TB3186

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.

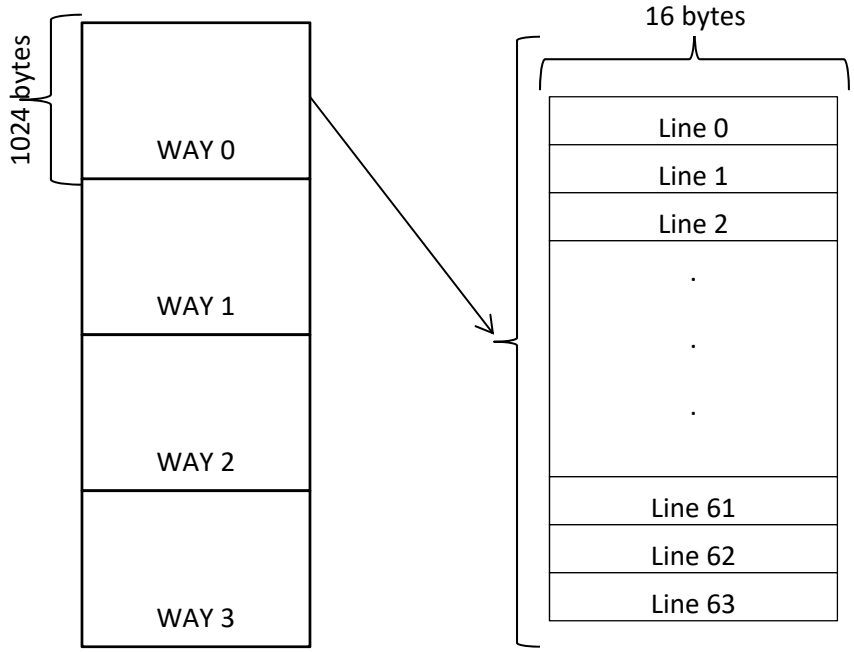
The effective way to achieve deterministic code performance of critical code is to execute the code from the Tightly Coupled Memory (TCM) to the processor, and avoid cache miss conditions. The ARM®Cortex®-M Cache Controller (CMCC) peripheral on Microchip's Cortex-M4 based microcontrollers (MCUs) provides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

Table of Contents

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

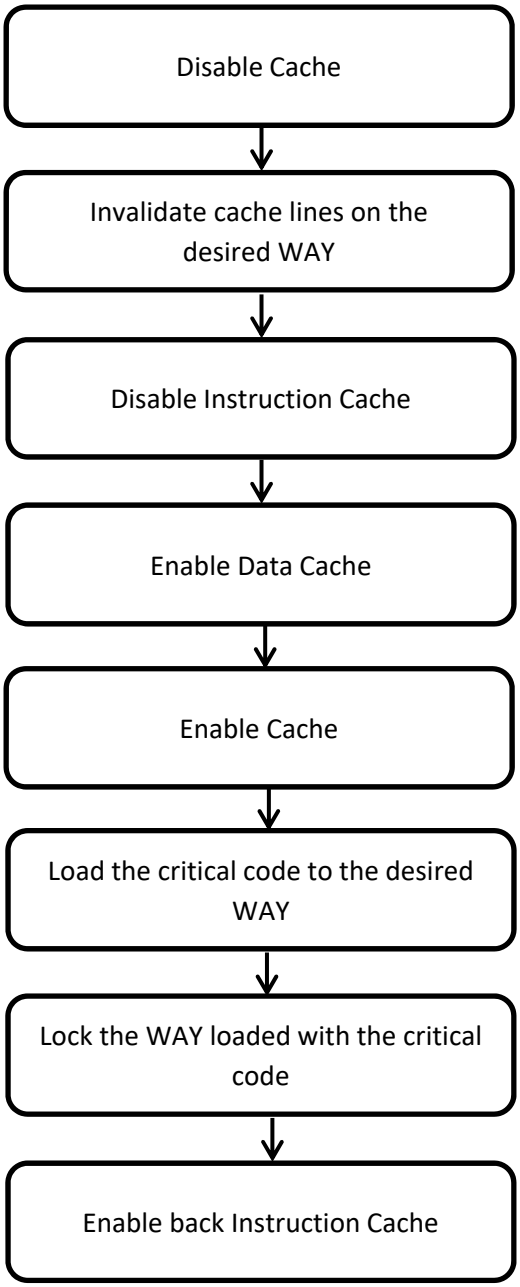


With CMCC, a part of the cache can be used as TCM for deterministic code performance by loading the critical code in a *WAY* and locking it. When a particular *WAY* is locked, the CMCC does not use the locked *WAY* for routine cache transactions. The locked cache *WAY* with the loaded critical code acts 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, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A };
```

Figure 2-3. Implementation of cmcc_loadnlock Function

```
void cmcc_loadnlock(uint32_t way_bitfield, void (*f)(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 indx = 0; indx < CMCC_NO_OF_WAYS; indx++) {
        if(way_index != indx) {
            /* 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;

    /* 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              ((Cmcc     *)0x41006000UL) /**< \brief (CMCC) APB Base Address */
typedef struct {
  __I  CMCC_TYPE_Type           TYPE;        /**< \brief Offset: 0x00 (R/  32) Cache Type Register */
  __IO CMCC_CFG_Type            CFG;         /**< \brief Offset: 0x04 (R/W 32) Cache Configuration Register */
  __O  CMCC_CTRL_Type           CTRL;        /**< \brief Offset: 0x08 ( /W 32) Cache Control Register */
  __I  CMCC_SR_Type             SR;          /**< \brief Offset: 0x0C (R/  32) Cache Status Register */
  __IO CMCC_LCKWAY_Type         LCKWAY;      /**< \brief Offset: 0x10 (R/W 32) Cache Lock per Way Register */
       RoReg8                   Reserved1[0xC];
  __O  CMCC_MAINT0_Type         MAINT0;      /**< \brief Offset: 0x20 ( /W 32) Cache Maintenance Register 0 */
  __O  CMCC_MAINT1_Type         MAINT1;      /**< \brief Offset: 0x24 ( /W 32) Cache Maintenance Register 1 */
  __IO CMCC_MCFG_Type           MCFG;        /**< \brief Offset: 0x28 (R/W 32) Cache Monitor Configuration Register */
  __IO CMCC_MEN_Type            MEN;         /**< \brief Offset: 0x2C (R/W 32) Cache Monitor Enable Register */
  __O  CMCC_MCTRL_Type          MCTRL;       /**< \brief Offset: 0x30 ( /W 32) Cache Monitor Control Register */
  __I  CMCC_MSR_Type            MSR;         /**< \brief Offset: 0x34 (R/  32) Cache Monitor Status Register */
} 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

```
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_WAYSIZE*). 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

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000017a0  00000000  00000000  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 _cc_function_section 00000010  000017a0  000017a0  000117a0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .relocate     00000014  20000000  000017b0  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .bkupram      00000000  47000000  47000000  00020014  2**0
                  CONTENTS
  4 .qspi         00000000  04000000  04000000  00020014  2**0
                  CONTENTS
  5 .bss          0000004c  20000014  000017c4  00020014  2**2
                  ALLOC
  6 .stack        00010000  20000060  00001810  00020014  2**0
                  ALLOC
```

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.

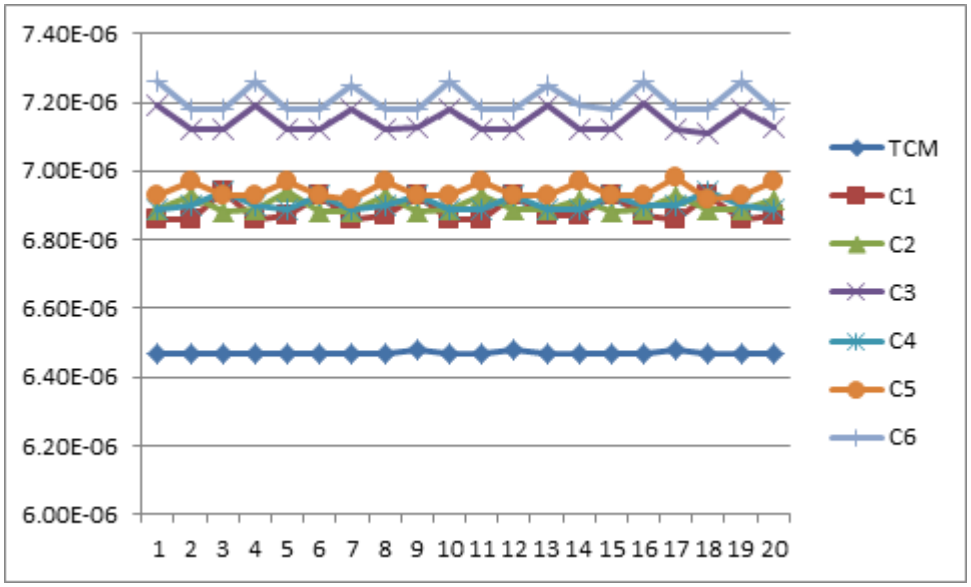
The following is the performance analysis of a simple application when the code is run from TCM (using the `loadnlock` implementation discussed previously) against the regular cache-enabled transactions.

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

```
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:
  http://ww1.microchip.com/downloads/en/DeviceDoc/60001507A.pdf
- SMART SAM E70 TCM Memory:
  http://ww1.microchip.com/downloads/en/AppNotes/atmel-42555-smart-sam-e70-tcm-memory_application%20note_at14971.pdf
- How to Optimize Usage of SAM S70/E70/V7x Architecture:
  http://ww1.microchip.com/downloads/en/appnotes/atmel-44047-cortex-m7-microcontroller-optimize-usage-sam-v71-v70-e70-s70-architecture_application-note.pdf

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.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

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

Trademarks

ISBN: 978-1-5224-2744-5

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, KEELOQ® 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

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820