Renesas RA8 Series

Getting Started with Cache on RA Dual-Core MCUs

Introduction

This document provides an overview of the Renesas RA8 dual-core MCU family, with a particular focus on its dual-core architecture and the cache memory subsystem. It aims to help developers understand the structure, configuration, and practical usage of cache memory while developing high-performance, real-time applications on RA8 devices.

This document explores the key aspects of Instruction and data caches in the RA8 dual-core MCUs, detailing their configuration, management, and techniques for achieving data coherency, deterministic execution, and enhanced system performance. It explains how the dual-core architecture integrates with bus masters and memory subsystems, and how instruction and data caches can be efficiently configured and utilized to reduce memory access latency, improve throughput, and enhance responsiveness in real-time embedded applications.

Note: Currently, D-Cache usage is entirely the user's responsibility. Users are required to perform all necessary D-Cache maintenance or ensure that data is placed in predefined non-cacheable regions.

By the end of this document, readers will gain a clear understanding of how to configure, manage, and optimize cache memory in RA8 MCUs using the Renesas Flexible Software Package (FSP), while maintaining data coherency across cores, external memories, and peripherals.

This application note covers the following key topics:

- Overview of the RA8 dual-core MCU bus and cache architecture: Understanding how the dual-core system and cache hierarchy are structured and how they interact with shared resources.
- Implementation and management of instruction and data caches: Detailed guidance on enabling, configuring, and maintaining cache operations for different application needs.
- Benefits of cache utilization: Demonstrating performance gains achievable through Instruction and data caches with practical examples.
- Configuration using the FSP: Step-by-step instructions on enabling and tuning cache features through the FSP framework.
- Best practices for system optimization: Recommendations to ensure data integrity, predictable execution, and cache consistency across multiple cores and hardware components.

Required Resources

- Flexible Software Package (FSP) v6.3.0
- Renesas e² studio - Version: 2025-12
- Segger J-Link RTT Viewer 8.74a or later

Supported Boards

- EK-RA8P1

Supported MCU Groups

At the time of the release, the supported MCU groups are:

- RA8P1 Group
- RA8T2 Group
- RA8D2 Group
- RA8M2 Group

Note**:** The App Note and the accompanying App Project are developed for the RA8P1 dual-core MCU as a reference. However, since RA8T2, RA8D2, and RA8M2 MCUs share the same architecture, these resources can also serve as valuable references for other dual-core MCUs. The application projects can serve as a reference for recreating applications for other RA8 dual-core MCUs.

Contents

1. Introduction

The Cache is a small, high-speed memory located adjacent to the CPU core. Its primary function is to reduce access latency by storing and retrieving frequently used data and instructions, thereby minimizing repeated access to main memory and slower external storage. Because external and internal memory devices may require multiple CPU cycles to complete read or write operations, the cache memory provides a critical performance advantage by enabling single-cycle access to recently used information.

Cache memory is not part of the system memory map and therefore has no physical address. Its contents are managed automatically by hardware control logic rather than by the programmer. Because of this design, cache data and control registers cannot be accessed by bus masters other than the CPU. Serving as a high-speed buffer between the CPU and main memory, cache reduces overall memory access time. Access speed depends on whether the requested data results in a cache hit or a cache miss, with hits being much faster and misses requiring additional cycles.

The following sections provide an overview of the dual-core system along with the cache architecture, cache organization, operation modes, and the performance considerations relevant to AXI-connected memory systems.

2. Dual-Core System Architecture

2.1 Overview of the Dual-Core System

The Renesas RA8 dual-core MCU family integrates a heterogeneous dual-core architecture designed to deliver high performance, security, and real-time responsiveness in advanced embedded systems.

The system architecture in a Renesas RA8 dual-core MCU is engineered to facilitate efficient communication and data exchange between the cores and connected peripherals. The system primarily employs a shared bus architecture, where both cores interface with a common system bus linked to memory and peripherals. To maintain orderly access and prevent conflicts, arbitration mechanisms are implemented to manage resource sharing and prioritize requests fairly. In addition, efficient bus mechanisms are integrated for accessing Tightly Coupled Memory (TCM), enabling low-latency and high-speed memory operations.

To manage shared resource access and coordinate operations between the two cores, the RA8 dual-core architecture implements IPC mechanisms. These include FIFO buffers, interrupt signaling, shared memory regions, and semaphores, all of which support synchronization and efficient data exchange. IPC ensures that both CPUs can work together effectively while preserving system stability and performance.

The underlying RA8 dual-core bus architecture is built on standard high-speed communication protocols such as AXI, AHB, and APB within Arm's AMBA (Advanced Microcontroller Bus Architecture) standard, which facilitate data transferring with memory and peripherals for achieving communication between the Cortex-M85 and Corex-M33 cores.

RA8P1 dual-core MCU features a high-performance Arm® Cortex®-M85 core operating at speeds of up to 1 GHz, alongside an Arm® Cortex®-M33 core running at up to 250 MHz, and offers the following key features:

- Up to 1 MB of MRAM for non-volatile memory storage
- Cache Memory of 32KB for Cortex®-M85
  - 16KB Instruction Cache(I-Cache)
  - 16KB of Data Cache(D-Cache)
- Cache Memory of 32KB for Cortex®-M33
  - 16KB Instruction Cache (C-Cache)
  - 16KB of Data Cache(S-Cache)
- 2 MB of SRAM, including:
  - 256 KB TCM RAM for the Cortex-M85 core
  - 128 KB TCM RAM for the Cortex-M33 core
  - 1.664 MB of user-accessible SRAM
- Integrated Arm® Ethos™-U55 Neural Processing Unit (NPU)
- Octal Serial Peripheral Interface (OSPI) for high-speed external flash memory access
- Layer 3 Ethernet Switch Module (ESWM), USB Full-Speed (USBFS), USB High-Speed (USBHS), and SD/MMC Host Interface

- Graphics LCD Controller (GLCDC) for advanced display support
- 2D Drawing Engine (DRW) for hardware-accelerated graphics rendering
- Support for MIPI DSI/CSI interfaces for display and camera connectivity
- Rich set of analog peripherals
- Comprehensive security and safety features for reliable operation

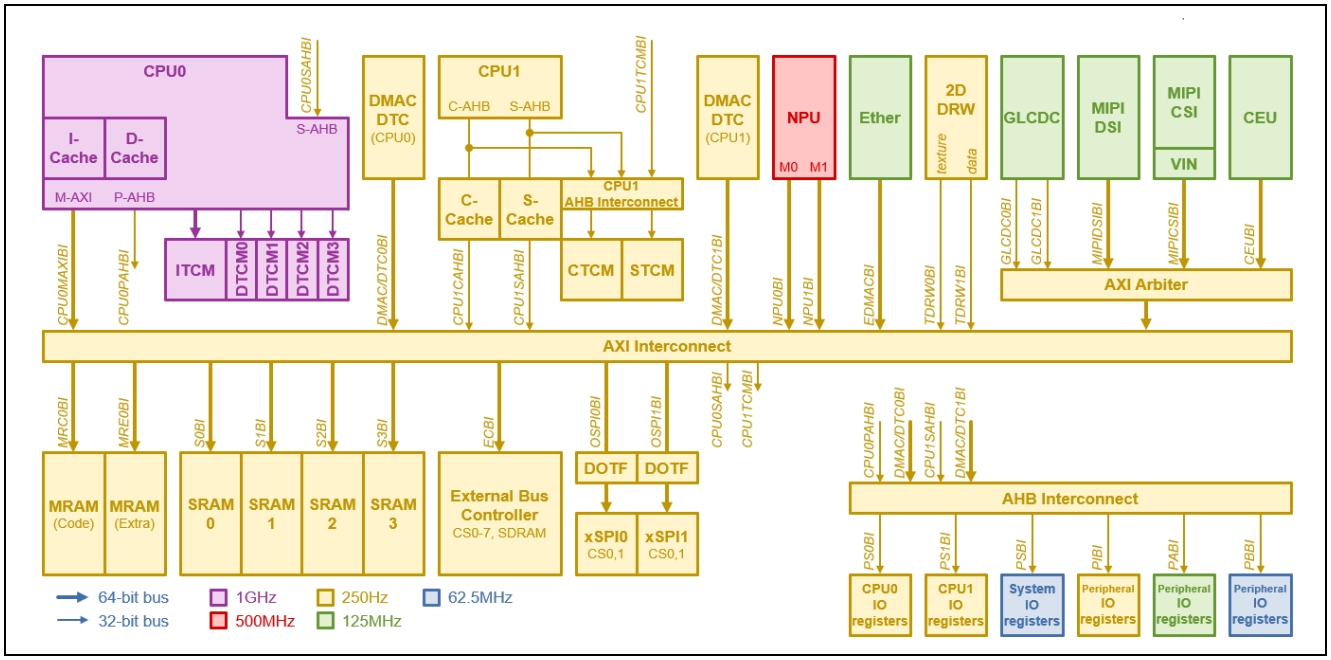

Figure 1. RA8P1 dual-core system architecture overview

For more information on the Architecture of RA8 Dual Core MCU, please refer to the “Developing with dual-core RA8P1 MCUs”(R01AN7881) App note.

2.2 System Interconnect and Bus Architecture

Two cores communicate and access shared system resources through a multi-layer AXI/AHB interconnect fabric managed by the RA8 MCU’s System Bus Matrix. Each master (CM85, CM33, DMA, and peripheral bus masters) can access multiple memory and peripheral regions concurrently.

Key components of the interconnect are:

- AMBA AXI Bus protocol(Advanced eXtensible Interface):
  Provides high-bandwidth, low-latency data transfers for the CM85 and DMA controllers. AXI channels support burst transactions and efficient cache line fetches for instruction and data access.
- AMBA AHB Bus protocol(Advanced High-performance Bus):
  Used primarily by the system peripherals requiring deterministic timing.
- Bus Masters:
  - CM85 Core (AXI Master)
  - CM33 Core (AHB Master)
  - DMA Controller(s)
  - Peripheral DMA Engines (e.g., Ethernet, USB)
  - Debug Access Port (DAP)

The System Bus Matrix handles arbitration and priority control among these masters. This ensures that high-priority real-time requests from either core are serviced promptly while maintaining coherency in shared memory transactions.

2.3 Overview of CM85 and CM33 Cores

The RA8 dual-core MCUs pair a high-performance Cortex-M85 (CM85) application core with a lower-power Cortex-M33 (CM33) management/security core. This asymmetric pairing is intended so the CM85 can run compute and data-intensive user applications (DSP/ML, signal processing, complex control loops), while the CM33 handles security, boot, low-power housekeeping, safe I/O, and real-time supervision, by improving both performance and system safety/power efficiency.

- Cortex-M85 Core (CM85):
  - Armv8.1-M architecture with Helium (M-Profile Vector Extension, MVE) for DSP and ML applications.
  - Single-precision FPv5-SP FPU support for floating point acceleration.
  - Typical operating points in the RA8 family reach up to 1.0 GHz on CM85 in the newest RA8 groups (earlier RA8M1 variants run at lower clocks like 480 MHz). These cores deliver very high CoreMark performance for embedded AI/vision/ML tasks.
  - Cache: 32 KB split (16 KB I-cache + 16 KB D-cache) on the CM85 matches typical RA8 designs, helps reduce external memory bandwidth and latency for real-time compute.
  - The CM85 typically executes user applications, middleware, and performance-critical control loops.
- Cortex-M33 Core (CM33):
  - Armv8-M with Arm TrustZone® for M (secure/non-secure worlds), ideal for secure boot, key management, crypto, peripheral protection, and running a minimal RTOS/monitor.
  - Runs at a lower clock (examples: 250 MHz in recent high-end RA8 parts) and has a 32 KB cache (16 KB for Instruction + 16 KB for Data).
  - The secondary core can handle system management, security control, and low-power operations. The CM33 can function as a security monitor, handling secure boot, cryptographic functions, or peripheral protection, while also acting as a coordination and communication layer for the CM85.

2.4 RA8 Dual-Core Cache Subsystem Overview

The RA8 cache is a small, high-speed memory located next to the CPU core. It stores frequently accessed data and instructions to reduce the number of accesses to slower main memory.

Main memory access may require multiple CPU cycles, particularly when external memory interfaces introduce wait states. To mitigate this delay, a cache provides single-cycle access to recently used information, improving overall system performance.

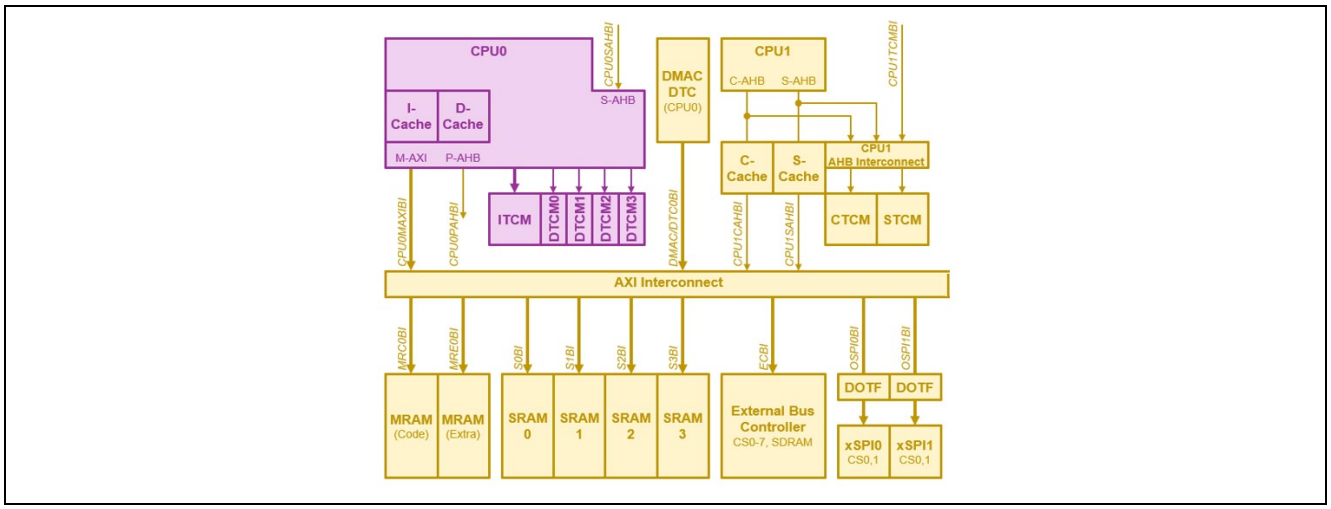

Figure 2. RA8P1 dual-core Subsystem Architecture Overview

Referring to the cache subsystem architecture shown in Figure 2, the RA8 dual-core MCU features independent cache structures for each CPU core. Both CM85 and CM33 have dedicated instruction and data caches, but their implementations differ in terms of location and design origin.

For Cortex-M85, both the Instruction Cache (I-Cache) and Data Cache (D-Cache) are integrated directly within the core itself. These are part of the standard ARM core implementation and are managed internally by the CPU.

Whereas Cortex-M33 does not have a cache feature built in. Cortex-M33 utilizes external cache units, specifically the Code-bus Cache (C-Cache) and System-bus Cache (S-Cache). These caches are located outside the CM33 core and are connected via the C-AHB and S-AHB buses, respectively. Unlike CM85's internal caches, these are custom implementations developed by Renesas to enhance memory access efficiency and system performance for CM33.

The RA8 cache is organized into multiple cache lines, which are grouped into sets. The number of lines per set (associativity) is defined by the hardware.

When the CPU issues a read operation, the cache determines whether the requested address is already stored:

- **Cache Hit:** The data is available in the cache and is returned immediately.
- **Cache Miss:** The data is not present and must be fetched from main memory. The fetched data is then stored in a cache line for future access.

Write operations follow one of two policies:

- **Write-Through:** Data is written to both the cache and main memory.
- **Write-Back:** Data is written only to cache. The cache line is marked dirty and written back to main memory only upon eviction.

If all lines in a selected set are occupied during a miss, the cache invokes its replacement algorithm (commonly pseudo-random) to select a line for eviction. If the line is dirty, it is written back to memory before reusing.

The cache memory provides significant performance benefits for memory accessed through the AXI bus(CM85)/AHB buses(CM33), including internal Flash, internal SRAM, and external memory devices. Tightly Coupled Memory (TCM) behaves differently and is not cacheable.

Four primary cache operations are supported: enable, disable, clean, and invalidate. Renesas FSP provides dedicated APIs to perform these operations. These are CMSIS APIs available for CM85 cache maintenance operations. FSP does not provide APIs for CM33 cache maintenance.

2.5 Cacheability and Shareability

The RA8 dual-core features a memory architecture where the memory can be configured as cacheable or non-cacheable via MPU configuration. To maintain the Coherency it is sometimes required to configure them as required for the application.

Memory Region	Typical Use	Cacheable	Accessible By
ITCM/DTCM	Fast deterministic code/data (CM85)	No	CM85/CM33
CTCM/STCM	Fast deterministic code/data (CM33)	No	CM33/CM85
SRAM0/1/2/3	Shared inter-core data and DMA buffers	Configurable	CM85 & CM33
External SDRAM	Graphics or large data buffers	Yes (via MPU)	Both cores
External Flash (OSPI/XIP)	Code storage and XIP	Yes (read)	Both cores

Both cores have independent access to shared SRAM regions, and each can mark regions as cacheable, non-cacheable, or bufferable through the MPU and SAU (Security Attribution Unit for TrustZone) configuration. Shared memories must be carefully managed to maintain cache coherence during simultaneous access.
The cache control is done by the cache control register, but the MPU can specify the cache policy and whether the region is cacheable or not.

Cache control and MPU configuration are per core, not shared between CM85 and CM33. Each core has its own MPU, I-Cache, and D-Cache, and its cache enable/disable, maintenance, and MPU region settings affect only that core's associated cache and memory accesses. Configuring the MPU or caches on CM85 does not impact CM33, and vice versa.

If both cores need the same behavior, they must independently configure identical MPU regions and cache policies.

3. Cache Architecture and Implementation

CM85 subsystem includes Arm implemented I-Cache and D-Cache units. CM33 subsystem includes Renesas implemented C-Cache and S-Cache units. The C-cache is similar to the I-cache used for Instruction fetch, and the S-cache is for Data access. Because the RA uses a Harvard architecture, instruction fetches and data accesses occur through separate interfaces. The I-Cache handles lookups and allocations only for instruction fetch operations, while the D-Cache performs lookups and allocations solely for data read and write operations.

Having an I-Cache, D-Cache, C-Cache, and S-Cache in RA8 MCUs significantly improves performance and efficiency. I-Cache and C-cache reduces instruction fetch latency by storing frequently used instructions close to the CPU, which is especially beneficial for loops and real-time tasks. D-Cache and S-Cache accelerate data access by caching frequently used variables and buffers, lowering memory access times and power consumption. Together, they minimize external memory traffic, enhance throughput, and support complex applications like DSP or RTOS-based systems. This combination leads to faster execution, better real-time behavior, and improved energy efficiency, making them essential for modern high-performance MCUs.

The Cortex-M85 CPU supports Harvard instruction and data caches to improve performance when accessing on-chip or external memory, but only transactions through the Master-AXI interface are cacheable. Cache maintenance operations are defined relative to two key points: the Point of Unification (PoU), where instruction and data caches share a consistent view of memory, and the Point of Coherency (PoC), where all system components see the same memory state. For Cortex-M85, both PoU and PoC are at the system level when caches are present, as indicated by CLIDR register fields. At reset, the CPU can automatically invalidate caches, though this can be disabled for retention scenarios.

During invalidation, cache operations are ignored, and a DSB instruction ensures completion. Software can also invalidate caches manually; one instruction suffices for the instruction cache, while the data cache requires iterating through all entries. Direct cache access registers allow secure profiling and debugging of cache contents, with access restricted by the LOCKDCAIC signal, because the data cache supports write-back, dirty data must be flushed before powering down.

More details of the ARM CM85 Cache Implementation can be found in the reference document - ARM Cortex-M85 Processor Technical Reference

3.1 Cache Types and Functions in RA8 MCUs

Instruction Cache (I-Cache):
The I-Cache minimizes instruction fetch latency by storing recently executed instructions from both internal and external memory sources. In addition to improving the code execution of frequently used code, this cache is particularly useful when executing code from external Flash or OSPI-mapped XIP (eXecute In Place) regions, as it helps mitigate slower memory access times.

Data Cache (D-Cache):
The D-Cache temporarily stores frequently accessed data, reducing redundant memory reads and lowering system bus traffic. This leads to faster data retrieval and improved CPU efficiency. Additionally, it is very helpful to reduce the latency for storing and retrieving from external memories like SDRAM.

Code-bus Cache (C-Cache):
This Cache is Renesas Implementation of I-Cache for CM33. The C-Cache serves CPU1 instruction accesses through the C-AHB interface, enhancing code fetch performance for external memory regions.

System-bus Cache (S-Cache):
This Cache is also Renesas Implementation of D-cache for CM33. The S-Cache optimizes system-level data access for CPU1 via the S-AHB interface, helping to reduce latency when communicating with peripherals or shared system memory.

3.2 Instruction and Data Cache (I/D Cache) in CM85

3.2.1 I-Cache and D-Cache Specification

Item	I-Cache	D-Cache
Size	16KB	16KB
Associativity	4-way set associative	4-way set associative
Line Size	256 bits	256 bits
Number Entry	128 entries/ways (4-way set associative)	128 entries/ways (4-way set associative)
Write Policy	Write-back with write-allocate, write-through	Write-back with write-allocate, write-through
Replacement Policy	Pseudo-LRU	Pseudo-LRU
Cache Maintenance	Invalidate only	Clean, Invalidate, Clean+Invalidate
Enable/Disable	Via SCB registers	Via SCB registers
Alignment for Ops	32-byte aligned	32-byte aligned

Figure 3. RA8 CM85 I-Cache and D-Cache specification

3.3 Code Cache and System Cache (I/D Cache) in CM33

3.3.1 C-Cache and S-Cache Specification

Item	C-Cache	S-Cache
Capacity	16 KB	16 KB
Way	4-way set associative	4-way set associative
Line size	256 bits	256 bits
Number of entry	128 entries/ways (4-way set associative)	128 entries/ways (4-way set associative)
Write way	Write-through/write-back Non write-allocate/write-allocate	Write-through/write-back Non write-allocate/write-allocate
Replace way	LRU	LRU
Error detection	ECC Data memory: SECDED Tag memory: SEDDED	ECC Data memory: SECDED Tag memory: SEDDED
Cache support area	0x0000_0000 – 0x1FFF_FFFF	0x2000_0000 – 0xDFFF_FFFF

Figure 4. RA8 CM33 C-Cache and S-Cache specification

3.3.2 Cache Block Structure

Figure 5 shows the C-Cache and S-Cache Block for CPU1, along with the C-TCM and S-TCM connections to the bus interconnect. Figure 6 shows the CM33 S-Cache and C-Cache structure.

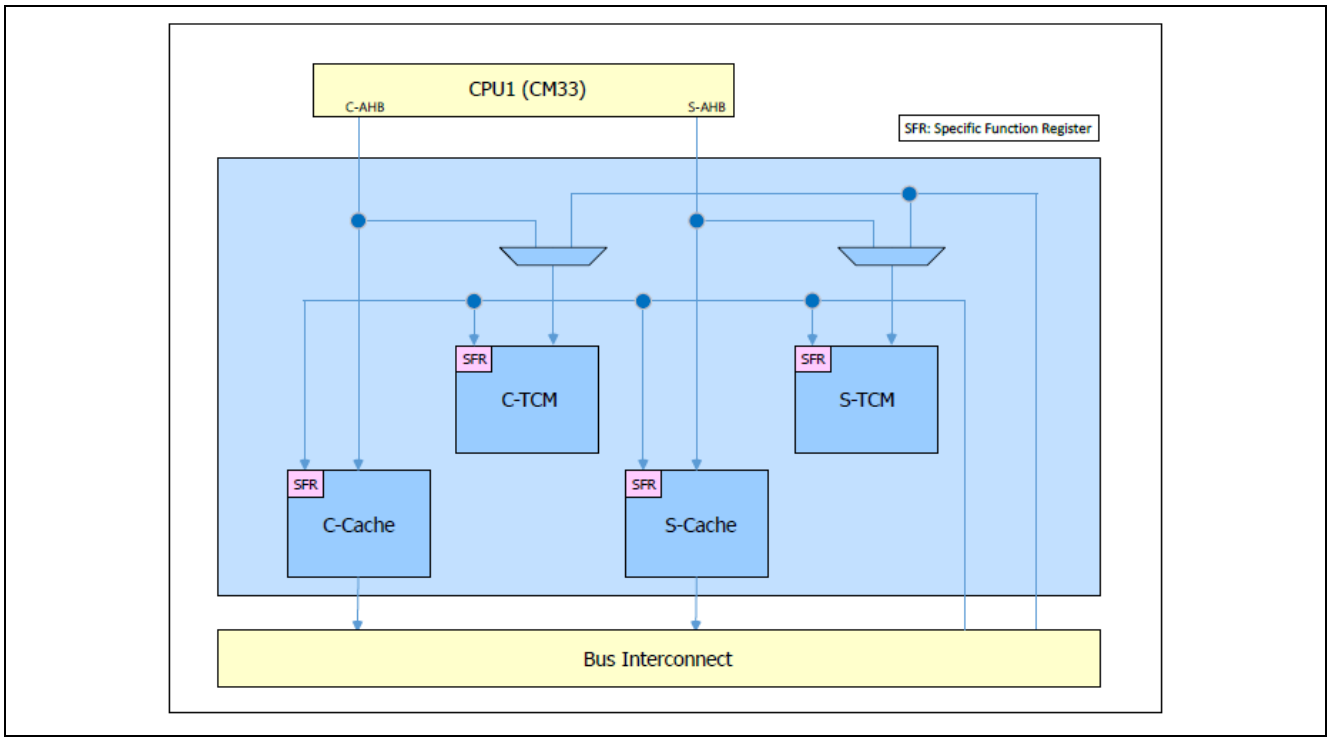

Figure 5. RA8 CM33 C-Cache and S-Cache Block

Each cache line in the Cortex-M33 includes several key components: the **Tag**, which stores the cached address bits 31:12, an ECC code for error detection, and a valid bit (V). The **V bit** indicates whether the data in the cache line is valid, while the **D bit** (dirty bit) shows if the cached data has been modified and needs to be written back to memory. The **Data** field holds 256 bits of actual cached data. Cache replacement uses an **LRU (Least Recently Used)** policy to determine which line to evict. A **Comparator** checks if the requested address matches the tag and if the valid bit is set, confirming a cache hit. Both the tag and valid bits are cleared during a cache flush operation.

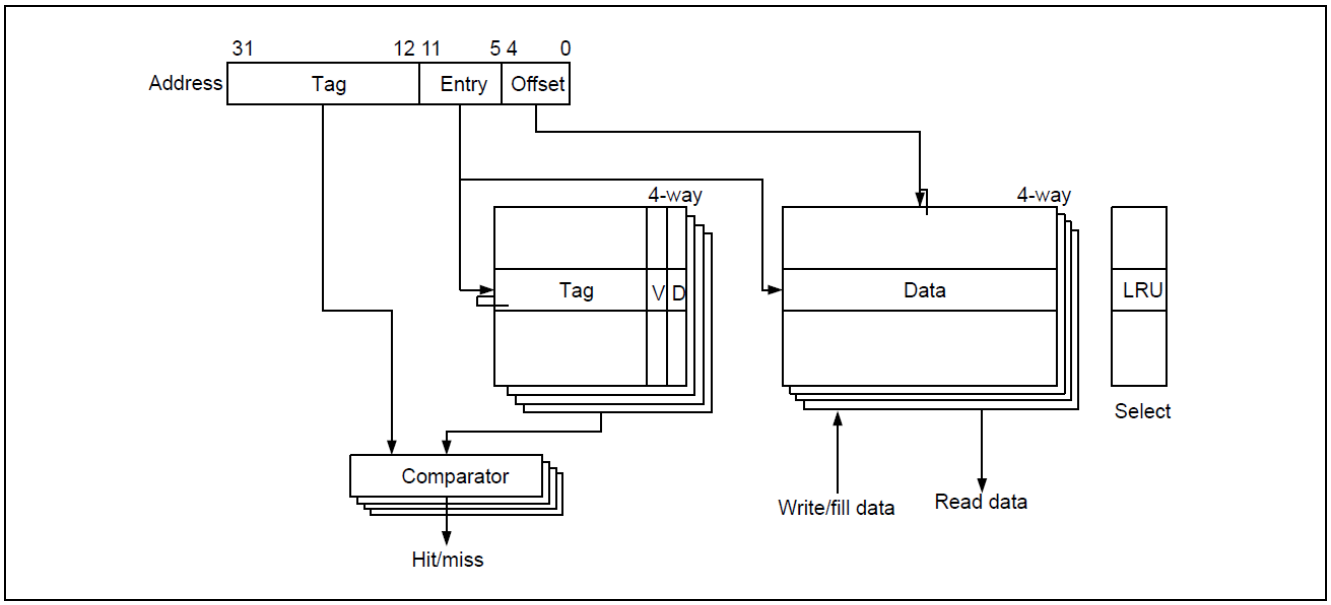

Figure 6. RA8 CM33 C/S-Cache structure

3.4 CM33 Cache Operation and Usage Notes

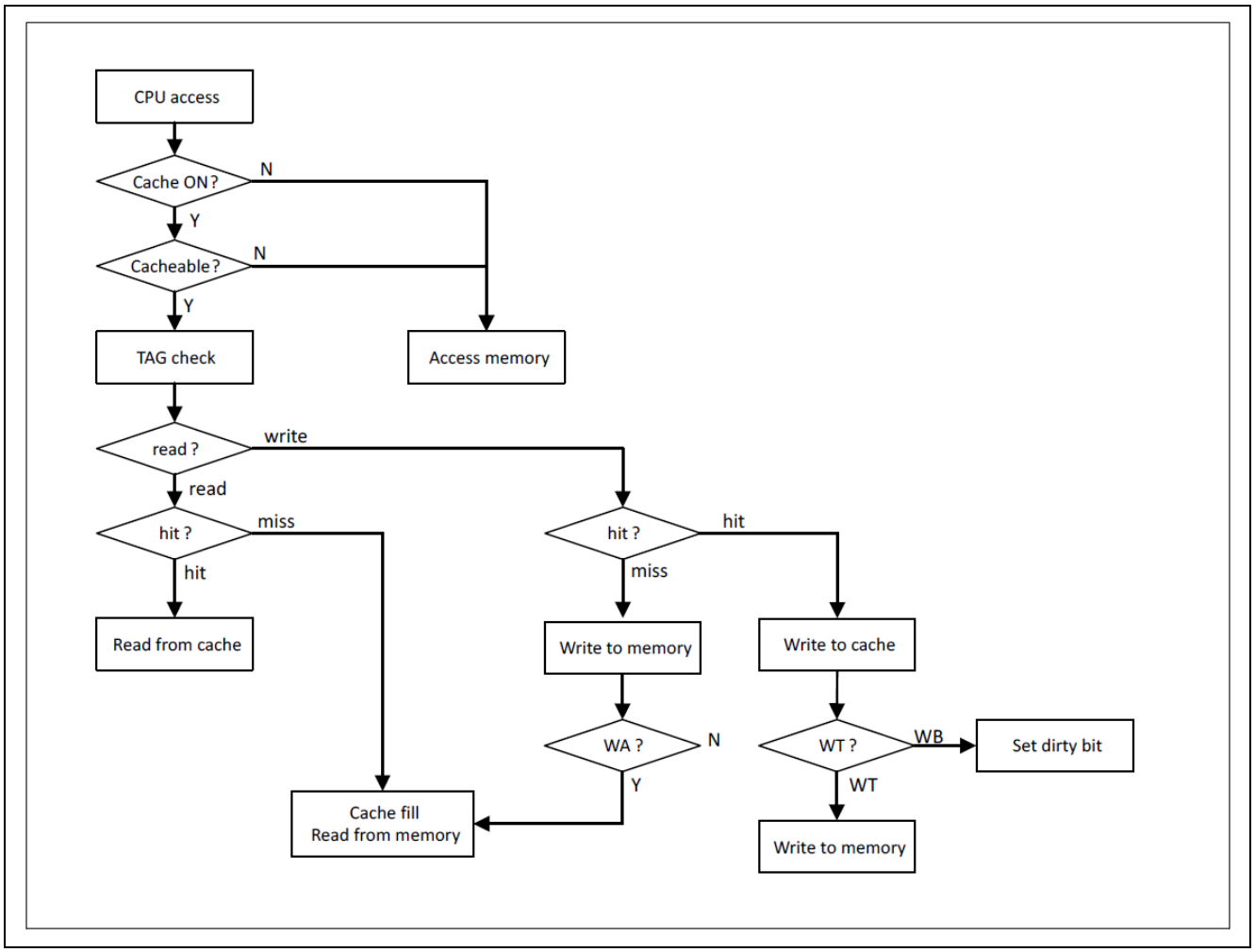

Figure 7. RA8 CM33 C/S-Cache Operation

3.4.1 Cache Operation

The cache operates when caching is enabled (CCACTL.ENC = 1, SCACTL.ENS = 1) and the CPU performs cacheable accesses. It compares the CPU's access address with the tag stored in the cache to determine whether the access is a hit or a miss.

- Read miss: The cache fetches an entire cache line from memory, stores it in the cache, and then returns the requested data to the CPU.
- Read hit: The cache supplies the requested data directly from the cache and completes the access as a zero-wait-state hit.
- Write miss: The cache writes the data to memory. If write-allocation is enabled (CCAWTA.WA = 1, SCAWTA.WA = 1) and the accessed region is configured for write-allocation, the cache also loads a line from memory into the cache.
- Write hit: The cache writes the data into the cache. If write-through mode is enabled (CCAWTA.WT = 1, SCAWTA.WT = 1) or the region is configured for write-through, the data is also written to memory. If write-through is disabled (CCAWTA.WT = 0, SCAWTA.WT = 0) and the region uses write-back, the cache marks the line as Dirty in the tag.

3.4.2 Cache Flush

To flush the cache, set the CCAFCT.FC and SCAFCT.FS bits to 1. This clears the Valid and Dirty bits as well as the Tag information. The Tag ECC value is reset to the code generated from zero. Note that the cache data itself is not cleared.

Note: If an address is later accessed with a different cacheability setting (as configured by the Arm MPU), you must perform a write-back and flush before any subsequent access. When CCAWTA.WT and SCAWTA.WT is both set to 1; a write-back is not necessary.
Accessing the same address with changed cacheability settings *without* performing a write-back and flush will result in undefined read/write data.

3.4.3 LRU and Replace

The cache uses an LRU (Least Recently Used) policy for its replacement algorithm. After determining whether a CPU access is a hit or a miss, the cache updates its LRU state by marking the accessed line as the most recently used. When a replacement is needed, the cache selects the line that has not been used for the longest time. This ensures that even when all cache ways are full, the cache can replace the oldest line based on the LRU information.

3.4.4 Usage Notes for C-Cache and S-Cache

- **Coherency**

Software must ensure coherence between the cache and both external memory and internal SRAM. When sharing memory among multiple bus masters (for example, between multiple CPUs or between a CPU and a DMAC) within a cache-enabled region, you should perform write-back and flush operations on the cache as necessary.

- **Cache Flush/WB Operation**

When CPU1 initiates a Cache Flush or Write-Back operation using *CAFCT/*CACTL, it must verify that the operation has completed by confirming that *CAFCT returns to 0 (* refers to either the C-Cache or S-Cache: CCAFCT/CCACTL or SCAFCT/SCACTL).

Do not trigger Flush or Write-Back operations from any bus master other than CPU1.
During a Cache Flush or Write-Back operation, the cache gives this operation higher priority than any access requests from CPU1.

- **Restrictions Relating to Security Attribution of C-Cache and S-Cache**

When using the C-Cache and S-Cache, one of the following conditions must be satisfied:

- CPU1 must operate exclusively in the secure state.
- CPU1 must operate exclusively in the non-secure state, which is only possible if CPU1's SECEXT (Security Extension) is disabled.
- Both the C-Cache and S-Cache must be used only by non-secure programs. In this case, all of the following requirements must also be met:
    1. Configure the MPU so that all secure memory regions used by secure software are marked as non-cacheable.
    2. Before a secure program accesses non-secure memory, perform a cache write-back and flush.
    3. If an on-chip WDT is used instead of an external WDT, it must be assigned a secure attribute and be configured either to interrupt CPU0 or to trigger a reset when a WDT error occurs.

4. Cache Configuration

4.1 Summary of FSP support for Caches on RA8 dual core

Cache Type	Register definition in io-define	Maintenance API	Cache enabling
CM85 I-Cache	Supported	CMSIS API as part of FSP	Always enabled
CM85 D-Cache	Supported	CMSIS API as part of FSP	Supported; Configurable in the BSP tab property
CM33 C-Cache	Supported	Not provided	Always enabled
CM33 S-Cache	Supported	Not provided	Not supported; Implement in user code

Note: Even though the S-cache and C-cache support is not available in the FSP, similar usage of C-cache and S-Cache can be considered and developed by referring to the Application Project provided in the " RA Family Guidelines for Using the S Cache on the System Bus (R11AN0538EU)" APN. Since the cache-related register structure may differ, please review the hardware user's manual during development.

4.2 Using BSP Settings

Cache configuration is available under the e² studio FSP configurator BSP tab property. D-Cache can be configured using e² studio under the BSP->Cache settings->Data cache properties panel for the project. Similarly, Force D-cache to use write-through for all cacheable writes can be enabled or disabled. By default, these settings are disabled.

Additionally, the ECC functions for the Cache can also be enabled or disabled under the BSP tab using e² studio under the BSP->OFS1 register settings->Tightly Coupled Memory (TCM)/Cache ECC properties panel for the project.

Note: The settings using BSP are available for both CPU0 and CPU1. Currently, the implementation of the S-cache drivers for CPU1 is not available in FSP.

4.3 Using CMSIS APIS

To use the CM85 Cache, CMSIS provides I-cache and D-cache API functions, which can be used in the application as required to configure and operate the caches. The API and its function for the I/D caches are listed below.

API Name	Function	Usage Notes
SCB_EnableICache	If I-Cache allocations are not already enabled, first invalidate the entire I-Cache, then enable I-Cache allocations using SCB.CCR.IC bit	This has no effect if I-Cache allocations are already enabled. The FSP enables the I-Cache at startup by setting SCB.CCR.IC bit directly rather than by calling this function.
SCB_DisableICache	Disable I-Cache allocations with SCB.CCR.IC bit, then invalidate the entire I-Cache.	This function ensures that all I-Cache lines are invalidated and that no further cache allocations occur. It is typically used in low-level initialization, debugging scenarios, or when executing code that must not be cached. Calling this function when the I-Cache is already disabled has no effect.
SCB_InvalidateICache	Invalidate the entire I-Cache.	This function is safe to call at any time since I-Cache lines are never dirty. It is used after modifying instructions in any memory region (Flash or RAM). The FSP invokes this function after initializing the predefined RAM code section during startup, and again when exiting the Code Flash program or erase mode in the Flash HP driver.
SCB_InvalidateICache_by_Addr	Loop to invalidate the instructions in the I-Cache, starting at a particular address and extending for the specified length in bytes.	This is safe to use at any time, because cache lines in the I-Cache can never be dirty. This can be used to more efficiently invalidate instructions at specific addresses. It will invalidate in increments of cache lines (32 bytes).

API Name	Function	Usage Notes
SCB_EnableDCache	If D-Cache allocations are not already enabled, loop to invalidate the entire D-Cache, then enable D-Cache allocations with SCB.CCR.DC bit.	Will do nothing if D-Cache allocations are already enabled. FSP automatically calls this function at startup if BSP_CFG_DCACHE_ENABLED is defined and non-zero.
SCB_DisableDCache	Disable D-Cache allocations with SCB.CCR.DC bit, then loop to clean and invalidate the entire D-Cache.	Disables the Data Cache in the System Control Block (SCB) Suggestion: The D-Cache may contain dirty lines when you disable it, clean first to avoid losing modified data.
SCB_InvalidateDCache	Loop to invalidate the entire D-Cache.	This function is typically used during system initialization; however, certain scenarios may require invalidating the entire D-Cache while the application is running.
SCB_CleanDCache	Loop to clean the entire D-Cache.	Writes back (flushes) all dirty lines from the entire D-Cache to main memory. Lines remain valid and stay in the cache.
SCB_CleanInvalidateDCache	Loop to clean and invalidate the entire D-Cache.	Performs a write-back of all dirty lines in the entire D-Cache to main memory and then invalidates those lines. After the call, the D-Cache no longer holds any valid data for those lines.
SCB_InvalidateDCache_by_Addr	Loop to invalidate the data in the D-Cache, starting at a particular address and extending for the specified length in bytes.	Will invalidate in increments of cache lines (32 bytes).
SCB_CleanDCache_by_Addr	Loop to clean the data in the D-Cache, starting at a particular address and extending for the specified length in bytes.	Will clean in increments of cache lines (32 bytes).
SCB_CleanInvalidateDCache_by_Addr	Loop to clean and invalidate the data in the D-Cache, starting at a particular address and extending for the specified length in bytes.	Will clean and invalidate in increments of cache lines (32 bytes).

4.4 Non-Cacheable Buffer Placement

Predefined non-cacheable sections are only treated as non-cacheable within their respective security state when TrustZone is enabled. This behavior results from the MPU configuration applied by FSP and the Armv8 M architecture, which selects the MPU corresponding to the current security state to determine memory attributes. Cache incoherency will occur if references to these sections are shared across security states. For example, passing a reference from a non-secure application’s non-cacheable section to a secure application will cause the secure application to treat that memory as cacheable.

Additionally, predefined non-cacheable sections are only recognized within their respective core. On dual-core devices, each core has its own MPU and may configure regions independently. Therefore, these sections cannot be used for coherent interprocessor communication when D Cache is enabled. Use the IPC peripheral for such communication.

The predefined non-cacheable regions configured by the MPU when D-Cache is enabled can be used to contain data that should not be cached, ensuring data coherency for that data. To use the predefined non-cacheable regions, place the data into the corresponding non-cacheable section defined by the linker script for the chosen toolchain. The predefined non-cacheable regions are not initialized by the BSP.

Examples of Placing Buffers in Non-Cacheable Regions with the option of Initialization for different compilers.

uint8_t uncached_uninitialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_noinit_nocache");

uint8_t uncached_zero_initialized_buffer_sram[1024] BSP_PLACE_IN_SECTION(".ram_nocache");

Region	Initialization	Name (GCC,IAR,LLVM)	Name (AC6)
SRAM	Uninitialized	.ram_noinit_nocache	.bss.ram_noinit_nocache
SRAM	Zero	.ram_nocache	.bss.ram_nocache
SDRAM	Uninitialized	.sdram_noinit_nocache	.bss.sdram_noinit_nocache
SDRAM	Zero	.sdram_nocache	.bss.sdram_nocache
OSPI0 CS0	Uninitialized	.ospi0_cs0_noinit_nocache	.bss.ospi0_cs0_noinit_nocache
OSPI0 CS0	Zero	.ospi0_cs0_nocache	.bss.ospi0_cs0_nocache
OSPI1 CS0	Uninitialized	.ospi1_cs0_noinit_nocache	.bss.ospi1_cs0_noinit_nocache
OSPI1 CS0	Zero	.ospi1_cs0_nocache	.bss.ospi1_cs0_nocache

5. Cache Coherency

5.1 Cache Management and Coherency

When using any type of cache in a system, data coherency must be carefully managed. A cache may hold data that differs from the main memory, resulting in incoherent data states. Effective cache management requires that users clearly understand data movement within the system, including which entities access the data, its source and destination, and the software responsible for each operation.

Generally Coherency can be maintained through hardware and/or software mechanisms. In Cortex-M devices such as the RA8 series, coherency must be managed entirely through software, as no automatic hardware coherency support is available.

Each cache line supports configurable write-back and write-through policies, controlled through the Memory Protection Unit (MPU) attributes. However, cache coherency is not automatically maintained between all system masters (such as CPUs, DMAs, and peripherals). Therefore, software intervention is required to maintain data consistency; developers must explicitly perform cache clean and invalidate operations when sharing data between cores or during DMA transfers.

By default, the FSP automatically enables CM85 I-Cache and manages its coherence where necessary. However, it does not handle coherency for these caches outside of FSP control. The CM85 D-Cache is disabled by default but can be optionally enabled through the BSP configuration settings. When enabled, additional coherency considerations arise. At present, FSP does not provide built-in D-Cache coherency management. However, the users can handle the coherency in the application using the provided driver APIs.

When the D-Cache is enabled, the most common coherency issue occurs when data is shared between the CPU and another bus master. In such cases, the D-Cache and the corresponding main memory region (typically SRAM) may become incoherent. To maintain coherence in this scenario, one of the following software management techniques must be applied.

These involve explicit instructions to clean, invalidate, or flush cache lines. To maintain cache coherency when the D-Cache is enabled, explicit cache maintenance operations are often required. These include Clean (Write-back), which writes modified (dirty) cache lines back to main memory without invalidating them, typically used before another bus master reads data that the CPU has modified. Invalidate marks cache lines as invalid without writing back to memory, ensuring the CPU reads fresh data from memory after a DMA update. Clean and Invalidate combines both actions, writing back dirty data and then invalidating the cache line, which is useful when full synchronization between memory and cache is needed. Another effective technique is the use of non-cacheable memory regions, where shared buffers (e.g., for DMA) are configured as non-cacheable in the MPU. This approach completely avoids coherency issues since CPU accesses bypass the cache, though it comes with a trade-off of reduced performance for CPU operations on that region.

The Renesas FSP manages the enabling and configuration of the Instruction Cache, while this application project demonstrates how to enable, disable, and flush the Data Cache through example software. It also provides best practices and sample code for maintaining data cache coherency.

5.2 Shared Memory (Between CM85 & CM33)

When I-Cache and D-Cache are enabled on a multi-core MCU like RA8 (Cortex-M85 + Cortex-M33), handling shared memory correctly is critical to avoid data inconsistency. Each core may cache shared memory regions independently. If one core updates data, the other might read stale data unless caches are synchronized. Without proper locking or barriers, race conditions can occur when both cores access shared memory.

Alternatively, set up the Shared Memory Region, and Mark shared memory as non-cacheable if the performance impact is acceptable. And use write-through caching or cacheable but shareable attributes in MPU/MMU configuration.

5.3 Peripheral-Specific Cache Management

Caches improve performance but can cause data coherency issues when peripherals (USB, Drawing Engine, Graphics LCD Controller, Camera Input Capture, NPU) access memory directly. The CPU may have updated data in D-Cache that is not yet written to RAM, or peripherals may update RAM without the CPU knowing. Proper cache maintenance is critical.

Figure 8 shows the system bus connection for RA8P1, here the CPU cores connect to a central system bus that links high-speed memories (such as SRAM) with bandwidth-intensive peripherals like USB controllers, graphics/LCD display controllers, and camera interfaces. These high-throughput blocks typically connect on AXI or AHB buses, while control and configuration registers may be accessed through lower-speed bridges. L1 caches and MPUs are private to each core, so cache and memory attributes must be configured independently per core, using write-back for normal memory and non-cacheable memory-mapped I/O interfaces. If DMA is used by USB, display, or camera blocks and is not hardware-coherent, software must manage cache clean and invalidate operations with proper APIs.

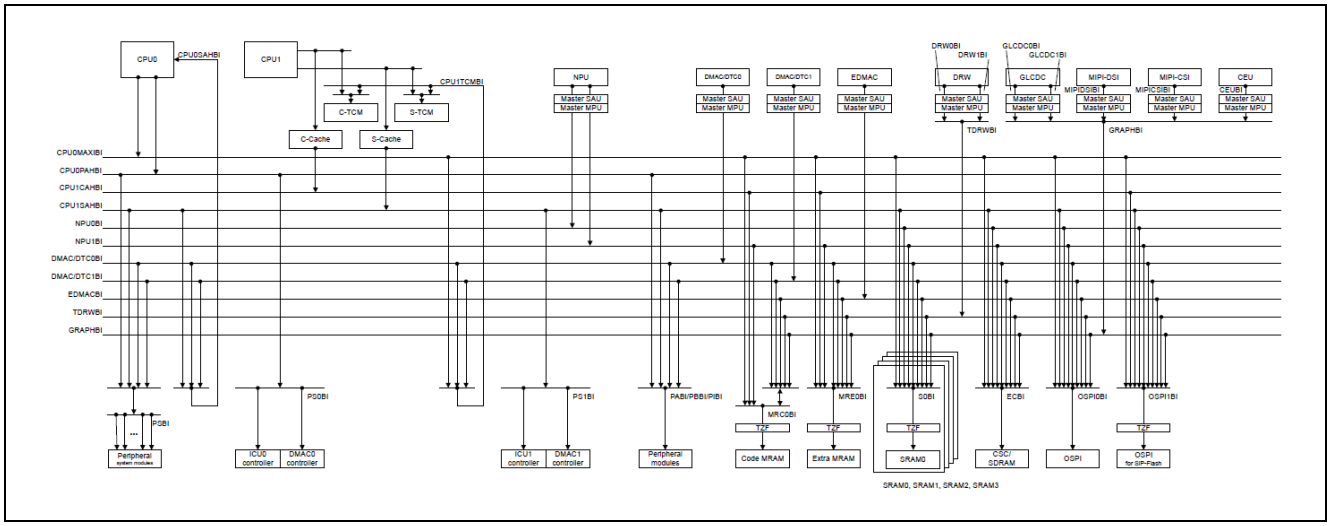

Figure 8. RA8P1 System Bus connection

Ethernet with EDMAC (GWCA function of ESWM) in RA dual-core MCUs

Place descriptor ring buffer in a non-cacheable SRAM region (preferred).

If cached, do explicit cache maintenance:

- Before handing descriptor to DMA: Clean (writeback) cache lines.
- After DMA updates descriptors: Invalidate cache lines before CPU reads.

Transmit Buffers: Before starting DMA transfer, Clean (flush) D-Cache lines for the buffer using SCB_CleanDCache_by_Addr(). This ensures all CPU writes are committed to RAM before DMA reads.

Receive Buffers: After DMA completes, invalidate D-Cache lines for the buffer using SCB_InvalidateDCache_by_Addr(). This ensures the CPU reads fresh data from RAM, not stale cache content.

Descriptor Tables: Apply the same clean/invalidate operations to Ethernet descriptor structures.

USB (Host or Device)

USB controllers often use a general DMAC module for endpoint buffers: Clean cache before giving buffer ownership to USB hardware. Invalidate cache after USB transfer completes.

For isochronous or bulk transfers, consider using non-cacheable memory regions for large buffers to avoid frequent maintenance.

DMA (General)

For any DMA transaction (ADC, SDHI, SPI, etc.): Before DMA read: Clean cache for source buffer.

After DMA write: Invalidate cache for destination buffer.

Use CMSIS APIs for cache maintenance:

SCB_CleanDCache_by_Addr(),

SCB_InvalidateDCache_by_Addr(),

Align buffers to cache line size (typically 32 bytes) to avoid partial line issues.

5.4 Best Practices

Enable and Initialize Caches Properly

1. Ensure I-Cache and D-Cache are enabled early in system initialization for maximum performance.
2. Invalidate caches at startup to avoid stale data or instructions from previous runs.

Optimize Memory Regions

1. Mark cacheable regions correctly in the MPU (Memory Protection Unit) settings.
2. Avoid caching for memory-mapped peripheral registers or regions accessed by DMA to prevent coherency issues.

Handle Cache Coherency

1. When using DMA or peripherals, flush (clean) or invalidate D-Cache lines before and after transfers.
2. Use CMSIS cache maintenance APIs (e.g., SCB_CleanDCache_by_Addr, SCB_InvalidateDCache_by_Addr) for safe operations.

Leverage Locality

1. Design code and data structures to exploit temporal and spatial locality:
2. Keep frequently accessed data together.
3. Use tight loops and avoid scattered memory access patterns.

Minimize Cache Thrashing

1. Avoid large working sets that exceed cache size.
2. Align data structures to cache line boundaries (typically 32 bytes) to reduce conflicts.

Maintenance During Modifying Code for MRAM or Flash

1. If your application updates code in Flash or MRAM, invalidate I-Cache after modifications to ensure correct execution.

Use Write-Back Policy Carefully

1. D-Cache often uses write-back with write-allocate; ensure data is cleaned before sharing with peripherals or other cores.

6. Example Projects and Reference Applications

To showcase the use of cache in RA8 MCUs, this App Note provides two sample projects. In these projects, the basic usage of cache, its benefits, cache maintenance, and coherence are demonstrated.

6.1 Sample Code for Cache Implementation

The attached project "**ID_Cache_With_MRAM_SRAM_SDRAM.zip**" demonstrates the impact of enabling and disabling I-Cache and D-Cache on system performance. The project runs a fixed set of test code under different cache configurations and measures execution time to quantify performance gains.

Performance Improvement with Caches: When both I-Cache and D-Cache are enabled, the performance improves by approximately 10x compared to scenarios where caches are disabled. This significant boost is due to reduced memory access latency and better utilization of CPU cycles.

Whereas the attached project "**D_Cache_With_MRAM_SRAM_SDRAM.zip**" focuses on demonstrating the behavior of D-Cache in different scenarios involving CPU and DMA operations. It highlights how cache configuration and management affect data coherency when transferring data between MRAM, SRAM and SDRAM, and the CPU. **D_Cache_With_MRAM_SRAM_SDRAM** project covers four distinct use cases as shown below:

Use Case 1: D-Cache Disabled.

Steps:

1. CPU writes known data to a SRAM buffer.
2. DMA transfers a fixed set of data from MRAM to the same SRAM buffer.
3. CPU reads the latest data from the SRAM buffer.

Behavior: Since D-Cache is disabled, the CPU accesses SRAM directly. Data coherency is maintained because there is no cached copy; the CPU always reads the updated data written by DMA.

Use Case 2: D-Cache Enabled (No Invalidation).

Steps:

1. CPU writes known data to the SRAM buffer.
2. DMA transfers a fixed set of data from MRAM to the same SRAM buffer
3. CPU reads the data from the SRAM buffer.

Behavior: D-Cache is enabled, but cache invalidation is not performed. CPU reads stale data from its cache instead of updated data in SRAM. Data coherency is NOT maintained, leading to incorrect results.

Use Case 3: D-Cache Enabled with Cache Invalidation.

Steps:

1. CPU writes known data to the SRAM buffer.
2. DMA transfers a fixed set of data from MRAM to the same SRAM buffer.
3. Before the CPU reads, the cache is invalidated.

Behavior: D-Cache is enabled, but invalidation ensures the CPU fetches fresh data from SRAM.

Data coherency is maintained, and the CPU reads the correct data written by DMA.

Use Case 4: D-Cache Enabled with Non-Cacheable SRAM Region.

Steps:

1. SRAM buffer is placed in a non-cacheable memory region.
2. CPU writes known data to the SRAM buffer.
3. DMA transfers data from MRAM to the SRAM buffer.
4. CPU reads the data from the SRAM buffer.

Behavior: Even though D-Cache is enabled globally, the buffer resides in a non-cacheable region. CPU reads directly from SRAM, ensuring data coherency is maintained without explicit invalidation.

Key Takeaways: Data coherency issues arise when DMA updates memory while the CPU relies on cached data. Proper cache management techniques, such as cache invalidation or using non-cacheable regions, are essential in systems with DMA.

These scenarios are critical for embedded systems and SoCs where CPU and DMA frequently share memory.

7. Running the Application Projects

7.1 Import the Projects

1. Launch the e^2 studio IDE.
2. Select any workspace in Workspace Launcher.
3. Select **File > Import**.
4. Select **General >Existing Projects into Workspace** from the **Import** dialog box.
5. Select the archive file "**ID_Cache_With_MRAM_SRAM_SDRAM.zip**" from the downloaded folder
6. Select the solution project and developed project samples on each core as shown below, and click **Finish.**

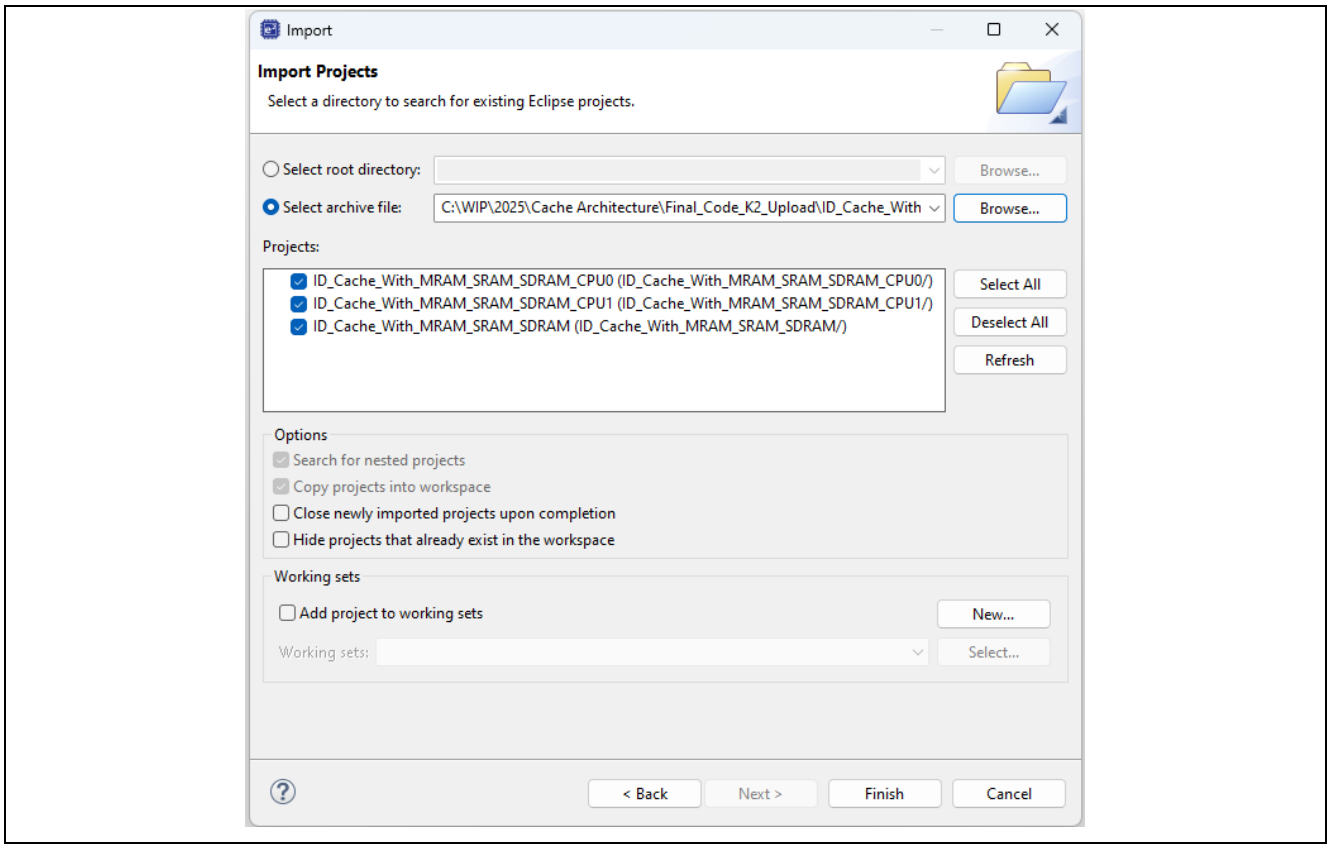

Figure 9. Importing Project.

7.2 Build Projects

Build the solution project "**ID_Cache_With_MRAM_SRAM_SDRAM** " from the imported bundle, as shown in the Figure 10. This will build the CPU0 project first, followed by the CPU1 project, as shown in the Figure 11.

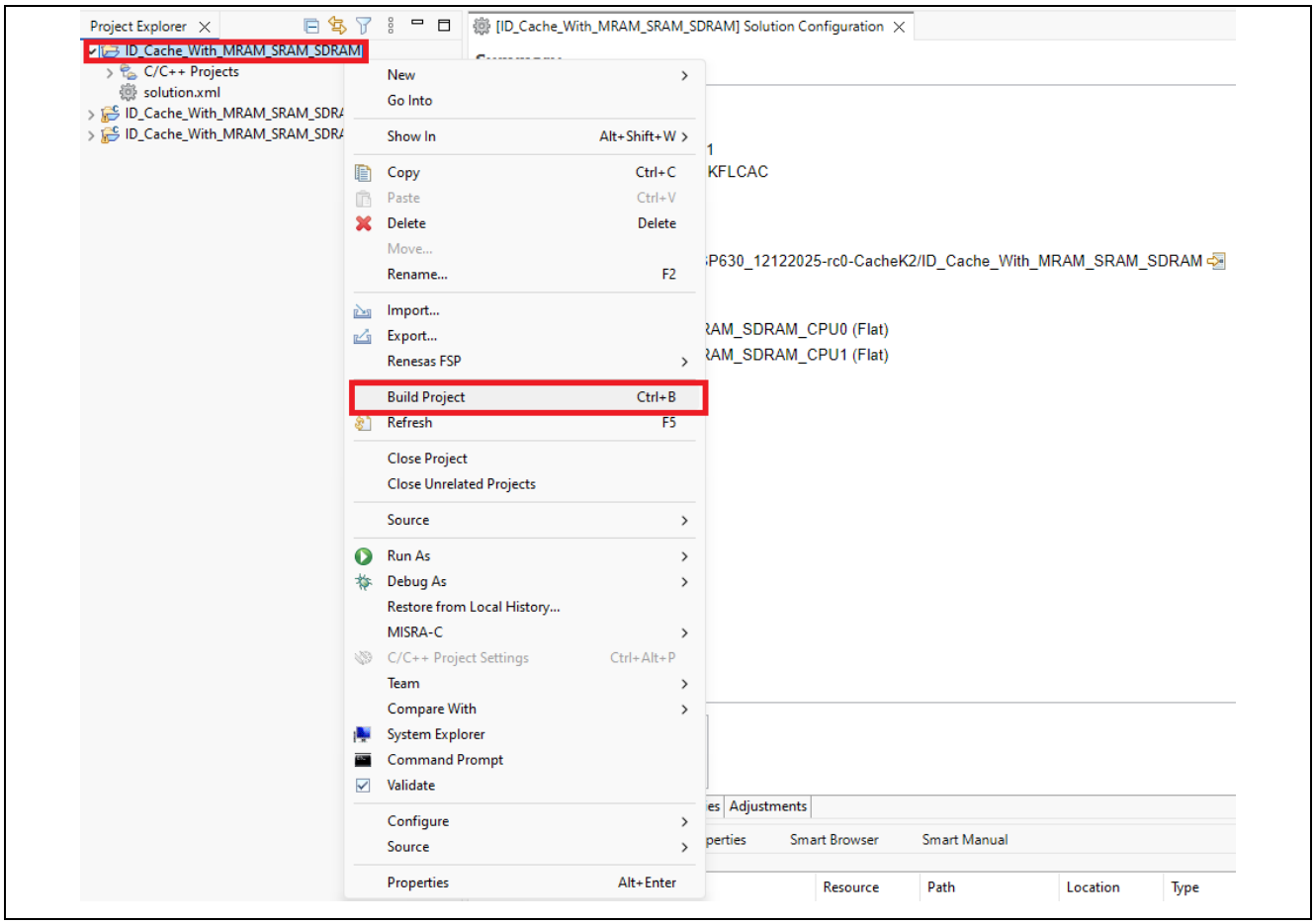

Figure 10. Example of Building Solution Project.

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

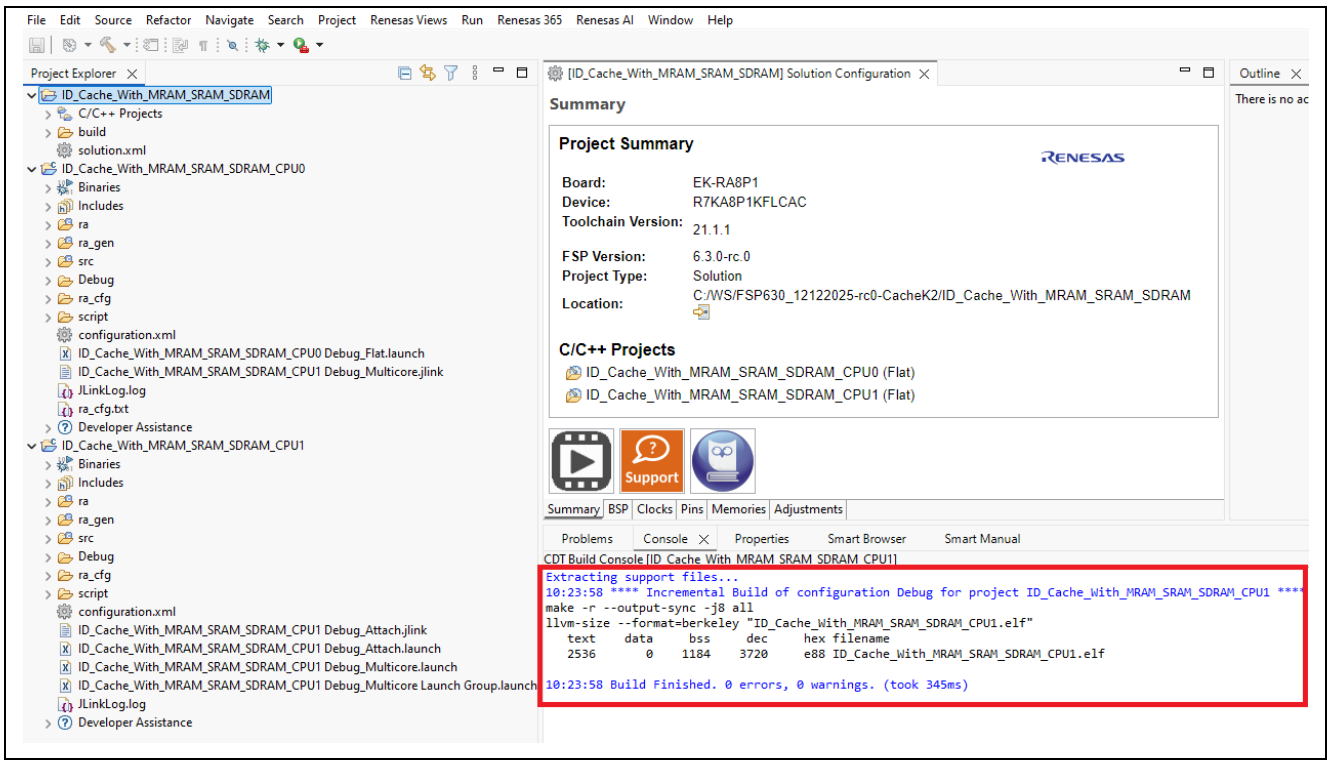

Figure 11. Example of the build log after the project is built.

7.3 Download, Run, and Verify the Projects

To start debugging both cores simultaneously:

1. Connect the USB-C cable to Debug connector(J10) and another end to host PC. This will be a J-Link connection for downloading the image and enables user interaction through the Segger J-Link RTT Viewer via a serial terminal.
2. Initially, the device must be initialized in the OEM_PL2 state, with no TrustZone boundary settings required.

   In the e² studio IDE, open the Renesas Device Partition Manager(Run→Renesas Debug Tools→Renesas Device Partition Manager).

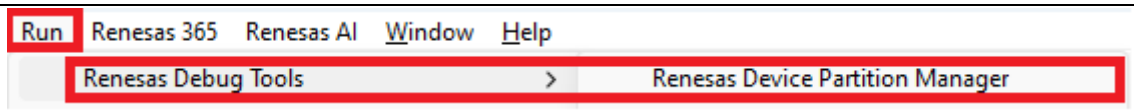

Figure 12. Open the Renesas Device Partition Manager

Note: To debug both cores simultaneously, begin by initializing your MCU using either the Renesas Flash Programmer or the Renesas Device Partition Manager. This step ensures that the device is set to Protection Level 2 and that the TrustZone boundary has not already been configured. If the boundary has been previously set, you must reset it to establish a suitable environment for debugging.

Completing this initialization is essential; without it, you may encounter issues when downloading project images or starting the debug session.

Figure 13 shows how to initialize the device back to factory default. Choose the connection method and then click Run.

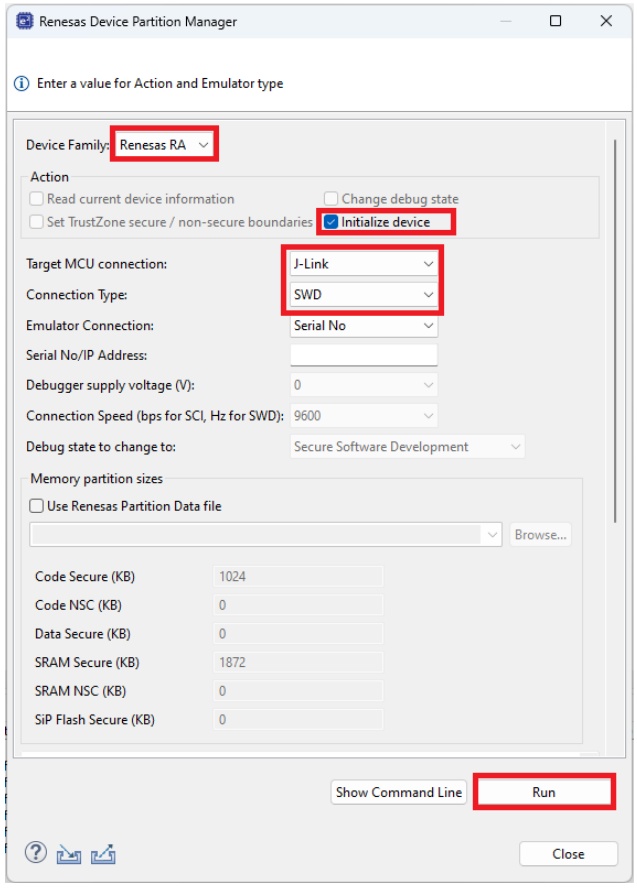

Figure 13. Initialize RA MCU Using Renesas Device Partition Manager

3. Open Debug Configurations as shown in Figure 14.
4. Select “ID_Cache_With_MRAM_SRAM_SDRAM_CPU1 Debug_Multicore Launch Group” as shown in Figure 15.
5. Click Debug to launch the dual-core debug session.

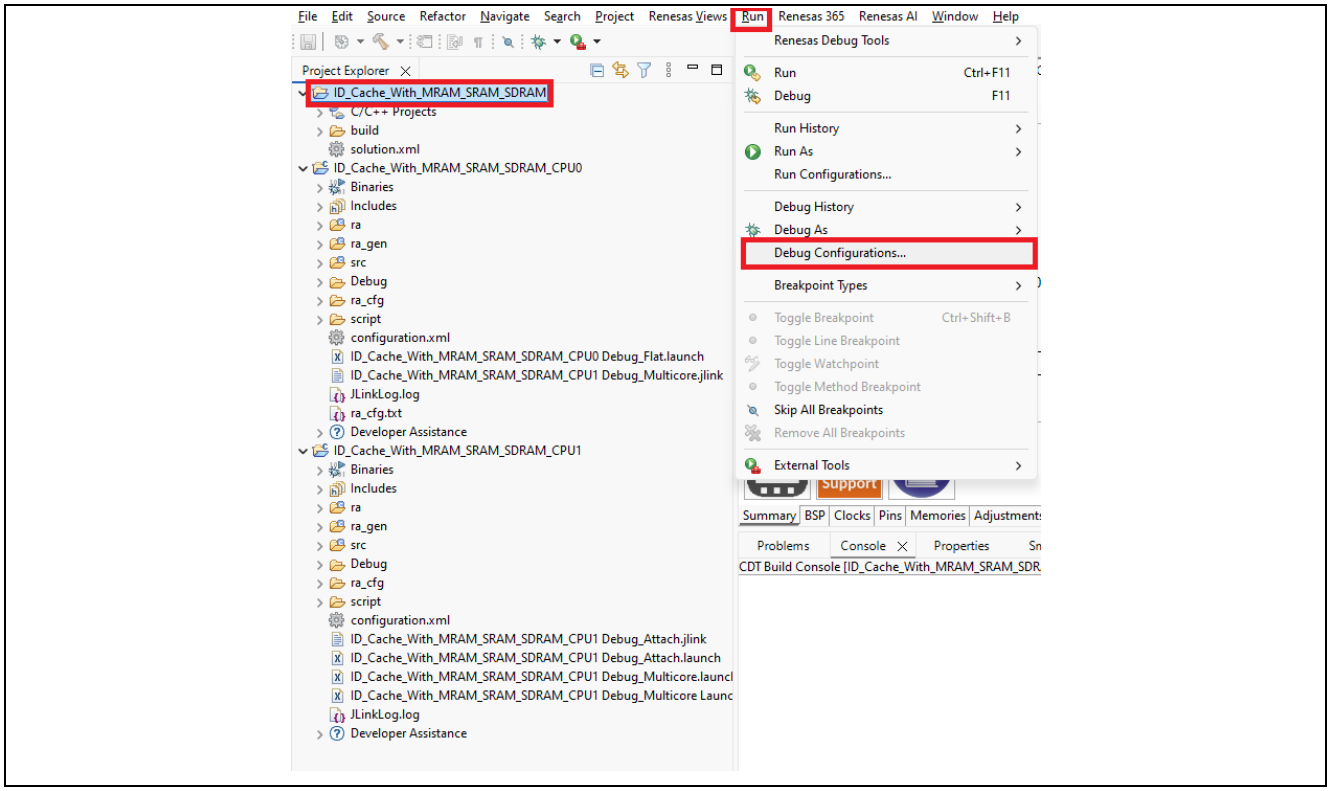

Figure 14. Open Debug Configuration

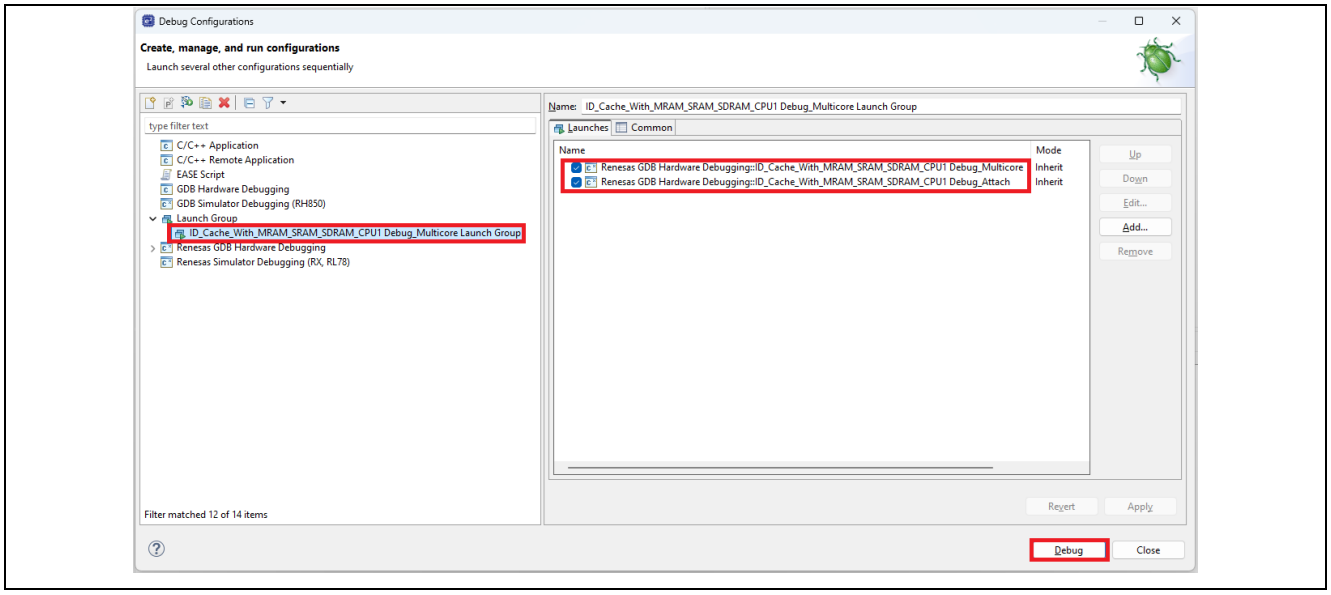

Figure 15. Initiating Debug in the solution project

7.4 Verifying the Application

7.4.1 Launching and connecting to RTT Viewer

Open the J-Link RTT viewer (File→ connect) and connect with the following settings: Connection to J-Link: USB, Specify Target Device: R7KA8P1KF_CPU0 (For – RA8P1), Select Force go on connect, Target Interface & Speed: JTAG 4000Khz, JTAG scan chain information: Auto detection, RTT Control Block: Address, search “_SEGGER_RTT” and update the correct address from the .map file and click Ok.

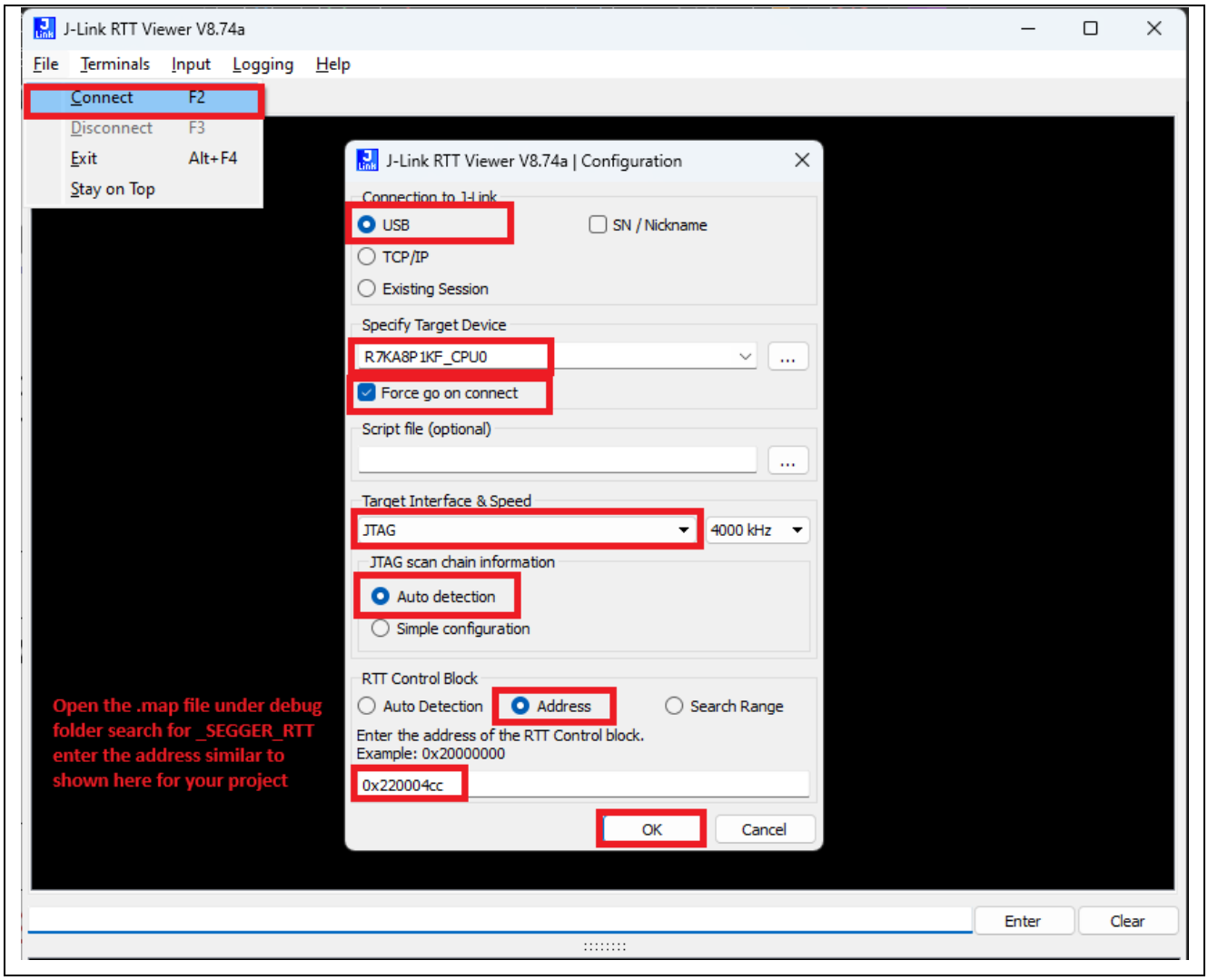

Figure 16. Example of Opening RTT Viewer and Connecting

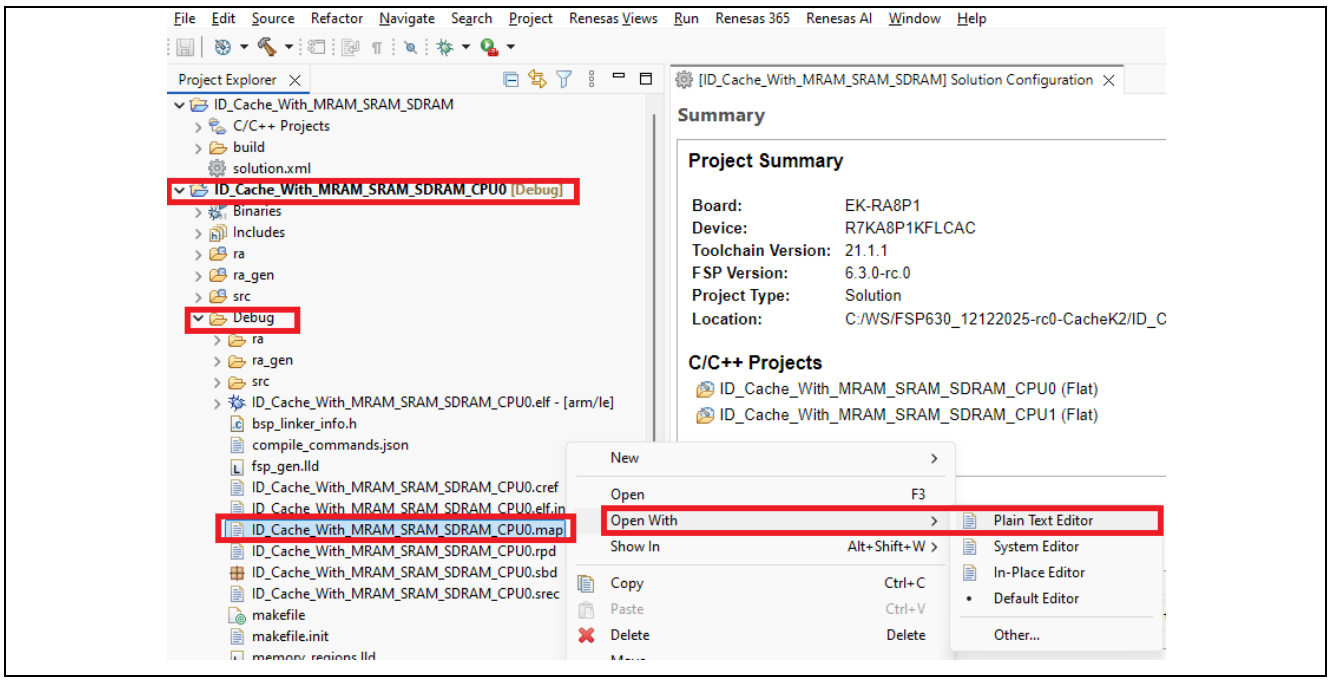

Figure 17. Obtaining RTT Control Block Address

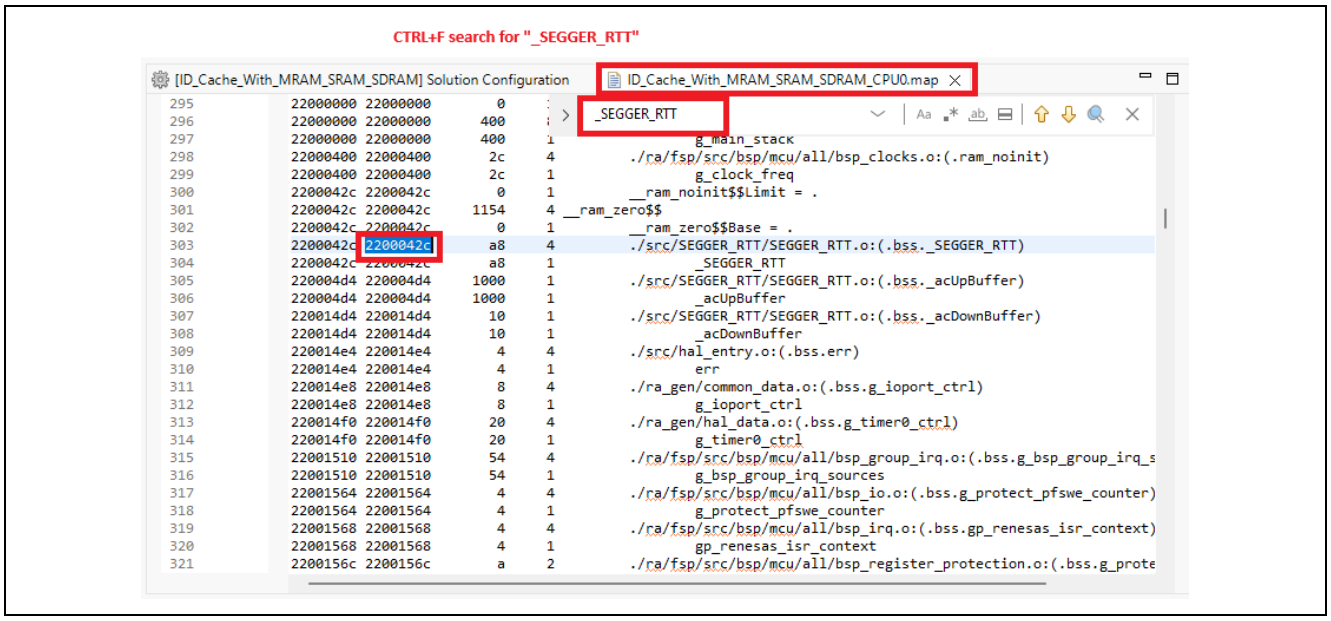

Figure 18. Search for the _SEGGET_RTT Control Block address

7.4.1 Running and verifying the Application

Return to e² studio and press Resume three times to execute the application across both cores. Upon completion, the sample project example “ID_Cache_With_MRAM_SRAM_SDRAM” will output its status to the terminal, as illustrated in Figure 19.

Note: Users can observe the LED1 (BLUE) and LED2 (GREEN) blinking as they are being controlled by CPU0 and CPU1. This also indicates that both cores are running during this execution.

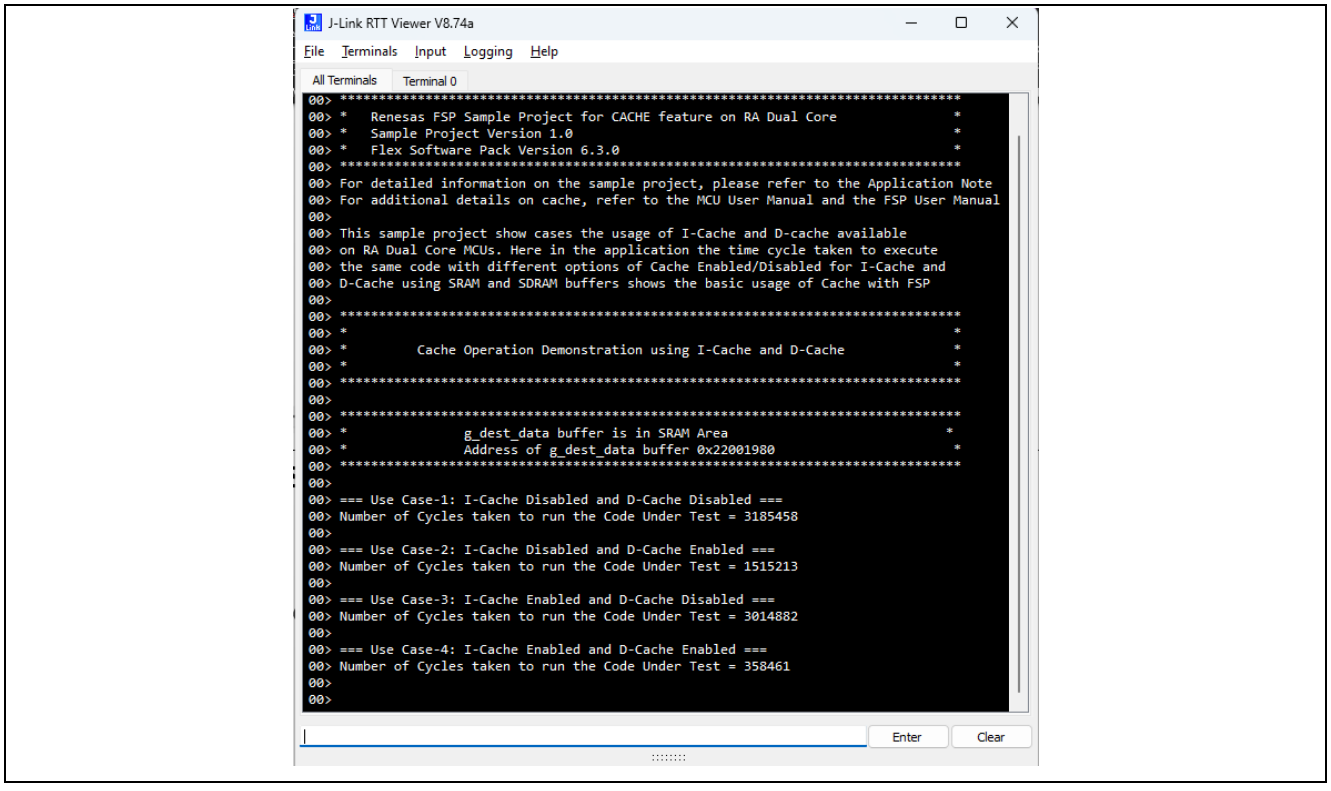

Figure 19. Console Log of Sample Project with SRAM Buffer

Note: The Sample project “ID_Cache_With_MRAM_SRAM_SDRAM” can be modified to run with an SDRAM buffer. To do this, open the “common_utils.h” and modify ***#define*** *MEMORY_REGION* ***SRAM*** → ***#define*** *MEMORY_REGION* ***SDRAM***. Now rebuild the project and run and validate the behavior of the cache while

the buffer is placed in SDRAM. You can see the difference between SRAM vs SDRAM buffer timings as shown in Figure 20.

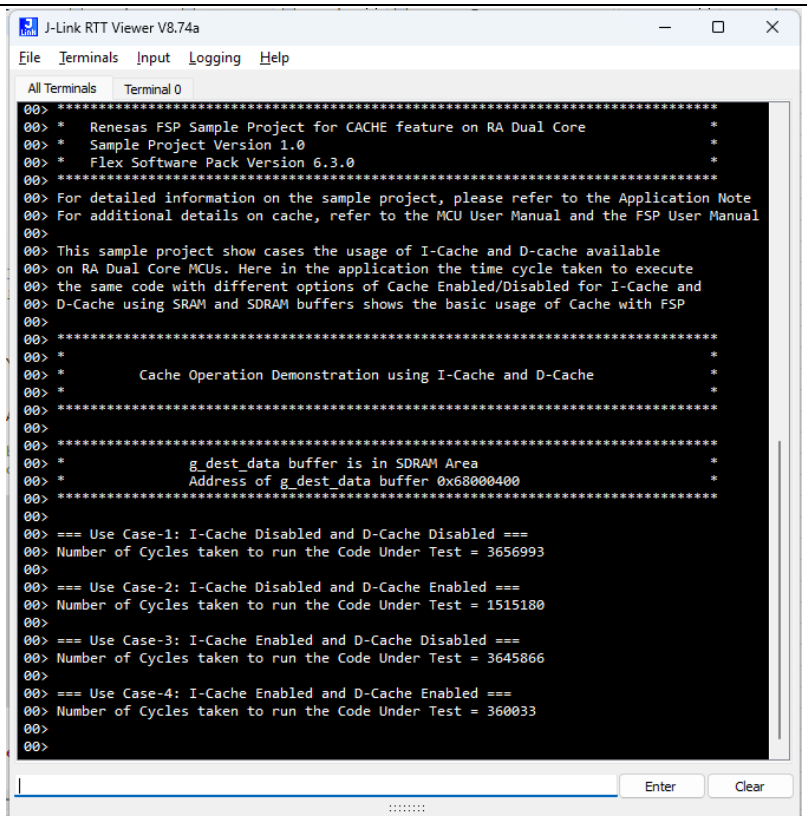

Figure 20. Console Log of Sample Project with SDRAM Buffer

7.5 Verifying the Cache Maintenance Application

Users can follow the above-mentioned steps in the section 7.1 and 7.2 to import, build, and section 7.3 to download and run the second project “D_Cache_With_MRAM_SRAM_SDRAM.zip” from the bundle.

The following sections provide a snapshot of the second sample project along with its use cases and expected outcomes.

D-cache behavior must be clearly defined when analyzing or reproducing system behavior, because cache write policy directly affects memory visibility and test results.

Force Write-Through changes system behavior. In write-through mode, every store is immediately written to main memory, so data becomes visible right away to other agents such as DMA engines. This can lead to different outcomes compared to normal operation, especially in systems where data is typically buffered in the cache.

Write-back mode keeps modified data in the D-cache until it is explicitly cleaned or evicted, which matches performance-optimized configurations used in practice. Testing in write-back mode helps expose cache-related issues and ensures results accurately reflect the behavior of the actual system.

1. Use Case 1: D-Cache Disabled
In this Use case scenario: D-Cache Disabled, Buffer: g_dest_data located in SRAM at address 0x22000020.

Steps:

1. Initialization: Buffer cleared with memset() → all values are set to 0.
2. CPU Update: CPU writes AA to all buffer locations → CPU reads to see if buffer shows contents as AA.

3. DMA Transfer: DMA writes a new data pattern (0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F) to the same buffer.
4. CPU Read: CPU reads updated data directly from SRAM after DMA transfer completion.

Key Behavior:

1. With D-Cache disabled, the CPU always accesses main memory directly.
2. Data coherency is maintained without any cache maintenance operations.
3. Final Buffer Contents: 0 1 2 3 4 5 6 7 8 9 A B C D E F

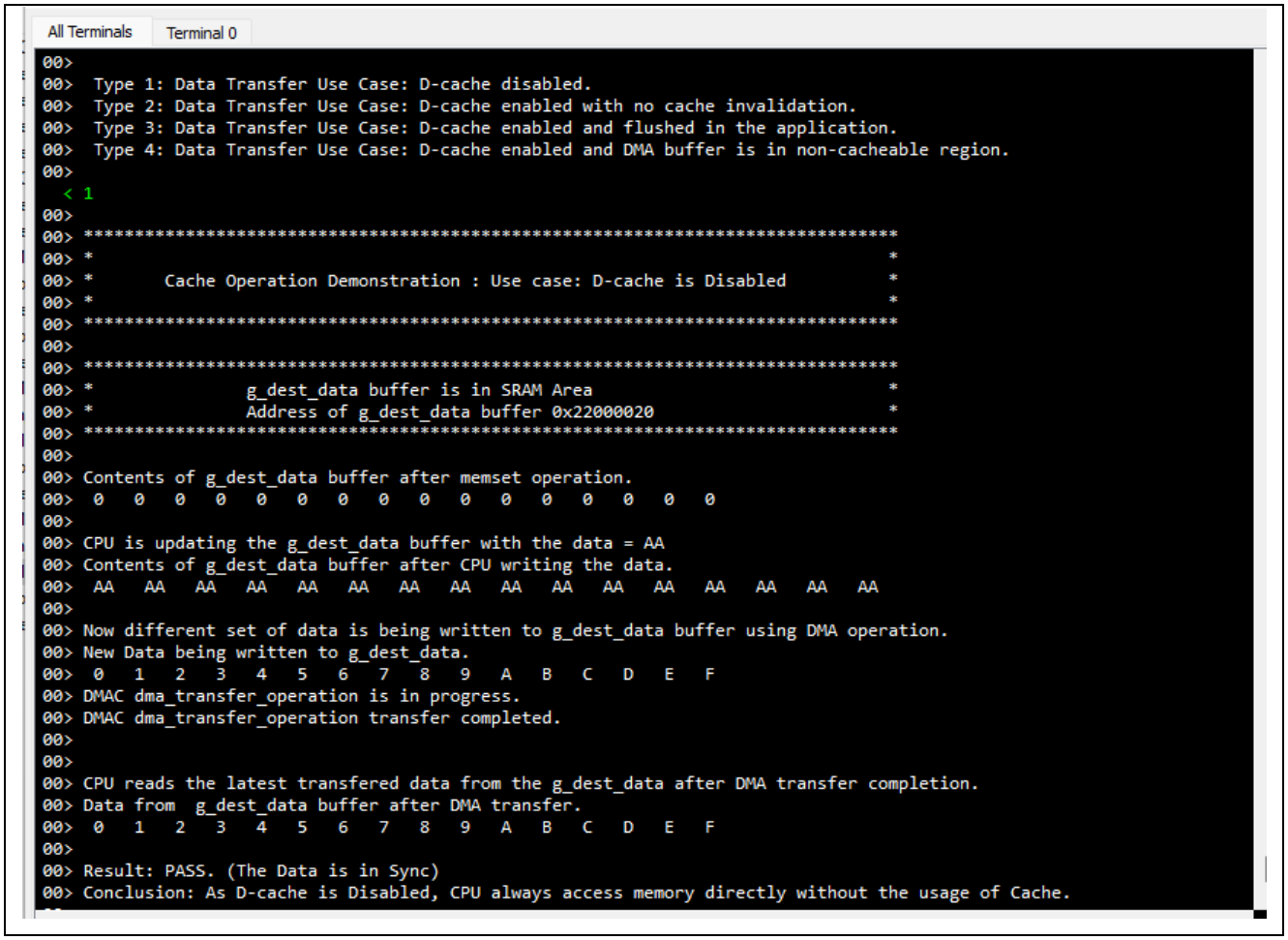

Figure 21. Console Log of Application for Cache behavior: D-cache Disabled

2. Use Case 2: D-Cache Enabled (No Invalidation).
In this Use case scenario: D-Cache Enabled, No Cache Flush. Buffer: g_dest_data located in SRAM at address 0x22000020.

Steps:

1. Initialization: Buffer cleared with memset()→ all values are set to 0.
2. CPU Update: CPU writes AA to all buffer locations → CPU reads to see if buffer shows contents as AA.
3. DMA Transfer: DMA writes a new data pattern (0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F) to the same buffer.
4. CPU Read: CPU reads stale data from D-Cache instead of updated memory content.

Key Behavior:

1. The cache was not invalidated before the CPU read in step 4, so the CPU displays outdated values.

2. Data coherency is NOT maintained. D-Cache and SRAM are out of sync.
3. Result: Incorrect data read by CPU after DMA transfer.

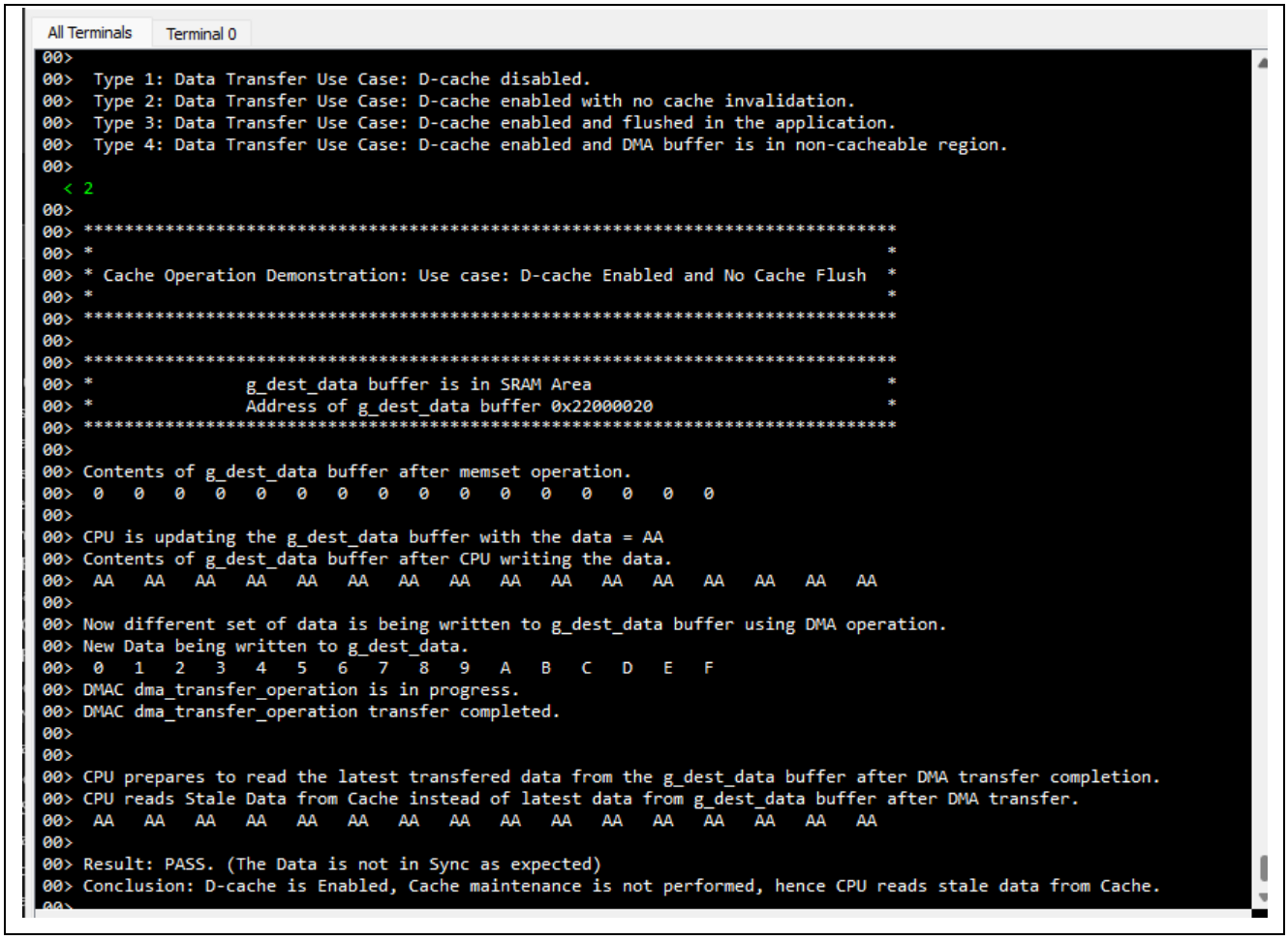

Figure 22. Console Log: D-cache Enabled and no invalidation

3. Use Case 3: D-Cache Enabled with Cache Invalidation

In this Use case scenario: D-Cache enabled, cache flush (invalidation) used. Buffer: g_dest_data located in SRAM at address 0x22000020.

Steps:

1. Initialization: Buffer cleared with memset() → all values are set to 0.
2. CPU Update: CPU writes AA to all buffer locations → CPU reads to see if buffer shows data as AA.
3. DMA Transfer: DMA writes a new data pattern (0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F) to the same buffer.
4. Cache Maintenance: Cache invalidation is performed before the CPU reads.
5. CPU Read: CPU fetches updated data from memory, not stale cache.

Key Behavior:

1. Cache invalidation ensures data coherency between CPU and DMA.
2. Final buffer contents after DMA: 0 1 2 3 4 5 6 7 8 9 A B C D E F.

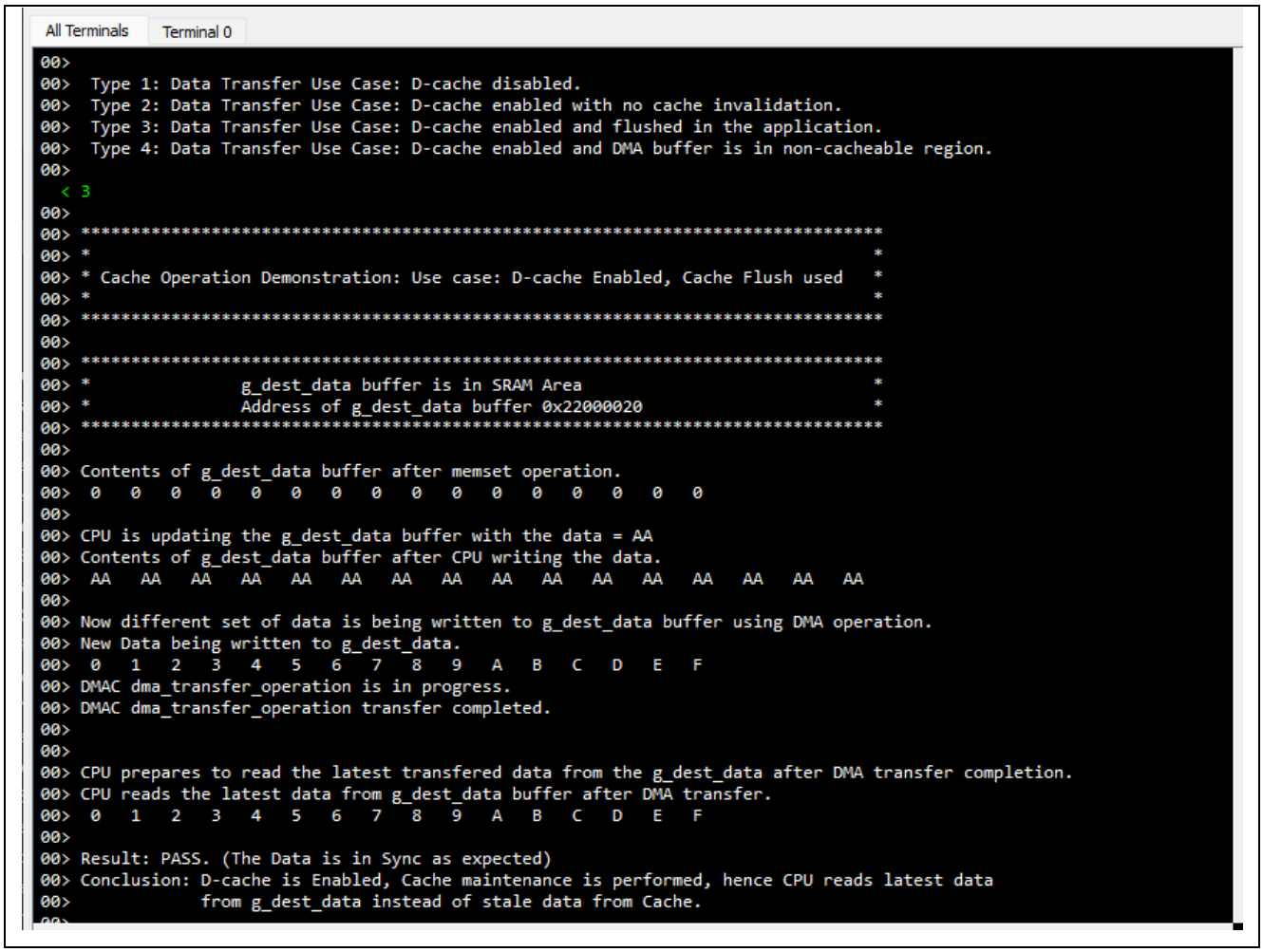

Figure 23. Console Log: D-cache Enabled and Cache Flush used

4. Use Case 4: D-Cache Enabled with Non-Cacheable SRAM Region

In this Use case scenario, D-Cache is enabled, but the buffer (g_dest_data) is placed in a non-cacheable SRAM region at address 0x22000020.

Steps:

1. Initialization: Buffer is cleared using memset() → all values are set to 0.
2. CPU Update: CPU writes AA to all buffer locations → CPU reads to see if buffer shows data as AA.
3. DMA Transfer: DMA writes a new data pattern (0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F) to the same buffer.
4. CPU Read: CPU reads the updated data after the DMA transfer.

Key Behavior:

1. Because the buffer is non-cacheable, caching is not applied to the buffer memory in CPU accesses.
2. Data coherency is maintained: CPU reads the latest data written by DMA.
3. Final buffer contents after DMA: 0 1 2 3 4 5 6 7 8 9 A B C D E F

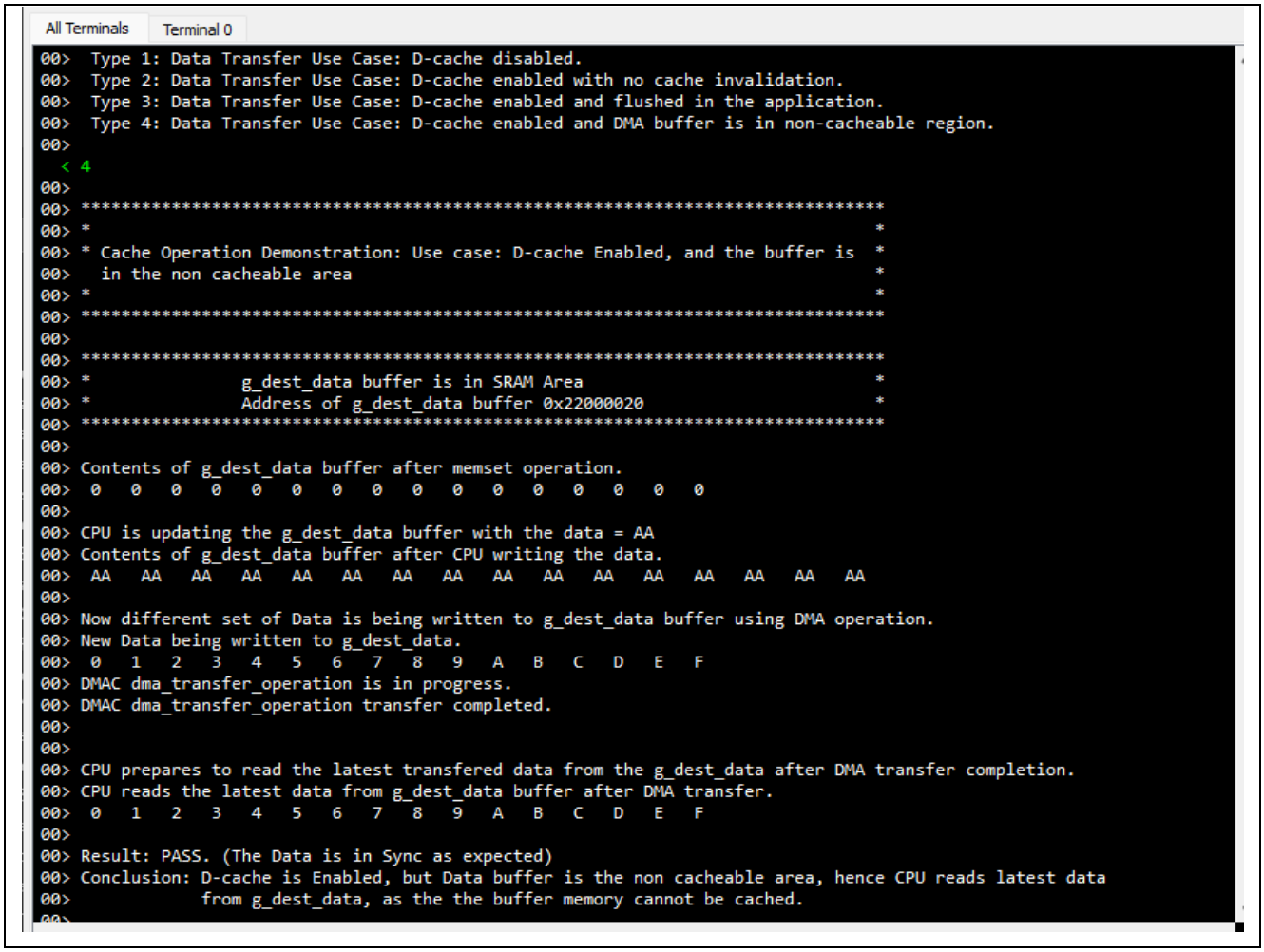

Figure 24. Console Log: D-cache Enabled and buffer is non-cacheable

Note: The Sample project “D_Cache_With_MRAM_SRAM_SDRAM” can be modified to run with an SDRAM buffer. To do this, open the “common_utils.h” and modify ***#define*** *MEMORY_REGION* ***SRAM*** → ***#define*** *MEMORY_REGION* ***SDRAM***. Now rebuild the project and run and validate the behavior of the cache while the buffer is placed in SDRAM.

8. Debugging and Troubleshooting

- To debug both cores simultaneously, it is required to begin by initializing your MCU using either the Renesas Flash Programmer or the Renesas Device Partition Manager. This step ensures that the device is set to Protection Level 2 and that the TrustZone boundary has not already been configured. If the boundary has been previously set, you must reset it to establish a suitable environment for debugging.

  Completing this initialization is essential; without it, you may encounter issues when downloading project images or starting the debug session.

- Also, using the Launch group, which combines individual launch configurations, helps to run and debug the dual-core projects.

- Additionally, make sure “R_BSP_SecondaryCoreStart()” gets called from the CPU0 project to run the CPU1 core. For debugging purposes, add a breakpoint to make sure this code is invoked to confirm the CPU1 code gets invoked.

9. Next Steps

• To learn more about the EK-RA8P1 kit, refer to the EK-RA8P1 user’s manual and design package available in the Documents and Download tabs, respectively, of the EK-RA8P1 webpage at renesas.com/ek-ra8p1.

• To learn how to create a new e² studio project from scratch, refer to Chapter 2, Starting Development, in the FSP User Manual (renesas.com/ra/fsp). To learn how to use e² studio, refer to the user manual provided on the e² studio webpage (renesas.com/software-tool/e-studio).

• Renesas provides several example projects that demonstrate different capabilities of the RA and the MCUs. These example projects can serve as a good starting point for users to develop custom applications.

10. References

A collection of important references, datasheets, and documentation related to dual-core memory architecture and system design.

The following documents can be referred to for a more detailed understanding of RA8P1

- RA8P1 Group User's Manual: Hardware, document No. R01UH1064EJ
- Renesas FSP User's Manual: https://renesas.github.io/fsp
- Getting Started with RA8 Memory Architecture, Configurations, and Topologies, document No. R01AN7880EU
- Arm Cortex-M85 Processor Technical Reference Manual document No. 101924_0002_05_en (https://developer.arm.com/documentation/101924/0101)
- RA8P1 Multicore setup and running Hello World on dual core, document No. R01AN7982EU
- RA8P1 Developing with RA8 Dual Core MCU, document No. R01AN7881EU
- RA8P1 MCU Quick Design Guide, document No. R01AN7883EU
- Arm Cortex®-M85 Processor Technical Reference Manual, document No. 101924, available from Arm.
- For More details on Cortex-M85 Cache Please refer to the FSP Documentation Section: RA Flexible Software Package Documentation: Cortex-M85 Caches.
- For the usage of S-cache and C-cache, the following App note can be referred: RA Family Guidelines for Using the S Cache on the System Bus (R11AN0538EU0130)

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 Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.02.26	-	Initial version
		-	

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 reaches the level at which resetting 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 system-evaluation test for the given product.

Notice

1. 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.
2. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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/.