Application Note

EEPROM Emulation with the eZ80F91 MCU

AN015803-0608

Abstract

This Application Note describes a method to utilize a portion of Zilog's eZ80Acclaim*Plus!*™ MCU's Flash memory to emulate the functionality of an EEPROM device. The on-chip Flash memory offered in the eZ80Acclaim*Plus!* series of microcontrollers can be used to store both program and data information. The data written to Flash memory remains protected from power loss and inadvertent writes, as special instruction sequences are required to write or erase Flash.

eZ80Acclaim*Plus!* Flash Microcontrollers

eZ80Acclaim*Plus!* on-chip Flash Microcontrollers are an exceptional value for designing high-performance, 8-bit MCU-based systems. With speeds up to 50 MHz and an on-chip Ethernet MAC (eZ80F91 only), you have the performance necessary to execute complex applications quickly and efficiently. Combining Flash and SRAM, eZ80Acclaim*Plus!* devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

The eZ80Acclaim*Plus!* Flash MCU can operate in full 24-bit linear mode addressing 16 MB without a Memory Management Unit. Additionally, support for the Z80-compatible mode allows Z80/Z180 customers to execute legacy code within multiple 64 KB memory blocks with minimum modification. With an external bus supporting eZ80®, Z80®, Intel®, and Motorola® bus modes and a rich set of serial communications peripherals, designers have several options when interfacing to external devices.

Some of the many applications suitable for eZ80Acclaim*Plus!* devices include vending machines, point-of-sale terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

Discussion

This section describes internal Flash memory within the eZ80F91 MCU.

eZ80F91 Internal Flash Memory

The on-chip program Flash memory in the eZ80F91 device utilizes the latest memory technology. It features non-volatile, linearly-addressable Flash memory with in-circuit write/erase capability. A Flash controller block manages physical programming and erase operations by controlling the timing of voltage applied to the Flash memory cells. During programming and erasing operations, the core voltage of 3.3 V is sourced internally and no external power supply is required.

Flash memory space is divided into pages, which consist of consecutive Flash addresses. The eZ80F91 MCU, for example, contains 8 blocks, each comprising 16 pages containing 2048 bytes. In summary, 128 pages comprise 256 KB of Flash memory addresses for eZ80F91. Each Flash byte location consists of 8 Flash cells, each of which represent an individual bit. For a Flash cell, a logical 1 represents the erased state, while a logical 0 represents the programmed state. As a result, a byte location in the erased state is represented as `0xFF` and can be programmed to any other value by flipping its ones to zeroes.

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

However, changing the state of a cell from 0 to 1 can only be performed by erasing the entire page. A *page erase* occurs when all the bytes in a page are erased at the same time, while 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. This guaranteed minimum number is termed as *endurance*.

Flash memory is written and read on a byte-by-byte basis. However, when a Flash location is written, it cannot be rewritten without erasing the page. To write a byte in a Flash location, it is necessary to first unlock the Flash controller by issuing specific instructions in a sequence. For more information, refer to the *eZ80F91 Product Specification (PS0192)*.

The primary purpose of the on-chip Flash memory on the eZ80F91 MCU is to store software programming instructions. Typically, after code is programmed, some portion of Flash memory remains free and unutilized. This unutilized portion of Flash can be used as a virtual EEPROM. A technique for emulating EEPROM is discussed in the following section.

EEPROM Emulation Technique

Generally, an EEPROM chip is used to store and retain data in the event of a power failure. The data is retrieved whenever required and can also be updated in the same location (up to the maximum device rating).

A *virtual EEPROM* can be used instead of an additional EEPROM chip. The virtual EEPROM described in this Application Note is a software written to emulate generic EEPROM functionality on the eZ80F91 device's on-chip Flash. This emulation can be performed in a variety of ways when taking Flash limitations and product requirements into consideration.

At a minimum, all EEPROM emulation implementations require that the data and address pairs are 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 on 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, say every hour, can utilize a virtual EEPROM to store the latest time and temperature value in eZ80F91 on-chip Flash. The only working limitations are that Flash has a finite number of erase cycles and a finite amount of memory.

As writing data into Flash is implemented by software in an emulated EEPROM, the data is prone to corruption or ambiguity due to power failure, fluctuations in voltage supply levels or loss of voltage. These conditions must be considered while designing the software.

This Application Note describes the implementation of an emulation method in the Software Implementation on page 4.

Developing an EEPROM Emulation with eZ80F91 Flash Memory

The EEPROM emulation implemented in this Application Note is based on the following specifications:

- The virtual EEPROM is accessible as a linearly-addressable range, located in the 8th block of eZ80F91 Flash memory. For example, a virtual EEPROM of 1 KB size is addressable from `0x000` to `0x3FF` (see Figure 1).
- The size of data to be stored is one byte in length (ranging from `0x00` to `0xFF`).
- A minimum of two Flash pages are required for storing variables (tag-data pairs)—one to store the 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 the transfer page. As a result, the smallest EEPROM that can be emulated is 128 bytes, and it requires 1 KB of Flash. When one Flash page is filled, only the valid entries are transferred to the other page (the transfer page) when the first page is erased.
- The maximum recommended size for the emulated EEPROM is 1920 bytes (15 pages x 128 variables per page = 1920 emulated bytes), which requires 32 KB of Flash space on the eZ80F91 device (15 pages x 2048 bytes per page = 30 KB + 1 transfer page of 2 KB).
- Each additional 128 bytes of EEPROM emulation requires an additional 2048 bytes of Flash (the size of a page). Therefore, to store 1024 variables (virtual EEPROM size = 1 KB), 9 pages are required (1024 variables ÷ 128 bytes + 1 transfer page), which consume 18 KB of Flash space.

Figure 1 displays an EEPROM mapping of the eZ80F91 MCU Flash memory space.

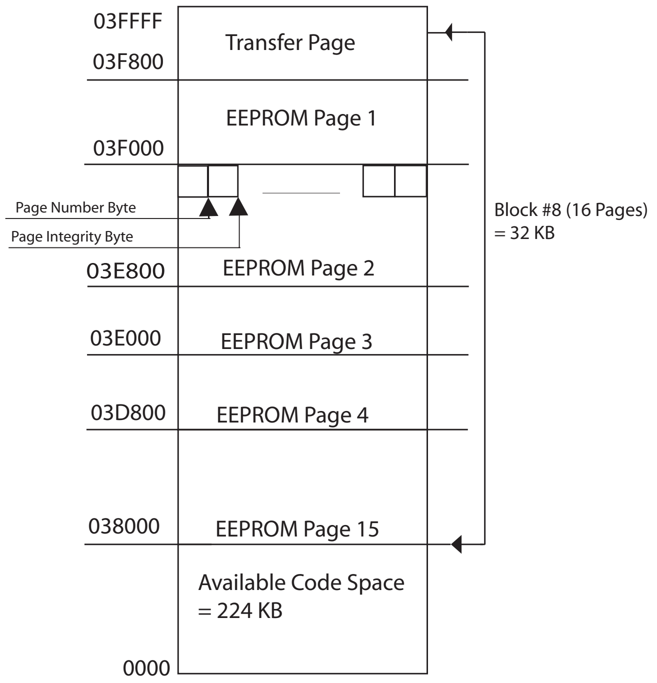


Figure 1. Flash Memory Map of the eZ80F91 Device—Storing Variables in EEPROM

Software Implementation

The software implementation of the EEPROM emulation using the eZ80F91 Flash memory space involves 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 comprise a *variable*. Thus, every variable requires a storage space of two bytes.

Figure 2 displays the tag byte, which is configured as follows:

- The 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 by the below equation:

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

  where,

  virtual EEPROM address = user specified address

  `0x80` = hex value for 128 Bytes that can be stored on a single page

  page number = virtual page number

  The virtual page number is calculated by the below equation:

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

> **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 2048 bytes on the eZ80F91 device) can store 128 independent variables. The first 128 variables (`0x00` to `0x7F`) are stored in virtual page number 1. Next, 128 virtual addresses (`0x80` to `0xFF`) are stored in virtual page number 2, and so on. Virtual page number `0xFF` is a transfer page. There is no virtual page number 0. The virtual page number is stored at the last physical location of the Flash page.

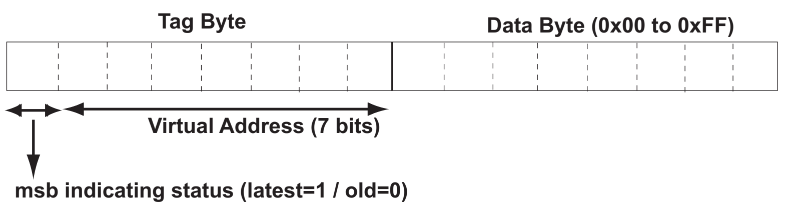


Figure 2. Structure of a Variable

Virtual Page Number Byte and Page Integrity Byte

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

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

A page integrity byte is used to indicate the validity of the data on a page. The page integrity byte values and interpretations are provided below:

- `0xFF` = indicates 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 a freshly-erased Flash memory, the final page of the device is maintained as a transfer page, the prior page acts as virtual page number 1, and so on, up to the allotted number of pages. This numbering changes as and when the data is modified and stored due to page erasures.*

On power-up, `Z_initialize_EEPROM()` API validates the stored data and recovers from any previous power failure situation and resultant corruption of data. This API is described in Appendix B—EEPROM Emulation APIs on page 12.

Figure 4 on page 8 displays the steps involved to initialize the virtual EEPROM. The initialization erases and formats the emulated EEPROM area.

Figure 6 on page 10 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, then the program skips the writing function to preserve storage space.*

Protection Against Power Loss

The software overhead associated with writing a byte to Flash memory can expose the data to corruption or ambiguity (for example, if the address information is corrupted) due to a power failure. Data corruption can also take place due to fluctuations in voltage supply levels or total loss of voltage while writing a byte to Flash. Errors due to these cases must be considered when designing the software.

In the `AN0158-SC01.zip` file, the `Z_Initialize_EEPROM()` routine in the `Z_API.c` file consists of code implementation to eliminate errors occurring due to power loss.

At the time of initialization, a check is conducted after a Power-On Reset 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. This action preserves the final data byte associated with the address and ensures that the old data is retained. The latest address byte is invalidated because the data byte associated with it is suspect and can become corrupted due to various reasons.

Testing the EEPROM Emulation APIs

You can modify EEPROM emulation APIs through an interactive application that drives an RS-232 terminal. The testing of the APIs is performed using the HyperTerminal program on a standard PC.

Raw Flash memory Reads and Writes are performed using additional functions as provided in the project.

Setup

Figure 3 displays the eZ80F91-based MCU and the PC (with the HyperTerminal application).

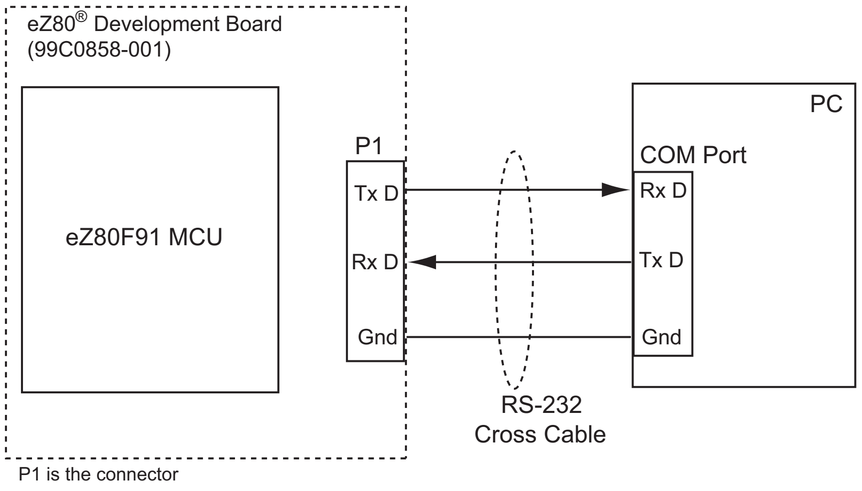


Figure 3. Testing EEPROM Emulation with the eZ80F91 MCU

System Configuration

The HyperTerminal application settings are provided below:

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

Equipment Used

The equipment used for testing include:

- eZ80F91 Development Kit featuring the eZ80F91 Module and an onboard MAX3245
- eZ80F91 eZ80Acclaim*Plus!*™ Development Kit (eZ80 ASSP)
- ZDS II IDE v4.11.0 for eZ80Acclaim!® devices
- PC running MS Windows 95/98/NT/XP with the HyperTerminal application

Procedure

Follow the steps below to test the EEPROM emulation software:

1. Build the project using the ZDS II IDE. All the source code files in the `AN0158-SC01.zip` file are used for testing.
2. Configure the HyperTerminal application as described in System Configuration.
3. Reset the microcontroller and wait for a prompt string on the HyperTerminal screen to appear. The screen displays a software version number and a help message. If it is the first instance of using the virtual EEPROM, then the `Z_Initialize_EEPROM()` API initializes and sets up the virtual EEPROM area (see Figure 4 on page 7).

On any subsequent call, this API checks on the virtual EEPROM area for data integrity and displays an appropriate message depending on the result.

4. To read raw data from Flash, enter `X` at the prompt, followed by the absolute Flash address. These entries verify the data contained in the Flash pages.

5. To write raw data to the Flash pages, enter `Y` followed by the absolute Flash address and the hexadecimal data to be written.

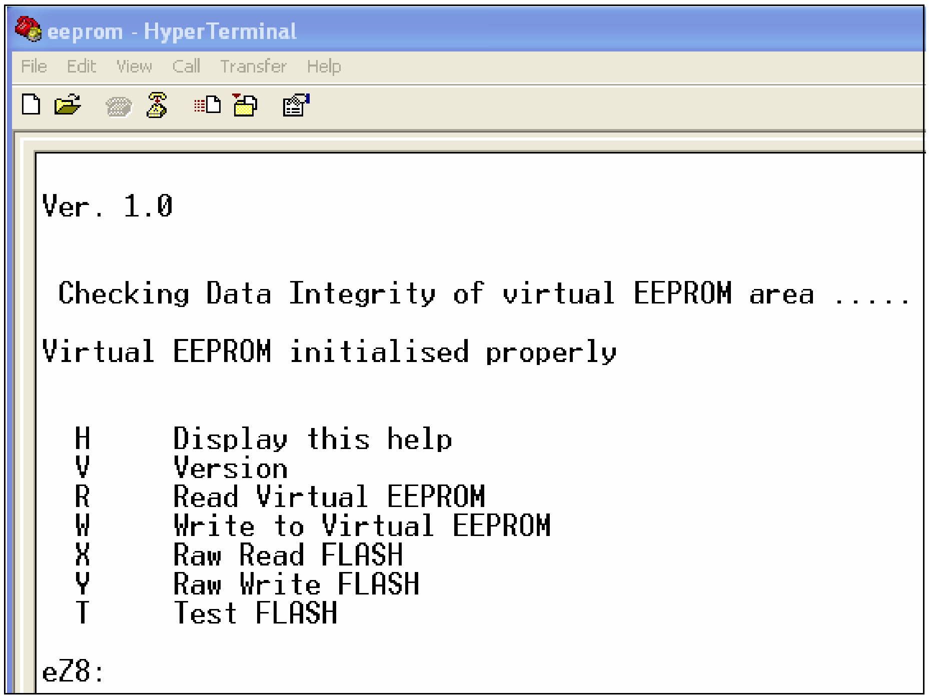
> **Note:** *Do not immediately write the data to the same Flash address (back-to-back).*

6. Use the `T` key to run the `Z_Initialize_EEPROM()` API for verifying data integrity in the virtual EEPROM space.

7. To write to any virtual EEPROM address, enter `W` at the prompt, followed by the virtual address (in 3 hex digits) and the data (2 hex digits). If the new data is same as the data stored in the virtual address location, then the write is not performed and Flash space is preserved. Otherwise, the new data and the virtual address are written to a new Flash location within the virtual EEPROM space.

8. To read any virtual EEPROM address, enter `R` at the prompt, followed by the virtual address (3 hex digits). The most recent data is read from the location and displayed onscreen. The value for a non-existing virtual address is `0xFF` by default.

9. Repeat steps 7 and 8 and verify the results by using the raw read command, `X`.



```
Ver. 1.0


 Checking Data Integrity of virtual EEPROM area .....

Virtual EEPROM initialised properly


  H     Display this help
  V     Version
  R     Read Virtual EEPROM
  W     Write to Virtual EEPROM
  X     Raw Read FLASH
  Y     Raw Write FLASH
  T     Test FLASH

eZ8:
```

Figure 4. Screen Shot of EEPROM Emulation on HyperTerminal Window

Results

The system setup was tested as per the steps described in Procedure on page 6 and the results obtained were as expected.

Summary

This Application Note describes a method for emulating EEPROM within the Flash memory space of the eZ80F91 MCU with ready-to-use APIs. These APIs utilize a minimum of two pages of Flash that are each 2048 Bytes in size. 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 requirement for storing more variables.

References

Further details about eZ80F91 and ZDS II can be found in the references listed below:

- eZ80F91 MCU Product Specification (PS0192)
- eZ80F91 Module Product Specification (PS0193)
- eZ80F91 ASSP Product Specification (PS0270)
- Zilog Developer Studio II –eZ80Acclaim!® User Manual (UM0144)
- eZ80® CPU User Manual (UM0077)

Appendix A—Flowcharts

Figure 4 displays the flowchart to initialize and set up the virtual EEPROM at power-up.

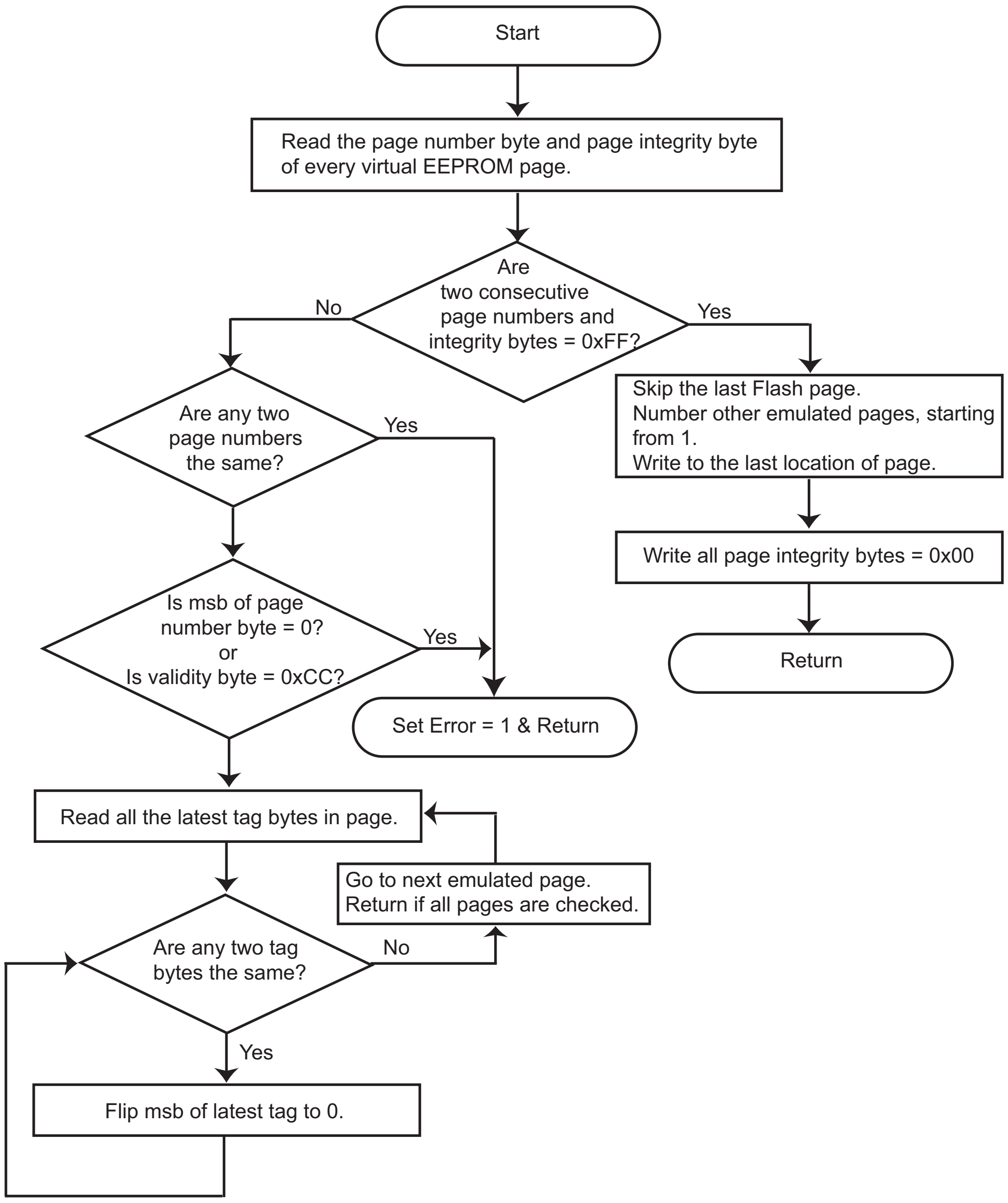

Start
Read the page number byte and page integrity byte of every virtual EEPROM page.
Are two consecutive page numbers and integrity bytes = 0xFF?
No
Yes
Skip the last Flash page. Number other emulated pages, starting from 1. Write to the last location of page.
Write all page integrity bytes = 0x00
Return
Are any two page numbers the same?
Yes
Is msb of page number byte = 0? or Is validity byte = 0xCC?
Yes
Set Error = 1 & Return
Read all the latest tag bytes in page.
Go to next emulated page. Return if all pages are checked.
Are any two tag bytes the same?
No
Yes
Flip msb of latest tag to 0.

Figure 4. Initializing and Setting Up the Virtual EEPROM at Power-Up

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

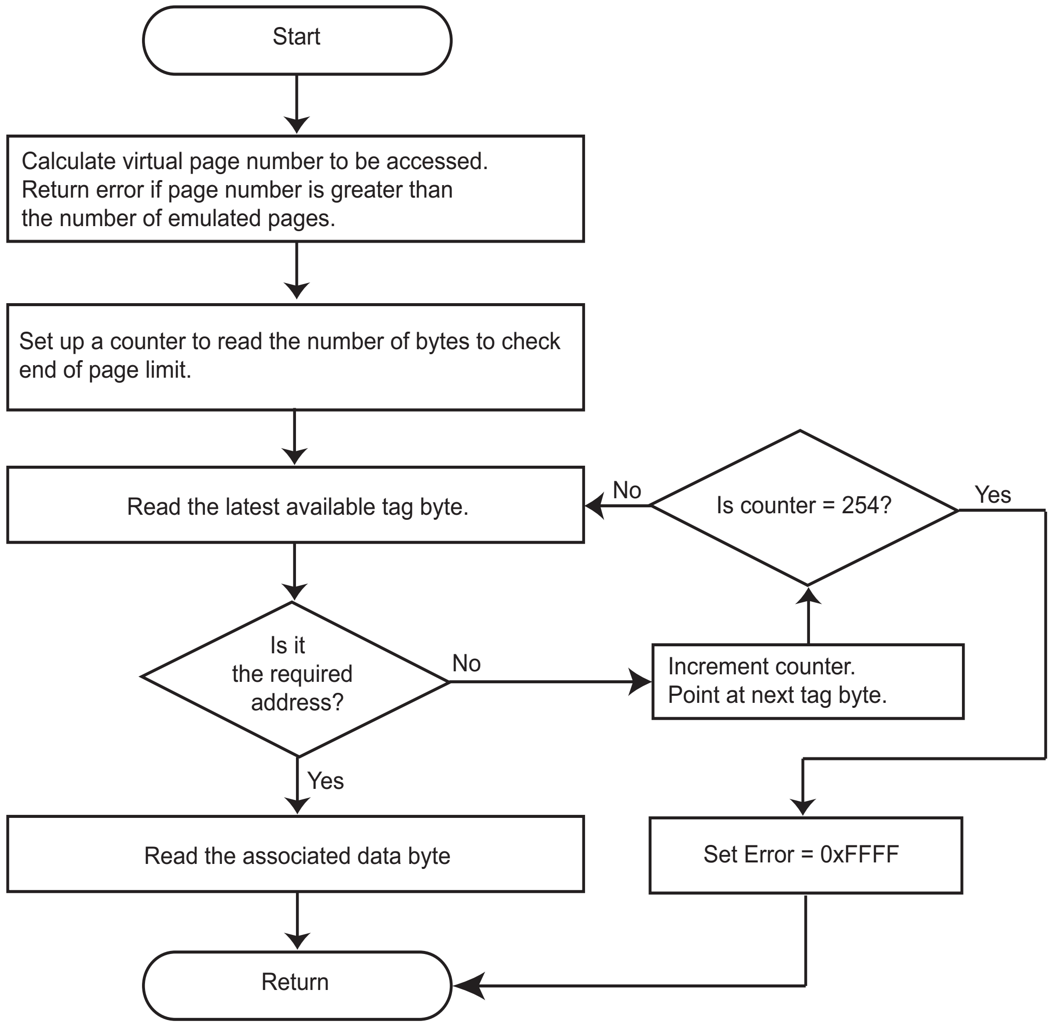

Start
Calculate virtual page number to be accessed. Return error if page number is greater than the number of emulated pages.
Set up a counter to read the number of bytes to check end of page limit.
Read the latest available tag byte.
No
Is counter = 254?
Yes
Is it the required address?
No
Increment counter. Point at next tag byte.
Yes
Read the associated data byte
Set Error = 0xFFFF
Return

Figure 5. Reading from a Virtual EEPROM Address

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

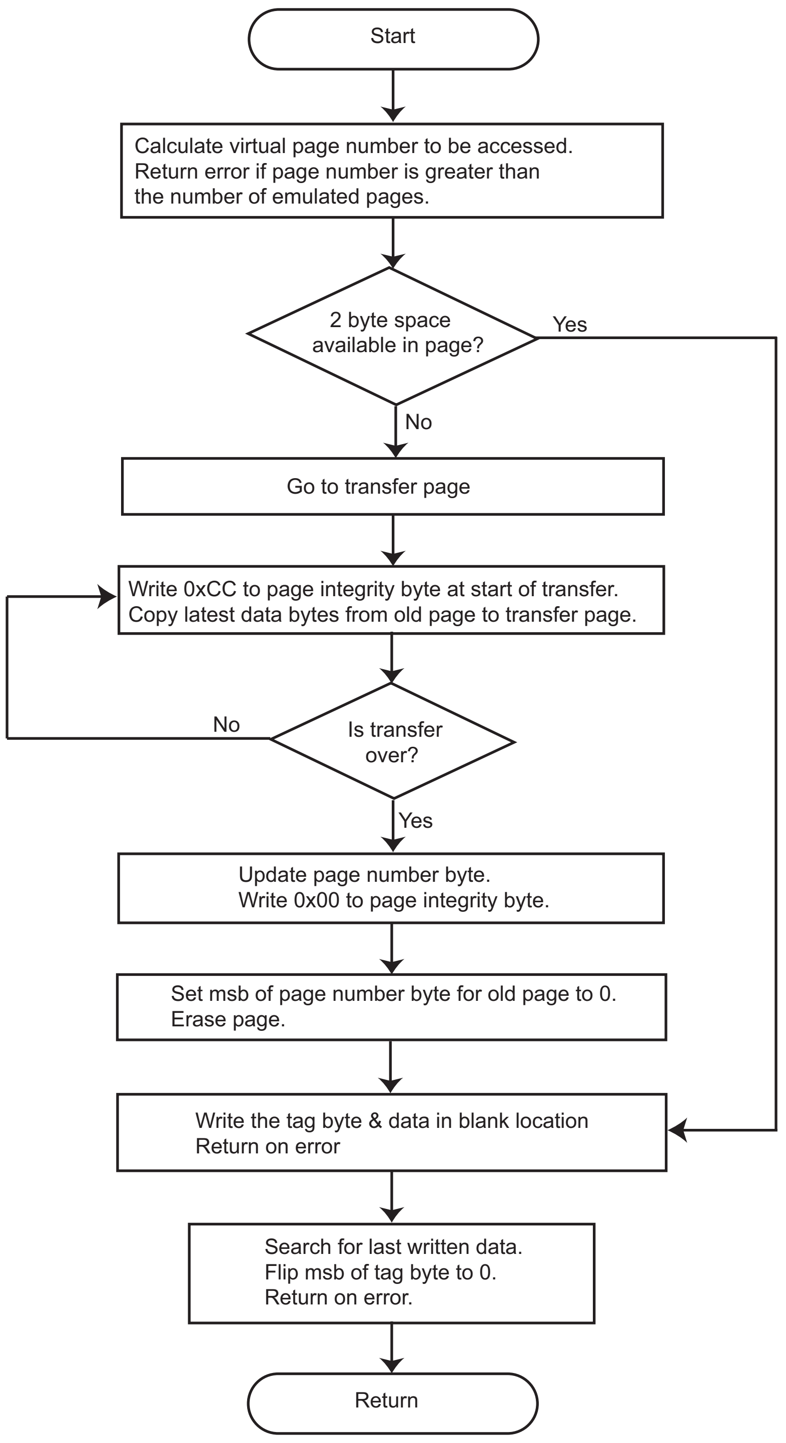

Start
Calculate virtual page number to be accessed.
Return error if page number is greater than
the number of emulated pages.
2 byte space
available in page?
Yes
No
Go to transfer page
Write 0xCC to page integrity byte at start of transfer.
Copy latest data bytes from old page to transfer page.
No
Is transfer
over?
Yes
Update page number byte.
Write 0x00 to page integrity byte.
Set msb of page number byte for old page to 0.
Erase page.
Write the tag byte & data in blank location
Return on error
Search for last written data.
Flip msb of tag byte to 0.
Return on error.
Return

Figure 6. Writing to a Virtual EEPROM Address

Appendix B—EEPROM Emulation APIs

This appendix describes the following EEPROM Emulation APIs which effect EEPROM emulation of an eZ80F91 Flash location:

- Initialization of API—Z_Initialize_EEPROM
- Read API—Z_Read_EEPROM
- Write API—Z_Write_EEPROM

Initialization of API—Z_Initialize_EEPROM

Description

The `Z_Initialize_EEPROM()` API sets up and initializes the emulated EEPROM on power-up. This API must be called on microcontroller power-up, before the other APIs. On successful execution, this API functions as a clean-up routine and checks the data integrity of the emulated EEPROM area. It returns an error on failure.

Parameters

None

Return Value(s)

`0x00`	On Success.
`0xFF`	On Failure due to inaccessible Flash.

Usage

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

Read API—Z_Read_EEPROM

Description

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 function returns a `0xFFFF`.

Figure 5 displays the process of reading a data byte from a virtual EEPROM address.

Parameters

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

Return Value(s)

`0x00XX`	On Success, where XX represents the data read.
`0xFFFF`	On Failure.

Usage

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

Write API—Z_Write_EEPROM

Description

The `Z_Write_EEPROM()` API writes the data byte to the virtual EEPROM address. On successful execution, this API returns `0x00` and returns error (`0xFF`) on failure.

Parameters

`int`	Virtual EEPROM address.
`unsigned char`	Data.

Returns

`0x00`	On Success.
`0xFF`	On Failure.

Usage

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

Warning: DO NOT USE IN LIFE SUPPORT