Freescale Semiconductor
Application Note

Document Number: AN4483
Rev. 1, 03/2012

Emulated EEPROM Routines for the S12P Family

by: Gordon Doughman
Field Applications Engineer

Contents

1 Introduction

A number of the devices in the 0.18µ S12 and S12X family contain Data Flash (DFlash) instead of EEPROM. Like the EEPROM contained in the 0.25µ S12 family, the DFlash resides in an array that is physically separate from the device's Program Flash (PFlash). Having a separate array of small sector size Flash for storing nonvolatile data simplifies the software required to effectively manage nonvolatile data. The separate array allows the nonvolatile data management software to execute out of the PFlash while erasing and programming the DFlash.

The S12P family contains 4096 bytes of DFlash. The 4KB of DFlash consists of 16 sectors of 256 bytes each. The DFlash can be programmed one to four words at a time and can be erased one sector at a time. In addition, each 16-bit word is "protected" by an additional 6 bits of Error Checking and Correction (ECC) Flash memory allowing transparent single-bit fault correction and double-bit fault detection during read operations.

Many applications require only small amounts of byte-programmable nonvolatile memory. This application note describes a set of routines that provides 255 bytes of byte-writable Emulated EEPROM (EEE). The routines performing the programming and erasure of the DFlash are interrupt driven, leaving the CPU free to perform other tasks while the

DFlash memory is being updated. In addition, a 256-byte RAM buffer is used to allow the EEE contents to be read at any time.

2 Data storage strategy

To keep the EEE routines simple, without requiring a sophisticated DFlash file system, this design uses 255 of the 256 bytes in a DFlash sector for data storage. The one additional byte in each DFlash sector is used to hold a “flag” byte indicating that the sector contains valid data. When writing a new byte of data to the EEE by calling the EEEWrite() function, the byte is written to the RAM buffer and the entire RAM buffer is copied to an erased DFlash sector. Writing an entire sector each time a single byte in the RAM buffer has changed may seem excessive, but it simplifies the overall data storage management, keeps the software small, and ensures that only a single byte of data will be lost if power is removed or a reset occurs during the update process. Even this frequency of updating the DFlash allows for 800,000 byte writes to the EEE when using all 16 of the DFlash sectors for EEE data storage. This number is derived from the guaranteed minimum of 50,000 erase/program cycles for each DFlash location. Note that even if a word location in a DFlash sector is in the erased state (0xffff) when an erase command is executes, the erase operation still counts as an erase/program cycle.

If an application requires a smaller number of byte writes and requires a portion of the DFlash for other constant data storage, the EEE routines can easily be configured to use as few as three of the sectors. The Configuration Macros section of this document provides details on how to modify the software.

For applications requiring a larger number of writes or for those that must coherently update multiple bytes of data, the EEE software supports a mode that suspends the updating of the DFlash each time a call is made to the EEEWrite() function. In this mode, calls to EEEWrite() only write new data to the EEE’s RAM buffer without updating the DFlash. Once all "suspended" writes have been completed, a call to the EEEFlushBuffer() function will write the RAM buffer to an erased DFlash sector. Note that if an MCU reset occurs prior to the application calling the EEEFlushBuffer() function, any new data written to the RAM buffer via calls to EEEWrite() will be lost.

3 Sector data organization

As mentioned previously, one byte of each DFlash sector is used as a sector status indicator. The EEE routines use the first byte of a sector as the indicator. If the first byte of a sector contains the value 0xaa, the sector contains valid data. Any other value indicates invalid data. The remaining 255 bytes of each sector contain application data.

4 Data integrity

Maintaining the integrity of the EEE data stored in the DFlash array is of primary importance both during EEE operations and in “brownout” conditions. Historically, a brownout condition has been associated with the random or inadvertent removal of power from a system. In embedded systems, “brownout” conditions can also occur as a result of an unintended reset of the MCU. Such unintended device resets can be the result of a software watchdog timer caused by errant software behavior or by external electrical noise entering the system and making its way to the MCU’s reset pin.

When writing new EEE data to the DFlash array, the software uses a write-before-erase approach to copy the entire EEE RAM buffer to an erased DFlash sector before erasing the sector containing the old data. In addition, the data is copied from the RAM buffer into the DFlash sector beginning at the highest address of the sector and RAM buffer. This ensures that the sector status indicator byte is part of the last word to be written to the DFlash. Using this method ensures that if the copy operation is interrupted at any point in the process, the partially copied DFlash sector contents will not be “marked” as valid since the first byte of the sector will not contain 0xaa.

The copy procedure described above can result in one of two different situations if the copy and erase operations are interrupted. The first and most obvious situation occurs when the copy process is interrupted before the DFlash sector valid byte (0xaa) is written. In this case, when the EEEInit() function is called after a power-up or reset, the sector containing the valid status byte will be copied to the EEE RAM buffer and the partially copied sector will be erased.

The second situation occurs if the copy of the RAM buffer to the DFlash sector is properly completed, but the copy/erase process is interrupted before the sector containing the old data can be erased. In this case, when the EEEInit() function is called after a power-up or reset, the DFlash array is checked for a sector containing the value 0xaa in the first byte. When found, the EEEInit() function will always check the next logical sector in the DFlash array for a value of 0xaa in the first byte and if found, its contents are copied to the RAM buffer and the previous sector is erased.

4.1 DFlash error handling

As mentioned previously, each 16-bit word of DFlash is “protected” by 6 bits of ECC Flash memory. This hidden portion of the DFlash array is automatically programmed with an ECC code when a word is written. When data is read from the DFlash array, an ECC code is generated based on the data read from the array. This code is compared to the ECC code that was written to the six ECC bits when the memory was programmed, and it transparently corrects single-bit faults if the codes do not match. Double-bit faults cannot be corrected but only detected.

The EEE routines do not initialize the flash memory module to generate an interrupt when either single- or double-bit faults occur as a result of reading the DFlash array. However, a "dummy" Interrupt Service Routine (ISR), FlashErrorISR(), is provided in the file EEEISRs.c. Since single bit faults are automatically corrected, it may not be necessary for an application to be notified when such an error occurs. However, because double bit errors can only be detected, an application ought to be notified via the ISR since data corruption will have occurred. Setting either the SFDIE or DFDIE bits in the Flash Error Configuration (FERCNFG) register will generate an interrupt when a single or double bit fault occurs, respectively.

5 RAM buffer data organization

The EEE RAM buffer is essentially a mirror of the currently active DFlash sector with the exception of the first byte, which is used for two purposes. The lower nibble of this byte contains the currently active DFlash sector number (0–15). Caching this value in the first byte of the RAM buffer prevents the software from having to scan the DFlash array to determine the currently active sector whenever new data is written to the EEE.

In addition, the uppermost bit of the first byte is used as a flag to indicate the current status of the EEE. If the most significant bit is zero, each time the EEEWrite() function is called, a byte is written to the RAM buffer and the RAM buffer is then copied to the DFlash. If the most significant bit is one, calls to EEEWrite() only write the new data to the EEE’s RAM buffer and do not update the DFlash. As mentioned previously, using this “delayed write” feature allows an application to coherently update multiple bytes of data and/or extend the endurance of the EEE beyond 800,000 byte writes.

6 EEE software performance

As stated previously, because the updating of the DFlash memory is performed via an ISR, the CPU is free to perform other tasks during this period. Table 1 below illustrates the small percentage of time the CPU spends in the FlashCCISR() function during the process of copying the EEE RAM buffer contents to a DFlash sector. Even at the minimum bus clock achievable using the internal reference, 2.0 MHz, the CPU requires only 6.7% of the CPU's bandwidth. While at the device's maximum bus speed of 32 MHz the function requires only 2.3%.

Table 1. EEE software performance

Bus speed (MHz)	Time in ISR (µS)	Time between ISRs (µS)	% of CPU time in ISR
2.0	117	1755	6.7
4.0	58	986	5.9
8.0	29	601	4.8
16.0	14.8	410	3.6
32.0	7.4	316	2.3

7 EEE routine overview

The Application Programming Interface (API) to the EEE routines consists of just five functions located in the file EEE.c and declared in the header file EEE.h. In addition to these five functions, two global variables, EEEState and EEEError, are used by the routines to define the current state of the EEE and indicate any errors that have occurred as a result of EEE operations, respectively. These variables are declared in EEEGlobals.h and defined in EEE.c. Application source code files referencing any of the EEE API functions must #include the files EEEGlobals.h and EEE.h, in that order.

As mentioned in the introduction, the routines performing the DFlash programming and erasure are interrupt driven. A single interrupt service routine (ISR), FlashCCISR() contained in EEEISRs.c, is executed when the Flash Technology Module (FTM) completes a command. The ISR uses a simple state machine to check for FTM errors, update the EEEState and EEEError variables, and, depending on the current EEE state, disable further interrupts.

8 EEE error return codes

The EEEInit(), EEERead(), EEEWrite(), EEEEnableDelayedWrite(), and EEEFlushBuffer() routines each return an error code indicating the success or failure of the requested operation. The returned error is of type EEE_Error as defined in EEEGlobals.h. A list of the returned error codes is presented in Table 2 below.

Table 2. Error return error codes

Error name	Error description
EEE_noErr	Requested operation completed with no errors
EEE_EraseErr	A DFlash sector erase failed
EEE_BlankCheckErr	Used internally by EEEInit() when checking for blank DFlash array
EEE_ProgErr	One or more words of DFlash did not program properly

Table continues on the next page...

Table 2. Error return error codes (continued)

Error name	Error description
EEE_BadAddress	Address of 0x00 passed to EEERead() or EEEWrite()
EEE_Busy	Call to EEEWrite() when EEEState != EEE_Idle
EEE_FlushBufferErr	Call to EEEFlushBuffer() without previous call to EEEEnableDelayedWrite()

In addition, some of these error codes can also be returned in the global EEEError variable at the completion of DFlash program and erase activity (that is, when EEEState == EEE_Idle). Prior to calling the EEERead() or EEEWrite() functions, EEEError should be checked to ensure that no errors occurred during previous operations.

9 Configuration macros

The file EEEGlobals.h contains several typedefs and macros used by the EEE routines. The macro names listed in Table 3 are the only macros that should be modified.

Table 3. Modifiable EEE macros

Macro name	Description
BusClock	S12P bus clock frequency in kHz
EEERAMStart	EEE RAM buffer start address
EEERAMEnd	EEE RAM buffer end address
DFStart	Start address of first DFlash Sector to use for EEE
DFEnd	End address of last DFlash Sector to use for EEE

The BusClock macro MUST be set to the S12P bus clock frequency, in kHz. Failure to configure this parameter properly will result in either inadequate programming or damage to the DFlash cells. The default value is 2000 kHz.

NOTE

If the actual bus clock frequency, in kHz, is not evenly divisible by 1000 and the remainder of the division is *greater than* 600, the value assigned to the BusClock macro must be rounded up to the next value evenly divisible by 1000. For example, an actual bus clock of 2700 kHz would need to be rounded up to 3000 kHz.

EEERAMStart and EEERAMEnd macros define the start and end address of the EEE RAM buffer. By default, the buffer is placed at the end of the on-chip RAM. These addresses MUST define a contiguous 256-byte area of RAM.

The DFStart and DFEnd macros define the start address of the first sector and the end address of the last sector used for EEE, respectively. The addresses must encompass a minimum of three sectors.

10 EEE Application Programming Interface

This section describes the Application Programming Interface (API) for the EEE software.

10.1 InitEEE()

The InitEEE() function performs all of the initialization necessary for the proper operation of the other EEE API functions. This includes:

- FTM clock divider initialization.
- Internal variable initialization.
- Creation of a single active sector containing data of 0xff, if DFlash is completely blank. The sector active status byte is programmed to 0xaa.
- Copying of the currently active sector to EEE RAM buffer.
- Erasure of 'brownout' sector, if applicable.
- Optional initialization of EEE with data.

In addition, if the optional parameter has a value other than NULL, the data pointed to by the parameter is copied into the EEE RAM buffer and then programmed into the DFlash.

10.1.1 Function prototype

EEE_Error EEEInit(const unsigned char *DefaultData);

Argument	Type	Description
DefaultData	const unsigned char *	Pointer to a 256-byte array of unsigned characters

If the first logical sector of the DFlash array is the currently active sector and the 255 bytes of data in that sector contain 0xff and DefaultData is not equal to the value NULL, then the data pointed to by DefaultData is copied into the EEE RAM buffer and then programmed into the next logical sector of the DFlash array.

NOTE

The parameter MUST point to a 256 byte unsigned char array. The value of the first byte of the array is not used. The remaining 255 bytes are copied into the RAM buffer and programmed into a DFlash sector.

10.1.2 Return values

Type	Possible values
EEE_Error	EEE_noErr, EEE_ProgErr

10.1.3 Comments

After the EEEInit() function initializes the Flash Module Clock Divider register (FCLKDIV), it performs a blank check of the entire DFlash array. If the array is completely blank, the ActiveSectorFlag value is programmed into the first byte of the first sector of the DFlash array and the sector data (0xff) is copied into the EEE RAM buffer.

If the DFlash array is not completely blank, the DFlash is searched, beginning with the first sector of the array, for a sector containing the ActiveSectorFlag value in the first byte. Sectors immediately surrounding this sector are also examined to recover from a possible brownout condition and ensure that the newest data is copied into the EEE RAM buffer.

The EEEInit() function should only be called ONCE after a power-on or other reset.

10.2 EEEWrite()

The EEEWrite() function is used to write data to the EEE RAM buffer and, by default, to write the contents of the RAM buffer to the next available DFlash sector.

10.2.1 Function prototype

EEE_Error EEEWrite(unsigned char EEEAddress, unsigned char Data);

Argument	Type	Description
EEEAddress	unsigned char	Address of the EEE data relative to the start of the RAM buffer. Valid address values are 1–255.
Data	unsigned char	Data to be written to the EEE.

10.2.2 Return values

Type	Possible values
EEE_Error	EEE_noErr, EEE_BadAddress, EEE_Busy

10.2.3 Comments

Calls to the EEEWrite() function should only be made when the global variable EEEState contains the value EEE_Idle. If EEEState contains any other value, the data passed in the parameter Data will not be written to the EEE RAM buffer and the error EEE_Busy will be returned.

If the function EEEEnableDelayedWrite() has previously been called, the data passed in the parameter Data will be written to the EEE RAM buffer, but the DFlash will not be updated.

10.3 EEERead()

The EEERead() function is used to read data from the EEE RAM buffer.

10.3.1 Function prototype

EEE_Error EEERead(unsigned char EEEAddress, unsigned char *Data);

Argument	Type	Description
EEEAddress	unsigned char	Address of the EEE data relative to the start of the RAM buffer. Valid address values are 1–255.

Table continues on the next page...

Argument	Type	Description
Data	unsigned char *	Pointer to an unsigned char variable.

10.3.2 Return values

Type	Possible values
EEE_Error	EEE_noErr, EEE_BadAddress

10.3.3 Comments

Calls to the EEERead() function can be made regardless of the current state of the EEE. In addition, data can be read directly from the EEE RAM buffer beginning at EEEStart + 1.

10.4 EEEEnableDelayedWrite()

The EEEEnableDelayedWrite() function is used to disable the automatic updating of DFlash each time the EEEWrite() function is called.

10.4.1 Function prototype

EEE_Error EEEEnableDelayedWrite(void);

Argument	Type	Description
None	N/A	N/A

10.4.2 Return values

Type	Possible values
EEE_Error	EEE_noErr, EEE_Busy

10.4.3 Comments

Calls to EEEEnableDelayedWrite() should only be made when the global variable EEEState contains the value EEE_Idle.

After calling EEEEnableDelayedWrite(), all data written to EEE via calls to EEEWrite() will only be written to the EEE RAM buffer.

After the call of EEEEnableDelayedWrite(), data may be written directly to the EEE RAM buffer beginning at location EEEStart + 1.

10.5 EEEFlushBuffer()

The EEEFlushBuffer() function is used to immediately write the contents of the EEE RAM buffer to the next logical DFlash sector.

10.5.1 Function prototype

EEE_Error EEEFlushBuffer(void);

Argument	Type	Description
None	N/A	N/A

10.5.2 Return values

Type	Possible values
EEE_Error	EEE_noErr, EEE_Busy, EEE_FlushBufferErr

10.5.3 Comments

Calls to EEEFlushBuffer() should only be made when the global variable EEEState contains the value EEE_Idle.

11 Revision history

Revision	Description of changes
0	Initial version
1	—Added "DFlash error handling" section. —Added "EEE software performance" section. —In "Modifiable EEE macros" table, added DFStart and DFEnd rows. Changed EEEStart to EEERAMStart and EEEEnd to EEERAMEnd. —Editorial changes and improvements.

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4483
Rev. 1, 03/2012