

Getting Started with Multicore SDK (MCSDK)



Contents

Chapter 1 Overview..... 3

Chapter 2 Multicore SDK (MCSDK) components..... 4

 2.1 Embedded Remote Procedure Call (eRPC)..... 4

 2.2 Multicore Manager (MCMGR).....5

 2.3 Remote Processor Messaging Lite (RPMsg-Lite).....5

Chapter 3 MCSDK demo applications..... 7

Chapter 4 Inter Processor Communication (IPC) levels..... 8

Chapter 5 Revision history.....9

Chapter 1

Overview

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

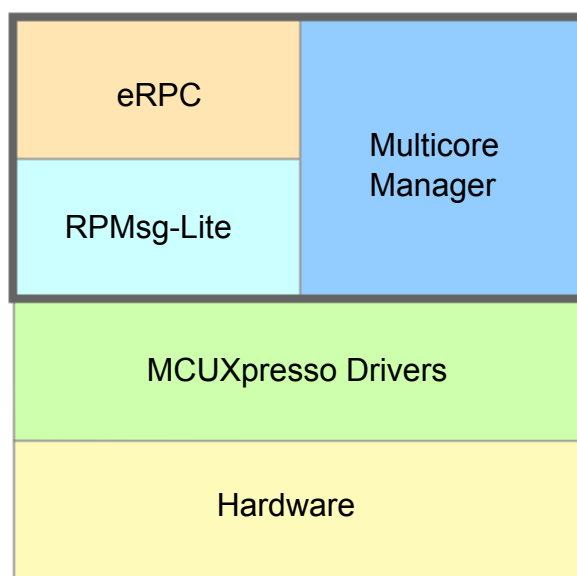


Figure 1. MCSDK layers

All the MCSDK-related files are located in `<MCUXpressoSDK_install_dir>/middleware/multicore` folder.

For supported toolchain versions, see the *Multicore SDK v.2.7.0 Release Notes* (document MCSDKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

Chapter 2

Multicore SDK (MCSDK) components

The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary core(s).
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter Processor Communication library.

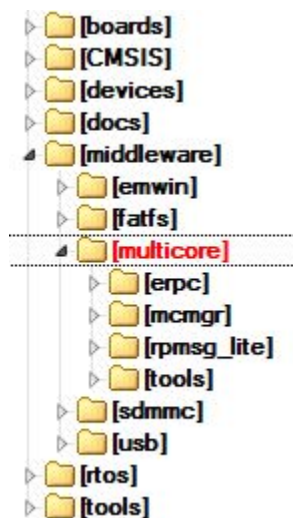


Figure 2. Multicore folder structure

2.1 Embedded Remote Procedure Call (eRPC)

The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshalled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, etc). The server unmarshalls the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshalled and sent back to the client.

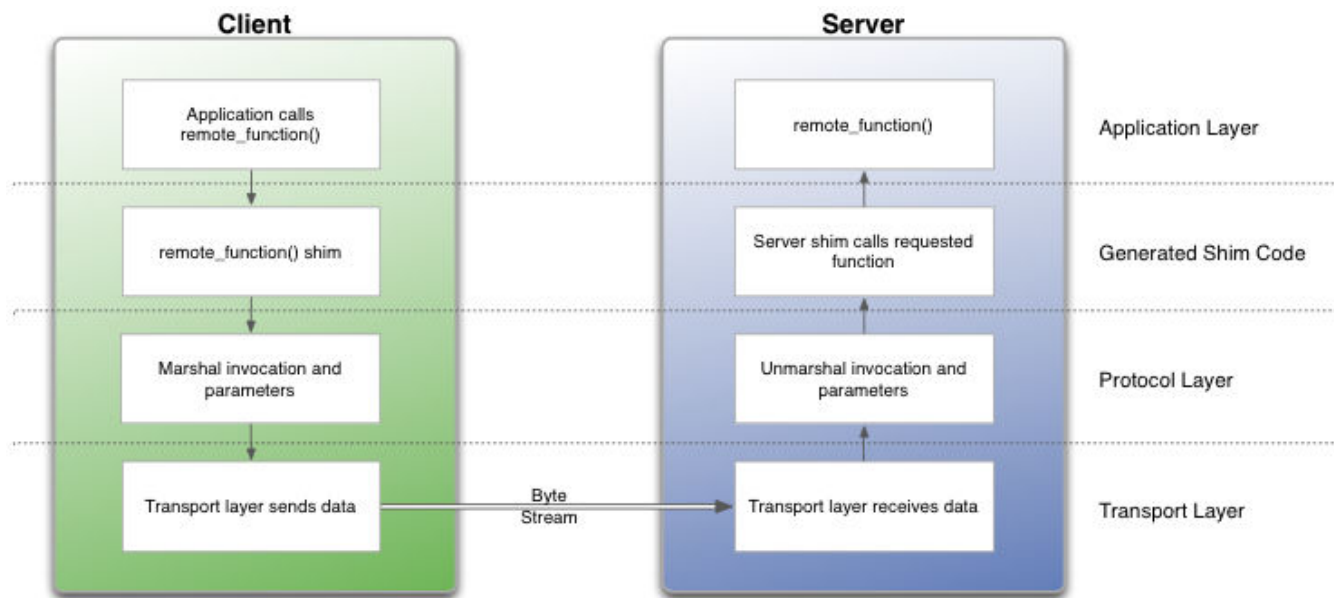


Figure 3. eRPC block diagram

RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshalling a function's parameters and building the data stream.

Main eRPC features:

- Scalable from bare metal to Linux® OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc` folder. For detailed information about the eRPC, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

2.2 Multicore Manager (MCMGR)

The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr` folder. For detailed information about the MCMGR library, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc` folder.

2.3 Remote Processor Messaging Lite (RPMsg-Lite)

RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

Multicore SDK (MCSDK) components

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.
- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPMsg-Lite library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite` folder. For detailed information about the RPMsg-Lite, see the RPMSG-Lite User's Guide located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/doc` folder.

Chapter 3

MCSDK demo applications

Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

Table 1. Multicore example applications

Location	Folder
Multicore example projects	<MCUXpressoSDK_install_dir>/boards/<board_name>/ multicore_examples/<application_name>/<core_type>/ <toolchain>
Multiprocessor example projects	<MCUXpressoSDK_install_dir>/boards/<board_name>/ multiprocessor_examples/<application_name>/<core_type>/ <toolchain>

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

Chapter 4

Inter Processor Communication (IPC) levels

The MCSDK provides several mechanisms for Inter Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner. The most lightweight method of IPC uses the MCUXpresso SDK lowlevel drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications. For details about individual driver API functions see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied by the RPSMsg-Lite documentation that shows how to use this API in multicore applications.

Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPSMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism. The RPSMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API. However, this costs several KB of code and data memory. The MCUXpresso SDK is accompanied by the RPSMsg-Lite documentation and several multicore examples. You can also obtain the latest RPSMsg-Lite code from the GitHub account github.com/NXPmicro/rpsmsg-lite.

Remote procedure calls

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPSMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API). The generated code can be either C/C++ or Python (In case the communicating peer is running as a Linux OS user-space application).

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, BLE, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: github.com/EmbeddedRPC/erpc. Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

Chapter 5

Revision history

This table summarizes revisions to this document.

Table 2. Revision history

Revision number	Date	Substantive changes
0	09/2015	Initial release
1	03/2016	Updated for the SDK 2.0.0 and the MCSDK 1.1.0
2	09/2016	Updated for the MCSDK 2.0.0 and the LPCXpresso54114 support
3	09/2016	Updated for the MCSDK 2.1.0
4	3/2017	Updated for the MCSDK 2.2.0
5	05/2017	Updated for the MCSDK 2.2.1, added Chapter 4, "Inter Processor Communication Levels"
6	11/2017	Updated for the MCSDK 2.3.0
7	05/2018	Updated for the MCSDK 2.4.0
8	12/2018	Updated for the MCSDK 2.5.0
9	06/2019	Updated for the MCSDK 2.6.0
10	12/2019	Updated for the MCSDK 2.7.0

How To Reach Us

Home Page:

www.nxp.com

Web Support:

www.nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: www.nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C - 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C - Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2019.

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 12/2019

Document identifier: MCSDKGSUG

arm