ANALOG DEVICES

AN-1452
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

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 is an ultra low power microcontroller unit (MCU) with integrated flash memory. Emulation of the EEPROM on the integrated flash reduces the 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.

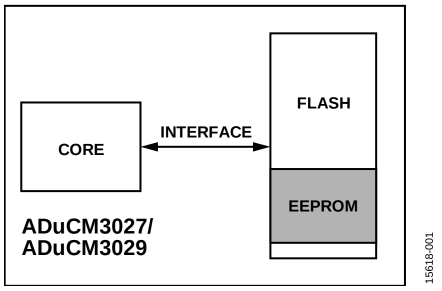

Figure 1. ADuCM3027/ADuCM3029 Internal Flash and EEPROM System Overview

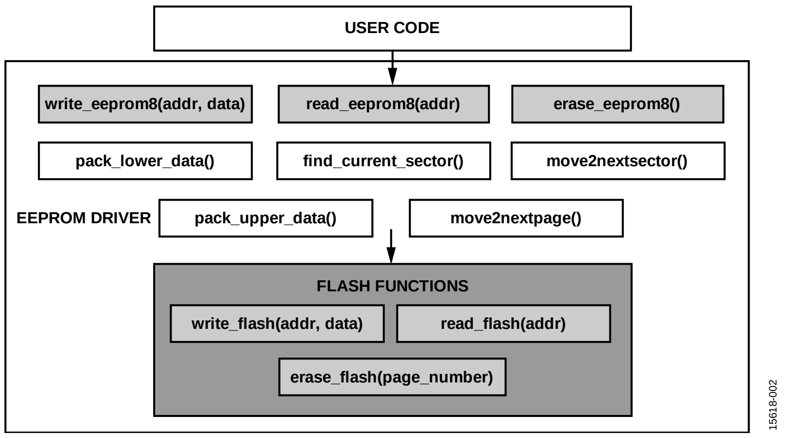

Figure 2. ADuCM3027/ADuCM3029 Flash EEPROM Emulation Software Structure

TABLE OF CONTENTS

Introduction 1
Background 1
Revision History 2
Working Principle 3
EEPROM 3
Flash 5
Limitations 6
Conclusion..................... 7

REVISION HISTORY

3/2017—Revision 0: Initial Version

WORKING PRINCIPLE

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 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 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 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.

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 64-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; if this data is equal to 0xFF, least significant bit (LSB) and most significant bit (MSB), 32-bit packets are created by masking the bits and performing a left shift operation on the EEPROM data to form a 64-bit wide data set to be written to the 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 and the LSB and MSB data packets.
5. After a successful write operation to the flash memory, 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 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 1 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 address location by calling the read_eerprom(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 2 for details regarding the read_eeprom(uint16_t addr_eeprom) function.

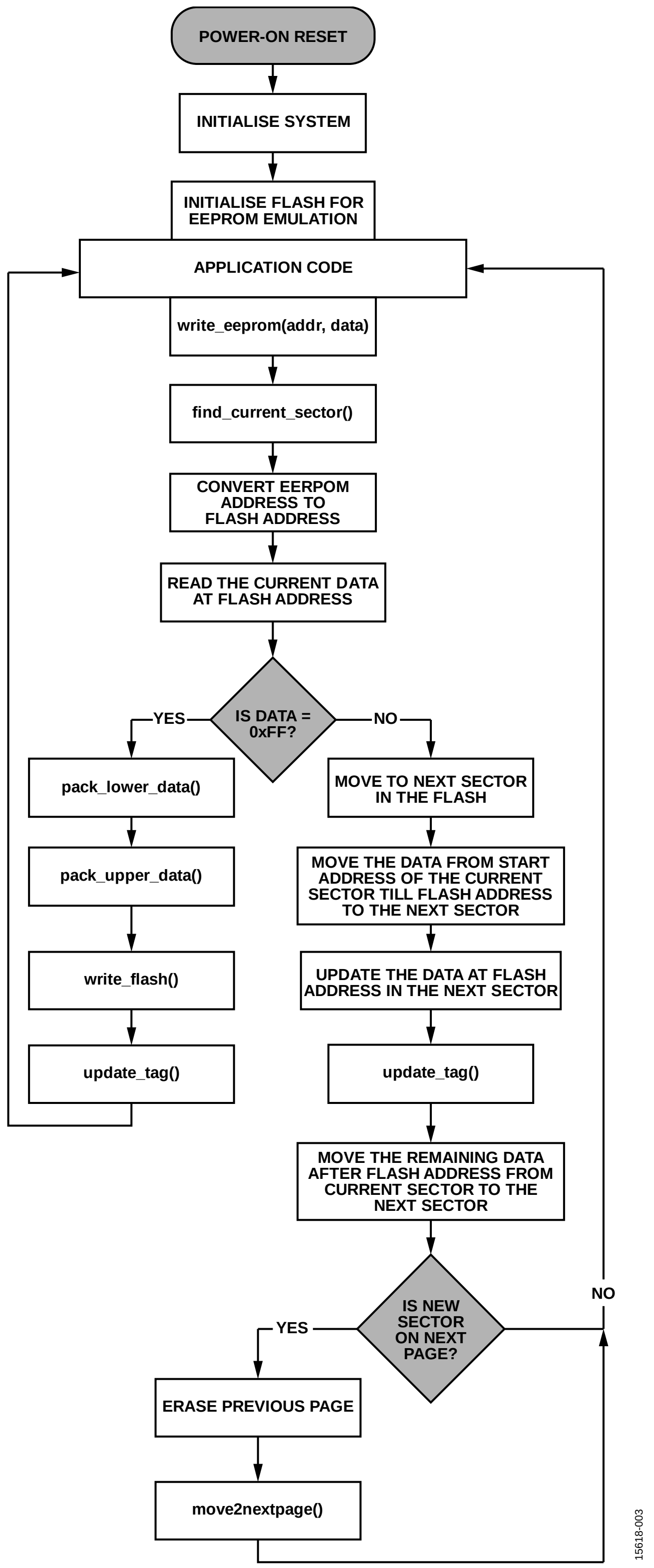
POWER-ON RESET
INITIALISE SYSTEM
INITIALISE FLASH FOR EEPROM EMULATION
APPLICATION CODE
write_eeprom(addr, data)
find_current_sector()
CONVERT EERPOM ADDRESS TO FLASH ADDRESS
READ THE CURRENT DATA AT FLASH ADDRESS
IS DATA = 0xFF?
YES
NO
pack_lower_data()
pack_upper_data()
write_flash()
update_tag()
MOVE TO NEXT SECTOR IN THE FLASH
MOVE THE DATA FROM START ADDRESS OF THE CURRENT SECTOR TILL FLASH ADDRESS TO THE NEXT SECTOR
UPDATE THE DATA AT FLASH ADDRESS IN THE NEXT SECTOR
update_tag()
MOVE THE REMAINING DATA AFTER FLASH ADDRESS FROM CURRENT SECTOR TO THE NEXT SECTOR
IS NEW SECTOR ON NEXT PAGE?
YES
NO
ERASE PREVIOUS PAGE
move2nextpage()
15618-003

Figure 3. EEPROM Write Operation

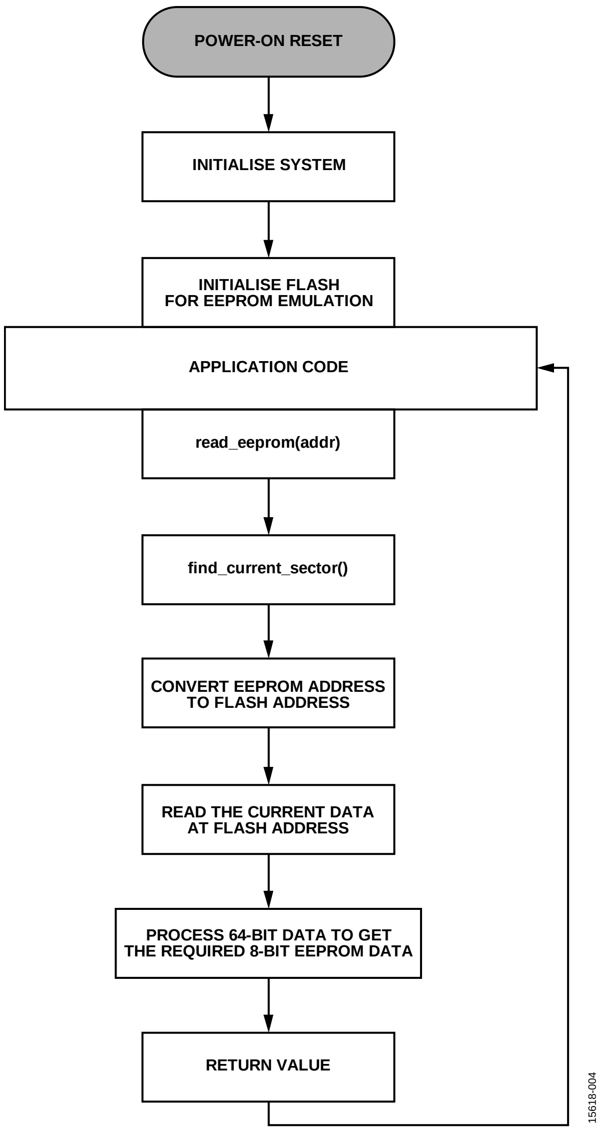
POWER-ON RESET
INITIALISE SYSTEM
INITIALISE FLASH FOR EEPROM EMULATION
APPLICATION CODE
read_eeprom(addr)
find_current_sector()
CONVERT EEPROM ADDRESS TO FLASH ADDRESS
READ THE CURRENT DATA AT FLASH ADDRESS
PROCESS 64-BIT DATA TO GET THE REQUIRED 8-BIT EEPROM DATA
RETURN VALUE
15618-004

Figure 4. EEPROM Read Operation

Erase EEPROM

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

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

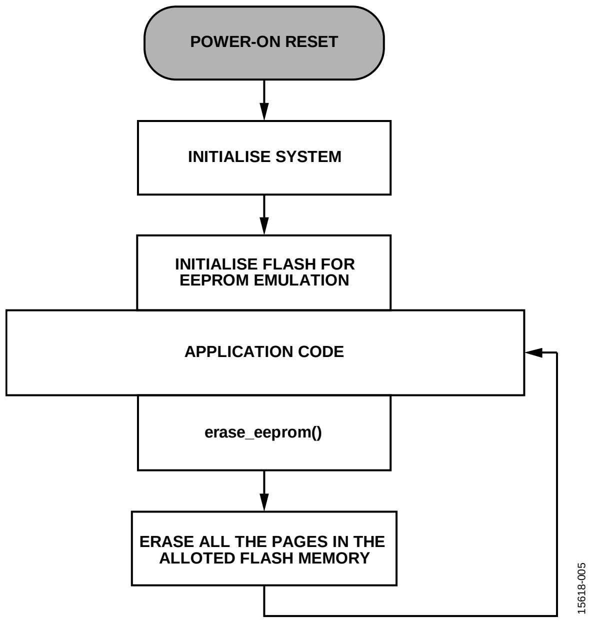
POWER-ON RESET
INITIALISE SYSTEM
INITIALISE FLASH FOR EEPROM EMULATION
APPLICATION CODE
erase_eeprom()
ERASE ALL THE PAGES IN THE ALLOTED FLASH MEMORY
15618-005

Figure 5. EEPROM Erase Operation

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 3 for details regarding the erase_eeprom() function.

FLASH

The ADuCM3027/ADuCM3029 processors include 128 kB and 256 kB of embedded flash memory, 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 error correction code (ECC) byte per access. The memory is organized in pages of 2 kB each, plus 256 bytes reserved for the ECC.

Flash Write

The flash memory operates by settings 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) operations.

LSB and MSB data packets, which 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 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 4 for details regarding the write_flash(uint32_t addr, uint32_t lower_data, uint32_t upper_data) function.

Flash Read

Flash memory is available to be read only after an automatic initialization process. Reading the flash memory returns a 64-bit 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 6 for details regarding the read_flash(uint32_t addr) 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 previously.

See Table 5 for details regarding the erase_flash(uint8_t PAGE) 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 remains 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 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 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 silicon area, input/output bus resources, manufacturing costs, and so on.

This application note provides the user with a large EEPROM size (from 64 bytes to 1024 bytes) for emulation. Because there is a trade-off between the size of the emulated EEPROM and the endurance of the flash memory, the user is advised to 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. Write EEPROM Function Description

write_eeprom(uint16_t addr_eeprom, uint8_t data_eeprom)[1]		
Parameter	**Description**	**Return Value**
addr_eeprom	Logical address in the EEPROM space where data is to be written	No error; write was successful
data_eeprom	Data to be written to the EEPROM space, pointed by addr_eeprom	Error; the given address is out of the available EEPROM memory space

[1] This function writes data to the EEPROM.

Table 2. Read EEPROM Function Description

read_eeprom(uint16_t addr_eeprom)[1]		
Parameter	**Description**	**Return Value**
addr_eeprom	Logical address in the EEPROM space where data is to be read	Value; the 8-bit data is returned to the application code
		Error; the given address is out of available EEPROM memory space

[1] This function reads data from the EEPROM.

Table 3. Erase EEPROM Function Description

erase_eeprom()[1]	
Parameter	**Return Value**
Not applicable	No error; erase was successful
	Error; the flash controller is busy and cannot perform the erase action

[1] This function erases the EEPROM memory space. Note that all the data is lost if this function is called.

Table 4. Write Flash Function Description

write_flash(uint32_t addr, uint32_t lower_data, uint32_t upper_data)[1]		
Parameter	**Description**	**Return Value**
addr	The address in the flash memory space allotted for the EEPROM emulation	No error; write was successful
lower_data	The lower 32 bits of the double word	Error; the given address is out of the available EEPROM memory space
upper_data	The highest 32 bits of the double word	

[1] This function receives the translated EEPROM address and data from the write_eeprom() function and issues a write command to the flash controller.

Table 5. Erase Flash Function Description

erase_flash(uint8_t PAGE)[1]	
Parameter	**Return Value**
Page; the page number of the allotted flash memory space	No error; page wise erase was successful Error; the given page value is out of the allotted flash memory space

[1] This function performs a page wise erase in the allotted flash memory space.

Table 6. Read Flash Function Description

read_flash(uint32_t addr)[1]		
Parameter	**Description**	**Return Value**
addr	The address in the flash memory space allotted for EEPROM emulation	Read data; the 64-bit data is returned for masking and is sent as the return value to the read_eeprom() function Error; the translated address is out of the allotted flash memory space

[1] This function receives the translated EEPROM address from the read_eeprom() function and issues a read command to the flash controller.