open it from the Project Explorer to inspect the contents of the image.



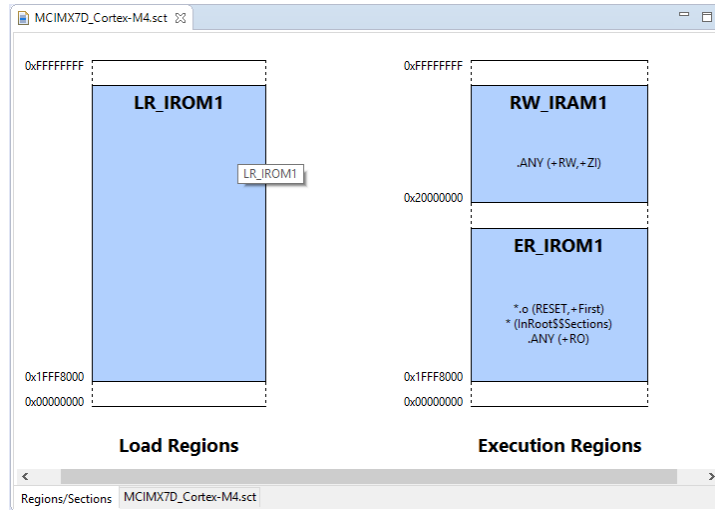
CMSIS Configuration Wizard

Right-click on a file in the Project Explorer and select **Open With → CMSIS Configuration Wizard** to modify files with CMSIS configuration wizard annotations in a graphical editor. Verify and adapt the contents directly in the graphical representation of the text file.



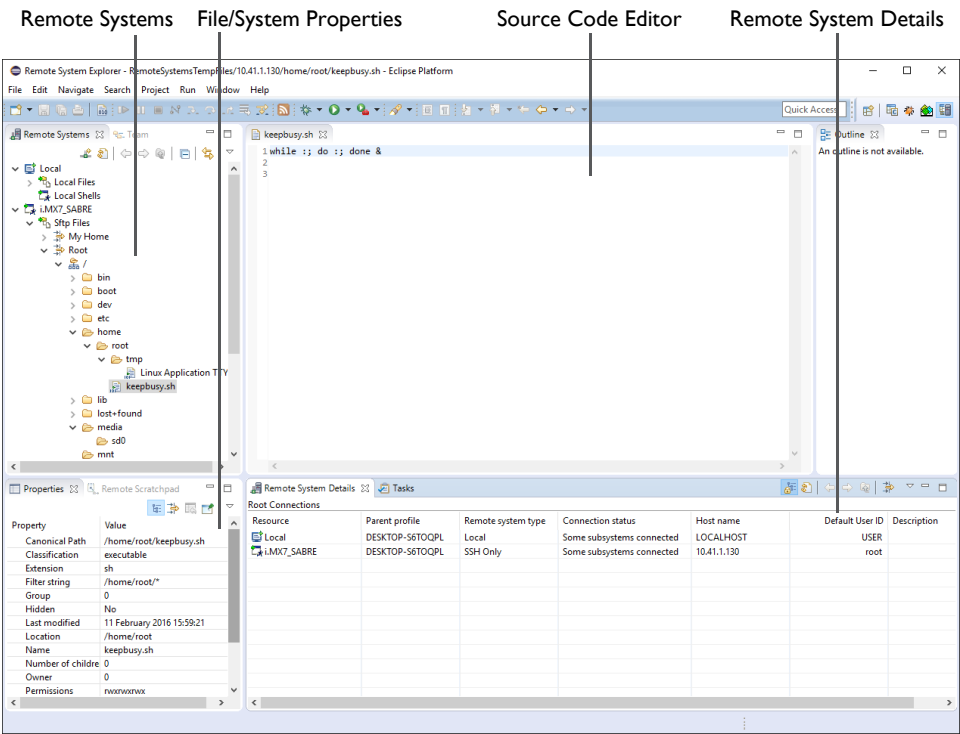
Scatter File Viewer

Scatter files (*.sct) are used to specify the memory map of an image to the linker. The **Scatter File Viewer** lets you inspect this text file in a graphical representation. Use the *filename.sct* tab to edit the scatter file contents (refer to **Adapt the Scatter File** on page 32).



Remote System Explorer Perspective

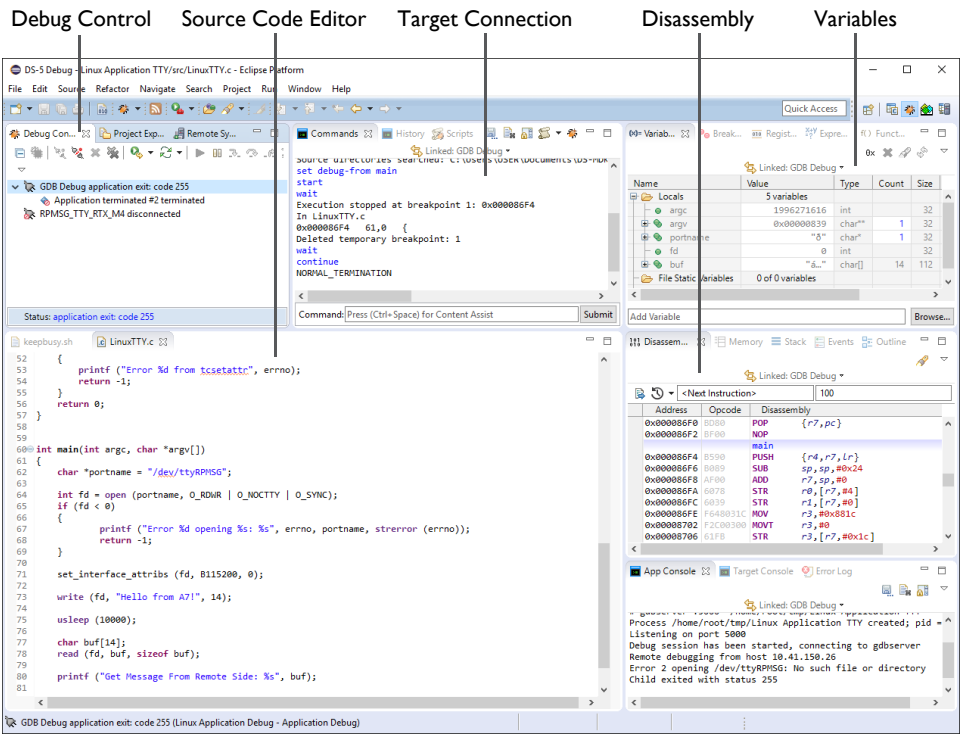
The **Remote System Explorer (RSE)** is a workbench perspective that allows you to connect and work with a variety of remote systems. With predefined plug-ins, you can look at remote file systems, transfer files between hosts, do remote search, execute commands and work with processes.



For more information, refer to the *RSE User Guide* in the Eclipse help system (**Help → Help Contents**).

DS-5 Debug Perspective

The DS-5 Debugger allows you to debug bare-metal, RTOS, and Linux applications with comprehensive and intuitive views, including synchronized source and disassembly, call stack, memory, registers, expressions, variables, threads, breakpoints, and trace.



For more information, refer to the *ARM DS-5 Debugger Documentation* in the *ARM DS-MDK Documentation* available from the Eclipse help system (**Help → Help Contents**).

Create Cortex-M Applications

This chapter guides you through the steps required to create and modify projects for the Cortex-M target in a heterogeneous system.

Blinky with CMSIS-RTOS RTX

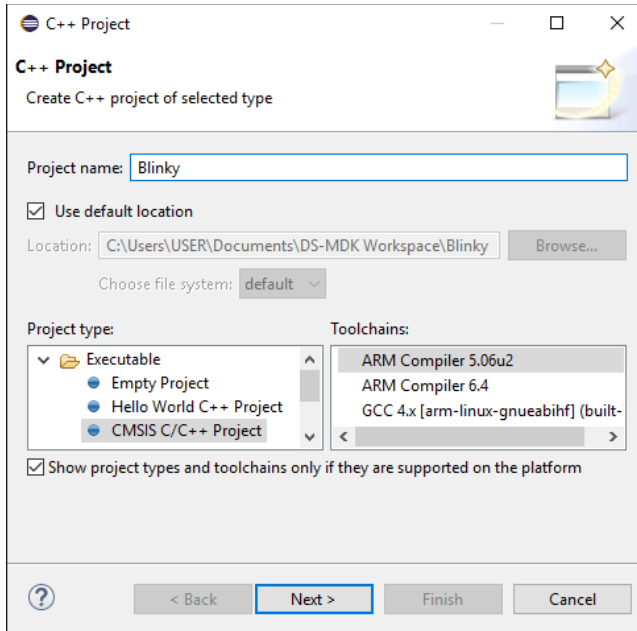
Follow these steps to create a project called *Blinky* using the real-time operating system CMSIS-RTOS RTX:

- **Setup the Project:** create a project and select the microcontroller device along with the relevant CMSIS components.
- **Select Software Components:** choose the required software components for the application.
- **Customize the CMSIS-RTOS RTX Kernel:** adapt the RTOS kernel.
- **Create the Source Code Files:** add and create the application files.
- **Build the Application Image:** compile and link the application.

For the *Blinky* project, you will create and modify the *main.c* source file which contains the *main()* function that initializes the RTOS kernel, the peripherals, and starts thread execution. In addition, you will configure the system clock and the CMSIS-RTOS RTX.

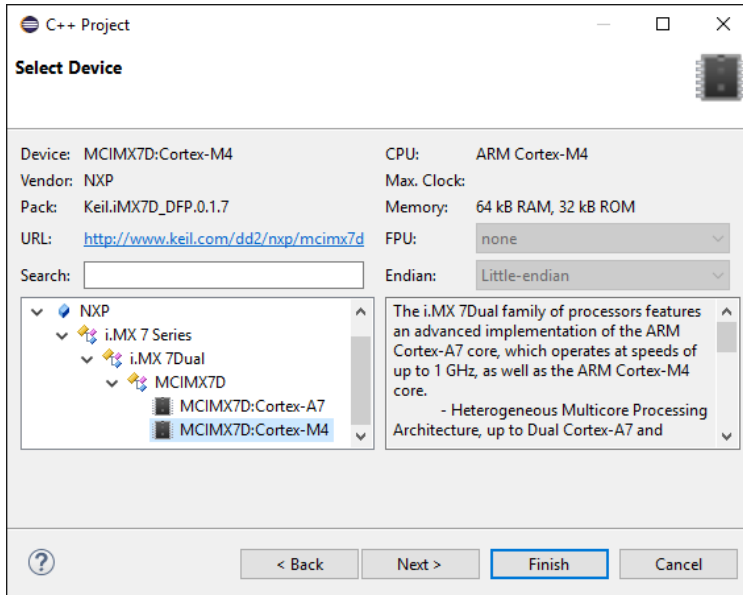
Setup the Project

From the Eclipse menu bar, choose **File** → **New** → **C Project**:

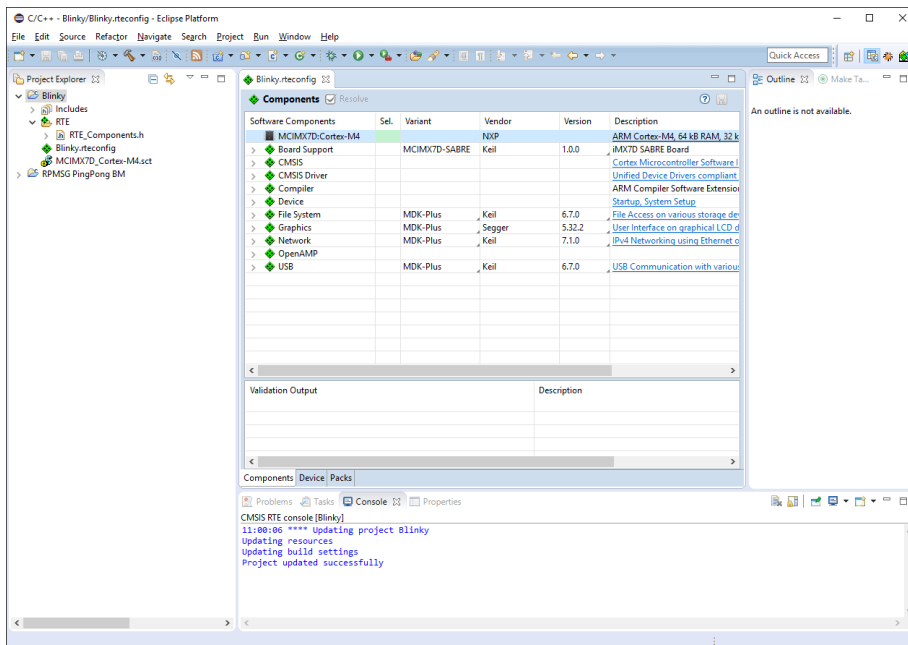


Select **CMSIS RTE C/C++ Project**, enter a project name (for example **Blinky**) and click **Next**. In the following window, you can select to create a default *main.c* file. Do not use this option. We will add a *main.c* template file later from a software pack, so click again **Next**.

Select your target device:



Select the **NXP** → **i.MX 7 Series** → **i.MX Dual** → **MCIMX7D:Cortex-M4** device and click **Finish**. The C/C++ Perspective opens and shows the project:

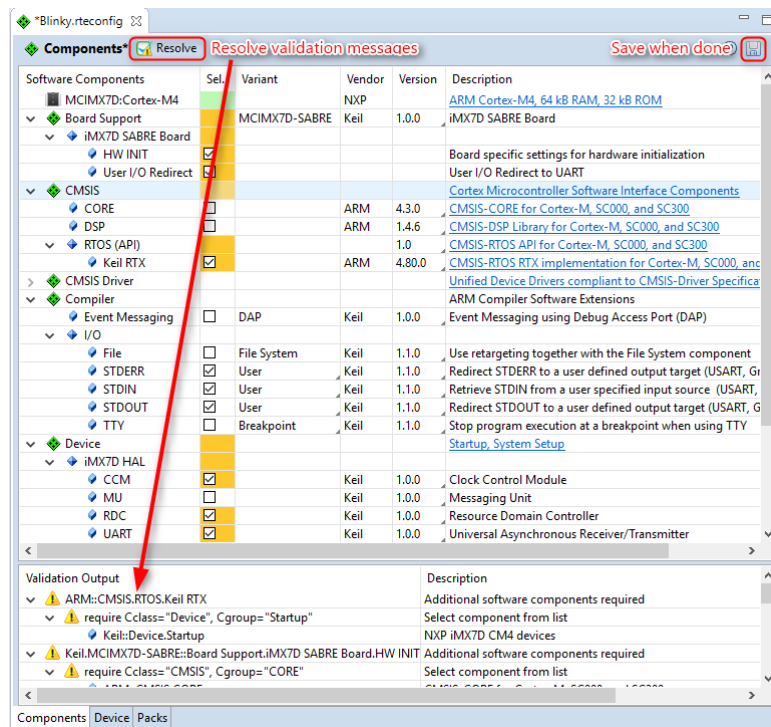


Select Software Components

For the Blinky project based on CMSIS-RTOS RTX, you need to select the following components:

- **CMSIS:RTOS (API):Keil RTX.**
- **Device:i.MX7D HAL:CCM**
- **Device:i.MX7D HAL:RDC**
- **Device:i.MX7D HAL:UART**
- **Compiler:I/O:STDERR** configured as variant **User**
- **Compiler:I/O:STDIN** configured as variant **User**
- **Compiler:I/O:STDOUT** configured as variant **User**
- **Board Support:IMX7D SABRE Board:HW INIT**
- **Board Support:IMX7D SABRE Board:User I/O Redirect**

Use the **Resolve** button to add other required components automatically. Finally, **save** your selection:



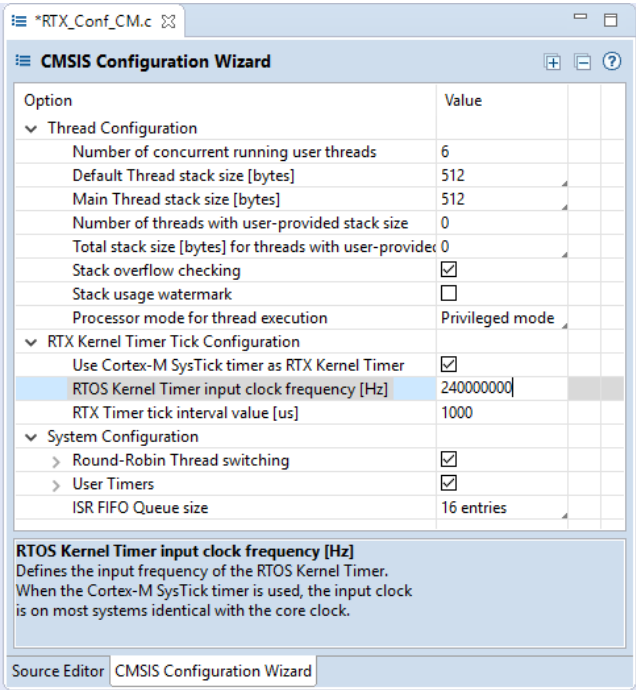
NOTE

Saving the RTE configuration triggers a project update and the selected software components become instantly visible in the Project Explorer.

Configure CMSIS-RTOS RTX Kernel

In the project, expand the group RTE:CMSIS, right-click on the file *RTX_Conf_CM.c*, and select **Open With → CMSIS Configuration Wizard**. Change the following settings:

- Default Thread stack size [bytes] 512
- Main Thread stack size [bytes] 512
- RTOS Kernel Timer input clock frequency [Hz] 240000000



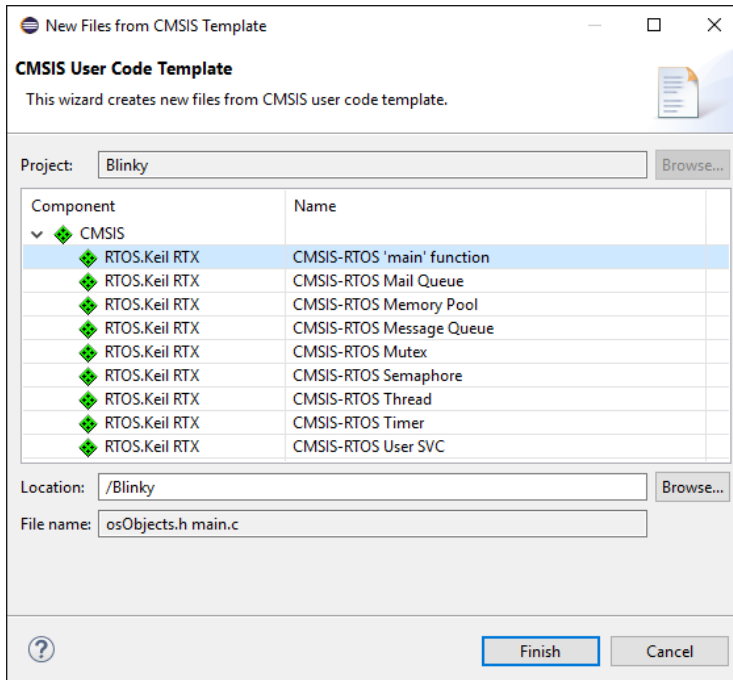
Save the file.

NOTE

If you have opened a file with the CMSIS Configuration Wizard once, your choice is stored and the file will be opened in this view automatically next time.

Create the Source Code Files

Pre-configured user code templates contain routines that resemble the functionality of a software component. Right-click on the project and select **New** → **Files from CMSIS Template**.



Expand the software component **CMSIS** and select the template **CMSIS-RTOS 'main' function**. Click **Finish**. Add application specific code to the file *main.c*:

```
/*-----
 * CMSIS-RTOS 'main' function template
 *-----*/

#define osObjectsPublic           // define objects in main module
#include "osObjects.h"           // RTOS object definitions

#ifdef _RTE_
    #include "RTE_Components.h" // Component selection
#endif
#ifdef RTE_CMSIS_RTOS            // when RTE component CMSIS RTOS is used
    #include "cmsis_os.h"        // CMSIS RTOS header file
#endif
#include "system_iMX7D_M4.h"
#include "retarget_io.h"
#include "board.h"
#include <stdio.h>
```

```
osThreadId tid_threadA; /* Thread id of thread A */

/*-----*/
/*      Thread A
/*-----*/

void threadA (void const *argument) {
    volatile int a = 0;
    for (;;) {
        osDelay(750);
        printf("Blinky   threadA: Hello World!\n");
    }
}

osThreadDef(threadA, osPriorityNormal, 1, 0);

/*
 * main: initialize and start the system
 */
int main (void) {
    /* Board specific RDC settings */
    BOARD_RdcInit();

    /* Board specific clock settings */
    BOARD_ClockInit();

    SystemCoreClockUpdate();
    InitRetargetIOUSART();

    tid_threadA = osThreadCreate(osThread(threadA), NULL);

#ifdef RTE_CMSIS_RTOS // when using CMSIS RTOS
    osKernelInitialize (); // initialize CMSIS-RTOS
#endif

    /* Initialize device HAL here */

#ifdef RTE_CMSIS_RTOS // when using CMSIS RTOS
    osKernelStart (); // start thread execution
#endif

    /* Infinite loop */
    while (1)
    {
        /* Add application code here */
        osDelay(1000);
        printf("Blinky main loop: Hello World!\n");

        /* initialize peripherals here

        // create 'thread' functions that start executing,
        // example: tid_name = osThreadCreate (osThread(name), NULL);

        osKernelStart (); // start thread execution
    }
}
```

Adapt the Scatter File

On the i.MX 7 devices, several types of memory are available. For deterministic, real-time behavior, the Cortex-M4 provides local Tightly Coupled Memory (TCM), which provides low-latency access. Multiple on-chip RAM areas (OCRAM) are available, which are larger, but not as fast.

The following table shows the memories and their load addresses for the different processors:

Region	Size	Cortex-A7	Cortex-M4 (Code Bus)
OCRAM	128 KB	0x00900000-0x0091FFFF	0x00900000-0x0091FFFF
TCMU	32 KB	0x00800000-0x00807FFF	
TCML	32 KB	0x007F8000-0x007FFFFF	0x1FFF8000-0x1FFFFFFF
OCRAM_S	32 KB	0x00180000-0x00187FFF	0x00000000-0x00007FFF/ 0x00180000-0x00187FFF

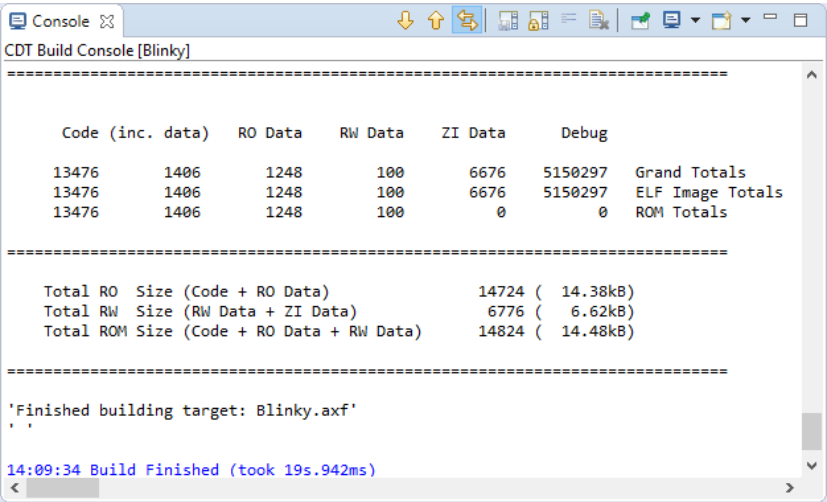
By default, the scatter file template uses the start address 0x0 for the load region command. To put the Cortex-M4 code into the TCM, change the address of the load region to 0x1FFF8000:

```
; *****
; ** Scatter-Loading Description File generated by RTE CMSIS Plug-in **
; *****

LR_IROM1 0x1FFF8000 0x00008000 {      ; load region size_region
  ER_IROM1 0x1FFF8000 0x00008000 {    ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00008000 {
    .ANY (+RW +ZI)
  }
}
```


Build the Cortex-M Image

Right-click on the project name and select **Build Project** to build the application. This step compiles and links all related source files. The **Console** shows information about the build process. An error-free build displays program size information:



Debug Cortex-M Application on page 37 guides you through the required steps to connect your evaluation board to the workstation and to debug the application on the target hardware.

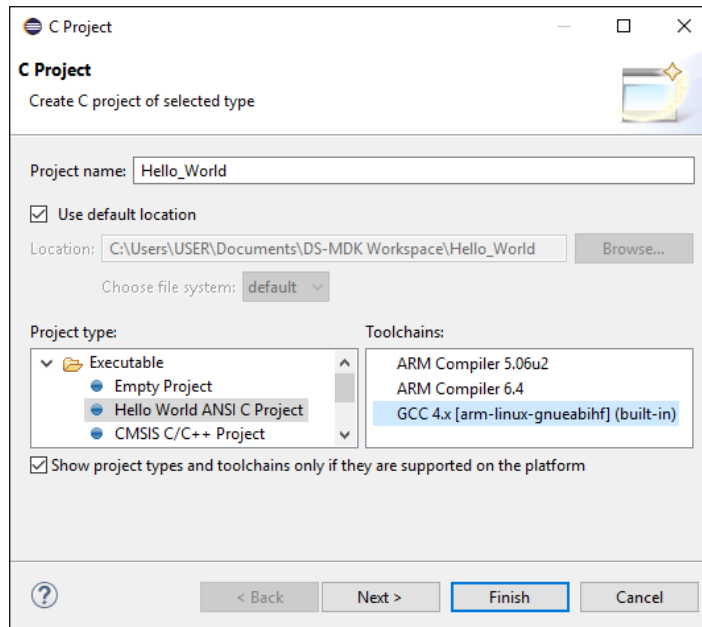
Create Linux Applications

This chapter guides you through the steps required to create and modify projects for an ARM Cortex-A class device running Linux:

- **Setup the Project:** create a project.
- **Build the Application Image:** compile and link the application.

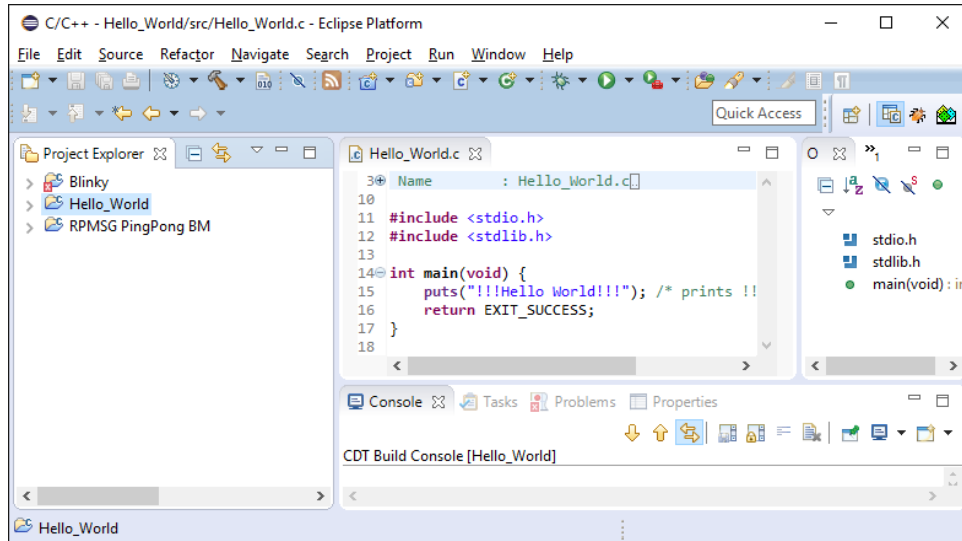
Setup the Project

From the Eclipse menu bar, choose **File** → **New** → **C Project**. Select the **Hello World ANCI C Project**:



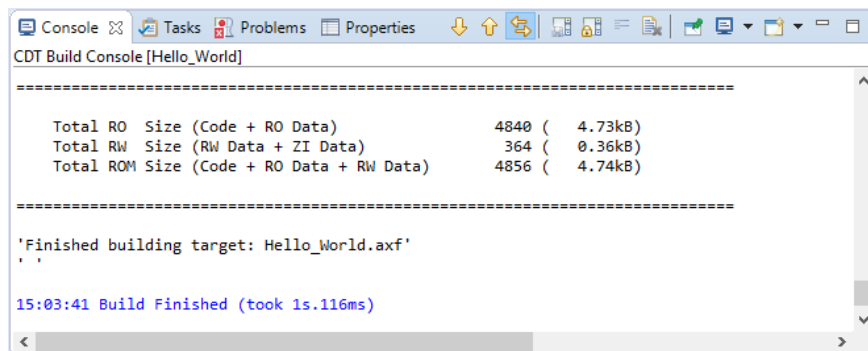
Enter a project name (for example `Hello_World`) and make sure that the **GCC [...] (built-in)** toolchain is selected before clicking **Finish**.

The C/C++ Perspective opens and shows the current project:



Build the Application Image

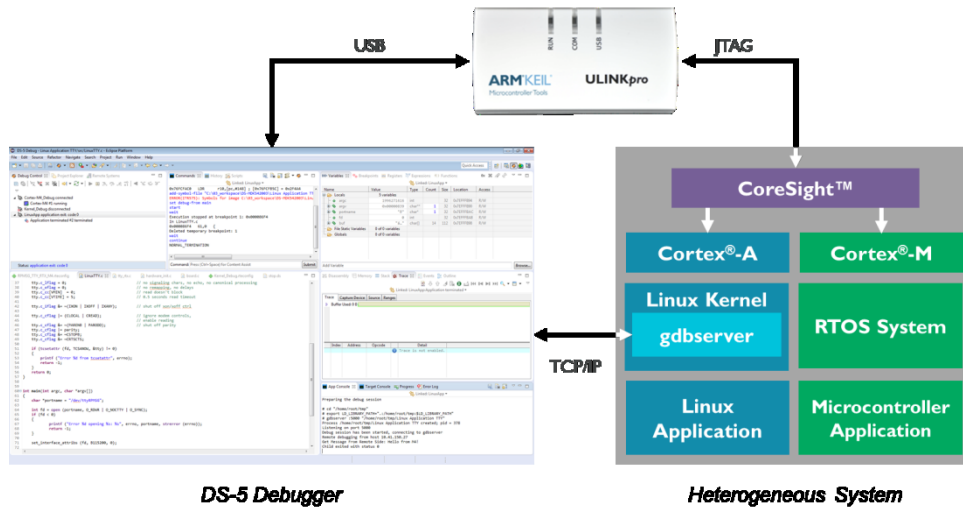
Right-click on the project name and select **Build Project**. This step compiles and links all related source files. The **Console** shows information about the build process:



The chapter **Debug Linux Application** on page 42 guides you through the required steps to connect your evaluation board to the workstation and to download the application to the target hardware.

Debug Applications

The DS-5 Debugger can verify all software applications that execute on a heterogeneous computer system. It enables complete system visibility using multiple simultaneous debug connections:

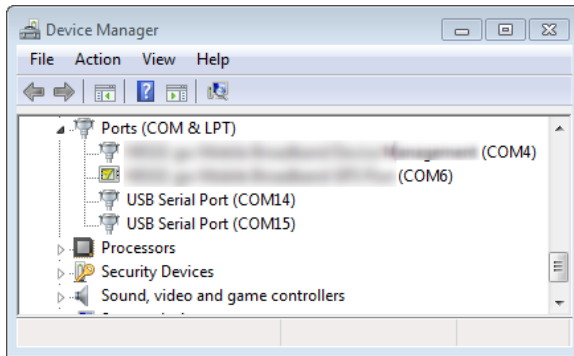


- The **Cortex-M application** is debugged using a ULINKpro debug unit (refer to www.keil.com/ulink for more information). Users can analyze the microcontroller application with RTOS aware-debugging and peripheral views.
- The **Cortex-A Linux kernel** is also debugged using a ULINKpro debug unit. The debugger lists kernel threads and processes.
- The **Cortex-A Linux application** is debugged via [gdbserver](#) across a TCP/IP network link. The debugger supports multi-threaded application debugging and shows pending breakpoints on loadable modules and shared libraries.

Prepare Terminal Views

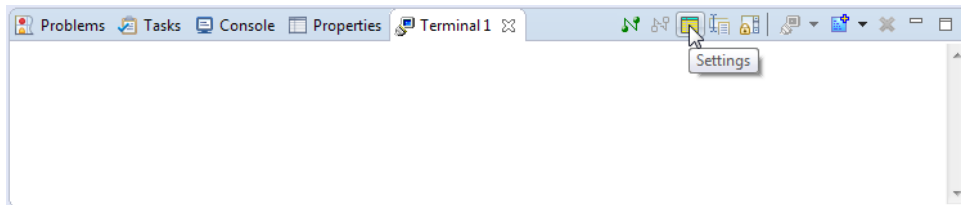
Many applications use a serial connection to display messages. A **Terminal** window shows these messages from serial COM ports.

The i.MX 7 SABRE development board contains a dual USB serial port device with two independent COM ports. Connect the board to your computer. Windows installs the drivers automatically and adds two new USB Serial Ports to your system. Check the exact numbers in the Windows **Device Manager** (to open it, type “device manager” in the Windows search bar):



The smaller number is the COM port of the Cortex-A processor, while the larger number is the COM port of the Cortex-M processor. To open a Terminal view, go to **Window → Show View → Other...** Select **Terminal → Terminal** and click **OK**.

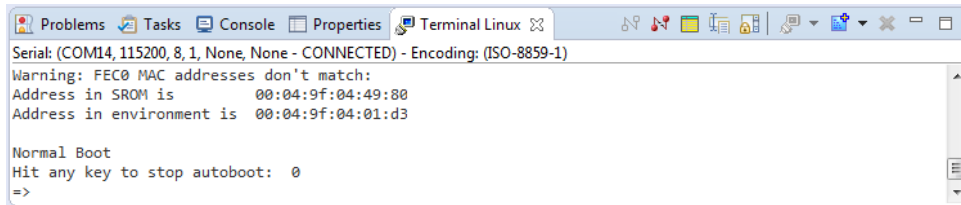
Open the settings dialog from the toolbar of the **Terminal 1** window:



Set the following:

- View Title: Terminal Linux
- Connection Type: Serial
- Port: Use the first of the new COM ports
- Baud Rate: 115200

Click OK. Press the RST button on the development board to observe the boot process in the Terminal window. Send any keyboard key to the terminal window to interrupt the boot process:




The screenshot shows a terminal window titled 'Terminal Linux'. The output text is as follows:

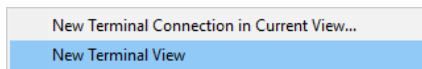
```
Serial: (COM14, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
Warning: FEC0 MAC addresses don't match:
Address in SR0M is      00:04:9f:04:49:80
Address in environment is 00:04:9f:04:01:d3

Normal Boot
Hit any key to stop autoboot: 0
=>
```

NOTE

You must halt the boot loader at this point to be able to launch the Cortex-M debug session.

Add another Terminal view to display the output of the Cortex-M processor. Simply use the drop-down selector next to the **New Terminal Connection in Current View...** icon  and select **New terminal View**:



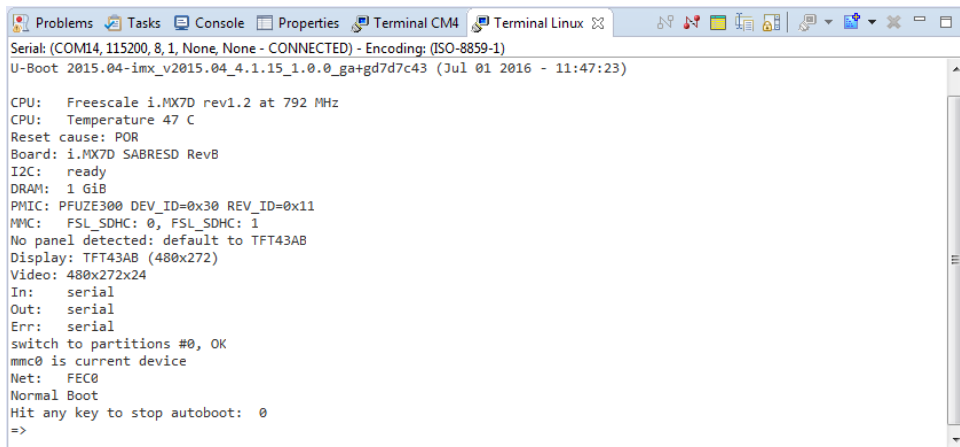
Select the larger COM port number and leave the other settings as they are. Name the Terminal view **Terminal M4**.

Debug Cortex-M Application

This section explains how to debug the microcontroller application running on the Cortex-M microcontroller. If you are debugging the *Blinky* application from the previous chapter, execute the following steps using that project. Here, we will continue with the *RPMSG_TTY_RTX_M4* project from the **Verify Installation with Example Project** chapter.

Stop in U-Boot

Stop in U-Boot to be able to connect to the target. Restart/reset the device and observe the bootloader output on the **Terminal Linux**. Press any key before the autoboot countdown expires:

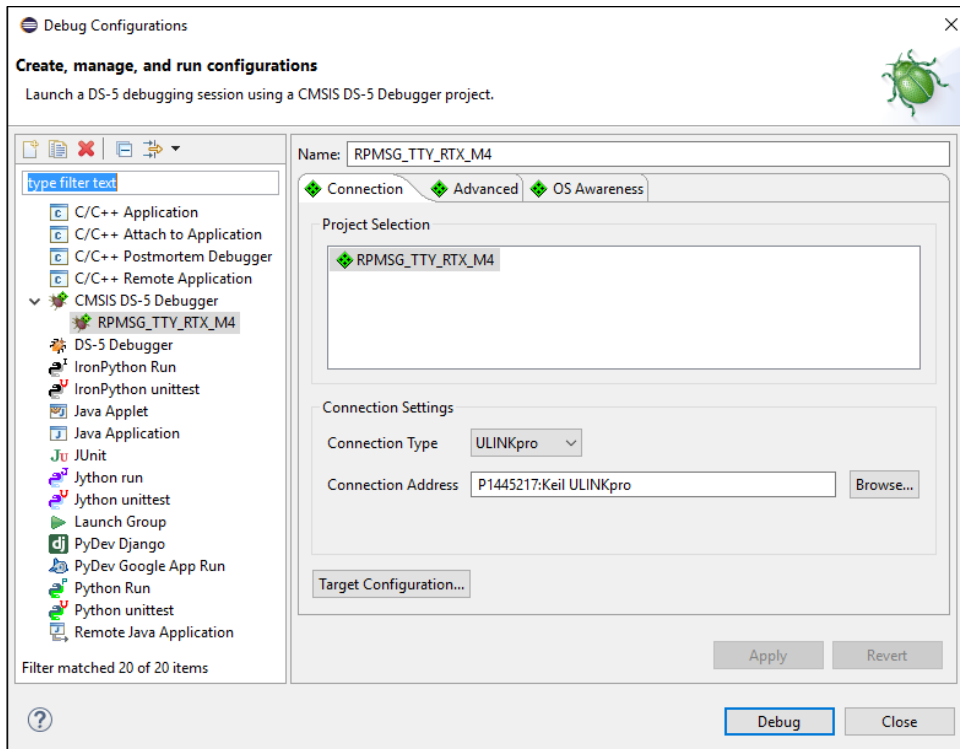


```
Problems Tasks Console Properties Terminal CM4 Terminal Linux
Serial: (COM14, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
U-Boot 2015.04-imx_v2015.04_4.1.15_1.0.0_ga+gd7d7c43 (Jul 01 2016 - 11:47:23)

CPU:   Freescale i.MX7D rev1.2 at 792 MHz
CPU:   Temperature 47 C
Reset cause: POR
Board: i.MX7D SABRESD RevB
I2C:   ready
DRAM:  1 GiB
PMIC:  PFUZE300 DEV_ID=0x30 REV_ID=0x11
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
No panel detected: default to TFT43AB
Display: TFT43AB (480x272)
Video: 480x272x24
In:    serial
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc0 is current device
Net:   FEC0
Normal Boot
Hit any key to stop autoboot: 0
=>
```

Configure CMSIS DS-5 Debugger

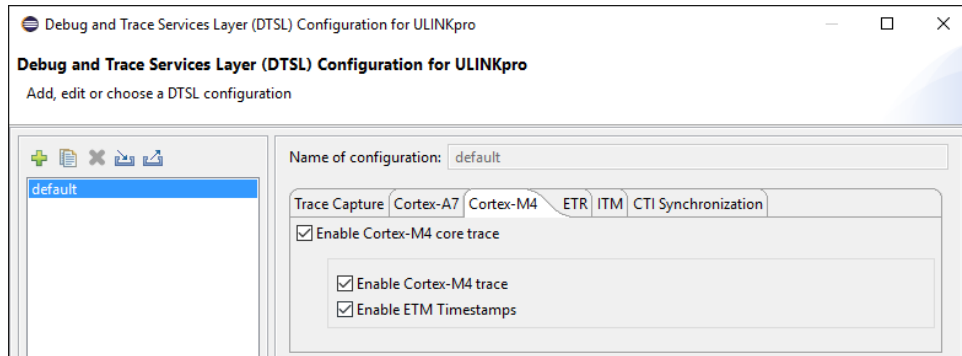
Right-click the **RPMSG_TTY_RTX_M4** project and select **Debug As → CMSIS DS-5 Debugger** to launch the debug configurations dialog:



Connection

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

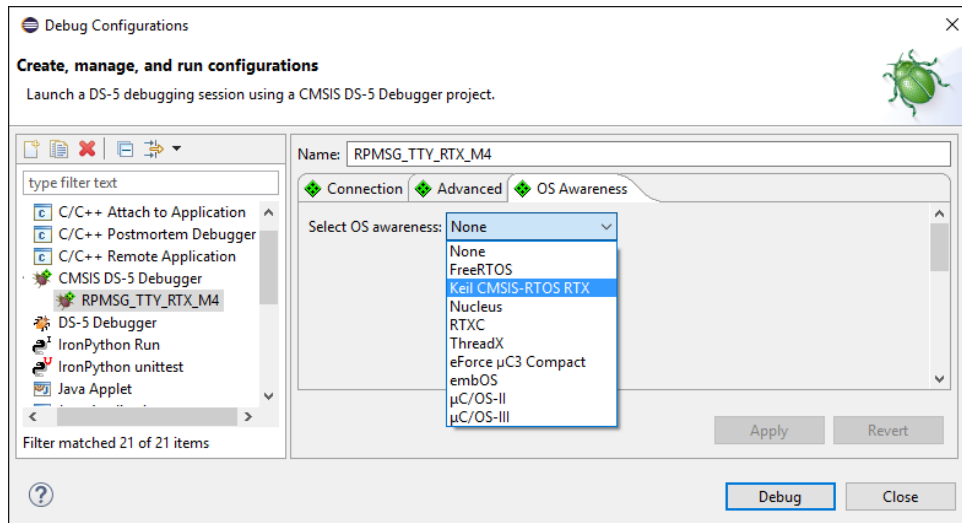
Click on **Target Configuration...** to setup the Debug and Trace Services Layer (DTSL).



- On the **Cortex-A7** tab, disable all trace options to avoid buffer overflows.
- On the **Cortex-M4** tab, **Enable Cortex-M4 core trace**.

OS Awareness

In the **OS Awareness** tab select the real-time operating system used in your application from the drop-down menu.



Click **Debug**.

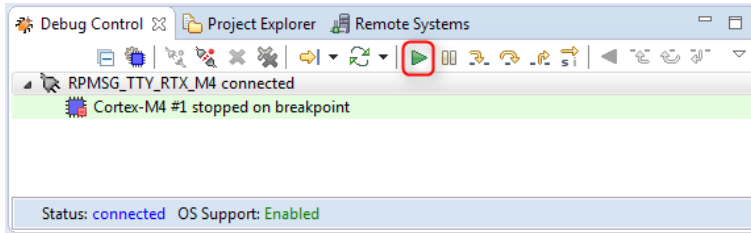
NOTE

The error message “**Failed to launch debug server**” most likely indicates that an incorrect **ULINKpro** connection address is selected.

Run Cortex-M Application

DS-MDK switches to the DS-5 Debug perspective.

The application loads and runs until main. To start the Cortex-M4 application click **Run** in the **Debug Control** view.



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

NOTE

You can add another Terminal view to the debug perspective by using **Window → Show View → Terminal**.

Debug Linux Application


This section explains how to debug the Linux application running on the Cortex-A7. If you are debugging the *Hello_World* application from the previous chapter, execute the following steps using that project. Here, we will continue with the *Linux Application TTY* project from the **Verify Installation with Example Project** chapter.

The DS-5 Debugger uses gdbserver for debugging Linux on the target hardware. Before connecting, you must:

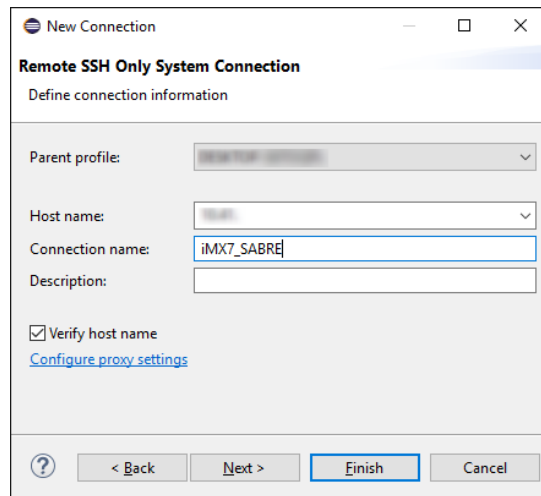
- Set up the target with Linux installed and booted. Refer to **Install the Linux Image** on page 12.
- Obtain the target IP address or name for the connection between the debugger and the debug hardware adapter. If the target is in your local subnet, click Browse and select your target.

Next, you should set up a Remote Systems Explorer (RSE) connection to the target to download the application onto the target's file system.

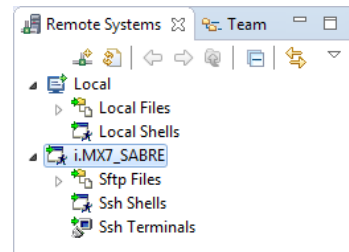
Setup RSE Connection

Go to **Window** → **Open Perspective** → **Other...**, then select **Remote System Explorer**. Use the  button to create a new connection. Select **SSH Only** and click **Next**.

RSE communicates with the target using TCP/IP. Thus, you need to enter the target's IP address into the **Host Name** field. Enter a meaningful name in the **Connection name** box:



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

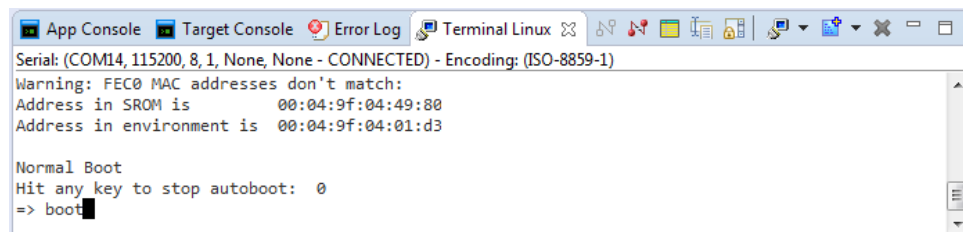


Boot Linux

NOTE


If you are debugging a microcontroller application simultaneously, you need to run the Cortex-M application, otherwise the Linux Terminal will not be accessible and you will not be able to boot Linux.

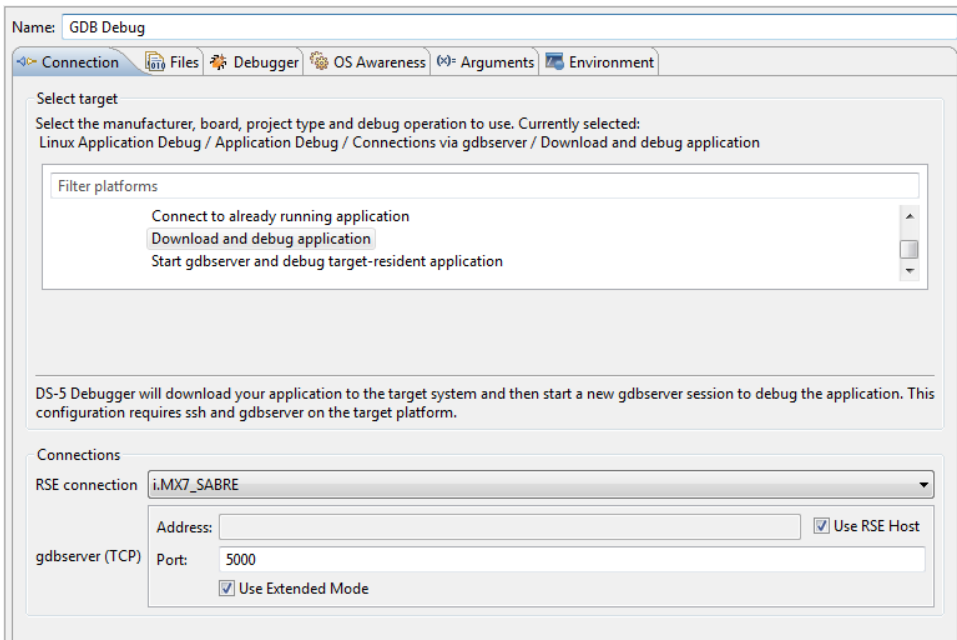
In the **Terminal Linux** enter “boot” to start the Linux system:



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

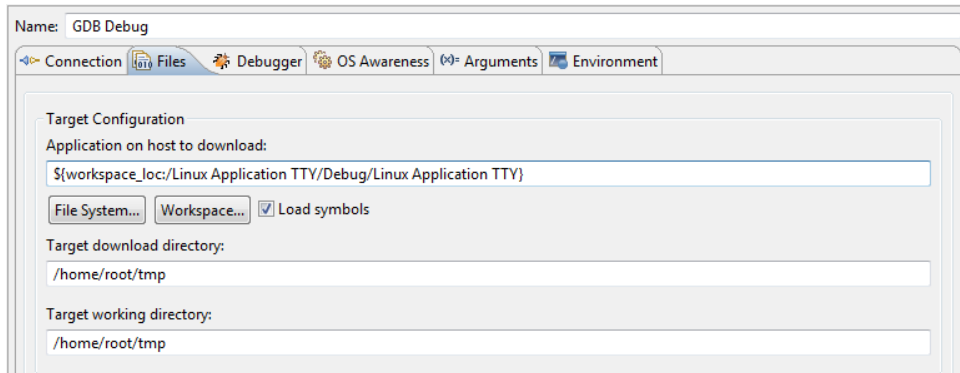
Configure DS-5 Debugger

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



On the **Files** tab, in **Target Configuration**, select the workspace build target for **Application on host to download**. Select an **existing** directory on the target file system, e.g. `/home/root/tmp` as the **Target download directory**.

Select an **existing** directory on the target file system, e.g. `/home/root/tmp` as the **Target working directory** (use the same directory as for **Target download directory**).



On the **Debugger** tab, under **Run Control** select **Debug from symbol** “main”. Click **Debug**.


Run the Linux Application

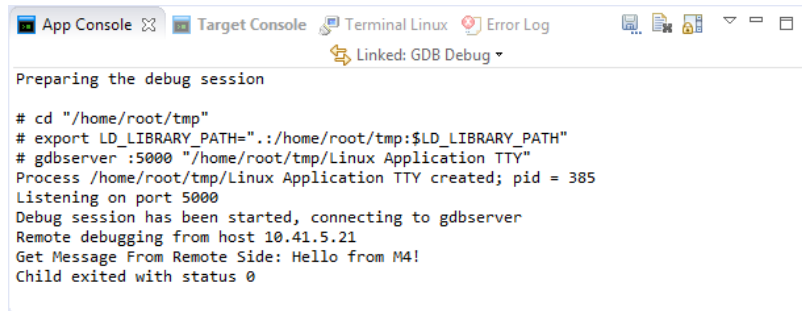
In the **Terminal Linux**, load the kernel module that communicates with the Cortex-M4 application with this command:

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

The kernel module should be loaded as shown below:

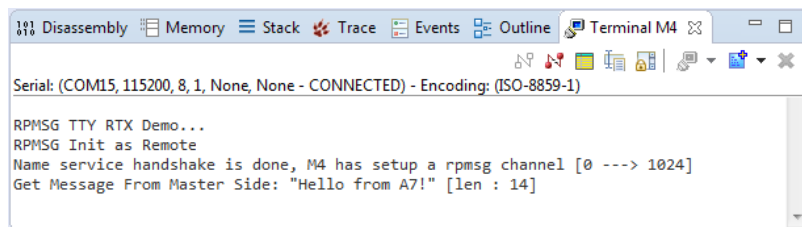
```
insmod /lib/modules/4.1.15-
1.1.0+ga4d2a08/kernel/drivers/rpmsg/imx_rpmsg_tty.ko
imx_rpmsg_tty rpmsg0: new channel: 0x400 -> 0x0!
Install rpmsg tty driver!
```

Use the **Continue**  button to run the Linux application. The **App Console** shows the application's messages:



```
App Console Target Console Terminal Linux Error Log
Linked: GDB Debug v
Preparing the debug session
# cd "/home/root/tmp"
# export LD_LIBRARY_PATH=".:/home/root/tmp:$LD_LIBRARY_PATH"
# gdbserver :5000 "/home/root/tmp/Linux Application TTY"
Process /home/root/tmp/Linux Application TTY created; pid = 385
Listening on port 5000
Debug session has been started, connecting to gdbserver
Remote debugging from host 10.41.5.21
Get Message From Remote Side: Hello from M4!
Child exited with status 0
```

Similarly, the **Terminal M4** shows the output of the microcontroller application:



```
Disassembly Memory Stack Trace Events Outline Terminal M4
Serial: (COM15, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
RPMSG TTY RTX Demo...
RPMSG Init as Remote
Name service handshake is done, M4 has setup a rpmsg channel [0 ---> 1024]
Get Message From Master Side: "Hello from A7!" [len : 14]
```

NOTE

You can add another Terminal view to the debug perspective by using **Window → Show View → Terminal**.

Store Cortex-M Image

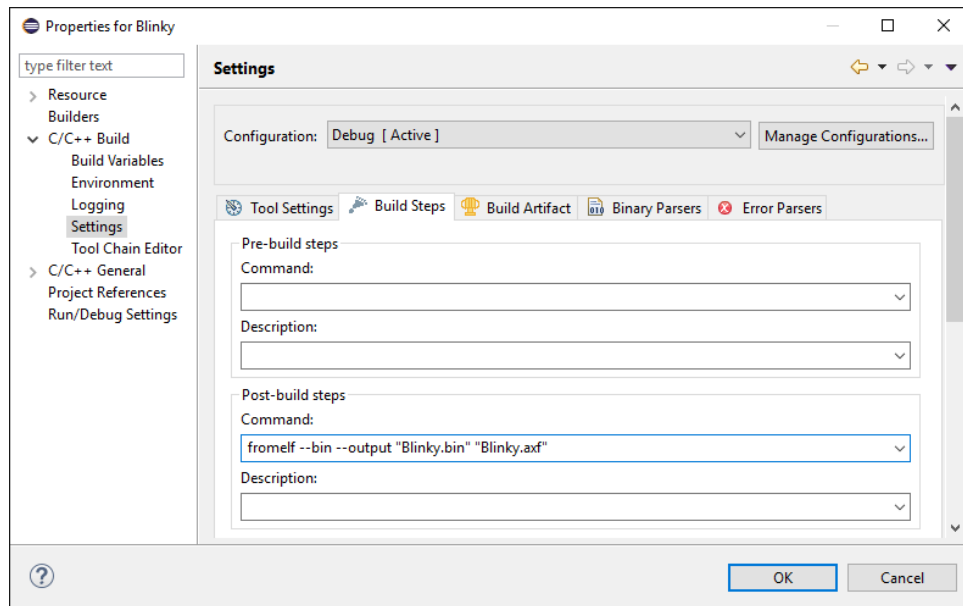
To store the Cortex-M image for execution at start up use the following steps:

1. Create a binary image (BIN) with the `fromelf` utility application.
2. Store this BIN image on SD card in the boot partition
3. Setup the U-Boot environment to start-up the BIN image file.

Create a Cortex-M Binary Image (BIN)

Right-click the project and select **Properties** → **C/C++ Build** → **Settings**. In the the **Build Steps** enter under **Post-build steps** the **Command**:

```
fromelf --bin --output "Blinky.bin" "Blinky.axf"
```



NOTE

*This example shows the steps for the Blinky application from section **Blinky with CMSIS-RTOS RTX** on page 22.*

Click OK and rebuild the project to get the BIN file generated.

Store Cortex-M BIN File on SD Card

The SD Card has two partitions:

- The **Linux** file system partition.
- The **FAT32** boot partition.

List the partitions with the `fdisk` command:

```
~# fdisk -l
...
Device           Boot Start      End  Sectors  Size Id Type
/dev/mmcblk0p1          8192    24575    16384     8M  c W95 FAT32 (LBA)
/dev/mmcblk0p2    24576 1236991 1212416   592M 83  Linux
```

Store the Cortex-M binary image in the **FAT32** boot partition to be able to execute it at system startup:

1. Create a sub-directory on the Linux file system, for example:

```
~# mkdir /media/sd0
```

2. Mount the Linux file system partition for access with RSE.

```
~# mount -t vfat /dev/mmcblk0p1 /media/sd0
```

3. Use RSE to copy the BIN file from your workspace to the `/media/sd0` directory.
4. Unmount the partition to ensure that the file is written correctly:

```
~# umount /media/sd0
```

5. Reboot the system and halt in U-Boot.

Run Cortex-M BIN File from U-Boot

At this point, the Cortex-M BIN file is stored in the boot partition. Use the `setenv` command to change the boot image to the new BIN file:

```
=> setenv m4image Blinky.bin; save
```

The `printenv` command shows the boot setup:

```
=> printenv
...
loadm4image=fatload mmc ${mmcdev}:${mmcpart} 0x7F8000 ${m4image}
m4boot=run loadm4image; bootaux 0x7F8000
m4image=Blinky.bin
```

Run `m4boot` to start the Blinky application:

```
=> run m4boot
```

NOTE

For more information refer to the U-Boot Command Line Interface in the U-Boot user's manual (www.denx.de/wiki/DULG/UBoot).

Index

A

Applications

Add Source Code	31
Blinky with CMSIS-RTOS RTX	25
Build	33
Build Cortex-M Image	33
Create	25
Create BIN File	47
Create Source Files	30
Customize RTOS	29
Debug	36
Run from U-Boot	49
Select Software Components	28
Setup the Project	26
Store BIN File	48

C

Console	16
---------------	----

D

Debug

OS Awareness	41
--------------------	----

Device Database	11
-----------------------	----

Documentation	17
---------------------	----

DS-MDK

Install	9
Installation Requirements	9
Introduction	7
License Types	8
Licensing	8

E

Eclipse

IDE	18
Perspectives	18

Example Project

Install	14
---------------	----

F

Flash Programmig

Scatter File	32
--------------------	----

I

i.MX7 SABRE

Hardware Connection	13
---------------------------	----

L

Linux

Create Image	12
Install Image	12

Linux Applications

Build Application Image	35
Development	34
Project Set Up	34

P

Perspective

C/C++	19
CMSIS Pack Manager	22
DS-5 Debug	24
Remote System Explorer	23

R

Remote System Explorer	43
------------------------------	----

S

Software Packs

Manage	11
--------------	----

T

Terminal View	37
---------------------	----