Safeguarding Flash Memory Self-Write Operations

Author: Justin O'Shea
Microchip Technology Inc.

INTRODUCTION

Many microcontrollers provide the ability to reprogram their own Flash memory at run time. This capability allows field updates of program memory through a boot-loader utility and storage of data in nonvolatile memory. The following discussions and examples are based on the dsPIC33 and PIC24 products; however, many of the techniques and considerations can be applied to any microcontroller. Flash program memory can be programmed in one of two ways: In-Circuit Serial Programming™ (ICSP™) with an external hardware programmer and Run-Time Self-Programming (RTSP). The following discussions are targeted for self-erase/write (RTSP), which is implemented in firmware.

The purpose of this application note is to provide guidance and best practices on how to prevent unintended Flash writes or erasures that can cause minor to catastrophic field failures. Adding Flash programming safeguards to the firmware will reduce the risk of problems and ensure robust field updates. The following material increases firmware robustness through understanding of the potential issues and provides methods to prevent them.

UNDERSTANDING POTENTIAL RISKS

In order to protect against unintentional erasure or write corruption, an understanding of how these types of problems can occur in an application is needed. Most issues are typically caused by the following:

- Hardware Issues/Instruction Misexecution
- Operational Specification Violations
- Conceptual and Architectural Weaknesses

This application note will address hardware issues that can lead to out-of-specification operation, and conceptual and architectural issues. Although software/code bugs may also contribute to self-write problems, general good coding practices are beyond the scope of this document.

HARDWARE ISSUES

All electronic designs are susceptible to hardware related issues that can be brought on by a variety of sources. The focus of this discussion is preventing instruction misexecution caused by power supply and clocking circuits.

Instruction Misexecution

Instruction misexecution occurs when signal timing is not met inside the microcontroller core during program instruction fetch from Flash memory. This is most typically caused from a low-voltage condition or an out of specification clock pulse. For example, consider the case when executing the `RETURN` instruction at the end of a common C function. In the dsPIC33 and PIC24 architectures, the `RETURN` instruction is encoded as '`0000 0110 0000 0000 0000 0000`'. If a timing violation occurs during the instruction fetch from Flash memory, the `RETURN` instruction could be misinterpreted as '`0100 0110 0000 0000 0000 0000`'. Although the value varies by only one bit, it profoundly changes the meaning of the instruction into the `ADD` instruction.

In a C program, if an `ADD` or other instruction opcode is executed in place of an intended `RETURN` instruction, the result can be one C function effectively flowing into the next that follows it in Flash memory. As shown in Example 1, unintended code flow scenarios can allow C functions, such as a Flash memory erase function, to get unintentionally called.

In addition to a `RETURN` instruction, `CALL`, `GOTO` or other Program Counter-relative branching instructions also appear in code frequently and can allow unexpected code to be reached. If the target address suffers a bit flip during read from Flash or decode and execution, the Program Counter may still point to implemented Flash memory, but not the intended address.

EXAMPLE 1: CODE FLOW

```
void    ExampleFunction(void)
{
    //...some code here..
}   //Compiler will place a RETURN instruction here.  If this gets mis-fetched, program flow will
      continue to the next flash location which may be the UnlockAndProg() function

void    UnlockAndProg(void)
{
    __builtin_write_NVM();  //Note: This function executes the unlock and program sequence
}
```

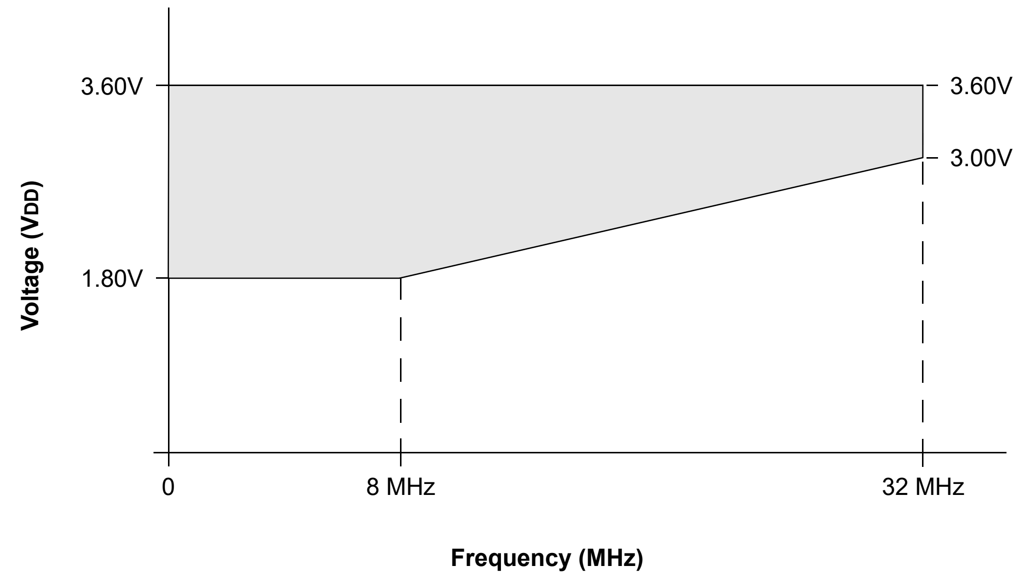
Power Supply Considerations

To avoid instruction misexecution, microcontrollers need to be supplied power within their respective electrical specifications to operate properly. There are many factors that can cause issues, including:

- Mechanical Contact Bounce on Switches/Buttons, Battery Contacts, Relays and Power Cables Interrupting V_{DD}
- Voltage Regulator Tolerances
- Temperature, Environmental and Mechanical Aging Induced Variations
- Poor Circuit Board Layout Practices
- High Supply Noise due to Poor Isolation or EMI Susceptibility

In addition to steady-state (normal) operation, the microcontroller must be able to properly handle power-up and power-down events. These transient conditions may create violations to the voltage and frequency timing requirement associated with it. Figure 1 shows a typical voltage-frequency graph. To operate at a given speed over temperature, a certain supply voltage must be provided. Consider the scenario where the system's power supply has not fully reached its nominal value, but the microcontroller has been configured to run at full speed. This low-voltage condition can lead to signal timing violations in the microcontroller and instruction misexecution. The same scenario can occur during power-down when the microcontroller is at full speed and supply voltage decays.

FIGURE 1: VOLTAGE-FREQUENCY CONSTRAINT



Special care and analysis are recommended for power-up and power-down behavior of the application. The power-up and power-down ramps may be short in human terms (microseconds to milliseconds), but can allow hundreds of thousands of instructions to execute.

Mechanical components, such as switches and connectors, can pose potential issues to the microcontroller's power supply. Contact bounce can force the power-up and power-down ramps to be repeated many times in quick succession for a single intended power-up state if on-board supply or V_{DD} capacitors are unable to sufficiently filter them. Compounding this problem, mechanical systems exhibit quite different electrical properties when new versus after being deployed in the field for several years and actuated a high number of times. If left idle for extended periods of time, surface oxides and contaminants might accumulate and further degrade behavior. When employing mechanical systems intended to disconnect power, capacitors placed after the switching element can reduce V_{DD} instability seen by the microcontroller, but an evaluation should also be done to ensure peak inrush current at switch turn-on does not exceed component ratings or prematurely wear out the contacting element.

BROWN-OUT RESET (BOR)

dsPIC33 and PIC24 microcontrollers feature a BOR circuit to mitigate undervoltage conditions and place the CPU in Reset to prevent unintended actions. While able to guard against typical V_{DD} transients that would otherwise produce instruction misexecution, a BOR circuit should not be regarded as a solution that can protect against everything.

BOR hardware is implemented using analog circuits that must produce their own internal voltage reference using the unknown, and possibly unstable V_{DD}, to power themselves. They also have to balance the needs of precise voltage sensing, fast response time and low quiescent power consumption while the microcontroller is in Sleep mode.

To handle competing requirements, most BOR designs implement multiple modes of operation. For example the BOR holds the system in Reset while V_{DD} is rising, then enters a more precise, faster response mode as V_{DD} approaches the BOR release threshold. It then reverts to a lower current state after the system begins executing and V_{DD} drifts far enough away from the BOR trip point. It may then reverse the state transitions while V_{DD} is decaying or possibly enters an even lower current state for operation during Sleep. The BOR can detect power loss in every state, but as each mode has some settling delay, the exact trigger voltage and response latency may vary if V_{DD} oscillates rapidly up and down through the trip point of the BOR.

Additionally, BOR only guards against low-voltage conditions. It cannot detect or prevent excessively fast clocking configurations from being selected or used near the BOR limit. Power events often coincide with extra or unusual electrical noise being generated in some circuits. For example, if V_{DD} is supplied by a DC-DC Switching Mode Power Supply (SMPS), turning the supply on could entail charging all of the capacitors on V_{DD} over the course of a handful of milliseconds. This implies V_{DD} will be high enough for the BOR to release the system for operation, meanwhile the power supply is still switching at high current levels to charge all of the V_{DD} capacitors. This is an opportunity to execute many thousands of instructions, yet a very noisy and undesirable time to risk letting your RTSP code potentially execute.

Starting the microcontroller at a slower speed (such as FRC) and later clock switching to a high-speed clock can increase robustness by reducing peak current needs at start-up and simultaneously reducing the number of instructions that will execute prior to V_{DD} reaching a steady state.

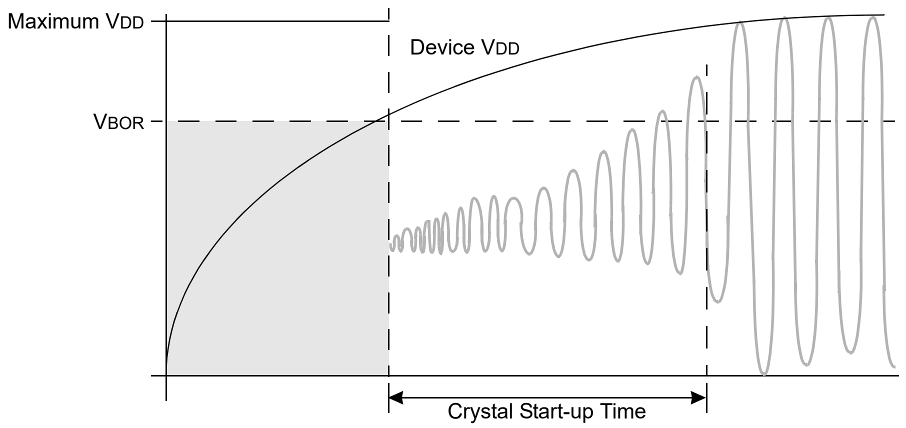
EXTERNAL VOLTAGE SUPERVISOR

An external voltage supervisor provides additional protection against undervoltage and out-of-specification conditions. They sense the power supply voltage and can place the microcontroller in Reset by controlling the $\overline{\text{MCLR}}$ pin. An external supervisor is especially useful for systems that have power supplies that can allow out-of-specification operation, such as voltage dips that do not go all the way to zero volts. When the power supply decays to sub-threshold voltages between 0.1 and 0.7 volts, some BOR and POR circuits internal to the microcontroller may not produce a known output, given insufficient voltage for itself to operate. Some devices specify that this condition is out-of-specification, and require V_{DD} to return to 0V before ramping back up to the minimum V_{DD} level. Refer to the device-specific data sheet's electrical specifications for details and restrictions. This can be especially problematic in battery-powered applications.

Clocking Considerations

A clean and in-specification clock is critical to proper microcontroller operation. Runt pulses, skewed duty cycles, noise and other anomalies that violate the minimum clock width, even for one cycle, can cause instruction misexecution. Oscillator start-up is a special case and should be observed on an oscilloscope over the full temperature range to ensure valid operation. It is possible that the system can start using a clock before it has reached sufficient amplitude and a stable period, as shown in Figure 2. Starting the microcontroller on the internal FRC and then switching to the external oscillator after it has stabilized can help improve robustness. If using a PLL to increase clock speed, ensure it is operating within its constraints and do not modify critical PLL settings while in use. Refer to the device-specific data sheet for details on PLL restrictions.

FIGURE 2: OSCILLATOR START-UP



A typical crystal or ceramic resonator circuit will have semi-sinusoidal oscillator waveforms, which have relatively slow edge slew rates compared to a square wave. These slow rate edges are more vulnerable to noise from external sources, potentially allowing a noise impulse near a $V_{DD}/2$ crossing to manifest as additional unwanted clock edges. Additionally, because the oscillator waveform is essentially sinusoidal, precise duty cycle and edge timing information would be destroyed if the microcontroller converted the analog waveform into a digital square wave clock using a Schmitt Triggered input buffer.

Special care is recommended to ensure that the oscillator input and output are free from externally generated noise sources, such as switching transients from power MOSFETs in a power electronics application. This can be mitigated at the PCB layout level by ensuring that the oscillator nets/traces are kept physically separated from any noise generating traces, and by adding grounded shield traces surrounding the oscillator circuit.

Protecting Against Instruction Misexecution

Below is a summary of the actions that can be taken to protect against instruction misexecution:

1. Start device on FRC and switch to the external oscillator after allowing it to stabilize, or when possible, implement all RTSP operations using the internal FRC or FRC+PLL at a conservative frequency to reduce IDD and increase VDD decay time on power loss.
2. Add a software delay at start-up.
3. Enable BOR. If the voltage level is configurable, set it to a high enough level to provide adequate protection.
4. Ensure bypass capacitors are present and sized appropriately according to the data sheet. Verify that the supply voltage is free from oscillations at power-up, run time and power-down.
5. For devices with a VCAP pin, ensure a properly sized capacitor is present.
6. Use an external voltage supervisor to ensure the microcontroller cannot execute code if VDD is out-of-specification.
7. Ensure PLL initialization and run-time clock switching code is implemented correctly.
8. Use ADC to verify sufficient supply voltage before attempting a program or erase operation.
9. Validate oscillator circuits across temperature on final PCB.
10. If configurable, enable and use the Power-up Timer (PWRT).

CONCEPTUAL AND ARCHITECTURAL CONSIDERATIONS

Application software that can perform RTSP operations should be architected in such a manner to minimize risk of unintended erase/write and Flash memory corruption. The following sections discuss architectural concepts and the methods to add safeguards. Architectural concepts that can make an application less vulnerable to unintended Flash erase/write corruption include:

1. Adding software delay at start-up.
2. Robust bootloader entry.
3. Use dynamic key for hardware unlock.
4. Single instance of unlock sequence.
5. Software unlock sequence.
6. Clearing the write enable bit.
7. Verify VDD is sufficient before erase/write operations.
8. Validating erase/write addresses.
9. Park erase/write addresses to safe location.

Adding Software Delay at Start-up

Some applications need to write data to nonvolatile memory at the beginning of the application, however, doing so can be dangerous and lead to unintended actions. Perhaps the most effective RTSP safeguard is to add a software delay at start-up. As soon as code starts executing out of Reset, regardless of if RTSP operations are planned or not, the right thing to do instead is enter a software delay loop to hold off primary execution for a few 10's or 100's of milliseconds. This delay will allow all other components on the PCB to reach a steady state, including VDD capacitors which should reach their expected operating voltage (ex: 3.3V) instead of their minimum operating voltage (ex: 3.0V) in which the part has just barely exceeded POR/BOR limits.

An additional benefit to ensuring software does nothing for a while after starting, is that it can suppress inadvertent ICSP™ programming failures and apparent RTSP erase/programming failures during development. When first implementing RTSP code, a common thought is to put an NVM erase and write testing routine directly in `main()`, then attempt to do ICSP read-back of Flash to see if the code executed as intended. A race condition can arise in which the MPLAB® X IDE ICSP programming tool reads DEVID, bulk erases the Flash, programs the Hex contents and finally performs read-back verification of Flash. Between the programming and read-back steps, the ICSP tool may release MCLR and allow the RTSP code to execute for a few milliseconds. Without a software delay, this can cause a false read-back verification failure in the IDE.

For software implemented using the MPLAB XC16 C Compiler, a predictable start up delay can be created by calling the `__delay32()` function at the start of the `main()` function. For further information on the `__delay32()` function, refer to the *"16-Bit Language Tools Libraries Reference Manual"* (DS50001456) at www.microchip.com.

If fast start-up is required from Deep Sleep type Reset events which do not relate to power-up/power loss, the `__delay32()` call can be executed conditionally based on the state of the POR or BOR status flags (normally located in the RCON SFR). Typically, these bits must be cleared in software after being tested and will only get reset in hardware when a true power-oriented Reset occurs.

Robust Bootloader Entry

Besides holding off software execution after a power event, robust bootloader entry methods are important for safeguarding an application. Overly simple methods of entry, such as detection of only a single UART byte, are not recommended. A 32-bit or longer detection sequence, containing a mix of '`1`' and '`0`' bits, is much less likely to accidentally decode from random communication noise. It is also recommended to implement a Flash memory unlock command in the bootloading protocol instead of hard coding in application software. The host application should be responsible for not only sending the command to enter Bootloader mode, but also a separate command to unlock erase/write operations before being allowed to send additional commands that actually modify the Flash memory contents.

Dynamic Key for Hardware Unlock

It is recommended to use a dynamic key for the NVM write/erase hardware unlock sequence. The specific key needed to unlock the commands should be passed in from the host and destroyed after it is used to further prevent unintended actions. The typical hardware unlock key for dsPIC33 and PIC24 devices is 0x55/0xAA, and should not be hard coded within the write/erase routine itself. The unlock keys can be stored in RAM during a secure bootloader entry sequence to be used later in a write/erase routine. Once the operation is complete, the keys should be destroyed so no additional write/erase operation can be done until the next secure bootloader entry is attempted. This prevents unintended code flow from performing the unlock sequence.

The XC16 compiler includes some built-in functions for the unlock sequence. These functions write to the NVMKEY register and set the Write bit to initiate a write or erase sequence. Instead of using the `__builtin_write_NVM()` function, it is recommended to use the secure version, `__builtin_write_NVM_secure()`. The secure version supports passing dynamic keys into the function.

Care should also be taken to ensure the compiler does not optimize the code by replacing variables with literal values of 0x55 and 0xAA in the actual instructions stored in Flash. Generally, this is prevented any time the variable's contents are read or derived from a communications SFR and not explicitly specified as any type of constant in code.

Single Instance of Unlock Sequence

dsPIC33 and PIC24 microcontrollers incorporate an unlock sequence before allowing Flash memory erase/write operations. It is possible for the application code to be structured such that the unlock sequence is hard coded and duplicated in multiple different software functions for different types of operations, including page erase, word write and row write.

If dynamic keys are not used, the hardware unlock sequence should only be executed in one location as part of a dedicated function. Since the hardware unlock keying and NVM operation can brick the device if unintentionally executed, code should instead be organized such that the unlock keying and NVM operation start code appear in only a single function and without any duplication anywhere else in the project. This function should receive the NVM operation value as a parameter, such that if the Program Counter ever reaches the unlock keying code accidentally through instruction misexecution, then the parameter will be invalid and no harm will occur. By isolating the unlock sequence code into a single location, only a single set of additional protection codes is needed, reducing vulnerability.

Software Unlock Sequence

The concept of having an unlock key or sequence to prevent unintended actions can also be applied to the application code. Any function that either directly erase/writes Flash memory, or calls a function that does, should have an unlock key input parameter to be passed to it. It is recommended that the software unlock key be implemented with a 32-bit or larger variable for best protection. Example 2 below provides some code that implements this concept, in addition to an address and VDD check.

EXAMPLE 2: USING A SOFTWARE UNLOCK KEY

```
#define  SOFTWARE_UNLOCK_KEY_VALUE  (uint32_t)0x600D92FE
#define  NVM_MIN_WRITEABLE_ADDRESS  (uint32_t)0x4000    //Application specific value, change this in your code.
#define  NVM_MAX_WRITEABLE_ADDRESS  (uint32_t)0x4FFF    //Application specific value, change this in your code.
#define  NVM_MIN_VOLTAGE_MILLIVOLTS (uint16_t)3100      //Application specific value, change this in your code.

bool     NVM_OpStart(uint32_t address, uint32_t softwareUnlockKey, uint16_t key1, uint16_t key2)
{
    uint16_t VDDMilliVolts;

    //Check to make sure the softwareUnlockKey passed to this function is correct
    if(softwareUnlockKey != SOFTWARE_UNLOCK_KEY_VALUE)
    {
        //Error!  The caller didn't pass the correct parameter to this function.
        return false;
    }

    //Make sure the address pointed to is valid for erase/write operations.
    if((address < NVM_MIN_WRITEABLE_ADDRESS) || (address > NVM_MAX_WRITEABLE_ADDRESS))
    {
        //The address was out of bounds.
        return false;
    }

    //Make sure the VDD has enough margin to do safe erase/write
    VDDMilliVolts = ADC_MeasureVDD();
    if(VDDMilliVolts < NVM_MIN_VOLTAGE_MILLIVOLTS)
    {
        return false;
    }

    //All the checks passed, initiate the erase/write operation.
    __builtin_write_NVM_secure(key1, key2);

    return true;
}
```

Safe Handling of the Write Enable Bit

The microcontroller hardware typically implements some kind of write enable bit (i.e., WREN), which must be set prior to executing an unlock sequence to erase or write Flash memory. Common practice is to set WREN immediately before needing it and clear it as soon as the erase or programming operation is done. The rationale is that minimizing the duration that a hardware interlock is disengaged, the lower the probability that an accident can occur. This, however, has the unintended side effect of making the code that performs an erase or programming operation behave as an atomic unit, where any inadvertent entry into the earlier erase/programming preparation code chain can continue along and carry out a real NVM erase/program operation. This is undesirable as it makes all code in the entire RTSP reprogramming chain risky and unpredictable if the Program Counter ever jumps or falls into any link on the chain. The WREN interlock essentially provides no added protection when disengaged on immediate demand.

To make WREN provide actual protection, a strategy should be adopted where WREN is set exactly once and only from a code location outside of all iterative code loops that make up the RTSP reprogramming chain. For example, set WREN only when VDD is good and a valid, 32-bit or longer bootloader “Hello” is received from the communications partner or a user input is physically commanded from a menu selection. WREN should then be cleared from multiple places in code, wherever a non-recoverable error is detected or ordinary RTSP code termination occurs. For example, clear WREN if a communications time-out or corruption event occurs, clear WREN when the bootloader has finished programming a complete application image to Flash, clear WREN in all trap handlers and clear WREN anytime a boodloader hands off execution to an existing application. In all other cases, WREN’s state should be retained unchanged. This paradigm will suppress unintended entry into the RTSP’s algorithmic chain from having any permanent consequences, regardless of where in the chain unintended entry occurred.

Verify VDD is Sufficient Before Erase/Write Operations

The microcontroller should not be reset while an erase or write operation is already in progress. Most microcontrollers will complete an already started NVM operation if Reset does occur due to a $\overline{\text{MCLR}}$ event, clock monitor failure, etc., so individual Flash words rarely ever see an intermediate, not-fully-erased/not-fully-programmed state. The risk of a Reset due to a low-voltage condition, such as a BOR, however, will cause immediate erase/write termination and should be algorithmically avoided by measuring VDD just prior to an erase or write operation. This can be accomplished by implementing code just prior to the Flash erase/write unlock sequence, which measures the application VDD, as shown in Example 2. If the VDD is too low, the code should avoid starting an erase/write operation. Select dsPIC and PIC24 microcontrollers implement an ADC channel tied to a band gap reference. This can be used to back compute the present run-time VDD level. Some microcontrollers also implement a High/Low-Voltage Detect (HLVD or LVD) module which may also be useful for measuring the application VDD level at run time.

Validating Erase/Write Addresses

Prior to executing the unlock function to initiate an erase or write operation, it is recommended to always check the intended target erase/write address. This is to ensure it is pointing to a valid region reserved by the application for erase/write operations. Bootloaders, for example, should normally check to make sure that the bootloader is not attempting to erase part of itself (unless explicitly designed to do so) and EEPROM emulation code should verify that the erase/write region only includes the Flash memory assigned for the emulated EEPROM. The check should be structured such that only legal target addresses are allowed and not left open-ended. For example, rather than check that the target is “above the bootloader”, instead check that the requested erase/write target address is above the bootloader, below the first unimplemented program memory address and has Least Significant alignment bits zeroed to match the hardware erase/write addressing granularity appropriate to the requested operation. Any deviation should be treated as a sustained lockout condition that clears WREN and/or other enabling criteria.

Park NVM Erase/Write Address to Safe Location

Some level of decoupling will always exist between the code that sets up for an erase/write operation, and the code which does unlock keying and final triggering of the NVM operation. Part of the setup sequence will typically write to an NVMADR register to indicate the target of the upcoming erase/programming operation. Harm from unintended entry onto the execution path between these two code sequences can be prevented by simply parking NVMADR at an unimplemented target address. If an unintended erase/write occurs with an invalid address, hardware will ignore it and no changes to memory will result. This can be accomplished by setting NVMADR to any invalid address, such as 0xFFFFFE, after Reset and after each erase/write event.

SUMMARY

There are many ways in which a self-write application can protect itself from Flash memory corruption. When designing an application that will incorporate self-write capability, it is recommended to analyze the hardware design and the software implementation to ensure that they are following the best practices and recommendations outlined in this document.

ISBN: 978-1-5224-5843-2

Worldwide Sales and Service

AMERICAS

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

ASIA/PACIFIC

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

ASIA/PACIFIC

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

EUROPE

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: 49-7131-72400

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: 47-7288-4388

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