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

Author: David Otten, Stephen Cowden,
Pradeep Budagutta, Priyabrata Sinha
and Harish Agari
Microchip Technology Inc.

INTRODUCTION

Microchip Technology Inc., has expanded its product portfolio to include a wide variety of cost-effective PIC® Microcontrollers (MCUs) without an internal data EEPROM.

Many applications store nonvolatile information in the Flash program memory using table read and write operations. Applications that need to frequently update these data may have greater endurance requirements than the specified Flash endurance for the MCU/Digital Signal Controller (DSC) devices.

The alternate solution of using an external, serial EEPROM device may not be appropriate for cost-sensitive or pin-constrained applications.

This application note presents a third alternative that addresses these issues. This algorithm features an interface similar to an internal data EEPROM, which uses available program memory and can improve endurance by a factor as high as 500.

Note: To use this solution, the device must have word write or double-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 use one location.

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 below 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 $N \times 255 - 1$, where N = the number of EEPROM banks.

PIC18 implementation supports 8-bit data and multiple EEPROM banks; PIC24/dsPIC33 implementation supports 16-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 and 16-bit implementations. For these formats, refer to [Table 1](#) and [Table 2](#).

TABLE 1: PIC18 DATA EEPROM INFORMATION FORMAT IN PROGRAM MEMORY

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

TABLE 2: PIC24/dsPIC33 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/dsPIC33, depending on whether an odd/even address is being written.

MPLAB® Code Configurator 16-Bit Data EEPROM Emulation Support

Microchip has added MPLAB Code Configurator 16-Bit Data EEPROM module (MCC 16-Bit DEE) support based on the PIC24/dsPIC firmware implementation of this application note. This helps ease the firmware configuration and setup. See the [“Software”](#) section on where to download.

A few notable differences between the MCC 16-Bit DEE version and the original application note's implementation:

File Names:

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

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

Function Names:

DEE_Init, DEE_Read, DEE_Write (equivalent to DataEEInit, DataEERead and DataEEWrite referenced in this document).

Note 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 16-Bit 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.

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 provided in [Table 3](#), 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 provided in [Table 4](#) through [Table 6](#).

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 7](#), 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. Since 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 provided in [Table 8](#). 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 3: 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	0xFF	0xFFFF
	•	
	•	
	•	
Page + 1022	0xFF	0xFFFF

TABLE 4: 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	0xFF	0xFFFF
	•	
	•	
	•	
Page + 1022	0xFF	0xFFFF

TABLE 5: 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	0xFF	0xFFFF
	•	
	•	
	•	
Page + 1022	0xFF	0xFFFF

TABLE 6: 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 7: 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
Page + 6	2	0x2222
Page + 8	0xA	0x0A0A
	•	
	•	
	•	
Page + 1022	7	0x7777

TABLE 8: PAGE AFTER PACK OPERATION

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

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 16-Bit DEE), is called before performing any other emulation routine. Since 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 should 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: 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`.

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 should 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 [Register 1](#), [Register 2](#) and [Register 3](#).

REGISTER 1: 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			

REGISTER 2: 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

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

bit 15-3 **Unimplemented:** Read as '1'

bit 2 **Page Expired**

1 = Page has not expired

0 = Page has expired

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

REGISTER 3: 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

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

bit 15-0 **Page Erase/Write Count:** Number of Page Erase/Write Cycles

INITIALIZATION OPERATION

The initialization routine, `DataEEInit` or `DEE_Init` (MCC 16-Bit 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:

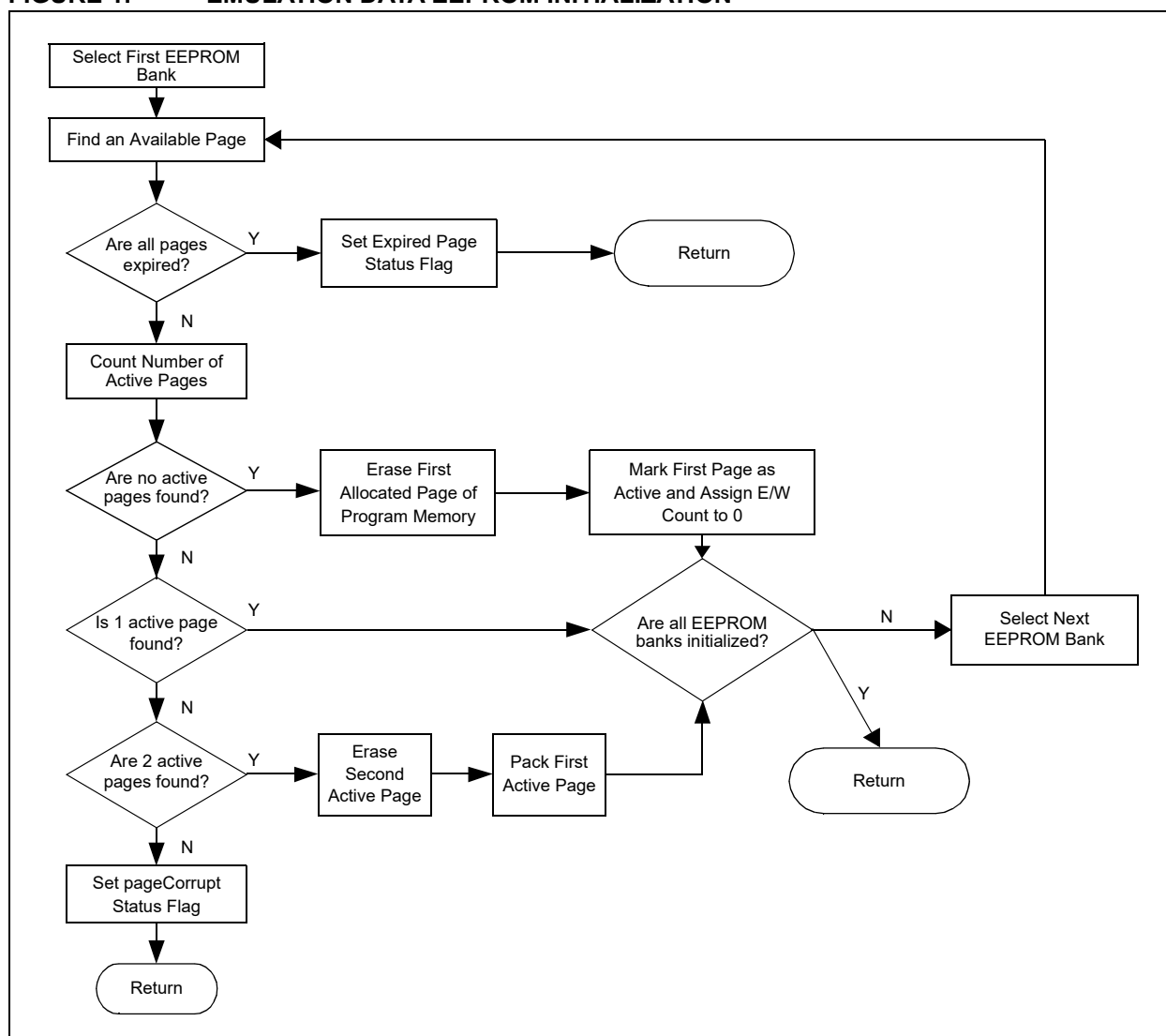
- If only one active page is found, the routine assumes a Reset occurred. No action is taken and the routine exits normally. Any read/write operation that may have been active during the Reset should be repeated.

- 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.

A flowchart of the initialization routine is illustrated in [Figure 1](#).

FIGURE 1: EMULATION DATA EEPROM INITIALIZATION



READ OPERATION

The `DataEERead` or `DEE_Read` (MCC 16-Bit DEE) function is used to 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

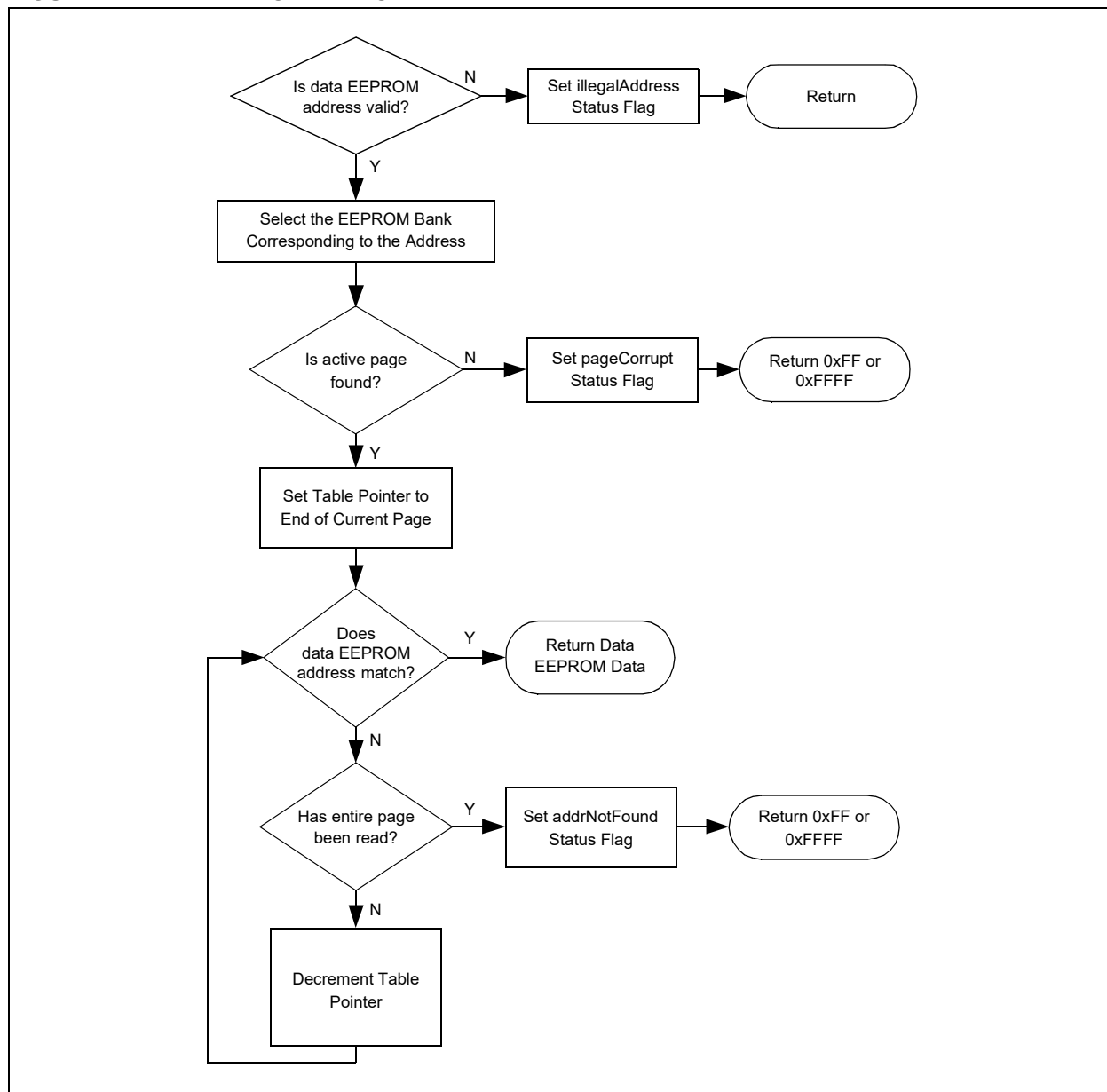
match, starting from the last location in the page. For details on how data EEPROM information is stored in program memory, refer to [Table 1](#) and [Table 2](#).

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.

A flowchart of the read operation is illustrated in [Figure 2](#).

FIGURE 2: READ OPERATION



WRITE OPERATION

To write emulated data EEPROM, the application uses the `DataEEWrite` or `DEE_Write` (MCC 16-Bit 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.

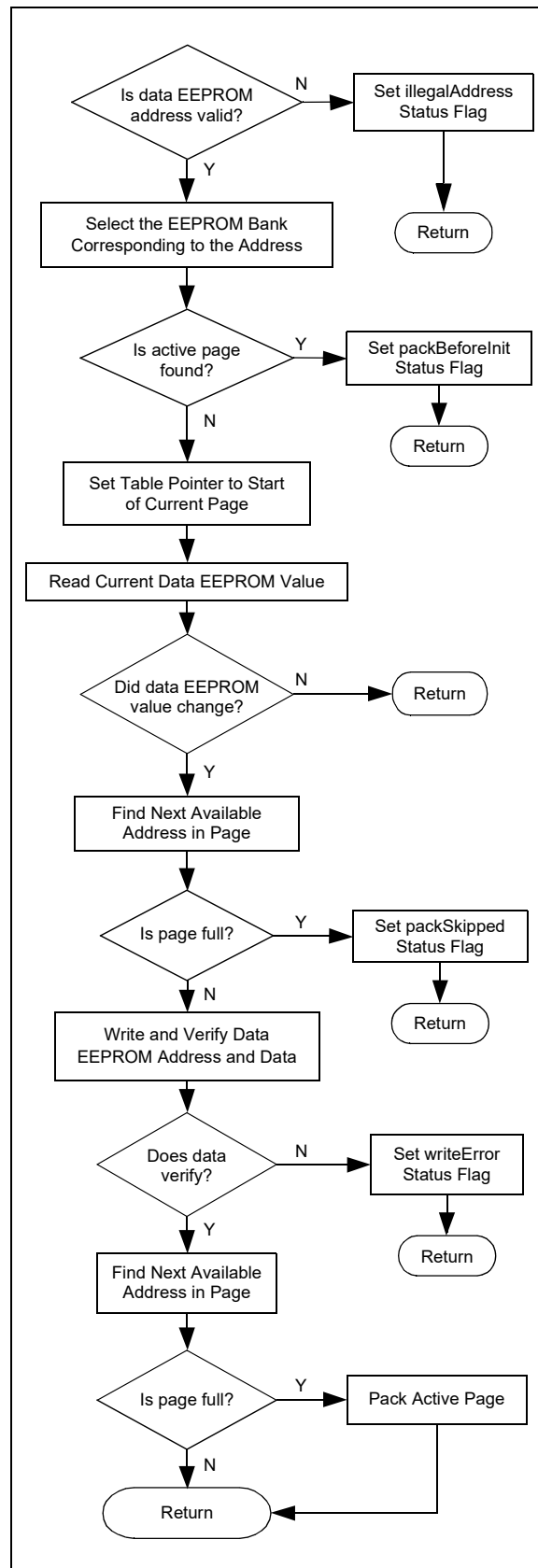
The algorithm is designed to maintain at least one available location in the active page for the next write operation. After a successful verification of the write operation, the `pack` routine is called if no available locations remain.

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

A flowchart of the write operation is illustrated in Figure 3.

Note: In all PIC24/dsPIC33 devices containing the 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 16-Bit DEE) function (but will be erased when the Flash page is erased).

FIGURE 3: WRITE OPERATION



PACK OPERATION

The pack routine, `PackEE` or `DEE_Pack` (MCC 16-Bit DEE, not a public function), is called from either a `DataEEWrite` or `DEE_Write` (MCC 16-Bit DEE) after the current page is filled, or by `DataEEInit` or `DEE_Init` (MCC 16-Bit 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.

A flowchart of the pack operation is illustrated in [Figure 4](#).

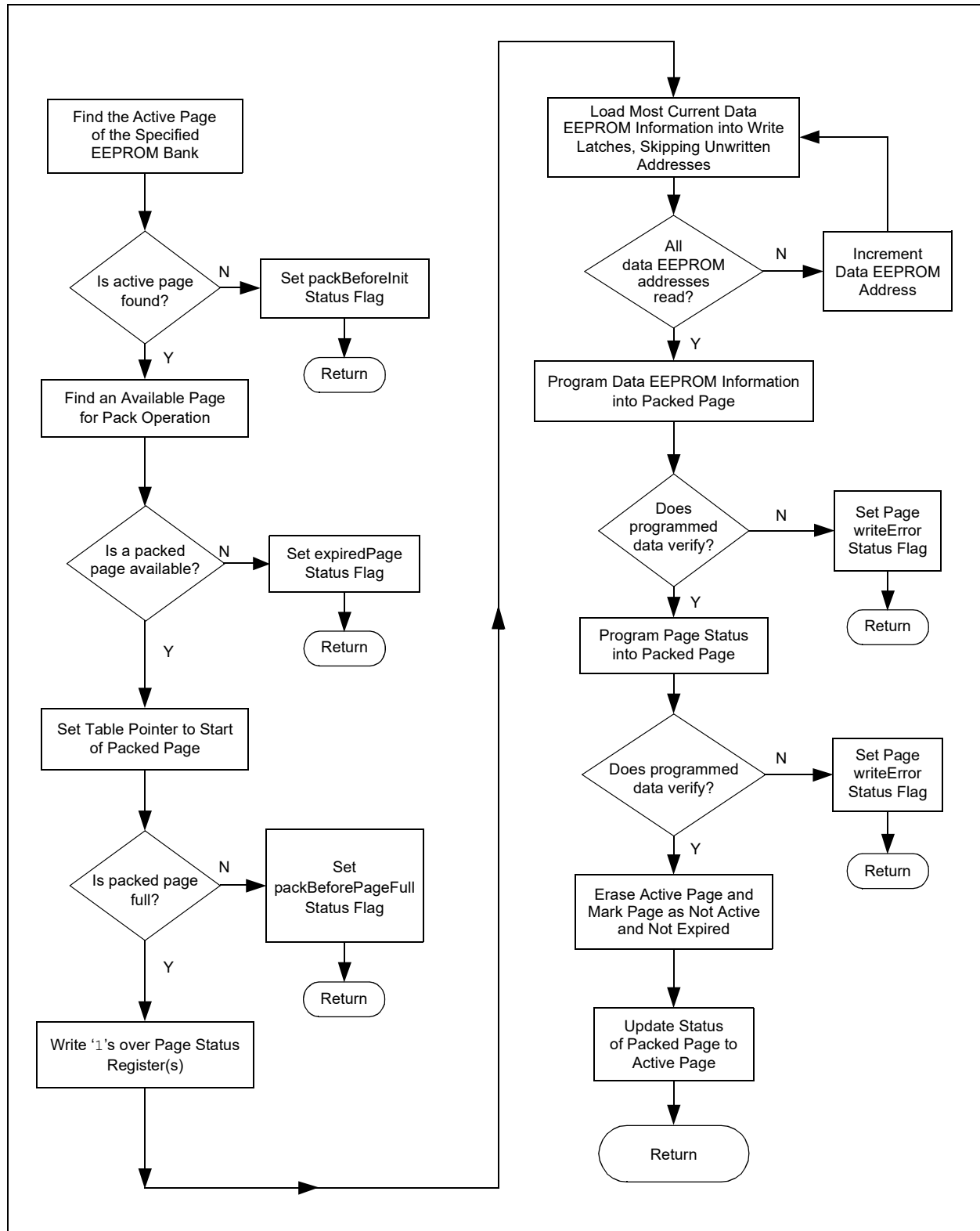
Note 1: The maximum data EEPROM size should be no greater than $N \times 255$, where N = number of EEPROM banks.

2: In PIC24/dsPIC33 devices, double-word write operations are used to program the packed page. In PIC24/dsPIC33 devices, row 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 should be used to allow at least half the page to be available after a worst-case DEE packed operation (i.e., 128).

FIGURE 4: PACK OPERATION



PERFORMANCE

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 either one location of program memory for the 16-bit algorithm or two for 8-bit. In addition, more locations will be 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, refer to ["Definition of Terms"](#).

Working through an example for the PIC24FJ128GA010, this device has a 512-word page. The 16-bit 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: EFFECTIVE ENDURANCE

$$\text{Total Effective Endurance} = (\text{Page Size} - \text{Page Status Size} - \text{Size of One Data EEPROM Bank}) \times \text{Number of Pages} \times \text{Endurance}$$

EQUATION 2: EFFECTIVE ENDURANCE CALCULATION EXAMPLE

$$\text{Total Effective Endurance} = (512 - 1 - 10) \times 2 \times 1000 = 1002000 \text{ Cycles}$$

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 16-Bit 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.

APPLICATION

To implement the data EEPROM emulation for an application, use any of the following checklists:

- [PIC18 Emulation Checklist](#)
- [PIC24/dsPIC33 Emulation Checklist](#)

Both the 8-bit and 16-bit algorithms require approximately 2.7 Kbytes of program memory. This does not include the amount of program memory reserved for data EEPROM information. They also require approximately 82 bytes of data memory.

Program memory for storing data EEPROM information is allocated using a two-dimensional array. The array size is dependent on the page erase size of the device and the number of pages reserved for emulation. For the 16-bit implementation, the compiler automatically aligns the array to the beginning of the next available page of program memory at compile time. For the 8-bit version, the user must specify the starting address for an available page. A compile-time error will generate if this address does not align with a page boundary. The array is used to determine program memory addresses for table operations. A compile-time error will be generated if the required amount of program memory is not available.

Note: For PIC18FXXJ and many 16-bit devices (refer to the specific device data sheet for details), 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 are not significantly affected by the size of the emulated data EEPROM.

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.

PIC18 Emulation Checklist

Perform the following changes to the `NoFilDee.inc` file. Refer to the specific device data sheet for information on program memory.

1. Specify emulation page start address in `EMULATION_PAGES_START_ADDRESS`.
2. Specify the number of EEPROM banks required in `DATA_EE_BANKS`. The maximum number is limited by the device 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 will generate if fewer than two pages are defined.
4. Specify the amount of data EEPROM needed in `DATA_EE_SIZE`. The maximum is 255 (0xFE). A compile-time error will generate if the data EEPROM size exceeds 255.
5. Verify the `ERASE`, `PROGRAM_ROW` and `PROGRAM_WORD` opcode values.
6. Specify the minimum page erase size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_PAGE` (512 typical).
7. Specify the maximum programming size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_ROW` (64 typical).
8. Select the maximum erase/write cycle count per location in `ERASE_WRITE_CYCLE_MAX`; the maximum is 65,535. A compile-time error will generate if the limit is exceeded.
9. Add a function call to `DataEEInit` prior to any other operation to emulated data EEPROM.
10. Add the following files to your project:
 - `DEE Emulation 8-bit.c`
 - `GenericTypeDefs.h`
 - `DEE Emulation 8-bit.h`
 - `NoFilDee.asm`
 - `NoFilDee.inc`
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 will generate if fewer than two pages are defined.
4. Verify the `ERASE`, `PROGRAM_ROW` and `PROGRAM_WORD` opcode values.
5. Specify the page erase size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_PAGE`. Refer to the specific device data sheet to verify the page erase size for the device.
6. Specify the maximum programming size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_ROW` (128 for the dsPIC33E/PIC24E devices and 64 for other device families).
7. Select the maximum erase/write cycle count in `ERASE_WRITE_CYCLE_MAX`. The maximum is 65,535. A compile-time error will generate if the 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`

PIC24/dsPIC33 Emulation Checklist

The following checklist is not required when using the MCC 16-Bit DEE version.

Perform the following changes to `DEE Emulation 16-bit.h`. Refer to the specific device data sheet for information on program memory.

1. Specify the number of EEPROM banks required in `DATA_EE_BANKS`; the maximum number is limited by the device memory.
2. Specify the amount of data EEPROM needed for each bank in `DATA_EE_SIZE`. The maximum is 255 (0xFE). A compile-time error will generate if the data EEPROM size exceeds 255.

Note 1: Total addresses are `DATA_EE_BANKS x DATA_EE_SIZE`, ranging from 0 to $(DATA_EE_BANKS \times DATA_EE_SIZE) - 1$.

2: In the PIC24/dsPIC33 device families, if the auxiliary Flash program memory option is selected by the user (i.e., the `_AUXFLASH` constant is defined), `DATA_EE_BANKS x 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 that consumed in devices without ECC.

SOFTWARE

The tools and versions used to create both the PIC24/dsPIC33 and PIC18 solutions are listed in Table 9. Later versions of the tools will also work, but should be tested for compatibility with any application.

TABLE 9: TOOLS USED FOR SOLUTIONS

Tool	Version
MPLAB® X IDE: www.microchip.com/mplabx	3.65 or later
MPLAB XC16 C Compiler: www.microchip.com/xc16	1.30 or later
MPLAB XC8 C Compiler: www.microchip.com/xc8	1.42 or later

TABLE 10: ADDITIONAL TOOLS FOR PIC24/dsPIC33

Tool	Version
MPLAB® Code Configurator (MCC) Plug-in: www.microchip.com/mcc	4.0.1 or later
MCC 16-Bit Data EEPROM Emulation Library: https://www.microchip.com/SWLibraryWeb/product.aspx?product=16-Bit_Data_EEPROM_Emulation	1.0.4 or later

The latest source code and development tools are available on Microchip Technology's website (www.microchip.com). For the latest information on this application note, read the associated Readme file included with the software.

CONCLUSION

Emulated data EEPROM is an effective solution for cost-sensitive applications that require high-endurance, nonvolatile data memory. Applications suited for Microchip Technology's cost-effective MCU and DSC devices can employ unused program memory and increase nonvolatile data endurance by a factor in excess of 500. This "effective endurance" can be customized by selecting the number of program memory pages, size of emulated data EEPROM and the erase/write limit. This flexible algorithm will enable you to add high-endurance data EEPROM to your applications.

REVISION HISTORY

Revision A (April 2007)

This is the initial released version of this document.

Revision B (April 2008)

Added multi-bank support for PIC24/dsPIC33.

Revision C (October 2009)

Added multi-bank support for PIC18.

Revision D (May 2011)

This revision includes the following updates:

- Notes:
 - 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](#)
- Sections:
 - 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](#)”
- Tables:
 - Updated [Table 9](#)
- 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

Revision E (February 2018)

This revision includes the following updates:

- Notes:
 - 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](#)”
- Sections:
 - 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](#)”
- Tables
 - Updated [Table 9](#)

Revision F (December 2020)

This revision includes the following updates:

- All references to devices have been changed to PIC24/dsPIC33.
- Sections:
 - Updated the “[Theory of Operation](#)” section.
 - Added new section, “[MPLAB® Code Configurator 16-Bit 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 the “[PIC24/dsPIC33 Emulation Checklist](#)” section.
- Notes:
 - Updated Note 3 and added Note 4 to the “[Theory of Operation](#)” section.
 - Added Note 3 in the “[Pack Operation](#)” section.
- Tables:
 - Updated [Table 9](#).
 - Added [Table 10](#).

NOTES:

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

- 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 and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- 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. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like 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.

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

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

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

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

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

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2007-2020, Microchip Technology Incorporated, All Rights Reserved.

ISBN: 978-1-5224-7355-8

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

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://www.microchip.com/support>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880-3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820