

# Getting started with DS-MDK

Create applications for heterogeneous ARM<sup>®</sup> Cortex<sup>®</sup>-A/Cortex-M devices

C/C+++ Blinky/Blinky.teconfig Eclipse Platform							-		>	×	
He Edit Source Refactor Navigate	Search Project Run Windo	w He } -   ⊀	ilp \$ • 🚺 • 🏊 • [	ء 🖍 🍅	□ 1 1 × 20	• (⇔ (⇔ • ⇒ •	iick Access	e	<b>10 #</b>	<u>()</u>	I
🏠 Project Explorer 🙁 😐 🗖	♦ Blinky.rteconfig 😒 💷 RTX	Conf_C	M.c					0 23	»1	- 1	-
🖻 🛸 🔻	Second Components C Resolve						0 🔛				
V 🛃 Blinky	Software Components	Sel.	Variant	Vendor	Version	Description		An outl availabl	ne is no e.	ət	
> Epi includes	MCIMX7D-Cortex-M4			NXP		ARM Cortex-M4 64 kB RAM 32 kB ROM					
> 👝 Debug	Board Support		MCIMY7D-SARRE	Keil	100	MVTD SARPE Roard					
V 😸 KIE			WICHWATD-JADRE	roen.	1.0.0	Center Microsontroller Software Interface Components					
<ul> <li>Board_Support</li> </ul>	COPE			APAA	42.0	CMSIS_CORE for Contex_M_SC000_and SC200					
> Le board.c [Keil.MCIMX/I				ADM	4.5.0	CMSIS-CORE for Contex-IM, SC000, and SC300					
> Le clock_freq.c [Keil.MCII	• D3P			ANIVI	1.4.0	Christer Dispersion of Contex-IM, SCOOD, and SCOOD					
> [g pin_mux.c [Keil.MCIM.	V KIUS (API)				1.0	CMSIS-RIOS API for Cortex-M, SCUUD, and SC300					
> 🛃 retarget_io.c [Keil.MCI	V Keil RI X			AKM	4.80.0	CMSIS-RIOS RIX implementation for Cortex-M, SCUUU, a	nd SC300				
V 🗁 CMSIS	> CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specific	cations				
> RTX_Conf_CM.c [ARM	> 👽 Compiler					ARM Compiler Software Extensions					
RTX_CM3.lib [ARM::CN	> 📀 Device					Startup, System Setup					
> 👝 Compiler	> Sile System		MDK-Plus	Keil	6.7.0	File Access on various storage devices					
> 👝 Device	> I Graphics		MDK-Plus	Segger	5.32.2	User Interface on graphical LCD displays					
> h RTE_Components.h	> 💠 Network		MDK-Plus	Keil	7.1.0	IPv4 Networking using Ethernet or Serial protocols					
> 🖻 main.c	> 💠 OpenAMP										
> 🔝 osObjects.h	> 💠 USB		MDK-Plus	_ Keil	6.7.0	USB Communication with various device classes					
💠 Blinky.rteconfig	<						>				
MCIMX7D_Cortex-M4.sct	Validation Output				Description						
✓ ₩ Hello_World	fondution o deput				beschption						
🗸 🖾 src											
✓ I Hello_World.c											
🛃 stdio.h											
🛃 stdlib.h	<						>				
main(void) : int	Components Device Packs										
> 😂 RPMSG PingPong BM											_
	🕒 Console 🐹 🔎 Tasks 🔐 F	roblem	s Properties					1 🗉 🗝	<b>-</b>	- [	а
	CMSIS PTE concole (PDMSG Ding)	one Ph	41								
	10:05:22 **** Undating on	oloct	DDNEG DingDong	PM							-
	Loading BTE configuration	oject	KEIISG FILIGEOING	Diri							
	Updating resources										
	Updating build settings										
	Project updated successfu	11y									
											$\sim$
< >	<									>	

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 © 1997-2017 ARM Germany GmbH All rights reserved.

ARM, Keil, µVision, 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<sup>TM</sup> 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.

**Working with example projects** explains how to get started with supported development boards using pre-built projects to verify hardware and software functionality.

**Creating projects from scratch** guides you through the process of creating and modifying projects using CMSIS and device-related software components for the Cortex-M microcontroller. It also shows you how to develop 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.

The **Appendix** contains further information, for example about the basic concepts of the Eclipse IDE and the most frequently used perspectives.

## Contents

Preface	
DS-MDK introduction	7
Solution for heterogeneous systems	7
DS-MDK licensing	8
Software and hardware requirements	8
Install DS-MDK	9
Documentation and support	
Working with example projects	
Install the Linux image	13
Hardware connection	14
Verify installation with example projects	15
Cortex-M application	
Cortex-A Linux application	23
Creating projects from scratch	
Create Cortex-M applications	
Blinky with CMSIS-RTOS RTX	
Create Linux applications	
Setup the project	
Build the application image	
Debug applications	
Debug Cortex-M application	41
Debug Linux application	45
Debug the Linux Kernel	46
Create a Linux Kernel debug project	47
Debug the Kernel: Pre-MMU stage	51
Debug the Kernel: post-MMU stage	
Debug a Linux Kernel module	55
Create a Linux Kernel module debug project	55
Debug the Kernel module	57
ARM Streamline	58
Store Cortex-M image	60
Create a Cortex-M binary image (BIN)	60
Store Cortex-M BIN file on SD Card	61
Run Cortex-M BIN file from U-Boot	62
Appendix	
Eclipse IDE	64
Perspectives	64

Additional links	.70
ndex	71

#### 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 devices.* 

Refer to the **Getting Started with MDK** user's guide for information how to create projects for ARM Cortex-M microcontrollers with the  $\mu$ 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.

Currently NXP i.MX 6, i.MX7 and VFxxx series devices are supported. These devices combine computing power for application-rich systems with real-time responsiveness: 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 (<u>CMSIS</u>) 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 <u>Keil® MDK</u> and the product requires a valid license in order to use it.

For information on how to obtain and set-up the license, please refer to the following page: <u>http://www.keil.com/mdk5/ds-mdk/licensing/</u>

### Software and hardware requirements

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

- A workstation running Microsoft Windows, Red Hat Enterprise Linux or Ubuntu Desktop Edition (only 64-bit OS/platforms are supported)
- Dual-Core Processor with > 2 GHz
- 4 GB RAM and 8 GB hard-disk space
- 1280 x 800 or higher screen resolution

### **Install DS-MDK**

Download the **DS-MDK** installer for your host platform (Windows or Linux) from **www.keil.com/mdk5/ds-mdk/install**.

The installation procedures for Windows and Linux are different and are both described below.

### Windows installation

Decompress the zip archive and run the installer setup.exe. Follow the instructions on the screen and make sure you install the device drivers for the debug probes.

To start DS-MDK, use **Eclipse for DS-MDK** from the Start menu (Windows 10: All apps  $\rightarrow$  ARM DS-MDK  $\rightarrow$  Eclipse for DS-MDK).

### Linux installation

Extract the installer from the downloaded archive file, run (not source) *install.sh* and follow the on-screen instructions. The installer unpacks DS-MDK into your chosen directory, and optionally installs device drivers and desktop shortcuts.

Note: The installer includes device drivers that require you to run with root privileges.

To start DS-MDK, from your desktop, select Eclipse for DS-MDK. Alternatively, launch *[DS-MDK install directory]/bin/eclipse* from the command line.

#### **Run DS-MDK**

The first time you run DS-MDK, a window would appear asking 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.

Workspace Launcher	×	
Select a workspace		
Eclipse Platform stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.		
Workspace: C:\Users\USER\Documents\DS-MDK Workspace		V Browse
Use this as the default and do not ask again		
	ОК	Cancel

The Eclipse-based IDE opens in the *CMSIS Pack Manager* perspective and a warning message is shown if the default CMSIS Pack directory is empty.



Click on the highlighted *click here* text to start populating the CMSIS Index: this operation requires an Internet connection to download the index files.

DS-MDK shows a progress bar during the download.



At the end of the process, the CMSIS Pack Manager view should be populated with the CMSIS Packs available.

File Edit Navigate Searc	h Project Run Window Help						
· · · · · · · · · · · · · · · · · · ·	N						
■ Devices # ■ Boards type filter text		Be Packs № 1 Examples Be E					
🗸 🍕 All Devices	3894 Devices	Device Specific					
> ABOV	10 Devices	v • Generic	18 Packs	Software Packs with generic content not specific to a devi			
> 🍳 Ambiq Micro	6 Devices	> S ARM.CMSIS	🕸 Install	CMSIS (Cortex Microcontroller Software Interface Standar			
> Analog Devices	21 Devices	> H ARM.CMSIS-Driver_Valida	atic 🔄 Install	CMSIS-Driver Validation			
> ARM	40 Devices	> S ARM.CMSIS-FreeRTOS	Install+	Bundle of FreeRTOS for Cortex-M and Cortex-A			
> Atmel	271 Devices	> HARM.CMSIS-RTOS_Valida	tior Install	CMSIS-RTOS Validation			
> Cypress	425 Devices	> # ARM.mbedClient	🄄 Install	ARM mbed Client for Cortex-M devices			
> 🤗 GigaDevice	70 Devices	> "# ARM.mbedTLS	🕸 Install	ARM mbed Cryptographic and SSL/TLS library for Cortex-			
> 🔮 Holtek	22 Devices	> H ARM.minar	🔅 Install	mbed OS Scheduler for Cortex-M devices			
> 🔮 Infineon	166 Devices	> Huawei.LiteOS	🔅 Install	Huawei LiteOS kernel Software Pack			
> 🤗 Maxim	4 Devices	> 🏪 Keil_ARM_Compiler	🄄 Install	Keil ARM Compiler extensions for ARM Compiler 5 and A			
> Mediatek	2 Devices	> # KeilJansson	🌣 Install	Jansson is a C library for encoding, decoding and manipul			
Microsemi	6 Devices	> " Keil.MDK-Middleware	Install+	Middleware for Keil MDK-Professional and MDK-Plus			
MindMotion	2 Devices	> "# IwIP.IwIP	Install	IwIP is a light-weight implementation of the TCP/IP proto.			
Nordic Semicondu	ct 10 Devices	> Hicrium.RTOS	Install	Micrium software components			
> <sup>o</sup> Nuvoton	436 Devices	> " Oryx-Embedded.Middlev	var 🕸 Install	Middleware Package (CycloneTCP, CycloneSSL and Cyclon			
> 🔮 NXP	576 Devices	> 1 RealTimeLogic.SharkSSL-	Lite Install	SharkSSL-Lite is a super small and super fast pre-compile.			
Renesas	3 Devices	> 🏪 RealTimeLogic.SMQ	Install	Simple Message Queues (SMQ) is an easy to use IoT publi			
> 🤗 Silicon Labs	397 Devices	> "# YOGITECH.fRSTL_ARMCN	x_F Deprecat	III DEPRECATED Product III YOGITECH fRSTL Functional Sa			
> SONIX	35 Devices	> # YOGITECH.fRSTL_STM32F	x_E Deprecat	III DEPRECATED Product III YOGITECH fRSTL Functional Sa			
> STMicroelectronics	953 Devices						
> 🤗 Texas Instruments	342 Devices						
> 🍳 Toshiba	90 Devices						
> 🤗 Zilog	7 Devices						

#### NOTE

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

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

The device database (<u>www.keil.com/dd2</u>) 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 are or double-click on \*.PACK files to manually install software packs.

### **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 <u>www.keil.com/support</u>.

## Working with example projects

### Install the Linux image

For every supported development board, a pre-configured Linux image with DS-MDK specific debug settings is available. This web page lists all supported development boards: <a href="http://www.keil.com/mdk5/ds-mdk/install#boards">www.keil.com/mdk5/ds-mdk/install#boards</a>

Download the compressed Linux kernel for your development board and unzip it.

### Copy the Linux image to an SD-Card (Windows)

Download and install the open source tool Win32 Disk Imager from <u>http://win32diskimager.sourceforge.net/</u> to flash the Linux kernel image onto an SD-Card.

Run the program. 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:

Image File	-			Device
re mage lase multi	alexant 20 kill	roc	otfs.sdcard	3
_				
Copy 🗌 MD5 Hash	:			
Copy MD5 Hash	:			
Copy MD5 Hash	:			

### Copy the Linux image to an SD-Card (Linux)

To write the image on the memory card on Linux it's sufficient to use the dd command where /dev/sdx is the device for your memory card.

```
NOTE
```

Make sure you select the right /dev/sdx device to avoid corruption of your data on your drives.

```
# sudo dd if=image_file_name of=/dev/sdx bs=1M`
```

### Hardware connection

In order to fully debug the target device you need to use a JTAG debugger such as DSTREAM or ULINK*pro*. The debugger needs to be connected to the host PC via USB (DSTREAM/ULINK*pro*) or Ethernet (DSTREAM only) and the target board via JTAG connector.

For the debug of Linux applications via gdbserver an Ethernet connection from the host PC to the board is required.

Another required connection during debug is the UART port used to interact with the Linux console: some boards have an RS232 connector whereas others have an USB interface that the operating system recognizes as virtual COM ports.

The picture below shows an example (NXP i.MX7 SABRE board) connected with JTAG, Ethernet and USB UART connections.



If you are not sure how to connect your board, please follow the instructions on the development board's support page.

### 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. For more information about the example used in this section, please refer to Remote Processor Messaging protocol example on page 63 in the Appendix.

### Prepare terminal views

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

The NXP i.MX 7 SABRE development board for example contains a dual USB serial port device with two independent serial ports. The configuration of the serial port is slightly different between Windows and Linux platforms.

### Windows

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.

### Linux

Connect the board to your computer. Linux should recognize the peripheral and you should be able to find ttyUSB0 and ttyUSB1 in your /dev/ directory.

Please make you set the right read/write permission to the device. For example, to give read/write permissions to all users on your machine type the following command:



The first device (e.g. /dev/ttyUSB0) is the serial port of the Cortex-A processor, while the second device (e.g. /dev/ttyUSB1) is the serial port of the Cortex-M processor.

### Windows and Linux

Image: On DS-MDK, go to Window → Show View → Other... to open a Terminal view. Select Terminal → Terminal and click OK.

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

Problems	🧟 Tasks	Console	Properties	🖉 Terminal 1 🔀	14	N 🖪 🖬 🚮	🖉 👻 蒙	×
						Settings		*
								-

🗇 Set the following and click OK:

- View Title: Terminal Linux
- Connection Type: Serial
- Port: Use the first of the new serial ports (e.g. COM14 or /dev/ttyUSB0)
- Baud Rate: 115200

#### NOTE

For the correct terminal settings and hardware connections of your development board refer to the board support pages.

Press the reset button on the development board to observe the boot process in the *Terminal* window. Press any keyboard key to interrupt the boot process:



#### NOTE

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

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:

```
New Terminal Connection in Current View...
New Terminal View
```

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

### **Cortex-M** application

### Copy the RPMSG TTY RTX example project

Select the device

In the CMSIS Pack Manager (
) perspective, select the board (MCIMX7D-SABRE) from the Boards tab on the left and click on Examples tab on the right-hand side of the window. Use filters in the toolbar to narrow the list of examples.

🔳 Devices 📓 Boards 🛱	▣ ▣   🕷   ? ▽ ▫ ▫	Acks ■ Examples ×		Only show examples from installed packs
SABRE	<i>I</i> _	Search Example		
Board	Summary	Example	Action	Description
✓ <sup>4</sup> S All Boards	219 Boards	CMSIS-RTOS Blinky (MCIMX7D-SABRE)	Install	CMSIS-RTOS RTX Blinky example for Cortex-M4
<ul> <li>MCIMX6SX-SABRE (Rev. A2)</li> </ul>	MCIMX6X1	CMSIS-RTOS2 Blinky (MCIMX7D-SABRE)	Install	CMSIS-RTOS2 RTX5 Blinky example for Cortex-1
4 Mounted Devices	1 Device	Frequency Bin (MCIMX7D-SABRE)	Install	CMSIS-RTOS RTX, CMSIS-DSP Lib, ADC and RPN
<ul> <li>MCIMX7D-SABRE (Rev B)</li> </ul>	MCIMX7D7	Linux Application TTY (MCIMX7D-SABRE)	Install	Linux Application TTY example
4 Mounted Devices	1 Device	RPMSG PingPong BM (MCIMX7D-SABRE)	Install	Bare-Metal RPMSG PingPong example for Corte
4 Compatible Devices	5 Devices	RPMSG PingPong RTX (MCIMX7D-SABRE)	Install	CMSIS-RTOS RTX RPMSG PingPong example for
		RPMSG TTY CMSIS-RTOS (MCIMX7D-SAB	🕸 Install	CMSIS-RTOS RTX TTY example for Cortex-M4
		RPMSG TTY CMSIS-RTOS2 (MCIMX7D-SA	Install	CMSIS-RTOS2 RTX5 TTY example for Cortex-M4

Click **Install** next to the **RPMSG TTY RTX** example if the packs are not installed (this might take a few minutes based on your internet connection).

Installing Pack Keil.iMX7D_DFP.1.5.1		×
Operation in progress		
Downloading Keil.iMX7D_DFP.1.5.1.pack from http://www.keil.com/pack/Keil.iMX7D_DFP.1.5.1.pack		
Always run in background		
Run in <u>Background</u> Cancel	<u>D</u> etails >	>

At the end of the installation the CMSIS Packs for the selected board should be installed locally and the examples are ready to be copied in your workspace.

Example	Action	Description
CMSIS-RTOS Blinky (MCIMX7D-SABRE)	🚸 Сору	CMSIS-RTOS RTX Blinky example for Cortex-M4
CMSIS-RTOS2 Blinky (MCIMX7D-SABRE)	🕸 Сору	CMSIS-RTOS2 RTX5 Blinky example for Cortex-M4
Frequency Bin (MCIMX7D-SABRE)	🕏 Сору	CMSIS-RTOS RTX, CMSIS-DSP Lib, ADC and RPMSG TTY example for Cortex-M4
Linux Application TTY (MCIMX7D-SABRE)	🐓 Сору	Linux Application TTY example
RPMSG PingPong BM (MCIMX7D-SABRE)	💠 Сору	Bare-Metal RPMSG PingPong example for Cortex-M4
RPMSG PingPong RTX (MCIMX7D-SABRE)	🕏 Сору	CMSIS-RTOS RTX RPMSG PingPong example for Cortex-M4
RPMSG TTY CMSIS-RTOS (MCIMX7D-SAB	🔄 Сору	CMSIS-RTOS RTX TTY example for Cortex-M4
RPMSG TTY CMSIS-RTOS2 (MCIMX7D-SA	💠 Сору	CMSIS-RTOS2 RTX5 TTY example for Cortex-M4

Click **Copy** next to the **RPMSG TTY RTX** example (make sure the corresponding pack is installed).

Confirm your selection by clicking on the Copy button.



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

C/C++ - RPMSG_TTY_RTX_M4/RPMSG_TTY_RTX_M4.rts	- 0	×						
File Edit Source Refactor Navigate Search Project	t Run Window Help							
📑 • 🗄 🕼 🛎   🌣 • 🗞 • 🗟 🗙 🔝 🔞 •	ᢨ • € • 6 • † † • 0 •	<b>%</b>	• 😂 🖋 • 🗉	n i 2 - F	• • ÷ ÷ ÷	*	Quick Access 🔡 🔛 🕯	ŧ 幽
🎦 Project Explorer 🙁 🛛 🕒 😓 🤝 🗖 🗖	RPMSG_TTY_RTX_M4.rteconfig	23					🗄 Outline 🛿 🛞 Make Ta 🖛	
PPMSG_TTY_RTX_M4     Mincludes	🚸 Components 🖂 Resolve					0	An outline is not available	
V 🏂 RTE	Software Components	Sel.	Variant	Vendor	Version	Description	An outline is not available.	
> 🗁 Board_Support	MCIMX7D:Cortex-M4			NXP		ARM Cortex-M4, 64 kB RAM, 32 k		
> 🗁 CMSIS	> 🚸 Board Support		MCIMX7D-SABRE	Keil	1.0.0	iMX7D SABRE Board		
> 🗁 Compiler	> 🚸 CMSIS					Cortex Microcontroller Software I		
> 🗁 Device	> 🚸 CMSIS Driver					Unified Device Drivers compliant		
> 🗁 OpenAMP	> 💠 Compiler					ARM Compiler Software Extension		
> h RTE_Components.h	> 🚸 Device					Startup, System Setup		
> 🔓 hardware_init.c	> 🚸 File System		MDK-Plus	Keil	6.8.0	File Access on various storage dev		
> 🖻 tty_rbc	> 💠 Graphics		MDK-Plus	Segger	5.32.2	User Interface on graphical LCD d		
MCIMX7D_Cortex-M4.sct	> 💠 Network		MDK-Plus	Keil	7.2.0	IPv4 Networking using Ethernet o		
RPMSG_TTY_RTX_M4.rteconfig	> 💠 OpenAMP							
	> 🚸 USB		MDK-Plus	Keil	6.8.0	USB Communication with various		

### **Build the application**

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

🏠 Project Explorer 🛛	🕒 🔄 🏷 🖳 🗖 🚸 R	RPMSG_TTY_RTX_M4.rteconfig 🔀	
V 🖉 RPMSG_TTY_R	TX_M4 🔺	Components 🖌 Resolve	
<ul> <li>&gt; minclu     <li>minclu</li> <li>minclu<!--</td--><td>New Go Into Open in New Window Copy</td><td>&gt; mponents Sel. IMX7D:Cortex-M4 rd Support SIS</td><td></td></li></li></ul>	New Go Into Open in New Window Copy	> mponents Sel. IMX7D:Cortex-M4 rd Support SIS	
	Paste Delete Move Rename	SIS Driver npiler ice System phics	
RPM:	Import Export CMSIS C/C++ Project	enAMP	
•	Build Project Clean Project		

The Console window shows information about the build process:

🖹 Problems 🧔 Tasks 📮 Console 🔀 🔲 Properties	- 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
CDT Build Console [RPMSG_TTY_RTX_M4]		
		^
Total RO Size (Code + RO Data) Total RW Size (RW Data + ZI Data) Total ROM Size (Code + RO Data + RW Data)	28276 ( 27.61kB) 31856 ( 31.11kB) 28396 ( 27.73kB)	
<pre>fromelfbinoutput=RPMSG_TTY_RTX_M4.bin R</pre>	PMSG_TTY_RTX_M4.axf	
15:55:26 Build Finished (took 37s.203ms)		
<		×

### Configure CMSIS DS-5 debugger

CMSIS DS-5 Debugger to launch the debug configurations dialog:

Debug Configurations				×
Create, manage, and run configuration Launch a DS-5 debugging session using a	o <b>ns</b> CMSIS DS-5 Debugger pro	ject.		Ť.
Image: Second Secon	Name: RPMSG_TTY_RTX Connection A Project Selection RPMSG_TTY_RTX Connection Settings Connection Settings Connection Address Target Configuration	M4 dvanced S OS Awareness M4 ULINKpro V P1445217:Keil ULINKpro	Apply	Browse
$\odot$			Debug	Close

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).

Debug and Trace Services Layer (D)	FSL) Configuration for ULINKpro	_	×
Debug and Trace Services Layer ( Add, edit or choose a DTSL configurat	DTSL) Configuration for ULINKpro		
💠 🖹 🗶 🖄 default	Name of configuration:       default         Trace Capture       Cortex-A7       Cortex-M4       ETR       ITM       CTI Synchronization         Image: Cortex-M4 core trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 core trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 core trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 core trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 core trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace       Image: Cortex-M4 trace         Image: Cortex-M4 trace		

- On the *Cortex-A7* tab, disable all trace options to avoid buffer overflows.
- On the *Cortex-M4* tab, check **Enable Cortex-M4 core trace**.
- G In the OS Awareness tab select the real-time operating system used in your application from the drop-down menu.

Debug Configurations		×
Create, manage, and run configurat Launch a DS-5 debugging session using	<b>ions</b> a CMSIS DS-5 Debugger project.	Ť
	Name: RPMSG_TTY_RTX_M4	
type filter text	Sconnection 🚸 Advanced 🚸 OS Awareness	
C C/C++ Attach to Application ∧ C C/C++ Postmortem Debugger C C/C++ Remote Application C CMSIS DS-5 Debugger RPMSG_TTY_RTX_M4 DS-5 Debugger IronPython Run IronPython nunittest Java Applet ∨	Select OS awareness: None None FreeRTOS Keil CMSIS-RTOS RTX Nucleus RTXC ThreadX eForce µC3 Compact embOS µC/OS-II µC/OS-II	×
Filter matched 21 of 21 items	Apply	Kevert
?	Debug	Close

#### 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  $\rightarrow$  Show View  $\rightarrow$  Terminal.

### **Cortex-A Linux application**

#### 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** in the same way we have done for the Cortex-M RPMSG TTY RTX example.

The Console should show an error-free build:



#### 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. Enter the target's IP address into the **Host Name** field. Enter a meaningful name in the **Connection name** box:

Define connection inf	ormation	 	
Parent profile:	DESKTOF (STORE)		
Host name:	10.41		
Connection name:	iMX7_SABRE		
Description:		 	_
Verify host name	ngs		

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 prompt in the Terminal Linux is not accessible.

In the *Terminal Linux* enter "boot" to start the Linux system if it hasn't started yet:



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

### Configure DS-5 debugger

G 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:

Connection	🛅 Files 🤺	🐕 Debugger   🎲 OS Awareness   🗱 Arguments   🖾 Environment	
Select target			
Select the manu	facturer, bo	ard, project type and debug operation to use. Currently selected:	
Linux Applicatio	on Debug / /	Application Debug / Connections via gdbserver / Download and debug application	
Filter platform	ms		
	Connect t	o already running application	
	Download	and debug application	
	Start gdbs	erver and debug target-resident application	-
DS-5 Debugger v configuration re Connections	vill downloa quires ssh a	ad your application to the target system and then start a new gdbserver session to debug nd gdbserver on the target platform.	the application. T
DS-5 Debugger v configuration re Connections RSE connection	will downloa quires ssh a i.MX7_SAE	ad your application to the target system and then start a new gdbserver session to debug ind gdbserver on the target platform. BRE	the application. T
DS-5 Debugger v configuration re Connections RSE connection	vill downloa quires ssh a i.MX7_SAE Address:	ad your application to the target system and then start a new gdbserver session to debug nd gdbserver on the target platform. BRE	the application. T
DS-5 Debugger ( configuration re Connections RSE connection gdbserver (TCP)	will downloa quires ssh a i.MX7_SAE Address: Port:	ad your application to the target system and then start a new gdbserver session to debug nd gdbserver on the target platform. BRE 5000	the application. T

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**).

ne: Cor	GDB Debug nnection [編 Files 🛛 拳 Debugger] 🏇 OS Awareness] 🕬= Arguments] 🚾 Environment]
Tar	aet Configuration
Ар	plication on host to download:
<b>\$</b> {	workspace_loc:/Linux Application TTY/Debug/Linux Application TTY}
Fi	le System) Workspace) 🗹 Load symbols
Tar	get download directory:
/h	ome/root/tmp
Tar	get working directory:
/h	ome/root/tmp

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

If asked for login, please insert the credential for the Linux target. If you are using one of the images downloaded from <u>www.keil.com</u> please use *root* as username and leave the password field empty.

#### **Run the Linux application**

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

root@imv7dsabresd:~# modprobe -v imx\_rpmsg\_tty

The kernel module will be loaded:

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

☑ Use the Continue ▶ button to run the Linux application. 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 using **Window**  $\rightarrow$  Show View  $\rightarrow$  Terminal.

You have verified that your development environment can connect to both the Cortex-M and the Cortex-A processor. Try other example projects such as the **Frequency Bin** that demonstrates how to use the CMSIS-DSP library in the Cortex-M processor. The following chapters will explain how to create projects for both from scratch and how to debug these applications.

## **Creating projects from scratch**

### **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.

C Projec	t		×
CMSIS C/	C++ Project		\$
Selected pr Output: Toolchain Toolchain:	oject type: Executable exe Adapter ARM Compiler 5		
Adapter:	Adapter for ARM C/C++ 5.x and 6.x toolchains		$\sim$
Family:	ARMCC V (passed model via Tcompiler filter attribute)		
Toolchain	adapter for ARMCC 5.x and 6.x compilers		
?	< <u>B</u> ack <u>N</u> ext > Einish	Cancel	

Select your target device from the list: in this example we would continue using **MCIMX7D7:Cortex-M4**. Make sure the selection on **FPU** is none so that we can avoid initializing it for our example.

⊖ C Project		—		×
Select Device				
Device: MCIMX7D7:Cortex-M4 Vendor: NXP Pack: Keil.iMX7D_DFP.1.5.1 URL: <u>http://www.keil.com/dd2/nxp/</u> Search:	CPU: Max. Clock: Memory: FPU: Endian:	ARM Cortex-M4 64 kB RAM, 32 kB RC none Little-endian	DM	<pre></pre>
<ul> <li>ARM</li> <li>NXP</li> <li>i.MX 7 Series</li> <li>i.MX 7 Dual</li> <li>i.MX 7D3</li> <li>MCIMX7D3</li> <li>MCIMX7D5</li> <li>MCIMX7D7</li> <li>MCIMX7D7.Cortex-A7</li> <li>MCIMX7D7.Cortex-M4</li> <li>MCIMX7Solo</li> </ul>	The i.MX 7 an advance Cortex-A7 up to 1 GH core. - He Architectu Cortex-M4 - Ex DDR3/DD - Fla ECC), Man - Ele	Dual family of proces ed implementation of core, which operates 4z, as well as the ARM eterogeneous Multicor re, up to Dual Cortex- 4 configuration ternal Memory Support R3L/LPDDR2/LPDDR3 ish Memory Support: N aged NAND (eMMC, etrophoretic Display (E	sors features i the ARM at speeds of Cortex-M4 e Processing A7 and rt: VAND (60-bit eSD) EPD) Controll	er v
? < <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel	

G Select the NXP → i.MX 7 Series → i.MX Dual → MCIMX7D7 → MCIMX7D:Cortex-M4 device and click Finish. The C/C++ Perspective opens and shows the project:

C/C++ - Blinky/Blinky.rteconfig - Eclipse Platform						– 🗆 ×
Eile Edit Source Refactor Navigate Search Proje	ct <u>R</u> un <u>W</u> indow <u>H</u> elp					
🖆 • 🗄 🐚 🛎   🗞 • 🗞 • 🗟 i 🗙 i 🔂 🗹 •	°° • ° • ° • † * • O	• 💁 • 🙋 🖋 • 🗉	n (a - 9	• \$ \$ • •	> <del>•</del>	Quick Access 🔡 🔛 🗟 🏶 🆄
🎦 Project Explorer 🛛 📄 🔄 🌣 🔍 🗖	🚸 Blinky.rteconfig 😒					📴 Outline 💥 🛞 Make Ta 📟 🗖
V 🐸 Blinky	🚸 Components 🕑 Resolve				?	An evalue is not evaluated
V ARTE	Software Components	Sel. Variant	Vendor	Version	Description	An outline is not available.
> h RTE_Components.h	MCIMX7D:Cortex-M4		NXP		ARM Cortex-M4, 64 kB RAM, 32 k	
Blinky.rteconfig	> 🚸 Board Support	MCIMX7D-SABRI	Keil	1.0.0	iMX7D SABRE Board	
MCIMX7D_Cortex-M4.sct	> 💠 CMSIS				Cortex Microcontroller Software I	
> 🥂 RPMSG PingPong BM	> 💠 CMSIS Driver				Unified Device Drivers compliant	
	> 🚸 Compiler				ARM Compiler Software Extension	
	> 💠 Device				Startup, System Setup	
	> 🚸 File System	MDK-Plus	Keil	6.7.0	File Access on various storage dev	
	> 💠 Graphics	MDK-Plus	Segger	5.32.2	User Interface on graphical LCD d	
	> 💠 Network	MDK-Plus	Keil	7.1.0	IPv4 Networking using Ethernet o	
	> 💠 OpenAMP					
	> 💠 USB	MDK-Plus	Keil	6.7.0	USB Communication with various	
	<				>	
	Validation Output			Description		
	<				>	
	Components Device Packs					
	📳 Problems 🦃 Tasks 🔲 Cor	sole 🛛 🔲 Properties				🔒 📰 📑 🖬 = 🖬 = 🗆
	CMSIS PTE console (Plinku)					
	11:00:06 **** Undation on	oject Blinky				4
	Updating resources	-,,				
	Updating build settings					
	Project updated successfu	11y				
						2
< >	<					

#### Select software components

- For the **Blinky** project based on CMSIS-RTOS RTX, you need to select the following components:
  - Board Support:iMX7D SABRE Board:HW INIT
  - Board Support: iMX7D SABRE Board: User I/O Redirect
  - CMSIS:RTOS (API):Keil RTX.
  - Compiler:I/O:STDERR configured as variant User
  - Compiler:I/O:STDIN configured as variant User
  - Compiler:I/O:STDOUT configured as variant User
  - Compiler:I/O:TTY configured as variant User
  - Device:i.MX7D HAL:CCM
  - Device:i.MX7D HAL:RDC
  - Device:i.MX7D HAL:UART

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

▶ *Blinky.rteconfig 🔀					] 🗆	
Tomponents* 🕞 Resolve	Re	solve validatio	n messa	iges	Save when done) 🔚	)
Software Components	Sel.	Variant	Vendor	Versio	on Description	^
MCIMX7D:Cortex-M4			NXP		ARM Cortex-M4, 64 kB RAM, 32 kB ROM	
<ul> <li>Soard Support</li> </ul>		MCIMX7D-SABRE	Keil	1.0.0	iMX7D SABRE Board	
<ul> <li>iMX7D SABRE Board</li> </ul>						
HW INIT					Board specific settings for hardware initialization	
User I/O Redirect	51				User I/O Redirect to UART	
V 💠 CMSIS					Cortex Microcontroller Software Interface Components	
CORE	b		ARM	4.3.0	CMSIS-CORE for Cortex-M, SC000, and SC300	
DSP			ARM	1.4.6	CMSIS-DSP Library for Cortex-M, SC000, and SC300	
RTOS (API)				1.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300	
Keil RTX			ARM	4.80.0	CMSIS-RTOS RTX implementation for Cortex-M, SC000, and	
> 🚸 CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifica	
🗸 💠 Compiler					ARM Compiler Software Extensions	
Event Messaging		DAP	Keil	1.0.0	Event Messaging using Debug Access Port (DAP)	
✓ ♦ I/O						
🔗 File		File System	Keil	1.1.0	Use retargeting together with the File System component	
STDERR	$\checkmark$	User	Keil	1.1.0	Redirect STDERR to a user defined output target (USART, Gr	
STDIN	$\checkmark$	User	Keil	1.1.0	Retrieve STDIN from a user specified input source (USART,	
STDOUT	$\checkmark$	User	Keil	1.1.0	Redirect STDOUT to a user defined output target (USART, G	
🖉 TTY		Breakpoint	Keil	1.1.0	Stop program execution at a breakpoint when using TTY	
🗸 💠 Device					Startup, System Setup	
🗸 🔶 iMX7D HAL						
🖉 ССМ			Keil	1.0.0	Clock Control Module	
MU			Keil	1.0.0	Messaging Unit	
🖉 RDC			Keil	1.0.0	Resource Domain Controller	
🖉 UART			Keil	1.0.0	Universal Asynchronous Receiver/Transmitter	¥
<					>	
Validation Output					Description	^
	TV				Additional and the second and the second	
ARM::CMSIS.RTUS.Kell R	1X				Additional software components required	
Fequire Colass= Devi	ce, cg	roup= startup			VICE INVESTIGATION INT	
Keil::Device.Starti	nb dr		Deserved Line		NAP IIVIA/D CIVI4 devices	
Kell.MCIMX/D-SABRE::B	oard Su	ipport.IMX/D SABRE	board.HV		Additional software components required	
require Cclass="CMS"	is, Cg	roup= CORE"			select component from list	¥
<					>	

#### 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

Ontion		Value	
. Thread (	Configuration		
<ul> <li>Mum</li> </ul>	ber of concurrent running user th	reads 6	
Defa	ult Thread stack size [bytes]	512	
Main	Thread stack size [bytes]	512	-
Num	her of threads with user-provided	stack size 0	-
Total	stack size [bytes] for threads with	user-provider 0	
Stack	overflow checking		-
Stack	usage watermark		
Proc	essor mode for thread execution	Privileged mor	de
RTX Kerr	el Timer Tick Configuration	i nviicgeu nio	uc _
Use (	Cortex-M SysTick timer as RTX Kerr	nel Timer 🔽	
RTOS	Kernel Timer input clock frequence	ry [Hz] 24000000	_
RTX	Timer tick interval value [us]	1000	_
<ul> <li>System (</li> </ul>	Configuration	1000	
> Rour	d-Robin Thread switching		
> User	Timers		
ISR F	IFO Queue size	16 entries	
TOS Kerne efines the ir When the Co on most sy	I Timer input clock frequency [H nput frequency of the RTOS Kernel rtex-M SysTick timer is used, the in stems identical with the core clock	<b>z]</b> Timer. nput clock c.	

 $\bigcirc$  Save the file using  $\blacksquare$  or CTRL+S.

#### 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.

 $\bigcirc$  Right-click on the project and select New  $\rightarrow$  Files from CMSIS Template.

🖨 New Fil	les from CMSIS Templa	_		Х		
CMSIS Use	MSIS User Code Template					
This wizar	d creates new files fron					
Project:	Blinky			Brow	wse	
Compone	ent	Name				
🗸 🚸 CN	VISIS					
	RTOS.Keil RTX	CMSIS-RTOS 'main' function				
- Å	RTOS.Keil RTX	CMSIS-RTOS Mail Queue				
	RTOS.Keil RTX	CMSIS-RTOS Memory Pool				
	RTOS.Keil RTX	CMSIS-RTOS Message Queue				
	RTOS.Keil RTX	CMSIS-RTOS Mutex				
-	RTOS.Keil RTX	CMSIS-RTOS Semaphore				
	RTOS.Keil RTX	CMSIS-RTOS Thread				
	RTOS.Keil RTX	CMSIS-RTOS Timer				
	RTOS.Keil RTX	CMSIS-RTOS User SVC				
Location:	/Blinky			Brov	vse	
File name:	osObjects.h main.c					
?		Fini	sh	Cance	I	

Expand the software component **CMSIS** and select the template **CMSIS**-**RTOS 'main' function**. Click **Finish**. Replace the content of *main.c* with the following application specific code:

```
/*-----
* 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 user.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");
  }
3
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
  }
}
```

Save the file using a or CTRL+S

### Adapt the scatter file

On the i.MX 7 devices, several types of memory are available. For deterministic, real-time behavior, the Cortex-M4 should use the 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 of the i.MX 7, open the file *MCIMX7D\_Cortex-M4.sct* and change the address of the load region to 0x1FFF8000:

 $\bigcirc$  Save the file using  $\blacksquare$  or CTRL+S.

#### **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 41 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

Grow the Eclipse menu bar, choose File → New → C Project. Select the Hello World ANSI C Project:

C Project	- 1	
C Project Create C project of selected type		
Project name: Hello_World Use gefault location Location: C:\Users\stecad01\Documents\DS-M Choose file system: default ~	IDK Workspace_new\Hello_Worl Bj	rowse
Project type: Executable Empty Project Hello World ANSI C Project CMSIS C/C++ Project Shared Library Static Library Makefile project Show project types and toolchains only if the	Toolchains: ARM Compiler 5 ARM Compiler 6 GCC 4 [arm-linux-gnueabihf] gev are supported on the platform	
⑦ < <u>B</u> ack 1	<u>N</u> ext > <u>F</u> inish	Cancel

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:

C/C++ - Hello_World/src/Hello_World.c - Ec	lipse Platform	_		×
<u>F</u> ile <u>E</u> dit <u>S</u> ource Refac <u>t</u> or <u>N</u> avigate Se <u>a</u> r	ch <u>P</u> roject <u>R</u> un <u>W</u> indow <u>H</u> elp			
📑 🕶 🔚 🕼 🗁   🛞 🕶 🗞 🖛 🕅 🕅 🔊	🕽 🔂 र 🛍 र 🖻 र 🞯 र 🎋 र 💽 र 💁 🖋 र 🌛	1		
包 - 径 - や - → -	Quick Access		🗄 🖗	; 🆄
🎦 Project Explorer 👷 📄 🔄 🔽 🗖	🗈 Hello_World.c 💥 🗖 🗖	0 % ,	* <mark>1</mark> <sup></sup>	
> 🚰 Blinky	3⊕ Name : Hello_World.c ∧	⊟ ↓ª₂	💘 🐋	•
> 🗠 Hello_World > 😂 RPMSG PingPong BM	<pre>10 11 #include <stdio.h> 12 #include <stdib.h> 13 14 int main(void) { 15    puts("!!!Hello World!!!"); /* prints !! 16    return EXIT_SUCCESS; 17  } 18 </stdib.h></stdio.h></pre>	⊽ 11 •	stdio.h stdlib.h main(vo	oid) : ir
	< >>	<		>
	📮 Console 🛱 🧔 Tasks   Problems 🔲 Properties			
	↓         ☆         ★         ↓	₿.   ₫	🖳 🔻 🗖	<u>*</u>
				÷.
B Hello_World	1			-

### Build the application image

**T** 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 45 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:



DS-5 Debugger



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

### **Debug Cortex-M application**

This section explains how to debug the microcontroller application running on the Cortex-M microcontroller. Once configured the debug configuration as shown in section *Configure CMSIS DS-5 debugger* at page 20, you can start the debugging session by clicking "Run" in the Debug Control view.

If specified in the configuration window, the debugger will run till the beginning of the function *main()*.

DS-MDK should automatically switch to the **Debug Perspective**, specifically designed to be used during the debug session on your device.

Let's look at some of the Views available in DS-MDK.

#### Variables

The Variables view shows the contents of local, file static, and global variables in your program. By default, the Variables view displays all the local variables. It also displays the file static and global variable folder nodes.

🝽= Variables 💥 🚥 Registers 💥 Expressions f() Functions 🔲 OS Data 🛛 🖓 🗖						
0× 🕱 🔗 🕫 🤝						🔗 🤣 🗢
	🔄 Li	nked: Blinky_M	4 -			
Name	Value	Туре	Count	Size	Location	Access
🖙 🗁 Locals	3 variables					
– P a 11		int		32	\$R2	R/W
– 🧬 b	int		32	\$R1	R/W	
_ <b>○</b> c	0	int		32	\$R0	R/W
🖶 🗁 File Static Variables	0 of 6 variables					
🖻 🗁 Globals	1 of 47 variables					
🗄 🦠 tid_threadA	osThreadId	1	32	0x20000018	R/W	
Add Variable Browse						

If you know the name of the specific variable you want to view, enter the variable name in the Add Variable field. This lists the variables that match the text you entered. Double-click the variable to add it to the Variables view.

### Registers

The **Registers** view displays the contents of processor and peripheral registers available on your target and allows modifying them.

🗵 Variables 📴 Register	s 🔀 🎇 Expressions 🛛 Fu	unctions	-	
		3 🔗	0x 🦑	$\bigtriangledown$
	🔄 Linked: Blinky_M4 👻			
Register Set: All registers				Ŧ
Name	Value	Size	Access	
B > Core	23 of 23 registers			
- • R0	0×00000001	32	R/W	
- • R1	0x303D0200	32	R/W	
- o R2	0x00000000	32	R/W	1
- 🛛 R3	0x00000000	32	R/W	L
- 🛛 R4	0x00000000	32	R/W	
– 🛛 R5	0x00000000	32	R/W	
- 🔿 R6	0x00000000	32	R/W	
– o R7	0x00000000	32	R/W	
- 🛛 R8	0x00000000	32	R/W	
- 🛛 R9	0x00000000	32	R/W	
-      R10	0x00000000	32	R/W	
- o R11	0x00000000	32	R/W	
-      R12	0x00000000	32	R/W	
- • SP	0x200010B8	32	R/W	
-  SP_MAIN	0x20001E48	32	R/W	
—	0x200010B8	32	R/W	
- o LR	0x1FFF94AD	32	R/W	
-  PC	0x1FFF94AC	32	R/W	
🕀 📎 xPSR	0x01000000	32	R/W	
🕀 🥎 PRIMASK	0x00000000	32	R/W	
🕀 📎 BASEPRI	0x00000000	32	R/W	
E S FAULTMASK	0×00000000	32	R/W	Ŧ

The search button at the top of the View allows searching for register by name to speed up debugging in targets with hundreds or thousands of different registers.

### Disassembly

The **Disassembly** view gives you a glimpse over the assembly code running on the device. When the target is stopped, DS-MDK automatically highlights the next instruction to be executed (content of the Program Counter).

The view shows the address, the OpCode and the decoded version of each instruction and can be used, as an example, to debug issues related to invalid addresses.

191	Disassembly 2	🛛 🗐 Mem	nory 🔳 Sta	ck 🔹 Trace 🔚 Events 🔚 Outline 👘 🔗 🗢 🗖	
			•	🔁 Linked: Blinky_M4 👻	
B	🖑 🔻 <next< th=""><th>Instruction</th><th>&gt;</th><th>100</th><th>⇒</th></next<>	Instruction	>	100	⇒
	Address	Opcode	Disassem	bly	
	0x1FFF949C	F001F85A	BL	osDelay ; 0x1FFFA554	
	0x1FFF94A0	A00F	ADR	r0,{pc}+0x40 ; 0x1fff94e0	
	0x1FFF94A2	F000F847	BL	2printf ; 0x1FFF9534	
	0x1FFF94A6	E7F7	В	threadA+8 ; 0x1FFF9498	
			main		
	0x1FFF94A8	F7FFF868	BL	BOARD_RdcInit ; 0x1FFF857C	-
¢e	0x1FFF94AC		BL	BOARD_ClockInit ; 0x1FFF8440	=
	0x1FFF94B0	F7FFFD40	BL	SystemCoreClockUpdate ; 0x1FFF8F34	
	0x1FFF94B4	F7FFFAD0	BL	InitRetargetIOUSART ; 0x1FFF8A58	
	0x1FFF94B8	2100	MOVS	r1,#0	
	0x1FFF94BA	4811	LDR	r0,[pc,#68] ; [0x1FFF9500] = 0x1FFFB788	
	0x1FFF94BC	F001F8AA	BL	osThreadCreate ; 0x1FFFA614	
	0x1FFF94C0	4910	LDR	r1,[pc,#64] ; [0x1FFF9504] = 0x20000018	
	0x1FFF94C2	6008	STR	r0,[r1,#0]	
	0x1FFF94C4	F001F852	BL	osKernelInitialize ; 0x1FFFA56C	
	0x1FFF94C8	F001F862	BL	osKernelStart ; 0x1FFFA590	-

#### **Memory view**

In order to display and to modify the contents of memory it's possible to use the Memory view. You can specify the start address of the memory range, either as an absolute address or as an expression, for example \$pc+256. The size of the memory range to display, in bytes, is the offset value from the start address.

111 Disi	assembly 🗐 N	Aemory ⊠ ≡	Stack 🐇 Trace	Events	<ul> <li>Outline</li> </ul>	
					🔟 🔻 🊕 ·	• x <sub>n</sub> • 🔗 🗢
			🔄 🔄 Linked: Bl	inky_M4 <del>•</del>		
J -	&("main.c"::tid	l_threadA)		200		4
0x2000	000 <mark>xx</mark> 000		Data (Hexa	decimal: 4 b	ytes)	
18	0x00000000	0x00000101	0x00000000	0x20000098	0x20000098	0x00000000
30	0xFFFFFFFF	0x00000000	0x00000000	0x00050000	0x00000000	0x2000156C
48	0x200015C0	0x20001614	0x20001468	0x00000000	0x200000CC	0x200001D0
60	0x00000034	0x01060800	0x00000000	0x20001168	0x00000000	0×00000000
78	0x00000000	0x00000000	0x00000000	0x00000000	0x00C80006	0x20000E70
90	0x20000DE8	0x1FFFA689	0x02040200	0x00000000	0x00000000	0x00000000
A8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x02000004
cø	0x20001078	0x20000EB8	0x1FFF94C7	0x20000100	0x00000000	0×00000000
D8	0x00000000	0x00000000				

The memory view allows specifying both the address and the size as a formula. A few examples:

- &("main.c"::tid\_threadA) refers to the address of variable tid\_threadA in file main.c
- \$PC refers to the value contained by the register PC
- sizeof(float) refers to the size of the type "float"

Please refer to the online manual for further options.

#### **Breakpoints**

Breakpoints can be set either directly on the source code editor or in the **Breakpoints** view. In the source code editor, right-click on the left side on the line you would like the execution to stop and click on **Toggle Breakpoint**.



The breakpoint will appear in the list in the **Breakpoints** view where it can edited, disabled or removed.



It is possible to access to the properties of the breakpoint by right-click on the breakpoint and then select **Properties**. The Properties window, showed below, allows using some of the advanced functionalities of the DS-MDK debugger such as Thread specific breakpoint, advanced Stop conditions and the ability to run scripts when the program stops.

e Breakpoint Properties					
Breakpoint Propertie	s				
Description: Host File Location: Compiled File Location: Type: State: Address:	main.c53 @ main+0x4 (T32) [#2 SW] C:\Users/stecad01\Documents\DS-MDK Clean\Blinky_M4\main.c53.0 C:\Users/stecad01\Documents/DS-MDK Clean\Blinky_M4/main.c53.0 Source Level Software Breakpoint Active @ • 0x1FFF94AC main+0x4 (#21] (T32)				
Break on Selected Th	reads or Cores				
Show Both Active	and Inactive Threads				
Cortex-M4 #1 stop main #3 stopped ( osTimerThread #2 os_idle_demon #4	Cortex-M4 #1 stopped on breakpoint Cortex-M4 #1 stopped (PID 2) osTimerThread #2 stopped (PID 1) os_idle_demon #4 stopped (PID 25)				
Only core selec	tion is persistent between connections				
Stop Condition:		•			
Ignore Count:	0				
On break, run script:					
	✓ File Syst	Workspace			
Continue Execution					
Hardware Virtualization: Break on Virtual Machin	Unsupported = ID:				
?	OK	Cancel			

Please refer to the online help for a detailed explanation of all the functionalities accessible from the **Properties** window.

### **Debug Linux application**

This section explains how to debug a Linux application running on the Cortex-A7.

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 13.
- 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, set up a Remote Systems Explorer (RSE) connection to the target to download the application onto the target's file system. Refer to **Setup RSE connection** on page 23 for more information.

Configure the debugger as described in *Configure DS-5 debugger* at page 24 and launch the application.

DS-MDK uses the same debug perspective as for bare metal when debugging Linux application so you do not need to learn a new environment or set of Views in order to start debugging.

### **Debug the Linux Kernel**

The DS-5 Debugger configuration dialog makes it easy to configure a debugging session to a specific target. The Linux kernel debug configuration type is primarily designed for post-MMU debug to provide full kernel awareness but – with some extra controls – can also be used for pre-MMU debug. This makes it possible to debug the Linux kernel, all the way from its entry point, through the pre-MMU stages, and then seamlessly through the MMU enable stage to post-MMU debug with full kernel awareness. You can do this all with source-level symbols, and without the need for tedious disconnecting, reconfiguring and reconnecting!

The Linux kernel, already built with debug info and a complete *vmlinux* symbol file, file system, and full source code, is available from the respective board support pages (see <a href="http://www.keil.com/mdk5/ds-mdk/install#boards">www.keil.com/mdk5/ds-mdk/install#boards</a>).

Unpack the Linux kernel sources (*kernel-source.tar.gz*) into your currently active DS-MDK Eclipse workspace. Be aware that on a Windows system you will not be able to fully unpack the sources. Some symbolic links and case-sensitive source files will not be created. Usually, this is not critical for Linux kernel debug.

### Create a Linux Kernel debug project

Create a new CMSIS C/C++ Project named Linux Kernel Debug and select NXP i.MX7Dual device *MCIMX7D7:Cortex-A7*.



Add the *vmlinux* file to the project folder using Windows Explorer. This file must match the kernel in the SD card on the board.

#### NOTE

The debug symbols in the vmlinux file have virtual addresses, so the usage of vmlinux file by the debugger assumes that the OS is up and running with the MMU enabled. It still can be used to debug pre MMU at source-level if there is no offset between physical and virtual addresses at the entry point.

G Add a debugger script to the project (right-click the project and select New → Other... → DS-5 Debugger → DS 5 Debugger Script) called stop.ds containing:

```
stop
set os enabled off
```

When OS awareness is enabled and kernel symbols are loaded from the vmlinux file, DS-5 Debugger will try to read some kernel structures. If the MMU is not yet on, the debugger may try to access invalid addresses, leading to data aborts, which is undesirable. This OS awareness support feature can be temporarily

disabled during the pre-MMU debug stage with the CLI command

set os enabled off, and later (post-MMU) re-enabled with the CLI command

set os enabled on.

C Restart the board and make sure you stop the boot of the Linux kernel by pressing a button when U-Boot is initializing in the *Terminal* view.



Right-click on the project, select Debug As..., then select CMSIS DS-5 Debugger... to open the Debug Configurations dialog.

In the *Connection* tab, select CPU Instance = SMP.

In the *Advanced* tab, tick **Run target initialization debugger script**, and select the **stop.ds** script in the workspace:

Debug Configurations	×
Create, manage, and run configurations (Connection): Connection address is empty	to the second se
Image: Second Secon	Name:       Linux Kernel Debug         Connection       Advanced         File Settings         Program image       \${workspace_loc:/Linux Kernel Debug/vmlinux}         File System       Workspace         Load symbols only         Run control         @ Connect only       Debug from entry point         Debug from symbol       main         Run target initialization debugger script (.ds / .py)       File System         \${workspace_loc:/Linux Kernel Debug/stop.ds}       File System         Run debug initialization debugger script (.ds / .py)       File System         Workspace       File System
Filter matched 23 of 23 items	Apply Revert
?	Debug

Click Debug. The *Commands* view will show:



In the Command (CLI) entry box, set a temporary hardware breakpoint (thbreak) on the entry point into the kernel, by typing in:

thbreak 0x80008000

Press the Submit button or the Enter key. 0x80008000 is the entry point for the kernel. This is the address to which U-Boot will pass control to boot Linux once it has completed its setup tasks.

- If Run the target by pressing the Continue button (▶) in the Debug Control view, or press F8.
- 🗇 In the Terminal view, tell U-Boot to boot the kernel, by typing in:

boot

Code execution will stop at the breakpoint, and the Disassembly view will show the assembly code at the entry point (labeled stext). If you have unpacked your kernel source code into the workspace, the Editor view will show the content of head.S.

If not, no source code is shown, because the path to the source code has not yet been configured. DS-5 Debugger will try to open .../arch/arm/kernel/head.s in its Editor view. If it does not find the kernel sources using the source paths within the vmlinux file, you can resolve this by setting a substitute source path, to re-direct paths from where the kernel was built, for example, from:

```
/home/munlin01/fsl-community-bsp-platform/build-core-image-
base/tmp/work-shared/imx7dsabresd/kernel-source
```

to a local copy of the kernel sources at:

C:\path\to\linux-imx\4.1.15-r0\git

Make sure that the "Image Path" and "Host Path" both end with a corresponding directory.

head.s will now open in the *Editor* view, and the *Disassembly* view will show the symbol stext, at the entry point for the kernel. If it doesn't, choose the **Path Substitution...** command from the *Debug Control* view's drop-down menu ( $\bigtriangledown$ ) and check that the final directory in the Image Path and Host Path correspond. Then right-click on an instruction in the *Disassembly* view, and select "Show in Source".



### Debug the Kernel: Pre-MMU stage

You can now set breakpoints and watchpoints, view registers, view memory, single-step, and other usual debug operations at this pre-MMU stage, all with source level symbols.

- At the kernel entry point, you can check the Core and CP15 system registers in the Registers view to check that they are set as recommended by kernel.org. Observe that:
- a. the CPU is in SVC (supervisor) mode; check Core → CPSR → M → SVC
- b. **R0** is 0
- c. R2 contains a pointer to the device tree. Right-click R2 and select Show Memory Pointed To By R2. Change the size of the memory displayed to 200 bytes for example by entering 200 in the text entry box in the top right of the *Memory* view.
- d. the MMU is off; check CP15  $\rightarrow$ SecureBanked  $\rightarrow$  S\_SCTLR  $\rightarrow$  M
- e. the Data cache is off; check CP15
   → SecureBanked → S\_SCTLR → C

«)= Variables 🛛 🔒 E	reakp 🚾 Registers 🔀	×+y ⁼? E	opressi	f() Functi	-	
				📄 🔗 Ox	ŝ	~
	🔄 Linked: Linux	Kernel I	Debug 🔻			
Register Set: All re	gisters					
Name	Value	Size	Access			
🗕 🗁 Core	50 of 50 registers					
– 👄 R0	0x00000000	32	R/W			
– 🖌 R1	0x00000000	32	R/W			
– 🖌 R2	0x83000000	32	R/W			
– 🖌 R3	0×0000000C	32	R/W			
– 👄 R4	0×80008000	32	R/W			
– 🛛 R5	0x00000000	32	R/W			
– 👄 R6	0x8131F828	32	R/W			
– 🛛 R7	0x00000000	32	R/W			
– 🧉 R8	0x83000000	32	R/W			
– 🧉 R9	0x410FC075	32	R/W			
– 👄 R10	0×00000000	32	R/W			
– 👄 R11	0x8131F820	32	R/W			
– 👄 R12	0x80CA8A54	32	R/W			
— 👄 SP	0x81320848	32	R/W			
– 🧉 LR	0x80CA8F10	32	R/W			
- 🖌 PC	0×80008000	32	R/W			
🖨 🗞 CPSR	0x800000D3	32	R/W			
- 🖌 N	0x1	1	R/W			
– 🖌 Z	0x0	1	R/W			
- 🖌 C	0x0	1	R/W			
– 🖌 V	0x0	1	R/W			
– 🧉 Q	0×0	1	R/W			
- 🛛 IT	0x00	8	R/W			
- 🛛 J	0×0	1	R/W			
– 🖌 GE	0x0	4	R/W			
- 🖌 E	0×0	1	R/W			
– 🖌 A	0x0	1	R/W			
- 🖌 I	0x1	1	R/W			
- 🖌 F	0x1	1	R/W			
- 🛛 T	0x0	1	R/W			
L⊚ M	svc 🔻	5	R/W			
🕀 🗁 IRQ	3 of 3 registers					l
🕀 🗁 FIQ	8 of 8 registers					
🕀 🗁 UND	3 of 3 registers					
🕀 🗁 ABT	3 of 3 registers					
🕀 🗁 SVC	3 of 3 registers					
🖶 🗁 USR	7 of 7 registers					

- f. the Instruction cache is either on or off; check CP15  $\rightarrow$  SecureBanked  $\rightarrow$  S\_SCTLR  $\rightarrow$  I
- To see when the MMU will be turned on, set a breakpoint:

thbreak \_\_turn\_mmu\_on

then continue running (or press F8). When <u>turn\_mmu\_on</u> is reached, note the value of SP. This contains the virtual address of <u>mmap\_switched</u> and is the place the code will jump to after the MMU is enabled.

In general, it is not possible to single-step through \_\_turn\_mmu\_on, so place a hardware breakpoint on the virtual address of \_\_mmap\_switched:

#### thbreak \*\$SP

then continue running (press F8). When the breakpoint at mmap switched is hit, the MMU is on.

- $\bigcirc$  Check that the MMU is now on, by looking in the Registers view at CP15  $\rightarrow$ **SecureBanked**  $\rightarrow$  **S SCTLR**  $\rightarrow$  **M** (should show Enable).

### Debug the Kernel: post-MMU stage

The main C code entry into the kernel, after all the architecture-specific setup has been done, is start kernel() in \source\init\main.c.

Gr Set a breakpoint on it:

```
thbreak start kernel
```

and then run to it.

To You can now safely enable OS support in DS-5 Debugger:

set os enabled on

C Check that the following appears in the Command view, to confirm Linux kernel support is enabled:

```
Enabled Linux kernel support for version "Linux 4.1.15-
1.1.0+ga4d2a08 #2 SMP PREEMPT Tue Jul 5 09:51:28 CEST 2016
arm"
```

The same Linux version information can be reported manually using:

info os-version

which will show for example:

Operating system on: Linux 4.1.15-1.1.0+ga4d2a08 #2 SMP PREEMPT Tue Jul 5 09:51:28 CEST 2016 arm

This is similar to:

output init nsproxy.uts ns->name

which will show for example:

```
{sysname = "Linux", nodename = "(none)", release = "4.1.15-
1.1.0+ga4d2a08", version = "#2 SMP PREEMPT Tue Jul 5 09:51:28
CEST 2016", machine = "armv71", domainname = "(none)"}
```

This may take a few moments to display, because DS-5 Debugger has to process the debug symbols.

When OS awareness is enabled and kernel symbols are loaded from the vmlinux file, DS-5 Debugger will try to access some locations in the kernel. For example, it will try to read init nsproxy.uts ns->name to get the kernel name and

version. It will also set breakpoints automatically on SyS\_init\_module() and SyS\_delete\_module() to trap when kernel modules are inserted (insmod) and removed (rmmod). You will see these breakpoints appearing in the *Breakpoints* view:



🕝 Set a breakpoint with:

thbreak kernel\_init

then run to it.

So far, CPU 0 has been doing all the work. Note that CPU 1 is still powered down:

```
Linux Kernel Debug connected VM:0
Cortex-A7_0 #0 stopped on breakpoint
Cortex-A7_1 #1 powered down
```

A very useful feature during kernel bring-up is to display early printk output in DS-5 Debugger's command window.

G Before the console has been enabled there will be no output from the serial port. You can view the entire log so far with:

info os-log



To view the log output line by line, as it happens, use:

#### set os log-capture on

kernel\_init() tries to start the init process. To see this, set a breakpoint at the end of kernel\_init() then run to it (set the breakpoint in the main.c file available in the *Editor* view). The init process now appears as an active thread. CPU 1 is now powered up.

Many of the above steps can be automated, either with a script file, or by fillingin the Debug Configuration's fields before launching (refer to the Appendix).

- Delete all user breakpoints and continue (F8). Let the kernel run all the way to the Login prompt. Login as root.
- Stop the target by pressing Interrupt (11/F9). In the *Debug Control* view, expand "Active Threads" and "All Threads". In "All Threads", you will see a large number of threads/processes have been created. Only two were actually running, one on each of the two cores. You can see these in "Active Threads".

Right-click on the connection and select Display Cores to see the state of both CPUs. You can view the state of the cores, threads and processes on the command-line with:

info cores info threads info processes

It is possible to single-step a core or a thread/process. To do so, select either the core or the thread/process in the *Debug Control* view, then press Step (→-/F5). Note that when single-stepping though a process, it might get

🖷 Memory 🚍 Stack 🖷 MMU 🛛 🞸 Trace 🔚 Events 🚼 Outline 🔄 Linked: Linux Kernel Debug 🔻 Translation Tables Memory Map Virtual Range Physical Range Type 
 S:0x00010000-0x00010FFF
 <unmapped>

 S:0x00010000-0x00010FFF
 SP:0xBF4E4000-0xBF4E4FFF

 S:0x00011000-0x00011FFF
 SP:0xBF4E3000-0xBF4E4FFF
 Normal RO Normal RO S:0x00012000-0x00012FFF SP:0xBF4E2000-0xBF4E2FFF Normal RO S:0x00013000-0x00013FFF SP:0xBF4E1000-0xBF4E1FFF Normal RO S:0x00014000-0x00014FFF SP:0xBF4E0000-0xBF4E0FFF Normal RO S:0x00015000-0x00015FFF SP:0xBF1C8000-0xBF1C8FFF RO Normal S:0x00016000-0x00016FFF SP:0xBF49B000-0xBF49BFFF Normal RO Current: Secure PL1&0

migrated to another core. If a breakpoint is set on a process, the debugger is able to track the migration of process-specific breakpoints to the other core.

- ✓ You can check the virtual-to-physical address map for Linux by using the MMU view. Continue to run the target (F8). Go to Window → Show View → MMU. Switch to the Memory Map tab and press the Show Memory Map button to refresh the values.
- Let's take a look at the kernel's thread\_info structure. Stop the target, then check the kernel's stack size with:

```
show os kernel-stack-size
```

For this ARMv7 kernel, the kernel stack size is 8K.

In the Expressions view, add a new expression into the field (type in the field at the bottom on the view):

(struct thread\_info\*) (\$sp\_svc & ~0x1FFF)

0x1FFF is 8K minus 1. Expand the tree structure to explore its contents. The list of threads in the *Debug Control* view is created from the same information, so they should match. For example, the thread name is held in task.comm.

To get a simple view into the workings of the scheduler, set a breakpoint on \_\_schedule() with:

hbreak \_\_schedule

#### NOTE

This time use hbreak to have a persistent hardware breakpoint instead of a temporary one.

Then continue running (press **F8**). At the breakpoint, continue running (press **F8**) again and again, and see the names of the active threads changing in "Active Threads", and different threads are scheduled-in.

Alternatively, instead of setting a breakpoint on \_\_schedule(), try to set a breakpoint on do\_fork(). If nothing forks, force a fork by typing e.g. 'ls'.

In summary, we have looked at how DS-MDK can be used to debug the Linux SMP kernel, both in pre-MMU enabled and post-MMU enabled stages, and looked at a few of the kernel's internal features.

### Debug a Linux Kernel module

Only a few things are required to make kernel module debugging work. This sections explains how to do this for the *imx\_rpmsg\_tty* module that is used in the example projects that are explained in detail on page **Error! Bookmark not defined.** 

### Create a Linux Kernel module debug project

Create a new CMSIS C/C++ Project named Linux Kernel Module Debug

As with the Linux kernel debug, add the *vmlinux* file to the project folder using Windows Explorer.

Add a debugger script to the project (right-click the project and select New → Other... → DS-5 Debugger → DS 5 Debugger Script) called stop.ds containing:

stop

G Add another debugger script to the project (right-click the project and select New → Other... → DS-5 Debugger → DS 5 Debugger Script) called load\_ko.ds containing:

add-symbol-file imx\_rpmsg\_tty.ko

#### **NOTE**

Make sure that the file imx\_rpmsg\_tty.ko is stored in the workspace so that DS-MDK can find it. Otherwise, specify the fully qualified path to it. You can download the file and the source code file from the board support page of your development board.

The stop command in the first script will halt the processor before loading the kernel symbols and the add-symbol-file command will load the kernel module object file.

 $\bigcirc$  Right-click the project and select **Debug As**  $\rightarrow$  **CMSIS DS-5 Debugger...** 

On the *Connections* tab, set the CPU Instance to either 0 or SMP. Go to the *Advanced* tab and specify the path to the *vmlinux* file and enable **Load symbols only**. Also, set the initialization debugger scripts as shown here:

Debug Configurations	×
reate, manage, and run configurations aunch a DS-5 debugging session using a CMSIS DS-5 Debugger project.	ect.
Image: Second	OS Awareness  R\Documents\DS-MDK Workspace\vmlinux File System Workspace  gger script (.ds / .py) g/top.ds) File System Workspace  ggr oad_ko.ds File System Workspace Apply Revert
2 Debug Close	Debug Close

Apply the settings and press Close (do not press Debug yet!).

### **Debug the Kernel module**

The following steps are required to come to a point where you can debug the kernel module:

**C** Restart your target and halt in U-Boot.

Debug and run the Cortex-M4 application RPMSG TTY RTX.

Boot Linux.

At the Linux prompt, issue the following command to install the driver for the kernel module: modprobe imx rpmsg tty

Debug and run the Kernel\_Debug project.

Now, you can open the *imx\_rpmsg\_tty.c* and set breakpoints.

Finally, debug the Linux Application TTY as well (make sure that the RSE connection is still live). When you run the application, the debugger will stop at the breakpoint you have set in the previous step.



### **ARM Streamline**

ARM Streamline performance analyzer gives you the ability to collect performance metrics, software tracing and statistical profiling from your Linux system and show that in its innovative user interface. Streamline helps you to identify code hotspots, system bottlenecks and other unintended effects of your code or the system architecture.

DS-MDK includes ARM Streamline in the MDK Professional edition: you can launch Streamline from the ARM DS-MDK Start menu.



Once launched, Streamline allows connecting via TCP/IP to a running Linux target. A target agent (gator) is required to run on the ARM Linux target for ARM Streamline to operate. If you downloaded the Linux image from <a href="http://www2.keil.com/mdk5/ds-mdk/install#boards">http://www2.keil.com/mdk5/ds-mdk/install#boards</a>, then gator is already installed so you do not need to rebuild the image.

To start collecting data, you can type the target hostname or IP address in the field box on the top-left side of the window and press the Start Capture button.



The interface would then show the acquired data in graphs which can be used to understand which parts of the code require optimizations or affect the performance of the system considerably.

Capture_C01_A01	and the second			
🔙 Timeline 🕖 Call Paths 🔞 Fun	nctions 🗟 Code < Call Graph 🗄 Stack 🤇	¥ Log		
		200ms		1:53,795 11
45 65 85 105 CPU Activity User System GPU Vertex	123         145         145         185         205           ■         1000         ■	22s 24s 26s 49.44 117 117 117 117 117	28s 30s 32s % avg. % avg.	345
GPU Fragment			% arg.	
Effective Shader Cycles	500 gde	● 67.30 ● 7.12	loyde	
Geometry Statistics			120	~~~^
GIDrawElements Statistics				
Mali GPU Vertex Processor		711,2	18,834	

For extra information on the capabilities of the product, please refer to the user guide available online at <u>https://developer.arm.com/docs/100769/latest/</u>.

### **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)

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

fromelfbin	output "Blinky.bin" "Blinky.axf"	
Properties for Blinky		— 🗆 X
type filter text	Settings	← → ⇒ → →
<ul> <li>Resource</li> <li>Builders</li> <li>C/C++ Build</li> <li>Build Variables</li> <li>Environment</li> </ul>	Configuration: Debug [Active]	A Manage Configurations
Longjing Settings Tool Chain Editor > C/C++ General Project References Run/Debug Settings	Tool Settings       Image: Build Steps       Build Artifact       Image: Binary Parsers       Image: Binary Parsers <t< td=""><td>vr Parsers</td></t<>	vr Parsers
?		OK Cancel

#### NOTE

This example shows the steps for the Blinky application from section **Blinky** with CMSIS-RTOS RTX on page 28.

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.
- **C** List the partitions with the fdisk command:

```
~# fdisk -1
...
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:
  - Create a sub-directory on the Linux file system, for example:
     <u>"# mkdir /media/sd0</u>

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.

**C** 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 (<u>www.denx.de/wiki/DULG/UBoot</u>).

## Appendix

### **Remote Processor Messaging protocol example**

The device family packs for NXP's i.MX devices contain 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-A processor and writes a message to a TTY device. The terminal of the *RPMSG TTY RTX* application running on the Cortex-M processor shows this message. The application itself responds on the TTY device. The Linux application reads this message and shows it in its **App Console**.

### **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 have 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  $\rightarrow$  Help Contents).

### **ELF** file viewer

An ELF 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.

RPMSG_TTY_RTX_M4.axf 🙁				
Header		See.		
Machine class	ELFCLASS32 (32-bit)			
Data encoding	ELFDATA2LSB (Little endian)			
Header version	EV_CURRENT (Current version)			
Operating System ABI	none			
ABI version	0			
File type	ET_EXEC (Executable file) (2)			
Machine	EM_ARM (Advanced RISC Machines ARM)			
Image entry point	0x1FFFB299			
Flags	<pre>EF_ARM_HASENTRY + EF_ARM_ABI_FLOAT_SOFT (0x05000202)</pre>			
Header Size	52 bytes (0x34)			
Segment header entry size	32 bytes (0x20)			
Section header entry size	40 bytes (0x28)			
Program header entries	1			
Section header entries	16			
Program header offset	5497220			
Section header offset	5497252			
Section header string table index	15			
Header Sections Segments Symbol Table I	Disassembly			

### **CMSIS Configuration Wizard**

Right-click on a file in the Project Explorer and select **Open With**  $\rightarrow$  **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.

I≣ RTX_Conf_CM.c 😢			
😑 CMSIS Configuration Wizard		÷ =	?
Option	Value		^
<ul> <li>Thread Configuration</li> </ul>			
Number of concurrent running user threads	6		
Default Thread stack size [bytes]	1024		
Main Thread stack size [bytes]	1024		
Number of threads with user-provided stack size	0		
Total stack size [bytes] for threads with user-provided stack size	0		
Stack overflow checking			
Stack usage watermark			
Processor mode for thread execution	Privileged mode		
<ul> <li>RTX Kernel Timer Tick Configuration</li> </ul>			
Use Cortex-M SysTick timer as RTX Kernel Timer	$\checkmark$		
RTOS Kernel Timer input clock frequency [Hz]	24000000		
RTX Timer tick interval value [us]	1000		
<ul> <li>System Configuration</li> </ul>			
<ul> <li>Round-Robin Thread switching</li> </ul>	$\checkmark$		
Round-Robin Timeout (ticks)	5		~
<		>	
Number of concurrent running user threads Defines max. number of user threads that will run at the same time. Default: 6			
Source Editor CMSIS Configuration Wizard			

### **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 **Save the file using** or CTRL+S

Adapt the scatter file on page 35).

MCIMX7D_C	ortex-M4.sct 🔀			
0xFFFFFFFF		0xFFFFFFF		
	LR_IROM1		RW_IRAM1	
	LR	ROM1	.ANY (+RW,+ZI)	
		0x2000000		
			ER_IROM1	
			*.o (RESET,+First) * (InRoot\$\$Sections) .ANY (+RO)	
0x1FFF8000		0x1FFF8000		
0x00000000		0x00000000	<u> </u>	
1	Load Regions		Execution Regions	
Regions/Section	ns MCIMX7D_Cortex-M4.sct			,

If you want to learn more about the scatter loading mechanism, look for the documentation at <u>https://developer.arm.com</u>.

### **CMSIS Pack Manager perspective**

The Pack Manager perspective offers the following functionality:

- Install or update software packs.
- List devices and boards supported by software packs.
- List example projects from software packs.

Use the E<sup>2</sup> icon and select **CMSIS Pack Manager**, to open this perspective.

Device Da	tabase	Availat	ole Pac	ks/Examples	Pack Properties
CMSIS Pack Manager - R <sup>P</sup> MS File Edit Navigate Search	G_TTY_RTX_M4/RPMSG_TTY_ Project Run Window He	RTX_M4.rteconfig - Eclipse Platform			- c ×
	- 9 🔗 🕅 🔍 + 🔗 +	2a - 2a - 4= 4 <b>-</b>			Quick Access
Devices 23 Bu Boards		Packs Examples 23	ly show examin	ies from installed packs   🕜   🤤 🥪 💞 👘 🗆	I Pack Properties 23
Search Device		Search Example			E C 🔍 🗸
Device	Summary ^	Example	Action	Description	Keil.iMX7D_DFP.0.1.10
V INXP	527 Devices	CMSIS-RTOS Blinky (MCIMX7D-SABRE)	A Conv	CMSIS-RTOS based Blinky example for Cortex-M4	V 🖪 Boards
v 🕂 i MY 7 Series	1 Device	Linux Application TTV (MCIMX7D-SABRE	Copy	Linux Application TTV example	NXP::MCIMX7D-SABRE
* i MX 7Dual	1 Device	RPMSG PingPong RM (MCIMX7D-SABRE)	Copy	Bare-Metal RPMSG PingPong example for Cortex-M4	V 💠 Components
MCIMY7D	ARM Cortex-07 ARM	RDMSG DingPong RTY (MCIMX7D-SARRE	Copy	CMSIS-RTOS RTX and Bare-Metal RPMSG PingPong evan	V Q Device
K Series	1 Device	RPMSG TTY RTX (MCIMX7D-SABRE)	Copy	CMSIS-RTOS RTX TTV example for Cortex-M4	Startup
× 45 K00 Series	2 Devices	(Weiwick-Shore)	( copy	chois has the completer contex-we	V IMX7D HAL
* K10 Series	23 Devices				VART
K20 Serier	41 Devices				♀ MU
Kan Series	6 Devices				CCM
K40 Series	6 Devices				RDC
× K50 Series	12 Devices				> Poard Support
> 15 K50 Series	12 Devices				> 💡 OpenAMP
KOO Series	4 Devices				V E Devices
S K/O Series	2 Devices				🗸 🍕 I.MX 7 Series
> 15 Kou Series	2 Devices				🗸 🔩 I.MX 7Dual
> Is KEAK Series	21 Devices				MCIMX7D
> 15 KEXt Series	21 Devices				✓ <sup>™</sup> Examples
> 15 KLOC Series	14 Devices				> CMSIS-RTOS Blinky (MCIMX7D-S
> IS KNOX Series	14 Devices				> Linux Application TTY (MCIMX7E
> 15 KSXX Series	2 Devices				> RPMSG PingPong BM (MCIMX7E
> 15 KVxx Series	23 Devices				> RPMSG PingPong RTX (MCIMX7I
> 13 KWxx Series	14 Devices				<ul> <li>RPMSG TTY RTX (MCIMX7D-SAB</li> </ul>
> 1\$ LPC800 Series	10 Devices				NXP::MCIMX7D-SABRE
> 15 LPCT100 Series	128 Devices				
> 1\$ LPC1200 Series	12 Devices				
> TS LPC1300 Series	24 Devices				
> 13 LPC1500 Series	13 Devices				
> T\$ LPC1700 Series	21 Devices				
> TS LPC1800 Series	21 Devices				
<ul> <li>MX LPC4000 Series</li> </ul>	16 Devices	4		>	
	,			>	· · · · · · · · · · · · · · · · · · ·
📮 Console 💥 🖷 Progress				÷ 🗘 🕀	si 🗊 🌆 = 🗟 i 🛃 🖬 🖬 🖬 🖬 👘
CDT Build Console [RPMSG_TTY_	RTX_M4]				
					^
Total RO Size (Code	+ RO Data)	28292 ( 27.63kB)			
Total ROM Size (Code	ta + 21 Uata) + RO Data + RW Data)	31830 ( 31.11KB) 28412 ( 27.75kB)			
.otal Non Size (Code	. no vala + ne vala)	20122 ( 21.1360)			~
<					>
ONLINE					

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

Remote System Explorer & modeSystem Server Rest Vo.11.130/nome/rock/texploxy.sh - Ecipice Platform File Edi Navigate Search Project Run Workow Help Remote System Server Run Workow Help Remote System Details Store Remote Server Run Run Workow Help Remote System Server Run	Remote	Systems Fi	le/S	System Prop	perties	Source	e Code Editor	Remo	te System	Details
File Ed. Naviget Sean Project Ru Widew Help   Rencols System Cells Si Tell An etime is not available. An etime is not available. Shift Si Tell Shift S	Remote System E	plorer - RemoteSystemsTemp	of iles/1	10.41.1.130/home/root/kee	pbusy.sh - Eclipse Platform	1			-	
Image: System Signer System Signer Signe	File Edit Navigate	Search Project Run W	findow	Help						
Remote Systems Signed Field Signed	📑 🗝 🔛 🕤 📥	🗎 🕩 U 🗉 M D. 7	n.	≒ 党 🔯 🎋 🔹 🕥	• 💁 • 🛷 • 🗉 🗉	2 × 9 × + 4	• ⇒ •		Quick Access	Ra 🐐 🌰 🔢
An addine is not available.     An addine is not avail	Remote Systems	23 😪 Team 🗧		🗎 keepbusy.sh 🛛				-	🔋 🗆 🚼 Outline 🛛	- 8
Image: Second Second Parent Properties 28 Remote Second Parent profile       Remote System Details 28 Tasks       Image: Second Parent Profile       Remote System Version Second Parent Parent Profile       Remote System Version Second Parent Parent Parent Profile       Remote System Version Second Parent		All ← Al	<b>Λ</b>	iwhile ;; do ;; 3	done &				An dufine is no	t available.
Properties № € memore Scratchpag       Image: Constraint of the scr	<	mnt	ľ	<						
Property     Value     Part Connections     Parent profile     Connection status     Host name     Description       Canonical Parent profile     Permits your profile     Parent profile     Some subsystems connected     LOCALHOST     USER       Classification     recurable     Permits your profile     Some subsystems connected     LOCALHOST     USER       Filter string     //mom/root/receptory.of     Discription     Discription     Some subsystems connected     LOCALHOST     USER       Filter string     //mom/root/receptory.of     Discription     Discription     Some subsystems connected     LOCALHOST     USER       Hidden     No     Discription     Discription     Discription     Some subsystems connected     LOCALHOST     USER       Name     Iterational //mom/root/receptory.sh     root     Filter string     Permits/root/receptory.sh     Filter string     Permits/root/receptory.sh     Filter string     Permits/root/receptory.sh       Name texeptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh       Name texeptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh     //mom/root/receptory.sh       Permitsions     //mom/root/receptory.sh     //mom/root/rece	Properties 92	Remote Scratchoad		Remote System Deta	ile S? . 🕅 Tasks				<u># 8   4 4 6  </u>	* ~
Property     Value     Resource     Parent profile     Remote system type     Connection status     Hot name     Default User/ID     Description       Classification     //mom/rout/kepbusy.sh     Image: Classification     Image:	E rioperates 23 O	Nemote Scrutenpus		Root Connections						+r U
Property     Value     Productive pression       Cannoid BM     //mom/cott/Repbuysh     Individual Status       Cannoid BM     //mom/cott/Repbuysh     Elocal       Classification     executable       Externion     sh       Filter string     //mom/cott/*       Group     0       Hidden     No       Last modified     11 February 2016 155921       Last modified     11 February 2016 155921       Name e keepbusgh     Keepbusgh       Name of childre     0       Permissions     momentance		📴 🕈 🌆 🗖	Ľ,	Personate	Darrant profile	Pamata putan tuna	Connection status	Hest erms	Default Hear I	Description
Canonical Path //home/nork/teepbury.dh Classification sh Externion sh Filter sting //home/nork/* Group 0 Hidden No Last modified 11 February 2016 15:59:21 Location //home/nork Namee Keepbury.dh Namee Keepbury.dh Namee Keepbury.dh Name	Property	Value	1	Et and	DESKTOR SETOOR	least	Connection status	LOCALLIOST	UCT I	beschption
Classification     executable     C4LMUX_SABRE     UEXIUP-SolUQPL     Solution     Some subsystems connected     IU.A1.1.30     root       Filter string     /homo/root/*     Group     0     Interview     Interview     Interview     Interview     Interview       Hidden     No     Interview     Interview     Interview     Interview     Interview     Interview       Last modified     Interview     Interview     Interview     Interview     Interview       Name er childre     0     Interview     Interview     Interview     Interview       Permisions     manwarex     X     Interview     X     Interview	Canonical Path	/home/root/keepbusy.sh		Local	DESKTOP-SOTOQPL	Local	Some subsystems connected	LUCALHUST	USER	
Edension sh Filter string // home/root/ Group 0 Last modified 11 February 2016 15:59:21 Location //home/root Name Keepburg/h Name of childre 0 Owner 0 Permissions moreowaw v	Classification	executable	- 8	L#I.MX7_SABRE	DESKTOP-SETOQPL	SSH Only	Some subsystems connected	10.41.1.130	root	
Filter string         /homo/root/*           Group         0           Hidden         No           Last modified         11 February 2016 1559:21           Location         /homo/root           Name er dchidre         0           Owner         0           Permissions         revrewsex	Extension	sh								
Group         0           Hidden         No           Last modified         11 February 2016 15:59:21           Laction         /home/root           Name         Keepburych           Name of childre         0           Owner         0           Permissions         rownowx	Filter string	/home/root/*								
Hidden No Last modified II February 2016 1559:21 Location //nom/root Name for Anitre 0 Name of Childre 0 Owner 0 Permissions manwarx	Group	0								
Last modified 11 February 2016 15:59:21 Location //horme/root Name keepbusy.ch Number of childre 0 Owner 0 Permissions rourvourvux v c	Hidden	No								
Location /home/nota Name for keepbusych Number of childre 0 Owner 0 Permissions reveneex × <	Last modified	11 February 2016 15:59:21								
Name     keepbury.sh       Number of childre     0       Owner     0       Permissions     rourwarwax       V     >	Location	/home/root								
Number of childre 0 Owner 0 Permissions NumwerNex C C C C C C C C C C C C C C C C C C C	Name	keepbusy.sh								
Owner O Permissions roorworkve > C	Number of childr	0								
Permissions nonvorvex v	Owner	0								
	Permissions	DAXDAXDAX								
	<	TTATTATTA	>	<						>

For more information, refer to the *RSE User Guide* in the Eclipse help system (Help  $\rightarrow$  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.

Debug Control Source Co	de Editor	Target Connec	tion	Disass	embly	Va	riat 	oles		
DS-5 Debug - Linux Application TTY/src/LinuxTTY.c - Eclipse P	atform							- c	2	×
File Edit Source Refactor Navigate Search Project Ru	Window Help									
🖆 🕶 🗟 🗞 🛔 🐘 🐥 🕶 🔝 💁 🕶 🏈 🕶 🍠	8 = 8 = % <b>0</b> = 0	⇒ <b>-</b>			[	Quick Access		s   Ec	۵	
🏘 Debug Con 🙁 🏠 Project Exp 📕 Remote Sy 📟 🛛	🔲 🖬 Commands 🖾	🖬 History 🚿 Scripts 🛛 🗟 👪 🚮	🎜 <b>• </b> 🖶 🗖	00- Variab 😒	💩 Break 💶 Regi	st 🖓 Expr	e f(	) Funct		
□ \$\\$\$\$ \$\\$\$\$ \$\\$\$\$ \$\\$\$\$ \$\\$\$\$ \$\\$\$\$ \$\\$\$\$ \$\\$ \$\\$\\$\$\\$	Source airectori	Linked: GDB Dibug + es searcneu: c: \users \user \user \user	cullencs \us-nuk		🔩 Linked	d: GDB Debug		9× 🗶 🥖	2 5	$\bigtriangledown$
CDB Debug application exit: code 255	start	•		Name	Value		Туре	Count	Size	
Application terminated #2 terminated	wait			🖻 🗁 Locals	5 var	riables				^
RPMSG_TTY_RTX_M4 disconnected	Tn LinuxTTY.c	d at breakpoint 1: 0x000086F4		– 🌒 argc	19	96271616	int		32	
	0x000086F4 61,	0 {		🗟 🗞 argv	0x	.00000839	char**	1	32	
	Deleted temporar	y breakpoint: 1		🕀 🗞 portnar	ne	"ð"	char*	1	32	
	wait			- 🛛 fd		0	int		32	
	NORMAL TERMINATI	ON		Let 🗞 but		"â"	char[]	14	112	
	_		~	- 🗁 File Static	Variables 0 of 0 v	/ariables				~
	<		>	<					)	۶
Status: application exit: code 255	Command: Press (Ct	trl+Space) for Content Assist	Submit	Add Variable					Brows	se
📄 keepbusy.sh 💽 LinuxTTY.c 😒			- 8	111 Disassem	3 🗄 Memory 🔳	Stack 🔚 Ev	ents 🖁	Outline		
52 {	1		^						1	÷
53 printf ("Error %d from tcsetattr", er	rno);				tinker	+ GDB Debug	-		~	
54 return -1;				D 10 - Mar	a lasta atlant	1000 00000				_
55 } 56 return 8:					d instruction>	100				_
57 }				Address	Opcode Disass	sembly				
58				0x000086F0	BD80 POP	{r7,pc	}			^
59				0000000012	main					
600 int main(int argc, char "argv[])				0x000086F4	B590 PUSH	{r4,r7	, lr}			
62 char *portname = "/dev/ttyRPMSG";				0x000086F6	B089 SUB	sp,sp,	#0x24			
63				0x000086F8	AF00 ADD	r7,sp,	#0			
64 int fd = open (portname, O_RDWR   O_NOCTT	Y 0_SYNC);			0x000086FC	6039 STR	r1,[r7	#01			
65 IT (10 K 0)				0x000086FE	F648031C MOV	r3,#0x8	881c			
67 printf ("Error %d opening %s: %s"	, errno, portname, s	trerror (errno));		0x00008702	F2C00300 MOVT	r3,#0				
68 return -1;				0x00008706	61FB STR	r3,[r7	,#0x1c	1		~
69 }				٩					,	<u> </u>
71 set_interface_attribs (fd, B115200, 0);				App Console	🔀 🔳 Target Consc	ole 🧕 Error	Log		-	
73 write (fd, "Hello from A7!", 14); 74					🔄 Linked	d: GDB Debug	÷	Q. 6	i 🔂	Ň
75 usleep (10000);				Process /home/	/root/tmp/Linux	Applicatio	n TTY	created	; pid	- ^
76 77 chan buf[14]				Listening on p	port 5000					
78 read (fd, buf, sizeof buf);				Debug session Remote debugge	has been starte	d, connect	ing to	gdbser	ver	
79				Error 2 openin	ng /dev/ttyRPMSG	: No such	file o	r direc	tory	
80 printf ("Get Message From Remote Side: %s	", buf);			Child exited w	with status 255					
01			×							×
			,						,	·
GDB Debug application exit: code 255 (Linux Application Debug	<ul> <li>Application Debug)</li> </ul>									

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

## **Additional links**

Kernel.org: http://www.kernel.org/doc/Documentation/arm/Booting

Debugging with scripts:

https://developer.arm.com/docs/dui0446/latest/debugging-with-scripts

Debug configurations: <u>https://developer.arm.com/docs/dui0446/latest/ds-5-</u> <u>debug-perspectives-and-views/debug-configurations-debugger-tab</u>

## Index

### Α

Applications	
Add Source Code	32
Blinky with CMSIS-RTOS RTX	26
Build	34
Build Cortex-M Image	34
Create BIN File	57
Create Cortex-M	26
Create Source Files	30
Customize RTOS	30
Run from U-Boot	59
Select Software Components	29
Setup the Project	27
Store BIN File	58

### С

Console 1	1			/	
-----------	---	--	--	---	--

### D

Debug	
Applications	
Cortex-M	
Linux	42
Linux Kernel	43
Linux Kernel Module	
Streamline	55
Device Database	10
Documentation	11
DS-MDK	
Install	9
Installation Requirements	8
Introduction	7
Licensing	8
e	

### Ε

Eclipse	
IDE	 61

Perspectives	61
Example Project	
Install	
Example Projects	

### F

Flash Programmig	
Scatter File	33

Linux	
Install Image	12
Linux Applications	
Build Application Image	36
Development	35
Project Set Up	35

### Ρ

Perspective	
C/C++	
CMSIS Pack Manager	65
DS-5 Debug	67
Remote System Explorer	

### R

### S

### Т

Terminal View	/ 1	14
---------------	-----	----