Application Note

EEPROM Emulation with a Z8 Encore! XP® Flash MCU

AN014005-0508

Z8 Encore! XP® Flash Microcontrollers

Abstract

This application note provides a method for utilizing a segment of Zilog's Z8 Encore! XP Microcontroller's (MCU's) Flash memory to emulate the functionality of an EEPROM device. The data written to Flash remains protected from power loss and inadvertent writes, as special instruction sequences are required to write or erase Flash memory.

Note: *The source code files associated with this application note,* AN0140-SC01, AN0140-SC02, *and* AN0140-SC03 *are available for download at* www.zilog.com.

Z8 Encore! XP Flash Microcontrollers

Zilog's Z8 Encore! XP products are based on the new eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows faster development time and program changes in the field. The high-performance register-to-register based eZ8 CPU core architecture maintains backward compatibility with Zilog's popular Z8® MCU.

The new Z8 Encore! XP microcontrollers (MCUs) combine a 20 MHz core with Flash memory, a linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP MCU suitable for various applications like motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

This section discusses internal Flash memory in Z8 Encore! XP MCUs.

Z8 Encore! XP Internal Flash Memory

The on-chip program Flash memory in Z8 Encore! XP MCU utilizes the latest memory technology featuring non-volatile, linearly-addressable, Flash memory with in-circuit write/erase capability. A Flash controller block handles programming and erase operations by controlling the timing of voltage applied to the Flash memory cells. During programming and erase operations, the core voltage of 3.3 V is sourced internally and no external power supply is required.

Contents of Flash memory are divided into pages which are blocks of consecutive Flash addresses. For example, the Z8F6403 MCU contains 128 pages with 512 bytes on each page. A Flash byte location comprises eight Flash cells with each cell representing bits. For a Flash cell, a logical 1 represents the erased state and a logical 0 represents the programming state. Thus, a byte location in the erased state reads as 0xFF and can be programmed to any other value by flipping its 1s to 0s.

As a consequence, a 0x0F (binary 1111) can be overwritten with a 0x0E (binary 1110), which can be overwritten with a 0x0C (binary 1100), which in turn can be overwritten with a 0x08 (binary 1000). Finally, this value can be overwritten by a 0x00. All these overwrites occur in the same Flash location without a page erasure. Writing the value 0xFF to an erased location does not affect any cell.

However, changing the state of a cell from 0 to 1 can be accomplished only by erasing an entire page. A *page erase* occurs when all the bytes in a page are erased at the same time. A *mass erase* refers to the erasure of all the pages at the same time. The CPU remains idle after issuing an Erase command to the Flash controller and resumes execution only when the Erase action is complete.

An *erase cycle* is an instance of erasing a page or a number of pages. Each flash of memory can perform a specified number of erase cycles and this minimum number is termed as *endurance*.

Flash memory is read and written on a byte-by-byte basis. However, after a Flash location is written, it cannot be rewritten without first erasing the page. To write a byte in a Flash location, you must first unlock the Flash controller by providing specific instructions in a sequence. For more information on these instructions, refer to the *Z8 Encore! XP F64XX Series Flash Microcontrollers Product Specification (PS0199)*.

The primary purpose of the Z8 Encore! XP MCU's on-chip Flash is to store software programming instructions. After this code is programmed, some portion of the Flash is not utilized and remains free. This unused portion can be used as a *virtual EEPROM*. An EEPROM emulation technique is discussed in the following section.

EEPROM Emulation Technique

Generally, an EEPROM chip stores and retains data in the event of a power failure. The data is retrieved whenever required and updated in the same location (up to the maximum device rating).

A virtual software version of an EEPROM can be used instead of a traditional EEPROM chip. It can emulate generic EEPROM functionality in Z8 Encore! XP on-chip Flash memory. This emulation is performed in various ways by considering Flash limitations and product requirements.

At a minimum, all EEPROM emulation implementations require data and address pairs to be stored in Flash locations for subsequent retrieval or update. When data is modified, the modified data associated with the earlier virtual address is stored in a new Flash location. During data retrieval, the modified data, in the latest Flash location is returned. If a new Flash location (blank location) on the appropriate page is not available, that page is erased after its valid contents are copied to a new page. The modified data is then written to the new page.

A virtual EEPROM is useful in products requiring frequent updates to data items, in the field, or during run time. For example, a remotely located weather data logger that needs to record the ambient temperature value at regular intervals (For example, every hour) can utilize a virtual EEPROM to store the latest time and temperature value in Z8 Encore! XP on-chip Flash. The only working limitation is that the Flash contains a finite number of erase cycles and a finite amount of memory.

As writing data into Flash is implemented in an emulated EEPROM, data is prone to corruption or ambiguity due to power failure, fluctuations in voltage supply, or loss of voltage. Consider these conditions carefully while designing the software.

Developing EEPROM Emulation with a Z8 Encore! XP Flash MCU

The EEPROM emulation implemented in this application note is based on the following specifications:

- The virtual EEPROM is accessible as a linearly-addressable range. For example, a 1 KB size is addressable from `0x000` to `0x3FF` (This is not the actual flash address) (see Figure 1 on page 3).
- The maximum recommended size for the emulated EEPROM is 4 KB, which requires 32.5 KB Flash space (4 KB÷128 variables per page = 32 KB + 1 transfer page of 0.5 KB size).

- The size of data to be stored is one byte long (ranging from 0x00 to 0xFF).
- A minimum of two Flash pages are required for storing variables as tag-data pairs—one to store tag-data pairs, and the other to be used as a transfer page when the first page is erased. The erased page is then used as a transfer page.
- The smallest EEPROM that can be emulated is 128 bytes and requires 1 KB of Flash. When one Flash page is filled, only the valid entries are transferred to the transfer page when the first page is erased.
- Each additional 128 bytes of EEPROM emulation requires additional 512 bytes of Flash. Therefore to store 1024 variables (virtual EEPROM size = 1 KB), 9 pages are required (1024 (variables) ÷ 128 (bytes) + 1 transfer page).

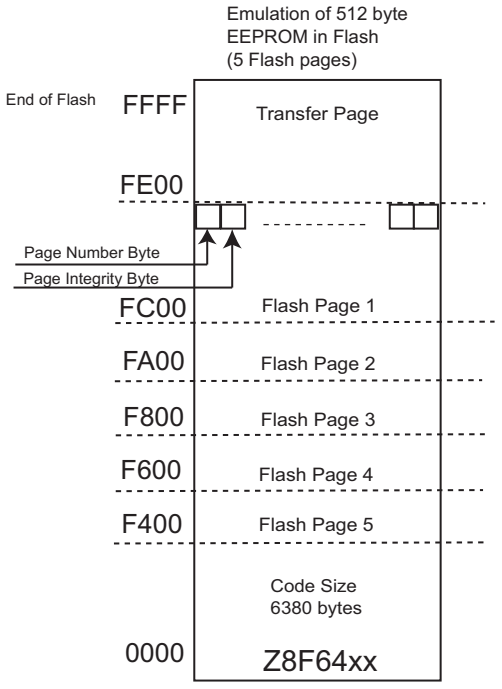

Figure 1. Flash Memory Map of Z8F64xx Devices to Store Variables in EEPROM

Software Implementation

The software implementation of the EEPROM emulation using Z8 Encore! XP® Flash includes formulating the variable structure and developing the APIs to initialize, read, and write to the EEPROM.

Variable Structure

The variables are stored in a Flash page, starting from its first physical address. A tag byte and its associated data byte combines to create a *variable*. Thus, every variable requires a storage space of two bytes.

The tag byte (Figure 2 on page 4) is configured as follows:

- The most significant bit (msb) is used as a dedicated bit to indicate whether the associated data byte is the *latest* (msb = 1) or *old* (msb = 0).
- The remaining seven bits are used to store a 7 bit virtual address value (ranging from 0x01 to 0x7F for 127 address values; the number 0 is not used).

The virtual address value on a page is calculated as follows:

```
Virtual address value = (virtual EEPROM address) - [0x80 * (page number - 1)]
```

where,

- virtual EEPROM address is a user-specified address
- 0x80 is the hex value for 128 bytes which can be stored on a single page
- page number is the virtual page number

The virtual page number is calculated as follows:

```
Virtual page number = 1 + (virtual EEPROM address/0x80)
```

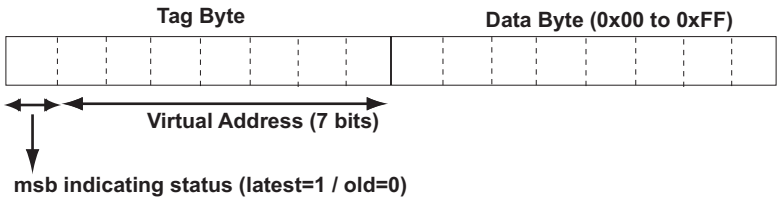

Figure 2. Structure of a Variable

Note: *The page number is always an integer.*

As an example, for a virtual EEPROM address of 0x240:

```
virtual page number = (0x240/
0x80) + 1 = 0x05

virtual address value = 0x240 –
[0x80 * 0x04] = 0x40
```

A Flash page of size 512 bytes on Z8 Encore! XP® MCU can store 128 independent variables. The first 128 variables (0x00 to 0x7F) are stored in virtual page number 1; the next 128 virtual addresses (0x80 to 0xFF) are stored in virtual page number 2; a page with virtual number 0xFF refers to the transfer page. There is no virtual page number 0. The virtual page number is stored at the last physical location of the Flash page.

Virtual Page Number Byte and Page Integrity Byte

The final two physical addresses in each Flash page are reserved as the virtual page number byte and the page integrity byte.

The msb of the virtual page number byte indicates the validity of the page. When this bit is set to 1, it indicates a valid page. This bit is flipped to 0 prior to initiating a page erase. If the page erase operation is unsuccessful (due to power loss), then this bit indicates that the data on that page is invalid.

A page integrity byte indicates the validity of data on a page. The page integrity byte values and interpretations are listed below:

- 0xFF indicates that the page is blank (no data).
- 0xCC indicates that data on this page is invalid (incomplete transfer).
- 0x00 indicates that data on page is valid.

Note: *For newly erased Flash memory, the final page of the device is maintained as a transfer page, the second last page acts as the virtual page number 1, and so on. This is up to the allotted number of pages. This sequence changes as and when the data is modified and stored due to page erasures.*

On powerup, the Z_initialize_EEPROM() API validates the stored data and recovers from any previous power failure and resultant corruption of data.

Protection Against Power Loss

The software overhead associated with writing a byte in Flash makes the data prone to corruption or ambiguity (for example, if the address information is corrupted) due to a power failure. Data corruption can also occur due to fluctuations in voltage supply or a total loss of voltage when writing a byte to Flash. Consider these errors due to these cases while designing the software.

In the software accompanying this application note, the `Z_Initialize_EEPROM()` routine in the `Z_API.c` file contains code to eliminate errors due to power loss.

A check is conducted at the time of initialization, after a Power-on Reset (POR), using the `Z_Initialize_EEPROM()` API. If more than one location containing the same valid EEPROM address is found, then the latest address byte is invalidated by flipping the msb to 0. Performing this task preserves the last data byte associated with the address and ensures that the old data is retained.The latest address byte is invalidated as the data byte associated with it is suspected to be corrupted due to various reasons.

EEPROM Emulation APIs

The following three APIs are provided to affect EEPROM emulation of a Z8 Encore! XP Flash location:

1. `Z_Initialize_EEPROM`
2. `Z_Read_EEPROM`
3. `Z_Write_EEPROM`

Initialization API

Z_Initialize_EEPROM

The `Z_Initialize_EEPROM()` API sets up and initializes the emulated EEPROM on powerup. It returns an error on failure. This API must be called on microcontroller powerup, prior to other EEPROM emulation APIs. When this API is called subsequently, it functions as a clean-up routine checking the data integrity of the emulated EEPROM area.

Parameter(s)

None

Return(s)

`0x00`	On successful execution of the API
`0xFF`	On failure due to inaccessible Flash

Usage

```
value = Z_Initialize_EEPROM();
```

Figure 4 on page 9 displays the steps to initialize the virtual EEPROM. Initialization erases and formats the emulated EEPROM area.

Read API

Z_Read_EEPROM

The `Z_Read_EEPROM()` API reads the data byte corresponding to the virtual EEPROM address. The msb indicates success (`0x00`) or failure (`0xFF`) and the lsb contains the read value. If the virtual address is not found, this API returns a `0xFFFF`.

Parameter(s)

`unsigned int` virtual address from where the data is to be read.

Return(s)

`0x00XX`	On successful execution of the API (`XX` represents the data read)
`0xFFFF`	On API failure

Usage

```
value = Z_Read_EEPROM(0x10);
```

The process of reading a data byte from a virtual EEPROM address is explained in the flowchart (See Figure 5 on page 10).

Write API

Z_Write_EEPROM

The `Z_Write_EEPROM()` API writes the data byte to the virtual EEPROM address and returns success `(0x00)` or error `(0xFF)`.

Parameter(s)

`int`	Virtual EEPROM address
`unsigned char`	Data

Return(s)

`0x00`	On successful execution of the API
`0xFF`	On API failure

Usage

```
value = Z_Write_EEPROM(0x10,0x50);
```

Figure 6 on page 11 displays the steps involved to write a data byte to a virtual EEPROM address.

Note: *If the new data to be written to a virtual address is the same as the existing data stored in Flash, the program skips the writing function to preserve storage space.*

Testing EEPROM Emulation APIs

The EEPROM emulation APIs are exposed through an interactive application that drives an RS-232 terminal. The testing of the APIs is performed using the HyperTerminal application on a standard PC. Raw Flash memory reads and writes are performed using additional functions as provided in the project.

Equipment Used

The following equipment and software are required for testing:

- Z8 Encore! XP Development Kit (Z8F64200100KIT) with a Z8F642x MCU
- ZDS II Encore! IDE v4.10.0
- Windows-based PC with HyperTerminal application

Test Setup

Connect the Z8 Encore! XP-based MCU (Z8F642x) to the PC (with the HyperTerminal application) as displayed in Figure 3.

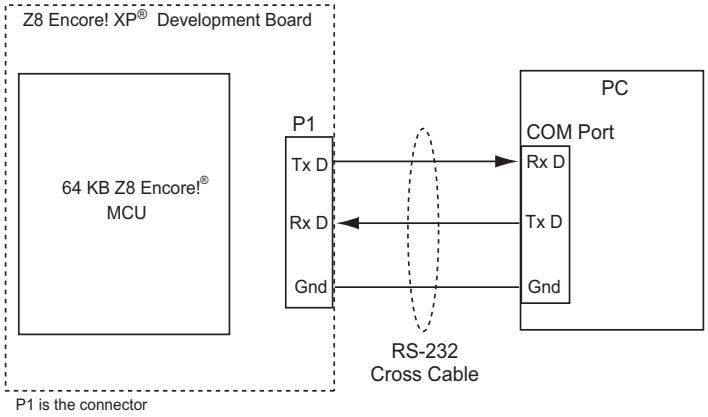

Figure 3. Testing EEPROM Emulation with Z8F642x MCU

System Configurations

The HyperTerminal settings are as follows:

- Serial Port: COM1 or COM2
- Baud Rate: 9600 bps
- Parity: None
- Data Bits: 8
- Stop bit: 1
- Flow Control: None

Procedure

Follow the steps below to test the EEPROM emulation application:

1. Connect the console port output to the COM port configured with the HyperTerminal as described in System Configurations.
2. Install *Z8 Encore! XP Applications Library* available under Application Sample Libraries at www.zilog.com.

3. Launch ZDS II and open the `flasheep-rom.zdsproj` project file located in the source folder.
4. Build and download the code to the Z8 Encore! XP® development board.
5. Execute the code. The user interface routine prints a list of menu items in the HyperTerminal window.
6. Select the appropriate menu item and follow the instructions to read and write data from emulated EEPROM. Relevant APIs are called automatically to perform the operation.

Results

The system setup was tested according to the steps provided in the Procedure, and the results obtained were as expected.

Summary

This application note describes a method for EEPROM emulation on Z8 Encore! XP Flash with ready-to-use APIs. This emulation technique uses minimum of two pages of Flash (512 bytes each). These Flash pages provide non-volatile storage capability for a minimum of 128 independent variables (virtual EEPROM addresses) in a code footprint of approximately 3.5 KB, while increasing system reliability. The emulated EEPROM implementation can be extended to suit the designer's requirement for storing more variables.

References

The documents associated with Z8 Encore! XP MCU and ZDS II available on www.zilog.com are provided below:

- Z8 Encore! XP F64xx Series Product Specification (PS0199)
- Z8 Encore! XP F64xx Series Development Kit User Manual (UM0151)
- Zilog Developer Studio II—Z8 Encore! User Manual (UM0130)
- eZ8™ CPU User Manual (UM0128)

Appendix A—Flowcharts

Figure 4 displays the flow to initialize and set up the virtual EEPROM at powerup.

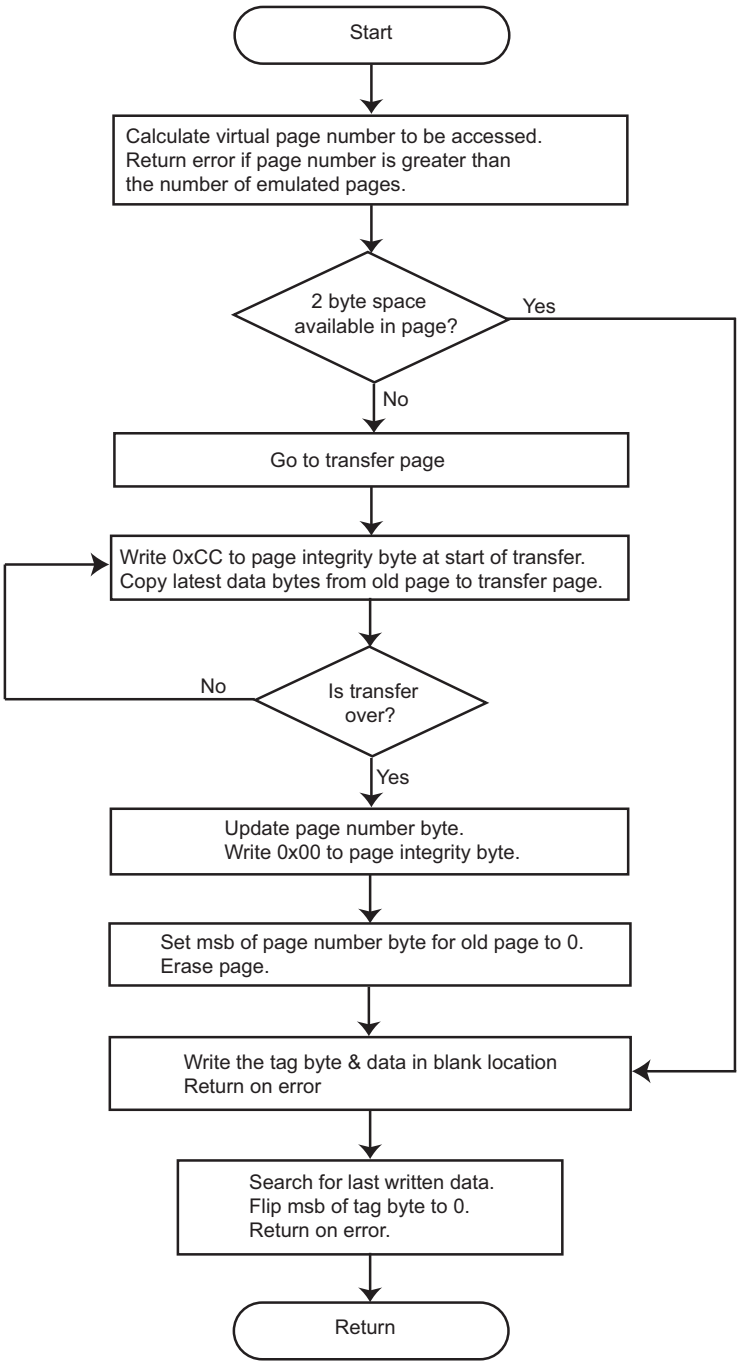

Figure 4. Initializing and Setting up the Virtual EEPROM at Powerup

Figure 5 displays the flow to read a data byte from a virtual EEPROM address.

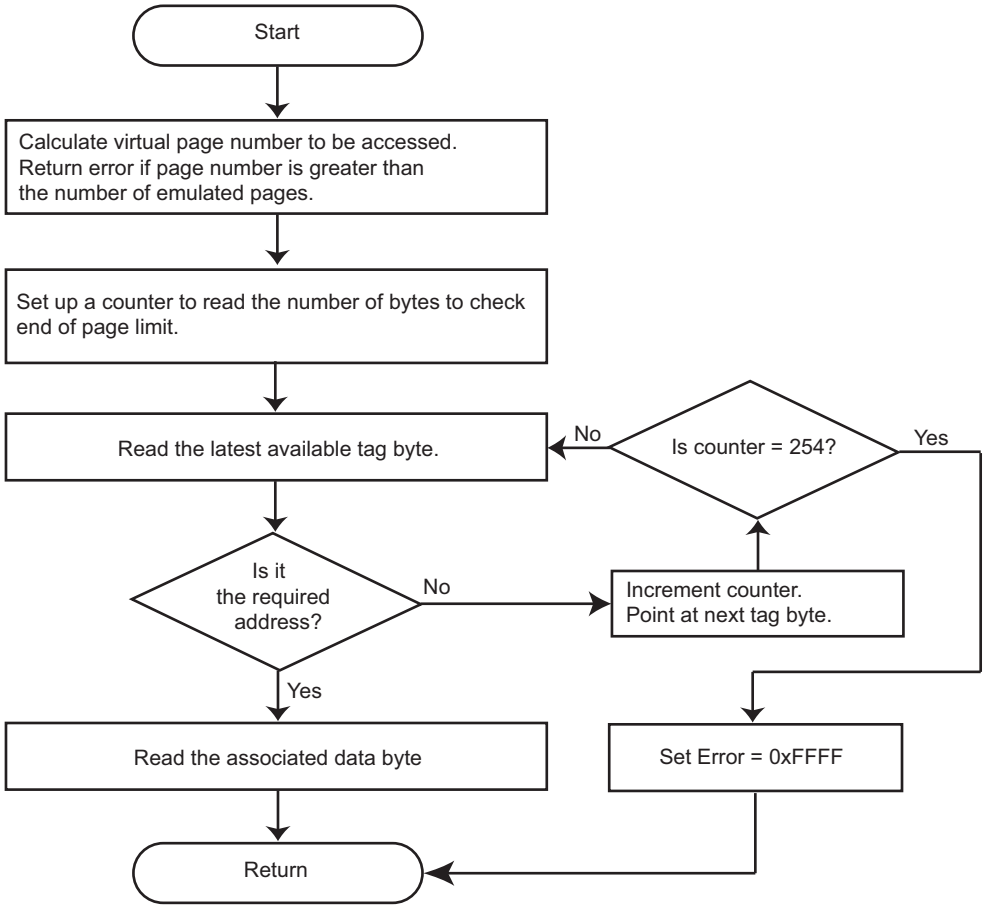

Figure 5. Reading from a Virtual EEPROM Address

Figure 6 displays the flow for writing a data byte to a virtual EEPROM address.

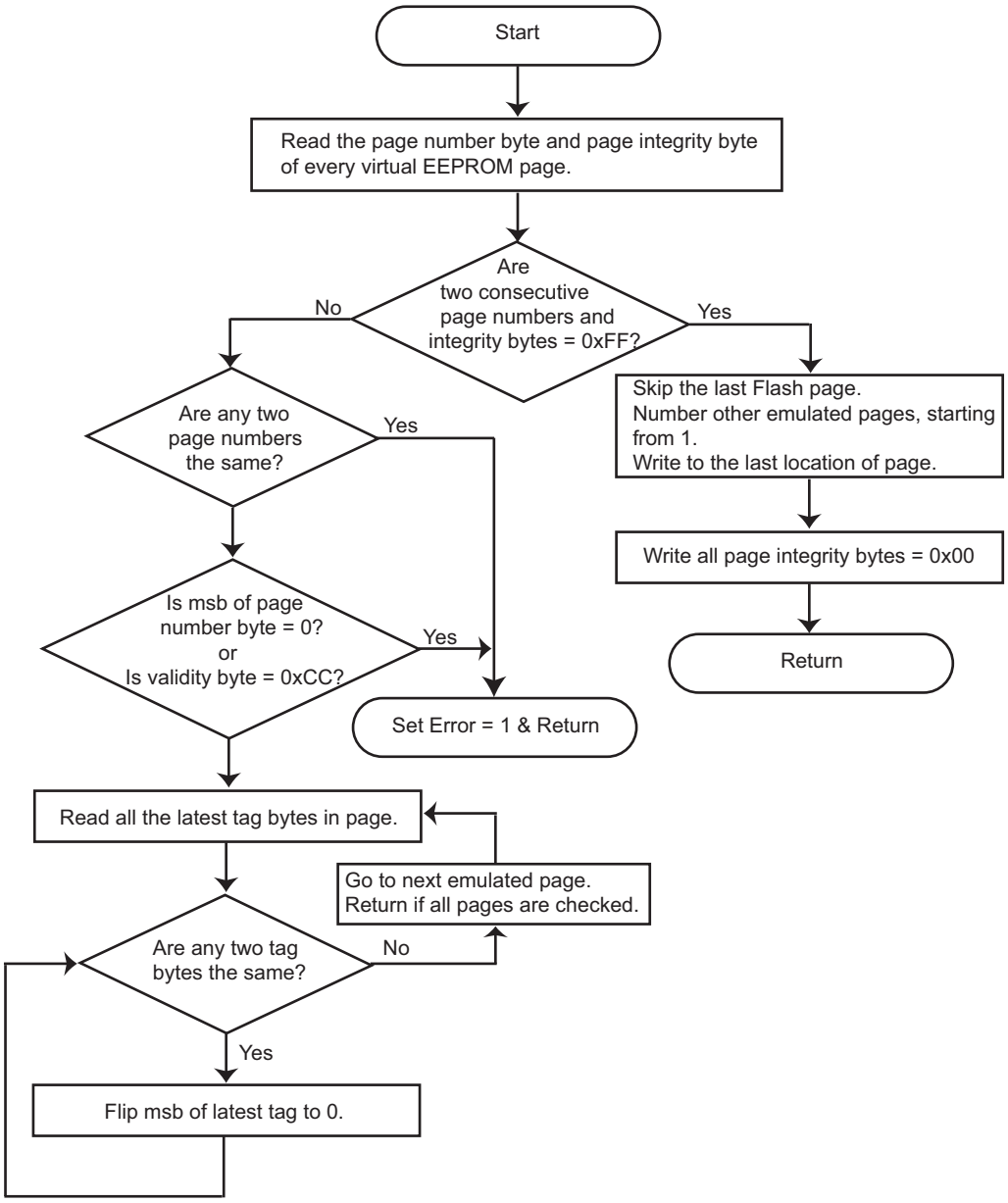

Figure 6. Writing to a Virtual EEPROM Address

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer