Emulating Data EEPROM for PIC18 and PIC24 Microcontrollers and dsPIC® Digital Signal Controllers

AN1095

Introduction

Microchip® Technology Inc. has expanded its product portfolio to include a variety of cost-effective Peripheral Interface Controller (PIC®) Microcontrollers (MCUs) that do not include internal data Electrically Erasable Programmable Read-Only Memory (EEPROM).

Many applications store nonvolatile data in Flash program memory using table read and write operations. Applications that frequently update this data require higher endurance than the Flash endurance specified for MCUs and Digital Signal Controller (DSC) devices.

Using an external serial EEPROM device is another option. However, this approach may not be suitable for cost-sensitive or pin-constrained applications.

This application note introduces a third alternative to address these challenges. The proposed algorithm provides an interface similar to internal data EEPROM, uses available program memory, and increases endurance by up to 500 times.

Note: To use this solution, the device must have word write or double word-word or quad-word write capability. Refer to the specific device data sheet to verify the availability of this feature.

Definition of Terms

- Page – The minimum amount of program memory affected by an erase operation.
- Row – The maximum amount of program memory affected by a programming operation.
- Erase/Write Cycle – The number of erase and write operation pairs.
- Endurance – A specification indicating the maximum number of erase/write cycles and associated conditions.
- Retention – A specification indicating the minimum time and associated conditions for the retention of data in Flash program memory.
- Effective Endurance – The improved endurance of the emulated data EEPROM as a result of using an efficient programming algorithm.
- Current (Active) Page – A page in program memory that is being written and read by the data EEPROM emulation algorithm.
- Packed Page – The new current page after the pack routine is complete.
- Page Status – Program memory locations at the beginning of the current page that store data EEPROM emulation status. The PIC18 implementation uses two locations and the PIC24/dsPIC33 devices with single word write uses one location, devices with double word write uses two locations and devices with quad-word write uses four locations.

Table of Contents

1. Theory of Operation

The algorithm in this application note supports selectable, multiple emulated data EEPROMs with a total size of up to multiples of 255 locations, with a single address space, ranging from 0 to the total size of the emulated data EEPROMs minus one (see the following note).

For example, if the application needs 260 total EEPROM locations, this will require more than one

EEPROM bank. The user could configure to use two EEPROM banks with 130 EEPROM data per bank. The total EEPROM address range available will be 0 to 259.

Note: The PIC18 and PIC24/dsPIC33 implementations support multiple EEPROM banks. Each EEPROM can have a maximum of 255 addresses. Therefore, the total addresses are from 0 to N x 255 – 1, where N = the number of EEPROM banks.

PIC18 implementation supports 8-bit data and multiple EEPROM banks. PIC24/dsPIC33E/C implementation supports 16-bit data and multiple EEPROM banks, dsPIC33A implementation support 32-bit data and multiple EEPROM banks. Due to architectural differences of the program memory the emulated data EEPROM information is stored differently for 8-bit, 16-bit and 32-bit implementations. For these formats, refer to the following tables.

Table 1-1. PIC18 Data EEPROM Information Format in Program Memory

Bits 15-8	Bits 7-0
Data EE Data	Data EE Address

Table 1-2. PIC24/dsPIC33E/C Data EEPROM Information Format in Program Memory

Bits 23-16	Bits 15-0
Data EE Address	Data EE Data

The algorithm takes advantage of the PIC MCU ability to self-program a single location of the program memory. This location is an 8-bit operation for PIC18, and either an 8-bit or 16-bit operation for PIC24/dsPIC33E/C, depending on whether an odd/even address is being written.

Table 1-3. dsPIC33A Data EEPROM Information Format in Program Memory

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x00	0x00	0x00	DEE Address
DEE Data			
0xFFFFFFFF			
0xFFFFFFFF			

Note: In dsPIC33A devices, each DEE Address stores 32 bit of DEE Data.

In devices with ECC and quad-word write, DEE algorithm uses LSB of first instruction to store the DEE address, second instruction is used to store the 32 bit data and the remaining 2 instructions are unused.

1.1 MPLAB® Code Configurator Data EEPROM Emulation Support

Microchip has added MPLAB Code Configurator Data EEPROM module (MCC DEE) support based on the PIC24/dsPIC firmware implementation of this application note. This helps ease the firmware configuration and setup. For more information about where to download, see the Software section.

A few notable differences between the MCC DEE version and the original application note's implementation:

File Names:

`dee.c` (equivalent to '`DEE Emulation 16-bit.c`' referred in this document).

`dee.h` (equivalent to '`DEE Emulation 16-bit.h`' referred in this document).

Function Names:

`DEE_Init`, `DEE_Read`, `DEE_Write` (equivalent to `DataEEInit`, `DataEERead` and `DataEEWrite` referred in this document)

Notes:

1. For more information on program memory organization, refer to the specific device data sheet.
2. In the dsPIC33 and PIC24 devices, the EEPROM emulation algorithm provides an option to use auxiliary Flash program memory instead of primary Flash program memory. The user can select this option by uncommenting the following line in the include file, `DEE Emulation 16-bit.h`": `#define __AUXFLASH 1`.
3. In all dsPIC33 and PIC24 devices containing the Flash Error Correction Coding (ECC) feature, the user must uncomment the following line in the include file, "`DEE Emulation 16-bit.h`": `#define __HAS_ECC`".

   In all such devices, every 2nd instruction word in each double-word pair will be unused by the data EEPROM emulation software. For example, if Flash memory address 0x2004 is used for data EEPROM emulation, then Flash memory address 0x2006 will not be used. This change is handled automatically in the MCC DEE version.
4. The PIC24/dsPIC33 implementation uses PSV memory mapping, so the total number of EEPROM addresses supported is limited. For the total effective endurance to increase, the total number of EEPROM addresses may need to decrease. PSV memory mapping is not applicable to the dsPIC33A devices.

1.2 PIC24/dsPIC33 Case Example

To understand how the algorithm works, a simple case example for the PIC24/dsPIC33 devices is described in this section.

After the first page of each EEPROM bank is initialized, the first location is reserved for the page status information. This indicates whether a page is active or expired, and how many erase/write cycles have been performed. This information is not directly accessible by the user, but is used by the algorithm to find the available pages and update status flags. After initialization, the first page is designated as the active page.

In this example, a write operation has been performed to store a data value of 0x0202 to data EEPROM address, 0x2. As shown in Table 1-4, this information is stored in the first available location in the page. As more writes are performed, the algorithm continues to write the information similarly, as shown in Table 1-5 through Table 1-7.

In this example, the data EEPROM address 0x7 is written with 0x0707, 0x2 is updated to 0x2222 and address 0xA is written with 0x0A0A.

In Table 1-8, the last location in the page is written with a rewrite to addresses, 0x7 to 0x7777. The data EEPROM information will move to the next available page because the currently active page is full. This new page is referred to as the packed page. The pack routine performs this task. Because only the most current data for each data EEPROM address are needed, the amount of information decreases.

After the data are moved, this page is designated as the current page. If the current page has incremented through all allocated pages in program memory, the erase/write count is incremented as shown in Table 1-9. The page is now ready to store more information through write operations. The PIC18 algorithm works in a similar way, but instead of reserving one location of program memory for page status information, two locations are used. Also, 8-bit data are stored instead of 16-bit data.

Table 1-4. WRITE Data EEPROM (0x0202, 2)

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0000
Page + 2	2	0x0202
Page + 4	0xFF	0xFFFF
Page + 6	0xFF	0xFFFF
Page + 8 ... Page + 1022	0xFF	0xFFFF

Table 1-5. WRITE Data EEPROM (0x0707, 7)

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0001
Page + 2	2	0x0202
Page + 4	7	0x0707
Page + 6	0xFF	0xFFFF
Page + 8 ... Page + 1022	0xFF	0xFFFF

Table 1-6. WRITE Data EEPROM (0x2222, 2)

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0000
Page + 2	2	0x0202
Page + 4	7	0x0707
Page + 6	2	0x2222
Page + 8 ... Page + 1022	0xFF	0xFFFF

Table 1-7. WRITE Data EEPROM (0x0A0A, 0xA)

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0000
Page + 2	2	0x0202
Page + 4	7	0x0707
Page + 6	2	0x2222
Page + 8	0xA	0x0A0A
...	...	...
Page + 1022	0xFF	0xFFFF

Table 1-8. WRITE Data EEPROM (0x7777, 7)

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0000
Page + 2	2	0x0202
Page + 4	7	0x0707

.....continued		
Page Address	**Data EE Address**	**Data EE Data**
Page + 6	2	0x2222
Page + 8	0xA	0x0A0A
...	...	...
Page + 1022	**7**	0x7777

Table 1-9. Page After Pack Operation

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0001
Page + 2	2	0x2222
Page + 4	7	0x7777
Page + 6	0xA	0x0A0A
Page + 8 ... Page + 1022	0xFF	0xFFFF

Only one erase/write cycle is consumed for the page as each location within the page is programmed once prior to the page erase. As a result, the algorithm multiplicatively improves the emulated data EEPROM effective endurance.

The previously filled page is erased only after the latest information has been programmed into the next available page and successfully verified. Through this process, the information is always stored in nonvolatile memory, which minimizes the effects of an unexpected loss of power.

As the program memory page is filled sequentially from beginning to end, the algorithm assumes the most current data EEPROM information is the closest instance to the end of the page. To simplify the read operation, the search begins at the end of the current program memory page and works toward the start of the page – looking for the specified data EEPROM address.

When a match is found, the associated data are returned for the first instance of the provided address. If the address is not found, the return value of all ones, 0xFF or 0xFFFF, is returned to emulate the result of an unwritten address in an independent data EEPROM.

Device with Double-Word Write Use Case

The following table list the device with double-word write use case.

Table 1-10. Device with Double-Word Write Use Case

Page Address	Data EE Address	Data EE Data
Page + 0	Page Status[23:16]	0x0001
Page + 2	0xFFFFFF	
Page + 4	0x10	0x2222
Page + 6	0xFFFFFF	
Page + 8	0x20	0x7777
Page + 10	0xFFFFFF	
Page + 12	0x30	0x0A0A
Page + 14	0xFFFFFF	
...	...	...
Page + 1022	0xFF	0xFFFF

Device with Quad-Word write Use Case

The following table list the device with quad-word write use case.

Table 1-11. Device with Quad-Word Write Use Case

Page Address	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Page + 0	0x00	Page Status[23:16]	0x0001	
Page + 4	0xFFFFFFFF			
Page + 8	0xFFFFFFFF			
Page + 12	0xFFFFFFFF			
Page + 16	0x00	0x00	0x00	0x10
Page + 20	0x2222			
Page + 24	0xFFFFFFFF			
Page + 28	0xFFFFFFFF			
Page + 32	0x00	0x00	0x00	0x20
Page + 36	0x7777			
Page + 40	0xFFFFFFFF			
Page + 44	0xFFFFFFFF			
Page + 48	0x00	0x00	0x00	0x30
Page + 52	0x0A0A			
Page + 56	0xFFFFFFFF			
Page + 60	0xFFFFFFFF			
Page + 64	0xFFFFFFFF			
Page + 68	0xFFFFFFFF			
...	...	...	...	...
Page + 1016	0xFFFFFFFF			
Page + 1020	0xFFFFFFFF			

1.3 Status Flags

Status flags have been provided to indicate whether an error or warning condition occurs during the emulation process. These indicators are accessed in the Data EEPROM Flags register; all flags are active-high.

Note: All EEPROM banks affect the same status flags.

The status bits and return values are defined as follows:

- addrNotFound(0xFF/0xFFFF) – A read operation occurred on a previously unwritten data EEPROM address.
- expiredPage(0x1) – The program memory erase/write cycle count has exceeded the user-defined limit. The algorithm will attempt to execute the write operation.
- packBeforePageFull(0x2) – The pack routine was called before the currently active page was full. The routine will attempt to move the latest data EEPROM information to the packed page, even though the active page is not full.
- packBeforeInit(0x3) – The pack routine was executed before the initialization routine. The pack operation was aborted.
- packSkipped(0x4) – A page was written beyond the page boundary. This may be a result of the pack routine not being executed properly. The pack operation was aborted.

- illegalAddress(0x5) – There was an attempt to write/read with a data EEPROM address equal to or greater than the size of data EEPROM. The read/write operation was aborted.
- pageCorrupt(0x6) – The page status information was corrupted. The current operation was aborted.
- writeError(0x7) – The information that was written into program memory failed the verification. The current operation was aborted.

The status flags differ in severity and how they are serviced. The informational status flags are expected to occur during normal processing and are serviced by simply clearing the flag with the associated macro. These include: addrNotFound, packBeforePageFull and illegalAddress flags.

Warning status flags indicate a condition has been exceeded but processing will continue. This includes the expiredPage status flag. With this flag set, the algorithm will attempt to process read and write requests, but the flag will be set after each operation.

The most severe flags are the system error status flags. These imply either the integrity of the data EEPROM information has been compromised and/or the algorithm cannot continue until the offending condition has been resolved. These include packBeforeInit, pageCorrupt and writeError flags.

To avoid a packBeforeInit event, ensure the initialization routine, `DataEEInit` or `DEE_Init` (MCC DEE), is called before performing any other emulation routine. Because this routine accesses the current state of the emulation process, it will take action only if it is required. Therefore, it can be called at any time during data EEPROM emulation.

The pageCorrupt and writeError flags indicate that a write operation failed to verify and the current operation was aborted. If this occurs, the integrity of the data EEPROM information has been compromised. No further emulation operations must be attempted. The only recourse is to erase all of the pages of program memory reserved for data EEPROM emulation and attempt to reinitialize them.

Macros are available to retrieve and clear the status flag values. Status flags are cleared only by the user. No operation is affected by the value of any flag, but the flag's value will indicate whether an operation has completed successfully.

Example 1-1. Macros Naming Convention Example

Macros: "`Getx`" "`Setx y`"

`x` = Flag name

`y` = Value assigned to flag

All of the flags can be read or cleared in a single operation using the 8-bit character, `dataEEFlags.val`.

1.4 Page Status

Each program memory page reserves space for the page status – using the first two-word locations for the PIC18 implementation or the first location for the PIC24/dsPIC33 implementation. The status contains information about the page, whether it is expired or active, and the number of erase/write cycles performed.

Note: Applications with bootloaders must not modify any of the pages that are used for data EEPROM emulation.

These values are used by the algorithm to monitor and control page information, and are not directly accessible by the user. For formats of PIC24/dsPIC33 and PIC18 page status information, refer to the following tables.

Table 1-12. Page Status for PIC24/dsPIC33 Algorithm

U-1	U-1	U-1	R-1	R-1	R-1	U-1	U-1
—	—	—	Page Expired	Page Current	Page Available	—	—
bit 23							bit 16

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
Page Erase/Write Count							
bit 15							bit 8

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
Page Erase/Write Count							
bit 7							bit 0

Bit address	Description
bit 23-21	**Unimplemented**: Read as '1'
bit 20	**Page Expired** • 1 = Page has not expired • 0 = Page has expired
bit 19	**Page Current** • 1 = Page is not current • 0 = Page is current
bit 18	**Page Available** • 1 = Page is available • 0 = Page is not available
bit 17-16	**Unimplemented**: Read as '1'
bit 15-0	**Page Erase/Write Count**: Number of Page Erase/Write Cycles

Legend:

- R = Reserved bit
- U = Unused bit, read as '1'
- -n = Value prior to initialization
- 1 = Bit is in erased state
- 0 = Bit is in programmed state

Table 1-13. Page Status for PIC18 Algorithm (Start of Page)

U-1	U-1	U-1	U-1	U-1	U-1	U-1	U-1
—	—	—	—	—	—	—	—
bit 15							bit 8

U-1	U-1	U-1	U-1	U-1	R-1	R-1	R-1
—	—	—	—	—	Page Expired	Page Current	Page Available
bit 7							bit 0

Bit address	Description
bit 15-3	**Unimplemented**: Read as '1'
bit 2	**Page Expired** • 1 = Page has not expired • 0 = Page has expired

.....continued	
Bit address	**Description**
bit 1	**Page Current** • 1 = Page is not current • 0 = Page is current
bit 0	**Page Available** • 1 = Page is available • 0 = Page is not available

Legend:

- R = Reserved bit
- U = Unused bit, read as '1'
- -n = Value prior to initialization
- 1 = Bit is in erased state
- 0 = Bit is in programmed state

Table 1-14. Page Status for PIC18 Algorithm (Start of Page + 2)

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
Page Erase/Write Count							
bit 15							bit 8

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
Page Erase/Write Count							
bit 7							bit 0

Bit address	Description
bit 15-0	**Page Erase/Write Count:** Number of Page Erase/Write Cycles.

Legend:

- R = Reserved bit
- U = Unused bit, read as '1'
- -n = Value prior to initialization
- 1 = Bit is in erased state
- 0 = Bit is in programmed state

2. Initialization Operation

The initialization routine, `DataEEInit` or `DEE_Init` (MCC DEE), must be called before any other data EEPROM operation can occur; this initializes all of the EEPROM banks. If the routine determines that program memory has not been initialized for emulation, it will find the first allocated page of program memory and initialize its status information. Thereafter, the read and write functions may be called as needed.

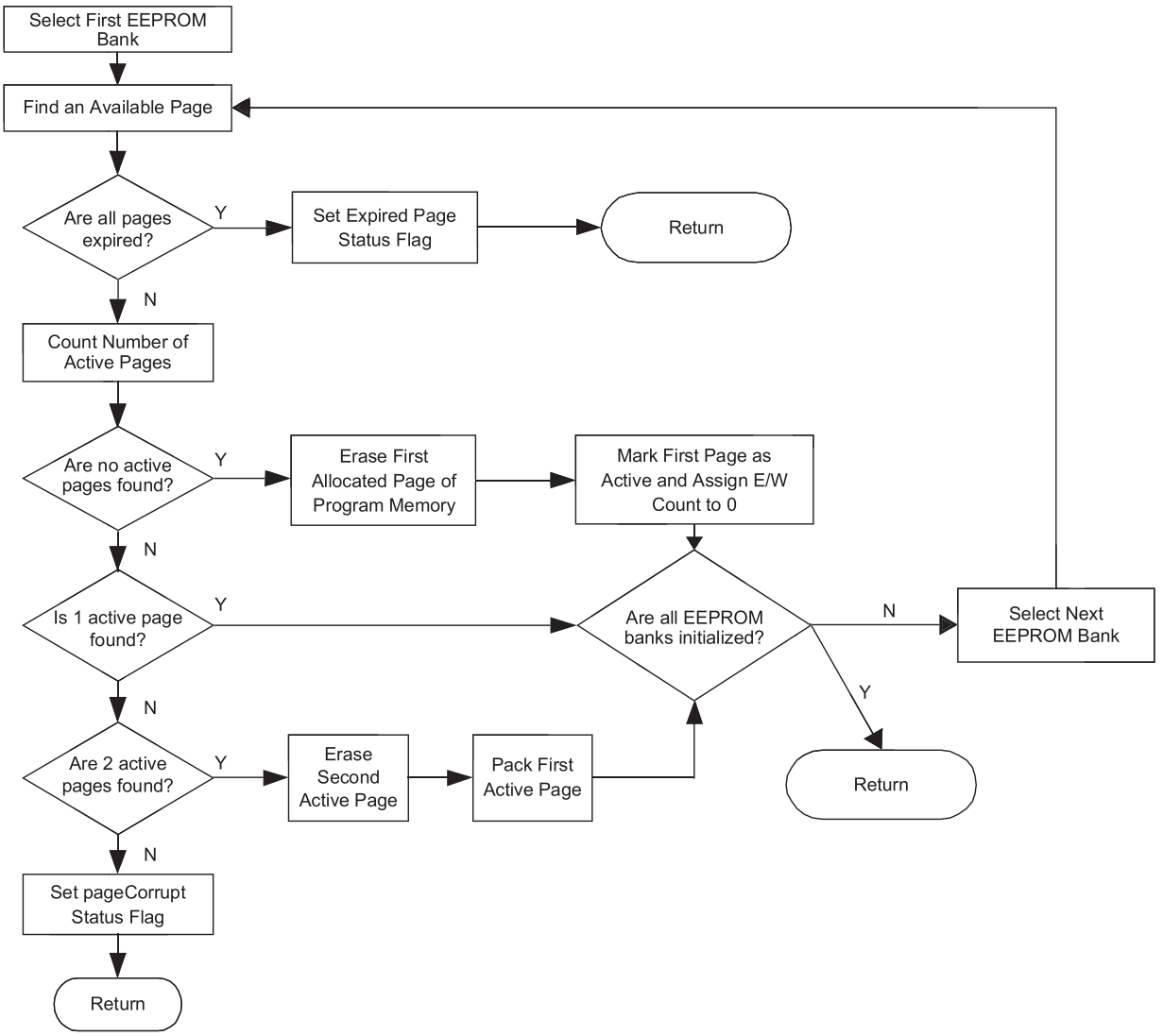
The routine may also determine whether data EEPROM emulation is already underway. If so, one of three scenarios may occur:

- If only one active page is found, the routine assumes a Reset occurred. No action is taken and the routine exits normally. Any read/write operation that may have been active during the Reset must be repeated.
- If two active pages are found, the routine assumes that an unexpected Reset occurred during a pack operation. The routine will erase the second active page and call the pack routine to permit the refresh to complete.
- If more than two active pages are found, the routine assumes program memory has been corrupted by the application code and sets the pageCorrupt flag.

It is important to monitor the page status bits, as well as the RCON and NVMCON (PIC24/dsPIC33) or EECON1 (PIC18) registers. By doing so, the application can respond appropriately to Resets and supply voltage changes.

The following figure shows the flowchart of the initialization routine.

Figure 2-1. Emulation Data EEPROM Initialization



3. Read Operation

The `DataEERead` or `DEE_Read` (MCC DEE) function retrieve data EEPROM information. It returns the data associated with the data EEPROM address. If the provided address is equal to or greater than the amount of defined data EEPROM, the illegalAddress flag is set and a value of all '1's is returned. This return value mimics the response of dedicated data EEPROM, where an unwritten address returns an erased value.

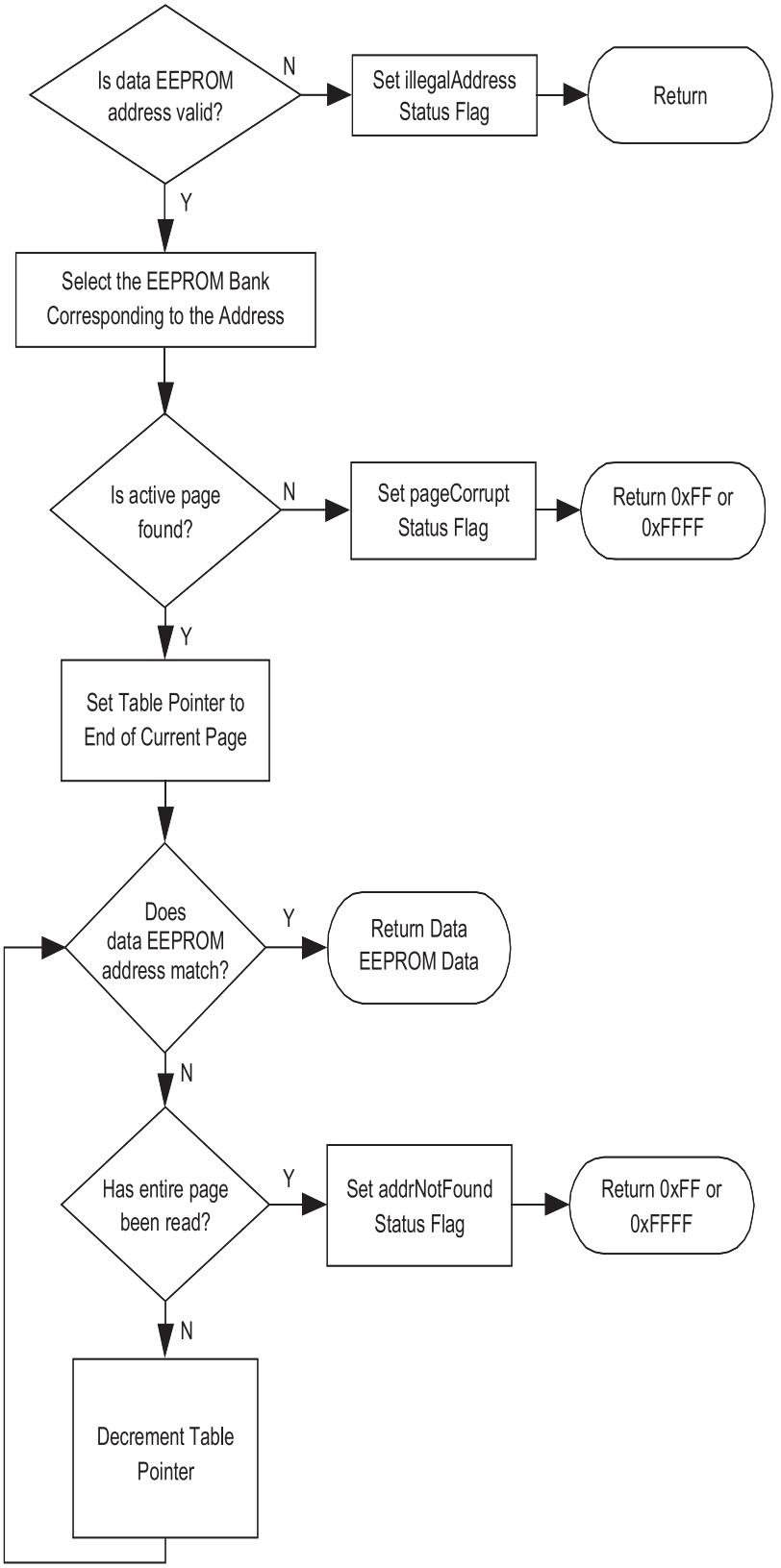
The routine then searches for the active page in the EEPROM bank corresponding to the address. Once located, the active page is searched for an address match, starting from the last location in the page. For more information on how data EEPROM information is stored in program memory, see Table 1-1, Table 1-2 and Table 1-3.

Because the page is filled sequentially, the latest data EEPROM information will be the first location found with the reverse search. Once found, the routine returns the data EEPROM data value associated with the data EEPROM address.

If an active page is not found, the pageCorrupt flag is set.

The following figure shows the flowchart of the read operation.

Figure 3-1. Read Operation



4. Write Operation

To write emulated data EEPROM, the application uses the `DataEEWrite` or `DEE_Write` (MCC DEE) function. Like the read function, it verifies that the data EEPROM address is between 0 and one less than the size of the emulated data EEPROM. If an unimplemented address is supplied, the illegalAddress flag is set and the write operation is aborted.

The routine then searches for the active page of the EEPROM bank corresponding to the address. After the active page is located, a read operation is performed. To minimize the number of erase/write cycles, the value is programmed only if it has changed.

If an active page is not found, the pageCorrupt flag is set and a non-zero value is returned.

A forward search of the active page returns the offset for the next available address. If the next available address is equal to, or greater than, the last address in the page, the packSkipped flag is set and the write operation is aborted. Otherwise, the data EEPROM information is written to the next available address in the page.

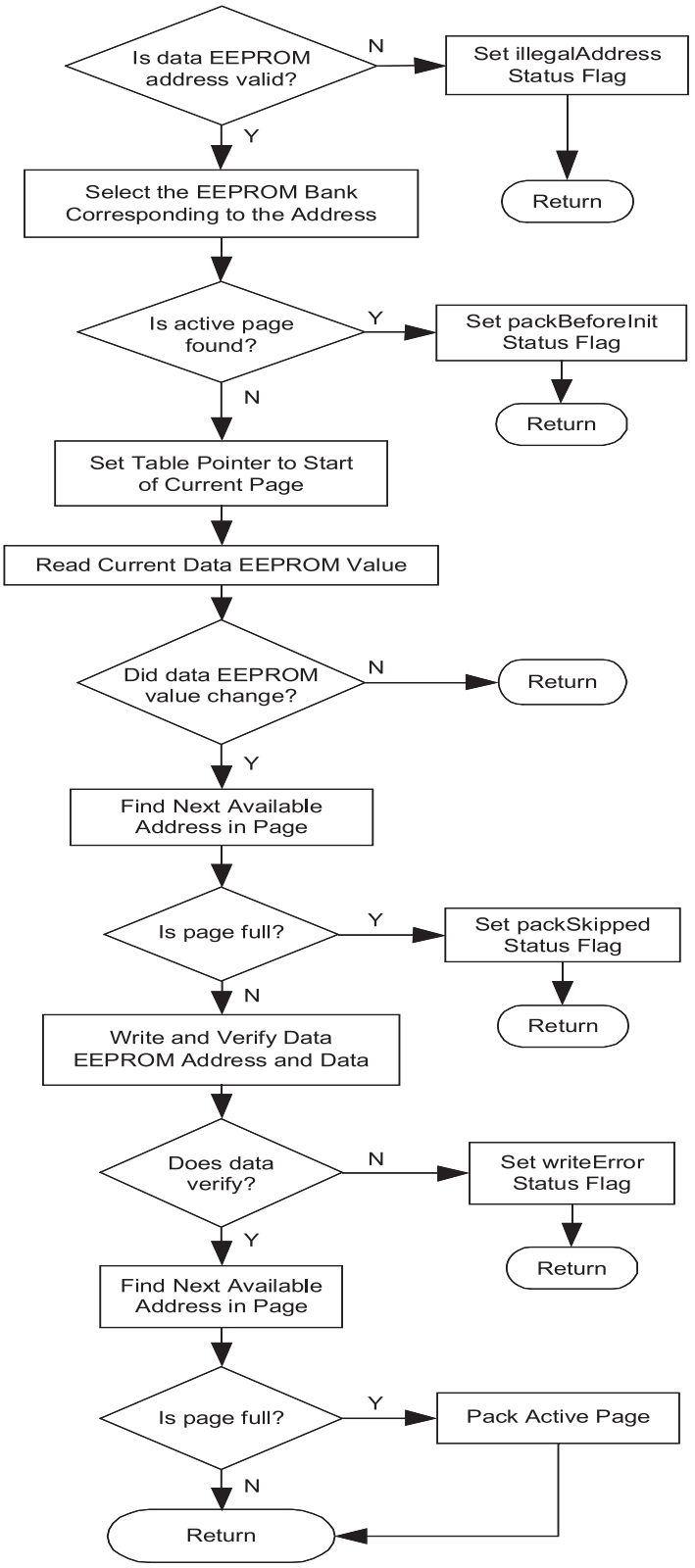
If the information does not verify, the writeError flag is set and a non-zero value is returned. The user can attempt to rewrite the data or respond as needed.

The algorithm is designed to maintain at least one available location in the active page for the next write operation. After a successful verification of the write operation, the pack routine is called if no available locations remain.

After the routine completes successfully, a zero value is returned.

The following figure shows the flowchart of the write operation.

Figure 4-1. Write Operation



Note:
In PIC24/dsPIC33 devices with double-word write and Flash Error Correction Coding (ECC) feature, every 2nd instruction word in each double-word pair will never be modified by the `DataEEWrite` or `DEE_Write`(MCC DEE) function (but will be erased when the Flash page is erased).

In PIC24/dsPIC33 devices with quad-word write and Flash ECC feature, only the first and second instruction words are used and the remaining 2 instructions words are not used.

5. Pack Operation

The pack routine, `PackEE` or `DEE_Pack` (MCC DEE, not a public function), is called from either a `DataEEWrite` or `DEE_Write` (MCC DEE) after the current page is filled, or by `DataEEInit` or `DEE_Init` (MCC DEE) to initialize program memory for data EEPROM emulation. It can also be called by the user, which may benefit time-sensitive applications. Because the routine performs multiple Flash operations which stall the CPU, it can be executed at a time more convenient for the application. The disadvantage in doing so is that effective endurance is reduced because unwritten program memory locations are spent.

The function begins by reading the Page Current status bit, for each page of program memory allocated, for emulation to find the filled page. If it is not found, the routine assumes that the pack function was called prior to initializing program memory. The packBeforeInit flag is then set and the operation is aborted.

A new page is needed to program the latest data EEPROM information. This page is referred to as the packed page. It always tries to assign the next page in program memory as the packed page. If all of the available pages have reached the user-specified erase/write limit, the expiredPage flag is set and the routine will continue the pack operation.

The erase/write counter is only incremented when the packed page rolls around to be the first page allocated for data EEPROM memory. At this point, every page has the same number of erase/write cycles. If the erase/write counter exceeds the specified limit, the page status is marked as expired by programming the Page Expired bit in the Page Status register.

As the number of pages of program memory and the erase/write limits are defined at compile time, all pages will expire sequentially. A search of every defined data EEPROM address is made into the active page using the read function. This information is written into the program memory write latches. After a row of write latches is filled, the data are programmed until all information is stored into the packed array. If the last row is not full, the remaining write latches are written to all '`1`'s prior to programming.

If the active page is not full, it is assumed the routine was called by the user. At this point, the packBeforePageFull status flag is set and the routine continues into the programming portion.

After all of the data have been programmed into the packed page, the current page data are read and compared to the packed page data. If a mismatch occurs, the writeError flag is set and the function exits with an error code. The page status information is also programmed and verified. If the verify routine is successful, the active page is erased and the packed page is designated as the new active page.

A zero return value from the pack function indicates the routine completed normally.

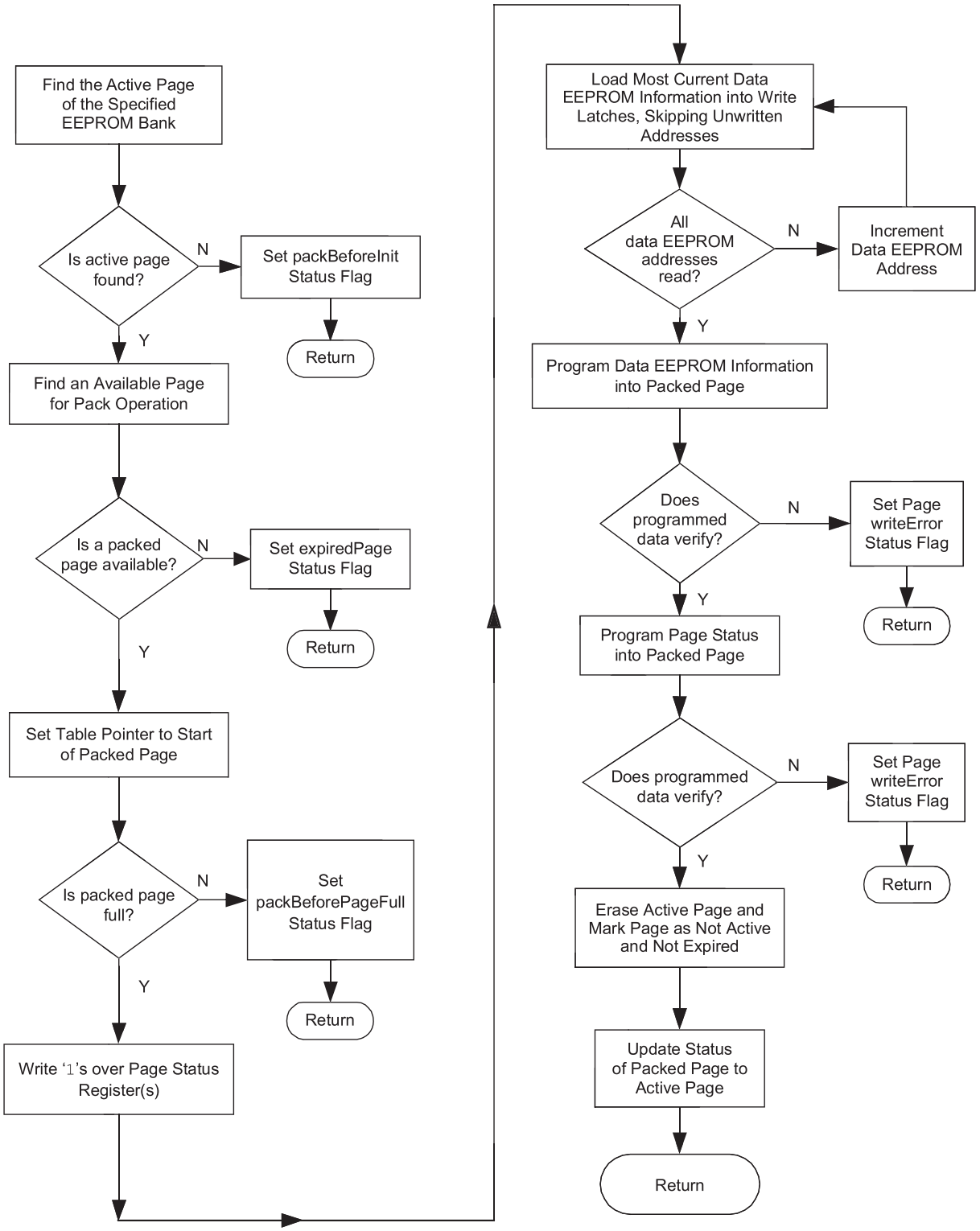
Notes:

1. The maximum data EEPROM size must be no greater than N x 255, where N = number of EEPROM banks.
2. In PIC24/dsPIC33 devices, word (single word, double-word and quad-word) write operations are used to program the packed page.
3. For PIC24/dsPIC33, choosing 255 as the maximum data EEPROM size per bank works for most devices.

   For devices with no ECC and have 256 or fewer instruction words per Flash page size, and devices with ECC and have 512 or fewer instruction words per Flash page size, a smaller maximum data EEPROM size per bank than 255 must be used to allow at least half the page to be available after a worst-case DEE packed operation (that is, 128).

The following figure shows the flowchart of the pack operation.

Figure 5-1. Pack Operation



6. Performance

6.1 Effective Endurance

Determining effective endurance is not a trivial calculation because it is dependent on many factors. Traditionally, endurance is defined as the number of times a single address can be safely written. This definition does not apply to emulated data EEPROM for a few different reasons.

First, writing a data EEPROM address five times does not mean five erase/write cycles of endurance were consumed. From the perspective of the program memory, five writes were made to five different program memory addresses. These five writes will not cost any additional endurance cycles until the page is filled and the pack routine is called.

Second, calculating effective endurance is more than simply multiplying program memory page size and the size of the emulated data EEPROM. The entire page is not available for emulation. The page status information is stored in the beginning of the page, which is the smallest writable block allowed by the device. In addition, more locations are consumed after the pack routine depending on how many data EEPROM addresses were written. As a result, writing an address once has a significant impact to endurance because one less location is available after the array is packed.

Based on the discussion to this point, a simplified equation (see Equation 1) can be made for total effective endurance. For more information on the terms, see Definition of Terms.

Working through an example for the PIC24FJ128GA010, this device has a 512-word page. For this device, the algorithm reserves one location for page status.

Equation 2 provides the formula for calculating two pages of program memory, ten locations of emulated data EEPROM and the typical endurance limit.

An average effective endurance can be calculated by dividing the total effective endurance by the size of the emulated data EEPROM bank, but this does not tell the whole story. It assumes that every data EEPROM address is updated at the same rate. In most applications, this is not true. Some data, such as calibration data or user information, may be rarely updated, while sensor information can be written more frequently. Addresses written more often will consume a greater amount of program memory endurance. Therefore, how writes are distributed across the data EEPROM addresses significantly affects effective endurance. Ratios could be assigned to each address to create a more accurate calculation, but this is still only an approximation. It is difficult to predict how often each address will be written during an application's lifetime.

Note: The EEPROM banks are considered as different EEPROMs, each having its own effective endurance.

Equation 1:Total Effective Endurance = (Effective Page Size − Page Status Size − Size of One Data EEPROM Bank) × Number of Pages × Endurance

Equation 2:Total Effective Endurance = (512 − 1 − 10) × 2 × 1000 = 1002000 Cycles

Note: In devices with double-word write and ECC, only one instruction store the DEE address and DEE data, remaining one instruction word is unused. The effective page size would be 1/2 of the actual page size.

Note: In devices with quad-word write and ECC, one instruction word store the DEE Address, one instruction word store the actual DEE data, remaining 2 instructions are unused. The effective page size would be 1/4 of the actual page size.

6.2 CPU Stall

During program memory operations, the CPU stalls until the write operation is complete. The algorithm performs a program memory write operation in the `DataEEWrite` or `DEE_Write` (MCC DEE) routine. When that routine is called, the CPU Stall time will depend on whether the page is filled.

If the write operation does not fill the page, the Stall time will be shorter – approximately the amount of time to program one word. If the write operation fills the current page, the delay will be longer. This is because the pack routine may perform multiple row or double-word program operations on the packed page and an erase operation on the active page.

The `GetNextAvailCount` function can be used to determine how full the current page is and how many writes can be made before the pack routine is invoked. This function returns the offset of the next available address in the current page. The range of values is between two and twice the page size times. The pack routine can be called before the current page is full, if desired. It can be helpful to perform a pack operation prematurely to minimize the impact of a CPU Stall time.
Note: For more information on program memory operation timings, refer to the specific device data sheet.

7. Application

To implement data EEPROM emulation in your application, follow one of these checklists:

- PIC18 Emulation Checklist
- PIC24/dsPIC33 Emulation Checklist

Both 8-bit and 16-bit algorithms require approximately 2.7 KB of program memory, excluding the memory reserved for data EEPROM information. They also require approximately 82B of data memory.

Program memory for storing data EEPROM information is allocated with a two-dimensional array. The array size depends on the device's page erase size and the number of pages reserved for emulation. For the 16-bit implementation, the compiler automatically aligns the array to the start of the next available page of program memory during compilation. For the 8-bit version, you must manually specify the starting address for an available page. If this address does not align with a page boundary, a compile-time error will occur. The array determine program memory addresses for table operations. If the required amount of program memory is unavailable, a compile-time error will also occur.
Note: For more information on PIC18FXXJ and many 16-bit devices, see the specific device data sheet. The last page of program memory stores the configuration word information. It cannot be allocated for data EEPROM emulation.

The code size and data memory requirements remain nearly unchanged regardless of the emulated data EEPROM size.

These algorithms are designed to be configurable, not only for different devices, but also for specific endurance needs (see Equation 1 and Equation 2). If greater endurance is needed, more pages can be allocated to program memory. Alternatively, the erase/write limit can be set to the typical endurance rating instead of the minimum. These options are selected by simply changing constants in the associated include file.

7.1 PIC18 Emulation Checklist

Perform the following steps to modify the `NoFilDee.inc` file. For more information on program memory, refer to the specific device data sheet.

1. Specify emulation page start address in `EMULATION_PAGES_START_ADDRESS`.
2. Specify the required number of EEPROM banks in `DATA_EE_BANKS`. The maximum is limited by the device's memory.
3. Specify the number of program memory pages per EEPROM bank in `NUM_DATA_EE_PAGES`. The minimum is two pages. A compile-time error occurs if fewer than two pages are specified.
4. Specify the amount of data EEPROM required in `DATA_EE_SIZE`. The maximum value is 255 (0xFE). A compile-time error occurs if this value exceeds 255.
5. Verify the values of `ERASE`, `PROGRAM_ROW` and `PROGRAM_WORD` opcodes.
6. Specify the minimum page erase size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_PAGE` (typically 512).
7. Specify the maximum programming size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_ROW` (typically 64).
8. Select the maximum erase/write cycle count per location in `ERASE_WRITE_CYCLE_MAX`. The maximum value is 65,535. A compile-time error occurs if this limit is exceeded.
9. Add a function call to `DataEEInit` prior to any other operation to emulated data EEPROM.
10. Add the following files in your project:
    - `DEE Emulation 8-bit.c`

- GenericTypeDefs.h
- DEE Emulation 8-bit.h
- NoFilDee.asm
- NoFilDee.inc

7.2 PIC24/dsPIC33 Emulation Checklist

The following checklist is not required when using the MCC DEE version.

Perform the following steps to DEE Emulation 16-bit.h. For more information on program memory, see the specific device data sheet.

1. Specify the required number of EEPROM banks in DATA_EE_BANKS. The maximum number depends on the device's memory
2. Specify the amount of data EEPROM needed for each bank in DATA_EE_SIZE. The maximum value is 255 (0xFE). A compile-time error occurs if this value exceeds 255.
3. Specify the number of program memory pages per EEPROM bank in NUM_DATA_EE_PAGES. The minimum is two pages. A compile-time error occurs if fewer than two pages are specified.
4. Verify the values of ERASE, PROGRAM_ROW, and PROGRAM_WORD opcodes.
5. Specify the page erase size (in instructions) in NUMBER_OF_INSTRUCTIONS_IN_PAGE. Refer to the device data sheet to confirm the page erase size for the device.
6. Specify the maximum programming size (in instructions) in NUMBER_OF_INSTRUCTIONS_IN_ROW:
   - 128 for dsPIC33E/PIC24E devices.
   - 64 for other device families.
7. Select the maximum erase/write cycle count in ERASE_WRITE_CYCLE_MAX. The maximum value is 65,535. A compile-time error occurs if this limit is exceeded.
8. In all dsPIC33 and PIC24 devices containing the Flash ECC feature (check the specific device data sheet to find out if ECC is present), uncomment the following line in the include file:

```
DEE Emulation 16-bit.h
#define __HAS_ECC.
```

9. For the dsPIC33E/PIC24E devices, comment or uncomment the _AUXFLASH preprocessor definition for using the primary Flash program memory or auxiliary Flash program memory, respectively.
10. Add a function call to DataEEInit prior to any other operation to emulated data EEPROM.
11. Add the following files to your project:
    - DEE Emulation 16-bit.c
    - DEE Emulation 16-bit.h
    - Flash Operation.s

Notes:

1. Total addresses are DATA_EE_BANKS × DATA_EE_SIZE, ranging from 0 to (DATA_EE_BANKS × DATA_EE_SIZE) – 1.
2. In the PIC24/dsPIC33 device families, if the auxiliary Flash program memory option is selected by the user (that is the _AUXFLASH constant is defined), DATA_EE_BANKS × NUM_DATA_EE_PAGES must not exceed 7.
3. In all 16-bit devices with the Flash ECC feature, the amount of Flash memory consumed for the purpose of data EEPROM emulation is twice (double-word write devices) or Quadruple (quad-word write devices) that consumed in devices without ECC.

8. Software

The following table lists the tools and versions used to create both the PIC24/dsPIC33 and PIC18 solutions. Later versions of the tools will also work, but must be tested for compatibility with any application.

Table 8-1. Tools used for Solutions

Tool	Version
MPLAB® X IDE	3.65 or later
MPLAB XC16 C Compiler	1.30 or later
MPLAB XC8 C Compiler	1.42 or later
MPLAB XCDSC Compiler	3.10 or later

Table 8-2. Additional Tools For PIC24/dsPIC33

Tool	Version
MPLAB® Code Configurator (MCC) Plug-in	4.0.1 or later
MCC 16-Bit Data EEPROM Emulation Library	1.0.4 or later
MCC Data EEPROM Emulation Library	1.0.0 or later

You can find the latest source code and development tools on our website at www.microchip.com. For updated information about this application note, see the `Readme` file included with the software.

9. Conclusion

Emulated Data EEPROM provides an efficient solution for cost-sensitive applications requiring high-endurance, nonvolatile data storage. Applications designed for our cost-effective MCUs and DSCs can leverage unused program memory to enhance nonvolatile data endurance by over 500 times.

You can customize the effective endurance by adjusting the number of program memory pages, the size of the emulated Data EEPROM, and the erase/write limit. This flexible algorithm allows you to integrate high-endurance data storage into your applications efficiently.

10. Revision History

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

Revision	Date	Description
G	12/2024	This revision includes the following updates: • Updated the document as per the latest microchip edit standards. • Added MPLAB XCDSC Compiler in Table 8-1. • Updated “page status” in Definition of Terms. • Replaced PIC24/dsPIC33 with PIC24/dsPIC33E/C in Theory of Operation. • Updated the Page address in Table 1-9. • Added Sections Device with Double-Word Write Use Case and Device with Quad-Word write Use Case. • Replaced MCC 16-Bit DEE with MCC DEE. • Updated the Note in section Write Operation. • Updated the Effective Endurance. • Updated the Note in PIC24/dsPIC33 Emulation Checklist. • Added tools in Table 8-1 and Table 8-2.
F	12/2020	This revision includes the following updates: • All references to devices have been changed to PIC24/dsPIC33. • Updated the Theory of Operation section. • Added new section, MPLAB® Code Configurator Data EEPROM Emulation Support • Updated the Status Flags section. • Updated the Initialization Operation section. • Updated the Read Operation section. - Updated the Write Operation section. • Updated the Pack Operation section. • Updated the CPU Stall section. • Updated the PIC18 Emulation Checklist section. • Updated thePIC24/dsPIC33 Emulation Checklist section.
E	02/2018	This revision includes the following updates: • Updated Note in Introduction • Added Note 3 in Theory of Operation • Added Note in Write Operation • Added Note 2 in Pack Operation • Updated Note in Application • Added Note 3 in PIC24/dsPIC33 Emulation Checklist • Updated the second paragraph of CPU Stall • Updated the second paragraph of Application • Modified Step 5 and Step 11 (old Step 10), and added Step 8 in PIC24/dsPIC33 Emulation Checklist • Updated Table 9

.....continued

Revision	Date	Description
D	05/2011	This revision includes the following updates: • Added Note 2 in Theory of Operation • Updated the Note in Page Status • Added Note 2 in PIC24/dsPIC33 Emulation Checklist • Registers:- Updated the title in Register 1 • Updated the title “PIC24/dsPIC33F/ dsPIC33E Scenario” to PIC24/dsPIC33 Case Example • Updated the second paragraph in PIC24/dsPIC33 Case Example • Updated the title PIC24/dsPIC33 Emulation Checklist • Updated step 5, step 6 and step 10, and added step 8 in PIC24/dsPIC33 Emulation Checklist- Updated Conclusion • All references to PIC24/dsPIC33F, and PIC24 and dsPIC33F were updated to PIC24/dsPIC33F/ dsPIC33E throughout the document • All references to Flash memory were updated to Flash program memory • All references to Primary Flash memory were updated to Primary Flash program memory • All references to Auxiliary Flash memory were updated to Auxiliary Flash program memory • Minor changes to the text and formatting were incorporated throughout the document
C	10/2009	Added multi-bank support for PIC18.
B	04/2008	Added multi-bank support for PIC24/dsPIC33.
A	04/2007	This is the initial released version of this document.

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-0265-8

Legal Notice

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

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

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

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

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

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