

## ADuCM3027/ADuCM3029 Flash EEPROM Emulation

### INTRODUCTION

Nonvolatile data storage is a necessity in many embedded systems. Data, such as bootup configuration, calibration constants, and network related information, is normally stored on an electronically erasable programmable read only memory (EEPROM) device. The advantage of using the EEPROM to store this data is that a single byte on an EEPROM device can be rewritten or updated without affecting the contents in the other locations.

The [ADuCM3027/ADuCM3029](#) are ultra low power microcontroller units (MCUs) with integrated flash memory. The [ADuCM3027/ADuCM3029](#) include 512 kB of embedded flash memory with a 72-bit wide data bus that provides two 32-bit words of data and one corresponding 8-bit error correction code (ECC) byte per access. The ECC check is enabled by default for the user space on the [ADuCM3027/ADuCM3029](#) and provides assurance that the flash initialization function works as expected. The ECC check is enabled for the entire user space in the flash of the [ADuCM3027/ADuCM3029](#). If there are ECC errors reported during any read operation, the ECC engine automatically corrects 1-bit errors and only reports on detection of 2-bit errors. When a read occurs on the flash, appropriate flags are set in the status register of [ADuCM3027/ADuCM3029](#). If interrupts are generated, the source address of the ECC error

causing an interrupt is available in the `FLCC0_ECC_ADDR` register for the interrupt service routine (ISR) to read.

Emulation of the EEPROM on the integrated flash reduces the bill of material (BOM) cost by omitting the EEPROM in the design. Therefore, the software complexity is also reduced.

### BACKGROUND

Flash memory is typically organized as an array of pages. A single page in the [ADuCM3027](#) is 2 kB. The contents of the page must be erased before writing data. The erase operation is universal to the page, whereas the read or write can be performed on a single addressable location (byte or word).

The challenges of performing the read or write on a single addressable location are as follows:

- Byte wide data read and write operations.
- The ability to erase or update data at any location while retaining data at other locations because flash memory erase operates on an entire page.

This application note describes the software using the [ADuCM3027/ADuCM3029](#) devices and the built in flash memory to emulate EEPROM, as shown in Figure 1.

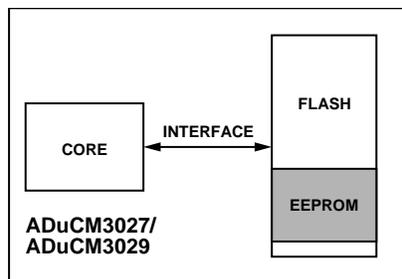


Figure 1. [ADuCM3027/ADuCM3029](#) Internal Flash and EEPROM System Overview

## TABLE OF CONTENTS

Introduction .....	1	EEPROM .....	3
Background .....	1	Flash .....	5
Revision History .....	2	Limitations .....	7
Working Principle.....	3	Conclusion.....	8

## REVISION HISTORY

### 6/2018—Rev. 0 to Rev. A

Changes to Introduction.....	1
Changes Figure 2, Initialize EEPROM Section, and Write EEPROM Section.....	3
Change to Read EEPROM Section and Figure 3.....	4
Changes to Figure 4, Figure 5, and Flash Section.....	5
Change to Limitations Section.....	7
Changes to Conclusion Section, Table 1, Table 2, Table 3, and Table 4 .....	8
Changes to Table 5, Table 6, and Table 7 .....	9

### 3/2017—Revision 0: Initial Version

## WORKING PRINCIPLE

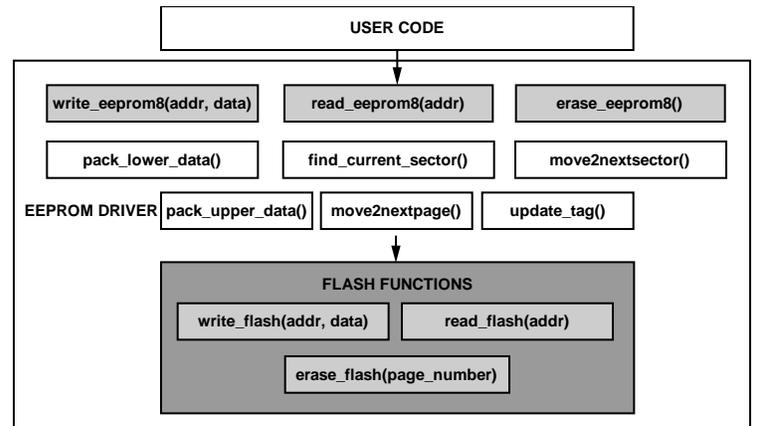


Figure 2. ADuCM3027/ADuCM3029 Flash EEPROM Emulation Software Structure

EEPROM emulation requires a dedicated portion of the flash memory. Most EEPROMs can update one byte in one write command. However, flash memory devices are equipped with multibyte, a writing capability, and update the data accordingly, only if erase sequences are followed between two write operations. To emulate a byte writable and readable EEPROM in the flash memory, it is necessary to follow a read, modify, write sequence, which is similar to EEPROM operation.

The procedure presented in this section uses two flash pages that can be extended to more than two pages, which are then divided into sectors consisting of a sector tag. This sector tag provides information about the current sector under process and number of data bytes written onto that sector. Note that the last location in every sector is reserved for the sector tag, which has a size that is equal to the data bus size of the flash memory. The sector size and the number of sectors in a flash page depends on the size of the emulated EEPROM.

### EEPROM

EEPROM writing and reading functions involve the processing of the application code input, such as EEPROM data and address information. The EEPROM application programming interface (API) handles processing and presenting the data and address information per the requirements of the flash interface.

#### Initialize EEPROM

The initialize EEPROM operation defines the size of the EEPROM and word length of the data in an EEPROM to be emulated. The maximum size of the emulated EEPROM is 2 kB (one flash page) because two flash pages are reserved for EEPROM emulation in this application software to keep the integrity of the data. This limit can be increased if the same application software is extended from reserving two flash pages to reserving four flash pages. Users have a configurable word length ranging from 8-bits to a maximum of 64-bits. See Table 1 for

details regarding the `init_eeprom(uint16_t eeprom_size, uint8_t word_length)` function.

#### Write EEPROM

Figure 3 shows the EEPROM write operation flowchart. The EEPROM write operation procedure is as follows:

1. Find the current sector by using the `find_current_sector()` function call. This search is based on the sector tag and the corresponding sector tag value; the value returned is the current sector start address (which is a physical location on the flash memory).
2. Convert the EEPROM address to the flash address with the help of the current sector start address. Because the ADuCM3027/ADuCM3029 flash memory has a 72-bit wide data bus and the emulated EEPROM has an 8-bit wide data bus, the software determines the number of shifts required using the EEPROM address.
3. Read the data at the obtained flash address and if the data is equal to 0xFF, then the data to be written is first packed into two 32-bit data (`pack_lower_data()` and `pack_upper_data()`) and a 64-bit wide data is prepared to be written onto flash memory.
4. Execute a write command on the flash controller by calling the `write_flash()` function. The input parameters for this function are the flash memory address, the least significant bit (LSB) data packet, and the most significant bit (MSB) data packet.
5. After the write operation to the flash memory is complete, the sector tag of the current sector is updated by calling the `update_tag()` function.

If data is already present at the obtained flash address, the data read function does not return 0xFF. In this case, the data before and after the obtained flash address is moved to the next or adjacent sector by calling the `move2nextsector()` function. The EEPROM data, which is converted to LSB and MSB data

packets, is written at the new flash address on the next sector. Thus, every time a write is issued at an already written location of the EEPROM, the data is moved to the next sector with the location containing modified data.

If the new sector resides on the next page, a flash page erase command is issued by calling `erase_flash(page_number)` to the previous page after the data is moved. All the address registers are updated by the `move2nextpage()` function.

See Table 2 for details regarding the `write_eeprom(uint16_t addr_eeprom, uint8_t data_eeprom)` function.

**Read EEPROM**

Figure 4 shows the EEPROM read operation flowchart. The EEPROM read operation procedure is as follows:

1. Read the EEPROM value stored at the address location by calling the `read_eeprom(addr)` function.
2. In an EEPROM read request from the application code, the software first determines the current sector, which consists of the latest data. A flash address is obtained using the EEPROM address and current sector start address.
3. Execute a read command by calling the `read_flash()` function with the obtained flash address.
4. Process the 64-bit wide data received from the flash address. The bits of this address are then masked, right shifted, and provided to the application code.

See Table 3 for details regarding the `read_eeprom(uint16_t addr_eeprom)` function.

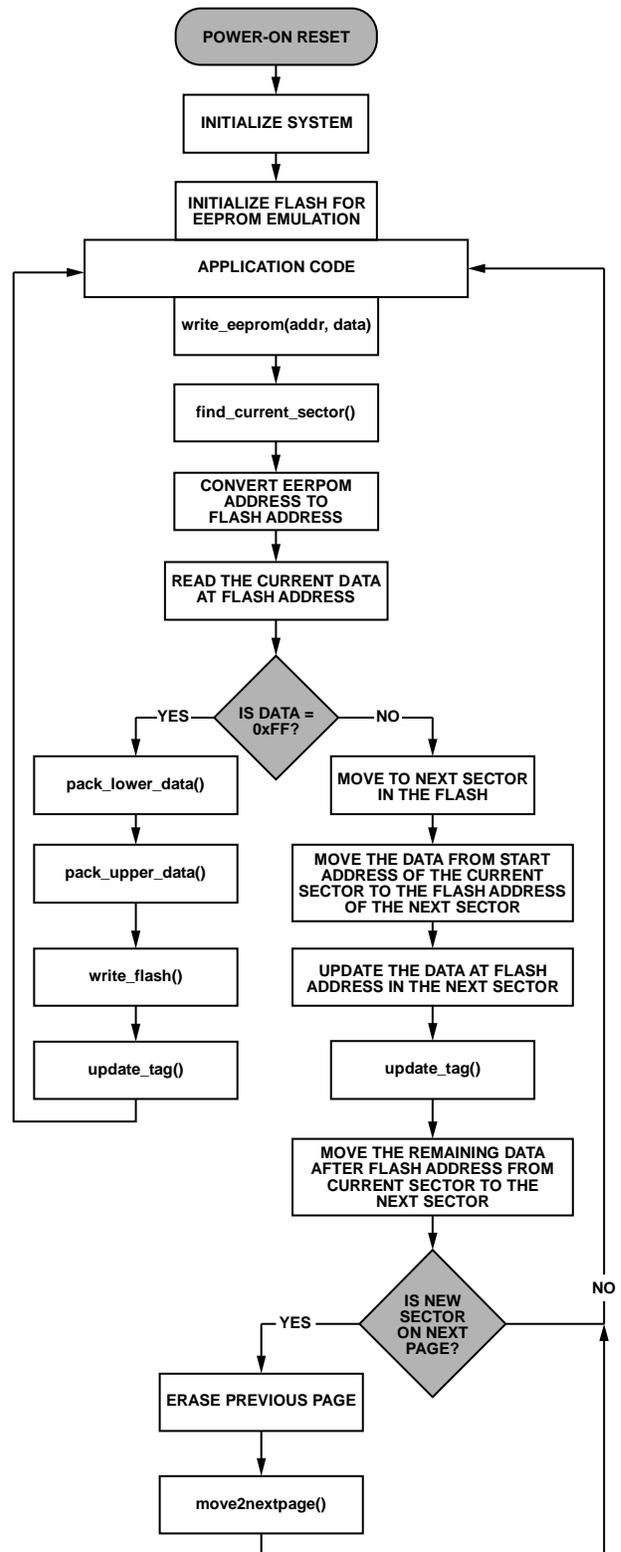


Figure 3. EEPROM Write Operation

156 18-003

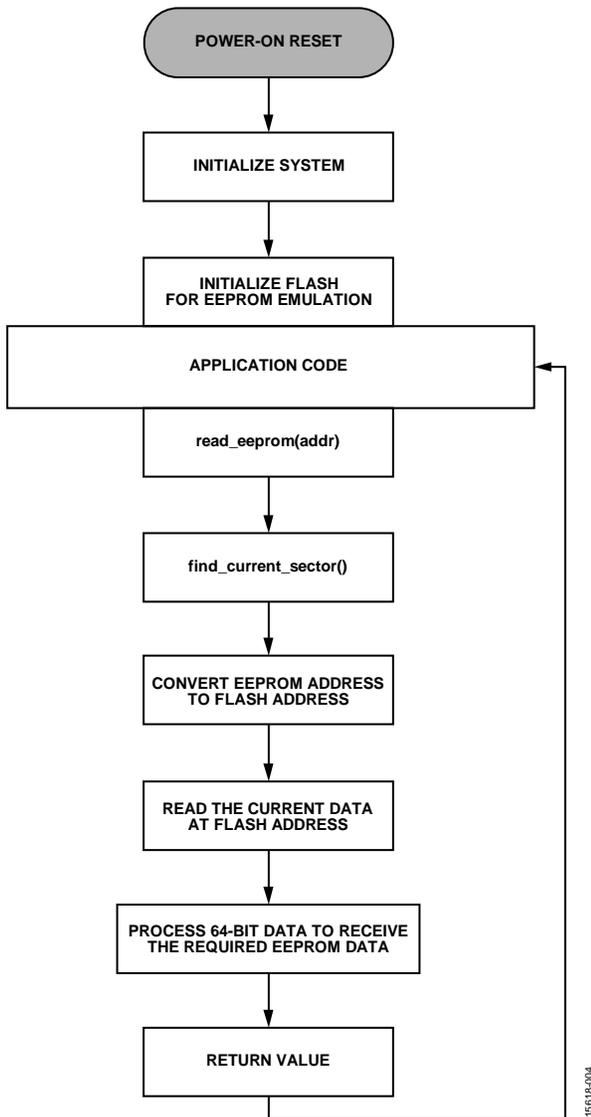


Figure 4. EEPROM Read Operation

15618-004

**Erase EEPROM**

Figure 5 shows the EEPROM erase operation flowchart. The EEPROM erase operation is as follows:

1. Erase the entire EEPROM space allotted in the flash memory by calling the erase\_eeprom() function.

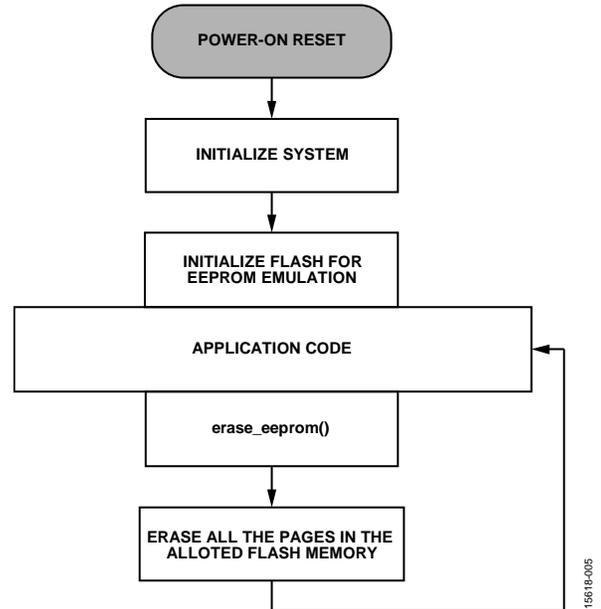


Figure 5. EEPROM Erase Operation

15618-005

All the pages in the flash memory dedicated to EEPROM emulation are erased. Therefore, exercise caution while using this operation in the application code.

See Table 4 for details regarding the erase\_eeprom() function.

**FLASH**

The ADuCM3027/ADuCM3029 processors include 128 kB and 256 kB of embedded flash memory, which are available for access through the flash controller. The embedded flash memory has a 72-bit wide data bus, providing two 32-bit words of data and one corresponding 8-bit ECC byte per access. The memory is organized in pages of 2 kB each, plus 256 bytes reserved for the ECC. ECC is, by default, enabled on the entire user space of flash in the ADuCM3027/ADuCM3029. A write to a location of flash updates the data as well as the ECC byte on that location. Because flash memory cannot change the bit value from Logic 0 to Logic 1 without an erase procedure, only one write operation is allowed. Therefore, ECC bytes update just one time. The next write to the same location without an erase procedure leads to ECC error generation.

**Flash Write**

The flash memory operates by setting bits to 1 when erased, and selectively clearing bits to 0 when writing (programming) data. No write operation is capable of setting any bit to 1 from 0. For this reason, generalized write accesses must be prefixed by an erase operation.

A keyhole write is an indirect write operation in which the user code programs memory mapped registers with target address and data values, then commands the flash controller to perform a write operation in the background. The flash controller supports write access to the flash memory only through keyhole writes. This constraint on write access enables the flash controller to guarantee that writes occur properly as atomic double word (64-bit data) operations.

LSB and MSB data packets that are created using EEPROM data are provided to keyhole data registers. After the assertion of a write command, the flash controller initiates a 64-bit data dual word write to the given flash address.

Note that word (32-bit), half word (16-bit), and byte (8-bit) writes are not supported.

See Table 5 for details regarding the `write_flash(uint32_t addr, uint32_t lower_data, uint32_t upper_data)` function.

### **Flash Erase**

When there is a page change during an EEPROM write, a page erase command is asserted on the previous page by calling the `erase_flash(page)` function. Before a page erase, data movement occurs as explained in the Write EEPROM section.

See Table 6 for details regarding the `erase_flash(uint8_t PAGE)` function.

### **Flash Read**

Flash memory is available to be read only after an automatic initialization process. Reading the flash memory returns a 64-bit data double word.

Flash address information is provided to the flash controller, which returns read data. This data is further processed by the EEPROM interface to achieve the EEPROM value.

See Table 7 for details regarding the `read_flash(uint32_t addr)` function.

## LIMITATIONS

In real EEPROMs, if one location is updated, only one erase cycle is counted, followed by a write operation to that particular address while other locations remain unchanged.

In this emulated EEPROM, updating one location causes movement of data from the current sector to the next sector that consumes the EEPROM size number of write cycles. Therefore, every time a location is updated, data is moved to the next sector and, if that sector resides on the next page, a page erase occurs. This behavior decreases the effective endurance of the flash memory.

To overcome these limitations,

- Use caution when selecting the EEPROM size for emulation. A small EEPROM size decreases the write cycles during data movement, which indirectly increases the endurance of the flash memory.
- Avoid unnecessary writes to the EEPROM. By doing so, the effective endurance of the flash memory increases. For example, the system must issue writes only during a power fail sequence. A RAM buffer can be used for storing the data during normal operations. Note that the software handles some of the unnecessary writes to the emulated EEPROM. For example, if the data to be written is 0xFF and the current data at that particular location is 0xFF, no write is issued to the flash memory.

## CONCLUSION

The application note intends to match the physical difference between the EEPROM and the flash memory using the [ADuCM3027/ADuCM3029](#). This emulated EEPROM is similar to a real EEPROM that overcomes the problems related to, for example, silicon area, input and output bus resources, and manufacturing costs.

This application note provides the user with a large EEPROM size (from 8 bytes to 2 kB (one page)) because this application software reserves two flash pages for EEPROM emulation. If the

software is extended to four flash pages, the maximum EEPROM size is 4 kB with a configurable word length of 8 bits to 64 bits.

Because there is a trade-off between the size of the emulated EEPROM and the endurance of the flash memory, select the appropriate size to enhance the hardware efficiency. In addition to this approach, the software handles some of the unnecessary writes to the [ADuCM3027/ADuCM3029](#) flash device, which effectively increases the endurance.

**Table 1. Initialize EEPROM Function Description—`init_eeprom(uint16_t eeprom_size, uint8_t word_length)`**

Parameter	Description	Return Value
<code>eeprom_size</code>	Size of the EEPROM to emulate.	No error. Write complete. Error. The given address is out of the available EEPROM memory space.
<code>word_length</code>	Length of the data word to be written in the emulated EEPROM. Valid values are 8, 16, 32 or 64.	No error. Write complete.  Error. The given address is out of the available EEPROM memory space.

**Table 2. Write EEPROM Function Description—`write_eeprom(uint16_t addr_eeprom, uint8_t data_eeprom)`<sup>1</sup>**

Parameter	Description	Return Value
<code>addr_eeprom</code>	Logical address in the EEPROM space where data is written.	No error. Write complete. Error. The given address is out of the available EEPROM memory space.
<code>data_eeprom</code>	Data written to the EEPROM space, pointed by <code>addr_eeprom</code> .	No error. Write complete. Error. The given address is out of the available EEPROM memory space.

<sup>1</sup> This function writes data to the EEPROM.

**Table 3. Read EEPROM Function Description—`read_eeprom(uint16_t addr_eeprom)`<sup>1</sup>**

Parameter	Description	Return Value
<code>addr_eeprom</code>	Logical address in the EEPROM space where data is read.	Value. The 8-bit data is returned to the application code. Error. The given address is out of available EEPROM memory space.

<sup>1</sup> This function reads data from the EEPROM.

**Table 4. Erase EEPROM Function Description—`erase_eeprom()`<sup>1</sup>**

Parameter	Return Value
Not applicable	No error. Erase complete. Error. The flash controller is busy and cannot perform the erase action.

<sup>1</sup> This function erases the EEPROM memory space. All the data is lost if this function is called.

**Table 5. Write Flash Function Description—write\_flash(uint32\_t addr, uint32\_t lower\_data, uint32\_t upper\_data)<sup>1</sup>**

Parameter	Description	Return Value
addr	The address in the flash memory space allotted for the EEPROM emulation.	No error. Write complete. Error. The given address is out of the available EEPROM memory space. Not applicable.
lower_data	The lower 32 bits of the double word.	No error. Write s complete. Error. The given address is out of the available EEPROM memory space. Not applicable.
upper_data	The highest 32 bits of the double word.	No error. Write complete. Error. The given address is out of the available EEPROM memory space. Not applicable.

<sup>1</sup> This function receives the translated EEPROM address and data from the write\_eeprom() function and issues a write command to the flash controller.

**Table 6. Erase Flash Function Description—erase\_flash(uint8\_t PAGE)<sup>1</sup>**

Parameter	Description	Return Value
Page	The page number of the allotted flash memory space.	No error. Page wise erase complete. Error. The given page value is out of the allotted flash memory space.

<sup>1</sup> This function performs a page wise erase in the allotted flash memory space.

**Table 7. Read Flash Function Description—read\_flash(uint32\_t addr)<sup>1</sup>**

Parameter	Description	Return Value
addr	The address in the flash memory space allotted for EEPROM emulation.	Read data. The data available at the desired address is returned to the function. Error. The translated address is out of the allotted flash memory space.

<sup>1</sup> This function receives the translated EEPROM address from the read\_eeprom() function and issues a read command to the flash controller.