# AN14746

## EEPROM Emulation for the KW45B41Z and K32W148

Rev. 1.0 — 18 July 2025 Application note

**Document information**

| Information | Content |
|---|---|
| Keywords | AN14746, EEPROM emulation, KW45B41Z, K32W148, flash memory, flash endurance, data retention |
| Abstract | This document describes the process for the EEPROM emulation for the KW45B41Z and K32W148. |

# 1 Introduction

The KW45B41Z and K32W148 microcontrollers contain different types of memories, flash memory, tightly coupled memory (TCM), static random-access memory (SRAM), and read-only memory (ROM).

The flash memory can be used to emulate an electrically erasable programmable read-only memory (EEPROM). The KW45B41Z and K32W148 microcontrollers embed up to 1024 kB of flash which the flash memory controller (FMC) controls. The FMC manages the accesses that the bus controllers of the system perform to the flash memory.

This document describes the process for the EEPROM emulation for the KW45B41Z and K32W148.

# 2 EEPROM vs flash

Before running the emulation, it is important to know about EEPROM characteristics, and which features of the KW45B41Z and K32W148 are used to achieve it.

An EEPROM is a non-volatile memory that has a good endurance and retention compared with other technologies, it provides a reliable place to store important information. When an EEPROM cannot be used, flash memory is normally used, since most microcontrollers have this type of memory embedded. Flash memory is an advanced EEPROM, which provides lower write and erase times, and improves power consumption. However, you cannot write and erase the flash memory many times, as the EEPROM. Therefore, some processes are necessary to achieve EEPROM emulation.

The main characteristics of an EEPROM are:

- Write: one of the advantages of EEPROM is that erase is not needed before a write operation.
- Erase: It can be executed in sizes of bytes, or full pages, with a single command.
- Read: It can be performed to bytes or full pages.
- Endurance: one of the main advantages of EEPROM, can go up to four million cycles.
- Retention: EEPROM can provide five to ten times more retention than a regular flash memory.

Table 1 shows the flash reliability specifications for the KW45B41Z and K32W148. For more details, see the *KW45 Product Family Data Sheet* (document KW45) and *K32W14x Product Family Data Sheet* (document K32W1480).

**Table 1. Flash reliability specifications**

| Description | Minimum | Typical | Unit |
|---|---|---|---|
| Data retention after up to 1k cycles | 20 | 100 | years |
| Data retention after up to 10k cycles | 10 | 50 | years |
| Data retention after up to 100k cycles | 5 | 50 | years |
| Sector cycling endurance | 10k | 500k | cycles |
| Sector cycling endurance for 256 kB per array block | 100k | 500k | cycles |

Table 2 shows the flash timing specifications for the KW45B41Z and K32W148. For more details, see the product Data sheet.

**Table 2. Flash timing specifications**

| Description | Typical | Maximum | Unit |
|---|---|---|---|
| Program phrase execution time (16 bytes) | 135 | 375 | µs |
| Program page execution time (128 bytes) | 450 | 1000 | µs |

**Table 2. Flash timing specifications***...continued*

| Description | Typical | Maximum | Unit |
|---|---|---|---|
| Erase sector execution time (8 kB) | 2 | 22 | ms |
| Mass-erase execution time (1024 kB) | - | 2800 | ms |
| Read phrase execution time (16 bytes) | - | 3.8 | µs |
| Read page execution time (128 bytes) | - | 4.4 | µs |
| Read sector execution time (8 kB) | - | 50 | µs |

To achieve the same performance as on a real EEPROM, some work must be done on the write/erase, endurance, and retention features of the flash memory.

# 3 EEPROM emulation techniques

You can use many techniques to perform an EEPROM emulation with a flash memory. The couple of techniques are analyzed, remarking the main differences, and relating them to the available flash characteristics.

As EEPROM allows writing operations with no previous erase, but flash memory does not allow write operation. It is necessary to use a minimum of two memory blocks exclusively for this task. The size of these blocks can be selected as the minimum size of memory that can be erased. For the flash of KW45B41Z and K32W148, it is a sector of 8 kB.

To ensure if a sector does not have enough space to store more values or update one of them, two memory blocks are needed. It uses another sector to copy the latest values and continue storing new data. This mechanism is known as block swap, and it is used in all EEPROM emulations with flash.

Some emulation techniques use different headers to identify the sector status and validate the information stored in the sector. Others use only one header for the sector, and use sub-headers for every record.

## 3.1 Using different sector headers

In this EEPROM emulation technique, the software must define an active and an alternate sector, before starting to store data. For this purpose, some sector states are defined: copy, active, valid, and erasing.

The emulation software uses these states to know the status of the sector. The states can be defined as follows:

- Valid: the sector is validated to store the information.
- Active: the sector is in operation and allows data writing.
- Copy: the sector is full and ready to copy the latest values to the alternate sector.
- Erasing: the sector is ready to be erased.
- Erased: the sector can be validated and marked as active to store data.

The sector structure after following this technique is depicted in Table 3.

**Table 3. Memory sector using different headers**

| | | | | | |
|---|---|---|---|---|---|
| **Header (Active)** | | | | | |
| **Header (Valid)** | | | | | |
| **Header (Erasing)** | | | | | |
| **Header (Copy)** | | | | | |
| Address | Cycle redundancy check (CRC) | Data | Data | Data | Data |

**Table 3. Memory sector using different headers***...continued*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Header (Active)** | | | | | | | |
| **Header (Valid)** | | | | | | | |
| **Header (Erasing)** | | | | | | | |
| **Header (Copy)** | | | | | | | |
| Address | | CRC | | 0x0A | 0x0A | 0x0A | 0x3C |
| Address | | CRC | | 0x75 | 0x20 | 0x61 | 0x72 |
| Address | | CRC | | 0x61 | 0x64 | 0x79 | 0x20 |
| Address | | CRC | | 0x69 | 0x6F | 0x6E | 0x3C |
| Address | | CRC | | 0x72 | 0x6F | 0x67 | 0x72 |
| Address | | CRC | | 0x62 | 0x5F | 0x35 | 0x2E |
| 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF |
| 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF | 0xFF |

The structure of the information inside every sector can be selected according to the application. Table 3 represents 32-bit data with cyclic redundancy check and a 16-bit address for variables.

### 3.1.1 Storing information

When the emulated EEPROM receives a request to store the data, the application performs the following steps:

1. Define the address for the variable.
2. Calculate the CRC according to the data to be stored.
3. Program address, CRC, and 32-bit data value in a 64-bit writing operation.

The application follows these steps every time new data must be stored. If there is an update for one of the previously written variables, a new writing must be performed with the same address as its replacement. When a swap or a read is performed on a variable, the software takes the latest value for the variable. This value is closer to the end of the sector.

### 3.1.2 Reading information

To read the data, the application performs the following steps:

1. Find the latest value stored in the sector according to the address.
2. Once the address is found, the software calculates the CRC and validates it with the stored address.
3. The data is available for processing.

## 3.2 Using reprogramming functionality

This emulation technique takes advantage of the flash memory property to allow reprogramming without generating an ECC error. The sector has a header with different values that define its status, and is organized with the following information:

- Sector or block status: This field can have different values according to the status of the sector. In this example, the following are the status of the sectors:
  - Active
  - Erased
  - Verified
  - Copy

- Record status: Defines the status of the record as valid or invalid.
- Record ID: The software uses this ID to find the correct data for reading.
- Record size: The data size can be different for every ID, from one byte to as many as the application requires.
- Data: This is the data to be stored.

When an EEPROM emulation uses this technique, the software creates a sector structure like the one showed in Table 4.

**Table 4. Flash timing specifications**

| Description | 4 bytes | | 4 bytes | | 4 bytes | | 4 bytes | |
|---|---|---|---|---|---|---|---|---|
| Record 1 | Block Status (valid) | | | | | | | |
| | Record 1 Status (valid) | | | | | | | |
| | ID Record 1 | | Record 1 Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| Record 2 | Record 2 Status (invalid) | | | | | | | |
| | ID Record 2 | | Record 2 Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| Record 3 | Record 3 Status (valid) | | | | | | | |
| | ID Record 3 | | Record 3 Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| Record 2 | Record 2 Status (valid) | | | | | | | |
| | ID Record 2 | | Record 2 Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| … | … | | | | | | | |
| Record N–1 | Record N–1 Status | | | | | | | |
| | ID Record N–1 | | Record N–1 Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| Record N | Record N Status | | | | | | | |
| | ID Record N | | Record N Data Size | | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| | Data | Data | Data | Data | Data | Data | Data | Data |
| Unused flash | ... | | | | | | | |
| | ... | | | | | | | |

### 3.2.1 Storing information

Before storing data to any block in the EEPROM emulation, the software uses the status in the sector header to check the active block. In this example, the status values are:

- Erased: The first state after erase is performed on a block.
- Verified: Software has verified that the block is erased and ready to be active.
- Active: The software can read and write information in this block.
- Copy: It indicates the software that the block or sector is used as an alternate block, and it is ready for a swap process.

The status values must be selected so that the ECC rules of the flash memory are obeyed.

Once the software has confirmed the active block, the programming of the data is ready. As the programming is sequential and the size of data varies, the software must calculate if there is enough space for storing the data. If the space is not enough to store the data, the software performs a swap process to copy the latest values to an alternate block.

In general, the storing process follows these steps:

1. Mark the record status as invalid.
2. Write the record ID and data size fields.
3. Write the data to be stored.
4. Mark the record status as valid.

If the current record is written or updated correctly, the valid and invalid record statuses allow the software to detect and mark the record status. Verify if a record can be copied to an alternate block when a swap process is performed.

### 3.2.2 Reading records

When a reading is performed to the emulated EEPROM, it can only be performed in the block marked as active. The software must follow the next steps:

1. To verify that the record ID matches with the requested ID, check the record ID and read the record status. If the record status is valid and the ID matches, the software makes note of this record.
2. To calculate the address of the next record, the software reads the size of the data. It continues searching for the same ID in the rest of the EEPROM as it is possible to have many records with the same ID.
3. The last valid record with the corresponding ID is returned as an answer for the request.

### 3.2.3 Swap operation

When erased memory is not enough in the active block to perform updates, the EEPROM emulation software copies the valid records from the active block to an alternate block to perform a block swap.

The software reads all the content of the active block, copies the latest record for each ID to the alternate block. The alternate block is then marked as the active block for further access.

The previous block is erased. To verify if the block is erased, the software checks that every memory location in the block reads back as all ones. Any block which is verified as erased can be used as the alternate block in a block swap. This verification process ensures that the software does not use a block, which is not erased fully as an alternate block.

The block status field is used in the same way as the record status field and cycles through the erased, verified, copy, and active states.

Figure 1 shows the steps of the swap process.

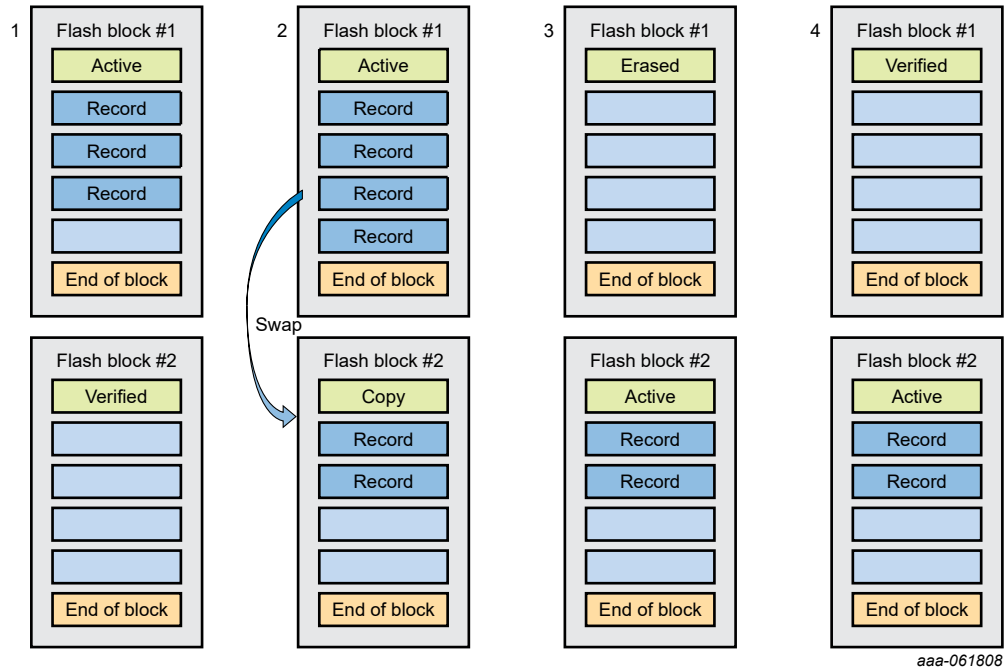


**Figure 1. Swap process between two blocks**

# 4 KW45/K32W1 EEPROM emulation implementation

The NXP offers an EEPROM emulation demo as one of the driver examples for the KW45B41Z and K32W148 software development kits (SDK). This example provides all the necessary application programming interfaces (API) to handle an EEPROM, allowing users to focus only on the application layer.

## 4.1 Sector structure

Figure 2 shows the structure for organizing the data records that the demo uses.

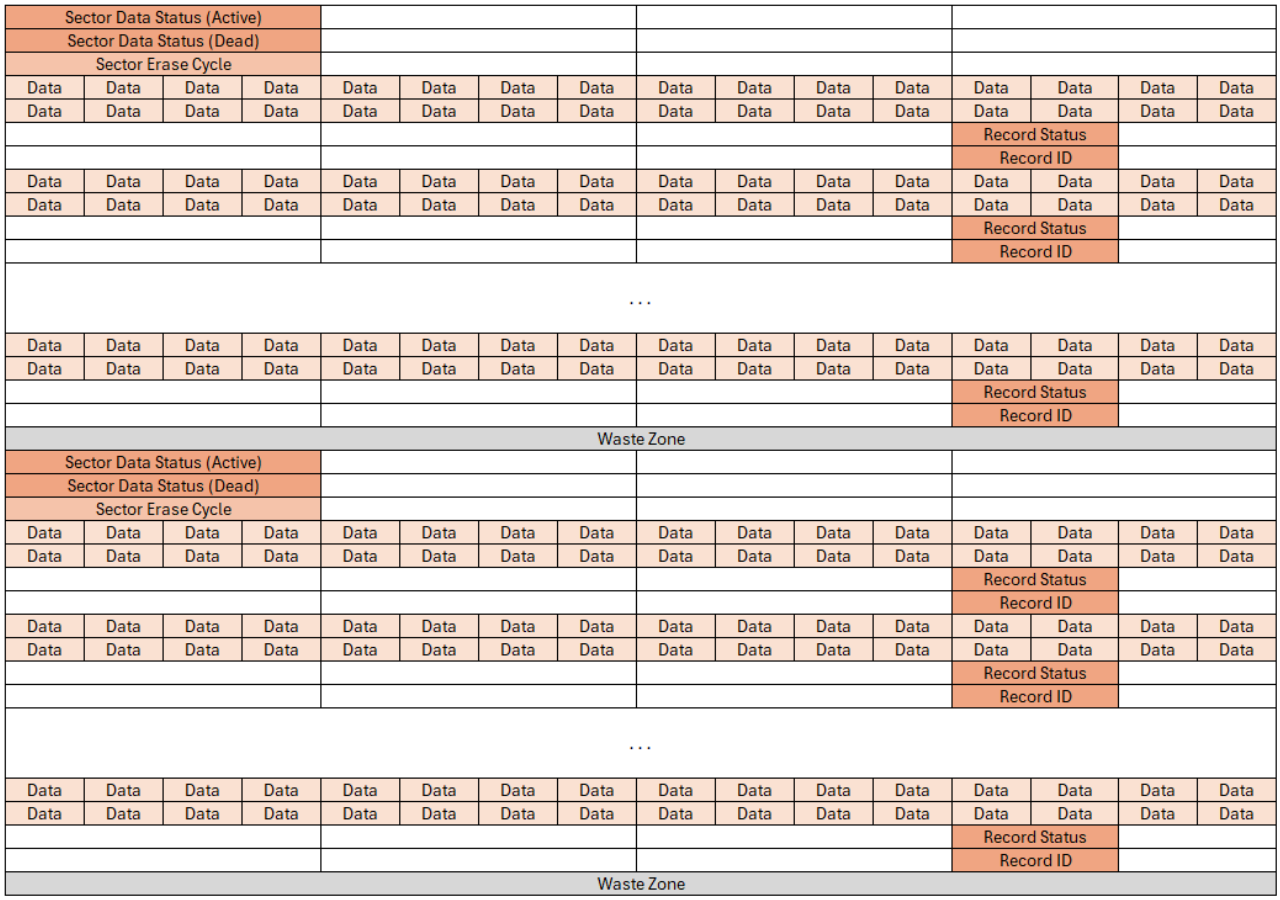


**Figure 2. Sector structure in EEPROM emulation demo**

Every sector has a status indicator to show whether it is active or dead, and the number of erase cycles that it has gone through. Every record has a constant size of 32 bytes, which is stored after the erase cycle, aligned to 16 bytes. Every record has a status indicator showing if it is valid or invalid, and the record ID.

As the size of the headers and records stored in a sector is not aligned with the total size of the sector, the waste zone must be available, If the size is16 bytes. The EEPROM drivers also consider this size when making the calculations to manipulate the flash.

## 4.2 Demo configuration

The demo in its default configuration emulates 1 kB of raw data. It uses 32-byte records. To accomplish this, it uses three active sectors and four ready sectors that can be used as alternate sectors when the active ones are full.

To change the size of the emulated EEPROM and the size of the data, modify the following macros defined in the app.h header file:

```
#define EEPROM_SIZE     1024
#define DATA_VALUE_SIZE 32
```

To change the number of extra active sectors and ready sectors, modify the following macros defined in the app.h header file:

```
#define EXTRA_READY_SECTORS      2
```

```
#define EXTRA_ACTIVE_SECTORS     2
#define ACTUAL_READY_SECTORS     2
```

Fix the sector size at 8 kB, as it is the minimum size that the KW45/K32W1 FMC can erase.

## 4.3 Running the demo

The demo application flow writes 27 records repeatedly in a loop until every active and alternate sector is erased three times. It reads back all the records and verifies their integrity.

After all writes and reads, if the read data corresponds to the previously written data, a success message is shown in the terminal as shown in Figure 3.

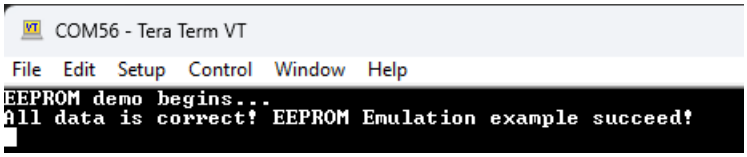


**Figure 3. Printed messages from EEPROM emulation demo**

If the flashing or erasing of a specific sector fails at any point, the demo is configured to retry two more times. If it fails to perform the desired operation, the whole sector is declared as dead, and is not used in the emulation anymore. If there is a failure, the application is also configured to retry the initialization and record writing routines five times. To configure both number of retries, modify the following macros defined in the app.h header file:

```
#define RETRY_MAX         0x02u
#define MAX_REPEAT_TIMES  0x05u
```

### 4.3.1 Writing records

Once the EEPROM is initialized with the configuration described in Section 4.2, it is simple to write on it. Write to EEPROM with the API called EE_WriteData, which is provided in fsl_eeprom_emulation.c. The arguments it receive are the EEPROM handle, the data ID, and the source buffer. An example of a written record with an ID of 0001 is shown in Figure 4. The indicator to show that the sector is active (0xFACF FACF) can also be observed.

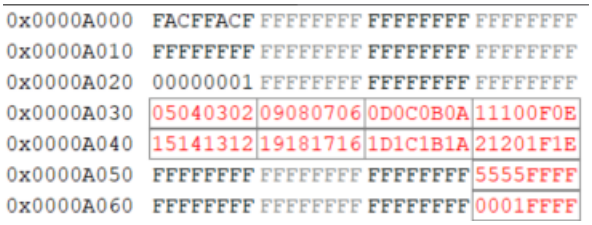


**Figure 4. Example of written record in EEPROM**

### 4.3.2 Reading

You can read a record that is written previously. An API called EE_ReadData is used to read the data, which provides the EEPROM handle, record ID, and a storage buffer as arguments. To retrieve a stored record, the application first checks the current active sector, starting from the end, as newer values are stored after the old ones. If the record is not found in this sector, then it checks in the second newest active sector, also starting from the end. When read record ID matches with the requested ID, the application validates the record status as valid and returns the address for the stored data.

### 4.3.3 Changing active sector

As the demo has three active sectors, when one of them is full, the application can start writing the new records in the next active sector if it is empty. Figure 5 shows a case in which the current active sector does not has enough space for writing another record, and the next sector, starting at 0xC000, is empty.

```
0x0000BFA0  FFFFFFFF FFFFFFFF FFFFFFFF 0013FFFF
0x0000BFB0  1C1B1A19 201F1E1D 24232221 28272625
0x0000BFC0  2C2B2A29 302F2E2D 34333231 38373635
0x0000BFD0  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000BFE0  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
0x0000BFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C000  FACFFACF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C030  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C040  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

**Figure 5. Current active sector full**

The application calculates the remaining space on the current active sector and when it detects a new record that does not fit, then writes it in the next sector, as shown in Figure 6.

```
0x0000BFD0  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000BFE0  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
0x0000BFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C000  FACFFACF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C030  1D1C1B1A 21201F1E 25242322 29282726
0x0000C040  2D2C2B2A 31302F2E 35343332 39383736
0x0000C050  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000C060  FFFFFFFF FFFFFFFF FFFFFFFF 0015FFFF
```

**Figure 6. Jump to next empty active sector**

### 4.3.4 Swapping sectors

Eventually, after many writes, all the active sectors become full. When the application detects that the next available sector is not an active one, it triggers the swap mechanism. This mechanism as described in Section 3.2.3 performs the following:

1. Copies the latest values of all records
2. Marks the new sector as active
3. Erases the oldest active sector to be used as an alternate sector later.

Figure 7 shows the end of the last active sector available and the start of the alternate sector next to it. After it reads the active and dead indicators, the application knows that the alternate sector is blank and alive. It can use this sector as the next active sector.

```
0x0000FFA0  FFFFFFFF FFFFFFFF FFFFFFFF 0003FFFF
0x0000FFB0  16151413 1A191817 1E1D1C1B 2221201F
0x0000FFC0  26252423 2A292827 2E2D2C2B 3231302F
0x0000FFD0  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000FFE0  FFFFFFFF FFFFFFFF FFFFFFFF 0004FFFF
0x0000FFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010000  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x00010030  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010040  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

**Figure 7. Jump to next empty active sector**

In the demo:

- All the records fit in a single sector
- There are multiple active sectors at the same time
- The application does not copy any records to the new sector
- The application sets the new sector as an active sector
- Erases the old active sector and updates the erase cycle counter, as shown in Figure 8

```
0x0000A000  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000A010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000A020  00000002 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000A030  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

**Figure 8. Erased old active sector and updated erase cycle counter**

To observe the copy of the swapping mechanism, the macro EXTRA_ACTIVE_SECTORS is set to zero, that is, the application has one active sector to work with. When the current active sector is full, the application copies the latest value of each record to the new sector. Then it erases the old active sector and sets the alternate sector as the new active sector. For more details, see Figure 9, Figure 10, and Figure 11.

```
0x0000BFA0  FFFFFFFF FFFFFFFF FFFFFFFF 0013FFFF
0x0000BFB0  1C1B1A19 201F1E1D 24232221 28272625
0x0000BFC0  2C2B2A29 302F2E2D 34333231 38373635
0x0000BFD0  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000BFE0  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
0x0000BFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C000  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C030  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C040  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

**Figure 9. The current active sector is full and next sector is blank but not active**

```
0x0000BFA0  FFFFFFFF FFFFFFFF FFFFFFFF 0013FFFF
0x0000BFB0  1C1B1A19 201F1E1D 24232221 28272625
0x0000BFC0  2C2B2A29 302F2E2D 34333231 38373635
0x0000BFD0  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000BFE0  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
0x0000BFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C000  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C030  1C1B1A19 201F1E1D 24232221 28272625
0x0000C040  2C2B2A29 302F2E2D 34333231 38373635
0x0000C050  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000C060  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
0x0000C070  1B1A1918 1F1E1D1C 23222120 27262524
0x0000C080  2B2A2928 2F2E2D2C 33323130 37363534
0x0000C090  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000C0A0  FFFFFFFF FFFFFFFF FFFFFFFF 0013FFFF
```

**Figure 10. The newest values of each record are copied to an alternate sector**

```
0x0000BFA0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000BFB0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000BFC0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000BFD0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000BFE0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000BFF0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C000  FACFFACF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C010  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C020  00000001 FFFFFFFF FFFFFFFF FFFFFFFF
0x0000C030  1C1B1A19 201F1E1D 24232221 28272625
0x0000C040  2C2B2A29 302F2E2D 34333231 38373635
0x0000C050  FFFFFFFF FFFFFFFF FFFFFFFF 5555FFFF
0x0000C060  FFFFFFFF FFFFFFFF FFFFFFFF 0014FFFF
```

**Figure 11. A new sector is marked as active while the old sector is erased**

# 5 Endurance

Endurance is an important metric to consider when implementing an EEPROM emulation. It determines the number of cycles for write and erase operations before storing the data incorrectly due to the lifetime of the flash memory.

When you use the minimum value for sector endurance (10k cycles) and the EEPROM emulation implementation uses seven sectors, this results in a theoretical endurance of 70k write-erase cycles. However, as each sector can fit 127 32-byte records, the application can handle 8.89M record writes before going out of spec.

$$Total\ Record\ Writes\ =\ floor\left(\frac{Sector\ Size - Sector\ Header\ Size}{Record\ Size + Record\ Header\ Size}\right) * Total\ Sectors * Sector\ Endurance$$

$$Total\ Record\ Writes\ =\ floor\left(\frac{8192 - 48}{32 + 32}\right) * 7 * 10000$$

$$Total\ Record\ Writes\ =\ 8890000$$

When you use the total number of allowed record writes and the minimum retention time for a sector (up to 10k cycles from Table 2), you can calculate that this EEPROM emulation can handle a record write about every 35 seconds for 10 years without failure.

$$Record\ Write\ Interval\ =\ \frac{Sector\ Retention\ Time(s)}{Number\ of\ Available\ Record\ Writes}$$

$$Record\ Write\ Interval\ =\ \frac{10 * 365 * 24 * 60 * 60}{8890000}$$

$$Record\ Write\ Interval\ =\ 35.47\ s/write$$

# 6 Abbreviations

Table 5 lists the acronyms used in this document.

**Table 5. Abbreviations**

| Acronym | Description |
|---|---|
| API | Application programming interface |
| CRC | Cycle redundancy check |
| EEPROM | Electrically erasable programmable read-only memory |
| FMC | Flash memory controller |
| ROM | Read-only memory |
| SDK | Software development kit |
| SRAM | Static random-access memory |
| TCM | Tightly coupled memory |

# 7 References

Table 6 lists the references used to supplement this document.

**Table 6. Related documentation/resources**

| Document | Link/how to access |
|---|---|
| KW45 Product Family Data Sheet (document KW45) | KW45 |
| K32W14x Product Family Data Sheet (document K32W1480) | K32W1480 |

# 8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 9 Revision history

Table 7 summarizes the revisions to this document.

**Table 7. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN14746 v.1.0 | 18 July 2025 | Initial public release |

# Legal information

## Definitions

**Draft** — A draft status on a document indicates that 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 in a draft version of a document and shall have no liability for the consequences of use of such information.

## 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. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

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 and its suppliers accept 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.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**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 competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**HTML publications** — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

**NXP B.V.** — NXP B.V. is not an operating company and it does not distribute or sell products.

## Trademarks

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

**NXP** — wordmark and logo are trademarks of NXP B.V.

## Tables



## Figures

# Contents