

Renesas RA Family **Developing with RA8 Dual Core MCU**

Introduction

This application project highlights the performance advantages of the dual-core RA8P1 Series, which includes the Cortex-M85 (CM85) core operating at 1 GHz and the Cortex-M33(CM33) core at 250 MHz. It discusses the dual-core architecture and provides use cases that illustrate how to effectively partition tasks between the two cores. Additionally, it addresses development and debugging processes specific to dual-core systems. This application note outlines the creation of applications that enhance performance with Renesas RA8 dual-core MCUs, utilizing both the CM85 core with Helium[™] and the CM33 core at the same time.

It provides guidance on the necessary steps to create an application for the RA8 dual-core MCU, including.

- Application highlights
- Use cases block diagram
- Tool configuration
- Example projects confirmation

Required Resources

The following resources are referenced throughout this application note.

Development Tools and Software

- e² studio version: 2025-04.1 (25.4.1)
- LLVM Embedded Toolchain for Arm v18.1.3
- Renesas Flexible Software Package (FSP) v6.0.0 or later.

Target Devices

Below are the Renesas MCU products to which the information within this document is applicable:

Hardware

- EK-RA8P1- v1.0
- The description in the application note uses PC running Windows[®] 10 OS as an example. Refer to the corresponding user manual for the Development Tools and Software for a complete list of Operating Systems supported.
- One USB device cable (type-C) is used to connect the EK-RA8P1 and the PC.

Reference Manuals

- RA Flexible Software Package Documentation Release v6.0.0
- Renesas RA8P1 Group User's Manual Hardware Rev.1.1
- EK-RA8P1-v1.0 Schematics



Contents

1. Example Application Overview	4
2. RA8 Dual Core MCU	4
3. Create RA8 Dual Core Application with Renesas e ² studio	4
3.1 Create A Solution Project for RA8P1 Dual Core MCU	4
3.2 Debug and Run RA8 Dual Core Project on Cortex [®] -CM85 Core and Cortex [®] -CM33 Core	8
4. Developing Application Using RA8 Dual Core MCU	14
4.1 Partition the system and maximize performance	14
4.2 Using Inter-Processor Communication in Application	14
4.2.1 Using Inter-Processor Interrupts	14
4.2.2 Using Inter-Processor Communication FIFO Messages	15
4.3 Using Shared Memory and Resources in RA8 Dual Core MCU	15
4.3.1 Using Share Memory and Resources in FSP Flat Projects	15
4.3.2 Using Share Memory and Resources in RTOS Based Projects	18
4.4 Utilize Caches and TCM In RA8 Dual Core Applications	18
4.4.1 Tightly Coupled Memory (TCM)s	18
4.4.1 Improve Performance Using ITCM	19
4.4.2 Improve Performance Using DTCM	22
4.4.3 Improve Performance Using CTCM	23
4.4.4 Improve Performance by Utilizing Data Cache Cortex®-CM85 Core	25
4.4.5 Using Neural Processing Unit (NPU)	25
5. Application Projects	27
5.1 IPC - Share Memory Project	27
5.1.1 Implement Inter-Processor Communication in Application.	29
5.1.2 Implement Share Memory between Two Cores in Application.	30
5.2 RTOS/IPC/Share Memory/TCM Projects	34
6. Verify the e ² studio Projects	35
6.1 Import The Projects	35
6.2 Build Projects	35
6.2.1 Compile Project Developed on CM85 Core	36
6.2.2 Compile Project Developed on CM33 Core	37
6.3 Download and Run Projects	38
7. Verify the FreeRTOS-Based Projects	42
7.1 Import The Projects	42
7.2 Build Projects	42
7.3 Download and Run Projects	43
8. References	46





1. Example Application Overview

The example application projects included in this document outline the fundamental procedures for developing an application on the RA8 dual-core MCU using Renesas FSP. They demonstrate methods for partitioning tasks between CPU cores by utilizing the Inter-Processor Communication (IPC) module, sharing memory and resources to enhance performance on dual-core MCUs. They also illustrate the uses of tightly coupled memory (TCM) and cache in a dual-core MCU.

2. RA8 Dual Core MCU

The RA8 dual-core is an asymmetric architecture with a High-performance 1 GHz Arm® Cortex®-M85 core and 250 MHz Arm® Cortex®-M33 core, enabling the concurrent execution of multiple tasks by leveraging both cores with the following key features.



Figure 1. Example of Key Features in RA8P1 MCU

3. Create RA8 Dual Core Application with Renesas e²studio

When creating a project, you need to select the project type, provide a name and location, and configure the project settings. The primary settings include the FSP version, target board, toolchain version, and debugger, as shown in Figure 3. This section offers step-by-step instructions for creating a dual-core project featuring a Blinky solution project. An FSP solution project includes a solution project along with separate projects for CPU0 and CPU1. The solution project establishes the overall memory configuration for both CPU cores, allowing users to define and configure memory partitions for all projects within the solution.

3.1 Create A Solution Project for RA8P1 Dual Core MCU

For RA8P1 MCU applications, follow these steps to create a dual-core project on e² studio. This section is based on the **Bare Metal-Blinky** project template. Another option available when creating a project is the Bare Metal-Minimal template.

Launch e² studio, click File > New > Renesas C/C++ Project > Renesas RA, and select Renesas RA FSP Solution > Next.



c,	New A Open File Open Projects from File System	lt+Shift+N ≻	63 62	Renesas C/C++ Project >> Makefile Project with Existing Code C/C++ Project	Renesas Debug Renesas RA
	Recent Files Close Editor Close All Editors Ctose All Editors Ctose All Editors Save Save As Save All Ctose All	Ctrl+W rl+Shift+W Ctrl+S Ctrl+Shift+S		Convert to a C/C++ Project (Adds C/C++ Nature) Source Folder Folder Source File Header File	X
6. 1	Revert Move Rename Refresh Convert Line Delimiters To Print	F2 F5 > Ctrl+P	8 1 1 1	File from Template Class Example Other	Renesas RA C/C++ Project Create an executable or static library C/C++ project for Renesas RA.
24 24	Import Export Properties	Alt+Enter		(RA)	Renesas RA FSP Solution Create an FSP Solution for Renesas RA comprising a Solution Project and C/C++ projects for each processor core (FSP 6.0.0 and
	Switch Workspace Restart Exit	>		<	· · · · · · · · · · · · · · · · · · ·

Figure 2. Example of Creating a Dual Core Project with RA FSP Solution

Assign a name for this new project such as **ek_ra8p1_blinky**, then **> Next**.

Selecting from the **Device and Tools** Selection. Select **Board** type as EK-RA8P1, the LLVM Embedded Toolchain for Arm for **Toolchains**, and J-Link ARM for **Debugger**, then **> Next**.

🔯 Renesas RA FSP Solution		_	
Renesas RA FSP Solution Device and Tools Selection			Ď
Device Selection FSP Version: 6.0.0 Board: EK-RA8P1 Device: R7KA8P1KFL Language: Image: C Image: C Image	AC Board Evalu Visit get ki Devic Trus	d Description ation kit for RA8P1 MCU G https://renesas.com/ra/ek- it user's manual, quick start re Details tZone Yes	roup ^ guide, v
IDE Project Type e ² studio managed build Toolchains LLVM Embedded Toolchain GNU ARM Embedded	for Arm	igger k ARM	~
18.1.3 v h	anage Toolchains < Back Next :	> Finish	Cancel

Figure 3. Example of Project Selections

Select Bare Metal – Blinky template for this example and click Finish.



Project Template Selection		
Project Template Selection		
Bare Me Bare metal runtime en	etal - Blinky I FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and vironment.	d the C
O Bare Me Bare metal	etal - Minimal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.	
Code Generation Settings	etal - Minimal IFSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.	
Code Generation Settings	etal - Minimal IFSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.	

Figure 4. Blinky Project Template Selection

Note that a dual-core project should be successfully created and built automatically after this step. It consists of three project folders, as shown in Figure 5.

The solution folder named ek_ra8p1_blinky is the solution project.

The CPU0 project folder named ek_ra8p1_blinky_CPU0 is the project for CPU0.

The CPU1 project folder named ek_ra8p1_blinky_CPU1 is the project for CPU1.

File Edit Source Refactor Navigate Search	n <u>P</u> roject Renesas <u>V</u> iews	<u>R</u> un Renesas Al <u>W</u> indow <u>H</u> e	lp
Project Explorer × 🖻 🔄 🖓 🖇 🗖 🗖	[ek_ra8p1_blinky] Solu	tion Configuration \times	
✓	olution Project		Generate Project Content
	List of Clark	PU projects in tion	Renesas
ek_ra8p1_blinky_CPU0	Board: Device: projects Toolchain Version: FSP Version: Location:	R7KA8P1KFLCAC clang_arm 18.1.3 6.0.0 /ek_ra8p1_blinky	
	Summary Memories		

Figure 5. Example of Dual-Core Project Creation Success



A call to the FSP inline function R_BSP_SecondaryCoreStart() in the hal_entry() activates the CPU1, as illustrated in Figure 6.



Figure 6. Example of Starting CPU1 in hal_entry.c within the Blinky Project.

In this blinky example, CPU0 blinks the LED1, and CPU1 blinks the LED2 on the board. This feature allows users to easily identify which core is currently toggling the LEDs, as shown in, as shown in Figure 7.







When rebuilding the project, do it in the following order:

- Build the CPU0 project first.
- Then build the CPU1 project.

Or simplify it by right-clicking on the dual-core solution project and selecting Build Project. The auto-build process will take place in this order: first, the CPU0 project will be built, and then the CPU1 project will follow.

<u>File Edit Source Ref</u>	factor	<u>N</u> avigate S	e <u>a</u> rch	<u>P</u> roject	Renesas	<u>V</u> iews	<u>R</u> u
🗐 📎 - - 🔌			ł	r 🔅	- 🌯 -	1	H
Project Explorer $ imes$	🖻 🔮	\$78□		🔅 [ek_ra	8p1_blink	y] Solu	utior
B ek_ra8p1_blinky				55			
> 😂 ek_ra8p1_blinky	Ν	lew				>	>
> 📂 ek_ra8p1_blinky	0	Go Into					
	S	show In			Alt+Sh	ift+W>	>
		Сору			(Ctrl+C	
(Ē P	aste			C	Ctrl+V	
:	X [Delete			C)elete	
	N	Nove					-1
	F	Rename				F2	
	2	mport					
E	Z E	xport					
	F	Renesas FSP				>	
	E	Build Project			(Ctrl+B	
6	8 F	Refresh				F5	

Figure 8. Example of Building the Dual-Core solution project.

Ensure that the project builds successfully for both cores and that the corresponding application images (ELF files) are generated in the Debug/ directory (e.g., Debug/*.elf).

3.2 Debug and Run RA8 Dual Core Project on Cortex[®]-CM85 Core and Cortex[®]-CM33 Core

To debug dual cores simultaneously, first use either Renesas Flash Programmer or Renesas Device Partition Manager to perform an initialization operation on your MCU to ensure that it is set to protection level 2 and that the trusted zone boundary has not been previously configured. If the trusted zone boundary has been configured, you will need to reset it to establish a proper environment for debugging. Once the initialization is complete, you can proceed to configure the dual-core settings and start the debugging process effectively. If you fail to complete this step, you might face difficulties in downloading your project images and initiating the debug process.

Initialize the device using the Renesas Device Partition Manager.



- 💁 -	Renesas Debug Tools	>		Renesas Device Partition Manager		
Q	Run	Ctrl+F11	T	TraceX >		
裕	Debug	F11	Ð	Tracealyzer >		
	C++ Exceptions	>		Renesas Device Partition Manager		ı x
	Run History	>	1	Refestas Device Function Manager	L	
0	Run As	>				
	Run Configurations			 Enter a value for Action and Emulator type 		
-bla	Debug History	>				
14	Debug As Debug Configurations	/		Device Family: Renesas RA		
Q	External Tools	>		Action		
_			- 11	Read current device information	Change debug	tate
				Set TrustZone secure / non-secure bounda	aries 🔽 Initialize device	
				Target MCU connection:	J-Link ~	
				Connection Type:	SWD ~	
				Emulator Connection:	Serial No V]
				Serial No/IP Address:		1
				Debugger supply voltage (V):	0 ~	
				Connection Speed (bps for SCI, Hz for SWD):	9600 ~	
				Debug state to change to:	Secure Software Develo	pment
				Memory partition sizes		~
				Show Com	mand Line Ru	n

Figure 9. Initialize MCU with RDPM

Figure 10 shows the message displayed after successful device initialization.

SRAM NSC (KB) 0 Display errors in : English Connecting Loading library : SUCCESSFUL! Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED. SUMMARY OF RESULT Connection : SUCCESSFUL!	-	Renesas Device Partition Manager	_		×	-
SRAM NSC (KB) 0 Display errors in : English Connecting Loading library : SUCCESSFUL! Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED. Connecting DISCONNECTED.						
SRAM NSC (KB) 0 Display errors in : English Connecting Loading library : SUCCESSFUL! Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED. SUCCESSFUL! Oisconnecting DISCONNECTED. ISCONNECTED. SUCCESSFUL! Disconnecting DISCONNECTED. ISCONNECTED. SUCCESSFUL! ISCONNECTED. SUCCESSFUL! ISCONNECTED. SUCCESSFUL! ISCONNECTED. SUCCESSFUL! ISUCCESSFUL! SUCCESSFUL!	-					
Display errors in : English Connecting Loading library : SUCCESSFUL! Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED.		SRAM NSC (KB) 0			^	
Connecting Loading library : SUCCESSFUL! Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting Disconnecting DISCONNECTED. SUMMARY OF RESULT Connection Connection : SUCCESSFUL!		Display errors in English				
Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL! CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting Disconnecting DISCONNECTED. SUMMARY OF RESULT Connection Connection : SUCCESSFUL!		Connecting Loading library : SUCCESSFUL!				
CONNECTED. Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTEDSUMMARY OF RESULT Connection : SUCCESSFUL!		Establishing connection : SUCCESSFUL! Checking the device's TrustZone type : SUCCESSFUL!				
Initializing device and rolling back Protection Level to PL2 SUCCESSFUL! Disconnecting DISCONNECTED. SUMMARY OF RESULT Connection : SUCCESSFUL!		CONNECTED.				
Disconnecting DISCONNECTED. SUMMARY OF RESULT Connection : SUCCESSFUL!		Initializing device and rolling back Protection Level to PL2 SUCCESSFUL!				
DISCONNECTEDSUMMARY OF RESULT Connection : SUCCESSFUL!		Disconnecting			- 11	
SUMMARY OF RESULT Connection : SUCCESSFUL!		DISCONNECTED.				
		Connection : SUCCESSFUL!				
Device initialization : SUCCESSFUL! END SUMMARY		Device initialization : SUCCESSFUL! END SUMMARY				
			1			
<v< td=""><td></td><td><</td><td></td><td></td><td>~</td><td></td></v<>		<			~	
Show Command Line Run		Show Command Line		Run		
Close		? è 4		Close	e	

Figure 10. Successful Device Initialization Message on RDPM



To initialize the device using Renesas Flash Programmer:

- 1. Open the Renesas Flash Programmer software.
- 2. Create a new project and establish a connection to the target MCU.
- 3. Navigate to the Target Device tab.
- 4. Click Initialize Device to perform the initialization operation.

File Target Device Operati Read Devic Read Mem Pr Read Flash Initialize D DLM Trans Program and User Ke	Help e Information ory Options vice	Connect Settings	Unique Code	
Command Program >> Verify	Start		Add/Remove Files	

Figure 11. Initialize MCU with RFP

The Status Message will be displayed on the console as Figure 12.

File Target Device Help	
Operation Operation Settings Block Settings Flash Options Connect Settings Unique Code	
Project Information	
Current Project: RA8P1.rpj	
Microcontroller: R7KA8P1ZFLCAC	
Program and User Key Files	
Add/Remove Files	
Command	
Program >> Verify	
<u>S</u> tart OK	
Device Code: 07	
Current state: OEM PL2 Current Authentication Level: AL2	
AL2 Key Trieston: No	
RMA Key Injection: No	
OEM Root Public Key U Injection: No OEM Root Public Key 1 Injection: No	
OEM Root Public Key 2 Injection: No OEM Root Public Key 3 Injection: No	
Frasing the target device	
Betting the target device	
Disconnecting the tool	
Clear status and message	

Figure 12. Successful Device Initialization Message on RFP.



After initializing the device with RDPM or RFP, access the Debug Configuration. Select ek_ra8p1_blinky_CPU1 Debug_Multicore, then navigate to the Debugger tab and click on Connection Settings. Ensure that the TrustZone boundary settings are disabled.

ŷ 🖻 🐢 🗊 🗙 🖻 🍸 ▼ type filter text	Name: ek_ra8p1_blinky_CPU1 Debug_Multicore		~~
🕆 🖻 🍋 🔚 🗶 📄 🖓 🕶	Name: ek_ra8p1_blinky_CPU1 Debug_Multicore		
type filter text			
	📄 Main 🕸 Debugger 🕨 Startup 🔲 Common 💺	Source	
C/C++ Application C/C++ Remote Application EASE Script	Debug hardware: J-Link ARM V Target Device:	R7KA8P1KF_CPU0	
C GDB Hardware Debugging	GDB Settings Connection Settings Debug Tool Set	tings	
💽 GDB Simulator Debugging (RH850)	Prevent Releasing the Reset of the Civis Core	res	× ^
🗸 🚭 Launch Group	Secure Vector Address		
🚭 ek_ra8p1_blinky_CPU1 Debug_Multicore Launch (Non-secure Vector Address	N	
🗸 💽 Renesas GDB Hardware Debugging	Hot Plug	NO	*
ek_ra8p1_blinky_CPU0 Debug_Flat	Disconnection Mode	Continue	~
ek_ra8p1_blinky_CPU1 Debug_Attach	→ SWV	0	
ek_ra8p1_blinky_CPU1 Debug_Multicore	Core clock (MHz)	0	
💽 Renesas Simulator Debugging (RX, RL78)	✓ IrustZone	Ne	
	Set IrustZone secure/non-secure boundaries	NO	~
	Authenticate device to Authentication Level (A	L) None	~
	Authentication key		
			~
			>
< >>		Daviant	Arraha
10 1 1 1 4 5 1 4 4 1 1		Keven	ADDIV

Figure 13. Example of Disabling TrustZone Boundary Setting

The e2studio provides efficient debug capability that allows debugging both core projects simultaneously. You can achieve this by using a "Launch Group" that combines both individual launch configurations. When e2studio generates the CPU1 project, it automatically creates this launch group.

Open the Debug Configurations dialog, select the Debug Multicore Launch Group that was created, and click Debug to begin the debug session.

Debug Configurations	_	- ×
Create, manage, and run configurations Launch several other configurations sequentially		TO.
Y Y Y type filter text C/C++ Application E C/C++ Remote Application F EASE Script C GDB Simulator Debugging GDB Simulator Debugging C GDB Simulator Debugging C Reflexable Dibinky_CPU1 Debug_Multicore Launch Group ✓ Reflexas GDB Hardware Debugging E ex_raBp1_blinky_CPU1 Debug_Hat C ex_raBp1_blinky_CPU1 Debug_Hat E ex_raBp1_blinky_CPU1 Debug_Multicore C Renesas Simulator Debugging (RX, RL78)	Name: ek_ra8p1_blinky_CPU1 Debug_Multicore Launch Group Launches Common Name C Name Renesas GDB Hardware Debugging:ek_ra8p1_blinky_CPU1 Debug_Multicore Inherit Inherit	Up Dowr Edit. Add. Remov
Filter matched 12 of 14 items	Reyert	Apply
0	Debug	Close

Figure 14. Example of Debug Multicore Launch Group

In the Debug Information tab, the debug session for the CPU0 project is launched and connected first, followed by the CPU1 project. The debugger will halt at the Reset_Handler() of the CPU0 ELF file. When you click the Resume button twice, CPU0 will start executing. After passing the main() function, the LED1 on the target board should start blinking, indicating successful operation.





Figure 15. Example of Startup Debug State for Dual-Core Execution.

As shown in Figure 16, once CPU0 invokes R_BSP_SecondaryCoreStart(), the CPU1 debug session fully activates, enabling standard debugging control.

Press the Resume button to continue execution; LED2 will start blinking, indicating that CPU1 has successfully started up.



Figure 16. Example of Debug State after R_BSP_SecondaryCoreStart()

After completing your first multicore debug session for your project pair, you can easily start additional debug sessions by selecting the appropriate Launch Group option from the drop-down menu.





Figure 17. Example of Quick Access Using the Debug Drop-Down Menu



4. Developing Application Using RA8 Dual Core MCU

When planning and designing an application that utilizes RA8 dual-core MCUs, it is crucial to consider several factors to maximize performance.

- Divide the application into multiple tasks that each CPU can process independently.
- Use the IPC (Inter-Process Communication) module to facilitate communication between tasks running on different cores. Additionally, manage shared resources carefully to prevent conflicts, maintain data integrity, and optimize the overall efficiency of the application.
- Utilize ITCM, DTCM, CTCM, and STCM to enhance performance.
- Leverage instruction and data cache to further improve performance.
- Assign heavier tasks, such as graphics-related operations, digital signal processing (DSP), and artificial intelligence/machine learning (AI/ML), to the CM85 core. This core can take advantage of its higher clock frequency, the M-Profile Vector Extension (MVE). In contrast, lighter tasks, including data acquisition (sensor inputs) and user messages (UART input/output), should be managed by the CM33 core.

4.1 Partition the system and maximize performance

We distribute real-time control, AI/ML, and graphics tasks across the CPU cores to leverage the dual-core architecture. For example, we can divide a typical graphics application between the two CPUs as follows.

- CPU0 manages the graphics module, JPEG decoder, and SDRAM access.
- CPU1 handles inputs/outputs and the user interface, including data acquisition, touch controllers, and output controls.

4.2 Using Inter-Processor Communication in Application

Inter-processor communication (IPC) allows for the sharing of hardware resources and the exchange of data between the two processors within the MCU. IPC also supports communication by generating interrupt events that help synchronize and coordinate actions among the processors.

This section highlights the key components of IPC, including:

- Data exchange
- Task synchronization
- Efficient resource sharing

4.2.1 Using Inter-Processor Interrupts

Inter-processor interrupts consist of both maskable and non-maskable types. As illustrated in Figure 18, a typical scenario involves one core managing event monitoring and sending a notification to a second core to trigger the necessary processing.

For implementation details regarding inter-processor interrupts in the application, refer to section 5.1.1.







4.2.2 Using Inter-Processor Communication FIFO Messages.

The IPC has four FIFOs. IPC00 and IPC01 are FIFOs that transmit data from CPU1 to CPU0, and IPC10 and IPC11 are FIFOs that transmit data from CPU0 to CPU1. It has 4 FIFO stages and a transfer data size of 32 bits.

Figure 19 illustrates the data exchange mechanism using IPC Message FIFOs.



Figure 19. Example of Data Exchange with IPC-Messages FIFO

For implementation details on using IPC Message FIFOs in application development, refer to Section 5.1.

4.3 Using Shared Memory and Resources in RA8 Dual Core MCU

Shared memory is one of the most efficient methods of exchanging large amounts of data between two cores in a dual-core system. This mechanism allows two cores to directly access a common memory area to read/write data, but concurrency control is required to avoid race conditions and data corruption.

4.3.1 Using Share Memory and Resources in FSP Flat Projects

When two cores need to exchange high-speed, large amounts of data (e.g., video streaming), shared memory is the ideal choice because of its higher speed compared to other IPC methods. However,



appropriate synchronization mechanisms are needed to avoid race conditions. Figure 20 demonstrates a method for data exchange and notifications to ensure synchronized access to a shared memory region.



Figure 20. Example of Shared Memory Data Exchange and Synchronization

When implementing shared memory in a dual-core system, several key considerations must be addressed.

- Memory Allocation: A dedicated memory region should be reserved and configured as shared memory that is accessible by both cores.
- Synchronization Mechanism: Appropriate synchronization methods, such as mutexes, semaphores, or hardware flags, are necessary to prevent data corruption and ensure correct access sequencing.
- Data Exchange: One core writes data to the shared memory region while the other core reads it as needed, facilitating coordinated communication between the cores.

Figure 21 illustrates the exclusive control flow that occurs when two CPUs execute a semaphore and send core notifications through an inter-processor maskable interrupt.

Refer to 5.1 for instructions on implementing shared memory in the application.





Figure 21. Example of Application Exclusive Control Flow

When using a dual-core system, dedicating certain services to a single core can optimize performance and avoid resource duplication. This strategy ensures that each core can concentrate on its designated tasks without interference, resulting in enhanced efficiency. By effectively managing workloads, users can enjoy smoother performance and faster response times in applications. This method also assists in optimizing code size and addressing resource-sharing challenges. When a task needs these services, it can send a request through the inter-processor communication channel.

Some types of services are shared between the two cores and handled on one core, such as peripheral device management (UART/SPI/I2C), file system management, and network stack management. Figure 22 illustrates the shared UART peripheral, which is managed by CPU1, while CPU0 accesses the UART service via IPC mechanisms.

This architecture is also demonstrated in the accompanying sample application.





Figure 22. Example of Share Resource - UART diagram

4.3.2 Using Share Memory and Resources in RTOS Based Projects

You can create a message queue wrap based on the IPC module in Renesas FSP to communicate and transfer data between the two cores. Refer to RTOS/IPC/Share Memory/TCM Projects for more details.

4.4 Utilize Caches and TCM In RA8 Dual Core Applications

For optimal performance, Tightly Coupled Memory (TCM) and cache can be leveraged in conjunction with Helium™ technology.

TCM typically offers deterministic, single-cycle access, minimizing latency in time-critical operations. Placing performance-critical code and frequently accessed data in TCM ensures faster and more predictable execution.

4.4.1 Tightly Coupled Memory (TCM)s

With the RA8 dual core MCU, each CPU has its own dedicated TCM resource.

- CPU0: Instruction Tightly Coupled Memory (ITCM); Data Tightly Coupled Memory (DTCM).
- CPU1: C-AHB Tightly Coupled Memory (CTCM); S-AHB Tightly Coupled Memory (STCM).

The 256 KB TCM memory in CPU0 consists of 128 KB ITCM and 128 KB DTCM.

The 128 KB TCM memory in CPU1 consists of 64 KB CTCM and 64 KB STCM.

Note: Accessing TCMs is not available in CPU Deep Sleep mode, Software Standby mode, and Deep Software Standby mode.

Figure 23 shows TCM in the local MCU subsystem.

Figure 23. Example of TCM Memory in Local MCU Subsystem

4.4.1 Improve Performance Using ITCM

To achieve optimal performance for time-critical functions, specific program instructions can be placed in Instruction Tightly Coupled Memory (ITCM). This configuration can be adjusted using the Linker Section settings in e²studio.

The following example illustrates how to allocate the arm_cmplx_mag_f32 function to ITCM memory within the RA8P1_DSP_example project using the Linker Sections Configuration interface.

- 1. Open the project in **e² studio**.
- 2. In the **Project Explorer**, double-click on configuration.xml.
- 3. Navigate to the Linker Sections tab (refer to Figure 24).
- 4. Assign the function arm_cmplx_mag_f32 to the designated ITCM section (Figure 25 to Figure 27).

Developing with Developing with RA8 Dual Core MCU MCU

Eile Edit Source Refactor Navigat Image: Image in the state i	te Se <u>a</u> rch <u>P</u> roject Renesas <u>V</u> iews <u>R</u> ▲ ▼] 🖓 ¶	un Renesas Al <u>W</u> indow <u>H</u> e	lp
Project Explorer × □	[RA8P1_DSP_example_cpu0] FSP Comparison	onfiguration ×	
> RA8P1_DSP_example	Linker Section Mapping Co	nfiguration	Generate Project Content
RA8P1_DSP_example_cpu0 \$ \$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$	User Mappings	🗐 New U	ser Mapping > 💼 Remove
) Includes) a ra) ragen) ragen) Debug) Debug) racfg) script) configuration.xml] JLinkLog.log 	Input Section(s) Default Mappings	Output Section	
 ra_cfg.txt RA8P1_DSP_example_cpu0 E RA8P1_DSP_example_cpu1 E O Developer Assistance S RA8P1_DSP_example_cpu1 	Input Section(s) *(.bss.g_heap) *(.bss.g_main_stack) Summary BSP Clocks Pins Interrupt	Output Section RAM Uninitialized RAM Uninitialized	Stacks Components

Figure 24. Open Linker Sections Tab

Click New User Mapping > ITCM > Code initialized from > ITCM code from FLASH.

[RA8P1_DSP_example_cpu0] FSP Configuration $ imes$			Outline	×		
inker Section Mapping Configuration		Generate Project Content	There is r	no active editor that provide	es an ou	tline.
Jser Mappings	된 New Use	r Mapping > 🔝 Remove				
Input Section(s)	Output Section	DATA_FLASH DTCM FLASH	+ > > >			
		OSPI0_CS0 OSPI0_CS1 OSPI1_CS0 OSPI1_CS1 RAM	> > > > >	Code initialized from Data initialized from ITCM Uninitialized ITCM Zeroed	> >	IICM code from OSPI0_CS1 ITCM code from OSPI1_CS1 ITCM code from DATA_FLASH ITCM code from FLASH
Default Mappings		SDRAM				
Input Section(s) *(.bss.g_heap) *(.bss.g_main_stack)	Output Section RAM Uninitialized RAM Uninitialized					

Figure 25. Example of Defining a New Section Mapping for ITCM

A new user mapping window appears. The input section name must adhere to a specific format. For sections that include instruction code, ensure the input section name follows the required naming convention, ".text.<Function Name>". Figure 26 illustrates an example of placing the arm_cmplx_mag_f32 function in ITCM memory.

New User Mapping	×	
.text.arm_cmplx_mag_f32		
	OK Cancel	
	OK Cancel	

Figure 26. Example of Defining Input Section Name for Instruction Code

After completing the configuration shown in Figure 27, follow these steps to apply the changes: Click on **"Generate Project Content"** and then select **"Build Project"** to generate the code and compile the updated project configuration.

[RA8P1_DSP_example_cpu0] FSP Co	🌼 [RA8P1_DSP_example_cpu0] FSP Configuration $ imes$	
Linker Section Mapping Co	Linker Section Mapping Configuration	
User Mappings		뒿 New User Mapping > 👔 Remove
Input Section(s)	Output Section	
*(.text.arm_cmplx_mag_f32)	ITCM code from FLASH	
<		>
Input Section(s)	Output S	Section
*(.bss.g_heap)	RAM Un	initialized
*(.bss.g_main_stack)	RAM Un	initialized
Summary BSP Clocks Pins Interrupts	Event Links Linker Section	ns Stacks Components

Figure 27. Example of Successful Setup for Function in ITCM section.

Upon a successful build, the placement of code or data in the specified memory section can be verified by inspecting the map file located at Debug/*.map. This file provides a detailed memory layout, including section assignments. Refer to Figure 28 for example illustrating the correct mapping of configured sections.

Figure 28. Example of Verifying Code Placement in ITCM

4.4.2 Improve Performance Using DTCM

To improve runtime performance, data buffers may be allocated in the Data Tightly Coupled Memory (DTCM) region. Management of these buffers can be performed through the Linker Sections tab in the e²studio configuration for the project.

The following procedure demonstrates how to locate the testInput_f32_10khz buffer used in the RA8P1_DSP_example project (as shown in Figure 29) in DTCM memory by utilizing the Linker Sections Configuration interface:

Click the Linker Sections tab > New User Mapping > DTCM > Data initialized from ... > DTCM data from FLASH, as shown in Figure 30.

This configuration ensures that the buffer is copied from MRAM to DTCM at startup and enables faster data access during execution.

Figure 30. Example of Defining a New Section Mapping for DTCM

Next, assign "Input section name" corresponding to the initialized data. Use the following format for section naming: .data.
buffer_name>

For example, to map the initialized buffer testInput_f32_10khz, specify the input section as: .data.testInput_f32_10khz.

Refer to Figure 31 for a visual example of this configuration.

New User Mapping
Enter the input section name or glob pattern for the new mapping: .data.testInput_f32_10khz OK Cancel

After finishing the configuration illustrated in Figure 32, click on Generate Project Content, and then choose Build Project to apply the changes and compile the updated project.

*[RA8P1_DSP_example_cpu0] FS	P Configuration $ imes$	
inker Section Mapping	Configuration Ge	nerate Project Conter
User Mappings	New User Ma	pping > 🔊 Remove
Input Section(s)	Output Section	
*(.text.arm_cmplx_mag_f32)	ITCM code from FLAS	4
*(data testInput f32_10kbz)	DTCM data from FLAS	н
<		2
Contract Stanpare (Stanpare (St	Output Section	-
<pre>< Default Mappings Input Section(s) *(bss.g heap)</pre>	Output Section RAM Uninitialized	2
<	Output Section RAM Uninitialized RAM Uninitialized	
<pre> (uddatestanpac_istiotal) (</pre>	Output Section RAM Uninitialized RAM Uninitialized	
Contracted an part of the second s	Output Section RAM Uninitialized RAM Uninitialized	
Contract of the second seco	Output Section RAM Uninitialized RAM Uninitialized	>>> >>

Figure 32. Successful Setup for Data in DTCM Section

Upon a successful build, the placement of code or data in the specified memory section can be verified by inspecting the map file located at Debug/*.map. This file provides a detailed memory layout, including section assignments. Refer to Figure 33 for examples illustrating the correct mapping of configured sections.

Figure 33. Example of Verifying Test Input Data Placement in DTCM

4.4.3 Improve Performance Using CTCM

To achieve optimal performance for time-critical functions on CPU1, specific program instructions can be placed in CTCM. This configuration can be performed using the Linker Sections settings available in the configuration.xml of the CPU1 project within e²studio. By explicitly mapping selected functions to CTCM, the application benefits from reduced instruction fetch latency and improved execution speed.

The following procedure demonstrates how to place the ipc_callback() function into CTCM on CPU1 using the Linker Sections configuration interface in e²studio. This approach ensures that the callback routine executes at high speed and has deterministic memory access, thereby enhancing response time during inter-processor communication events.

- 1. Open the project in **e² studio**.
- 2. In the **Project Explorer**, double-click on configuration.xml.
- 3. Navigate to the Linker Sections tab.
- 4. Assign the function ipc_callback() to the designated CTCM section (Figure 34 and Figure 35).

inker Section Mapping Configuration		Generate Project Content		There is no active editor that provides an outline.			
Jser Mappings		🔊 New User Mappin	CTCM	>	Code initialized from	CTCM code from OSPI0_CS1	
Input Section(s)	Output Section		DATA_FLASH FLASH OSPI0_CS0 OSPI0_CS1 OSPI1_CS0 OSPI1_CS1 RAM SDRAM STCM		CTCM Uninitialized from > CTCM Uninitialized CTCM Zeroed	CTCM code from OSPL2S1 CTCM code from DATA_FLASH CTCM code from FLASH	
Default Mappings							
Input Section(s)	Output Section						
*(.bss.g_heap)	RAM Uninitialize	ed					
*(.bss.g_main_stack)	RAM Uninitialize	ed					

Figure 34. Example of Defining a New Section Mapping for CTCM

📴 New	User Mapping		×
Enter th	e input section name or glob pattern for th	he new mapping:	
.text.ip	oc_callback		
	[OK	Cancel

Figure 35. Example of Defining Input Section Name for Instruction Code on CPU1

After completing the configuration, click on "Generate Project Content," and then select "Build Project" to generate the code and compile the updated project configuration. Once the build is successful, you can verify the placement of code or data in the specified memory section by inspecting the map file located at Debug/*.map, as illustrated in Figure 36.

4.4.4 Improve Performance by Utilizing Data Cache Cortex®-CM85 Core

In the default configuration for RA8 devices, the FSP always enables the CM85 Instruction Cache (I-Cache) and manages its coherency as necessary. The FSP also allows for the optional enabling of the CM85 Data Cache (D-Cache) in the BSP configuration settings, as illustrated in Figure 37, although it is disabled by default. When utilizing any type of cache within a system, it's worth thinking about coherency. For further details, refer to the Cortex-M85 Caches documentation.

EK-RA8P	1	
Settings	Property	Value
	series	8
	✓ RA8P1 Device Options	
	> OFS Registers	
	✓ RA8P1 Family	
	> SDRAM	
	> Security	
	> Clocks	
	✓ Cache settings	
	Data cache	Enabled
	Enable inline BSP IRQ functions	Enabled
	Main Oscillator Wait Time	8163 cycles
	✓ RA Common	
	Main stack size (bytes)	0x1000
	Heap size (bytes)	0

Figure 37. Example of CM85 Data Cache (D-Cache) Enabled

4.4.5 Using Neural Processing Unit (NPU)

The RA8P1 dual core MCU integrates Ethos-U55, which offers support for transformer-based models at the edge, the foundation for newer language and vision models, and scales from 32 to 256 MAC units, enabling higher-performance edge AI use cases in a sustainable way. Offering the same toolchain as previous Ethos-U generations, partners can benefit from seamless migration and leverage investments in Arm-based machine learning (ML) tools. The Ethos-U NPU works alongside the host CPU, providing efficient AI/ML acceleration for applications like computer vision, speech recognition, and anomaly detection.

By offloading ML computations from the CPU to the Ethos NPU, the system achieves higher performance and energy efficiency, enabling AI at the edge without a significant power overhead. Figure 38 and Figure 39 are an NPU block diagram and an example of Ethos configuration in Renesas FSP.

Figure 38. NPU Block Diagram

Figure 39. Example of Ethos Support in Renesas FSP

Refer to the following application notes for developing AI applications on RA8 Dual Core MCUs: "Reference System Design for Vision AI Design using Ethos-U NPU", document No. R11AN0995, and "Using the Ethos-U NPU with RA8 MCUs", document No. R01AN7712.

5. Application Projects

5.1 IPC - Share Memory Project

The implementation of this application project adheres to the following specifications.

CPU1 is responsible for managing the user interface and logging data to the terminal. CPU0 will handle realtime control and sensor data processing.

The task distribution across both cores is depicted in Figure 40.

The peripheral resources utilized in this application project are listed in Table 5.1 below.

Table 5.1. List of the resources used in the project

CPU0	CPU1
Inter-Processor Communication (IPC0 & IPC1)	Inter-Processor Communication (IPC0 & IPC1)
Hardware Semaphore	Hardware Semaphore
Real-Time-Clock control	Serial communication interface UART
12bit-A/D Converter – Temprature Sensor	
I/O Port control – User LEDs	

In this sample application, all runtime information is transmitted to the Tera Term terminal console via the SCI-UART interface, which is managed by CPU1.

Meanwhile, CPU0 is responsible for handling the real-time clock (RTC), sensor data acquisition, and user LED control.

Figure 41 shows the functionality of this application.

Figure 40. Task Partition on Dual Core BareMetal Example

Figure 41. Application Using IPC/semaphore and shared-memory Feature.

Table 5.2 summarizes the IPC and hardware semaphore APIs utilized in this application. These APIs facilitate inter-core communication and synchronization between the processors in the dual-core system.

Table 5.2. Inter Processor Comunication APIs

Functions	Description
R_IPC_Open	Configure an IPC instance
R_BSP_lpcNmiEnable	Assign the user callback for IPC-NMI
R_IPC_MessageSend	Send the message to IPC-Message FIFO
R_IPC_EventGenerate	Generate IPC maskable interrupt to another core
R_BSP_lpcSemaphoreTake	Take hardware semaphore
R_BSP_lpcSemaphoreGive	Release hardware semaphore
R_BSP_IpcNmiRequestSet	Trigger non-maskable interrupt to another core

5.1.1 Implement Inter-Processor Communication in Application.

Inter-Processor Communication (IPC) facilitates both hardware resource sharing and data exchange between two processors within the same CPU system.

The IPC module supports up to 16 hardware semaphores, enabling efficient synchronization between cores.

Additionally, IPC is capable of generating interrupt-driven events to support inter-core signaling, including both maskable and non-maskable interrupts (NMIs).

Figure 42 and Figure 43 illustrate the implementation of IPC maskable interrupts and IPC non-maskable interrupts in an application.

Figure 42. Example of IPC Maskable Interrupt Sample Application

Figure 43. Example of IPC Non-Maskable Interrupt Sample Application.

In this application, one-way FIFOs are also used to exchange simple data between the two CPUs. Specifically, temperature data is sent from CPU0 to CPU1, while the user alarm setting value is sent from CPU1 to CPU0, as illustrated in Figure 44.

Figure 44. Example of IPC Message-FIFO Sample Application

5.1.2 Implement Share Memory between Two Cores in Application.

To implement a shared memory mechanism between two cores, a dedicated memory region must be explicitly defined within the solution project. In this application, the shared memory is allocated to the last 32 KB of CPU0's RAM. The configuration procedure for defining this memory region is described in the following steps:

Change the RAM size of RAM_CPU0_S to 0xE2000.

Memories	5						Generate Proje	ect Co	ntent
Name		Start	Size	Cor	e	Security	^ Add	Partiti	on
📋 RAM_N	٧S	0x32000000	0x1D4000			Non-secure	Add	raruu	
🗸 📋 RAM		0x22000000	0x1D4000			Secure	Remov	/e Part	ition
🔳 RAI	M_CPU0_S	0x22000000	0xEA000	CPU	10	Secure	Clear	Doutiti	0.00
🔳 RAI	M_CPU0_C	0x220EA000	0x0	CPU	10	Non-secure Callable	Clear	Partiti	Ons
R4	Memories							Gener	ate Project Conten
> FLAS	Name	Sta	rt	Size	Core	Securit	y	^	Add Partition
DATA	RAM NS	0x3	2000000	0x104000		Non-se	cure		Add Partition
> DATA	V 🚺 RAM	0x2	2000000	0x104000		Secure			Remove Partition
> SDRA	RAM_CPU0_S	0x2	2000000	0xE2000	CPU0	Secure			
> 🚺 OSPIC	RAM_CPU0_C	0x2	20EA000	0x0	CPU0	Non-se	cure Callable		Clear Partitions
> 🚺 OSPIC	RAM_CPU1_S	0x2	20EA000	0xEA000	CPU1	Secure			
> 🚺 OSPI1	RAM_CPU1_C	0x2	21D4000	0x0	CPU1	Non-se	cure Callable		Sort Partitions
> 🚺 OSPI1	FLASH_NS	0x1	2000000	0x100000		Non-se	cure		
> 🚺 OPTI(> 🚺 FLASH	0x0	2000000	0x100000		Secure			
ummary M	DATA_FLASH_NS	0x3	7000000	0x0		Non-se	cure		
	> 📋 DATA_FLASH	0x2	7000000	0x0		Secure			
	> 📋 SDRAM	0x6	8000000	0x8000000					
	> 🚺 OSPI0_CS0	0x8	0000000	0x10000000					
	> 🚺 OSPI0_CS1	0x9	0000000	0x10000000					
	> 📋 OSPI1_CS0	0x7	0000000	0x8000000					
	\ ■ OCDI1 CC1	0v7	2000000	0~800000				\sim	

Click to RAM > Add Partition.

Memories					Generate Project Content
Name	Start	Size	Core	Security	Add Partition
📋 RAM_NS	0x32000000	0x1D4000		Non-secure	Add Farmon
✓ 📋 RAM	0x22000000	0x1D4000		Secure	Remi ve Partition
RAM_CPU0_S	0x22000000	0xE2000	CPU0	Secure	Char Dartitions
RAM_CPU0_C	0x220EA000	0x0	CPU0	Non-secure Callable	Cear Partitions
RAM_CPU1_S	0x220EA000	0xEA000	CPU1	Secure	
RAM_CPU1_C	0x221D4000	0x0	CPU1	Non-secure Callable	Sort Partitions
FLASH_NS	0x12000000	0x100000		Non-secure	
> 🧻 FLASH	0x02000000	0x100000		Secure	
DATA_FLASH_NS	0x37000000	0x0		Non-secure	
> 📋 DATA_FLASH	0x27000000	0x0		Secure	
> 📋 SDRAM	0x68000000	0x8000000			
> 📋 OSPIO_CSO	0x80000000	0x10000000			
> 📋 OSPI0_CS1	0x90000000	0x10000000			
> 📋 OSPI1_CS0	0x70000000	0x8000000			
> 📋 OSPI1_CS1	0x78000000	0x8000000			
> [] OPTION_SETTING_OFS0	0x02C9F040	0x4		Secure	~
<					>

Figure 46. Example of Adding Partition for Shared Memory Region

In the New Partition pop-up window, as in Figure 47, complete all required fields and click OK.

💽 New F	artition			×
Memory	resource			
Name: R	AM			
Start: 0	<22000000			
Size: 0	<1D4000			
Memory	partition			
Name:	SHARED_MEM			
Start:	0x220E2000			
Size:	0x8000			
Core:	CPU0			\sim
Security:	Secure			\sim
		ОК	Cancel	

Figure 47. Example of Defining a Shared Memory Partition

If an error message appears indicating, "Partition does not follow address order," this means that the memory partitions are not arranged in a strictly increasing address sequence.

To resolve this issue, click Sort Partitions to reorder them correctly, as illustrated in Figure 48. This ensures proper memory mapping and prevents address conflicts during the build process.

Memories					Generate Project Content
Name	Start 0x32000000	Size	Core	Security Non-secure	Add Partition
✓ Ø RAM	0x22000000	0x1D4000		Secure	Remove Partition
RAM_CPU0_S	0x22000000	0xE2000	CPU0	Secure	Clear Partitions
RAM_CPU0_C	0x220EA000	0x0	CPU0	Non-secure Callable	Clear Partitions
RAM_CPU1_S	0x220EA000	0xEA000	CPU1	Secure	
RAM_CPU1_C	0x221D4000	0x0	CPU1	Non-secure Callable	Sort Partitions
SHARED_MEM	0x220E2000	0x8000	CPU0	Secure	
FLASH_NS	0x12000000	0x100000		Non-secure	
> 📋 Flash	0x02000000	0x100000		Secure	
DATA_FLASH_NS	0x37000000	0x0		Non-secure	
> 📋 DATA_FLASH	0x27000000	0x0		Secure	
> 📋 SDRAM	0x68000000	0x8000000			

Figure 48. Example of Shortening Partitions in RAM

The successfully set up partitions are shown in Figure 49 below.

Memories					Generate Project Content
Name	Start	Size	Core	Security	Add Partition
RAM_NS	0x32000000	0x1D4000		Non-secure	
🗸 📋 ram	0x22000000	0x1D4000		Secure	Remove Partition
RAM_CPU0_S	0x22000000	0xE2000	CPU0	Secure	Clear Partitions
SHARED_MEM	0x220E2000	0x8000	CPU0	Secure	Clear Partitions
RAM_CPU0_C	0x220EA000	0x0	CPU0	Non-secure Callable	
RAM_CPU1_S	0x220EA000	0xEA000	CPU1	Secure	Sort Partitions
RAM_CPU1_C	0x221D4000	0x0	CPU1	Non-secure Callable	
FLASH_NS	0x12000000	0x100000		Non-secure	-
> 📋 Flash	0x02000000	0x100000		Secure	
DATA_FLASH_NS	0x37000000	0x0		Non-secure	
> 📋 DATA_FLASH	0x27000000	0x0		Secure	
SDRAM	0v68000000	0v800000			

Figure 49. Example of Successful Setup of Shared Memory Partitions

After completing the configuration, click Generate Project Content.

Next, right-click the solution project and select Build Project to compile the changes.

Upon a successful build, verify the creation of the shared memory partition by double-clicking the .sbd file located at <solution_project>/build/*.sbd. This file confirms the correct partitioning of the shared memory region.

0x1D4000 0x1D4000 0xE2000 0x8000 0x0 0x0 0xE4000	CPU0 CPU0 CPU0	Non-secure Secure Secure Secure Non-secure Callable	
0x1D4000 0xE2000 0x8000 0x0 0xE4000	CPU0 CPU0 CPU0 CPU0	Secure Secure Secure Non-secure Callable	
0xE2000 0x8000 0x0 0xEA000	CPU0 CPU0 CPU0	Secure Secure Non-secure Callable	
0x8000 0x0 0xEA000	CPU0 CPU0	Secure Non-secure Callable	
0x0 0xEA000	CPU0	Non-secure Callable	
0xEA000	CDUIA		
0//L/1000	CPU1	Secure	
0x0	CPU1	Non-secure Callable	
0x100000		Non-secure	
0x100000		Secure	
0x0		Non-secure	~
	0x0 0x100000 0x100000 0x0	0x0 CPUT 0x100000 0x100000 0x100000 0x0	0x0 CPU1 Non-secure Callable 0x100000 Non-secure 0x100000 Secure 0x100000 Secure 0x0 Non-secure

Figure 50. Example of Successful Creation of Shared Memory Partitions

From this point forward, the application can allocate buffers within the shared memory area. The implementation process is illustrated in Figure 51. This configuration guarantees that buffers reside in the shared memory space, enabling seamless data exchange between the two cores.

Figure 51. Example of Placing the Sharing Buffer in the Shared Memory

The fundamental principle of shared memory management is to guarantee exclusive access by a single CPU at any given time. This application employs a combination of hardware semaphores and inter-processor maskable interrupts (MIs) to coordinate and control access arbitration between the cores efficiently.

5.2 RTOS/IPC/Share Memory/TCM Projects

These example projects leverage the RTOS, IPC module, IO interfaces, and TCM memory with data cache enabled to facilitate communication between two CPU cores.

The application is adapted from the official Arm CMSIS-DSP FFT example to run on CPU0 (CM85 core), with computational tasks executed within TCM to maximize performance. FFT results are transmitted to the second core through an RTOS queue based on the on-chip IPC module. The second core then outputs the results via UART to a terminal interface.

When porting CMSIS-DSP examples from Arm, refer to application note R01AN5865EU for detailed integration guidelines.

In this implementation, a FreeRTOS-based queue is integrated with a hardware IPC message FIFO to manage message transactions between the two cores, as illustrated in Figure 52.

Figure 52. FreeRTOS-based Inter-core Messaging via IPC Hardware

In the provided example, FreeRTOS queues are used in combination with hardware IPC to implement data transactions in a dual-core system.

Alternatively, FreeRTOS message buffers or stream buffers can also be employed for inter-core communication depending on the use case and data flow requirements.

6. Verify the e²studio Projects

The e²studio project employs a split project development model to support dual-core application development. Each core project is created as a separate one, utilizing Inter-Processor Communication (IPC) to enable efficient communication and shared memory management between the two cores.

To build and run the example application project ek_ra8p1_dualcore in e²studio, please follow the procedure outlined below.

6.1 Import The Projects

- 1. Launch e² studio IDE.
- 2. Select any workspace in Workspace launcher.
- 3. Close the **Welcome** window.
- 4. Select File > Import.
- 5. Select Existing Projects into Workspace from the Import dialog box.
- 6. Select archive file "ek_ra8p1_dualcore.zip" in the file named ra8x_dual_core.zip.
- 7. Select solution project and developed project samples on each core as shown below, click Finish

📴 Import	
Import Projects Select a directory to search for existing Eclipse projects.	7
O Select roo <u>t</u> directory: ∨ B <u>r</u> owse	
Select <u>a</u> rchive file: C:\ek_ra8p1_dualcore.zip Browse	
Projects:	
✓ ek_ra8p1_dualcore_CPU0 (ek_ra8p1_dualcore_CPU0/) ✓ ✓ ek_ra8p1_dualcore_CPU1 (ek_ra8p1_dualcore_CPU1/) ✓ ✓ ek_ra8p1_dualcore (ek_ra8p1_dualcore/) ✓ Refresh	
Options Search for nested projects Copy projects into workspace Close newly imported projects upon completion Hide projects that already exist in the workspace	
Working sets	
Add project to working sets New Working sets: Select	
(?) < <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel	

Figure 53. Example of Importing Projects into Workspace.

6.2 Build Projects

The easiest method is to click on the solution project (ek_ra8p1_dualcore) and select the "**Build Project**". Building the projects individually requires building the CPU0 project first, then the CPU1 project.

6.2.1 Compile Project Developed on CM85 Core

Double-click on configuration.xml located in the ek_ra8p1_dualcore_CPU0 project > Click "Generate Project Content.

Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X Project Explorer X	<u>F</u> ile <u>E</u> dit <u>S</u> ource Refactor <u>N</u> avigate Se <u>a</u> rch <u>P</u> ro	oject Renesas <u>V</u> iews <u>R</u> un Renesas Al <u>W</u> indow <u>H</u> elp	
Project Explorer × ● \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	🗑 🔻 🔨 🕶 📑 🔯 🕪 💷 🛤 🖎 🕫	→R 検 - Q - B m	
 ▶ ek_ra8p1_dualcore ▶ ek_ra8p1_dualcore_CPU0 [Debug] > e fra < e fra<td>Project Explorer 🗙 🕞 🔄 🛱 🍸 🖇 🖵 🗖</td><td>💮 [ek_ra8p1_dualcore_CPU0] FSP Configuration $imes$</td><td>- 6</td>	Project Explorer 🗙 🕞 🔄 🛱 🍸 🖇 🖵 🗖	💮 [ek_ra8p1_dualcore_CPU0] FSP Configuration $ imes $	- 6
> Dincludes > Gragen > Grage	> 🔁 ek_ra8p1_dualcore Y 😂 ek_ra8p1_dualcore_CPU0 [Debug]	Stacks Configuration	Generate Project Content
> ♥ Developer Assistance > ♥ ek_ra8p1_dualcore_CPU1 Objects New Object > ♥ Remove) Includes <	Threads New Thread Remove HAL/Corr Image: Second Sec	mmon Stacks € New Stack >
Summary BSP Clocks Pins Interrupts Event Links Linker Sections Stacks Components	 Bevelopel Assistance Bek_ra8p1_dualcore_CPU1 	Objects New Object > Remove Summary BSP Clocks Pins Interrupts Event Links Linker Section	ns Stacks Components

Figure 54. Example of Generating Project Content on CPU0 Project

After generating the project content, right-click on ek_ra8p1_dualcore_CPU0, then select Build Project to compile the CPU0 core application.

<u>F</u> ile <u>E</u> dit <u>S</u> ource Refactor <u>N</u> avigate		New Go Into	>	ow <u>H</u> elp
Project Explorer ×		Open in New Window Show In Alt Show in Local Terminal	t+Shift+W > >	on ×
k_ra8p1_dualcore_CPU0 [Debug] Includes Pagen Pagen Pagen Pagen Pactor		Copy Paste Delete Source Move Rename	Ctrl+C Ctrl+V Delete > F2	e
ter figuration.xml		Export Renesas FSP	>	
 > ⑦ Developer Assistance > 6 ek_ra8p1_dualcore_CPU1 	Ş	Clean Project Refresh Close Project Close Unrelated Projects	F5	
		Build Targets Index	>	move

Figure 55. Example of Building CPU0 Dual-Core Project

Verify that the build completes successfully by checking the output in the Build Log console.

6.2.2 Compile Project Developed on CM33 Core

In the ek_ra8p1_dualcore_CPU1 project, double-click on configuration.xml, then click Generate Project Content to apply the configuration settings for the CPU1 core.

Figure 56. Example of Generate Project Content on CPU1 Project

After the content is generated, right-click on ek_ra8p1_dualcore_CPU1, then select Build Project to compile the CPU1 core application.

<u>F</u> ile <u>E</u> dit <u>S</u> ource Refactor <u>N</u> avigate		New Go Into	>	ow <u>I</u>	<u>H</u> elp
Project Explorer ×		Open in New Window Show In Show in Local Terminal	Alt+Shift+W > >	ion >	<
CPU0 ek_ra8p1_dualcore_CPU0 CPU1 [Debug] Includes		Copy Paste	Ctrl+C Ctrl+V	Ē	HA
> 🔑 ra > 🔑 ra_gen > 🍋 src	×	Delete Source Move	Delete >		+
> 🗁 Debug > 🇁 ra_cfg		Rename Import	F2		(
> 🥟 script 🐉 configuration.xml 👔 ek_ra8p1_dualcore_CPU1 Debug_		Export Renesas FSP Build Project	>		
 k_ra8p1_dualcore_CPU1 Debug_ ek_ra8p1_dualcore_CPU1 Debug_ ra_cfg.txt 	ন্ধী	Clean Project Refresh Close Project	F5	ve	
> ⑦ Developer Assistance		Close Unrelated Projects			

Figure 57. Example of Build CPU1 Dual Core Project

Ensure that the build completes successfully by checking the output in the Build Log console.

Alternatively, the build process for both cores can be executed through the solution project. Right-click on the ek_ra8p1_dualcore solution project and select Build Project, as shown in Figure 58. This command will sequentially build all projects within the solution, following the order: CPU0 \rightarrow CPU1.

Figure 58. Example of Build RA Solution Project Dual Core

6.3 Download and Run Projects

As described in the debug settings in Section 3.2, the device must be initialized in the OEM_PL2 state, and TrustZone boundary settings are not required for this configuration.

To start a dual-core debug session, open the Debug Configurations dialog as shown in Figure 59.

Select "ek_ra8p1_dualcore_CPU1 Debug_Multicore Launch Group," then click Debug as illustrated in Figure 60 to simultaneously launch debug sessions for both cores.

Figure 59. Example of Select Debug configuration on Toolbars

Debug Configurations	— 🗆 X
Create, manage, and run configurations Launch several other configurations sequentially	
🕆 🖻 🖗 🗎 🗶 🖻 🍸 🔸	Name: ek_ra8p1_dualcore_CPU1 Debug_Multicore Launch Group
 C/C++ Application C/C++ Remote Application EASE Script GDB Hardware Debugging GDB Simulator Debugging (RH850) Launch Group ek_ra8p1_dualcore_CPU1 Debug_Multicore Launch Group ek_ra8p1_dualcore_CPU0 Debug_Flat ek_ra8p1_dualcore_CPU1 Debug_Attach ek_ra8p1_dualcore_CPU1 Debug_Multicore Renesas Simulator Debugging (RX, RL78) 	Launches Common Name Mode Image: Common intervention of the state of th
	< >>
Filter matched 12 of 14 items	Revert Apply
?	Debug Close

Figure 60. Example of Debug with Multicore Launch Group

The debugging information will appear on the debug console. Before clicking Resume to start application execution, it is necessary to configure the Tera Term terminal to monitor the output and verify application behavior.

To set up the terminal, follow the steps below:

- 1. Open Tera Term (Figure 61).
- 2. Select the appropriate J-Link CDC UART Port from the serial connection options to establish communication with the target device (Figure 62).
- 3. Enable Local Echo by navigating to Setup > Terminal and checking the Local Echo option (Figure 62).
- 4. Configure the terminal settings by selecting Setup > Serial Port, then adjust the baud rate and other serial parameters according to the project requirements (Figure 63).

Tera	Term: New connection		××
C	TCP/IP Host: Service:	myhost.example.com ☑ History ○ Telnet ◎ SSH ◎ SSH ○ Other Protocol: UNSPEC	 ✓ ✓
۲	Serial Port:	COM3: Intel(R) Active Management Tec COM3: Intel(R) Active Management Tec COM138: JLink CDC UART Port (COM13 Cancer Heip	c v chnology 38)

Figure 61. Example of Connect to Tera Term terminal

COM138 - Tera Term VT File Edit Setup Control Window Help Terminal Window Font Keyboard Serial port Proxy	Tera Term: Serial port setup X
SSH SSH Authentication SSH Forwarding SSH KeyGenerator TCP/IP General Additional settings	Port: COM138 OK Speed: 115200 OK Data: 8 bit Cancel Parity: none Help Stop bits: 1 bit Help
	Transmit delay 0 msec/char 0 msec/line

Figure 63. Example of Serial Port Set Up in Tera Term Terminal

After completing the TeraTerm terminal setup, return to e²studio and click Resume three times to start the application execution.

The initial screen layout should appear as shown in Figure 64.

Depending on the selected user input options, the terminal layout will dynamically change. Figure 65 illustrates the various terminal layouts corresponding to different user inputs.

Temperature log in Degree Celsius <u>File Edit Setup Control W</u> indow <u>Help</u>	Real-ti Calenda	me Clock r Date and Time
Date 25: 06 : 2025 Time 15: 25 : 44 Set RIC Calendar Date and Time. Set RIC alarm to toggle the LED at a specific second. S : Stop/Start Scan Temperature Jser Input :		A Banner information
[INFO]: Scanning Temperature [INFO]: The LED will change its state when alarm occurs		and User input
Runtime log Information		

Figure 65. Overview of Application Terminal Layout

Note: The RTC alarm will be activated when the alarm setting for seconds matches the seconds counter in the RTC.

To operate the application project, follow this sequence: option $1 \rightarrow$ option $2 \rightarrow$ option 3. This sequence is necessary because both the temperature printing and the RTC alarm are triggered by the RTC timer.

7. Verify the FreeRTOS-Based Projects

7.1 Import The Projects

- 1. Launch e² studio IDE.
- 2. Select any workspace in Workspace launcher.
- 3. Close the Welcome window.
- 4. Select File > Import.
- 5. Select Existing Projects into Workspace from the Import dialog box.
- 6. Select archive file "dsp_example_dual_core.zip" in the file named ra8x_dual_core.zip.
- 7. Select solution project and developed project samples on each core as shown below, click Finish.

📴 Import			
Import Projects			
Select a directory to search for existing Eclipse projects.			
○ Select roo <u>t</u> directory:	~	B <u>r</u> owse	
Select <u>a</u> rchive file: C:\dsp_example_dual_core.zip	~	B <u>r</u> owse	
Projects:			
RA8P1_DSP_example_cpu0 (dsp_example_dual_core/RA8P1_DSP_example_cpu0/)		Select All	
		Deselect All	
	┛╽	R <u>e</u> fresh	
Options	Options		
Search for nested projects			
✓ Copy projects into workspace			
Close newly imported projects upon completion			
Hide projects that already exist in the workspace			
Working sets			
Add project to working sets		Ne <u>w</u>	
Working sets:	\sim	S <u>e</u> lect	
(?) < <u>Back</u> Next > Finis'	h	Cancel	

Figure 66. Example of Build DSP Example Project.

7.2 Build Projects

Build the projects sequentially in the order CPU0 \rightarrow CPU1, as detailed in Section 6.2. Alternatively, rightclick on the solution project and select Build Project to compile all contained projects at once, as illustrated in Figure 67.

Renesas RA Family

Figure 67. Example of Build DSP Example Project.

Ensure that the build completes successfully for both CPU0 and CPU1 projects by verifying the build status in the Build Log console.

7.3 Download and Run Projects

As described in Section 3.2, the device must be initialized in the OEM_PL2 state, with no TrustZone boundary settings required.

To start debugging both cores simultaneously:

- 1. Open Debug Configurations as shown in Figure 68.
- 2. Select "RA8P1_DSP_example_cpu1 Debug_Multicore Launch Group." as shown in Figure 69.
- 3. Click Debug to launch the dual-core debug session.

Figure 68. Example of Open Debug Configuration in DSP Example

Debug Configurations	-	
Create, manage, and run configurations Launch several other configurations sequentially		Ś
[🕆 🕫 🗊 🗶 🖻 🏹 🗸	Name: RA8P1_DSP_example_cpu1 Debug_Multicore Launch Group	
type filter text	Launches Common	
 C/C++ Application C/C++ Remote Application EASE Script GDB Hardware Debugging GDB Simulator Debugging (RH850) I aunch Group RA8P1_DSP_example_cpu1 Debug_Multicore Launch Group Renesas GDB Hardware Debugging RA8P1_DSP_example_cpu0 Debug_Flat RA8P1_DSP_example_cpu1 Debug_Attach 	Name Mode Image: Renesas GDB Hardware Debugging::RA8P1_D! Inherit Image: Renesas GDB Hardware Debugging::RA8P1_D! Inherit	Actic
RA8P1_DSP_example_cpu1 Debug_Multicore Renesss Simulator Debugging (BX, B178)	<	>
Filter matched 12 of 14 items	Revert	Apply
?	Debug	Close

Figure 69. Example of Debug DSP Dual Core DSP Example

Open the serial terminal and connect to the J-Link CDC UART Port with the following settings: Baud rate 115200 bps, 8-bit data, none parity, 1 stop bit, and no flow control. The Figure 70 and Figure 71 illustrate the connection process and configuration steps using Tera Term terminal.

Tera Term - [disconnected] VT File Edit Setup Control Window Help	ion X
ОТСР/ІР	Host: myhost.example.com
• Serial	Port: COM3: Intel(R) Active Management Tec ~ COM3: Intel(R) Active Management Technology - S0 COM154: JLink CDC UART Port (COM154)

Figure 70. Example of Connect to JLink CDC UART Port

COM154 - Tera Term VT File Edit Setup Control	Window Help		- 🗆 X	
	Fera Term: Serial port setup Port: Speed: Data: Parity: Stop bits: Flow control: Transmit dela 0 mse	COM154 ~ 15200 ~ 8 bit ~ none ~ 1 bit ~ none ~ vy ec/char 0 ms	OK Cancel Help	

Figure 71. Example of Set Up Tera Term Serial Terminal

Return to e² studio and press Resume ^{III} three times to execute the application across both cores. Upon completion, the CMSIS-DSP FFT example will output its status to the terminal, as illustrated in Figure 72.

💆 COM8 - Tera Term VT		_		\times	
File Edit Setup Control	I Window Help				
The DSP application Press s1 button to s ====================================	======================================			^	
FFT Size Sampling Rate Frequency Res. Peak Frequency Bin Peak Frequency Peak Magnitude	: 1024 : 10000 Hz : 9 Hz : 213 : 2080.00 Hz : 327.57				
Reference Bin Test Status CPU Cycles Used Execution Time	: 213 : ARM_MATH_SUCCESS : 195020 cycles (DWT) : 195.02 (us)				
The DSP application has finished execution. Press s1 button to start application ¬					

Figure 72. The Successfully Result Log CMSIS-DSP FFT Example

8. References

The following documents were used in creating this Quick Design Guide:

- RA8P1 Group User's Manual: Hardware, document No. R01UH1064
- RA8P1 Memory Architecture App Note, document No R01AN7880
- Reference System Design for Vision AI Design using Ethos-U NPU, document No. R11AN0995
- Using the Ethos-U NPU with RA8 MCUs, document No. R01AN7712
- Arm Cortex[®]-M85 Processor Technical Reference Manual, document No. 101924, available from Arm

Website and Support

Visit the following vanity URLs to learn about key elements of the RA family, download components and related documentation, and get support.

RA Product Informationwww.renesas.com/raRA Product Support Forumwww.renesas.com/ra/forumRA Flexible Software Packagewww.renesas.com/FSPRenesas Supportwww.renesas.com/support

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	June.30.25	-	Initial release

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power is supplied until the power is supplied until the power reaches the level at which reseting is specified.

3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a systemevaluation test for the given product.

Notice

- Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
- Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
- 3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
- 4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
- 5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
- 6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

- 7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
- 8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
- 9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
- 10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
- 11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
- 12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
 Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas
- Electronics products.
- (Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.