AN11008

Flash based non-volatile storage

Rev. 1 — 5 January 2011

Revision history

Rev	Date	Description
1	20110105	Initial version.

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The NXP LPC1100, LPC1300, LPC1700 and LPC2000 family of ARM7 and ARM Cortex microcontrollers provide on-chip flash memory for storage of firmware. A frequent requirement for embedded systems is the storage of variables whose values need to persist through resets of the system. For microcontrollers without on-chip EEPROM there are two basic options. One is to attach an external EEPROM device, perhaps using I^2C or SPI. The second is to use the on-chip flash memory for variable storage.

Attaching an external device requires additional integration work as well as increasing hardware production costs and board space. Using flash memory is problematic as flash memory is not arranged in an optimal way for storage of small pieces of changing data.

There are three key issues with using flash memory for storage of variables:

- Erasing memory is on a per-sector basis
- Flash memory lifespan
- Reprogramming locations

This application note describes a simple solution to the problem of using flash memory for variable storage, which takes into account erasing flash memory in a way that avoids data loss and minimizes the effect of flash memory lifespan.

2. Storing variables in flash memory

When a flash memory location is erased its value becomes 0xFF. Each bit can then be programmed to logic 0 but it cannot be returned to logic 1 without erasing the entire sector. Subject to some limitations this allows multiple writes to a single flash sector to add data to it, without needing to erase the entire sector each time.

Each location in flash has a finite lifespan, rated in terms of the number of programming and erase cycles. For the Microcontrollers covered by this application note it is a minimum of 10,000 programming and erase cycles.

To continually erase a sector and program the same location each time a variable changed would result in continual use of the same location until the point of failure, while leaving thousands of other locations in the same sector unused.

The solution presented in this application note is to make use of all the locations in a single sector until it is full, then reorganize the memory usage so the process can start again.
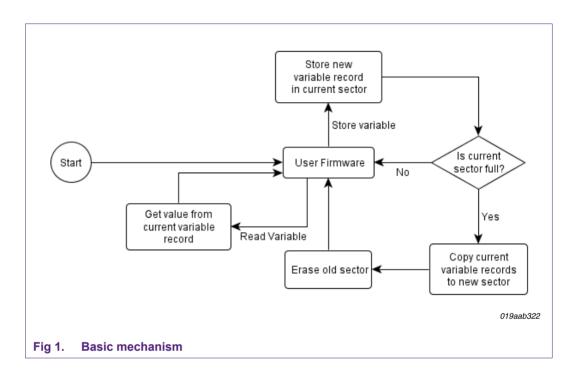
2.1 Basic mechanism

Initially flash sector A is empty (i.e., erased). When a variable needs to be stored a variable record is programmed into sector A. Later when the variable needs to be updated a new record is programmed into sector A and the old record is ignored.

Eventually sector A will become full, mostly with old records that are no longer used. On the next variable write when there isn't space in sector A, a second sector, sector B is employed.

All the current records in sector A are copied to sector B and sector A is then erased.

When a variable needs to be updated, the update goes into sector B, which is the sector currently in use. Once sector B has been filled the current records are copied to sector A and the process repeats.

AN11008

© NXP B.V. 2011. All rights reserved.

Application note **Rev. 1 — 5 January 2011** **3 of 12**

By swapping between two sectors of the same size it can be ensured that current variable values are always stored somewhere in flash memory while erasing takes place. This helps to ensure that if a reset or power failure occurs; the risk of data loss is minimized.

019aab322

Fig 1.   Basic mechanism

2.2 Variable records

The following table describes the structure of a variable record.

Table 1.   Variable record structure

Field	Size in Bytes	Description
Flags	1	0xFF = no entry, 0xAA = valid data
ID	2	A unique identifier for the variable
Data	N	The data stored in native-endianess, maximum of 12 bytes
Padding	12 – N	Always set to 0x00
Checksum	1	A 2's compliment checksum of the ID, data and padding

Each record is always a multiple of 16 bytes due to a system called Error Correction Code (ECC) which can be found on some NXP microcontrollers. ECC allows one write per 16 consecutive bytes of flash memory.

If needed the record size could be increased, which would allow larger variables to be used at the expense of the number of variables that can be stored, however it must always be a multiple of 16 bytes. It is also possible to store more complex data types, such as structures, in a variable record.

AN11008

Application note   **Rev. 1 — 5 January 2011**   **4 of 12**

The flags field indicates the status of the record. Erased memory has the effect of setting the flags to 0xFF which indicates no entry. This field takes advantage of the fact that it is possible to continually reprogram a location in flash memory without erasing, providing that each successive write only changes bits from logic 1 to logic 0. Therefore the lifecycle of a record is: no entry to valid data.

Because multiple variables are supported, it must be possible to identify the variable that a record contains. Therefore a unique identifier is used. For example if a variable called Counter is used, the ID might be defined using the C preprocessor:

```
1    #define COUNTERID 1
```

The same identifier must be used for a specific variable throughout the operation of the firmware.

The checksum is calculated by adding together the ID, data and padding using an eight byte variable, then subtracting from 0x100.

2.3 Sector organization

The start of both sectors used contains a single 48 byte sector record:

Table 2.   Sector record

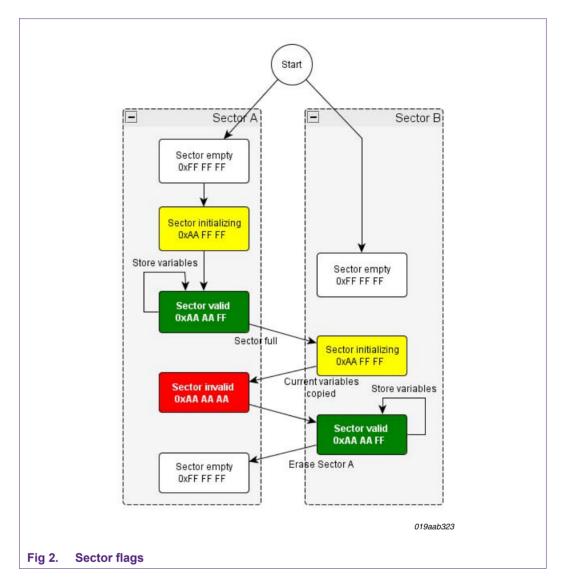| Field	Size in Bytes	Description
Flags 1	1	0xFF = sector empty, 0xAA = sector not empty
Padding	15	Reserved. Always set to 0x00
Flags 2	1	0xFF = sector empty or initializing, 0xAA = sector valid or invalid
Padding	15	Reserved. Always set to 0x00
Flags 3	1	0xFF = sector empty, initializing or valid, 0xAA = sector invalid
Padding	15	Reserved. Always set to 0x00

The three flags fields indicate the status of the sector. An erased sector will have the effect of setting all the flags to 0xFF, which indicates the sector is empty. These fields take advantage of the fact that it is possible to continually reprogram a location in flash memory without erasing, providing that each successive write only changes bits from logic 1 to logic 0. The lifecycle of a sector is: empty to initializing to valid to invalid.

The flags are spaced 16 bytes apart. This is due to the ECC functionality previously mentioned, that is featured on some microcontrollers. By spacing the flags apart, any update to the flags results in a single write per 16 consecutive bytes.

The initializing state for a sector indicates that the sector is in the process of being filled with current variable records from the old sector. Once the copying has finished the sector will be marked as valid. When the sector has been filled and the current variable records have been copied to the new sector, it will be marked as invalid.

In the following diagram the flags for a sector have been grouped into 24-bit values with Flags 1 occupying the most significant byte, and Flags 3 the least significant byte.

Fig 2.   Sector flags

2.4 Data access

To find the current value of a variable it is necessary to search through the current flash sector to find the last written valid record for the variable. If the value of the variable has changed then there will be several old records for the variable, which will need to be skipped. In the worst case the sector will be almost full and the variable record will be located at the end of the sector.

To speed up variable access a RAM-based lookup table is used. Every time a variable record is written an offset into the flash sector is stored in a table along with the variable's ID. To read the value of a variable the lookup table is consulted and then a direct access to the variable record is made.

Similarly the next free location in the current sector is also stored in RAM, increasing the speed at which new variable records can be written.

Each entry in the RAM-based lookup table occupies eight bytes. As supplied the library supports a maximum of 100 variables (see #define MAX_VARIABLES), resulting in a

AN11008

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2011. All rights reserved.

Application note

Rev. 1 — 5 January 2011

6 of 12

lookup table size of 800 bytes. If RAM usage is critical the RAM-based lookup table could be easily removed at the expense of slower non-volatile variable access.

2.5 Recovery after reset

The most important aspect of storing variables in flash memory is the ability to retain values even if the microcontroller is reset or power is interrupted. Even if power is interrupted while variables are being copied from one sector to another the system must be able to recover without data loss.

There is protection for this on two levels – variables level and sector level.

2.5.1 Variable protection

When the value for a new variable is stored in flash, the variable record has a flags field and a checksum field that ensure only complete and correctly written variable records are used. When reading the current value of a variable the flags must be set to 0xAA and the checksum must be correct.

After writing the variable record flash memory is read back to compare the contents and make sure the write was successful. If the write failed for some reason then the application layer is informed.

If the microcontroller is reset during the write of a variable record then after the next reset when the RAM-based lookup table is reconstructed, partially written or invalid records will be ignored. This will have the effect of using the last successfully written value for the variable.

2.5.2 Sector protection

During the switch between sectors the microcontroller could be interrupted at any point. After the following reset the microcontroller needs to continue where it left off switching sectors. To achieve this, a decision table is used. The flags of both sectors are examined to determine the current state of the system and what to do next. This takes place automatically during initialization. Note that not all states are possible during normal operation, however there has to be an action for every combination of sector flags.

In Table 3, the flags for a sector have been grouped into 24-bit values with Flags 1 occupying the most significant byte, and Flags 3 the least significant byte.

Table 3.   Sector decision table

Sector A Flags	Sector B Flags	Action
Invalid	X (don't care)	Erase sector A
X (don't care)	Invalid	Erase sector B
0xFFFFFF	0xFFFFFF	Mark sector A as valid (flags = 0xAAAAFF)
0xFFFFFF	0xAAFFFF	Mark sector B as valid (flags = 0xAAAAFF)
0xFFFFFF	0xAAAAFF	No action
0xFFFFFF	0xAAAAAA	Treat sector B as full and swap to sector A
0xAAFFFF	0xFFFFFF	Mark sector A as valid (flags = 0xAAAAFF)
0xAAFFFF	0xAAFFFF	Erase sector B. Mark sector A as valid (flags = 0xAAAAFF)

AN11008

Application note **Rev. 1 — 5 January 2011** **7 of 12**

Sector A Flags	Sector B Flags	Action
0xAAFFFF	0xAAAAFF	Erase sector A. Treat sector B as full and swap to sector A
0xAAFFFF	0xAAAAAA	Mark sector A as valid (flags = 0xAAAAFF). Erase sector B
0xAAAAFF	0xFFFFFF	No action
0xAAAAFF	0xAAFFFF	Erase sector B. Treat sector A as full and swap to sector B
0xAAAAFF	0xAAAAFF	Erase sector B
0xAAAAFF	0xAAAAAA	Erase sector B
0xAAAAAA	0xFFFFFF	Treat sector A as full and swap to sector B
0xAAAAAA	0xAAFFFF	Mark sector B as valid (flags = 0xAAAAFF). Erase sector A
0xAAAAAA	0xAAAAFF	Erase sector A
0xAAAAAA	0xAAAAAA	Erase sectors A and B. Mark sector A as valid (flags = 0xAAAAFF)

2.5.3 Code Read Protection

All ARM-based microcontrollers from NXP feature Code Read Protection (CRP), which is a method of securing the device by programming special values into a predefined memory location.

The flash-based system described in this application note can function when the CRP is disabled or when it is set to CRP level one. However it will not function if the CRP is set to level two or three.

3. Application Programmer Interface

This section describes the interface to the library implementing the system of storing variables in flash memory.

3.1 Functions

3.1.1 NVOL_Init

Initializes the non-volatile library. This function must be called before any other functions. Returns TRUE for success and FALSE for an error. An error can occur if the flash memory cannot be placed into a known state.

3.1.2 NVOL_SetVariable

Stores a value for a variable in flash memory. Passed is an ID unique to that variable along with a pointer to the variable and the size of the variable in bytes. This generic prototype allows for variables of any size and type to be stored. Returns TRUE for success and FALSE for an error.

3.1.3 NVOL_GetVariable

Gets the value of a variable previously stored in flash memory. Passed is an ID unique to that variable along with a location for the value to be stored and the size of the variable. This generic prototype allows for variables of any size and type to be read. Returns TRUE for success and FALSE for an error.

3.2 Configuration

Configuration options are set using the #define statements in flash_nvol.c.

3.2.1 Sector descriptions

The sector start addresses, numbers and sizes are set using SECTOR1_STARTADDR, SECTOR2_STARTADDR, SECTOR1_NUM, SECTOR2_NUM and SECTOR_SIZE. The sector number must match the numbering scheme given in the flash memory section of the NXP User Manual for the device.

3.2.2 CPU clock

The frequency of the CPU clock in kHz is set using CPU_CLK. This is needed by the IAP routines for timing.

3.2.3 Interrupts

Different compilers use different non-standard ways of enabling and disabling interrupts. The macros ENABLEIRQ and DISABLEIRQ must be set to the appropriate statements or function calls for the compiler you are using.

3.2.4 IAP access point

The value of IAP_LOCATION must be set to the memory location for IAP routines. This address can be found in the NXP User Manual of the device.

3.2.5 Number of variables and size

The maximum size of a variable in bytes is set using MAX_VARIABLE_SIZE. This must always be a multiple of 16 minus four (e.g., 12, 28, 44). A larger size results in less efficient usage of flash memory.

The maximum number of variables is set using MAX_VARIABLES. This has to be small enough to ensure that all variables can fit into a single flash sector. Note that 48 bytes are always allocated at the start of each sector for sector management. Also four bytes are needed per variable record. The smaller this number is, the less RAM is needed for the lookup table.

4. Limitations

There are two key limitations with the system. Firstly the flash lifespan limits the number of times that a variable can be written. Secondly there will be a delay in writing variables to flash memory.

4.1 Flash lifespan

The flash memory in the microcontrollers covered by this application note is rated at a minimum of 10,000 program and erase cycles.

If there is a single 12-byte variable and the flash sectors are 4096 bytes in size, then the variable can be written (4096 – 16) / 12 = 340 times before the sector is full. At that point the variable will move to the other sector where it can be written an additional 340 times.

The result is that the variable can be written 680 times for a single erase in each sector.

The lifespan is therefore a minimum of 10,000 x 680 = 6.8 million variable writes.

Now consider the worst case for sectors 4096 bytes in size. It would be 340 variables of 12-bytes in size. Every write of a value would cause the flash sectors to be swapped,

AN11008

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2011. All rights reserved.

Application note

Rev. 1 — 5 January 2011

9 of 12

resulting in one erase cycle of a specific sector every two variable writes. The lifespan would therefore be a minimum of 20,000 variable writes.

Note that the library only writes a variable if the value has changed. Consecutive writes with the same value will have no effect.

4.2 Write delays

Each time a new value for a variable is written it has to be programmed into flash memory. The speed of this operation is limited by the flash programming and erase speed.

The worst case write time for a single variable when the current sector is not full is approximately 1.05ms.

If the sector is full then the write will cause a swap of the two sectors used in the system. The worst case time for the entire operation will be approximately 105ms + 1.05ms per variable + 1.05ms for the current variable. For example if there are 20 variables then the approximate worst case time will be 105 + (1.05 x 20) + 1.05 = 127.05ms.

A possible optimization that could be implemented is to add a function that allows the application layer to determine how close the current sector is to being full and then manually perform a swap of the sectors. This would avoid unexpectedly long write delays at undesirable moments by ensuring that the sector swapping always takes place during idle periods.

4.3 IAP RAM usage

Calls to erase and program the flash memory using IAP require RAM. The location and amount of RAM used is defined in the NXP User Manual for the device, but is typically 32 bytes at the top of the RAM space and a maximum of 128 bytes on the user stack.

AN11008

Application note **Rev. 1 — 5 January 2011** **10 of 12**

5. Legal information

5.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

5.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

5.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

AN11008

All information provided in this document is subject to legal disclaimers.

© NXP B.V. 2011. All rights reserved.

Application note

Rev. 1 — 5 January 2011

11 of 12

6.   Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.