How to Achieve Deterministic Code Performance Using a Cortex™-M Cache Controller

Introduction

In microcontroller-based embedded applications, the software is stored and run from non-volatile memory, which is typically Flash memory. Although Flash memories provide an efficient medium to store and execute code, a number of factors limit the deterministic code performance when executed from Flash. One of the important factors impacting the deterministic code behavior is the intricacies of a system bus matrix. The issues of deterministic code performance are also seen when the code is run from SRAM due to the same reasons as that for the Flash memory.

Non-deterministic code performance is primarily due to the varying propagation time of code from the memories to the CPU. The system bus matrix that interfaces the memories to CPU contributes to the varying propagation time. The non-deterministic code performance is more evident in systems with multiple entities accessing the system bus.

In real-time applications, there are situations where small, critical pieces of code require time bound execution. It is not advised to run such code from Flash memory or SRAM, as it might not achieve the intended deterministic timing due to system bus arbitration, that is, a cache miss condition requires the code to be fetched from the Flash memory or SRAM. The code access time might vary depending on the availability to access system bus as it is arbitrated between multiple bus entities.

The effective way to achieve deterministic code performance of critical code is to execute the code from the Tightly Coupled Memory (TCM) to the processor, and avoid cache miss conditions. The ARM® Cortex®-M Cache Controller (CMCC) peripheral on Microchip's Cortex-M4 based microcontrollers (MCUs) provides support to run the critical code from the cache memory and achieve deterministic performance.
# Table of Contents

Introduction.............................................................................................................................................. 1

1. Concept.................................................................................................................................................. 3

2. Solution.................................................................................................................................................. 4

3. Deterministic Performance Analysis................................................................................................. 10

4. Relevant Resources............................................................................................................................. 12

The Microchip Web Site......................................................................................................................... 13

Customer Change Notification Service.................................................................................................. 13

Customer Support.................................................................................................................................... 13

Microchip Devices Code Protection Feature.......................................................................................... 13

Legal Notice................................................................................................................................................ 14

Trademarks................................................................................................................................................ 14

Quality Management System Certified by DNV....................................................................................... 15

Worldwide Sales and Service.................................................................................................................. 16
1. **Concept**

CMCC on Cortex-M4 based MCUs (i.e., SAME54) has a dedicated Four-way L1 set associative cache of 4 KB, as shown in the following figure.

*Figure 1-1. Four-way L1 Set Associative Cache Memory*

With CMCC, a part of the cache can be used as TCM for deterministic code performance by loading the critical code in a *WAY* and locking it. When a particular *WAY* is locked, the CMCC does not use the locked *WAY* for routine cache transactions. The locked cache *WAY* with the loaded critical code acts as an always-getting cache hit condition.
2. **Solution**

The following flow sequence shows the code to implement deterministic code performance.

**Figure 2-1. Flow Sequence to Implement Deterministic Code Performance**

The following code examples show the implementation of functions for achieving deterministic code performance and its usage on Cortex-M4 based MCUs (i.e., SAME54).
Figure 2-2. Implementation of Macros and Variables Used in cmcc_loadnlock Function

```
#define CMCC_NO_OF_WAYS 4
#define CMCC_WAYSIZE 1024
#define CMCC_WAY_NO_OF_LINES 64
#define CMCC_WAY_LINE_SIZE 16

#define CMCC_WAY0 (1 << CMCC_MAINT1_WAY_WAY0_Val)
#define CMCC_WAY1 (1 << CMCC_MAINT1_WAY_WAY1_Val)
#define CMCC_WAY2 (1 << CMCC_MAINT1_WAY_WAY2_Val)
#define CMCC_WAY3 (1 << CMCC_MAINT1_WAY_WAY3_Val)

const uint8_t load_pass[CMCC_WAY_LINE_SIZE] = {
    0xA5, 0xA5, 0xA5, 0xA5,
    0xA5, 0xA5, 0xA5, 0xA5,
    0xA5, 0xA5, 0xA5, 0xA5,
    0xA5, 0xA5, 0xA5, 0xA5
};
```
void cmcc_loadnlock(uint32_t way_bitfield, void (*)(void), uint32_t size)
{
    volatile uint32_t dummy;
    uint8_t* p_foo;
    int8_t way_index;

    /* Disable the cache */
    CMCC->CTRL.reg = 0;
    while ((CMCC->SR.bit.CSTS != 0));
    /* Invalidate by line the whole desired WAY */
    for (volatile uint32_t i = 0; i < CMCC_WAY_NO_OF_LINES; i++) {
        CMCC->MAINT1.reg = CMCC_MAINT1_WAY(way_bitfield) | CMCC_MAINT1_INDEX(i);
    }

    /* Disable instruction cache (icdis register is set ) */
    CMCC->CFG.bit.ICDIS = 1;

    /* Enable data cache (dcdis register is clear) */
    CMCC->CFG.bit.DCDIS = 0;

    /* Enable the cache */
    CMCC->CTRL.reg = CMCC_CTRL_CEN;
    while ((CMCC->SR.reg != CMCC_SR_CSTS));
    /* Find the WAY index */
    if(CMCC_WAY3 == way_bitfield) {
        way_index = ((way_bitfield >> 1) - 1);
    } else {
        way_index = way_bitfield >> 1;
    }

    /* Parse through the WAYS to find the desired WAY */
    for(uint32_t index = 0; index < CMCC_NO_OF_WAYS; index++) {
        if(way_index != index) {
            /* Not the desired WAY, Still need to pass through by loading the entire WAY by loading dummy data */
            for (uint32_t i = 0; i < CMCC_WAY_NO_OF_LINES; i++) {
                dummy = *((uint8_t*)load_pass;
            }
        } else {
            /* Desired WAY found, Load with the critical code from flash */
            size = CMCC_WAY_LINE_SIZE;
            p_foo = (uint8_t*)f;
            for (uint32_t i = 0; i < size; i++) {
                dummy = *p_foo;
                p_foo += CMCC_WAY_LINE_SIZE;
            }
            break;
        }
    }

    /* Then write the lock per way register */
    CMCC->LCKWAY.bit.LCKWAY[way_bitfield] = way_bitfield;

    /* Re-enable instruction cache (icdis register is clear ) */
    CMCC->CFG.bit.ICDIS = 0;
}
The variable CMCC in the above implementation is defined as shown in the following example. The implementation provided is for the Cortex-M4 based device, SAME54P20A. The details of the structure implementation, and the definitions of macros CMCC_MAINT1_WAY, CMCC_MAINT1_INDEX, CMCC_CTRL_CEN and CMCC_SR_CSTS can be found in Microchip’s Atmel START (ASF4) library.

Figure 2-4. Declaration of CMCC Structure

```
#define CMCC
typedef struct {
    I CMCC_TYPE Type;
    I CMCC_CFG Type;
    I CMCC_CTRL Type;
    I CMCC_MAINT Type;
    I CMCC_LCKWAY Type;
} CMCC;
```

Implementation

Refer to the following steps to implement the cmcc_loadnlock function:

1. When the cache is enabled, the CMCC uses the four WAYs of the set associative cache in round robin fashion starting with WAY0. This happens immediately after enabling the cache.
2. The cmcc_loadnlock function scans through the cache WAYs to locate the desired WAY (refer to the code example labeled A).
3. In a pass, if the desired WAY is not found, the cmcc_loadnlock function loads the entire WAY with values from an array so as to pass over the WAY under process and proceed to check whether the next WAY is the desired WAY (refer to the code example labeled B).
4. When the desired WAY is found, the cmcc_loadnlock function reads the critical code from Flash into a dummy variable to ensure that the critical code is stored in a cache WAY (refer to the code example labeled C).
5. In the labels marked B and C, the cmcc_loadnlock function loads only one word (4 bytes) out of the required four (16 bytes), this is because the WRAP4 feature of AHB fills in the gaps by loading the entire line of 16 bytes.
6. The cmcc_loadnlock function locks only the desired cached WAY to trap the critical code in the cache. When a particular WAY is locked, the CMCC discontinues using the locked WAY for the usual round robin cache transactions. The locking of a WAY and trapping the critical code in a cache, emulates a situation of calls made to the critical code getting 100% cache hit.
7. The implementation of the cmcc_loadnlock function shown above works for critical code size less than or equal to the size of a cache WAY (CMCC_WAYSIZE), that is, 1024 bytes. If the critical code size is larger than the size of a WAY, the critical code needs to be accommodated in a consecutive second WAY. To accommodate critical code of larger size the implementation of cmcc_loadnlock function needs to be enhanced.

Consider the following guidelines for enhancing the implementation of the cmcc_loadnlock function.

7.1. The parameter, size, is to be evaluated to identify whether the critical code fits in the available cache of 4 KB. If it fits in the available cache, the implementation needs to identify how many cache WAYs are needed.
7.2. The implementation needs to identify if the required number of consecutive WAYs are available to be allocated for the critical code.

7.3. The implementation should invalidate the consecutive WAYs (refer to the code example labeled 1).

7.4. The implementation should load the consecutive WAYs with the critical code (refer to the code example labeled C).

7.5. The implementation should lock the consecutive WAYs so as to trap the code in cache (refer to the code example labeled 2)

Usage

The following code example shows the usage of the cmcc_loadnlock function.

Figure 2-5. Usage of the cmcc_loadnlock Function

```c
static void _critical_code_function(void __attribute__((section(".cc_function_section"))));
int main(void)
{
    atmel_start_init();

    cmcc_enable();
    cmcc_loadnlock(CMCC_WAY0, _critical_code_function, 0x10);

    _critical_code_function();

    while (true)
    {
        ;
    }
}
```

static void _critical_code_function(void)
{
    /* Implement your critical code */
}

1. The cmcc_loadnlock function accepts three parameters.
   1.1. The first parameter `way_bitfield` is the number of the cache WAY, this would take one of the four WAY values (CMCC_WAY0, CMCC_WAY1, CMCC_WAY2, CMCC_WAY3).
   1.2. The second parameter, `f`, is the address of the critical code function that needs to run always from the cache.
   1.3. The third parameter, `size`, is the size of the critical code function. The size should not be greater than the size of the cache WAY (CMCC_WAYSIZ). If the size is greater than the WAY size, enhance the implementation as mentioned in the above step seven.

2. The exact size of the critical code function `_critical_code_function` can be obtained from the project's listing file. Use the following steps to find the size of the critical code function.
   2.1. In the previous example, the critical code function is declared with the section attribute `_cc_function_section`. This is an instruction to the GNU linker to place the function `_critical_code_function` in a code segment with the name `_cc_function_section`.
   2.2. Place a dummy size in the cmcc_loadnlock function call.
   2.3. Clean and build the project.
2.4. Open the Project’s listing file (.lss) and locate the linker symbol _cc_function_section. The line with linker symbol shows the size of the section. This is the size of the critical code function.

Figure 2-6. Locating the _cc_function_section Attribute

<table>
<thead>
<tr>
<th>Sections:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idx</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

2.5. Replace the size of the critical code function in the call cmcc_loadnlock with the size from the .lss file.

2.6. Build and program.

3. The cmcc_loadnlock function can be used to dynamically store more than one critical code functions in the cache WAYs by calling it consecutively more than once.

Note:

1. This document covers usage of the unified four WAY L1 associative cache for deterministic code performance optimization only. The CMCC also allows data caching, and to use a portion of the cache as data TCM. For additional information, refer the Cortex-M4 based MCU (SAME54) data sheet.

2. The deterministic code performance implementation discussed in this document comes with a trade-off by reducing the active cache size.
3. Deterministic Performance Analysis

In a cache-enabled system, code performance is determined by the frequency of cache hit or cache miss conditions. The more the cache hit condition, the better the performance. In a simple application, for example, an application where there is only one function performing the key operation iteratively, there is a higher probability of cache hits. In contrast, in a complex real-time application, where multiple entities, such as System Bus Masters access the non-volatile memory, the code performance is limited by the higher probability of cache miss conditions due to the non-deterministic call sequence. In addition, there are delays due to the system bus matrix.

The following is the performance analysis of a simple application when the code is run from TCM (using the loadnlock implementation discussed previously) against the regular cache-enabled transactions.

- The following functions of 1 KB size has the same code that are used for profiling.

```c
    critical_code_function (); // [Referred as TCM]
    _function_1 (); // [Referred as C1]
    _function_2 (); // [Referred as C2]
    _function_3 (); // [Referred as C3]
    _function_4 (); // [Referred as C4]
    _function_5 (); // [Referred as C5]
    _function_6 (); // [Referred as C6]
```

- The critical code function [TCM] always runs from the cache (is locked in a cache WAY by calling loadnlock) while the other function runs from Flash and is cached by the CMCC-based on the round robin sequence.

- The functions above are called indefinitely in the same sequence.

- A 20-sample profile of the above function sequence running on a Cortex-M4 based processor (SAME54 MCU) at 120 MHz, 15 NVM wait states, and no code optimization is shown in the following figure. The vertical axis in the graph shows the time taken in microseconds (µs).

**Figure 3-1. Deterministic Code Performance Analysis**

- Summary:
  - The time taken by the code, running from TCM, is deterministic as it takes 6.47 or 6.48 µs. This is in contrast to the non-deterministic time taken by code running through routine cache...
transactions. In a single pass, such as Pass 1, the time consumption for the code running through normal cache operations varies from 6.86 µs to 7.26 µs.

- The possible occurrences of cache miss conditions when the functions C3 and C6 are run. There is a sudden spike in the time consumed (C3 taking 7.19 µs and C6 taking 7.26 µs) for running these functions.

- In addition to the deterministic code performance, there is performance benefit for the code running from TCM against the code running from the cache-enabled transactions. For the simple application, in a single pass, such as Pass 1, the time consumed for the code running from TCM is 6.47 µs as against the code running through normal cache operations that varies from 6.86 µs to 7.26 µs.

The code TCM implementation discussed above provides deterministic performance model for the critical code. It also provides performance benefits over code implementation with cache-enabled systems. The performance benefits are subject to the complexity of the application. As the complexity of the application increases, the performance benefits of the code executed from TCM appears evident.
4. **Relevant Resources**

For additional information, refer to these documents:

- SAM D5x/E5x Family Data Sheet:

- SMART SAM E70 TCM Memory:

- How to Optimize Usage of SAM S70/E70/V7x Architecture:
The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip’s customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.


Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

**Legal Notice**

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

**Trademarks**

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BeaconThings, BitCloud, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, LINK MD, maXSylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, RightTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.


SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.
Quality Management System Certified by DNV

ISO/TS 16949
Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.
## Worldwide Sales and Service

<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Office</td>
<td>Australia - Sydney Tel: 61-2-9868-6733</td>
<td>India - Bangalore Tel: 91-80-3090-4444</td>
<td>Austria - Wels Tel: 43-7242-2244-39</td>
</tr>
<tr>
<td>2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: <a href="http://www.microchip.com/support">http://www.microchip.com/support</a></td>
<td>China - Beijing Tel: 86-10-8569-7000</td>
<td>India - New Delhi Tel: 91-11-4160-8631</td>
<td>Denmark - Copenhagen Tel: 45-4450-2828</td>
</tr>
<tr>
<td></td>
<td>China - Chengdu Tel: 86-28-8665-5511</td>
<td>India - Pune Tel: 91-20-4121-0141</td>
<td>Finland - Espoo Tel: 358-9-4520-820</td>
</tr>
<tr>
<td></td>
<td>China - Chongqing Tel: 86-23-8980-9588</td>
<td>Japan - Osaka Tel: 81-6-6152-7160</td>
<td>France - Paris Tel: 33-1-69-53-63-20</td>
</tr>
<tr>
<td></td>
<td>China - Dongguan Tel: 86-869-8702-9880</td>
<td>Japan - Tokyo Tel: 81-3-6880-3770</td>
<td>Germany - Garching Tel: 49-8931-970</td>
</tr>
<tr>
<td></td>
<td>China - Guangzhou Tel: 86-20-8755-8029</td>
<td>Korea - Daegu Tel: 82-53-744-4301</td>
<td>Germany - Haan Tel: 49-2129-3766400</td>
</tr>
<tr>
<td></td>
<td>China - Hangzhou Tel: 86-571-8792-8115</td>
<td>Korea - Seoul Tel: 82-2-554-7200</td>
<td>Germany - Heilbronn Tel: 49-7131-67-3636</td>
</tr>
<tr>
<td></td>
<td>China - Hong Kong SAR Tel: 852-2943-5100</td>
<td>Malaysia - Kuala Lumpur Tel: 60-3-7651-7906</td>
<td>Germany - Karlsruhe Tel: 49-721-625370</td>
</tr>
<tr>
<td></td>
<td>China - Nanjing Tel: 86-25-8473-2460</td>
<td>Malaysia - Penang Tel: 60-4-227-8870</td>
<td>Germany - Munich Tel: 49-89-627-144-0</td>
</tr>
<tr>
<td></td>
<td>China - Qingdao Tel: 86-632-8502-7355</td>
<td>Philippines - Manila Tel: 63-2-634-9065</td>
<td>Germany - Rosenheim Tel: 49-8031-354-560</td>
</tr>
<tr>
<td></td>
<td>China - Shanghai Tel: 86-21-3326-8000</td>
<td>Singapore Tel: 65-6334-8870</td>
<td>Israel - Ra’anana Tel: 972-9-744-7705</td>
</tr>
<tr>
<td></td>
<td>China - Shenyang Tel: 86-24-2334-2829</td>
<td>Taiwan - Hsin Chu Tel: 886-3-577-8366</td>
<td>Italy - Milan Tel: 39-0331-742611</td>
</tr>
<tr>
<td></td>
<td>China - Shenzhen Tel: 86-755-8864-2200</td>
<td>Taiwan - Kaohsiung Tel: 886-7-213-7830</td>
<td>Taiwan - Padova Tel: 39-049-7625286</td>
</tr>
<tr>
<td></td>
<td>China - Suzhou Tel: 86-186-6233-1526</td>
<td>Taiwan - Taipei Tel: 886-2-2508-8600</td>
<td>Netherlands - Drunen Tel: 31-416-690399</td>
</tr>
<tr>
<td></td>
<td>China - Wuhan Tel: 86-27-5980-5300</td>
<td>Thailand - Bangkok Tel: 66-2-694-1351</td>
<td>Norway - Trondheim Tel: 47-7289-7561</td>
</tr>
<tr>
<td></td>
<td>China - Xinan Tel: 86-29-9833-2752</td>
<td>Vietnam - Ho Chi Minh Tel: 84-29-5449-2100</td>
<td>Poland - Warsaw Tel: 48-22-3325737</td>
</tr>
<tr>
<td></td>
<td>China - Xiyang Tel: 86-592-2388138</td>
<td></td>
<td>Romania - Bucharest Tel: 40-21-407-87-50</td>
</tr>
<tr>
<td></td>
<td>China - Zhuhai Tel: 86-756-3210040</td>
<td></td>
<td>Spain - Madrid Tel: 34-91-708-08-90</td>
</tr>
</tbody>
</table>

© 2018 Microchip Technology Inc.