Renesas RA Family

Guidelines for Using the S Cache on the System Bus

Introduction

Caches can effectively improve instruction or data access speed for microcontroller and microprocessor systems with mismatch between CPU and slower SRAM. Even though there are no internal caches in the Arm® Cortex®-M23 and Cortex-M33 processors, for some Renesas RA Family Cortex-M33 MCUs, there are system level caches for both instruction cache and data cache present, which help to improve instruction and data fetch speed.

The cache enabling and configuration for the instruction cache are handled by the Renesas Flexible Software Package (FSP). The cache enabling, disabling, and flushing functionality for the data cache are demonstrated in this application project with reference software projects provided. In addition, this application project provides guidelines and example code for keeping the data cache coherent. Use this application project as a reference resource for S Cache operations.

The data cache is named S Cache in the Renesas RA Family Cortex-M33 MCU Hardware User's Manual. The S Cache is on the MCU's system bus. The instruction cache is named C Cache and is on the code bus. This application note is focused on the data cache usage of the RA MCUs. For consistency, this application note uses **S Cache** throughout the rest of the application note. At the time of the release of this application project, the RA Family MCU groups that support the S Cache are RA6M5, RA6M4, RA6E1, RA6T2, and RA4M3. The user can review the MCU Hardware User's Manual "Buses" section and look for the Cache section to understand whether any new MCUs include S Cache and its general specifications.

For other RA6 Series MCUs which do not have S Cache, they are provided with SRAMHS. Access to the SRAMHS is always no wait state. Use the SRAMHS on these MCUs when improved SRAM access is needed.

The example project provided is based on EK-RA6M5. You can easily port the example project to other MCUs which support S Cache. The performance improvement of using S Cache on an MCU varies based on the MCUs memory access speed, memory size, the nature of the SRAM access pattern of the application code. The user needs to analyze all these aspects when evaluating the S Cache.

Required Resources

Development tools and software

- The e^2 studio ISDE v2023-01
- Renesas Flexible Software Package (FSP) v4.3.0
- SEGGER J-link® USB driver

The above three software components: the FSP, J-Link USB drivers and e^2 studio are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

Hardware

- EK-RA6M5 Evaluation Kit for RA6M5 MCU Group (http://www.renesas.com/ra/ek-ra6m5)
- Workstation running Windows® 10
- One USB device cables (type-A male to micro-B male)

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas e^2 studio IDE development. You must be familiar with importing, building, and debugging a Renesas RA Family MCU project based on FSP packages. In addition, users are required to read the entire Hardware User's Manual Caches section prior to proceeding to the rest of this application note:

The intended audience is product developers who wish to use the S Cache feature to improve the system performance.

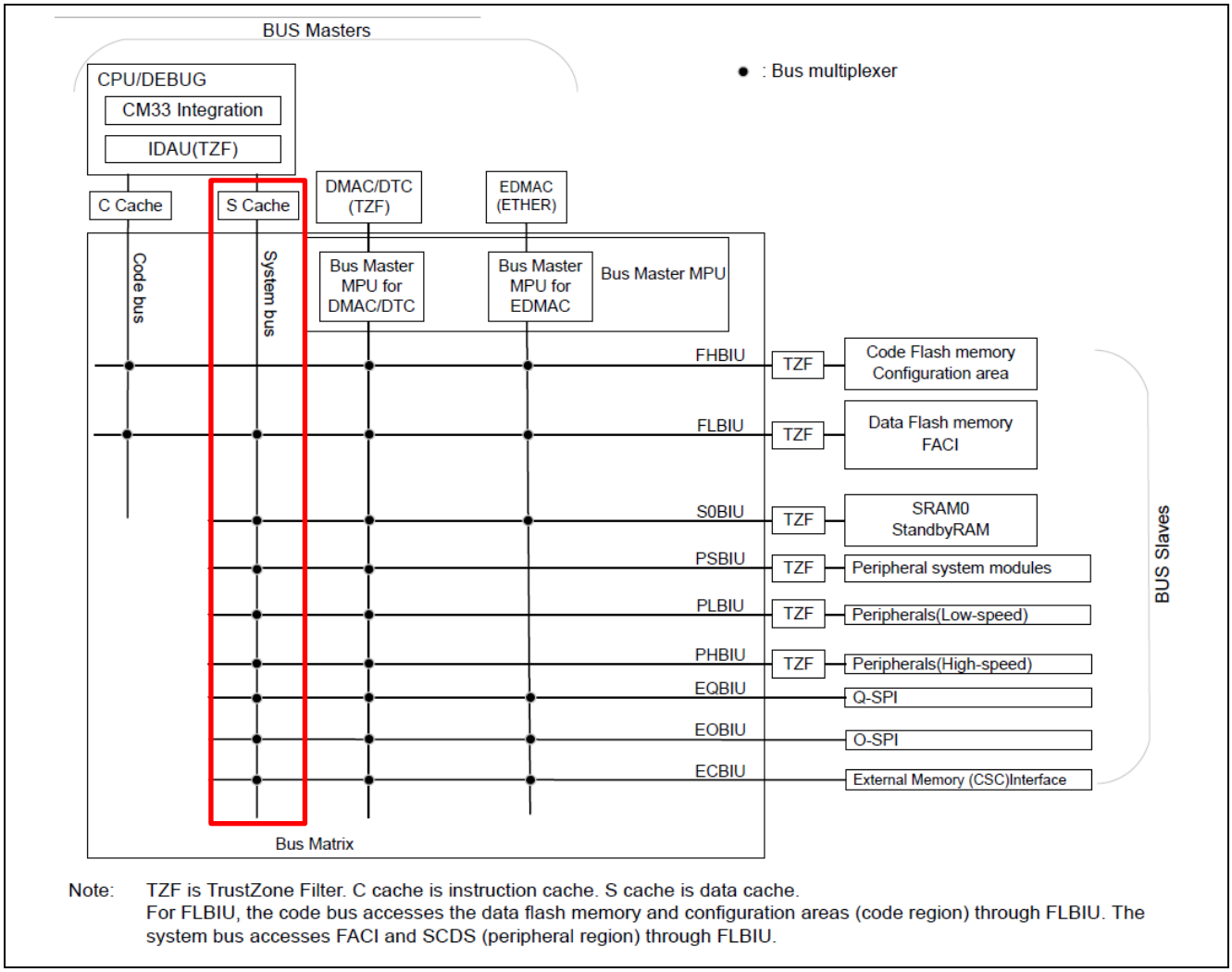
Contents

1. Overview of the S Cache on the System Bus

A cache is a smaller, faster memory, located closer to a processor core than main memory. It stores copies of the data from frequently used main memory locations. Some RA Family Arm® Cortex®-M33 MCUs implement both C Cache on the Code Bus and S Cache on the System Bus to reduce the average cost (time or energy) to access data from the main memory.

1.1 S Cache Architecture

Read the **Buses** > **Overview** section in the Renesas RA Family Cortex-M33 MCU Hardware User's Manual to see whether the MCU supports S Cache and understand the S Cache architecture. The bus system architecture for RA Family Cortex-M33 MCUs that have S Cache is shown in the following graphic.



Note: TZF is TrustZone Filter. C cache is instruction cache. S cache is data cache.
For FLBIU, the code bus accesses the data flash memory and configuration areas (code region) through FLBIU. The system bus accesses FACI and SCDS (peripheral region) through FLBIU.

Figure 1. Bus Architecture for RA6M4 and RA6M5

The bus architecture for RA6E1, RA6T2, and RA4M3 is similar to RA6M4 and RA6M5 regarding the S Cache operation, however, these devices do not have external memory interface like QSPI and OSPI.

Table 1 is the bus master specification for the RA6M4 and RA6M5 MCUs with S Cache. For arbitration between masters, the analysis in this application note is based on the following priority sequence:

EDMAC > **DMAC/DTC** > **CPU**

Table 1. Bus Specification for RA6M4 and RA6M5 MCUs with S Cache Support

Bus Master Name	Bus Interface Maximum Frequency	Synchronization	Specifications
Code bus	200 MHz	ICLK	Connected to the CPU Instruction Cache (C Cache) for instructions and operands
System bus	200 MHz	ICLK	Connected to the CPU Data Cache (S Cache) for data access operations
DMAC/DTC	200 MHz	ICLK	Connected to the DMAC/DTC
EMAC (Ether)	100 MHz	PCLKA	Connected to the EDMAC

Note that other MCUs with S Cache support may have different ICLK and DMAC clock configurations. In addition, some MCUs with S Cache support may not include Ethernet support, the user is required to reference the specific MCU Hardware User's Manual to understand the specific configurations.

1.2 S Cache Specifications

Read the **Buses** > **Caches** > **Overview** section in the Renesas RA Family Arm® Cortex®-M33 MCU Hardware User's Manual to understand the S Cache specifications. The following table has a summary of the key features for RA6M4 and RA6M5. Other MCUs with S Cache may have different capacity, number of entries, and so forth. The user is required to reference the specific MCU Hardware User's Manual to understand the specific configurations.

Table 2. S Cache Specifications for RA6M4 and RA6M5

Parameter	S Cache
Capacity	2 KB
Way	2-way set associative
Line size	32/64 bytes (defaults to 32 bytes)
Number of entries	32/16 entry/way (defaults to 32 entry per way)
Write way	Write through, non-write allocate
Replace way	2 way: LRU (Least recently used)
S Cache support area	0x20000000-0xDFFFFFFF except Standby SRAM area (0x2800_0000 to 0x2FFF_FFFF). Note: Peripheral area 0x4000_0000 to 0x5FFF_FFFF and QSPI I/O register area 0x6400_0000 to 0x67FF_FFFF must not have the cacheable attribution in the Arm® MPU.

Use caution when updating the Memory Protection Unit (MPU) configurations to avoid accidentally making these sections cacheable. Note that based on the Cortex-M33 default memory map, RA6 Standby SRAM region is also cacheable. Renesas RA6 MCUs with S Cache control have chosen a different configuration in this area and made this section as non-cacheable. This is controlled by hardware; user does not need to set the Standby SRAM area as non-cacheable.

In addition, the Quad Serial Peripheral Interface (QSPI) registers of the RA6 MCUs with S Cache are located in the Normal memory region based on the Cortex-M33 default memory map as shown in Figure 2 and Figure 3 . User needs to use the Arm Memory Protection Unit (MPU) to set this area as Non-cacheable. Also, if the Cortex-M33 default memory map is used, the peripheral area memory type is Device nGnRE, the cache attribute is not available for this area. As such, there is no action needed to additionally set this area as Non-cacheable. Example code is provided in section 2.4.2 Figure 5 to set the QSPI register area as Non-cacheable.

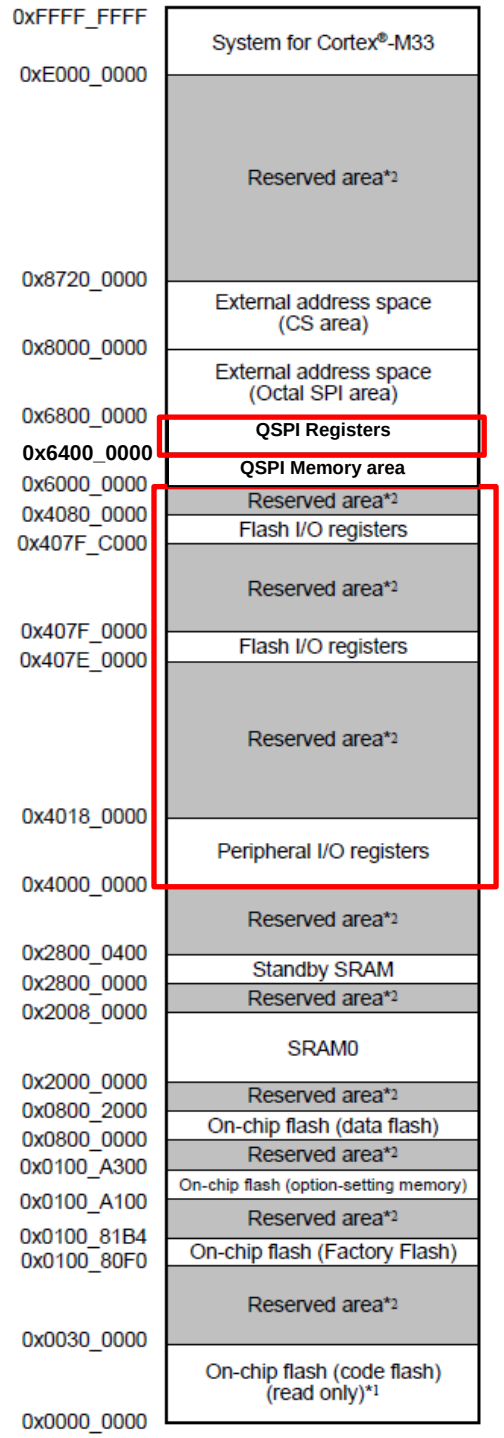
2.2.5 Behavior of memory accesses

Summary of the behavior of accesses to each region in the memory map.

Table 2-16 Memory access behavior

Address range	Memory region	Memory type	Shareability	XN	Description
0x00000000-0x1FFFFFFF	Code	Normal	Non-shareable	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal	Non-shareable	-	Executable region for data. You can also put code here.
0x40000000-0x5FFFFFFF	Peripheral	Device, nGnRE	Shareable	XN	On-chip device memory.
0x60000000-0x9FFFFFFF	RAM	Normal	Non-shareable	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device, nGnRE	Shareable	XN	External device memory.
0xE0000000-0xE003FFFF	Private Peripheral Bus	Device, nGnRnE	Shareable	XN	This region includes the SCS, NVIC, MPU, SAU, BPU, ITM, and DWT registers.
0xE0040000-0xE0043FFF	Device	Device, nGnRnE	Shareable	XN	This region is for debug components. Contact your implementer for more information.
0xE0044000-0xE00FFFFF	Private Peripheral Bus	Device, nGnRnE	Shareable	XN	This region includes the ROM tables.
0xE0100000-0xFFFFFFFF	Vendor_SYS	Device, nGnRE	Shareable	XN	Vendor specific.

Figure 2. Arm® Cortex®-33 Default Memory Map



Note 1. See Table 4.1. The capacity of the flash differs depending on the product.
Note 2. Do not access reserved areas.

Figure 3. Memory Areas that Need to be Non-Cacheable

1.3 Defining the Memory Attribute using the Memory Protection Unit

The RA6 and RA4 MCU groups which have the S Cache support also includes the optional Arm Memory Protection Unit (MPU). The MPU is a programmable peripheral that can define memory access permissions, such as privileged access only, and memory attributes, for example Cacheability, for different memory regions.

When S Cache is enabled, whether a memory region is cacheable depends on the MPU configuration. The MPU is programmable and the configuration of the MPU regions is managed by several memory mapped MPU registers. The MPU can be used to protect memory regions by defining access permissions.

Although the Arm® Cortex®-M23 and Cortex-M33 processors do not have an internal level 1 cache, the cache attributes produced by the MPU settings are exported to the processor's top level. The RA Family MCU S Cache can utilize this feature to vary the cacheable setting for the SRAM regions. For example, for any algorithms where the variables need to be updated and flushed very frequently by multiple bus masters, using the MPU to configure these areas as non-cacheable may benefit the system.

When enabled, the MPU can override Cortex-M33's the default memory access behavior. The attributes and permissions of all regions, except that targeting the NVIC and debug components, can be modified using an implemented MPU.

The user can set up the MPU to define additional memory regions as non-cacheable. Section 2.4.2 explains the use case of using the Arm MPU to achieve S Cache coherency.

The example project demonstrates how setting the SRAM region that is used by the DMA and CPU as non-cacheable can avoid the cache coherency issue. User can reference section 3 for the details.

1.4 S Cache Operation

Read the **Buses** > **Caches** > **Operation** section in the Hardware User's Manual to understand the access flow from CPU to S cache. Once the S cache is enabled, access to the cacheable area follows the access flow as shown in Figure 4.

The S cache function works when it is enabled, and cacheable access is performed from the CPU. When an SRAM access to the cacheable area is initiated, the cache first checks the address of CPU access request and compares the address with the entries in the cache tag. Then based on this, the CPU determines whether the CPU access is a hit or a miss.

If the access is a read, the system behavior varies according to the following rules:

- For a read hit, the cache reads required data from the cache data and returns it to CPU. In a cache read hit, there is a 0 bus wait cycle.
- For a read miss, the cache reads one cache line data from memory and stores it into the cache data. The cache then returns the required data. In cache read miss, the number of bus cycles used is same as when cache is disabled.

If the access is a write, the system behavior varies according to the following rules:

- For a write hit, the cache processes a write cycle to cache data and a write cycle to memory.
- For a write miss, the cache processes a write cycle to memory. There is no impact on cache data.

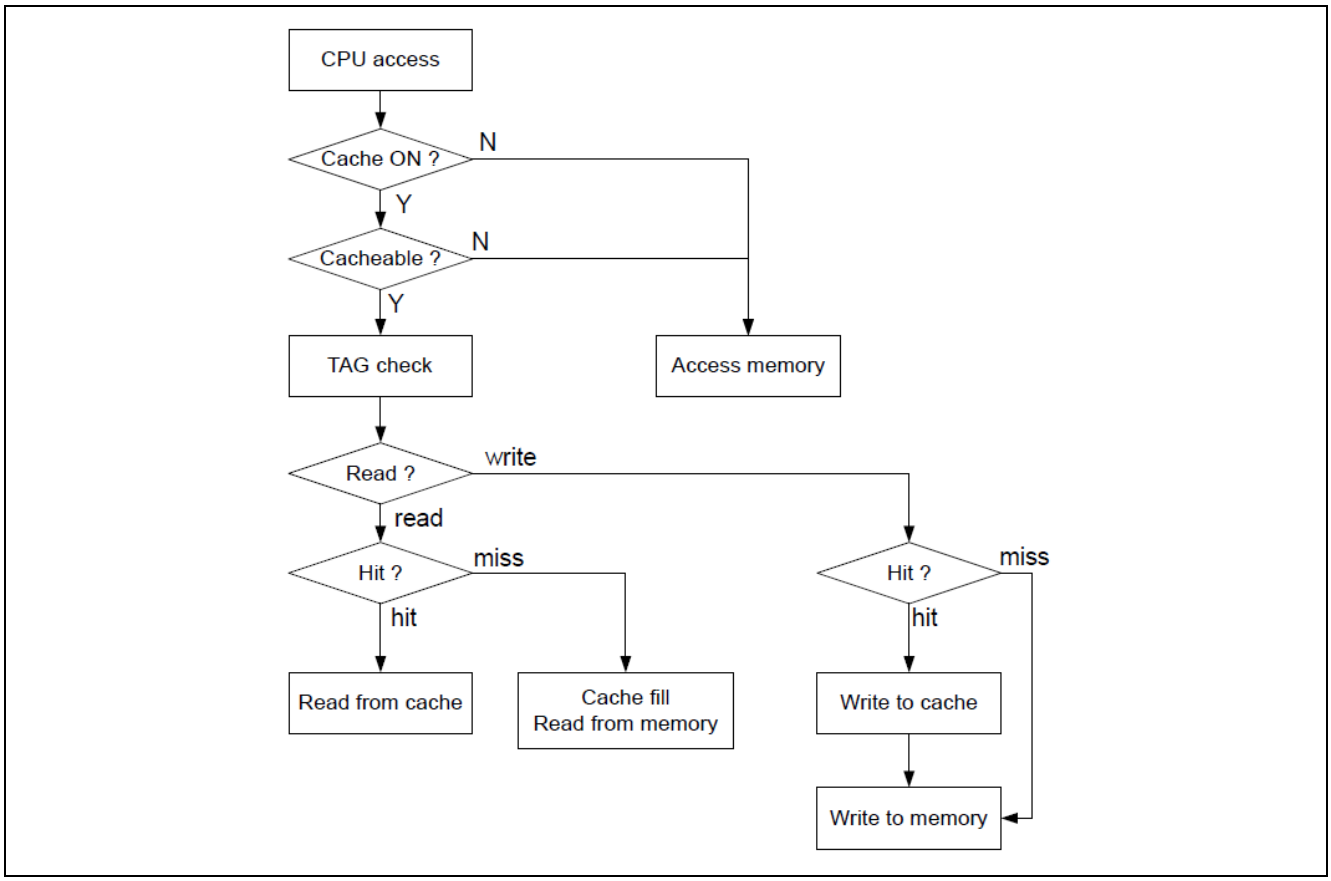


Figure 4. Access flow from CPU to S cache

2. Using S Cache in An Application

Consider using S cache for improved MCU performance based on the analysis in this section. Guidelines on when to use S cache in an application, usage notes for using S cache and how to keep S cache coherent are addressed in this section.

2.1 Using S Cache to Improve MCU Performance

The description in this section uses RA6M4, RA6M5 and RA4M3 as examples. User can adapt the same analysis to other MCUs wherever it applies.

RA6M5 and RA6M4 have a maximum system clock of 200 MHz. Access to SRAM is a slower process compared to the CPU speed. The analysis of using S cache on RA MCU assumes the Error Correction Code on the SRAM is disabled (which is the default setting from the MCU and FSP point of view). Under this condition, the read access to S cache is 1 cycle with cache hit and access to SRAM is 4 cycles (with 1 wait state) when the system bus is operating at over 100 MHz. This application project demonstrates the MCU performance improvement when the CPU is operating at 200 MHz. When operating at 100 MHz or less, read access to S cache is 1 cycle with cache hit and access to SRAM is 3 cycles (with 0 wait state).

For RA4M3, the maximum system clock is 100 MHz. Accessing SRAM is always 0 wait states. Read access to S cache is 1 cycle with cache hit and read access to SRAM is 3 cycles (with 0 wait state). For this reason, enable S cache if improved system performance is desired.

For RA MCUs with S cache support, consider enabling S cache to boost system performance when the data processed by the CPU exhibits a significant spatial locality, like in the case of a working buffer which does not need to be updated frequently.

Note that when S cache is enabled, and the above condition is met, the more frequently the data in S cache is used without needing an update, the larger the benefit of using S cache.

For a cache miss, the bus access cycle is same as when the cache is disabled for all MCUs which support S cache. And there is no performance improvement from cache write operations. For data not frequently used, filling the cache is an initial operation that will not be repeated or is repeated with very low frequency.

2.2 Configuring the S Cache Registers on RA6M5

The following table summarizes the S cache registers, their functionality and the application functions used in this application project to configure these registers. Refer to the included example projects to look at the detailed definitions for these functions.

Table 3. S Cache Register Configuration Demonstrated in the Application Project

Registers	Functionality	API created in application code
SCACTL: S Cache Control Register	Enable and disable S cache	`void enable_s_cache(void);` `void disable_s_cache(void);`
SCAFCT: S Cache Flush Control Register	Flush or do not flush the S cache	`void flush_s_cache(void);`
SCALCF: S Cache Line Configuration Register	Configuration register that configures the S cache line size to 32 or 64 (default is 32)	`void select_s_cache_line_size(bool line_size_32);`

For other S Cache related registers, the application project uses the default setting after MCU reset. Table 4 is a summary of these registers and their default settings used in the application project.

Table 4. Registers Configured at Default MCU Reset State

Registers	Functionality	Default Settings used in the Application Project
CSAR: Cache Security Attribution Register	This register defines the security attributes of registers for Cache Control, Line Configuration, and Cache Error.	This register is write-protected by the PRCR register. The default setting is used in the application project. Both secure and non-secure projects can use these attributes.
CAPOAD: Cache Parity Error Operation After Detection Register	This register defines the action the MCU will take when a Cache Parity Error is detected. The options are Non-Maskable Interrupt or Reset.	The default setting is Non-Maskable Interrupt. This setting is used in the application project. Demonstrations on the handling of the NMI interrupt are out of the scope of this application project.

Some tips to maximize the MCU bus performance are discussed. And finally, guidelines on how to design the software to benefit from the S Cache update scheme are provided.

2.3 Improving the CPU Performance

As explained in section 2.1, enabling S cache can improve system performance for some applications. The analysis in section 2.1 focuses on the time saving from the bus cycle access point of view. Aside from the bus access, instruction cycles are also a factor which influences the system performance. Therefore, the perceived system performance improvement will not be proportional to the bus cycle savings.

The analysis of the system performance improvement based on the example project provided in this application project is provided in later sections.

2.3.1 Allocating Memory Access to Maximize the MCU Bus Performance

Several guidelines for memory allocations should be considered when designing the software for the purpose of improved performance, for example, when S Cache is enabled on RA MCUs.

- Variables often accessed together should be close to one-another in memory. This increases the likelihood that the other variable will already be in the cache after the processor has accessed the first variable, thus avoiding cache misses.
- When accessing data linearly, use vectors or arrays. Linked lists, hash maps, dictionaries and so forth are great data structures for many things, but they are not cache friendly. Iterating through such a data structure involves many cache misses. If performance is important, stick to arrays. In addition, use arrays of values instead of arrays of pointers. Accessing the variable using a pointer invariably involves a cache miss. So, for fast array access, dispense with the pointers and go with values.

2.3.2 Designing for Data Structure Grouping and Alignment

When looking at how a program accesses memory, design decisions can be made that will take the most advantage of cache. If a data set that a program is working on is smaller than the cache line size of the processor, it is important to make sure that the data is read into one cache line. This is done by grouping the data together in a structure and aligning that structure, so it stays in a cache line.

For example, suppose a function uses local variables i and j as subscripts into a 2-dimensional array, they might be declared as follows:

```
int i, j;
```

These variables are commonly used together, but they can fall in different cache lines, which could be detrimental to performance. If the variables are used in a part of the program that is performance-critical, we could instead declare them as follows:

```
struct { int i, j; } sub;
```

This relies on the compiler's default alignment for structures. This default alignment is typically enough to ensure that the structure would be aligned in cache such that both indexes would be in the same cache line. `i` and `j` must now be referred to as `sub.i` and `sub.j`.

The alignment of the structure can be specified if the compiler supports this feature. Here is an example using the attribute feature of GCC to align a structure on an 8-byte boundary:

```
struct { int i, j; } sub __attribute__ ((aligned (8)));
```

2.3.3 Understanding the S Cache Update Strategy

The RA MCUs use the Least Recently Used (LRU) policy as the cache replace method. With the Cache Write-through, no-write allocate policy, the cache is filled upon read miss as shown in Figure 4.

To benefit from the LRU policy, design the system with the following points in mind to avoid cache replace events whenever possible:

- Use the data while still in cache. Consider that data usage and if possible, load data from the memory to the cache just once, use them or do some modifications on them, and then return it back to the operating memory. If we need to store the same data from SRAM to cache, we are not using the cache optimally.
- Reduce the number of times data which is already saved to the cache is written to memory if these variables are updated. For example, in a sorting algorithm, we can reduce the instances of writing the original array by employing some intermediate variables.

2.4 Keeping S Cache Coherent

Cache coherency needs to be considered when the cacheable region is accessed by both the CPU and other bus masters (such as DTC, DMAC). For shared memory between MCU and other bus masters (DTC and DMAC), S cache needs to be flushed prior to CPU access or the shared memory area can be set as Non-cacheable using the MPU. Otherwise, it might use stale data from the S cache since other bus masters might have updated the SRAM.

2.4.1 Flushing the S Cache

Flushing the S Cache can be achieved by one of two ways:

- Flush S Cache in the application code
  Software developers know which regions are common for CPU and other masters and they know when the CPU or another master writes to the command regions, the software developer can decide on what regions are cacheable by setting up the MPU.

  The S-Cache is a write through cache, when the CPU writes to an address, and that address is already in the cache, a cache HIT occurs for the write. The data is written to the cache, then the cache will subsequently write the data out to the main system memory, so the cache and main system memory will be coherent after the main system memory is written. This means for a CPU write to memory, the only cache coherency issue occurs for a short time while the Cache is doing the write to main memory. For the cacheable regions, the recommendation is for software developer to flush the S cache prior to the CPU's read access to the common region. This method is demonstrated in this application project.

- Flush S cache at the end of bus master transfer
  This method may incur more overhead when frequent transfers are needed. This method is demonstrated in this application project.

Keep in mind that flushing the S Cache invalidates the entire S Cache, not just the shared regions. Flushing the S Cache should be done as infrequently as possible to maintain the best system performance.

2.4.2 Using the Arm® Memory Protection Unit

User can reference below link to understand the fundamental of the Arm v8-M MPU:

Memory Protection Unit (MPU) Version 1.0 (arm.com). For a more detailed description of the Arm v8-M MPU, user can reference the Armv8-M Architecture Reference Manual.

Here are some of the key points that are covered in the above links that are related with the usage of S Cache. User should keep these in mind when using the MPU for S Cache control.

- The memory model and address space.
- The MPU programmers' model: Renesas support 8 MPU regions when TrustZone is not enabled and 8 MPU regions each for secure and non-secure region when TrustZone is enabled. 32 bytes as the smallest size, 32 bytes aligned addressing, configurable by series of memory mapped-registers.
- The difference between Armv7-M and Armv8-M.1 MPU
- The memory types and attributes: Normal memory are cacheable by default when the MPU is enabled. Note that Renesas MCU architecture defined the Standby SRAM region as Non-cacheable by hardware when S Cache is enabled. MPU can selectively set some Normal memory regions as Non-cacheable. Device memory is always non-cacheable.
- Memory Barrier Instructions:
  - A Data Memory Barrier (DMB) operation is recommended to force any outstanding writes to memory before enabling the MPU.
  - A DSB is used after enabling the MPU to ensure that the subsequent ISB instruction is executed only after the write to the MPU Control register is completed. The ISB instruction is used after the DSB to ensure the processor pipeline is flushed and subsequent instructions are re-fetched with new MPU configuration settings.
- MPU register overview
- Configuring an MPU region (reference Configuring an MPU Region).

Setting up a memory region as Non-cacheable is very easy with the CMSIS API which follows the recommendations mentioned above (CMSIS support for MPU). An example of using CMSIS API to set up a memory area used by both the CPU and DMA master as Non-cacheable to resolve the S Cache coherency issue is demonstrated in this application project. Table 5 is a brief description on the CMSIS MPU APIs, all of these APIs are inline functions included in `mpu_armv8.h` which is automatically included when establishing a project template using the RA Smart Configurator. These CMSIS-APIs already are including the necessary memory barrier instruction calls.

Table 5. CMSIS MPU API

CMSIS MPU Configuration API	Functionality	Comments
ARM_MPU_Disable	Disable the MPU	This API should be called every time the MPU configuration is to be updated. This is to provide portability of the MPU configuration code. This API is demonstrated in the example project.
ARM_MPU_SetRegion	Configure the MPU region number, MPU Base Address Register and MPU Limit Address Register	This function is used to configure the location of one MPU region. This API is demonstrated in the example project.
ARM_MPU_SetMemAttr	Set up the MPU region attribute	This function is used to configure the attribute of one MPU region. This API is demonstrated in the example project.
ARM_MPU_Enable	Enable the MPU with the default memory map as background and define whether to enabled MPU during hardfault NMI.	This API is demonstrated in the example project.
ARM_MPU_Load	Configure a number of MPU regions using a table.	An example of using this API is included in the example code in Figure 5. to set the QSPI register region as Non-cacheable. This region is not set as Non-cacheable from the default memory map and must be included in any projects which utilize the QSPI.

As explained in section 1.2, the QSPI register area should be non-cacheable, when using the QSPI, user should set the IO register region as non-cacheable. Optionally user can set the memory area as nonn-cacheable as well.

```
#define MPU_REGION_0        0U
#define MPU_REGION_1        1U
#define REGION_0_ATTR_IDX   0U
#define REGION_1_ATTR_IDX   1U
#define READ_WRITE          0U
#define READ_ONLY           1U
#define PRIVILEGED_ONLY     0U
#define ANY_PRVILEGE        1U
#define EXECUTION_PERMITTED 0U
#define NO_EXECUTION        1U
const ARM_MPU_Region_t mpuTable[1][2] = {
{
    //  BASE     SH      RO   NP   XN     LIMIT   ATTR
    { .RBAR = ARM_MPU_RBAR(0x60000000UL, ARM_MPU_SH_NON, 1UL, 1UL, 0UL), .RLAR =
ARM_MPU_RLAR(0x63FFFFFUL, 1UL) },
    { .RBAR = ARM_MPU_RBAR(0x64000000UL, ARM_MPU_SH_NON, 0UL, 1UL, 1UL), .RLAR =
ARM_MPU_RLAR(0x67FFFFFUL, 2UL) }
```

```
}
};
/* Disable MPU */
ARM_MPU_Disable();
ARM_MPU_Load(0, mpuTable[0], 2);
ARM_MPU_SetMemAttr(REGION_0_ATTR_IDX, ARM_MPU_ATTR(ARM_MPU_ATTR_MEMORY_(0, 0, 1, 0),
ARM_MPU_ATTR_MEMORY_(0, 0, 1, 0))); //ARM_MPU_ATTR_MEMORY_(NT, WB, RA, WA)
ARM_MPU_SetMemAttr(REGION_1_ATTR_IDX, ARM_MPU_ATTR(ARM_MPU_ATTR_DEVICE_nGnRnE,
ARM_MPU_ATTR_DEVICE_nGnRnE));
/* Enable MPU, enable default memory map as background, MPU enabled during fault and NMI
handlers */
   ARM_MPU_Enable(MPU_CTRL_PRIVDEFENA_Msk | MPU_CTRL_HFNMIENA_Msk);
```

Figure 5. Setting the QSPI Register Region and Memory Region as Non-cacheable

Note that in order to use the MPU on the default memory map, user needs to enable the MPU and enable the privileged mode. See below MPU_CTRL register attributes based on the (https://developer.arm.com/documentation/100235/0004/the-cortex-m33-peripherals/security-attribution-and-memory-protection/mpu-control-register). In our example project, the default memory map is used, so both ENABLE bit and PRIVDEFENA bit are enabled to configure the MPU regions.

Bits	Name	Function
[31:3]	-	Reserved, res0.
[2]	PRIVDEFENA	Enables privileged software access to the default memory map. When the MPU is enabled: 0 Disables use of the default memory map. Any memory access to a location that is not covered by any enabled region causes a fault. 1 Enables use of the default memory map as a background region for privileged software accesses. When enabled, the background region acts as if it has the lowest priority. Any region that is defined and enabled has priority over this default map. If the MPU ignores this bit.
[1]	HFNMIENA	Enables the operation of MPU during HardFault and NMI handlers. When the MPU is enabled: 0 MPU is disabled during HardFault and NMI handlers, regardless of the value of the ENABLE bit. 1 The MPU is enabled during HardFault and NMI handlers. When the MPU is disabled, if this bit is set to 1 the behavior is UNPREDICTABLE.
[0]	ENABLE	Enables the MPU: 0 MPU is disabled. 1 MPU is enabled.

Figure 6. MPU_CTRL Register

2.4.3 Choosing the Preferred Method

Which method to use in the user application to avoid S Cache coherency issue is highly application dependent. To reduce S Cache flushing influence on CPU performance, when to flush the S Cache needs to be carefully considered. In addition, user should design the application based on the recommendations from section 2.3 so the benefit of using the S Cache is maximized, which is also helpful to offset the overhead of the operations to avoid S Cache coherency issue.

If frequent S Cache flushing is inserted synchronously to the flow of the application, the performance of the system might be negatively influenced under certain conditions. For example, when using EDMAC with Ethernet applications, the transfer speed is very fast and the shared region is used very frequently, in this case, setting the shared memory buffer as non-cacheable can be a better option than S Cache flushing to achieve S Cache coherency.

3. Example Project

3.1 Overview

This example project demonstrates how to enable and disable S Cache, how to handle S Cache coherency and how to use the cycle counter on the debug unit Data Watchpoint and Trace Unit (DWT) to evaluate the CPU performance improvement when S Cache is enabled.

System setup:

- A sine and cosine data set are stored in code flash.
- The data set is then transferred to the SRAM via a DMA channel.
- Next, the standard deviation of $sine^2$+$cosine^2$ are calculated by reading the sine cosine data from the buffer in the SRAM.
- The standard deviation should be 0 if there is cache coherency. When the S Cache is enabled, since both CPU and MPU access the shared area, the content in this area can lose coherency. The S Cache coherency issue is manifested by enlarged standard deviation. The example project demonstrated three methods to keep the cache coherent and hence recover the correct standard deviation.

The FSP modules used in this example project include r_dma, r_agt, and Arm® CMSIS DSP library. Their functionalities are explained briefly as follows:

- r_dmac: transfer data to DAC register to generate the sine and cosine wave
- r_agt: time the DMA transfer of the DAC data
- Arm® CMSIS DSP module: calculate the standard deviation of ($sine^2$ + $cosine^2$)
- Arm CMSIS MPU API: set up the shared SRAM region as non-cacheable

In addition, the cycle counter on the debug unit Data Watchpoint and Trace Unit (DWT) is used to track CPU cycles used in a fixed set of calculations when S Cache is disabled or enabled.

Analysis of S Cache Usage:

- The deviation of ($sine^2$ + $cosine^2$) will be larger if S Cache coherency is broken. See section 3.3 for this analysis.
- $sine^2$ + $cosine^2$ calculation should be faster when S Cache is enabled. See section 3.4 for this analysis.
- When the SRAM region which is shared by CPU and DMA is set as Non-cacheable, the $sine^2$ + $cosine^2$ calculation took slightly longer time with S Cache enabled compared with flushing the S Cache.
- This application project provides routines to update S Cache line size. But it does not demonstrate the line size configuration to CPU performance. For set associative cache, line size primarily influences the cache miss time penalty. Larger line size means larger penalty in time when a cache miss happens because it takes longer to bring the line in to the cache.

  To show the set associative cache line size influence on the CPU performance, frequent S Cache misses need to be simulated. This is not demonstrated in this example project because there is no frequent S Cache miss designed in the performance analysis routine. On the other hand, for a cache of constant size, using larger line size increases spatial locality which can be helpful for some applications. User should analyze the application at hand to select the line size that supports the best performance of the system. This is typically achieved through empirical investigation. Once the line size is determined for a system, it should not be randomly changed unless a new analysis is performed.

3.2 Import and Run the Example Project

Import project `using_s_cache_ra6m5` into an e² studio workspace. Click **Generate Project Content** and compile the example project. Next, connect **J10 USB Debug** port on EK-RA6M5 to the development PC. Right click on the project `using_s_cache_ra6m5` and select **Debug As > Renesas GDB Hardware Debugging**.

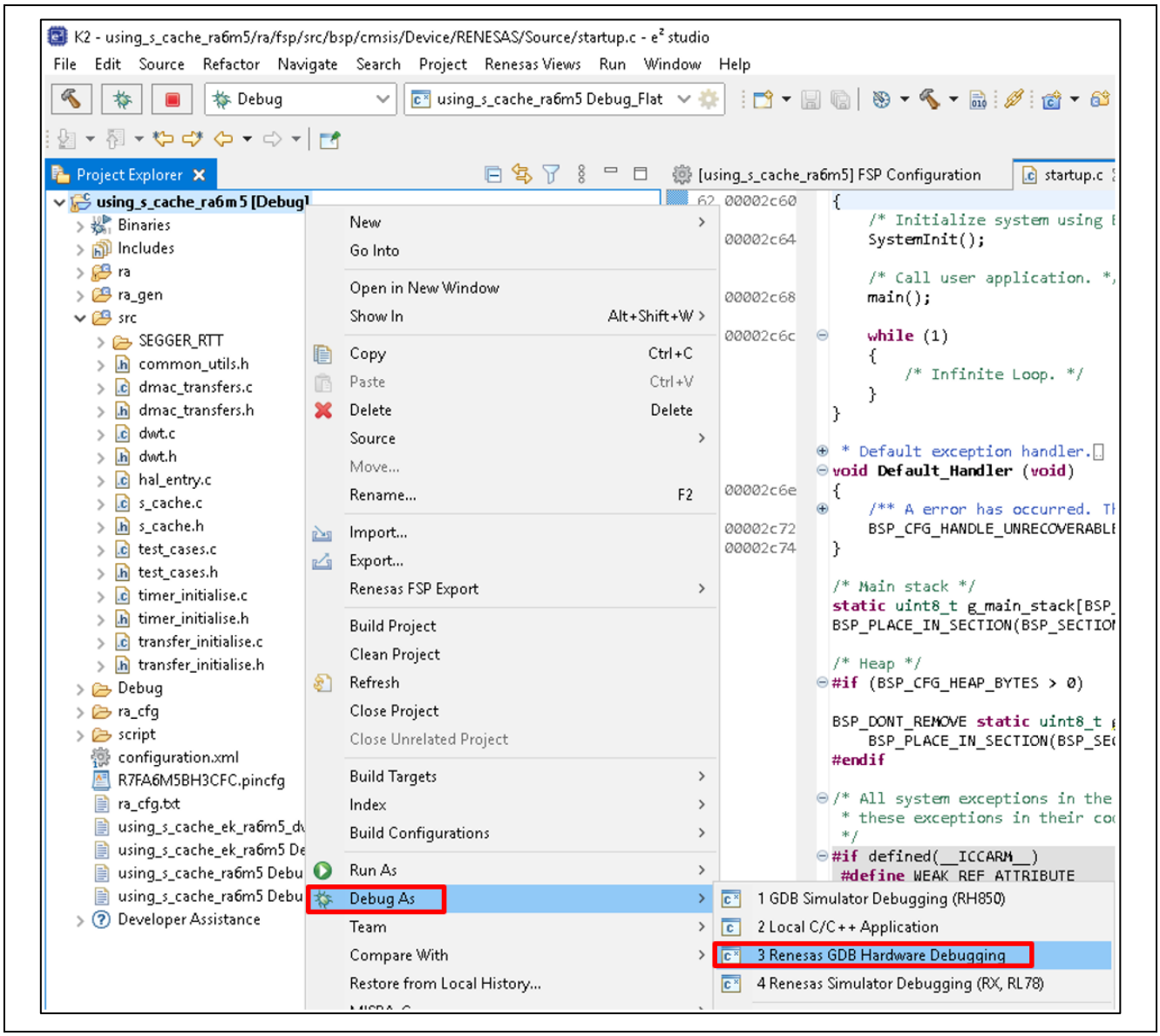


Figure 7. Using S Cache Example Project

Connect to RTT viewer.

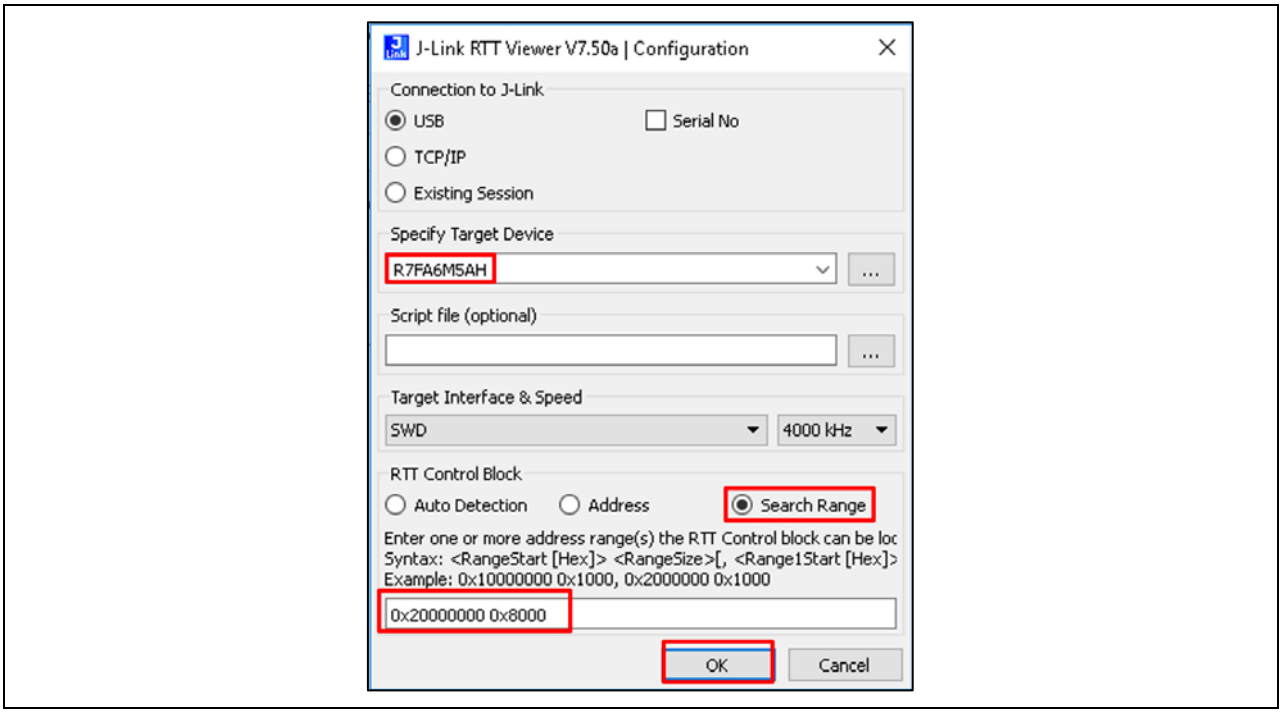


Figure 8. Connect to SEGGER RTT Viewer

The actions a user can take through the RTT user interface are: S Cache configuration, whether to flush S Cache, where to Flush S Cache, as well as the S Cache line configuration.

```
input 1 to calculate the standard deviation with s cache disabled

input 2 to calculate the standard deviation with s cache enabled with no cache invalidation

input 3 to calculate the standard deviation with s cache enabled and flushed in DMA_Complete interrupt

input 4 to calculate the standard deviation with s_cache enabled and flushed in application

input 5 to calculate the standard deviation with s_cache enabled and DMA buffer in non-cacheable region

input 6 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with s cache disabled

input 7 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with s cache flushed in DMA_Complete IRQ callback
        with line size 32

input 8 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with s cache flushed in app with line size 32

input 9 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with sram region used by DMA as non-cacheable
              with line size 32

input 10 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with s cache flushed in DMA_Complete IRQ callback
        with line size 64

input 11 to evaluate the DWT cycles used in 180000 sine^2 + cosine^2 calculations with s cache flushed in app with line size 64
```

Figure 9. Actions Users Can Perform via RTT User Menu

3.3 Demonstration of How to Keep S Cache Coherent

When the S Cache is enabled and filled, the calculation uses the data from S Cache, which can be different from the data transferred to the SRAM via the DMA transfer. This example project demonstrated that when S Cache is disabled, the standard deviation of ($sine^2 + cosine^2$) is 0 as expected.

When S Cache is enabled, the S Cache is corrupted after DMA transfers data to SRAM. When ($sine^2 + cosine^2$) is calculated, the corrupted S Cache is used and hence generates larger standard deviation.

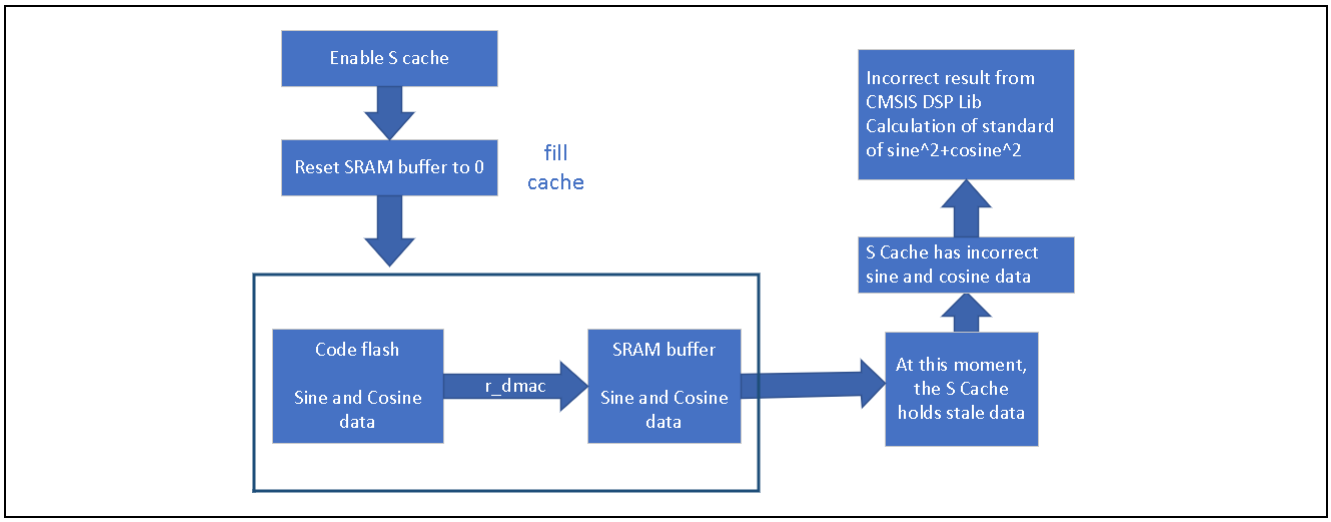


Figure 10. S Cache Coherency is Broken due to DMA Transfer to Common Area

When the S Cache is flushed in a DMA transfer complete interrupt callback and in the user application prior to the calculation of ($sine^2 + cosine^2$), S Cache coherency is restored.

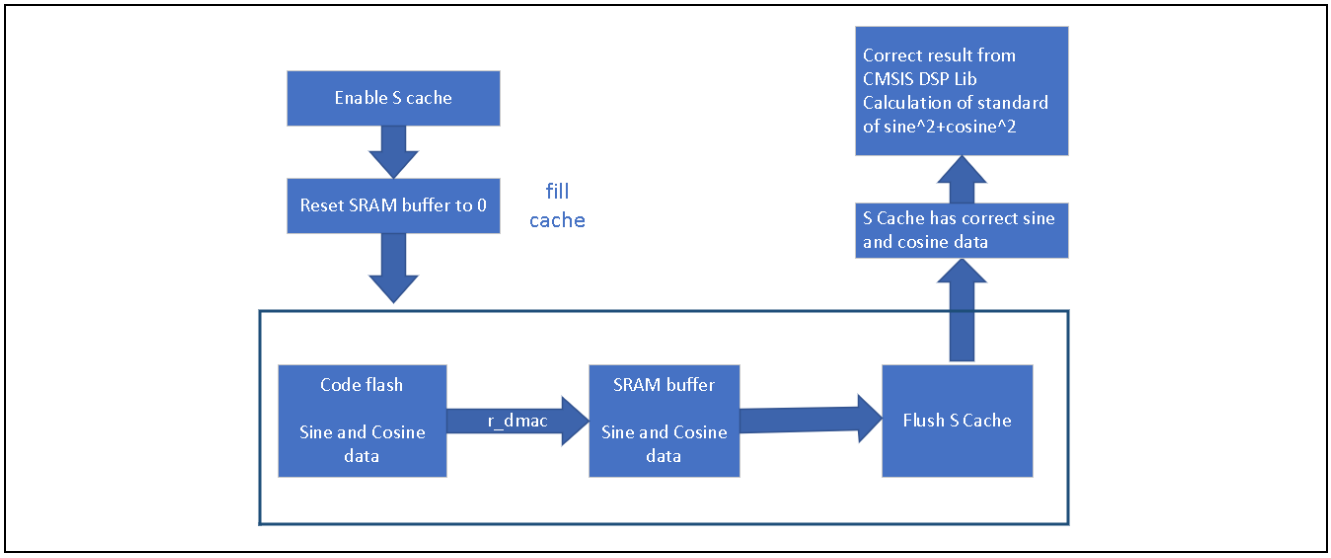


Figure 11. S Cache Coherency is Restored – Flush S Cache in Application Code

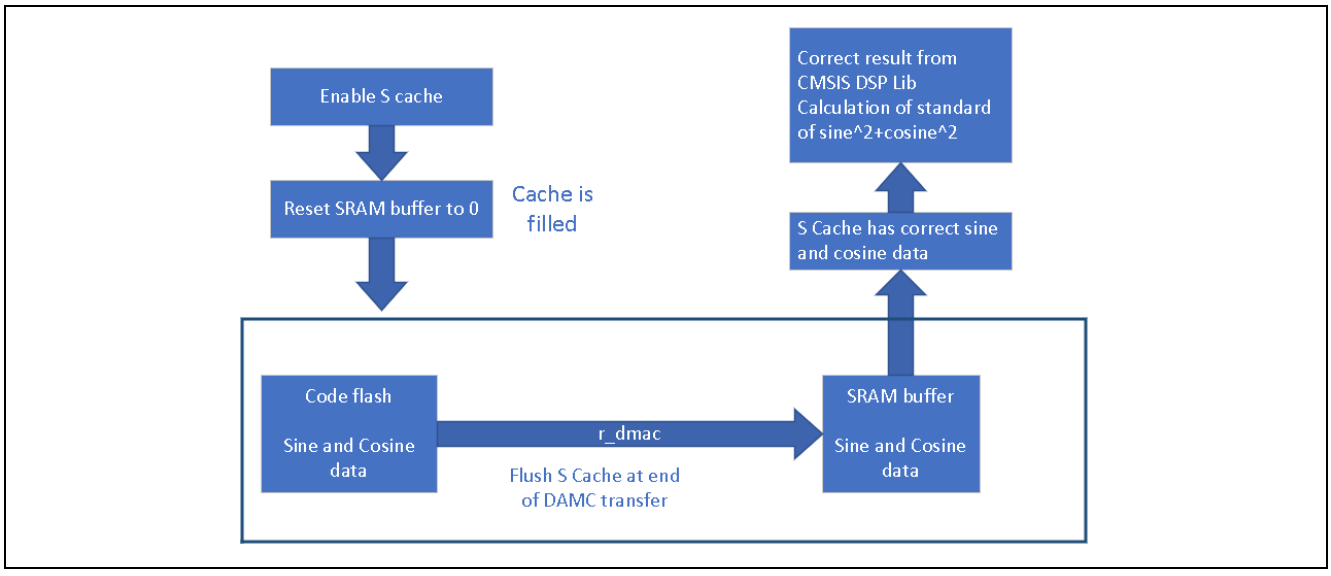


Figure 12. S Cache Coherency is Restored – Flush S Cache in DMA Transfer Complete Callback

Another method to achieve S Cache coherency is to set the SRAM Sine and Cosine data area as non-cacheable. Doing so will slightly reduce the performance of the system compared with flushing the S Cache based on the example project.

Figure 13 is an example run of the S Cache coherency handling routines provided in this application project.

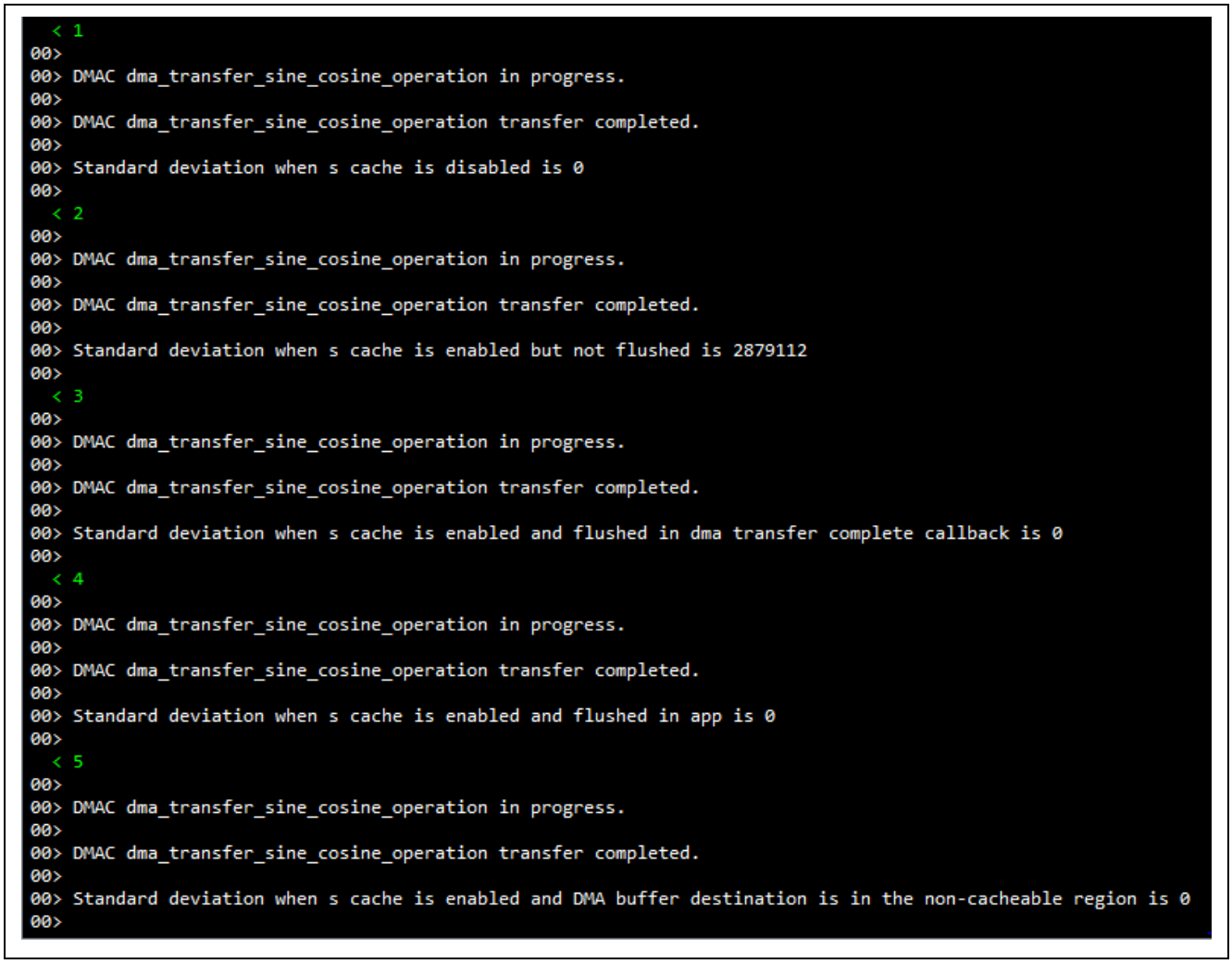


Figure 13. Demonstration of How to Keep S Cache Coherent

Table 6. Standard Deviation of Sine² + Cosine²

S Cache Configuration	Standard Deviation
Disabled	0
Enabled but S Cache not Flushed after DMA Transfer	Around 2879112
Enabled and S Cache Flushed in DMA Complete Transfer	0
Enabled and S Cache Flush in Application Code	0
Enabled and SRAM region used by DMA and CPU is non-cacheable	0

3.4 Demonstration of MCU Performance Improvement

In this example project, 1000 cycles of 180 (sine² + cosine²) calculations are performed. The number of DWT cycles used for this calculation is captured and displayed on the RTT Viewer.

```
  < 6
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used when s cache is disabled is 14520604
00>
  < 7
00>
00> Test setup is: S cache is enabled with line size set to 32 and S cache is flushed in DMA complete interrupt.
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used is 7938274
00>
  < 8
00>
00> Test setup is: S cache is enabled with line size set to 32 and S cache is flushed in application.
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used is 7938295
00>
  < 9
00>
00> Test setup is: S cache is enabled with line size set to 32 and SRAM region set as non-cacheablen.
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used is 8490346
00>
  < 10
00>
00> Test setup is: S cache is enabled with line size set to 64 and S cache is flushed in DMA complete interrupt.
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used is 7938245
00>
  < 11
00>
00> Test setup is: S cache is enabled with line size set to 64 and S cache is flushed in application.
00>
00> DMAC dma_transfer_sine_cosine_operation in progress.
00>
00> DMAC dma_transfer_sine_cosine_operation transfer completed.
00>
00> DWT cycle used is 7938242
```

Figure 14. Demonstration of CPU Performance Improvement when S Cache is Enabled

From the output presented in the above example, the CPU performance improvement is about 50%. This presented CPU performance increase depends on savings from bus access as well as instruction cycle access. When the SRAM area used by the DMA and CPU is set as non-cacheable, the performance improvement is slightly lower than flushing the S Cache with a drop of about 7%.

As explained in the overview section 3.1, this example project does not demonstrate the line size influence on the CPU performance. The number of DWT cycle counter stays about the same for 32-byte or 64-byte line size configuration.

Also, notice that the CPU performance stays about the same when using the three different flushing methods, whether flushing at the end of the DMA transfer or in the application or setting the shared region as non-cacheable.

4. References

RA6M5 Group User's Manual: Hardware: https://www.renesas.com/document/man/ra6m5-group-users-manual-hardware?language=en&r=1493931

5. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA6M5 Resources	renesas.com/ra/ek-ra6m5
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.0.0	Jan.06.22	-	First release document
1.1.0	May.03.23	-	Add MPU example code and description

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/.