Freescale Semiconductor
Application Note

Document Number: AN3615
Rev. 0, 03/2008

Design Considerations for Implementing EEPROM using the MC9S08DZ60

by: Jesse Beeker
Lydia Ziegler
Field Applications Engineering
Detroit Automotive Technical Center

1 Background and Overview

This document is intended to serve as an additional reference for the EEPROM on the MC9S08DZ and MC9S08DN family of microcontrollers. Much of what is contained is applicable to most of Freescale's HCS08 microcontrollers that contain embedded EEPROM. Various applications and implementations of EEPROM, with regard to automotive applications, will be discussed. Application and usage of EEPROM as well as strategies specific to the MC9S08DZ microcontroller will be presented at a hardware and software level.

Contents

1 Background and Overview 1
1.1 New S08 Technology 2
1.2 EEPROM Uses and Applications 2
1.3 Types of EEPROM Devices 3
1.4 EEPROM Arrangement 5
2 Applications and Usage of EEPROM 8
2.1 Determining Application Requirements 8
2.2 Extending EEPROM Life 9
3 Simple Programming Through the Command Interface 13
3.1 State Machine Command Interface 13
3.2 State Machine Commands 14
4 Additional Strategies for Protecting EEPROM Data Integrity 17
4.1 Loss of Power 17
4.2 MCU Bus Clock 18
4.3 Software Runaway 19
5 Conclusion/Summary 19

freescale semiconductor

1.1 New S08 Technology

To address the competitive demands of the automotive industry, the MC9S08DZ60 and its derivatives offer flexibility through scalability. Migration among family members is easy with pin-compatible devices ranging from 16 K to 128 K of flash. Employing third generation 0.25 micron flash technology, Freescale is making flash devices more affordable, as the smaller geometry closes the price gap between flash and ROM technologies. In addition to the flash's smaller size and higher performance, this new flash memory technology allows read operations at device voltages as low as 1.8 volts and a typical data retention period of 100 years (15 years minimum over voltage and temperature). Additional features include a greatly simplified self-timed programming interface, flexible block protection, security, and on-chip EEPROM, making the MC9S08DZ family an excellent choice for mid- to high-end 8-bit applications.

1.2 EEPROM Uses and Applications

The requirements for nonvolatile data storage are very common in automotive applications. While program flash can be used to contain data that will not change during the life of a module, EEPROM has traditionally been used to contain data that may change over the life of a module or data that is specific to a particular module. This data might be as simple as an electronic serial number or as extensive as motor positioning data. Data that might potentially be stored in EEPROM would include:

- Odometer
- Serial number
- Test history and date of manufacture
- Calibration information
- Default application tables
- User configurable data
- Position data
- Encryption keys
- Dynamic network address
- Error code information
- Diagnostic test codes
- Black box recording
- Software feature activation

The inclusion of on-chip EEPROM in the MC9S08DZ family has a number of advantages related to both software and hardware design. Some of these advantages include:

- Fast access to data
- Reduction of printed circuit board components
- Lower microcontroller pin count requirement
- Simplified software handler
- Continued application execution during programming and erase procedures
- Fast programming time

- Small sector size
- Automated program and erase timing

1.3 Types of EEPROM Devices

During the conceptual phase of a design, several approaches can be considered when implementing EEPROM memory storage for data. The use of external EEPROM, emulation of EEPROM with program flash, and on-chip EEPROM are approaches that each have advantages and disadvantages in cost and reliability. In addition, each approach will have an impact on the software used to manage the data. The following sections explore the advantages and disadvantages of each approach in terms of cost, software, and hardware.

1.3.1 External EEPROM

External EEPROM devices are available from a number of manufacturers in a wide variety of sizes and configurations. While these devices can be very inexpensive, they add an additional level of complexity to a design in a number of ways. In addition to the actual cost of the component, an external EEPROM will require some minimum number of support components, such as bypass capacitors. Furthermore, an external EEPROM device is usually connected to the microcontroller via an SPI or IIC interface. This may result in the forced selection of a more costly microcontroller containing additional pins and one or both of the serial interfaces.

Using a serial interface to communicate with an external EEPROM device will also add considerable overhead to the firmware design. Having to communicate over a serial bus each time a byte is read or written can result in additional latencies in the system and degrade overall performance, as application tasks must wait for data to be available. Such driver firmware can further contribute to the total amount of flash memory required by the application.

One often-overlooked cost of an external EEPROM is the additional printed circuit board space required by the device. While the cost of additional board space may not be great, many automotive module designs are space-constrained and simply may not have room for an additional component on the board. In addition, the board layout can become more complex as device placement and routing of high-speed clock and data lines can lead to problems with radiated emissions or susceptibility issues. This in turn can lead to the addition of filter and/or termination components to assure proper operation of the external EEPROM.

1.3.2 EEPROM Emulation

Many low-cost microcontrollers containing flash program memory do not contain a separate array of EEPROM memory. Applications using these devices will often resort to using an external EEPROM. However, another strategy is to use a portion of the flash program memory to emulate EEPROM. Initially this approach seems attractive, because no additional components are required. However, any approach used to emulate EEPROM using program flash has drawbacks that can lead to a potential loss of data and to additional system costs.

In general, there are two common approaches when using program flash to emulate EEPROM. The first approach is to keep a copy of the EEPROM data in a RAM buffer and periodically write the entire contents

of the buffer to the program flash. This approach is relatively simple to implement, permits the data to be read from the RAM buffer at any time, and allows control of the number of program/erase cycles. The obvious drawback to this approach is the amount of RAM that must be dedicated to the emulated EEPROM buffer. In addition, there is a risk of losing data if a reset occurs after updating a RAM buffer, but before the data is programmed into flash.

A second approach uses multiple sectors of program flash to store nonvolatile data using a flash file system. Depending on the implementation, this method will usually require less RAM than the first approach. The major disadvantage to this approach is the size and complexity of the firmware required to implement a robust flash file system, such that it minimizes the risk of losing data when an unexpected reset or loss of power occurs.

Both of these approaches can require multiple program and erase operations when changing a single piece of data. This can lead to a significant increase of the amount of bulk capacitance required on the microcontroller's power supply lines to guarantee completion of all program and erase operations in the event of loss of power.

Another disadvantage of EEPROM emulation is the fact that, in general, code cannot be executed from a flash array while it is being programmed or erased. This restriction has two implications. First, this requires that the program and erase routines be located in RAM, decreasing the amount of on-chip RAM available for an application's variables. Second, because an application's interrupt vectors are located in flash, all interrupts must be masked during emulated EEPROM operations. Depending on an application's interrupt latency requirements, this may impose some severe limitations on the system.

1.3.3 On-Chip EEPROM

On-chip EEPROM, like the kind that is present on the MC9S08DZ microcontroller family, has significant system-level advantages over emulated or externally connected EEPROM. It can significantly reduce software complexity and has the least impact on the hardware design.

An often overlooked advantage that on-chip EEPROM offers is direct access to the memory itself in development and production environments. Accessing the EEPROM of the MC9S08DZ does not require a secondary tool or special test code, because the EEPROM is directly mapped into the memory map of the MCU. Development tools and production test equipment can quickly and easily program and read data through the background debug controller (BDC) interface.

In addition to programming convenience, on-chip EEPROM requires no hardware provisions, which gives it distinct advantages when it comes to hardware design. This saves on system cost and increases reliability through reduced component count and PCB board space. Having the EEPROM structure completely internal also improves performance. Communication to the memory is not regulated by external bus delays associated with common interfaces, such as IIC or SPI. Furthermore, placing an application's nonvolatile data in a separate yet on-chip array has the advantage of allowing independent write protection schemes that are best suited for the type of information stored in program flash and EEPROM.

1.4 EEPROM Arrangement

The size of the EEPROM on each MC9S08DZ device varies among family members. On all devices, the EEPROM array is divided into 8-byte sectors. Each byte within a sector may be individually programmed — however, all eight bytes within the sector must be erased at the same time.

As shown in Figure 1, the EEPROM in the MC9S08DZ family is accessed through a page window within the 64 K address space. The size of the page window is equal to one half of the total amount of EEPROM contained on a device.

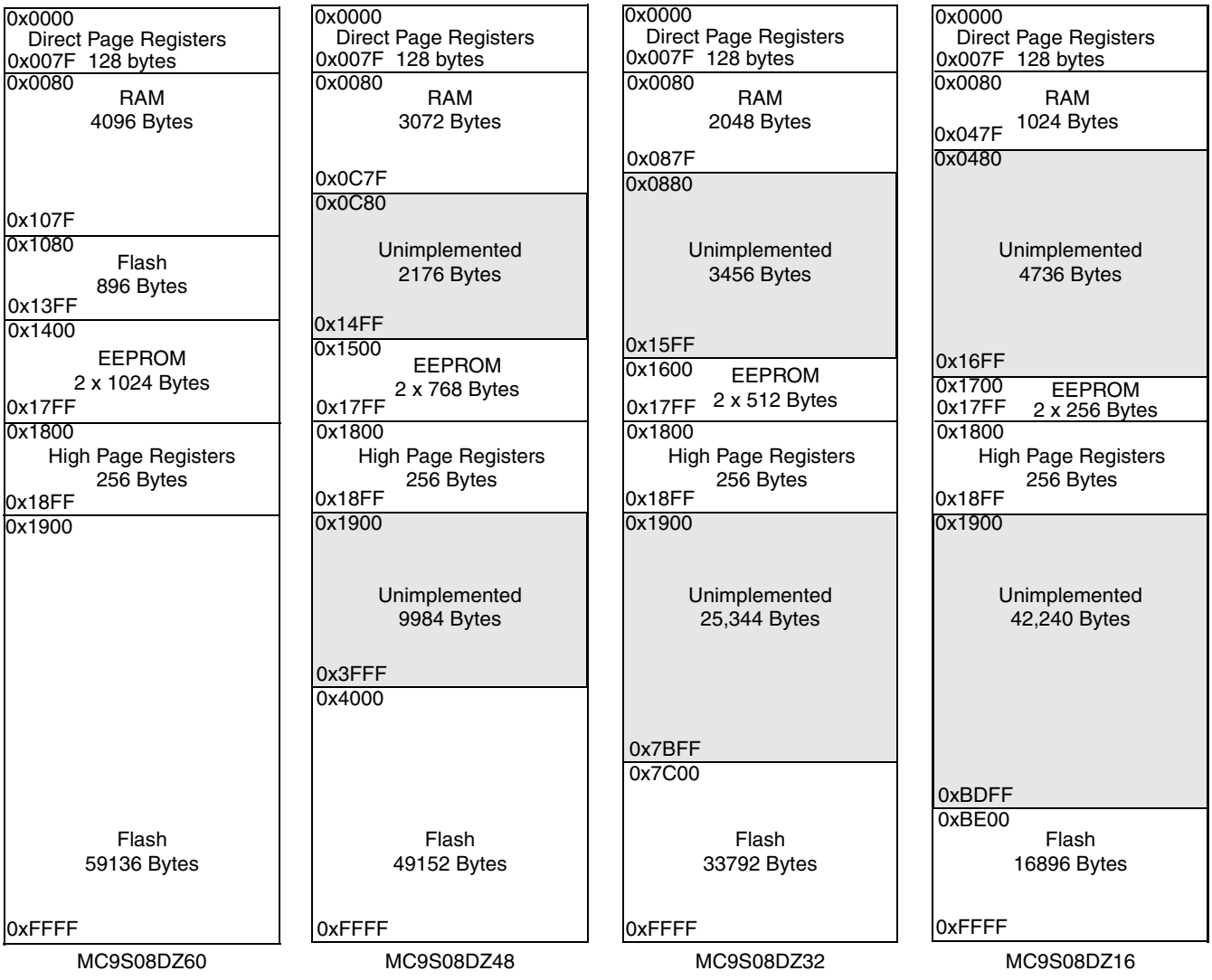

Figure 1. MC9S08DZ Family Memory Map

The paging mechanism allowing access to both halves of the EEPROM array is controlled by the EPGSEL bit in the FCNFG register. At reset, the value of EPGSEL is zero, placing EEPROM page zero in the foreground and page one in the background. This bit may be read or written at any time, thus providing application software the ability to switch pages or determine which page is currently in the foreground.

An additional level of flexibility is provided for applications by allowing a choice of how many bytes of each sector will appear at consecutive addresses within a page. In the so-called 4-byte sector mode, access to each 8-byte sector is split, placing four consecutive bytes at the same address on each page, as shown in Figure 2. As the figure shows, access to the four bytes of EEPROM on each page begins at an address that is an even multiple of four. Note that when performing a sector erase operation, four bytes beginning at the same address in *each* page will be erased.

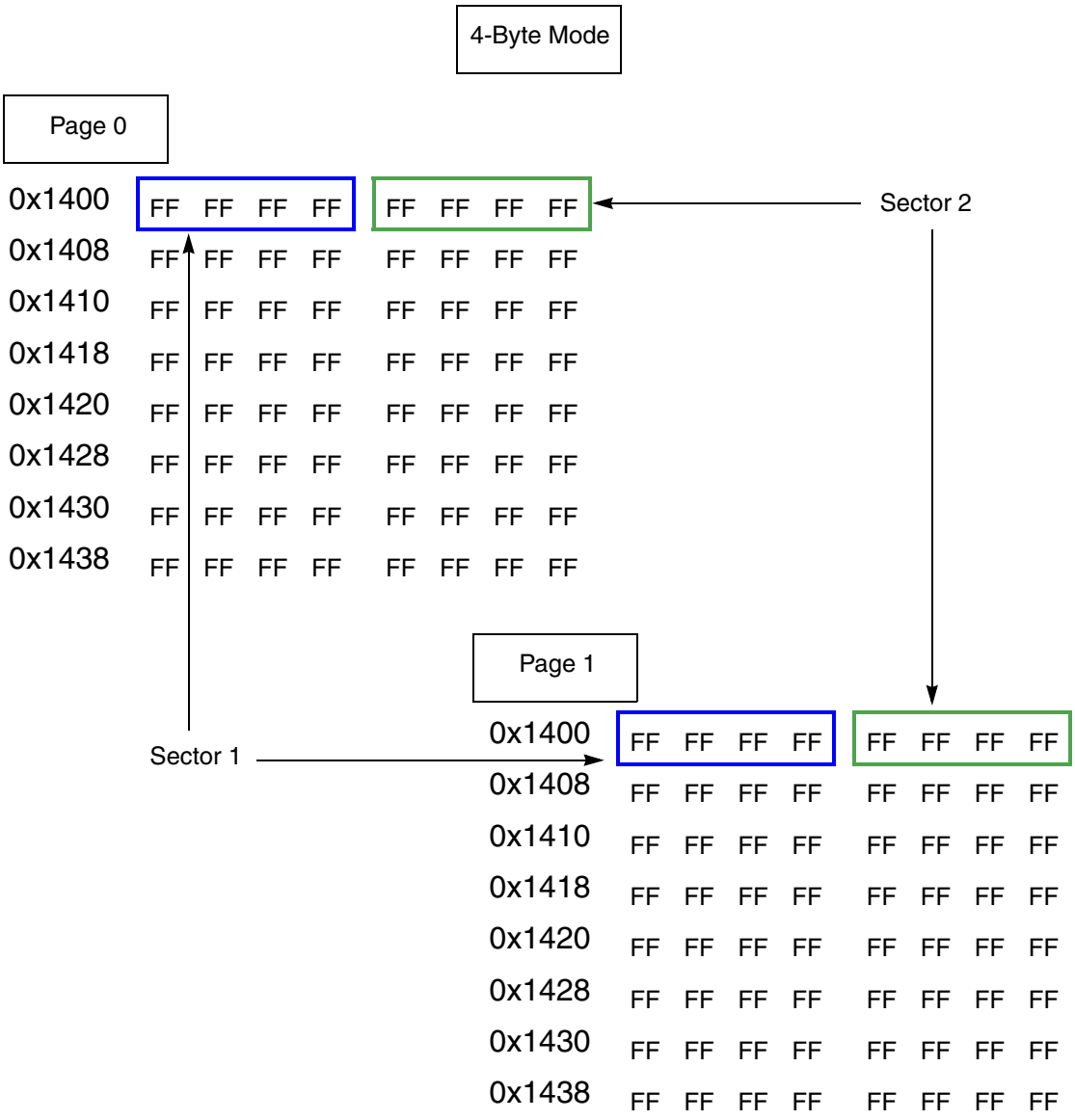

Figure 2. EEPROM Configured for 4-Byte Mode

The so-called 8-byte sector mode places each 8-byte sector at eight consecutive addresses in the same page, as shown in Figure 3. This arrangement allows all eight bytes of the sector to be accessed when performing program or erase operations without performing a page switch.

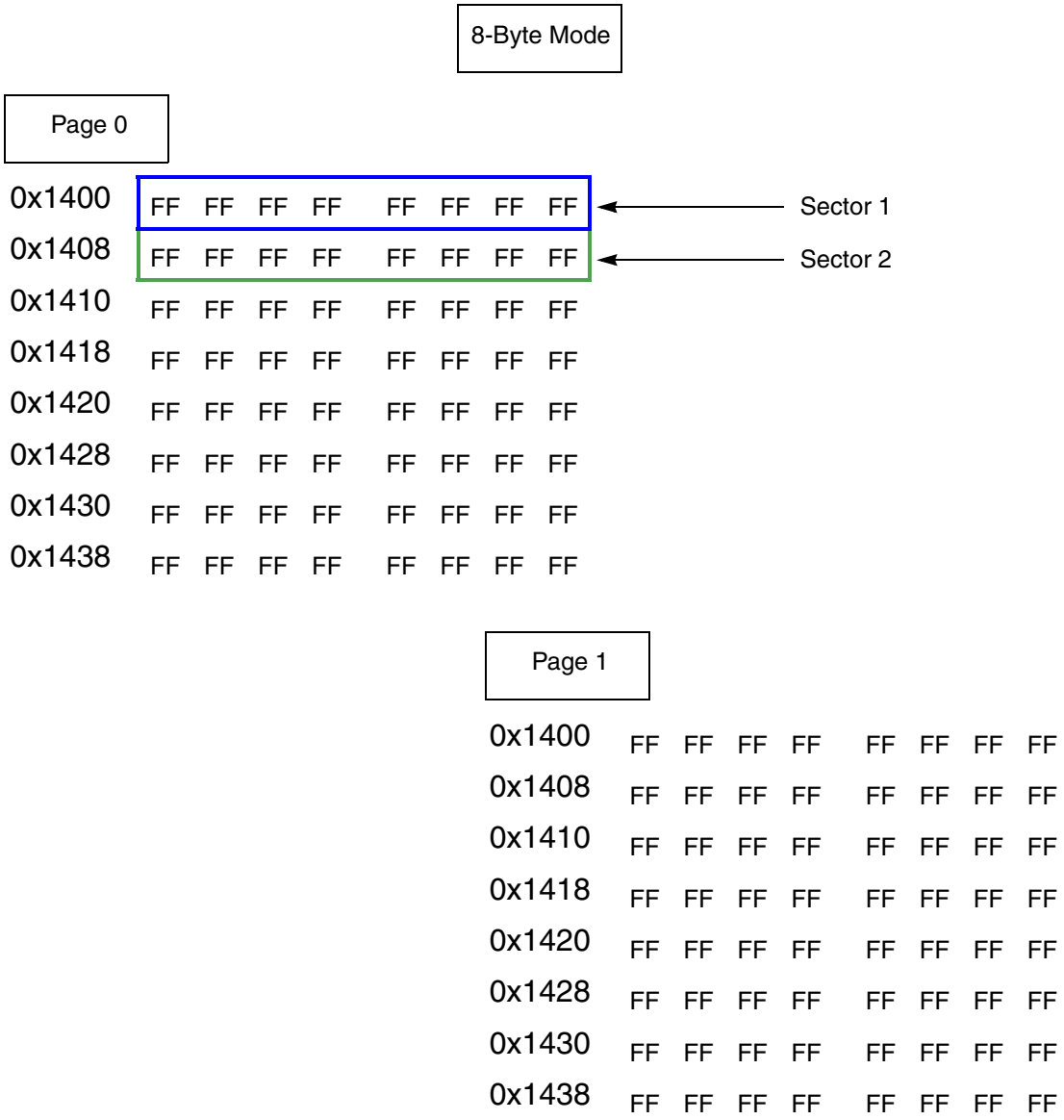

Figure 3. EEPROM Configured for 8-Byte Mode

Setting of the 4-byte or 8-byte sector mode is controlled by the EPGMOD bit in the FOPT register. Because the value of the EPGMOD bit is loaded from program flash location NVOPT (0xFFBF) during reset, the sector operating mode cannot be changed without erasing and reprogramming location NVOPT and then forcing an MCU reset. For the vast majority of applications, one of the two modes would be chosen during the firmware design, based on application need. This is important to consider during the generation of the S-record as well as when configuring the programming tool.

The decision of whether to use so-called 4-byte or 8-byte sector mode will be based on the size characteristics of the data being stored in EEPROM. If small individual pieces of data need to be stored, the 4-byte sector mode would likely be the best choice, because it would result in the most efficient use of the EEPROM by placing a larger number of sectors within the EEPROM page window. Remember, however, that erasing four bytes on one page also erases the four bytes of EEPROM at the corresponding

address on the other page. Therefore, the 4-byte sector mode may not work well if the number of 4-byte sectors required by the application exceeds the number visible in a single page window. Instead, the 8-byte sector mode would have to be used, packing multiple pieces of data into a single 8-byte sector. On the other hand, if most of the data elements stored in EEPROM are greater than four bytes, the 8-byte sector mode would likely be more efficient, because it would allow the data to be contained in fewer sectors with contiguous addresses.

2 Applications and Usage of EEPROM

The presence of dedicated on-chip EEPROM on the MC9S12DZ family of microcontrollers provides a valuable addition for embedded applications that require the storage of nonvolatile data that is updated frequently. Like flash program memory, EEPROM has a limited number of erase/write cycles. Therefore, careful consideration must be given to an application's nonvolatile data storage requirements. The remainder of this section will discuss application EEPROM requirements and methods that can be used to effectively extend EEPROM endurance for nonvolatile data whose contents will change more often over the life of an application than the guaranteed minimum number of erase/write cycles.

2.1 Determining Application Requirements

Determining the actual number of update cycles required for each piece of data stored in EEPROM involves the consideration of a significant number of factors. Each piece of data and its relationship to the application must be carefully examined to establish how often new values must be written to EEPROM. Writing values more often than necessary could result in EEPROM locations exceeding the guaranteed minimum specification before the end of a product's life. Writing values too infrequently could result in less than optimum application performance, if the most recent data is lost due to a system failure. In addition, factors such as product life cycle and warranty length must be considered. As an application's requirements for nonvolatile data storage increases, determining the requirements becomes increasingly important because each EEPROM location has a guaranteed minimum number of erase/write cycles in which data retention is guaranteed for a period of time.

For applications having relatively simple nonvolatile data storage requirements, determining the total number of update cycles is relatively easy. Such applications might include the storage of calibration data, serial numbers, test history, date of manufacturer, encryption keys, or software feature activation codes in EEPROM. Such data, likely to be written at end-of-line testing, may be written to the EEPROM only once in the product's life, or perhaps rewritten if the product's firmware is updated at the end customer. Any usage of the on-chip EEPROM is comparatively simple if the number of nonvolatile data updates does not exceed the guaranteed minimum number of erase/write cycles over a product's expected life cycle.

There are application requirements, however, such as the storing of odometer data, user configuration data, error code information, or diagnostic test codes, where the number of nonvolatile data updates will easily exceed the guaranteed minimum 10,000 erase/write cycle specification of the MC9S08DZ family. One example of such an application would be an odometer application where mileage must be recorded to 1,000,000 miles with a one-mile accuracy. Clearly, such an application will require a nonvolatile data storage strategy to address such a high endurance application requirement.

2.2 Extending EEPROM Life

As mentioned in the previous section, some applications can require the updating of nonvolatile data that exceeds the guaranteed minimum number of erase/write cycles of the MC9S08DZ family. Fortunately, several simple strategies can be used to extend the effective number of erase/write cycles for such data. Providing a robust solution requires careful examination of the nonvolatile data storage requirements of an application.

As previously discussed, the EEPROM contained on the MC9S08DZ family is arranged in 8-byte sectors. While each byte within a sector can be individually programmed, all eight bytes of the sector must be erased at the same time. Therefore, in terms of guaranteed minimum erase/write cycles, even if only one byte within a sector is programmed before the sector is erased, the erase operation counts as an erase/write cycle for all eight bytes within the sector. This characteristic of the EEPROM must be carefully considered when developing a strategy to effectively extend the number of erase/write cycles.

One of the least complicated strategies to use for nonvolatile data exceeding the guaranteed minimum erase/write cycles is to simply use multiple sectors to contain the data over a product's life cycle. For example, a product that may need to write a single byte of nonvolatile data 30,000 times over a product's life could simply use three sectors to contain the data. This strategy essentially involves moving the data to a new, unused sector each time the guaranteed minimum erase/write cycles of a sector is reached. In addition, a method must be devised to determine which sector contains the most current data. Also, when moving data from one sector to another, careful consideration must be given to protecting the integrity of the data as it is moved.

2.2.1 Organization of Data

To provide the most efficient use of the on-chip EEPROM when moving data among sectors, you should group together data that is updated at similar rates, under the same conditions, and requiring the same number of cycles. Grouping data in this manner can allow the use of a single cycle count variable (in addition to any other required data management information) that will be shared by the entire group. This helps to reduce the overhead associated with the use of such a strategy.

Because each EEPROM sector consists of eight bytes and resides on an 8-byte boundary, it is unlikely that an application's nonvolatile data structures will conform to an exact multiple of the sector size. Therefore, even when you are grouping data that has similar characteristics, it is unlikely that 100 percent of a group of allocated sectors can be used. Locations that are not used for data storage can be allocated to cycle count or other data management variables that are required by a specific nonvolatile data storage strategy. Alternately, any unused locations within a group could be used for future expansion.

When defining groups of data, care should be taken to ensure that the data structure containing the variables begins at the start of a sector and contains a number of bytes that is an even multiple of the sector size. Allowing data structures to cross sector boundaries can greatly increase the likelihood of data corruption, and significantly complicate the software required to manage nonvolatile data. When programming in C, this may require the addition of some number of padding bytes to a data structure. Figure 4 provides a simple example of how data can be grouped together based on usage.

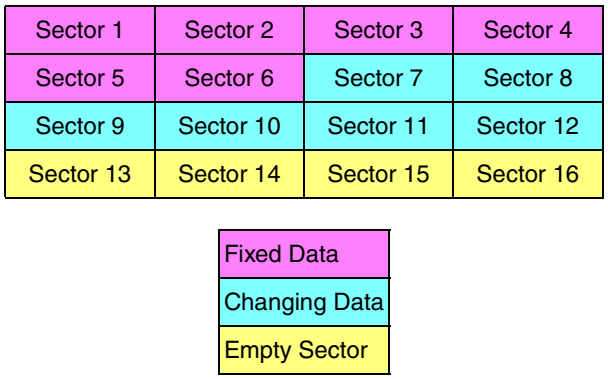

Figure 4. Grouping Similar Data

2.2.2 Data Integrity

When writing data to or erasing a sector of the EEPROM on the MC9S08DZ family, there is always the possibility of an unintended reset interrupting the operation. When this occurs the integrity of the stored data can be compromised, unless a strategy is developed to detect and recover from such possibilities. Detecting corrupt data in an EEPROM block or record is usually as simple as adding a checksum or CRC value to each record. Each time a record is updated, a checksum or CRC is computed and stored in the EEPROM as part of the record. At each MCU reset, a check of each data record can be made to ensure that a computed checksum or CRC matches the one stored as part of each record. If the data integrity check fails, the application or nonvolatile memory software will need to take the proper steps to correct the invalid record. This may involve setting the elements of the record to default values from flash program memory, or using values from the last known valid record.

The use of a checksum or CRC to ensure data integrity generally works best for data that remains constant, or data that seldom changes over the life of a product. For data that constantly changes or data that is moved to effectively extend the EEPROM's erase/write cycle endurance, other methods must be used. When saving multiple pieces of data within a sector or even data spanning multiple sectors, there is always a chance that the erase or program operations can be interrupted at any point in the process.

For example, if a single 32-bit value were stored in an 8-byte sector, updating the value would require a sector erase operation, followed by four byte-programming operations. If this process were interrupted by a system reset just after the erase operation, all record of the value would be lost. To prevent the loss of data in a case such as this, it would be prudent to keep a minimum of two copies of the data in two separate sectors. If a system reset occurred just after the erase of one of the sectors, a valid copy of the previous value would always exist in the second sector.

Keeping multiple copies of data in separate sectors solves the problem of an unintended reset after a sector erase operation. However, it does not solve the problem of a reset occurring before all four bytes of the 32-bit data are programmed into the sector. In this case, depending on the number of bytes programmed before a reset occurred, an application may not be able to distinguish between valid and invalid data. To solve this issue, an additional validation flag can be added to each record, allowing software to confirm that all writes of the data record were properly completed. This validation flag would be the last byte written to a record and generally consist of a pattern of alternating ones and zeros, i.e. 0x55 or 0xAA.

Combining these two techniques with additional software can allow an application to not only detect, but also recover from, an untended interruption of updating a block of data in EEPROM.

2.2.2.1 EEPROM Protection

In addition to software strategies that can be used to protect the integrity of on-chip EEPROM data, the MC9S08DZ family includes a hardware protection mechanism that can be used to prevent the accidental erasure or programming of a portion of the EEPROM. This mechanism is controlled by the two most significant bits of the FPROT register, EPS1:0.

Table 1. EEPROM Protection Control Bits

EPS1:0	Protected Address Range	Protected Memory Size (Bytes)	Number of Protected Sectors
11	N/A	0	0
10	0x17F0 – 0x17FF	32	4
01	0x17E0 – 0x17FF	64	8
00	0x17C0 – 0x17FF	128	16

At reset, the contents of the FPROT register are loaded from program flash location NVPROT (0xFFBD). As shown in Table 1, the erased state of the EPS bits allows the entire EEPROM array to be erased or programmed. The remaining combinations of the two EPS bits allow a maximum of 128 bytes to be protected. Note that for each combination, half of the protected bytes are in EEPROM page 0 and half in page 1. The EPS bits may be read at any time; however, these bits can be written only to a value that increases the number of sectors protected. This feature allows all of the EEPROM to remain in an unprotected state immediately out of reset and in a protected state at a later point. While this feature might seem to negate the benefit of automatic protection of EEPROM data immediately out of reset, it can be useful if a bootloader requires the ability to reprogram the data in the protected memory range.

In general, the hardware EEPROM protection mechanism is used to prevent the accidental erasure or programming of data that is considered critical to the operation of a product. This might include data such as calibration constants or serial numbers.

2.2.3 A Simple Example

Consider an application with a requirement to record a 32-bit value one million times over the life of the product. Because a 32-bit value will easily fit within a single sector and each sector can be erased and written a minimum of 10,000 times, one hundred sectors are required to record the data without exceeding the MC9S08DZ EEPROM specifications for any single sector. As shown in Figure 5, each of the 8-byte sectors reserves room not only for the 32-bit data, but also contains a data validation flag. As discussed in the previous section, this data byte is written with a value other than the erased state only after all four bytes of the data are successfully programmed into the sector. After the data validation flag for a new record has been programmed and verified, the software could erase the sector containing the previous data. This is analogous to a make-before-break mechanical switch.

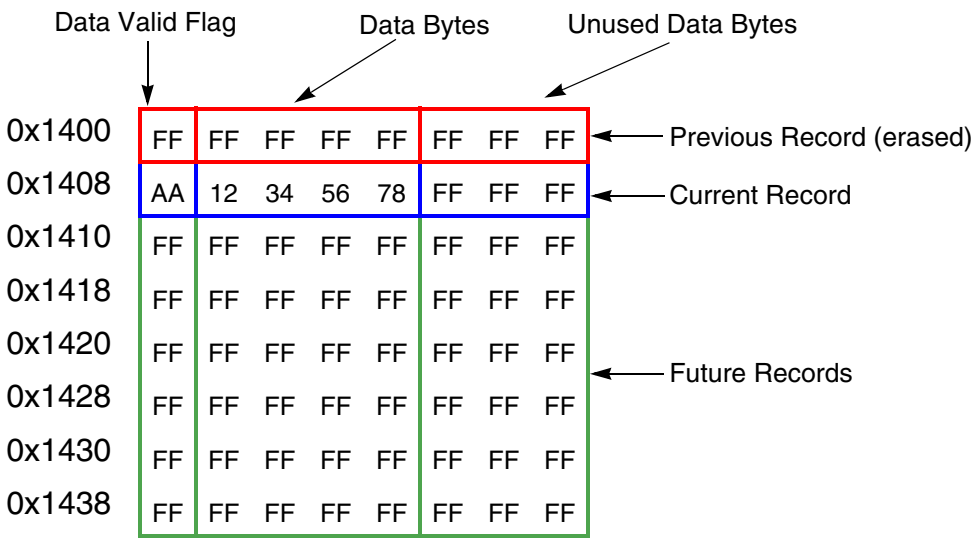

Figure 5. Moving Data Example

As discussed in Section 2.2.2, "Data Integrity," the possibility of an unintended system reset always exists. This may lead to an interruption of the data storage process. In the context of the current example, this would allow for two distinct scenarios — the first would be where two full data records exist in an array, and the second would be where a full record and a partial record exist. To account for this possibility, the application software would have to scan the EEPROM data array after each reset to determine if any two adjacent sectors are in a non-erased state.

If the sector at the logically higher address does not have the data validation byte programmed, it indicates that the writing of the new record was interrupted, as shown in Figure 6. This record should be erased and the data in the previous sector used as the most current valid data.

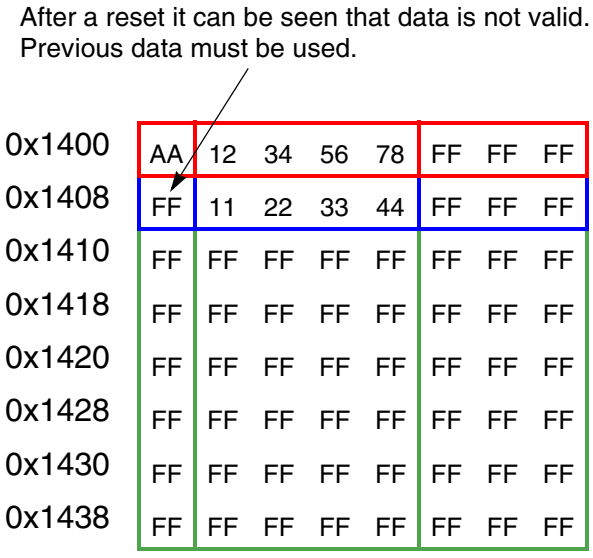

Figure 6. Interruption of Programming — Scenario 1

When scanning the array, if two adjacent sectors are in a non-erased state and both contain programmed data validation flags, then the update process was interrupted before the old record could be erased. In this case, the sector at the lower logical address should be erased. This is illustrated in Figure 7.

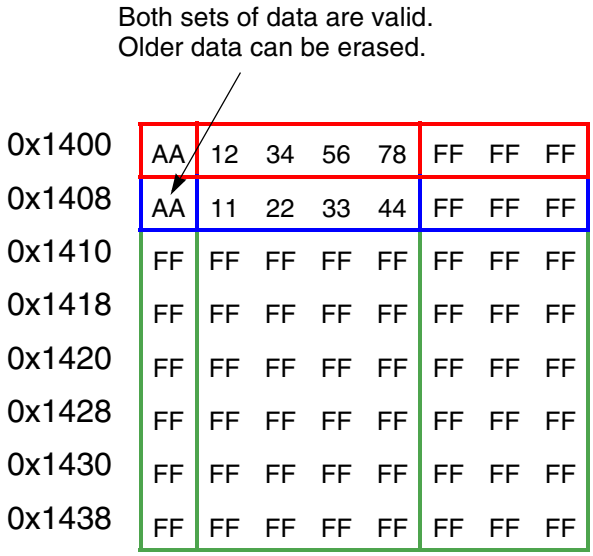

Figure 7. Interruption of Programming — Scenario 2

In general, where two full records or a full and a partial record exist in the data array, the older record would reside at the lower address. The exception to this rule is if the newest record resides in the first sector of the array and the older record resides in the last sector of the array. When software needs to retrieve the current value, it can search the array until the single valid record is found.

3 Simple Programming Through the Command Interface

The erasure and programming of flash and EEPROM requires the application of precisely timed and sequenced high voltage pulses to each bit cell, to obtain the maximum number of erase/write cycles and long-term reliability. Therefore the MC9S08DZ family contains dedicated state machine logic for the programming and erasure of the on-chip flash and EEPROM. The state machine logic not only provides a layer of abstraction for the application software, it allows the software to perform other tasks during the EEPROM program and erase operations.

3.1 State Machine Command Interface

The interface to the flash and EEPROM state machine consists of six 8-bit registers, as shown in Figure 8.

0x1820	FCDIV	DIVLD	PRDIV8	DIV					
0x1821	FOPT	KEYEN	FNORED	EPGMOD	0	0	0	SEC	
0x1822	Reserved	—	—	—	—	—	—	—	—
0x1823	FCNFG	0	EPGSEL	KEYACC	Reserved	0	0	0	1
0x1824	FPROT	EPS		FPS					
0x1825	FSTAT	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
0x1826	FCMD	FCMD							

Figure 8. Flash and EEPROM Control and Status Registers

Before performing any EEPROM erase or program operations, the FCDIV register must be initialized to provide a timing clock to the flash state machine. Because the input clock to the flash and EEPROM module is the MCU bus clock, a divider chain controlled by the contents of the FCDIV register must be configured to provide the required 150 kHz – 200 kHz clock. The lower six bits of the FCDIV register directly control a modulus down-counter allowing the bus clock to be divided by a maximum of 64 (DIV[5:0] + 1). Using only these six bits would limit the bus clock to a maximum of 12.8 MHz (200 kHz * 64). However, writing a one to the PRDIV8 control bit inserts an additional three bit binary counter (÷8) into the divider chain, allowing the use of higher bus frequencies. The formulas for determining the correct value of the lower seven bits of the FCDIV register are shown in Figure 9.

```
if (BusClock <= 12800000)
 {
        FCDIV = BusClock / FCLK;
        if ((BusClock % FCLK) == 0)
                FCDIV -= 1;
 }
else
 {
        FCDIV + ((BusClock / FCLK) / 8) + 0x40;
        if ((BusClock % (FCLK * 8)) == 0)
                FCDIV -= 1;
 }
```

Figure 9. Formulas for Determining the Correct FCDIV Value

The most significant bit of the FCDIV register, DIVLD, is a read-only status bit indicating that the FCDIV register has been written since the last time the device was reset. This bit can be used by application software, before performing any flash or EEPROM operations, to validate that the FCDIV register has been written. Note that even if the DIVLD bit is set, it does not indicate that a valid divider value has been written to the lower seven bits of the FCDIV register. Finally, the FCDIV register may only be written once after reset. Any additional writes are ignored.

3.2 State Machine Commands

The MC9S08DZ nonvolatile memory state machine supports six unique commands as shown in Table 2.

Table 2. MC9S08DZ Nonvolatile Memory State Machine Commands

Command	FCMD	Equate File Label
Blank check	0x05	mBlank
Byte program	0x20	mByteProg
Burst program	0x25	mBurstProg
Sector erase	0x40	mSectorErase
Mass erase	0x41	mMassErase
Sector erase abort	0x47	mEraseAbort

Each of the six commands initiates an operation performed independently by the hardware state machine. This allows the application software to perform other operations, while the state machine performs the critically timed operations required to program or erase the EEPROM. After a command is initiated, it will run to completion unless an error occurs. Note that for programming operations, the location(s) being programmed must be in the erased state (0xFF), prior to launching a program command. Failure to observe this restriction can result in values at other locations of the EEPROM array being changed. This phenomenon is known as program disturb.

Executing a command requires the six steps shown in Figure 10. Program code implementing these steps must be followed rigorously. Reads or writes of any other memory locations after the write to the flash or EEPROM (step 2) and before launching the command (step 4) will cause the command to be aborted and the FACCERR bit to be set. Because of this restriction, the code implementing steps two through four should have all interrupts disabled, to prevent the unintended termination of a program or erase command. This constraint during a program or erase operation is intended to prevent accidental erasure or programming of the flash or EEPROM due to runaway code.

1. Write 0x30 to the FSTAT to clear possible error conditions from any previous commands.
2. Write a data value to a valid address in the EEPROM array. (For a sector or mass erase command, the data written is unimportant.)
3. Write the command code for the desired command to the FCMD register.
4. Write a 1 to the FCBEF bit in FSTAT to clear FCBEF and launch the command.
5. After launching the command, check the state of the FPIVOL and FACCERR for errors.
6. Wait until the FCCF bit in FSTAT is set. As soon as FCCF = 1, the operation has completed successfully. (When executing the Burst Program Command, wait until FCBEF bit is set and continue at step two.)

Figure 10. Flash/EEPROM Command Execution

Note that because a single hardware state machine and interface registers are used to perform operations on both the on-chip flash and EEPROM, the state machine distinguishes between flash and EEPROM operations solely by the address latched in the command buffer in step two.

The following sections contain a brief description of each command.

3.2.1 Blank Check Command

The blank check command can be used to check the entire flash or EEPROM for the erased state. If the command completes successfully and the entire array is blank, the FBLANK bit in the FSTAT register will be set. Note that for the EEPROM, both pages are checked and must be in the erased state for the FBLANK bit to be set.

3.2.2 Byte Program Command

The byte program command can be used to program a single byte of the flash or EEPROM. As with all erase and program commands, the byte program command will provide the correct timing sequence and application of the internal programming voltages to the EEPROM to program a single byte. The command requires nine state machine clock cycles (FCLK) to complete. Using a 200 kHz FCLK frequency, the byte program command executes in 45 µs.

3.2.3 Burst Program Command

The burst program command is similar to the byte program command, but with an important difference.

When a byte program command is executed, an internal high voltage charge pump associated with the flash memory must be enabled to supply the programming voltage to the EEPROM array. Upon completion of the command, the charge pump is turned off. These operations account for much of the time required for the execution of the byte program command.

The burst program command operates in much the same manner as the byte program command, except that at the completion of the command the high voltage charge pump remains enabled if two conditions are met. First, the next burst program command sequence begins before the FCCF bit is set in the FSTAT register. Second, the address of the next burst program command must reside within the same burst block. A burst block in this flash memory consists of thirty-two bytes, with a new burst block beginning at each 32-byte address boundary.

Note that the first byte of a burst program command will take the same amount of time as a byte program command. However, subsequent bytes will program at the burst program rate of four state machine clock cycles (FCLK) or 20 µs at a 200 kHz FCLK frequency.

3.2.4 Sector Erase Command

The smallest amount of EEPROM that can be erased is a single sector, which is eight bytes. This is true whether the so-called 4-byte or 8-byte sector mode is selected. Executing the sector erase command when the 4-byte sector mode is selected will erase four sequential bytes beginning at the same address on each of the two EEPROM pages. If 8-byte sector mode is selected, eight sequential bytes on the selected page will be erased. The sector erase command requires 4000 state machine clock cycles (FCLK) or 20 ms at a 200 kHz FCLK frequency.

3.2.5 Mass Erase Command

The mass erase command can be used to initialize all locations in the EEPROM array to the erased state. Note that it will erase both pages of the EEPROM array, regardless of the state of the EPGSEL bit in the FCNFG register. After the mass erase command has been launched, it cannot be terminated. If a range of EEPROM has been write protected, execution of the mass erase command will be immediately terminated, and the FPVIOL bit in the FSTAT register will be set. Execution of the command requires 20,000 state machine clock cycles (FCLK) or 100 ms at a 200 kHz FCLK frequency.

3.2.6 Sector Erase Abort Command

In the event a sector erase command is executing and an unexpected event occurs requiring the application to have immediate access to data in the EEPROM array, the sector erase abort command can be used to prematurely terminate a sector erase command. If the sector erase abort command successfully completes and the sector is not completely erased, the FACCERR bit in the FSTAT register will be set. However, if the sector erase abort command is launched near the end of a sector erase command, the sector erase command may complete normally. In this case, the FACCERR bit will be cleared.

To ensure EEPROM data integrity, if a sector erase is terminated prematurely, the affected sector must be erased again before any byte within the sector can be programmed. Note that the sector erase abort command should be used sparingly, because a sector erase operation that is aborted counts as a complete program/erase cycle.

4 Additional Strategies for Protecting EEPROM Data Integrity

As discussed in Section 2.2.2, “Data Integrity,” when writing data to or erasing a sector of EEPROM on the MC9S08DZ family, there is always the possibility of an unintended reset interrupting the operation. When this occurs, the integrity of the stored data can be compromised, unless a strategy is developed to detect and recover from such possibilities. In addition to unintended resets, several other system failure modes can lead to potential EEPROM corruption unless strategies are developed to protect the data. The remainder of this section will discuss three areas in automotive systems that require specific attention to prevent EEPROM data corruption.

4.1 Loss of Power

Analysis of a system’s power source and power supply design is critical to many aspects of a design. Factors such as power source stability, power up/power down characteristics, active device decoupling, and bulk charge storage are key areas of consideration in a robust design. Because of hardware protection mechanisms designed into the MC9S08DZ family, such as the power-on reset (POR) and low voltage reset (LVR) circuitry, there is no potential for EEPROM data corruption if there are no program or erase operations in process if power is unexpectedly lost. However, if program or erase operations are in process when system power is unexpectedly removed, EEPROM data corruption can occur unless proper precautions are taken.

The unexpected removal of power from a module can be the result of manual battery disconnect or power connection problems due to an intermittent mechanical connector. In either case precautions must be taken, using a combination of hardware and software to allow EEPROM operations to complete before power to the MCU drops below its minimum operating levels. From a hardware design standpoint, the main considerations are: providing an early indication to the software that power has been lost, and providing enough reserve power to complete any pending EEPROM operations.

Providing an early indication of power loss can be achieved by monitoring the battery input voltage to the module and notifying the software via an interrupt when the battery voltage drops below a predetermined level. This can be achieved using the MC9S08DZ's internal analog comparator, or by using the analog-to-digital converter to perform periodic measurements. Choosing the battery voltage level at which to notify the software of a loss of power will involve careful design and characterization of the power supply, analysis of maximum loads, and determination of the time required to perform any EEPROM operations.

Providing a power reserve for a module after a loss of power is typically supplied by bulk capacitance on both the input and output of the module's voltage regulator. While adding additional bulk capacitance to extend the amount of time that software has to complete EEPROM writes is relatively simple, it will add additional hardware costs to a design. In addition, space constraints and in-rush current requirements may not allow the addition of extra bulk capacitance. To keep the amount of required bulk capacitance to a minimum, additional strategies may have to be employed that will reduce module current consumption after a loss of power is detected. One such strategy might include selectively removing power from any circuitry within the module that is unnecessary for MCU operation.

From a software perspective, the main determinant in the time required to complete EEPROM operations after a loss of power will be the number of byte write and sector erase operations that must be performed. With the sector erase command requiring a minimum of 20 ms and the burst program command requiring a minimum of 20 µs, it is clear that the number of sector erase operations required at loss of power will dominate the time requirements.

4.2 MCU Bus Clock

As discussed in Section 3.1, "State Machine Command Interface," the EEPROM hardware state machine requires an input clock between 150 kHz and 200 kHz. Because the state machine clock is derived from the MCU's bus clock, a basic understanding of the clock sources within MC9S08DZ's multi-purpose clock generator (MCG) module is essential.

The MCG module provides several clock source choices for the MC9S08DZ devices. The module contains a frequency-locked loop (FLL) and a phase-locked loop (PLL) that are controlled by either an internal or an external reference clock. The output of the FLL, PLL, and internal or external reference clocks can be selected as a source for the MCU bus clock. Any of the four clock sources may optionally be divided by two, four, or eight, to reduce the bus frequency and consequently the MCU's power consumption.

As described in Section 3.1, "State Machine Command Interface," the value written to the FCDIV register controls a modulus down-counter, allowing the bus clock to be divided until it is within the 150 kHz to 200 kHz range required by the EEPROM hardware state machine. Because the FCDIV register can only be written once after reset, the bus frequency must remain within a range that will provide the proper clock frequency to the hardware state machine during EEPROM program and erase operations. If an application

needs to reduce the MCU's bus frequency to reduce current consumption, proper logic must be added to the EEPROM software to prevent program or erase operations when the bus frequency would produce a state machine clock outside the part's specified range.

When using either the FLL or PLL in an application, noise conditions in the system can cause these clock sources to lose their lock condition. This can cause the FLL or PLL, and hence the bus frequency, to temporarily operate outside the nominal frequency setting. Because the FLL and PLL can operate up to nearly ±6% of the nominal frequency without indicating an unlock condition, it would be good design practice to set the EEPROM hardware state machine frequency 6% below the specified maximum of 200 kHz.

In addition to the LOCK status bit in the MCGSC register, the loss of lock status (LOLS) bit is an additional bit used to indicate that the FLL or PLL output frequency has fallen outside the lock exit frequency tolerance. When this sticky bit (in other words, when set, this bit must be written with a value of 1 to clear the bit) is set, it can optionally generate an interrupt, if the loss of lock interrupt enable (LOLIE) bit in the MCGC3 register is set. This can be used to indicate to application software that the FLL or PLL is operating at more than ±6% of the nominal frequency, and take appropriate action with respect to EEPROM operations.

4.3 Software Runaway

Software runaway or errant code execution can occur for a variety of reasons, but the main cause is usually faulty software. For applications that perform EEPROM erase and program operations, the software that executes these functions must necessarily be a part of the application. Therefore, the possibility exists that those portions of code that erase and program EEPROM could be unintentionally executed in a software runaway situation, resulting in EEPROM data corruption. Care should be exercised when designing the EEPROM management routines so that EEPROM contents cannot be corrupted if the code is executed because of software runaway. Information on how to protect the software from runaway conditions can be found in EB398, *Techniques to Protect MCU Applications Against Malfunction Due to Code Run-Away.*

5 Conclusion/Summary

While EEPROM technology is often overshadowed by flash memory technology, it still provides essential functionality to automotive applications. Using EEPROM in a design requires the consideration of numerous aspects, involving both hardware and software. The architecture of the MC9S08DZ family minimizes these complications by incorporating the EEPROM on-chip. Applications requiring EEPROM can further benefit from the MC9S08DZ family's internal NVM hardware state machine for program and erase operations, flexible sector sizes, and optional hardware protection settings. Through proper organization and data management, the toughest automotive demands for EEPROM usage can be met by the MC9S08DZ family.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3615
Rev. 0
03/2008

freescale™
semiconductor