PIC32CX-BZ3 Programming Specification

PIC32CX-BZ3

Introduction

This programming specification applies to the PIC32CX-BZ3 family of wireless microcontroller and guides the developers of external programming tools. This document also applies to the module WBZ35x as it contains PIC32CX-BZ3.

In the development of a programming tool, it is necessary to understand the internal Flash program operations of the target device and the Special Function Registers (SFRs) that control Flash programming, as an external programming tool and its software use the same operations and registers. For more details on these operations and control registers, see *Program Flash Memory* in the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

The programming tool is responsible for executing the necessary programming steps and completing the programming operation by sending Device Service Unit (DSU) commands over the Serial Wire Debug (SWD) interface of the PIC32CX-BZ3.

Table of Contents

1. Quick References

1.1 Reference Documentation

The following documents are referenced in this document for general understanding or detailed information:

- *ARM® v7-M Architecture Reference Manual* – ARM DDI 0403E.c
- *ARM Debug Interface v5 Architecture Specification* – IHI0031A or higher
- *ARM Debug Interface v5 Architecture Specification ADIv5.1 Supplement* – DSA09-PRDC-008772 1.0
- *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541)

ROM Tables Handling

- *ARM CoreSight™ Architecture Specification v2.0* – ARM IHI 0029D

ETM/ETB Trace

- *CoreSight ETM™ M4 Technical Reference Manual* – ARM DDI 0440C

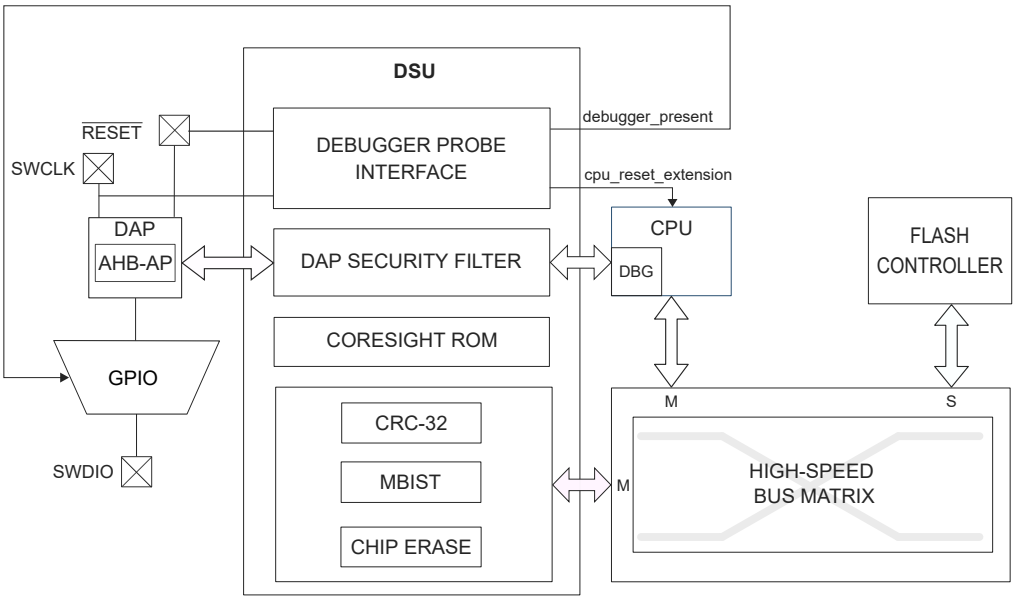
2. Overview

2.1 Architecture

The PIC32CX-BZ3 is based on an ARM Cortex® M4F core with a CPU clock running up to 64 MHz. The Flash Controller (FC) abstracts the Flash topology for programming the single internal embedded Flash. The Device Service Unit (DSU) is the programming interface, with the ARM 2-wire SWD interface.

The Device Service Unit (DSU) module on the device provides a means to detect debugger probes. This enables the ARM Debug Access Port (DAP) to have control over multiplexed debug pads and CPU Reset. Serial Wire Debug (SWD) is a two-wire protocol for accessing the ARM debug interface (Refer to *ARM Debug Interface Specification v5*).

Figure 2-1. DSU Block Diagram



2.2 Device Service Unit (DSU)

The DSU provides basic services to allow on-chip debug using the ARM Debug Access Port (DAP) and the ARM processor debug resources:

- CPU Reset extension
- Debugger probe detection (Cold- and Hot-Plugging)
- Chip-Erase command and status
- ARM CoreSight-compliant device identification
- Debug access port security filter (code protection feature)

2.3 Programming Interface

This device uses the 2-wire Serial Wire Debug (SWD) for all user debug, programming and emulation features. For more information on the SWD interface, refer to the *ARM Core-sight Documentation*.

The SWD consists of two lines:

- SWDIO – A bidirectional data line
- SWCLK – A clock driven by the debugger/programmer

When designing the debug header, it is common to also include the RESET pin ($\overline{\text{MCLR}}$ pin) and one pin for sensing the target board voltage in addition to ground.

2.4 Memory

SRAM memories are accessed at their address in a memory space for both reads and writes at any granularity (byte, half word or word). No special controller is needed to access the SRAM memory.

Flash memories are accessed for read at their address in memory space. Read granularity is byte, half word or word. FC erases and writes the Flash memories. The Flash memory supports quad-word read (128-bit) and single-word write (32-bit). The page size is 4 KB with 256 words per row and four rows per page.

The FC supports erase for the entire Program Flash Memory (PFM) or page erase for the unprotected pages. It supports the Single Word Program (32-bit), Quad Word Program (128-bit) and row programming with built-in DMA for reading data from SRAM with a four- or eight-word buffer.

In addition, there are 32 KB Flash NVR pages (Boot Flash Memory (BFM)) separate from the Program Flash Memory (PFM) partition, with eight pages (0 to 7) where:

- One page is allocated to Factory Test Memory
- One page is allocated to Cal Space (called OTP page)
- One page is allocated to Boot/Device Configuration (BCFG)
- Five pages is allocated to user Boot Code

Each NVR page has write-protect capability. Factory Test Memory is not programmable by the user.

The device memory supports a partial write feature, but using partial write-only that are '1' can be set to '0'. The user can erase the entire Flash memory except the OTP and Factory test memory pages using a dedicated FC command.

Refer to the device data sheet for Flash and SRAM memory size and the memory map of a specific device variant.

2.5 Secure Boot ROM and eFuse

The PIC32CX-BZ3 family of devices has ROM memory dedicated for the secure boot firmware for the root of trust. On a Power-on Reset (POR), secure boot firmware authenticates the rest of the program Flash memory. The secure boot code is already programmed in Microchip Silicon production on the Masked ROM.

Some of the immutable data that cannot be modified after they are determined and programmed include: secure boot key, SECCFG, UUID and Device ID. Immutable data are stored in non-erasable storage called eFuse memory. For more details, refer to the eFuse memory section in the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

2.6 Boot, BCFG and OTP Pages (BFM)

Programming tools support reading and writing of NVR pages (BFM), as they contain critical parameters for the device to run in the user-defined conditions, as well as to boot code. For more details on each NVR page content, refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

One page in the BFM region is implemented as a one-time programmable called OTP. The user can only write and cannot erase by page or chip erase. This region stores the user system calibration data that must survive a Flash erase. The user can use OTP to store and preserve identification values such as Bluetooth/Zigbee MAC address, OEM/ODM version numbers, board version numbers and so on. The other uses are for calibration values of circuits external to the chip but constant for a board design where the chip exists.

There is no difference in the programming algorithm for the BFM pages from programming the Program Flash Memory (PFM) pages except that the DSU.TESTMODE[3] but must be set to '1'. The user needs to manipulate the differences in the memory protection registers.

2.7 Code Protection and Secured Device

There are two protection mechanisms in the PIC32CX-BZ3 devices. One is code protection and the other is secured device and are described as follows:

1. When the code protection is enabled, the device is locked from programming and debugging. Only chip erase can retrieve the device to normal programming and debugging condition.
2. DEBUG_LCK bits in SECCFG register in the root of trust determines if the device is locked for debug. If the DEBUG_LCK bits are non-zero, the device is a secured device. Securing of the device implies:
   a. No unauthenticated firmware can be executed.
   b. The debug features of the device are not available and are locked down.
   c. Device programming through SWD is available. The debugger can be plugged in only through the cold-plugging procedure. The hot plugging feature is not available. (Refer to the DSU section of device data sheet for cold and hot plugging.)
   d. The DEBUG_LCK bits are in eFuse (one time programmable memory); therefore, once locked, the device is permanently locked for debug, unlike the code protection mechanism, which can be cleared on a chip erase.

2.8 Memory Map

The programming tools used refer to the `CMSIS *.SVD` files provided for each part or device data sheet. The CMSIS file contains the most up-to-date information on the device.

3. Device Reset

This section describes the device's available Reset sources. For more detailed information, refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

3.1 System Reset

CRU, RSWRST register and SWRST bit (RSWRST[1]) reset the software. This register is in the CRU at address 0x4400_0A00. A write of a logic '1' to this bit enables the software Reset. A subsequent read of this register triggers the system Reset sequence. The system unlock sequence must be done before the bit can be written. The system unlock register belongs to the CFG register CFG_SYSKEY at address 0x4400_00B0.

Run the following sequencer to unlock:

1. Set CFG_SYSKEY=0x00000000 (reset key)
2. Set CFG_SYSKEY=0xAA996655
3. Set CFG_SYSKEY=0x556699AA

3.2 Vector Catch

The vector catch feature allows the CPU to halt after a Reset or other exception. For more details, see the *ARMv7-M Architecture Reference Manual*.

3.3 External Reset

A debugger can pull the external Reset pin low to reset the device. (The $\overline{\text{MCLR}}$ pin of the device must be pulled to a logic low level for 2 µS minimum pulse width).
Note: The SWCLK pins must not be pulled low by the debugger when the $\overline{\text{MCLR}}$ pin rises; it triggers a CPU Reset Extension (see 3.4. CPU Reset Extension).

3.4 CPU Reset Extension

The DSU module on the device has the CPU Reset extension feature that allows it to connect a debugger while the CPU is on Reset. This is also known as cold-plugging or, also, a core-hold Reset. Refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

This feature requires the debugger to control the level of SWCLK and $\overline{\text{MCLR}}$ pins. The CPU Reset extension is detected on an MCLR release event when SWCLK is low.

The main usage is to avoid executing potentially corrupted code at boot and allow a safe way to connect to a device without any assumption on its current state. The debugger can read/write memory and peripherals without the CPU interference.

Note: After the initial connect sequence, the debugger must release the CPU Reset extension to write to memory located outside the CPU (in other words, SRAM, Flash and peripherals) using MEM-AP. When the CPU is held in the Reset extension phase, the CPU Reset Extension (bit1) of the Status A register (STATUSA.CRSTEXT) @0x4100_0101 of the DSU module is set. To release the CPU, write a '1' to STATUSA.CRSTEXT. STATUSA.CRSTEXT will, then, be set to '0'. Writing a '0' to STATUSA.CRSTEXT has no effect. To keep the CPU from executing the user code, the debugger can rely on the Reset vector catch.

3.5 CPU Boot

At Reset, the Cortex-M4 processors always boot from a vector table at address zero. The processor reads the Main Stack Pointer value from address 0x0 and code entry point (Reset vector) from address 0x4. In the PIC32CX-BZ3 family of devices, the 0x0 address points to the secure boot ROM where the CPU starts after Reset. The Reset handler and interrupt vectors are stored here.

After the boot code is done running and the secure boot code is done with the authentication procedure, the program jumps to the application code image in the Flash section. The stack pointer

and Reset vector are permanent in the boot ROM and will be pointing to the address in ROM code itself. Therefore, the debugger does not need to write the application Flash image-specific vector into these addresses.

4. Serial Wire Debug Port (SW-DP)

This section describes accessing the Serial Wire Debug Port (SW-DP):

- SWD version 1 protocol is used (refer to *DSA09-PRDC-008772*)
- *ARM Debug Interface v5 Architecture Specification* (ADIv5 revision)

4.1 Overview

Out of Reset, both SWDIO and SWCLK are connected to SW-DP. The serial wire debug protocol, as described in the *ARM Debug Interface v5 Architecture Specification* (ARM IHI 0031 Revision A or higher), must be followed to communicate with the device.

4.2 Operation Sequences

4.2.1 Initialize SWD

There are no additional steps apart from what is described in the *ARM IHI 0031*. The SW-DP starts in the SWD mode and does not require a switching sequence, such as JTAG to SWD.

4.2.2 Uninitialize SWD

There are no additional steps apart from what is described in the *ARM IHI 0031*.

5. Access Port(s) (MEM-AP)

This section describes the access port used to access the device memory. When access to SW-DP is enabled, the debugger can access the MEM-AP. When the device security bit is cleared, two MEM-AP (AP0 and AP1) are accessible. The second AP may be used for non-intrusive debugging.

Even if it is not mandatory for this device, before accessing a MEM-AP, the debugger can wait for the CSW.DeviceEn bit to be set. (Refer to *ARM IHI 0031*). A CoreSight ROM table is attached to each AP, describing the available debug components. Alternatively, the debugger can rely on the DSU device ID.

5.1 Implementation Defined Bits

MEM-AP Control/Status Word Register(CSW) protection bits [30:24] are implementation-defined by Microchip. 0x23 needs to be used for these bits [30:24]. The lower two bits correspond to HPROT lines on the AHB bus and 0x23 gives maximum access, which is useful for a debugger/programmer.

5.2 Device Protected State

When the device is locked through code protection, MEM-AP access range is limited to DSU external range. For more details on Intellectual Property Protection, refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541). When the device debug is locked through a secure device apart from DSU external registers, Flash Controller registers and Flash Memory addresses are placed under a MEM-AP-permissible address range to allow programming. The debugger must first read the status bit to handle this case. See 6.7.1. Chip Erase for unprotecting the device if the device is protected through code protection.

Note: DSU external range (0x100-0x2000) is denoted as **DSUEXT** in this document.

5.3 Access Primitives

Based on MEM-AP access, the host (debugger PC utility) can rely on the following primitives:

Refer to *ARM IHI 0031* for MEM-AP related sequence of transactions to read or write the memory.

Table 5-1. Access Primitives

Memory Access Primitive	Description
Status ReadD8(U32 Addr, Value)	Read a byte (8 bits) from device memory at address Addr Returns OK, FAULT, WAIT or ERROR
Status ReadD16(U32 Addr,Value)	Read half-word (16 bits) from device memory at address Addr Returns OK, FAULT, WAIT or ERROR
Status ReadD32(U32 Addr,Value)	Read a word (32 bits) from device memory at address Addr Returns OK, FAULT, WAIT or ERROR
Status WriteD8(U32 Addr,Value)	Write a byte (8 bits) to device memory at address Addr Returns OK, FAULT, WAIT or ERROR
Status WriteD16(U32 Addr,Value)	Write half-word (16 bits) to device memory at address Addr Returns OK, FAULT, WAIT or ERROR
Status WriteD32(U32 Addr,Value)	Write a word (32 bits) to device memory at address Addr Returns OK, FAULT, WAIT or ERROR

Note: The user can use the speed-optimized primitives for block access using the auto-increment feature of the MEM-AP, but they are not mandatory to support all the device features. Refer *ARM IHI 0031*.

The rest of the document relies on this MEM-AP layer and the above-defined primitives to describe the interaction between the device and debugger.

6. Programming

This section describes the MEM-AP sequences required to program the areas of memory using the programmer/debugger.

6.1 Operation Sequences

Following is the SW-DP sequence:

1. Perform a line Reset (refer to 3. Device Reset).
2. Enter into the SW-DP mode by following initialization (see SW-DP section).

Once access to SW-DP is enabled, the debugger can access the MEM-AP.

Following is the MEM-AP sequence:

1. Get Device ID.
2. Get code protection and secure device status.
3. Read memory (each area of memory (SRAM, peripheral registers, Flash, eFuse) requires programming operations to access).
4. Write memory (each area of memory (SRAM, peripheral registers, Flash, eFuse) requires programming operations to access).
5. Read Debug Halting Control and Status Register (DHCRS).

6.2 Get Device ID

The code is equivalent to reading a word at the address DSUEXT.DID at (0x41000118) (DSU_DID register). The value is read from the DSUEXT range; therefore, this register is always accessible even on protected and secured devices. Each PIC32CX-BZ3 variant of the device is identified with a unique Device ID. Refer to the device data sheet for the specific DID value for the specific device variant.

Notes:

1. It is recommended that only the lower 16 bits of this register be used to identify the device.
2. The DSUEXT range starts at offset 0x0100.

Table 6-1. Reference

Steps	mem_ap primitive
Read DSU DID register	ReadD32(DSUEXT.DID)

6.3 Get Device Protection State

As described in 2.7. Code Protection and Secured Device, if the device is code protected or a secured device or both, there is a limitation to programming and debugging functionality and accessing the memory.

1. Read the DSUEXT.STATUB (@0x41000102) (DSU_STATUSB register).
2. Check the PROT[bit 0] bit set for code protection enabled.
3. Read the DSUEXT.SECCFG (@0x4100012C) (DSU_SECCFG register).
4. Check for DEBUG_LCK [7:6] bits for a non-zero value if set for secured device.

Note: Similarly, some of the eFuse content as listed below are also mapped to DSU registers to be accessible when the device is a secured device. For more details on address mapping, refer to the DSU section in the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541).

- **Unique Identifier**: 128 bits. Unique identifier of the device.
- **Secure Configuration bits**: 32 bits. All the security related configurations are a part of this register.

- **Roll back Counter**: 8 bits. Roll back counter associated with the current firmware image.
- **Boot Status**: 8 bits. This indicates the status of the previous boot. Firmware dictates the codes and their corresponding status message. An example can be value `8'hF0` may indicate authentication failure during the previous boot.
- **Life Cycle Counter**: 5 bits. Current stage of the device in its life cycle.
- **Boot Key**: 384 bits. Public boot or OTA authentication key.

Table 6-2. Reference

Steps	mem_ap primitive
Read DSU STATUSB register	ReadD32(DSUEXT.STATUSB)
Read DSU SECCFG register	ReadD32(DSUEXT.SECCFG)

6.4 Read SRAM and Peripherals Registers (SFR)

Reading from SRAM or peripherals requires no specific handling. The MEM-AP sequence of commands are followed.

Note: MEM-AP transactions may be handled differently than CPU transactions to provide non-intrusive debugging.

Table 6-3. Reference

Steps	mem_ap primitive
Read 8 bits item from address	ReadD8(Addr,Value)
Read 16 bits item from address	ReadD16(Addr,Value)
Read 32 bits item from address	ReadD32(Addr,Value)

6.5 Write SRAM and Peripherals Registers (SFR)

Writing to SRAM or peripherals requires no specific handling.

See the Note in 3.4. CPU Reset Extension.

Table 6-4. Reference

Steps	mem_ap primitive
Write 8-bit item from address	WriteD8(Addr,Value)
Write 16-bit item from address	WriteD16(Addr,Value)
Write 32-bit item from address	WriteD32(Addr,Value)

6.6 Read Debug Halting Control and Status Register (DHCRS)

Reading DHCRS requires no specific handling. The code is just equivalent to reading a word at the address 0xE000EDF0 (System Control Block DHCSR register).

Table 6-5. Reference

Steps	mem_ap primitive
Read 32-bit item from address	ReadD32(Addr)

6.7 Flash Memory Programming

The Flash Controller (FC) works with a single panel made from 4 KB pages, with each page containing four rows of Flash data. A row is the largest selectable region for contiguous programming of write words.

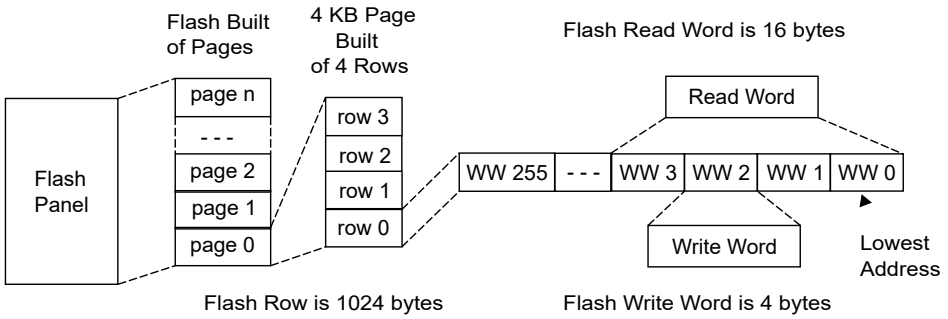
The FC state machine accepts commands from the NVMCON register NVMOP commands (NVMCON.NVMOP).

A page of Flash is the smallest unit of memory that can be erased in a single operation. The whole Program Flash Memory (PFM) region can be erased in a single operation.

The FC supports the following functions:

- Row Programming - Word-by-word programming until the whole row is programmed. The FC reads the data from the system SRAM
- Quad Word Programming - The FC performs four writes of data from holding registers
- Single Word Programming - The FC performs one write of data from holding registers
- Page Erase

Figure 6-1. 128-Bit-Wide Flash Module Application View



The following procedures must be used to program the device Flash using DSU. Programming the Flash is only possible when the device is not code protected. If the device is already code protected, issue the chip erase procedure to make the device unprotected and programming ready.

Programming whole Flash (PFM and BFM regions or only PFM regions):

1. Issue the cold plugging procedure to enter into the DAP mode.
2. Issue a chip erase procedure (only if device is code protected as described in 6.7.1. Chip Erase).
3. This step is needed only if programming the BFM memory region.
   a. Perform the system unlock sequence and write to the NVMLBWP register (0x440006F0)= 0x80000000.
   b. Clear LBWPn bits of NVMLBWP to remove the write protect on the corresponding page.
      Note: At Reset, all LBWPn bits of NVMLBWP are set to logic '1', write protecting all user accessible NVR pages (BFM).
   c. Set DSUEXT.TESTMODE (@0x410001FC) bit 3 to '1'.
4. Perform the procedures mentioned in 6.7.2.2. Row Programming Sequence.

Programming BCFG single boot page for Configurations:

One page of the BFM boot region, called BCFG, is where the configuration bits are stored. These bits are set with #pragma directives embedded in the source file(s) and can be programmed as mentioned above in programming the whole Flash procedure. It can also be programmed with external IDEs like Microchip's MPLAB® X, using the configuration bits window by writing to a single BCFG page alone as mentioned below.

1. Issue the cold plugging procedure to enter into the DAP mode.
2. Perform the below steps to disable write protection.
   a. Perform the system unlock sequence and write to the NVMLBWP register (0x440006F0)= 0x80000000.
   b. Clear the LBWP5 bit (Page 5) of NVMLBWP to remove the write protect on the corresponding page.

c. Set DSUEXT.TESTMODE (@0x410001FC) bit 3 to '1'.

3. Issue a page erase procedure.
4. Perform the procedures mentioned in 6.7.2.2. Row Programming Sequence.

6.7.1 Chip Erase

Erase the entire Flash (PFM and BFM memory regions) except the OTA BFM page using a DSU command.

When the device is protected, the debugger must reset the device to be detected. This ensures that internal registers are reset after the protected state is removed. The Chip Erase operation is triggered by writing a '1' to the Chip Erase bit in the Control register (DSUEXT.CTRL.CE). This command will be discarded if the DSU is protected by the Peripheral Access Controller (PAC).

The chip erase operation depends on clocks and power management features that can be altered by the CPU. For that reason, it is recommended that a chip erase be issued after a cold-plugging procedure to ensure that the device is in a known and safe state.

The recommended sequence is as follows:

1. Perform the cold plugging procedure (refer to the *Cold Plugging* from DSU section in the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541)). The device, then:
   a. Detects the debugger probe
   b. Holds the CPU in Reset
2. Perform the chip erase command by writing a '1' to DSUEXT.CTRL.CE at (0x41000100 address). The device, then:
   1. Clears the system volatile memories
   2. Erases the whole Flash array (excluding OTP page)
   3. Erases the code protection row, removing the code protection security bit protection
3. Check for completion by polling DSUEXT.STATUSA.DONE at (0x41000101 address) (read as '1' when completed).
4. Reset the device.

Table 6-6. Reference

Steps	Commands
Clear flags in STATUSA	WriteD8(@DSUEXT.STATUSA.DSU_STATUSA_DONE_MASK)
Issue chip erase	WriteD8(@DSUEXT.CTRL, DSU_CTRL_CE)
Wait until erase is done	ReadD8(@DSUEXT.STATUSA,StatusValue) While ((StatusValue& DSU_STATUSA_DONE) == 0) { ReadD8(@DSUEXT.STATUSA,StatusValue) }

Bitfields Definitions

- DSU_CTRL_CE = 0x10
- DSU_STATUSA_MASK = 0x1F
- DSU_STATUSA_DONE = 0x1

Note: The recommended time-out for waiting for erase completion is 20 seconds.

6.7.2 Programming Sequences

The following tables describe the register addresses and bit fields for the Flash controller.

Table 6-7. Register Addresses for the NVMCON Register

Register	Address
NVMCON	0x4400_0600
NVMCONCLR	0x4400_0604
NVMCONSET	0x4400_0608
NVMKEY	0x4400_0620
NVMADDR	0x4400_0630
NVMDATA0	0x4400_0640
NVMDATA1	0x4400_0650
NVMDATA2	0x4400_0660
NVMDATA3	0x4400_0670
NVMSRCADDR	0x4400_06C0

Table 6-8. Bit Fields for the NVMCON Register

Bit Field	Bit Mask
WREN_MASK	0x4000
NVMWR_MASK	0x8000
NVMERR_MASK	0x2000
BORERR_MASK	0x1000
NVMOP_MASK	0x000F
WORD_NVMOP	0b0001
QUAD_NVMOP	0b0010
ROW_NVMOP	0b0011
PAGE_ERASE_NVMOP	0b0100
PFM_REGION_ERASE_NVMOP	0b0101

6.7.2.1 Single/Quad Word Programming Sequence

Follow this sequence of steps for single word or quad word programming:

1. Write data to be programmed into the NVMDATA0 register for single word, NVMDATA0-3 for quad word.
2. Load NVMADDR with the address to be programmed.
3. Run the sequencer start unlock sequence using word (quad) program command to start the sequence:
   a. Set NVMCON.WREN=1 (allow writes to NVMCON.NVMWR) and set NVMCON.NVMOP to the desired operation WORD_NVMOP or QUAD_NVMOP (using a single write).
   b. Unlock the sequence to be followed for the programming to take place.
      i. Set NVMKEY = 0x00000000 (Reset key)
      ii. Set NVMKEY = 0xAA996655
      iii. Set NVMKEY = 0x556699AA
   c. Write to the target register NVMCONSET to set the NVMWR bit (not the NVMCON register itself). This starts the FC operation.

   Note: The program sequence completes when the hardware clears the NVMCON.NVMWR bit. This pulses a Flash event.
4. Clear the NMVCON.WREN bit.
5. Check the NVMCON.NVMERR and NVMCON.BORERR bits to ensure that the programming is successful.

6.7.2.2 Row Programming Sequence

The largest block of data that can be programmed by a single NVMOP command is one row. A row is 1024 bytes of data. NVMADDR is the row-aligned address where the Flash address starts programming the data. The controller ignores the sub-row address bits and always starts programming at the beginning of a row.

Follow this sequence of steps for row programming:

1. Write the entire row of data to be programmed into the system SRAM. The source address must be word-aligned but is otherwise unrestricted.
2. Set NVMADDR with the start address of the Flash row to be programmed.
3. Set NVMSRCADDR with the 32-bit physical source address from step 1.
4. Run the following sequencer start unlock sequence using the row program command to start the sequence:
   a. Set NVMCON.WREN = 1 (allow writes to NVMCON.NVMWR) and set NVMCON.NVMOP to ROW_NVMOP (using a single write)
   b. Set NVMKEY = 0x00000000 (reset key)
   c. Set NVMKEY = 0xAA996655
   d. Set NVMKEY = 0x556699AA
   e. Write to the target register NVMCONSET to set the NVMWR bit (not the NVMCON register itself). This starts the FC operation.

   Note: The program sequence completes when the hardware clears the NVMCON.NVMWR bit. This pulses the Flash event.
5. Clear the NVMCON.NVMWREN bit.
6. Check the NVMCON.NVMERR and NVMCON.BORERR bits to ensure that the programming is successful.

6.7.3 Erase Sequences

6.7.3.1 Page Erase

A page erase performs an erase of a single page of main program (PFM), boot/configuration (BFM). The page to be erased is selected using NVMADDR. The lower bits of the address given by NVMADDR are ignored in page selection. A page of Flash can be erased if its associated page write protection is not enabled.

Follow this sequence of steps for page erase:

1. Set NVMADDR with the address of the page to be erased.
2. Run the following unlock sequence using the page erase command to start the sequence:
   1. Set NVMCON.WREN = 1 (allow writes to NVMCON.NVMWR) and set NVMCON.NVMOP to PAGE_ERASE_NVMOP (using a single write)
   2. Set NVMKEY = 0x00000000 (reset key)
   3. Set NVMKEY = 0xAA996655
   4. Set NVMKEY = 0x556699AA
   5. Write to the target register NVMCONSET to set the NVMWR bit (not the NVMCON register itself). This begins the FC operation.

   Note: The erase sequence completes when the hardware clears the NVMCON.NVMWR bit. This generates an interrupt.
3. Clear the NVMCON.NVMWREN bit.

4. Check the NVMCON.NVMERR and NVMCON.BORERR bits to ensure that the erase sequence is completed successfully.

6.7.3.2 PFM Region Erase

The whole PFM can be erased using the PFM_REGION_ERASE NVMOP command. This command leaves the boot or config Flash intact. The whole PFM region can be erased if its associated page write protection is not enabled.

Follow this sequence of steps for the PFM region erase:

1. Run the following unlock sequence using the PFM erase command to start the sequence.
    1. Set NVMCON.WREN = 1 (allow writes to NVMCON.NVMWR) and set NVMCON.NVMOP to PFM_REGION_ERASE (using a single write)
    2. Set NVMKEY = 0x00000000 (Reset key)
    3. Set NVMKEY = 0xAA996655
    4. Set NVMKEY = 0x556699AA
    5. Write to the target register NVMCONSET to set the NVMWR bit (not the NVMCON register itself). This begins the FC operation.

    Note: The erase sequence completes when the hardware clears the NVMCON.NVMWR bit. This generates an interrupt.
2. Clear the NVMCON.NVMWREN bit.
3. Check the NVMCON.NVMERR and NVMCON.BORERR bits to ensure that the erase sequence is completed successfully.

6.8 eFuse Memory Programming

The secure boot keys and credentials required for code authentication are stored in eFuses. There are three blocks of 128x8 bits one-time-programmable eFuses available in the PIC32CX-BZ3 family of devices.

The default eFuse bit value is '0'. The eFuse controller is required to program the bits to change from '0' to '1'. The programming and reading of the eFuse is done through the eFuse controller.

Table 6-9. Register Addresses for the eFuse Controller

Register	Addresses
HLD_REG	From 0x44002C00 to 0x44002DFF
EFUSE_RWDATA	0x44003804
EFUSE_CON	0x44003808

Table 6-10. EFUSE_CON Register Bit Definitions

Bitfields	Bit Definitions
PGM_1BIT	0x1
PGM_MOD	0x2
EN_PGM	0x80
EN_LD	0x20
EN_LD_ALL	0x40
EN_OTP_LDO	0x10000

6.8.1 eFuse Programming Sequence

For eFuse programming operations, the eFuse controller will program 1 bit or 8 bits together at a time.

1. Write EFUSE_CON.PGM_MODE = 0.

2. Write EFUSE_CON.EN_LD = 0 and EFUSE_CON.EN_LD_ALL = 0.
3. Write EFUSE_CON.EN_OTP_LDO = 1. This enables the PMU OTP LDO output to 1.5V.
4. Write the EFUSE_RWDATA register with the offset address of eFuse and the data to be written.
   a. For single-bit write operation: Write eFuse offset address on EFUSE_RWDATA.ADDR[11:0] and data on EFUSE_RWDATA.DATA[0].
   b. For 8-bit write operation: Write eFuse offset address on EFUSE_RWDATA.ADDR[11:3] and data on EFUSE_RWDATA.DATA[7:0].
      The above options are controlled by the EFUSE_CON.PGM_1BIT register bit as in Step 5.2.
5. All the below steps must be done in a single step:
   a. Set the EFUSE_CON.PGM_MODE register bit.
   b. Set the EFUSE_CON. PGM_1BIT register bit for a 1-bit write operation. Clear the bit for the 8-bit write operation.
   c. Set the EFUSE_CON.EN_PGM register bit.
6. The eFuse controller clears the EFUSE_CON.EN_PGM register bit when the eFuse programming is complete.
7. Repeat the steps 4 to 5 until all the values are programmed.
8. Write EFUSE_CON.EN_OTP_LDO = 0 to switch off PMU OTP LDO.

6.8.2 eFuse Reading Sequence

The read from the eFuse panel happens at boot time and the holding registers are loaded with the eFuse values. Therefore, the memory map of the eFuse is in 1:1 correspondence with the memory map of the holding registers. If required to verify the values in eFuse, reset the device after the eFuse programming sequence is done. The modified eFuse values are read from the eFuse and loaded on holding registers at boot-up.

Read the holding registers in the same way as reading the peripheral registers, and, then, verify these registers.

6.9 Configuring Code Protection

The CP bit (Bit 29) of FCPN0 in BCFG NVR page determines the code protection. When the CP bit is set, the device is locked from programming and debugging. Only the chip erase can return the device to the normal programming and debugging condition.

As this code protection is part of the BCFG NVR Flash page, the steps mentioned in "Programming whole flash (PFM and BFM regions or only PFM regions)" can be followed to enable Code Protection.

6.10 Configuring Debug Lock

The DEBUG_LCK[1:0] bits in FSECCFG[7:6] eFuse at 0x0000 offset address determine if the device is locked for debug. If the DEBUG_LCK[1:0] bits are non-zero, the device is a secured device. On a secured device, programming is still possible but debugging is completely locked because these bits are in eFuse.

Follow the procedure mentioned in 6.8.1. eFuse Programming Sequence with the eFuse offset address at 0x0000 to write these bits.

7. In-Circuit Debugging

7.1 Debug State Side Effects

The following are side effects of the debug state:

- The device debug mode is entered as soon as the SW-DP CTRLSTAT.CDBGPWRUPREQ bit (Refer to *ARM Debug Interface v5 Architecture Specification*) is set (and acknowledged by the system when the start-up sequence is complete). In this device debug mode:
  - Sleep modes behavior is changed to allow debugging
  - Debug clock is running
  - Internal clock requests are altered
- Peripherals behavior on CPU halt is controlled by DBGCTRL registers in the device's individual peripheral module.

7.2 Debug Logic Reset

- CPU debug logic is reset by any Reset source, except SysResetReq.
- Peripherals DBGCTRL registers are reset by any Reset source, except SysResetReq.

7.3 Debug and Trace Support

- The Data Watchdog and Trace (DWT) unit provides four data watchpoint comparators and execution monitoring.
- The Flash Patch and Breakpoint (FPB) unit provides up to eight hardware breakpoint comparators that debuggers can use.
- Implements Embedded Trace Buffer (ETB) for execution tracing.

CoreSight ETM-M4 Technical Reference Manual – ARM DDI 0440C provides a detailed description of the ETB or ETM operation. Configuration can be done by the debugger or the CPU but the debugger must know where the trace data will be placed in RAM.

7.4 Debugger Access Support

Debugger access port used to access device memory. When access to SW-DP is enabled, the debugger can access the MEM-AP.

Table 7-1. Debugger Access Support

ARM Debug Mode	JTAG ID
SW-DP (2-wire)	0x0bc11477
JTAG (4-wire)	N/A

7.4.1 Debug and Access Ports (SW-DP And MEM-AP Subblocks)

Identify the Arm Debug Interface v5 Architecture Specification revision (applicable).

ARM does not specify the revision of the specification. The differences between the revisions are minimal; therefore, any recent revision must be applicable.

7.5 Operation Sequences

The following table shows the list of debug registers and their corresponding address from the *ARM Cortex M4 Processor Reference Manual*.

Table 7-2. Debug Registers

Address	Name	Type	Reset	Description
0xE000ED30	DFSR	RW	0x00000000	Debug Fault Status Register Power-on Reset (POR) only
0xE000EDF0	DHCSR	RW	0x00000000	Debug Halting Control and Status Register
0xE000EDF4	DCRSR	WO	—	Debug Core Register Selector Register
0xE000EDF8	DCRDR	RW	—	Debug Core Register Data Register
0xE000EDFC	DEMCR	RW	0x00000000	Debug Exception and Monitor Control Register

The following provides the detailed operation sequence:

- Enter the Debug mode
- Exit the Debug mode
- Register Reads and Writes
- Run
- Halt
- Get Halt status
- Get Core registers
- Get Program Status Register (PSR)
- Get and Set Program Counter
- Get and Set Stack Pointer
- Single Step
- Set Core Registers

7.5.1 Enter the Debug Mode

To force the processor to enter the Debug state as soon as it comes out of Reset, a debugger sets **DHCSR.C_DEBUGEN** to '1' to enable halting debug and sets **DEMCR.VC_CORERESET** to '1' to enable vector catch on the Reset exception.

When the processor comes out of Reset, it sets **DHCSR.C_HALT** to '1' and enters the Debug state.

For more details, refer to the *ARM v7-M Architecture Reference Manual* - ARM DDI 0403E, Debug and Reset.

Table 7-3. Reference

Steps	Commands
Enable Debug	WriteD32(DHCSR, (DBGKEY | C_DEBUGEN))
Enable Reset Vector Catch	WriteD32(DEMCR, VC_CORERESET)
Issue Halt	WriteD32(DHCSR, (DBGKEY | C_HALT))

7.5.2 Exit Debug Mode

The processor exits the Debug state:

- When the debugger writes '0' to DHCSR.C_HALT
- On receipt of an external restart request

If software clears DHCSR.C_HALT to '0' when the processor is in the Debug state, a subsequent read of the DHCSR that returns '1' for both C_HALT and S_HALT indicates that the processor has re-entered the Debug state because it has detected a new debug event.

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, Debug stepping control using the DHCSR table.

Table 7-4. Reference

Steps	Commands
Exit Debug Mode	WriteD32(DHCSR, (DBGKEY | 0))

7.5.3 Register Reads and Writes

This operation is the same as described in 7.5.3. Register Reads and Writes. The MEM-AP has access to the entire device memory map as described in the CMSIS standard SVD (`.atdf`) files.

Table 7-5. Reference

Steps	Commands
Read register	ReadD32(Register, Value)
Write register	WriteD32(Register, Value)

7.5.4 Run

The processor exits the Debug state or goes into the Run state:

- When the debugger writes '0' to DHCSR.C_HALT
- On receipt of an external restart request

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, Debug stepping control using the DHCSR table.

Table 7-6. Reference

Steps	Commands
Exit Debug Mode (Run)	WriteD32(DHCSR, (DBGKEY | 0))

7.5.5 Halt

The Halt state can be reached after the CPU hits a breakpoint.

A debugger can use halting debug stepping to exit from the Debug state, execute a single instruction and, then, re-enter the Debug state. Halting debug stepping is active when all the following apply:

- DHCSR.C_DEBUGEN is set to '1', halting debug enabled
- DHCSR.C_STEP is set to '1', halting stepping enabled
- The processor is in the Non-debug state

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, Debug stepping control using the DHCSR table.

Table 7-7. Reference

Steps	Commands
Debug halt	WriteD32(DHCSR, (DBGKEY | DHCSR_C_DEBUGEN | DHCSR_C_MASKINTS | DHCSR_C_HALT))

7.5.6 Get Halt Status

Check the state of Debug Halting Control and Status Register (DHCSR) Halt Flag.

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, Debug stepping control using the DHCSR table.

Table 7-8. Reference

Steps	Commands
Read Debug Halting Control and Status Register	ReadD32 (DHCSR, Value)
Poll for Halt Status bit set	S_HALT = 1

7.5.7 Get Core Register

To transfer a data word from an ARM core register, special-purpose register or floating-point extension register, a debugger:

- Ensures the CPU is in the Debug state by reading DHCSR.S_HALT bit and places the CPU in the Debug state if this is not the case
- Writes to the DCRSR with the REGSEL value indicating the required register and the REGWnR bit as '0' to indicate a read access. This write clears the DHCSR.S_REGRDY bit to '0'.
- Polls DHCSR until DHCSR.S_REGRDY reads as '1'. This shows that the processor has transferred the value of the selected register to DCRDR.
- Reads the required value from DCRDR

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, C1.6.3 Debug Core Register Selector Register, DCRSR.

Table 7-9. Reference

Steps	Commands
Write REGSEL	WriteD32 (DCRSR, RegSel)
Poll DHCSR for S_REGRDY	S_REGRDY = 1
Read value from DCRDR	ReadD32 (DCRDR, value)

Where RegSel is one of the CPU register:

Register name	CoreReg
R0	0
R1	1
..	..
R12	12
SP	13
LR	14
PC	15
xPSR	16
..	..

Note: For complete table details, refer *to the Arch Manual C1.6.3 Debug Core Register Selector Register, DCRSR*.

7.5.8 Set Core Register

To transfer a data word to an ARM core register, special-purpose register or floating-point extension register, a debugger:

- Ensures the CPU is in the Debug state by reading DHCSR.S_HALT bit and places the CPU in the Debug state if this is not the case
- Writes the required value to DCRDR
- Writes to the DCRSR, with the REGSEL value indicating the required register and the REGWnR bit as '1' to indicate a write access. This write clears the DHCSR.S_REGRDY bit to '0'.

- Polls DHCSR until DHCSR.S_REGRDY reads as '1'. This shows that the processor has transferred the DCRDR value to the selected register

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, C1.6.3 Debug Core Register Selector Register, DCRSR.

Table 7-10. Reference

Steps	Commands
Write value to DCRDR	WriteD32 (DCRDR, value)
Write REGSEL	WriteD32 (DCRSR, RegSel)
Poll DHCSR for S_REGRDY	S_REGRDY = 1

For more details, refer to the RegSel description in the *Arch manual C1.6.4 Debug Core Register Selector Register, DCRSR*.

7.5.9 Get Program Status Register (PSR)

See 7.5.7. Get Core Register with RegSel = xPSR.

7.5.10 Get Program Counter

See 7.5.7. Get Core Register with RegSel = PC.

7.5.11 Set Program Counter

See 7.5.8. Set Core Register with RegSel = PC.

7.5.12 Get Stack Pointer

See 7.5.7. Get Core Register with RegSel = SP.

7.5.13 Set Stack Pointer

See 7.5.8. Set Core Register with RegSel = SP.

7.5.14 Single Step

To single step CPU instructions, a debugger needs to:

- Ensure the CPU is in the Debug state by reading the DHCSR.S_HALT bit and place the CPU in the Debug state if this is not the case
- Write DHCSR = C_MASKINTS | C_STEP | C_DEBUGEN
- Write DHCSR = C_MASKINTS | C_HALT | C_DEBUGEN

For more details, refer to the *ARMv7-M Architecture Reference Manual* - ARM DDI 0403E, C1.5.1 Debug stepping.

Table 7-11. Reference

Steps	Commands
Single step	WriteD32(DHCSR, DBGKEY | C_MASKINTS | C_STEP | C_DEBUGEN)
Ensure halt after step	WriteD32(DHCSR, DBGKEY | C_MASKINTS | C_HALT | C_DEBUGEN)

8. DSU.DID vs SW-DP ID

The SW-DP ID is a 32-bit ID comprised of an 11-bit JEDEC Manufacturer's ID Code (assigned to Microchip), the 20-bit Device ID and a fixed LSB of '1'. The SW-DP ID can be read through the SWD interface using the ARM IDCODE command. The device ID to be used by the external debugger or programmers must be within the DSU [DSU.DID register] located at address 0x4100_00118 (external address). Only the lower 16 bits of this register are used for the purpose of external tools to identify the device; the upper 16 bits must be masked off on read.

Many of the fields in the SW-DP ID and the DSU.DID register are the same but positioned differently.

Refer to the following figure for how the SW-DP ID and DSU.DID are correlated, and refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541) for the specific variant part number values.

The Mask ID for the PIC32CX-BZ3 family of devices is 0x9E, and JEDEC Manufacturer's ID Code for Microchip is 0x029. Refer to the *PIC32CX-BZ3 and WBZ35x Family Data Sheet* (DS70005541), as the Device ID and Revision ID vary from variant to variant.

Figure 8-1. DSU.DID VS SW-DP ID

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SW-DP ID	Revision ID				Mask ID								Device ID			
DSU.DID	Revision ID				Family ID					Reserve	Series ID					

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW-DP ID	Device ID				JEDEC Manufacturer's ID Code											Always '1'
DSU.DID	Mask ID								Device ID							

9. Document Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Section	Description
A	11/2023	Document	Initial revision

Microchip Information

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable". Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications 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. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". 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 ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

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, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, TimeCesium, TimeHub, TimePictra, TimeProvider, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, EyeOpen, GridTime, IdealBridge, IGaT, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, MarginLink, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mSiC, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, Power MOS IV, Power MOS 7, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, Turing, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks 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.

ISBN: 978-1-6683-3410-2

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	**Australia - Sydney**	**India - Bangalore**	**Austria - Wels**
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	**China - Beijing**	**India - New Delhi**	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	**Denmark - Copenhagen**
Fax: 480-792-7277	**China - Chengdu**	**India - Pune**	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	**China - Chongqing**	**Japan - Osaka**	**Finland - Espoo**
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	**China - Dongguan**	**Japan - Tokyo**	**France - Paris**
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	**China - Guangzhou**	**Korea - Daegu**	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	**Germany - Garching**
Fax: 678-957-1455	**China - Hangzhou**	**Korea - Seoul**	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	**Germany - Haan**
Tel: 512-257-3370	**China - Hong Kong SAR**	**Malaysia - Kuala Lumpur**	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	**Germany - Heilbronn**
Westborough, MA	**China - Nanjing**	**Malaysia - Penang**	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	**Germany - Karlsruhe**
Fax: 774-760-0088	**China - Qingdao**	**Philippines - Manila**	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	**Germany - Munich**
Itasca, IL	**China - Shanghai**	**Singapore**	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	**China - Shenyang**	**Taiwan - Hsin Chu**	**Germany - Rosenheim**
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	**China - Shenzhen**	**Taiwan - Kaohsiung**	**Israel - Ra'anana**
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	**China - Suzhou**	**Taiwan - Taipei**	**Italy - Milan**
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi, MI	**China - Wuhan**	**Thailand - Bangkok**	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	**Italy - Padova**
Houston, TX	**China - Xian**	**Vietnam - Ho Chi Minh**	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	**Netherlands - Drunen**
Indianapolis	**China - Xiamen**		Tel: 31-416-690399
Noblesville, IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	**China - Zhuhai**		**Norway - Trondheim**
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			**Poland - Warsaw**
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			**Romania - Bucharest**
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			**Spain - Madrid**
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			**Sweden - Gothenberg**
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			**Sweden - Stockholm**
San Jose, CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			**UK - Wokingham**
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			
Fax: 905-695-2078			