

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

*Author: David Otten and
Stephen Cowden
Microchip Technology Inc.*

INTRODUCTION

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

Many applications store nonvolatile information in the Flash program memory using table write and read operations. Applications that need to frequently update this data may have greater endurance requirements than the specified Flash endurance for the device.

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, 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 capability. Refer to the specific device data sheet to verify this feature is available.

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 length of time and associated conditions for the retention of data in Flash 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 location(s) at the beginning of the current page that stores data EEPROM emulation status. The PIC18 implementation uses two locations and the PIC24/dsPIC33F uses one.

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.

For example, if the implemented size of the data EEPROM is 5, and two data EEPROMs are used, only the addresses in the range, 0 to 9, are available.

- Note 1:** PIC18 implementation supports only one EEPROM bank.
- 2:** PIC24/dsPIC33F implementation supports multiple EEPROM banks.
- 3:** Each EEPROM can have a maximum of 255 addresses. Hence, the total addresses are from 0 to $N \times 255 - 1$, where N = number of EEPROM banks.

PIC18 implementation supports 8-bit data and only one EEPROM bank; PIC24/dsPIC33F 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, see 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/dsPIC33F 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 microcontroller's ability to self-program a single location of program memory. This location is an 8-bit operation for PIC18 and either an 8 or 16-bit operation for PIC24 and dsPIC33F, depending on whether an odd or even address is being written, respectively.

| |
|---|
| Note: For more information on program memory organization, refer to the applicable product data sheet. |
|---|

PIC24/dsPIC33F Scenario

To better understand how the algorithm works, here is a simple scenario for the PIC24 and dsPIC33F.

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 available pages and update status flags. This page is designated as the current or 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 in a similar fashion, as provided in Table 4 through Table 6.

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

In Table 7, the last location in the page is written with a rewrite to address 0x7 to 0x7777. Since the currently active page is now full, the data EEPROM information will move to the next available page. 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 is needed, the amount of information decreases.

After the data is 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 displayed in Table 8. The page is now ready to store more information via 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 is 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 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 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 |

Since each location within the page is programmed once prior to the page erase, only one erase/write cycle is consumed for the page. 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.

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 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 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 |

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

Status flags differ in severity and how they are serviced. 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`, is called prior to 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 completed successfully.

Macros use this 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 PIC24/dsPIC33F. Status contains information about the page, whether it is expired or active, and the number of erase/write cycles performed.

These values are used by the algorithm to monitor and control page information and are not directly accessible by the user. Refer to Register 1, Register 2 and Register 3 for formats of PIC24/dsPIC33F and PIC18 page status information.

Note: Applications with bootloaders should be careful to not change the page status information. This can be done by programming '1's into these locations.

REGISTER 1: PAGE STATUS FOR PIC24 AND dsPIC33F 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 | | | |

AN1095

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'

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 not expired

0 = Page expired

bit 1 **Page Current**

1 = Page not current

0 = Page current

bit 0 **Page Available**

1 = Page available

0 = Page 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'

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`, must be called before any other data EEPROM operation can occur; this initializes all 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:

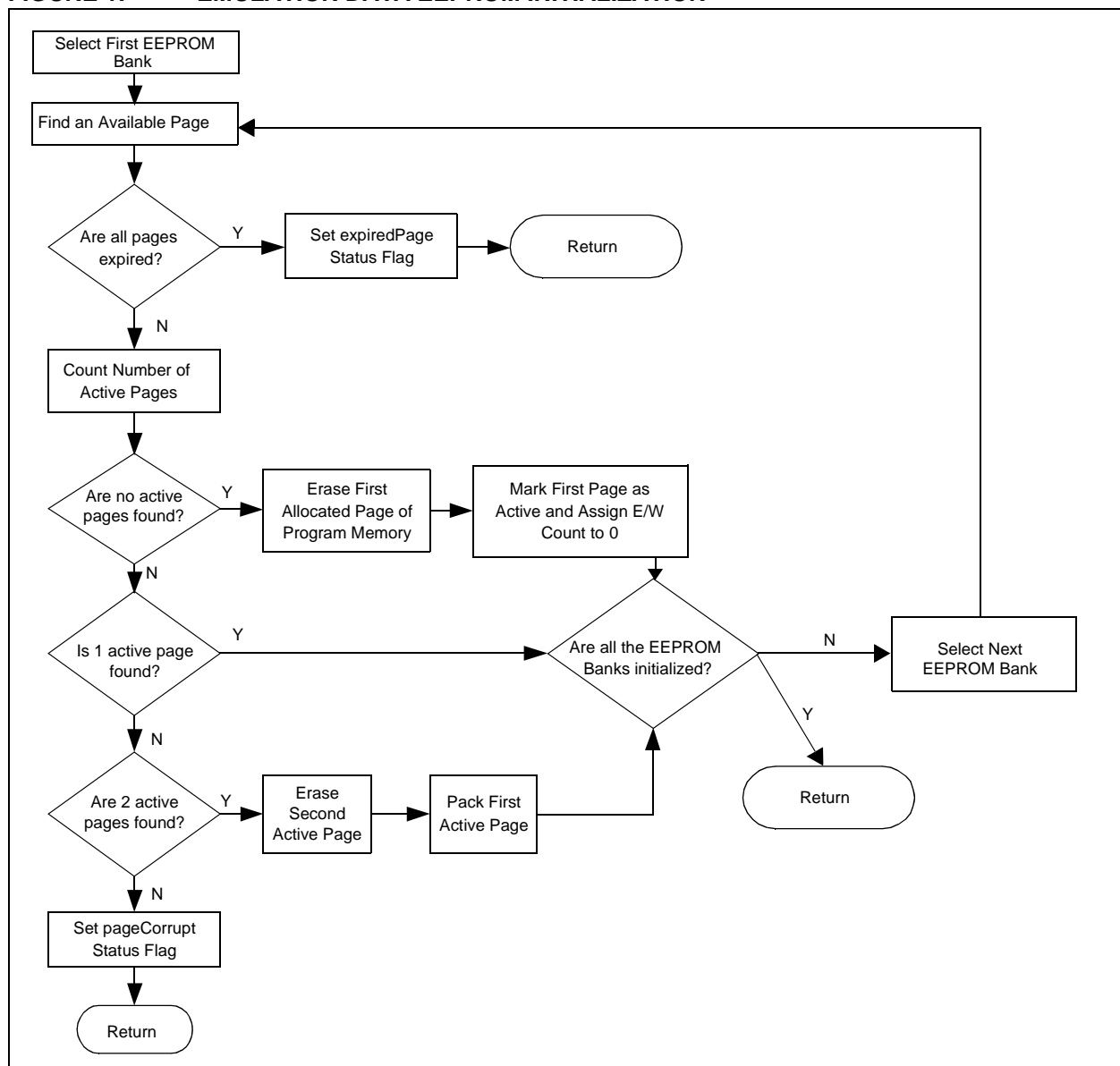
- If only one active page is found, the routine assumes a Reset occurred. No action is taken and the routine exits normally. Any read or write operation that may have been active during the Reset should be repeated.

- 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/dsPIC33F) or `EECON1` (PIC18) registers. By doing so, the application can respond appropriately to Resets and supply voltage changes.

For a flowchart of the initialization routine, see Figure 1.

FIGURE 1: EMULATION DATA EEPROM INITIALIZATION



READ OPERATION

The `DataEERead` 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.

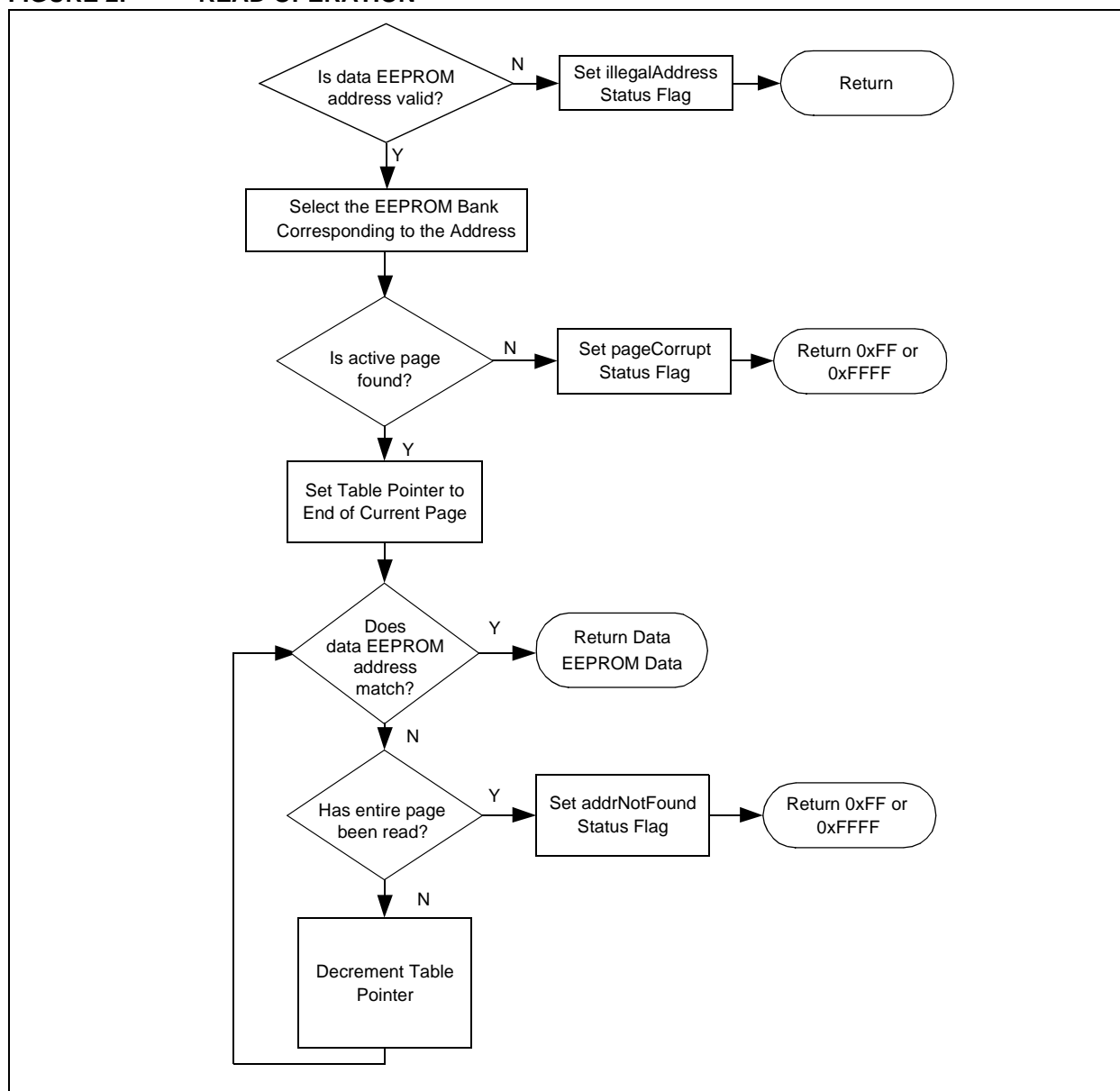
The routine then searches for the active page. Once located, the active page is searched for an address match, starting from the last location in the page. For details on how data EEPROM information is stored in program memory, see Table 1 and Table 2.

Since 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.

For a flowchart of the read operation, see Figure 2.

FIGURE 2: READ OPERATION



WRITE OPERATION

To write emulated data EEPROM, the application uses the `DataEEWrite` 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 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 nonzero 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 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.

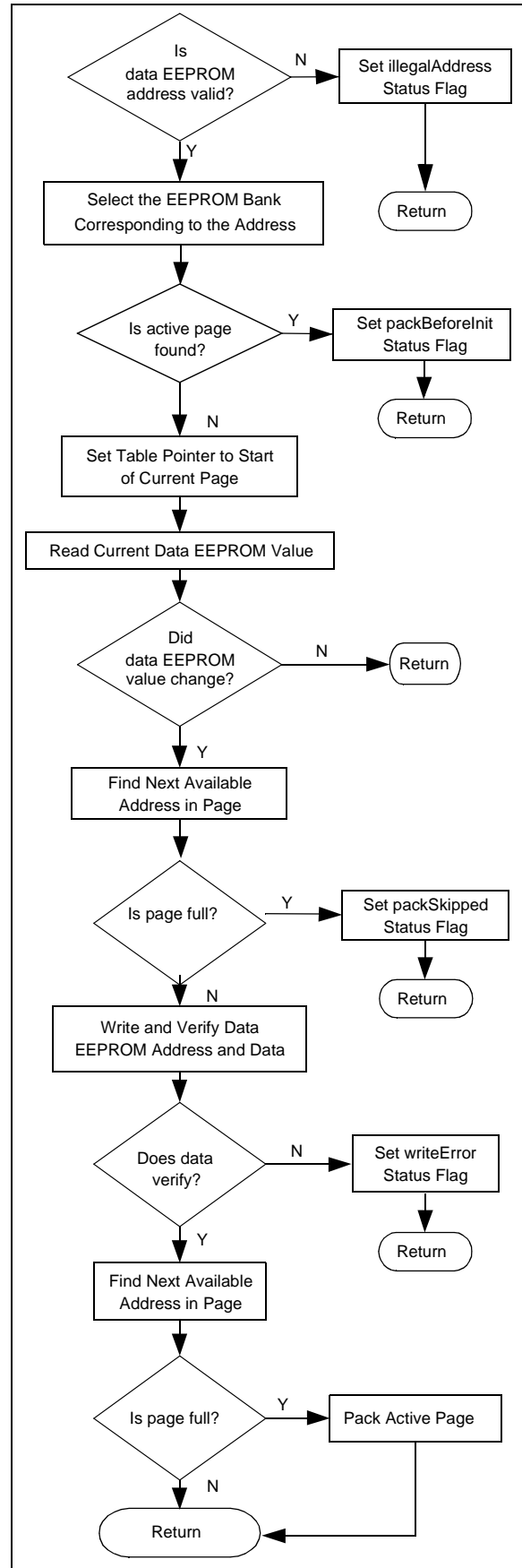
If the information does not verify, the `writeError` flag is set and a nonzero value is returned. The user can attempt to rewrite the data or respond as needed.

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.

For a flowchart of the write operation, see Figure 3.

FIGURE 3: WRITE OPERATION



PACK OPERATION

The pack routine, `PackEE`, is called from either a `DataEEWrite`, after the current page is filled, or by `DataEEInit`, to initialize program memory for data EEPROM emulation. It can also be called by the user, which may benefit timing-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 since 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.

Since 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 is 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.

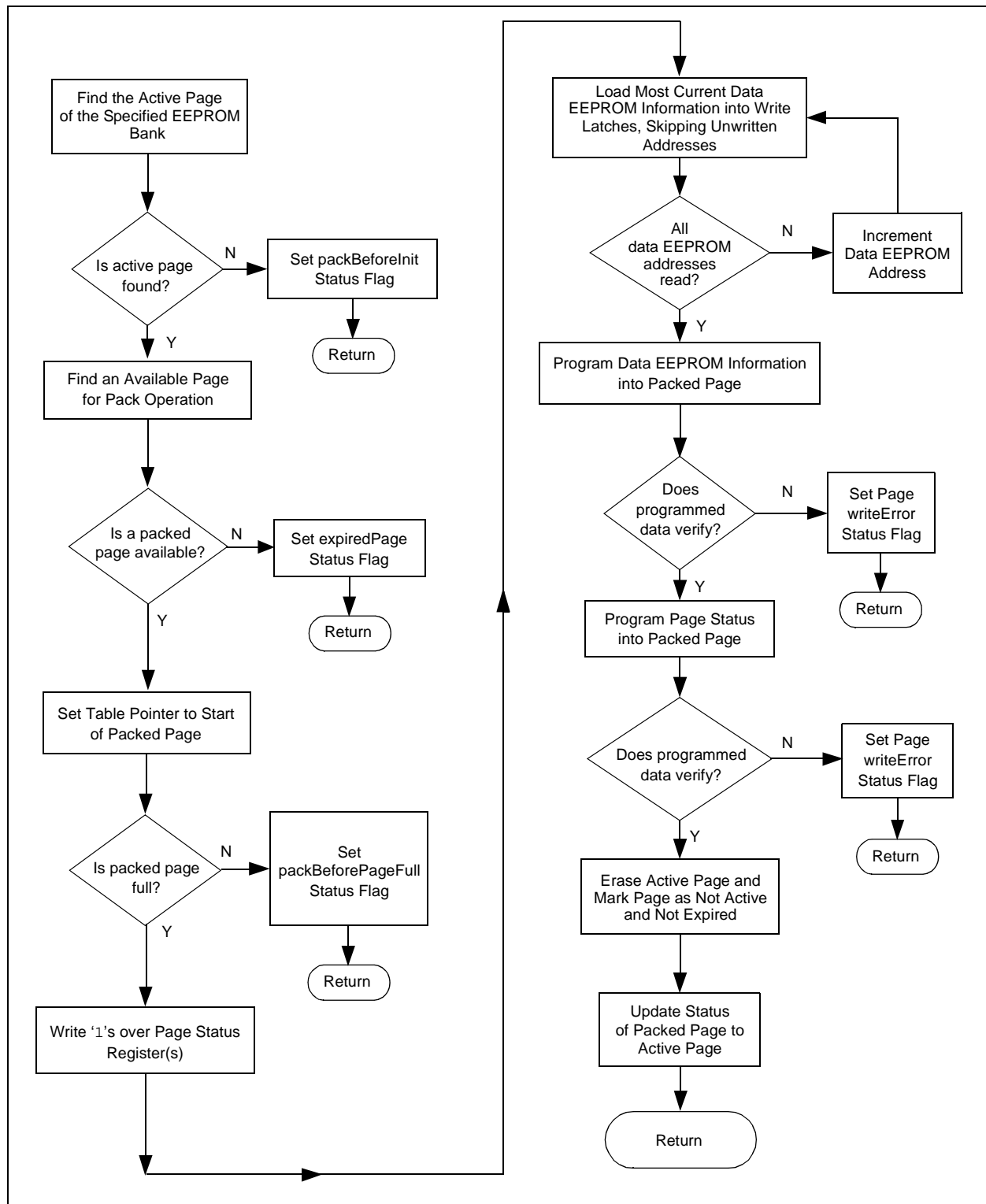
After all of the data has been programmed into the packed page, the current page data is read and compared to the packed page data. If a mismatch occurs, the `writeError` flag is set and the function exits with an error code. The page status information is also programmed and verified. If the verify 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.

For a flowchart of the pack operation, see Figure 4.

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

FIGURE 4: PACK OPERATION



PERFORMANCE

Effective Endurance

Determining effective endurance is not a trivial calculation since it is dependent on many factors. Traditionally, endurance is defined as how many times a single address can be safely written. This definition doesn't 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. 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 since 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. Refer to the “**Definition of Terms**” section for more information on the terms.

EQUATION 1:

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

EQUATION 2:

$$\text{Total Effective Endurance} = (512 - 1 - 10) \times 2 \times 1000 = 1,002,000 \text{ Cycles}$$

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 calculation for two pages of program memory, 10 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.

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` 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 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 2 and Page Size times two. 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 applicable product data sheet.

APPLICATION

To implement the data EEPROM emulation for an application, use the appropriate checklist given in the sections, “**PIC18 Emulation Checklist**” or “**PIC24/dsPIC33F Emulation Checklist**”.

Both the 8 and 16-bit algorithms require approximately 2.7 Kbytes of program memory for software. This total does not include the amount of program memory reserved for data EEPROM information. They also require approximately 82 bytes of data memory. The MPLAB® C30 C compiler used optimization level “s” to minimize code size, although other settings can be used.

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 PIC24F devices, the last page of program memory stores the Configuration Word information. It can not be allocated for data EEPROM emulation.

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

Make the following changes to the `NoFilDee.inc` file. Refer to the product data sheet for information on program memory.

1. Specify emulation page start address in `EMULATION_PAGES_START_ADDRESS`.
2. Specify the number of program memory pages in `NUM_DATA_EE_PAGES`.
The minimum is two pages. A compile-time error will generate if fewer than two pages are defined.
3. 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.
4. Verify the `ERASE`, `PROGRAM_ROW` and `PROGRAM_WORD` opcode values.
5. Specify the minimum page erase size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_PAGE` (512 typical).
6. Specify the maximum programming size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_ROW` (64 typical).
7. 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.
8. Add a function call to `DataEEInit` prior to any other operation to emulated data EEPROM.
9. Add the following files to your project:
 - `DEE Emulation 8-bit.c`
 - `GenericTypeDefs.h`
 - `DEE Emulation 8-bit.h`
 - `NoFilDee.asm`
 - `NoFilDee.inc`

PIC24/dsPIC33F Emulation Checklist

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

1. Specify the number of program memory pages in `NUM_DATA_EE_PAGES`.
The minimum is two pages. A compile-time error will generate if fewer than two pages are defined.
2. Specify the number of EEPROM banks required in `DATA_EE_BANKS`. The maximum number is limited by the memory the device has.
3. 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.
4. Verify the `ERASE`, `PROGRAM_ROW` and `PROGRAM_WORD` opcode values.
5. Specify the minimum page erase size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_PAGE` (512 typical).
6. Specify the maximum programming size (in instructions) in `NUMBER_OF_INSTRUCTIONS_IN_ROW` (64 typical).
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. Add a function call to `DataEEInit` prior to any other operation to emulated data EEPROM.
9. Add the following files to your project:
 - `DEE Emulation 16-bit.c`

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

SOFTWARE

The tools and versions used to create both the PIC24/dsPIC33F 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® IDE | 8.10 |
| MPLAB C30 C Compiler | 2.01 |
| MPLAB C18 C Compiler | 3.02 |

The latest source code and development tools are available on Microchip Technology's web site (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 0.25 µm PIC 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.

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "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 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. 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 ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. 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.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rLAB, Select Mode, Total Endurance, UNI/O, WiperLock 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.

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

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



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://support.microchip.com>
Web Address:
www.microchip.com

Atlanta

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

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

Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara

Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto

Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office

Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney

Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu

Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR

Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing

Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao

Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai

Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang

Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen

Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan

Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen

Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian

Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai

Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore

Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi

Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune

Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama

Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu

Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul

Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang

Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila

Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore

Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu

Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung

Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei

Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok

Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels

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

Denmark - Copenhagen

Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris

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

Germany - Munich

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

Italy - Milan

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

Netherlands - Drunen

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

Spain - Madrid

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

UK - Wokingham

Tel: 44-118-921-5869
Fax: 44-118-921-5820