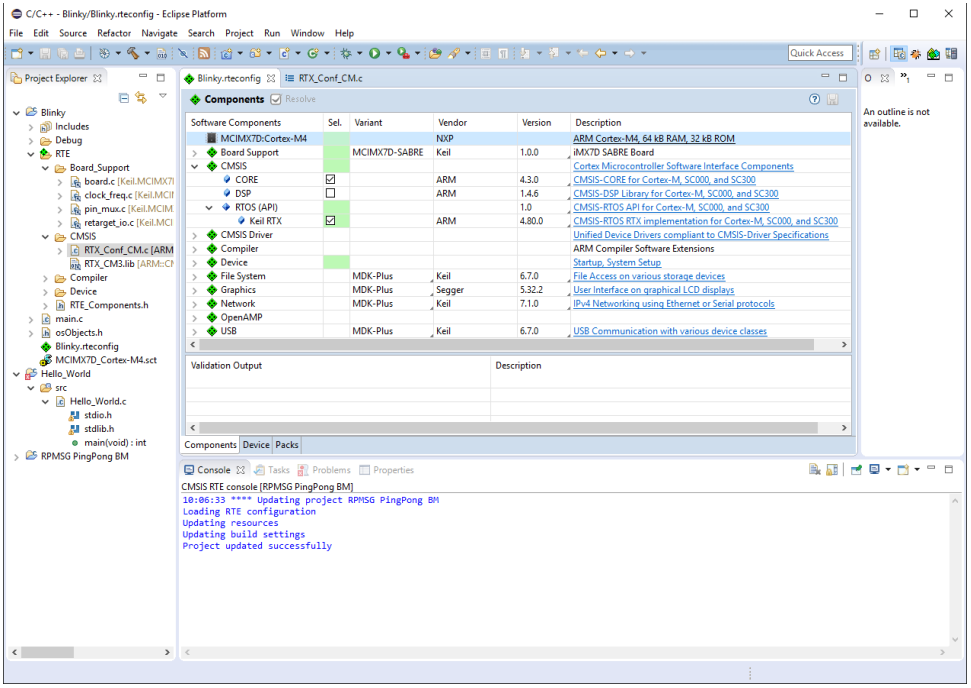
ARM KEIL

Microcontroller Tools

Getting Started with DS-MDK

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



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

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-M4 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-M4 Image gives further details on how to store the application image on the target and how to run it at start up time.

Contents

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.

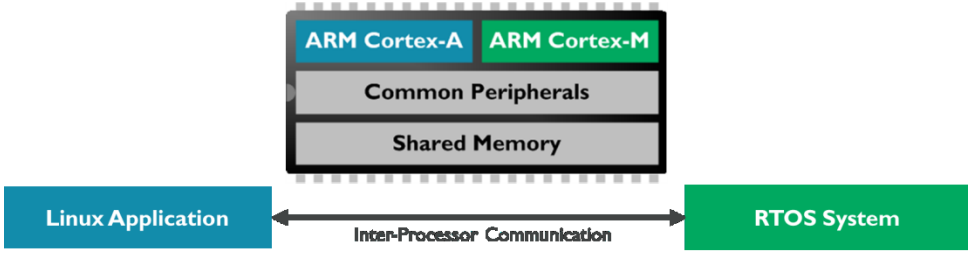
*Refer to the **Getting Started with MDK** user's guide for information how to create projects for ARM Cortex-M microcontrollers using the µVision® IDE/Debugger.*

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:

- It allows managing Cortex-A Linux and Cortex-M RTOS projects in the same development environment.

- It fully supports the Cortex Microcontroller Software Interface Standard (**CMSIS**) development flow for efficient Cortex-M programming. Software Packs may be added any time to DS-MDK making new device support and middleware updates independent from the toolchain. They contain device support, CMSIS libraries, middleware, board support, code templates, and example projects. The IDE manages the provided software components that are available for the application as building blocks.
- The DS-MDK Debugger offers full visibility for multicore software development.

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 PC running a Microsoft Windows (32-bit or 64-bit) operating system*
- *Dual-Core Processor with > 2 GHz*
- *4 GB RAM and 8 GB hard-disk space*
- *1280 x 800 or higher screen resolution*

Install MDK

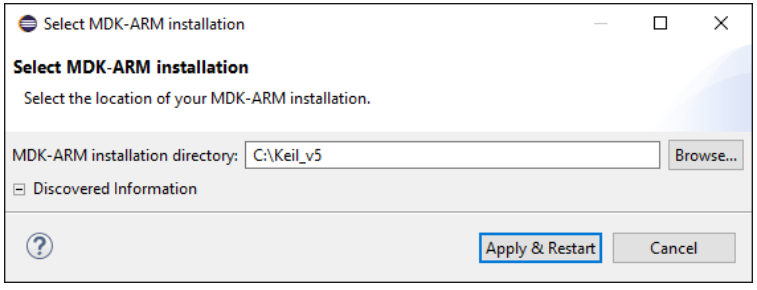
Download **MDK** from **www.keil.com/download** - Product Downloads and run the installer. It also adds the Software Packs for ARM **CMSIS** and MDK **Middleware**.

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

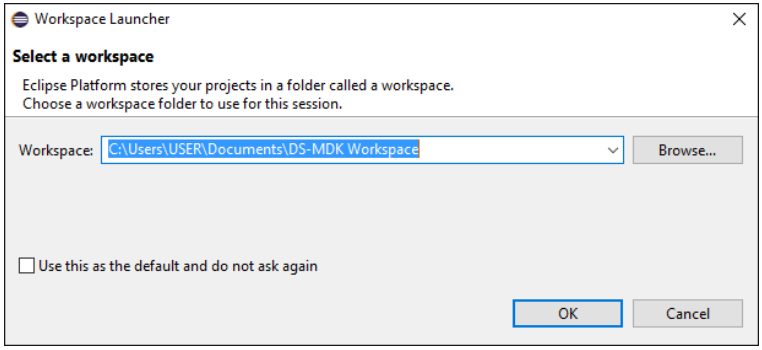
Download **DS-MDK** from **www.keil.com/mdk5/ds-mdk/install** and run the installer. Having finished the installation, start DS-MDK by clicking on **Eclipse for DS-MDK** in the Start menu (Windows 10: **All apps → ARM DS-MDK → Eclipse for DS-MDK**).

When starting the product the first time, you will be presented with a window showing MDK installation detected in your system:

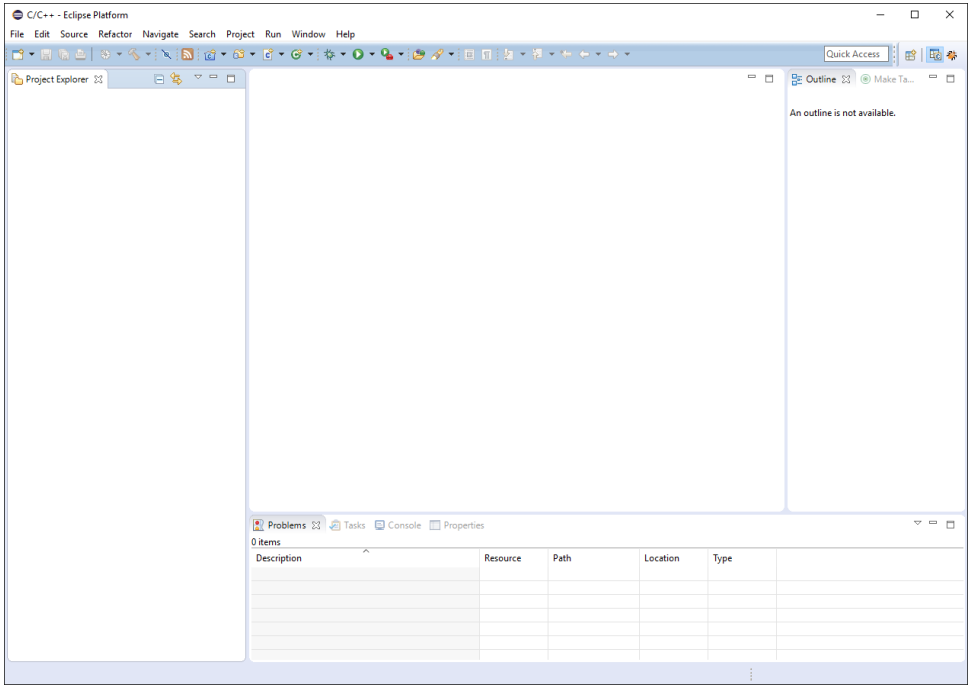


If required, change the installation destination.

Now, you need to 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.*

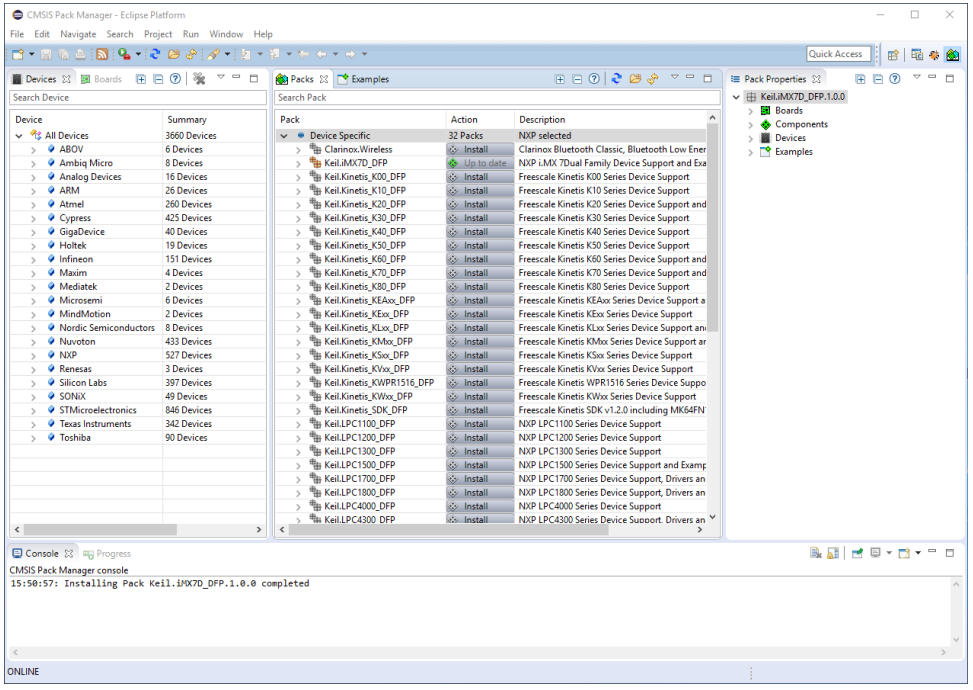
Manage Software Packs

Use the **CMSIS Pack Manager** perspective for managing Software Packs on the local computer.

Open this perspective using **Window → Open Perspective → CMSIS Pack Manager**. You should now install the Software Pack 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 can manually install Software Packs using the **Import existing packs** icon or by double-click on *.PACK files.

Install the Linux Image

Currently, DS-MDK supports the following development board:

- NXP i.MX 7 SABRE development board: MCIMX7SABRE

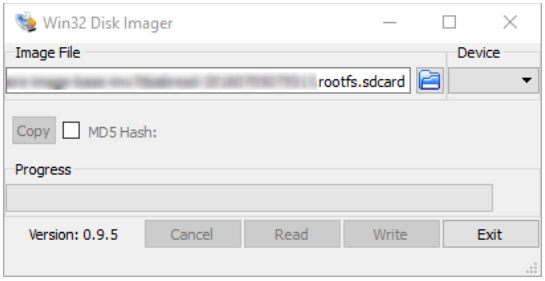
For this development board, a pre-configured Linux image with DS-MDK specific debug settings is available. Please download the zipped image file here: **www.keil.com/mdk5/ds-mdk/imx7reference**

All steps to create a Linux image for the MX7DSABRESD board are explained in the documentation. Please refer to the **Freescale Yocto Project User's Guide** and the **i.MX Linux User's Guide**. The above website also links to these documents and explains the changes that are required for DS-MDK debugging using ULINK*pro*.

Copy the Linux Image to an SD-Card

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

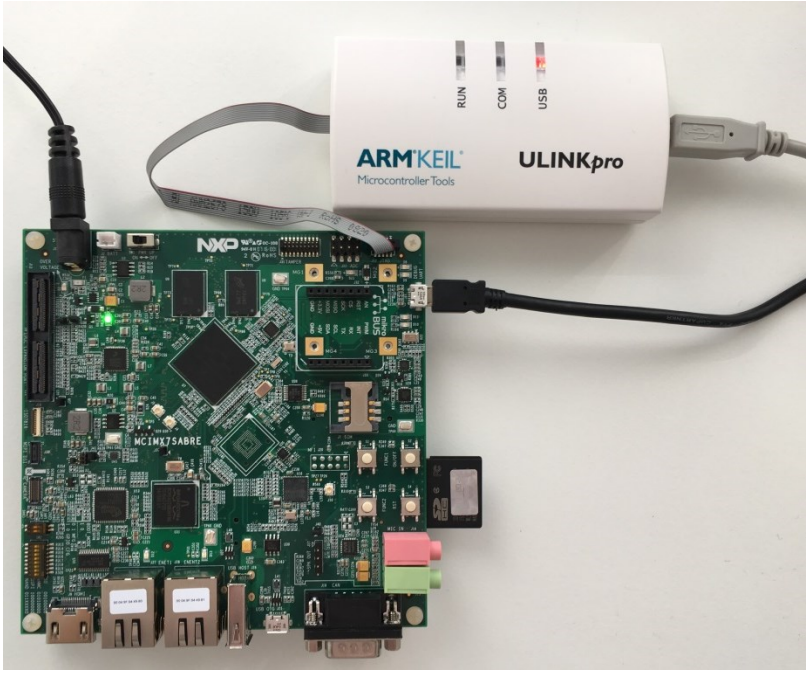
Install and run the tool. To write the image to the memory card, specify the location of the image file, select the **Device** letter of the SD card and press the **Write** button:



Hardware Connection

i.MX 7 SABRE Board

- Insert the SD-Card with the Linux image into the slot labelled **SD1 BOOT**.
- Connect the ULINK*pro* debug adapter using the 10-pin ribbon cable to **J12 JTAG**.
- Connect your computer using a Micro-USB cable to the USB connector labelled **DEBUG UART**. Your Windows PC will automatically detect a dual USB serial port component and will install the required drivers.
- Connect the 5V power supply to **J1**.

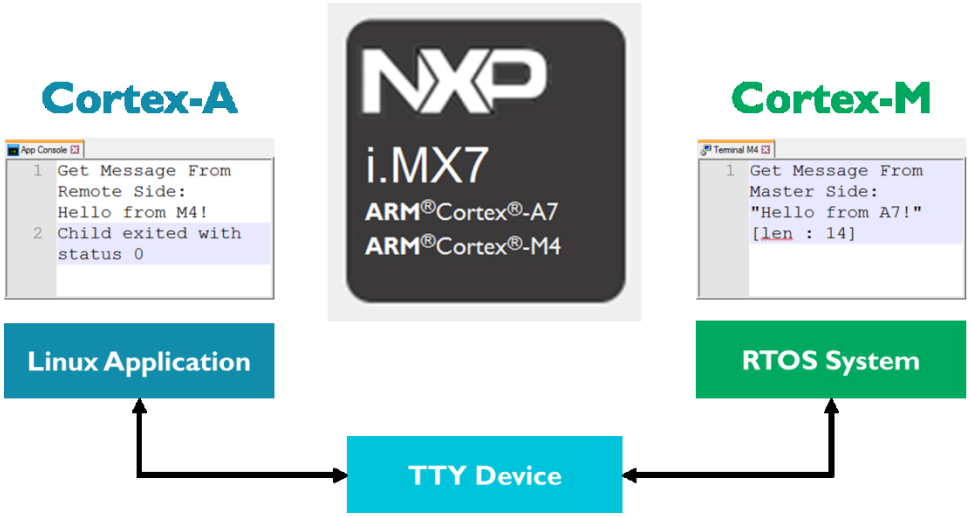


Verify Installation using Example Projects

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

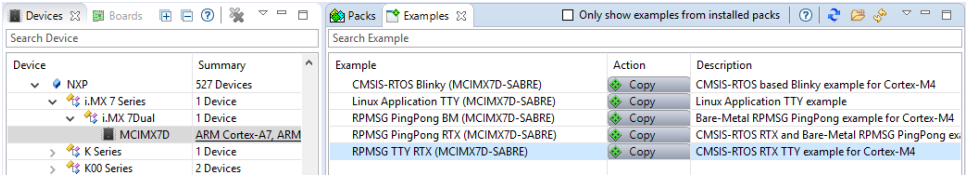
The i.MX 7 Device Family Pack contains two example projects that show how the two processors communicate with each other using the remote processor messaging protocol (RPMSG) via a TTY serial device. The TTY device is installed on the Linux system using a Linux kernel module (`imx_rpmsg_tty.ko`).



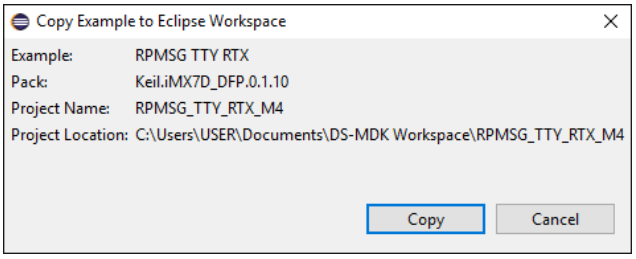
The *Linux Application TTY* runs on the Cortex-A7 processor and writes a message to the TTY device. This message is shown in the terminal of the *RPMSG TTY RTX* application running on the Cortex-M4 processor. This application responds on the TTY device, which is read by the Linux application and shown in its console.

Copy the RPMSG TTY RTX Example Project

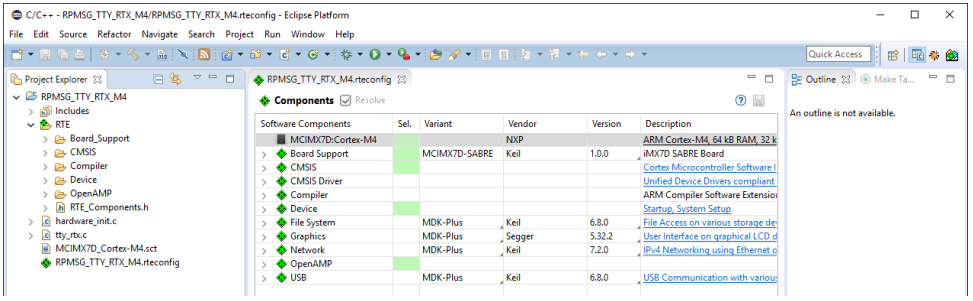
In the **CMSIS Pack Manager** perspective, select the **Examples** tab. Use filters in the toolbar to narrow the list of examples.



Click **Copy** next to the **RPMSG TTY RTX** example. A new window opens asking you to verify the selected example project:

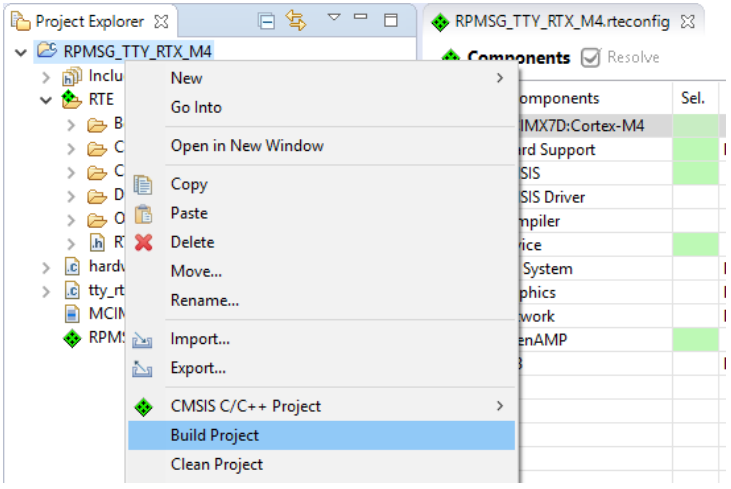


CMSIS Pack Manager copies the example into your workspace and switches automatically to the C/C++ perspective:

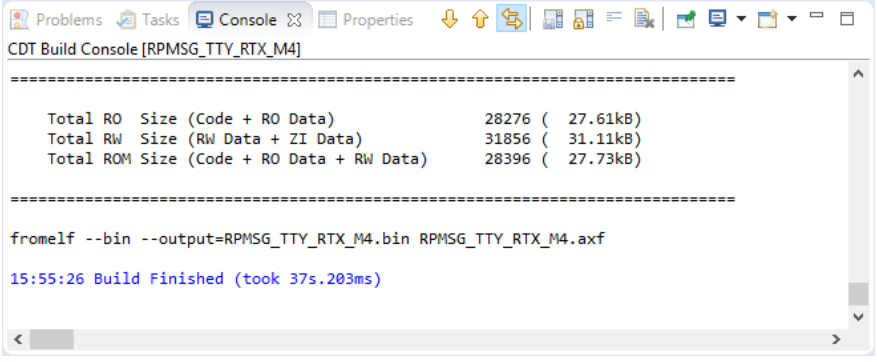


Build the Application

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



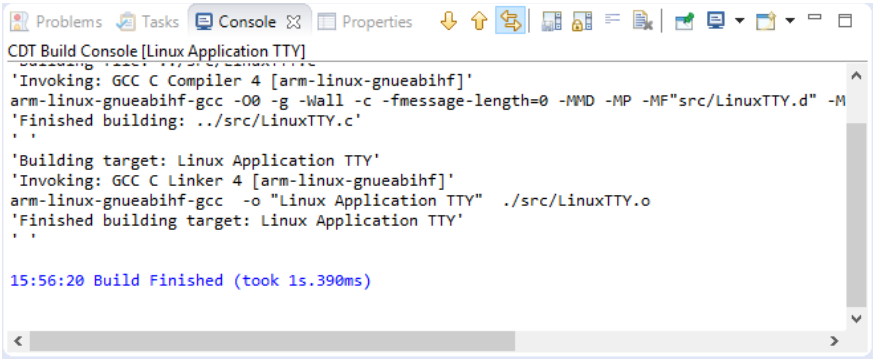
The **Console** window shows information about the build process:



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:



The chapter **Debug Applications** on page 36 explains how to debug both applications using 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 uses Eclipse for DS-5, an Integrated Development Environment (IDE) that combines the Eclipse IDE from the Eclipse Foundation with the compilation and debug technology of the ARM tools.

You can use Eclipse for DS-5 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.

Workbench

The workbench is the main development environment where you can manage individual projects, associated sub-folders, and source files.

Each workbench window links to one workspace. If you want to use different workspaces at the same time, you can launch several workbench windows and link each one to a different workspace.

Perspectives

A workbench contains multiple perspectives. Each perspective contains an initial set and layout of views. It aims at accomplishing a specific type of task, such as project creation and build, debugging, and Pack management. 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:

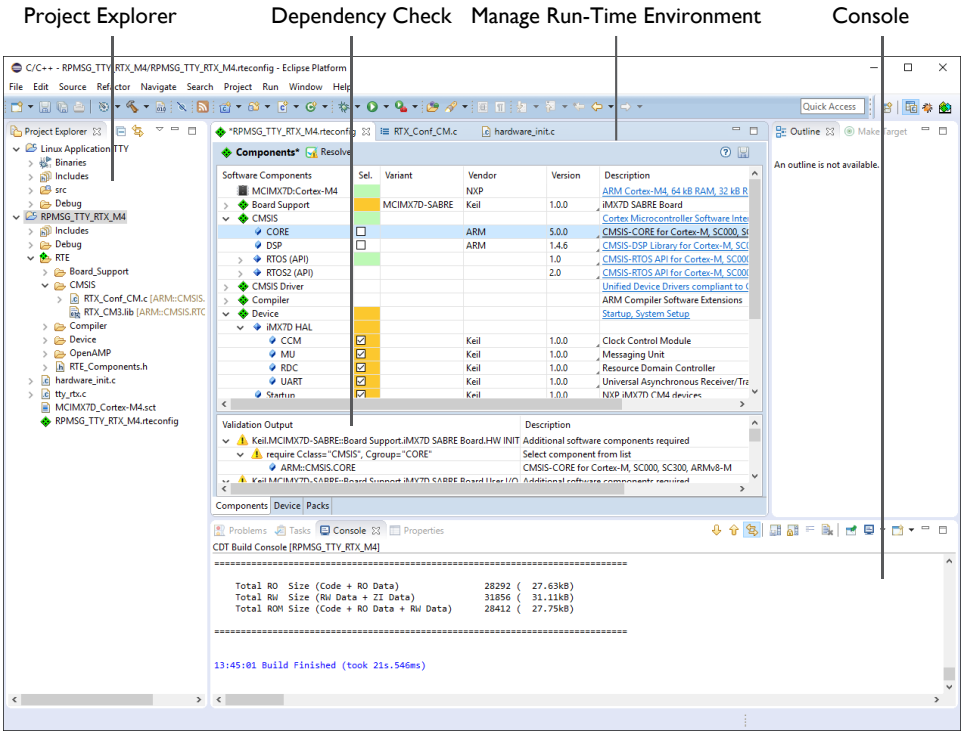
- **C/C++ Perspective**
- **CMSIS Pack Manager Perspective**
- **Remote System Explorer Perspective**
- **DS-5 Debug Perspective**

C/C++ Perspective

This perspective is designed for working with C/C++ projects. By default, it consists of an editor area and views for project management, tasks, properties, and a console for messages.

The editor area of the C/C++ perspective in DS-MDK includes the Manage Run-Time Environment window that lets you select software components, target devices, and Software Packs for the current project.

It also features a graphical editor for files that have CMSIS Configuration Wizard annotations.

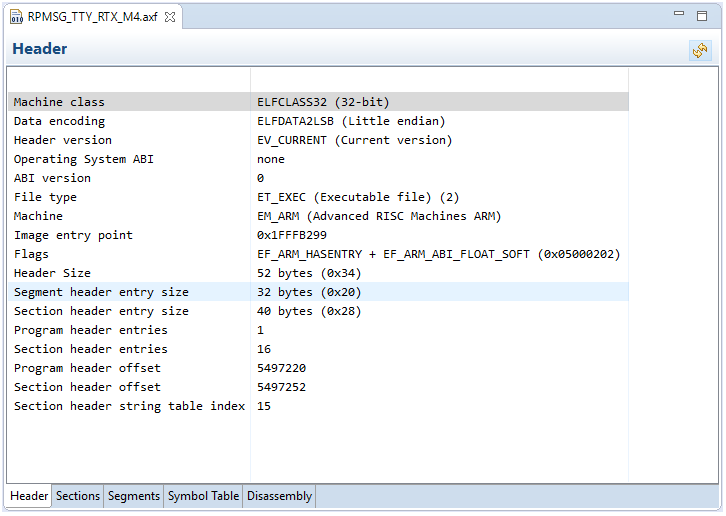


For more information, refer to the *C/C++ Development User's Guide* and the *CMSIS C/C++ Development User's Guide* available from the Eclipse help system (**Help → Help Contents**).

The C/C++ perspective contains views that are tailored for specific needs.

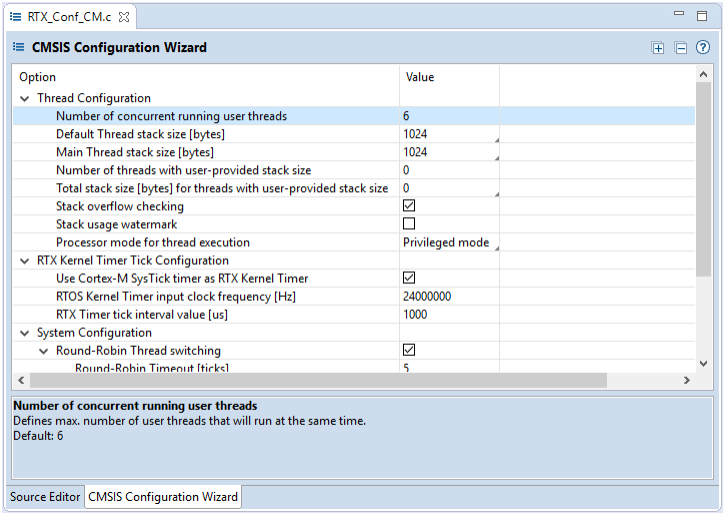
AXF File Viewer

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.



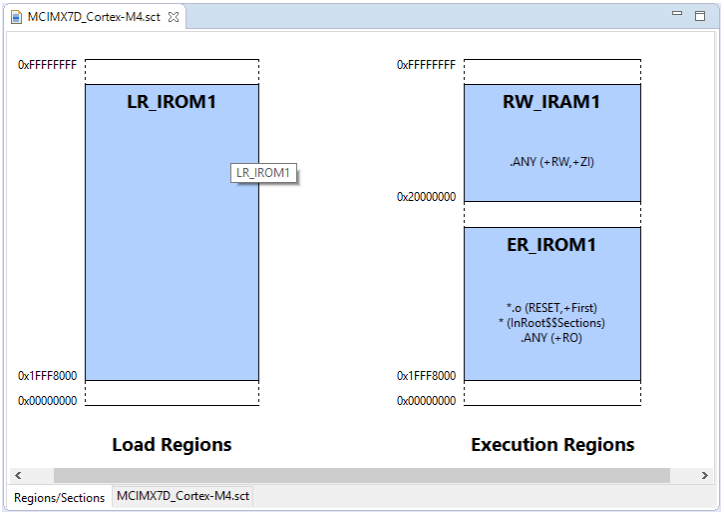
CMSIS Configuration Wizard

Files containing configuration annotations may be modified using a graphical editor. Right-click on a file in the Project Explorer and select **Open With → CMSIS Configuration Wizard**. 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. Edit the file contents using the *filename.sct* tab (refer to **Adapt the Scatter File** on page 32).

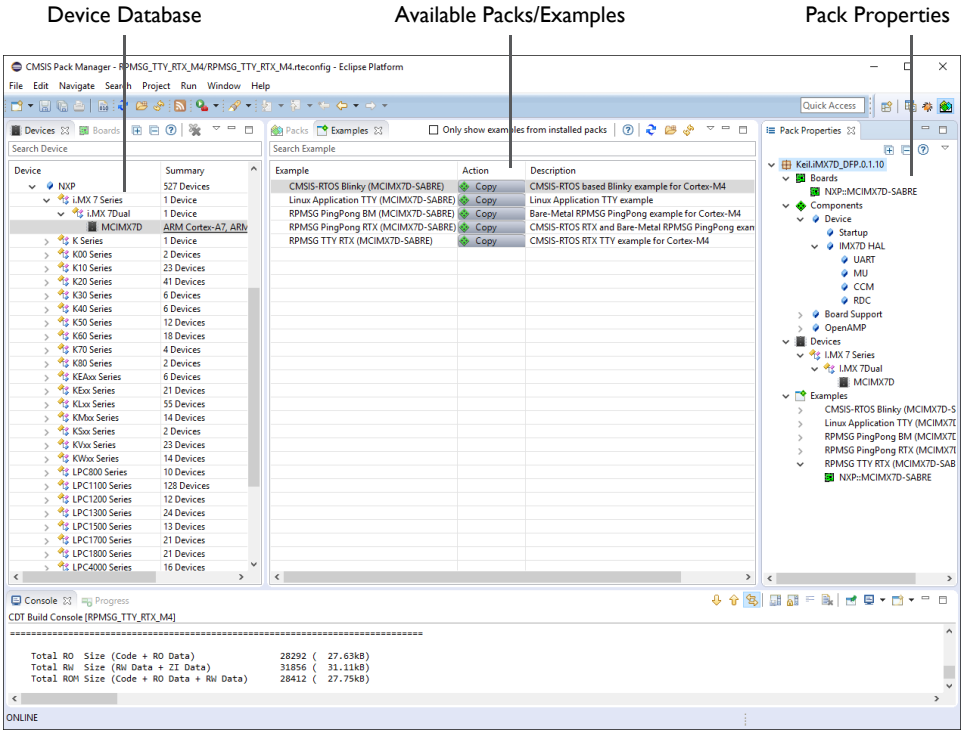


CMSIS Pack Manager Perspective

The Pack Manager perspective offers the following functionality:

- Install or update Software Packs.
- List devices and boards that are supported by Software Packs.
- List example projects that can be copied into the Eclipse workspace.

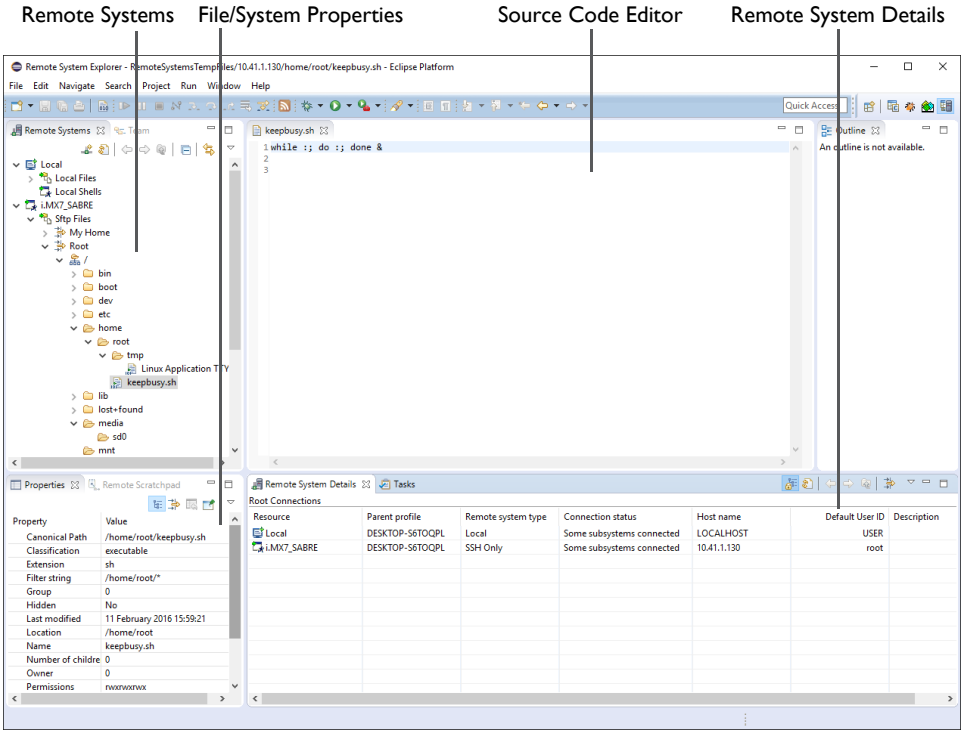
To open this perspective, use the  icon and select **CMSIS Pack Manager**



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

Remote System Explorer Perspective

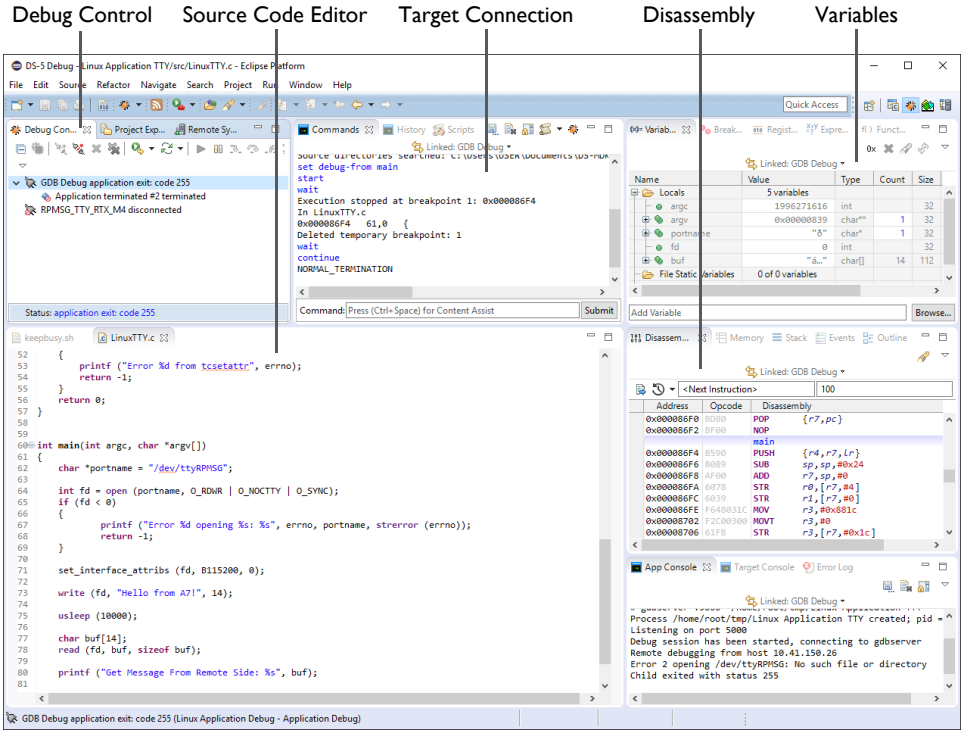
The Remote System Explorer (RSE) is a workbench perspective that allows you to connect and work with a variety of remote systems. With predefined plug-ins, you can look at remote file systems, transfer files between hosts, do remote search, execute commands and work with processes.



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

DS-5 Debug Perspective

Using DS-5 Debugger, you can 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-M4 Applications

This chapter guides you through the steps required to create and modify projects for the Cortex-M target in a heterogeneous system. The tutorial creates a project called *Blinky* using the real-time operating system CMSIS-RTOS RTX.

Blinky with CMSIS-RTOS RTX

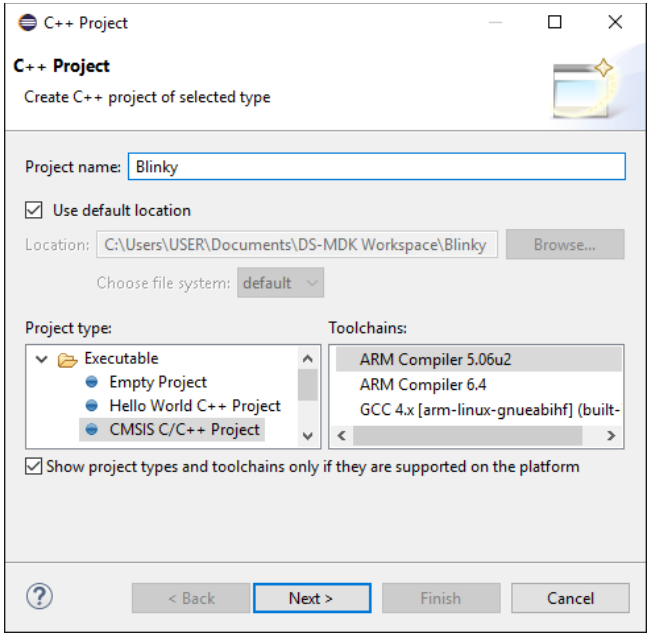
The section explains the creation of the project using the following steps:

- **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.

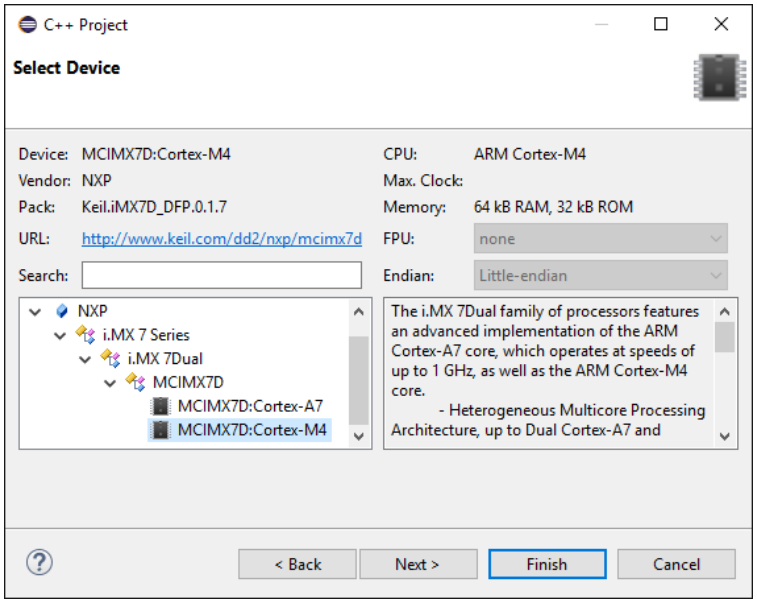
Setup the Project

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

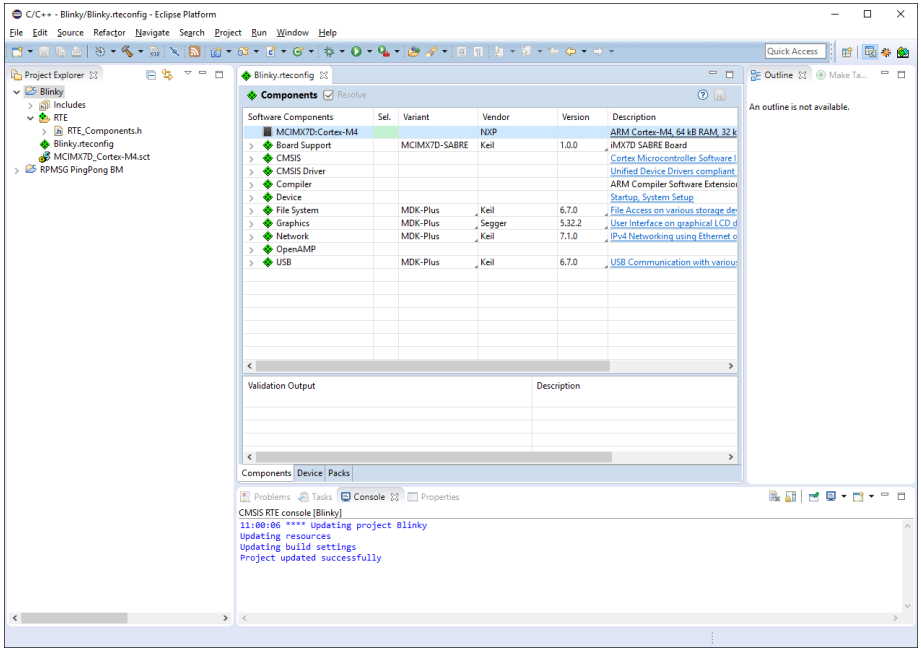


Select **CMSIS RTE C/C++ Project**, enter a project name (for example **Blinky**) and click **Next**. In the following window, you can select to create a default *main.c* file. Do not use this option. We will add a *main.c* template file later from a Software Pack, so click again **Next**.

In the following step, 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 will open and show the current project:

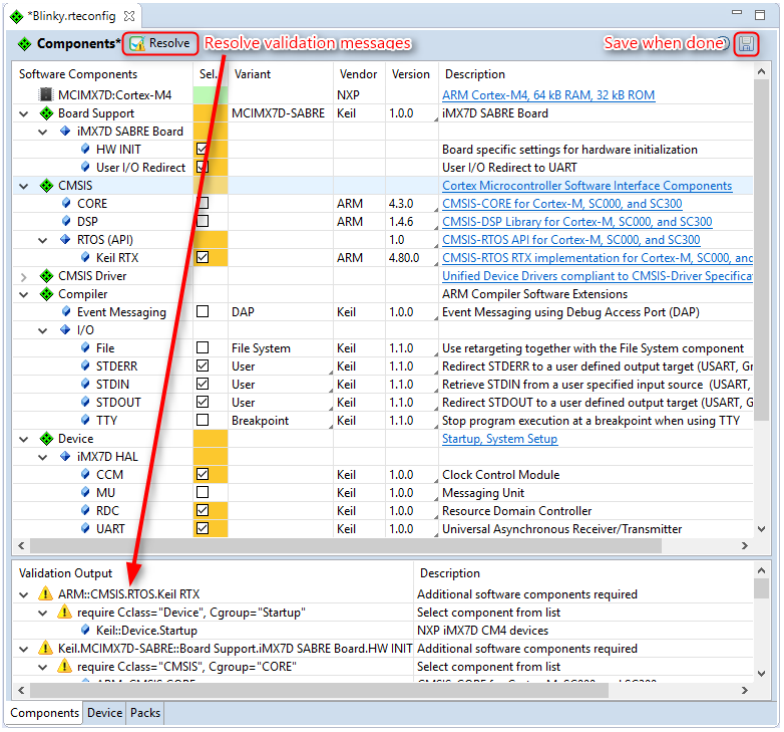


Select Software Components

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, s**ave** your selection:



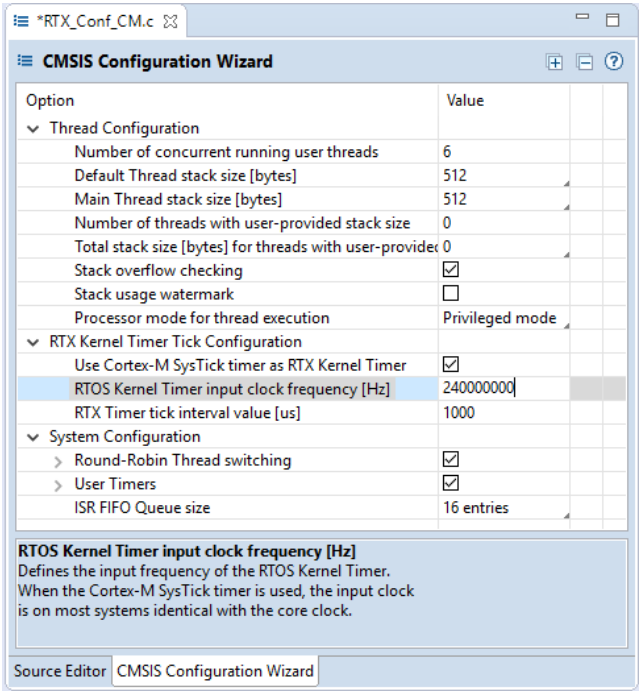
NOTE

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



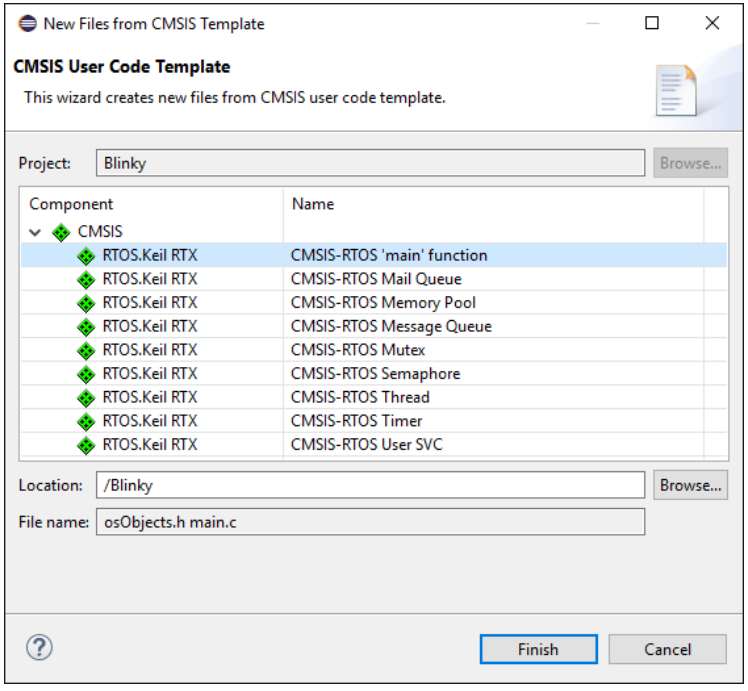
Save the file.

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

Add your application code using pre-configured User Code Templates containing routines that resemble the functionality of the 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	128KB	0x00900000-0x0091FFFF	0x00900000-0x0091FFFF
TCMU	32KB	0x00800000-0x00807FFF	
TCML	32KB	0x007F8000-0x007FFFFF	0x1FFF8000-0x1FFFFFFF
OCRAM_S	32KB	0x00180000-0x00187FFF	0x00000000-0x00007FFF/ 0x00180000-0x00187FFF

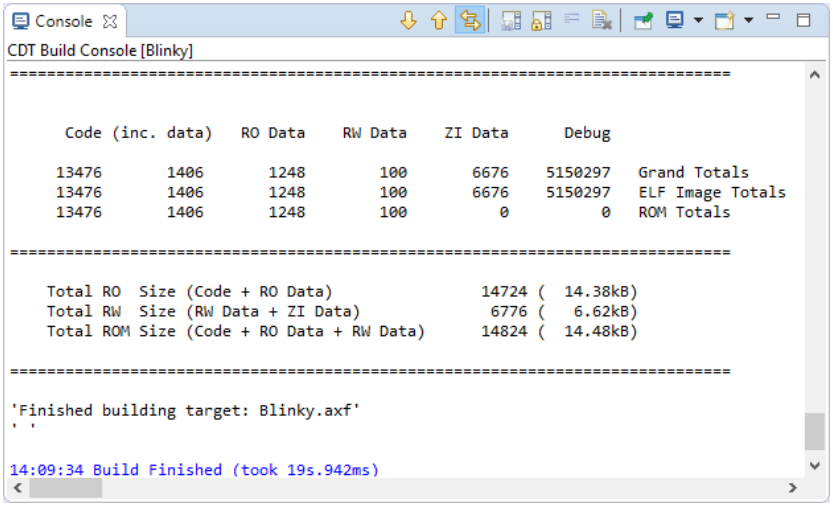
By default, the scatter file template uses the start address 0x0 for the load region command. To put the Cortex-M4 code into the TCM, change the address of the load region to 0x1FFF8000:

```
; *************************************************************
; ** 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-M4 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-M4 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.

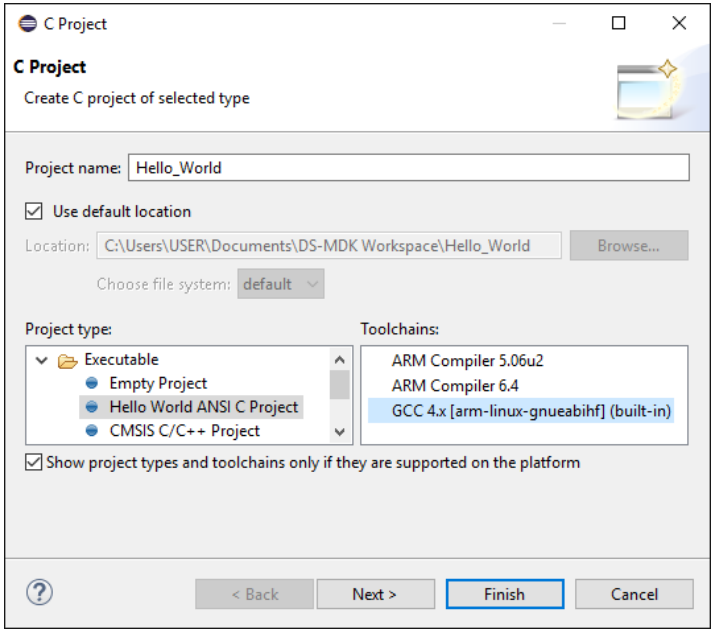
Create Linux Applications

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.

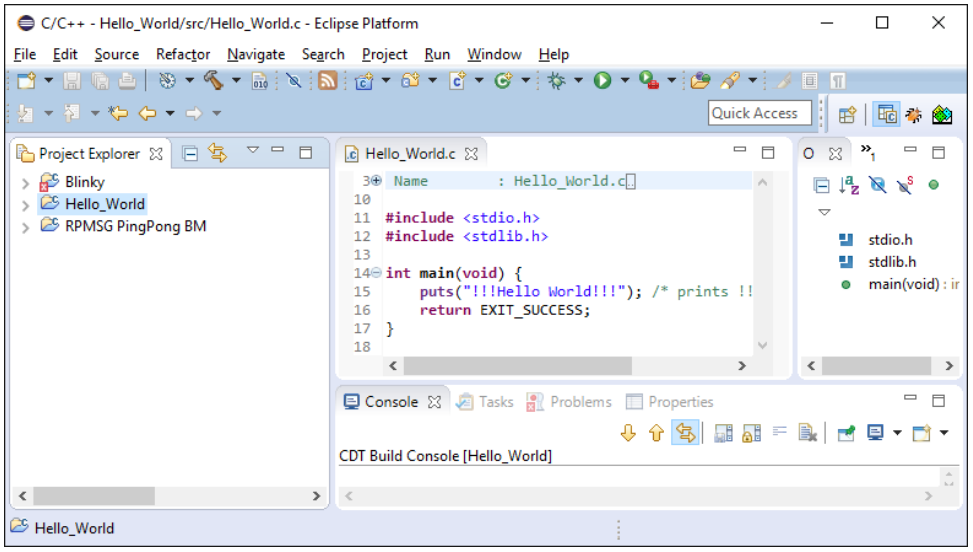
Setup the Project

From the Eclipse menu bar, choose **File → New → C Project**. In the upcoming window, select the **Hello World ANCI C Project**:



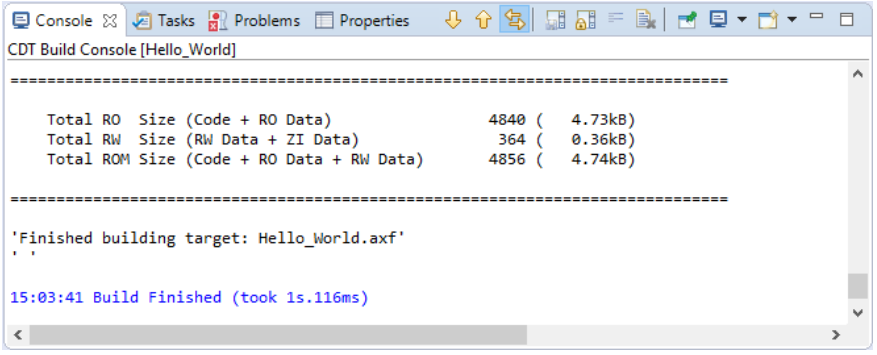
Enter a project name (for example `Hello_World`) and make sure that the **GCC [...] (built-in)** toolchain is selected before clicking **Finish**.

The C/C++ Perspective will open and show the current project:



Build the Application Image

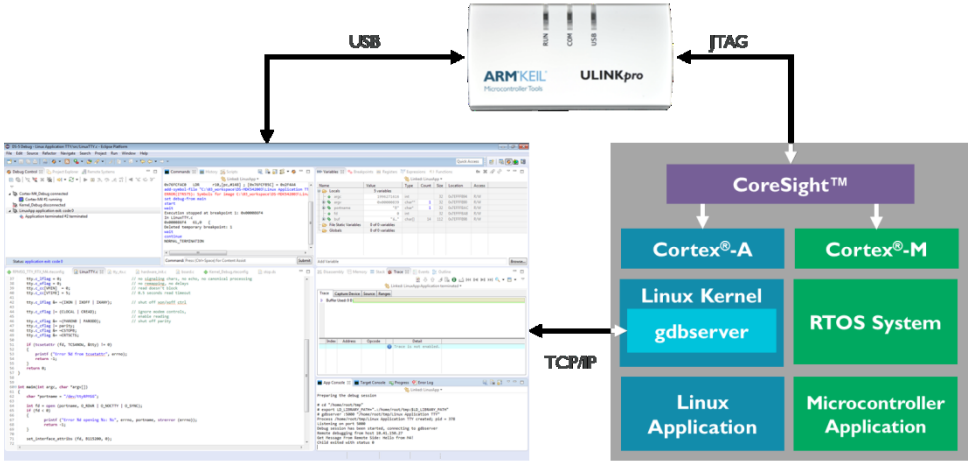
Right-click on the project name and select **Build Project** to build the application. This step compiles and links all related source files. The **Console** shows information about the build process



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. Complete system visibility is enabled using multiple simultaneous debug connections:

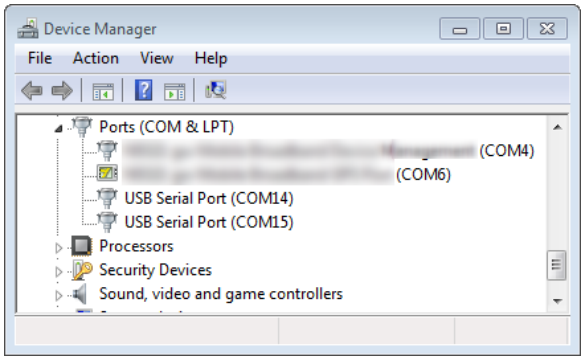


- The **Cortex-M application** is debugged using a ULINK*pro* debug unit (refer to **www.keil.com/ulink** for more information). Users can analyze the microcontroller application using RTOS aware-debugging and peripheral views.
- The **Cortex-A Linux kernel** is also debugged using a ULINK*pro* debug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

Preparing the Terminal Views

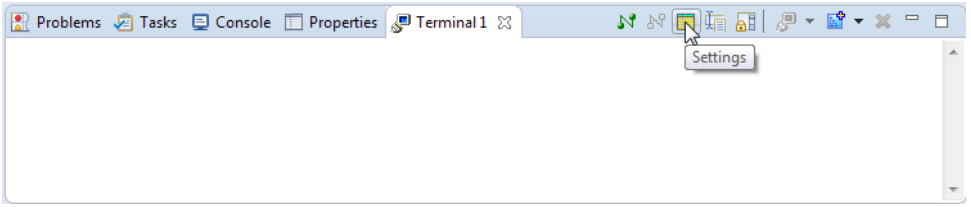
Many applications use a serial connection to display messages. To be able to view these messages, use a **Terminal** window that shows data coming from serial COM ports.

The i.MX 7 SABRE development board contains a dual USB serial port device that offers two independent COM ports. Connect the board to your computer and Windows will install the drivers that will add 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**.

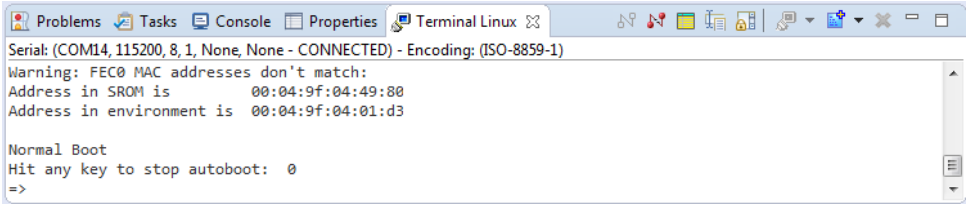
Open the settings dialog from the toolbar of the **Terminal 1** window:



Change the settings to the following:

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

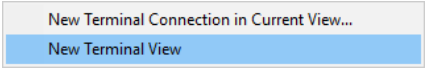
Click OK. Press the RST button on the development board to observe the boot process in the Terminal window. Send any keyboard key to the terminal window to interrupt the boot process:



NOTE

You must halt the boot loader at this point to be able to launch the Cortex-M4 debug session. Before starting to debug, copy and build the Linux application.

Add another Terminal view to display the output of the Cortex-M4 processor. Simply use the drop-down selector next to the **New Terminal Connection in Current View…** icon and select **New terminal View**:



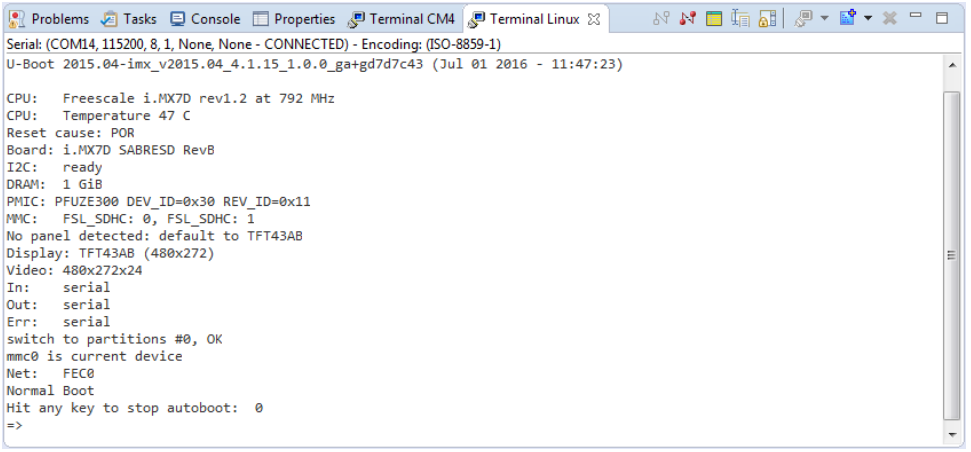
Select the larger COM port number and leave the other settings as they are. Name the Terminal view **Terminal M4**.

Debug Cortex-M4 Application

This section explains how to debug the microcontroller application running on the Cortex-M4. 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 using Example Project** chapter.

Stop in U-Boot

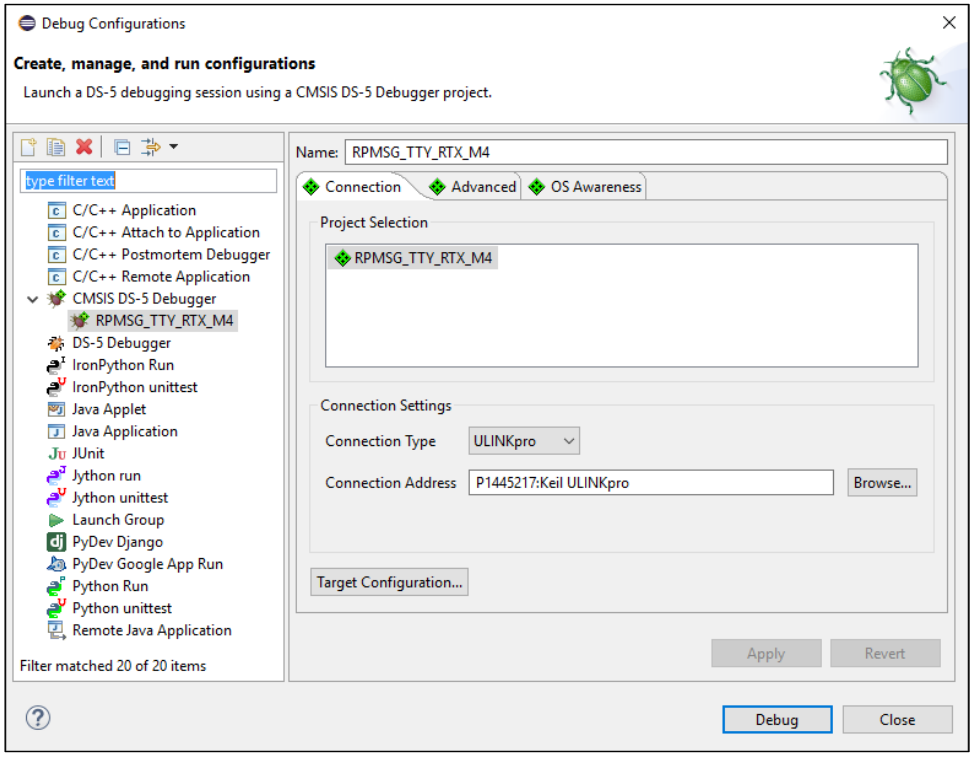
To be able to connect to the target, you need to stop U-Boot before it is actually loading the Linux kernel. Restart/reset the device and observe the bootloader output on the **Terminal Linux**. Press any key before the autoboot countdown expires:



Configure CMSIS DS-5 Debugger

Launch the DS-5 Debugger using the Cortex-M4 project context menu.

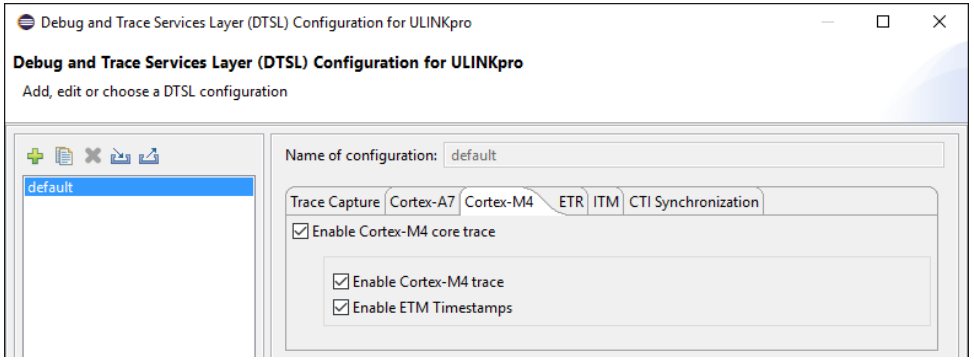
Right-click the **RPMSG_TTY_RTX_M4** project and select **Debug As → CMSIS DS-5 Debugger**:



Connection

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

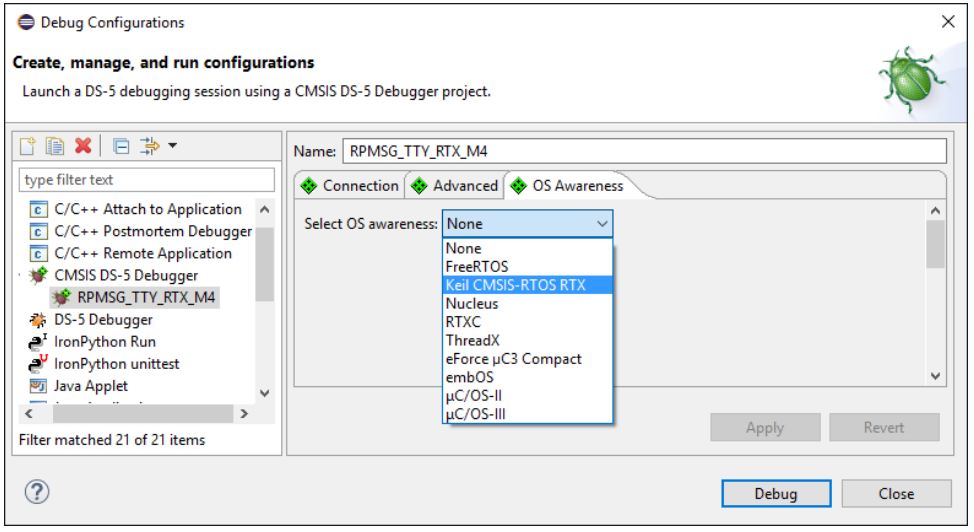
Click on **Target Configuration…** to setup the Debug and Trace Services Layer (DTSL).



- 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 chose the real-time operating system used in your application using the drop-down menu.



To start debugging, click **Debug**.

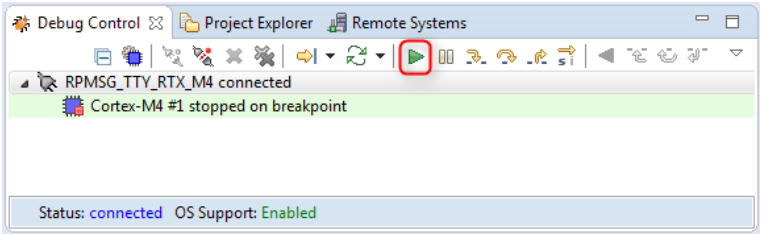
NOTE

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

Run Cortex-M4 Application

Starting the debugger, changes 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 simply opening it 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 using Example Project** chapter.

For Linux application debugging, the DS-5 Debugger supports connections to the target using gdbserver.

Before connecting, you must:

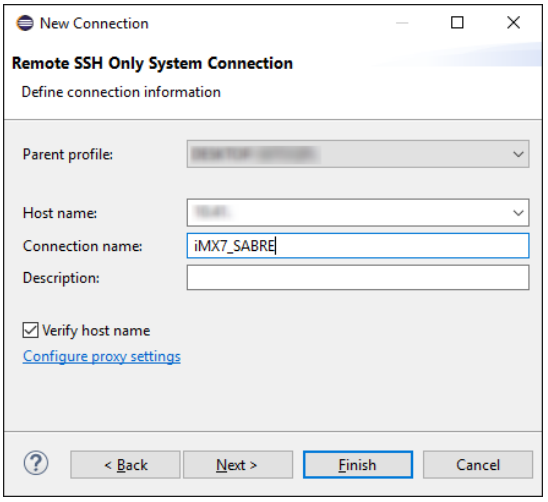
- Set up the target with Linux installed and booted. Refer to **Install the Linux Image** on page 12.
- Obtain the target IP address or name for the connection between the debugger and the debug hardware adapter. If the target is in your local subnet, click Browse and select your target.

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

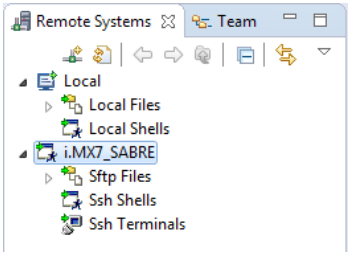
Setup RSE Connection

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

RSE is using Ethernet to communicate with the target, thus you need to enter the target's IP address into the **Host Name** field. Enter a meaningful name in the **Connection name** box:



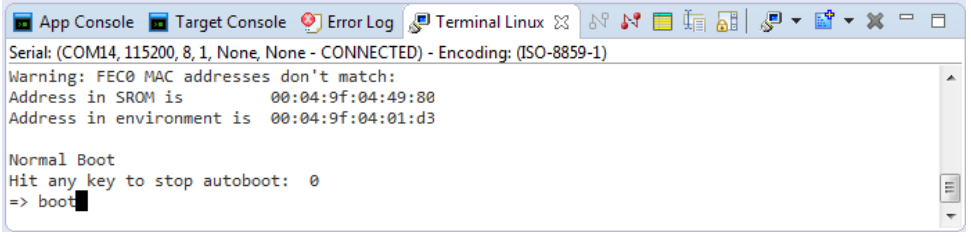
Click **Finish**. Your connection shows up in the **Remote Systems** window:



Boot Linux

NOTE
If you are debugging a microcontroller application simultaneously, you need to run *the Cortex-M4 application, otherwise the Linux Terminal will not be accessible and you will not be able to boot Linux.*

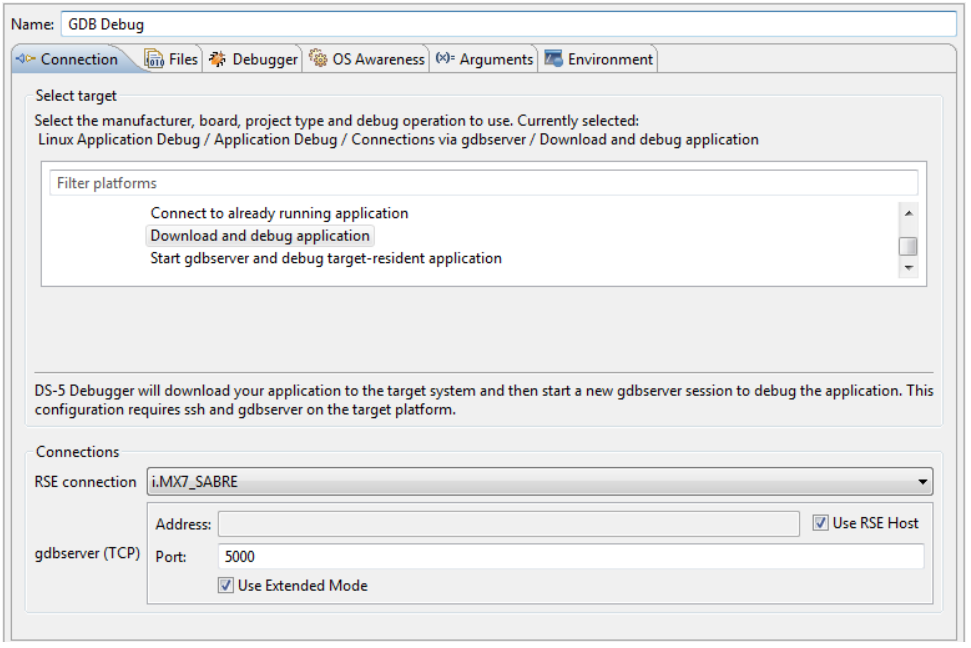
In the **Terminal Linux** enter “boot” to start the Linux system:



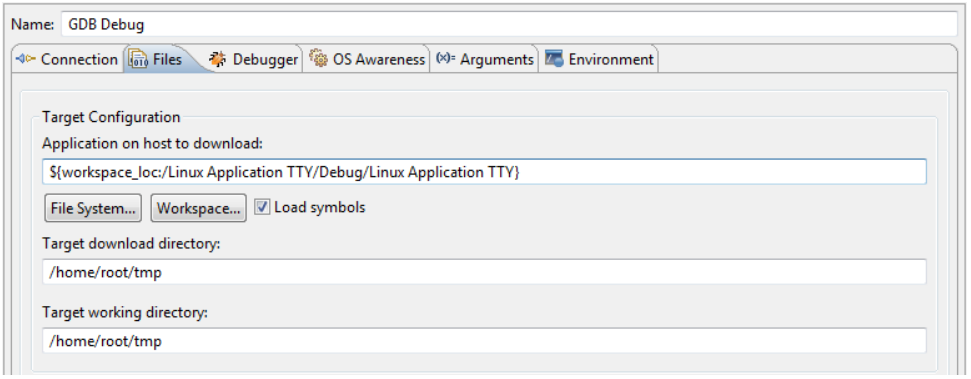
Once the Kernel is running, log in as **root** (no password required).

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 Debug* 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 Linux Application

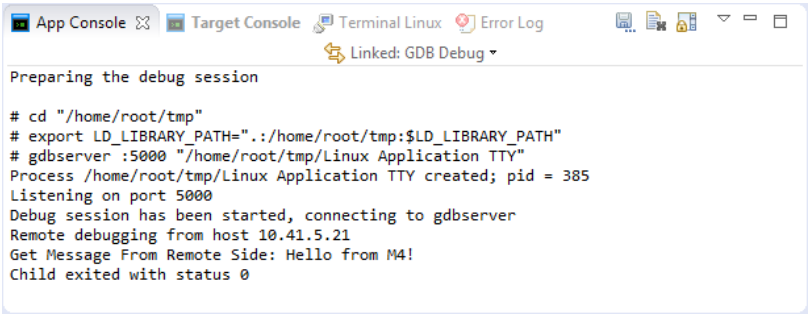
In the **Terminal Linux**, load the kernel module that communicates with the Cortex-M4 application using the following command:

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

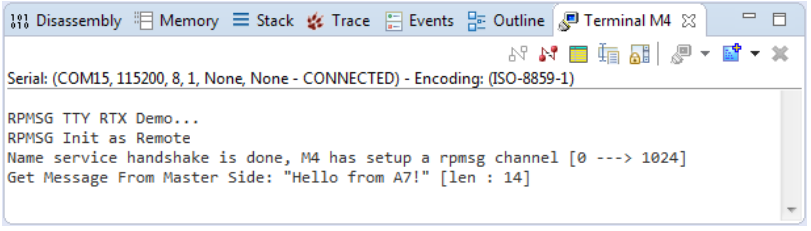
The kernel module should be loaded as shown below:

```
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!
```

Now you can run the Linux application by clicking the **Continue** button. The **App Console** shows the application's messages:



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



NOTE

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

Store Cortex-M4 Image

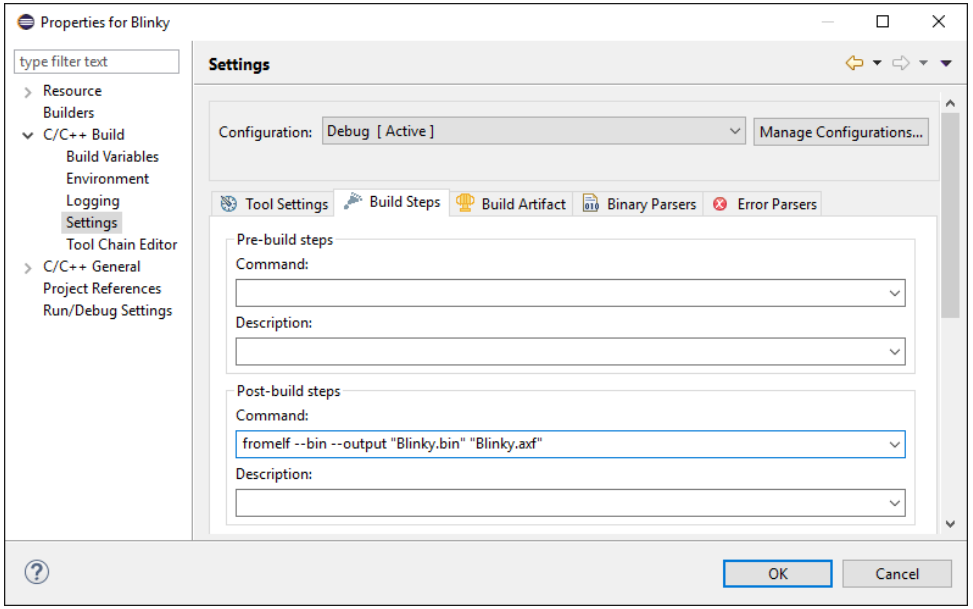
To store the Cortex-M4 Image for execution at start up use the following steps:

1. Create BIN image file using 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-M4 BIN Image File

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-M4 BIN Image File on SD Card

The SD Card has two partitions:

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

List them using 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
```

To execute the binary Cortex-M4 image at system startup, it must be stored in the **FAT32** boot partition using the following steps:

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 binary image 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-M4 BIN Image File from U-Boot

At this point, the Cortex-M4 BIN image file is stored in the **boot partition**.

Use the `setenv` command at the U-Boot prompt to change the U-Boot setup the new image file:

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

Verify the boot setup using the `printenv` command:

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

Run the m4boot process 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

C

D

E

F

I

L

P

R

S

T