Full Efficient NVM Usage For MCU

Huseyin KARACALI[1*], Nevzat DONUM[2*], Efecan CEBEL[3*]

[1] TTTechAuto Turkey, Software Architect, huseyin.karacali@tttech-auto.com, 0000-0002-1433-4285
[2] TTTechAuto Turkey, Embedded Software Engineer, nevzat.donum@tttech-auto.com, 0000-0002-8293-8267
[3] TTTechAuto Turkey, Embedded Software Engineer, efecan.cebel@tttech-auto.com, 0000-0002-2027-0257
[*] Correspondence: efecan.cebel@tttech-auto.com

(First received January 16, 2023 and in final form March 23, 2023)

Reference: Full Efficient NVM Usage For MCU. The European Journal of Research and Development,3(1), 115-128.

Abstract

Non-volatile memory (NVM) is a type of embedded device memory that can retain stored data even when power is disconnected. Its significance is growing due to the increasing need for fast access and access to large volumes of data. Efficient usage of NVM is crucial to maximize its potential. Efficiency can be achieved by increasing the read/write speed as well as by using the available space efficiently. Moreover, supply chain problems that have emerged in recent years have once again demonstrated the importance of using NVMs efficiently. The subject of this study is the 100% efficient use of memory. A multi-layered structure has been designed to increase the usage efficiency of NVM. Algorithmically, the basic logic is to fill the NVM so that there are no garbage bytes left in one sector of the NVM. This algorithm is structured to provide 100% efficiency by increasing the number of write/read operations. In studies on NXP S32K148, it has been observed that NVM is used with full efficiency. The operating speeds of the high-level read/write functions have not changed at a level that will affect the system. In summary, even if the low-level read/write number changes, it does not disrupt the operation of the system. The developed NVM Management System does not have a power safe feature. It can be made more reliable with the power safe feature in new studies.

Keywords: NVM, MCU

1. Introduction

The proliferation of the Internet of Things (IoT) has necessitated the development of increasingly sophisticated embedded devices that require high performance and reliability. Non-Volatile Memory (NVM) is a fundamental component of such devices, serving as a repository for critical data and program code and facilitating fast and dependable system startup. The digital information age's explosive data generation and consumption have created a demand for durable, dependable, and high-performance storage systems, and NVM has emerged as a promising solution for embedded devices that can meet these criteria. As NVM becomes increasingly prevalent in modern computing systems, it is crucial to understand how to use it effectively. While NVM provides numerous advantages over traditional storage devices, such as faster access times, lower power consumption, and higher endurance, extracting the maximum value from it necessitates a distinct approach to software design and data management. This article outlines effective strategies for writing, deleting, and updating data in NVM. By optimizing NVM utilization and enhancing memory management, it is possible to extend NVM's lifespan, lower the risk of data loss, and improve the durability and reliability of stored data.

2. Materials

2.1. NVM

Non-volatile memory (NVM) is an essential computer memory that can store data even without power. The importance of NVM has been increasing in recent years, given the need to store and access large amounts of data quickly. In contrast to volatile memory like DRAM, which loses data when the power is off, NVM is designed to hold data even without power, making it suitable for persistent data storage in devices like embedded systems, mobile devices, and data centers. Different types of NVM, such as flash memory, phase-change memory, magnetic RAM, and resistive RAM, each have unique characteristics such as speed, durability, and power consumption, making them ideal for different applications. Maximizing the efficiency of NVM is critical to achieving its potential and enhancing the durability and reliability of stored data.

2.2. C Programming Language

C Programming Language is a that is widely used in the computer industry for developing efficient and reliable software. It was developed by Dennis Ritchie at Bell Labs in the 1970s as an extension of the B programming language. C has become one of the most popular programming languages due to its ability to produce efficient code and its simplicity, allowing developers to create low-level systems and high-level applications. C is a procedural language, which means that it follows a step-by-step approach to execute instructions, making it a structured language that is easy to read and understand. C is also a compiled language, which means that source code is compiled into machine code that can be executed directly by the computer's processor. C has a rich library of built-in functions and operators, making it suitable for a wide range of applications, including operating systems, embedded systems, graphics, and gaming. C has been used to develop many important software applications, including the Unix operating system and the Linux kernel. The language is also used in many academic settings to teach programming concepts and algorithms due to its simplicity and structured approach.

2.3. Microcontroller Unit

Microcontrollers (MCUs) are integrated circuits that combine a microprocessor with memory and input/output peripherals, all on a single chip. MCUs are designed for embedded systems and are often used in devices that require a degree of intelligence or automation, such as automotive electronics, household appliances, medical devices, and industrial machinery. The microprocessor core in an MCU typically has a reduced instruction set architecture (RISC) and is optimized for low power consumption, high performance, and low cost. The memory in an MCU is often non-volatile, such as flash memory, which allows the MCU to retain its program and data even when the power is turned off. The input/output peripherals on an MCU typically include analog-to-digital converters, digital-to-analog converters, timers, and communication interfaces such as UART, SPI, and I2C. These peripherals allow the MCU to interact with the outside world, such as reading sensor data, controlling motors, and communicating with other devices. MCUs are often programmed using C or assembly language, and the software is typically developed using an integrated development environment (IDE) that provides a text editor, compiler, debugger, and other tools for programming and testing the MCU. Because MCUs are often used in real-time systems that require rapid response to external events, the software must be carefully designed and optimized for performance, efficiency, and reliability.

3. Method

Data structures optimization and read-write frequency changes have been the basis of studies and file system projects to increase performance and efficiency in non-volatile memories. In the study of fully efficient non-volatile memory usage, the control in the writing process of the data and the indexed structure of this data were emphasized as a methodology.

This write control mechanism is provided by the index table in the last 2 sectors. This index table, which is a lookup table, maps the addresses of the data in memory, providing easy access if the data is to be used. The index table is first updated in case of any operation on the data.

In this section, the configurations required for fully efficient non-volatile memory usage and use cases such as data write, delete, and update are discussed.

3.1 Configuration

One of the first and most important parts of the implementation for this study is its configuration. The feature of being usable easily and on most platforms can be provided by this configuration method. Thanks to a header file named 'config.h', where parameters are specified according to the platform to be used, non-volatile memory is made ready to be used in full efficiency. This header file contains the following variables, respectively:

- flash_start_address: This variable represents the starting address of the non-volatile memory which may differ from MCU to another one where data can be stored. Any data to be saved is first written starting from this address. It must be typed in bytes.
- sector_size: This value, which can take different values on different platforms, refers the size of the smallest erasable or programmable block in memory. It must be typed in bytes.
- flash_end_address: This variable represents the last address in memory where data can be saved. In this study, although the last 3 sectors are not reserved for data storage, the last address must be assigned as the value in bytes.

Moreover, there are 2 more variables in this file that are automatically determined with the values entered above:

- allocated_sector_start_address: This variable specifies the starting address of the sector to be used in case the data to be written does not fit into the RAM during the data writing process. It is obtained by subtracting 3 sector sizes (sector_size) from the last address (flash_end_address) of the memory.
- index_table_start_address: It refers to the starting address of the table where the data in the memory and their addresses are kept. This variable is obtained by subtracting 2 sector sizes (sector_size) from the last address (flash_end_address) of the memory.

After editing the config.h file, the non-volatile memory (NVM) structure will be as follows.

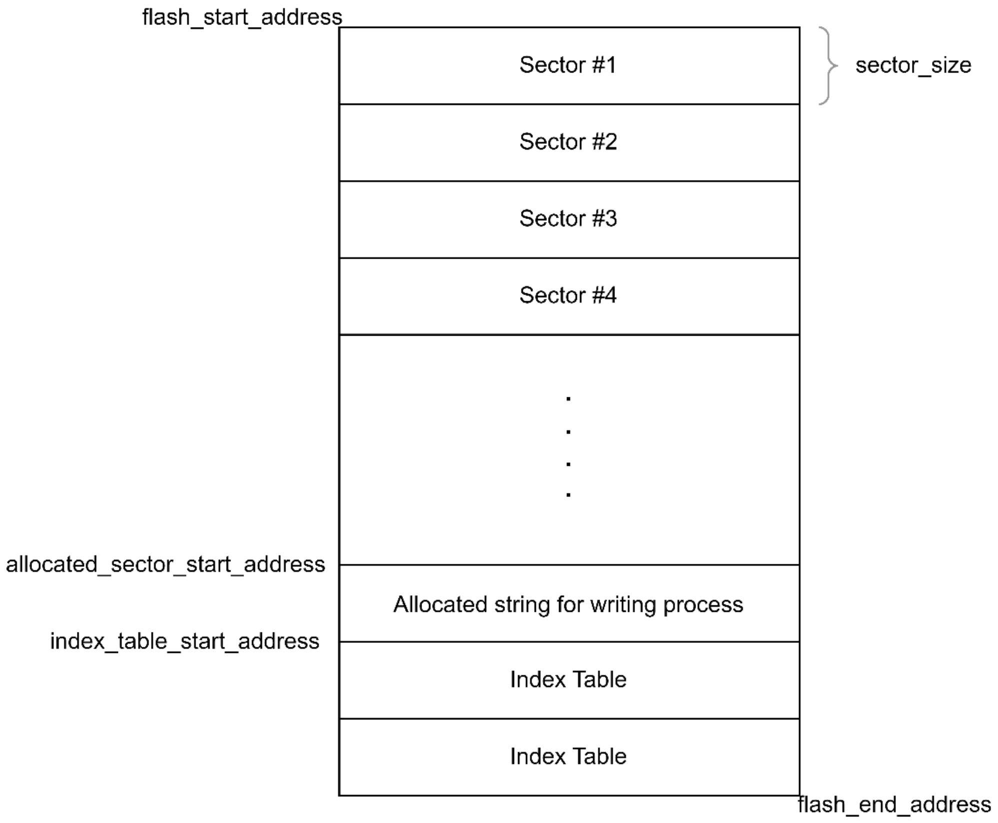


Figure 1: NVM Structure After the Configuration

3.2 Use Cases

The use cases in this study can be grouped under 3 main headings as data insertion, deletion, and updating.

3.2.1 Data Insertion

The data insertion scenario is the process of storing data of any size in memory according to MCU's endianness. The data to be written is written to a previously erased sector in an initialized memory. If there is no purpose to write it right after to the previous one when another data comes, its efficiency will be low.

In this study, firstly, the size of the incoming data to be written and the largest address kept in the index table are sum and it is checked whether it overflows into the allocated sector. If an overflow is to occur to this allocated sector, the system will completely stop writing before this data is written. If there is no overflow, it is checked whether the data is small enough to be written to RAM. Since the sector where the data is intended to be saved cannot be completely deleted due to the possibility of losing data, it is desirable to keep it in RAM temporarily if possible. Otherwise, it is saved to the sector allocated for write operation for later deletion.

After keeping the data in RAM or in the allocated sector, it goes to the largest address held in the index table and the data is written to the memory along its size, starting from this address. Immediately after, the index table is updated with the new data and the address of that data is stored. As the last step, if the allocated sector was used for temporary storage, the system clears this sector first, and if it was not used, the system directly ends the data insertion process successfully.

The following figure represents all these insertion processes as a flowchart.

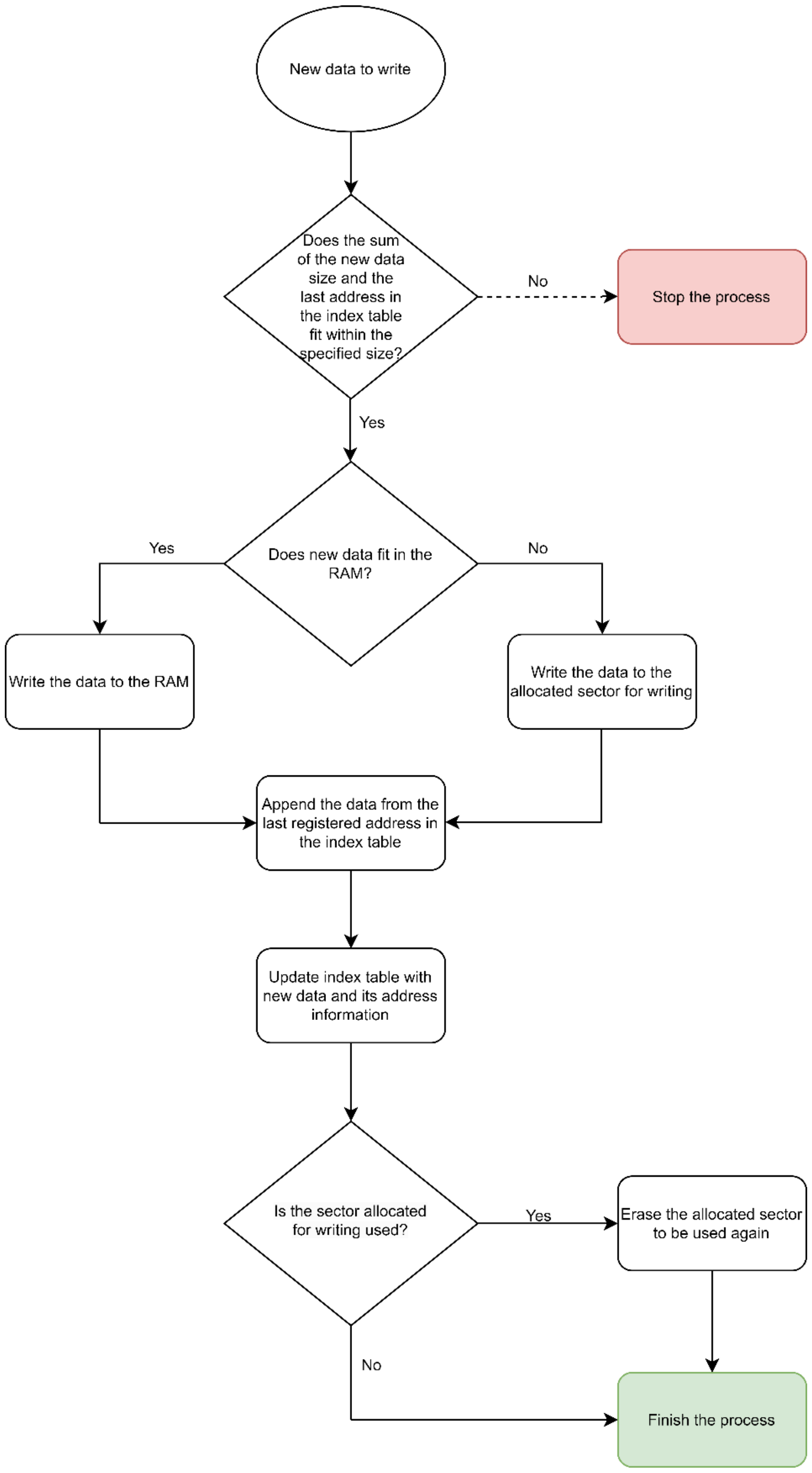

New data to write
Does the sum of the new data size and the last address in the index table fit within the specified size?
No
Stop the process
Yes
Does new data fit in the RAM?
Yes
No
Write the data to the RAM
Write the data to the allocated sector for writing
Append the data from the last registered address in the index table
Update index table with new data and its address information
Is the sector allocated for writing used?
Yes
Erase the allocated sector to be used again
No
Finish the process

Figure 2:Data Insertion Flowchart

3.2.2 Data Deletion

Data deletion is to remove a data existing in the memory, in other words, to write 0xFF values to the place where the values of the data are stored. There may not be any new data in the appropriate size to the address where this deleted data is located, or the new data may be written to a different address. Therefore, this portion remains unused and the efficiency of non-volatile memory decreases.

In the data deletion stages of this study, firstly, the address of the data from the index table of the deletion request is read. Thus, the data is deleted from the memory by writing 0xFF value throughout its size, starting from the address read. Immediately after deletion, all information about this data is also removed from the index table.

The part that will provide full efficiency comes after these processes. In order that there is no lost part between the values found before and after the deleted data, all values found after the deleted value are shifted by the size of the deleted data until the last data is shifted, and whereupon each shift, the index table is updated with new addresses recursively.

The following figure represents all these deletion processes as a flowchart.

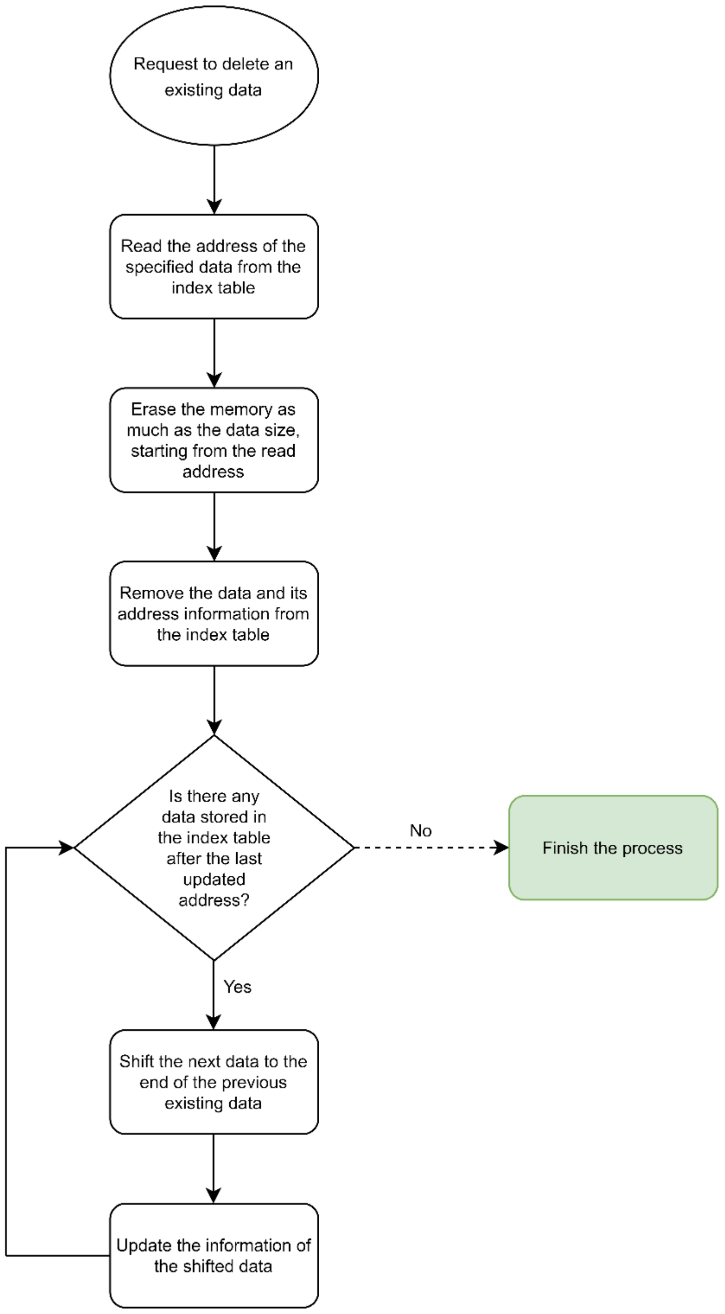

Request to delete an existing data
Read the address of the specified data from the index table
Erase the memory as much as the data size, starting from the read address
Remove the data and its address information from the index table
Is there any data stored in the index table after the last updated address?
No
Finish the process
Yes
Shift the next data to the end of the previous existing data
Update the information of the shifted data

Figure 3:Data Deletion Flowchart

3.2.3 Data Updating

The process of updating the value of a data is the process of replacing the values at the address of that data with new ones. In fact, it happens by first deleting those values and then writing new values to that address.

In this study, for the update process, first of all, the address of the data to be changed is read from the index table, which provides a quick access to that address. Since the size of the data to be updated will not change at run-time, the system goes to the address it reads and completes the process by writing a new one after deleting the value.

The following figure represents all these updating processes as a flowchart.

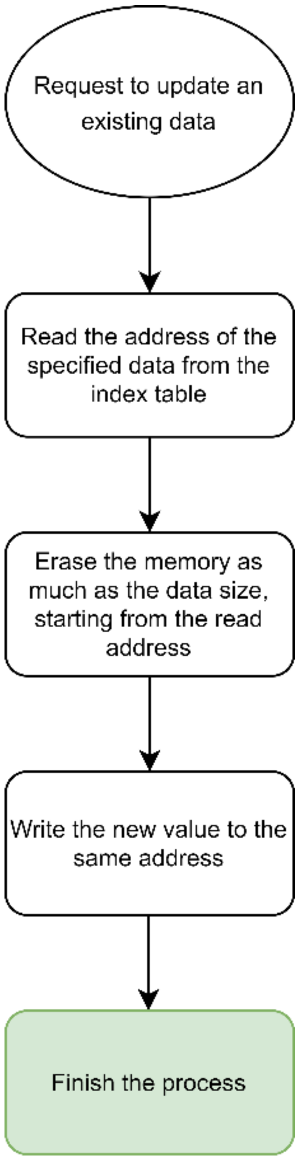

Request to update an existing data
Read the address of the specified data from the index table
Erase the memory as much as the data size, starting from the read address
Write the new value to the same address
Finish the process

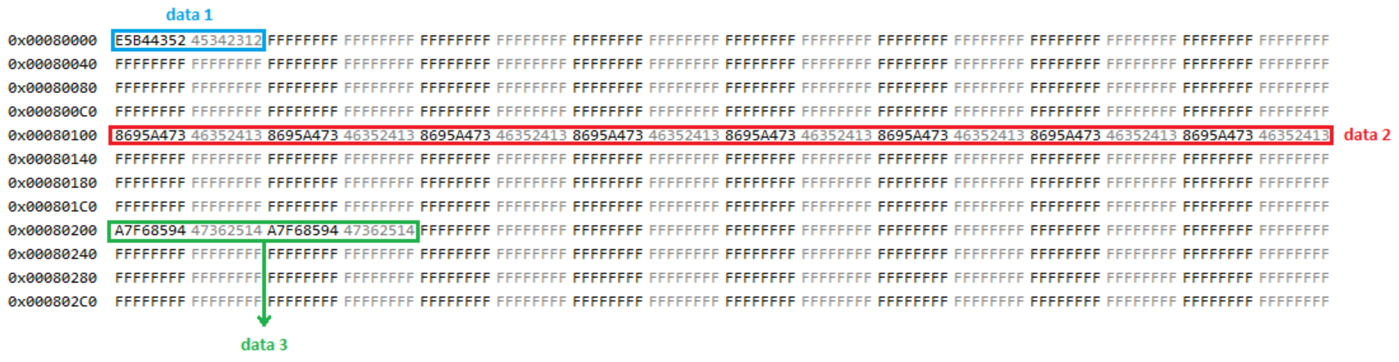
Figure 4:Data Updating Flowchart

4. Result

This section will present the results of study with the output from the fully efficient non-volatile memory (NVM) usage tests on the S32K148 test platform by exporting the memory browser. There are separate tests for insertion, deletion and updating operations. In these tests, the first data is selected as 8 bytes, the second one is 64 bytes, and the third one is 16 bytes. Also, each sector size equals to 128 bytes.

4.1 Data Insertion

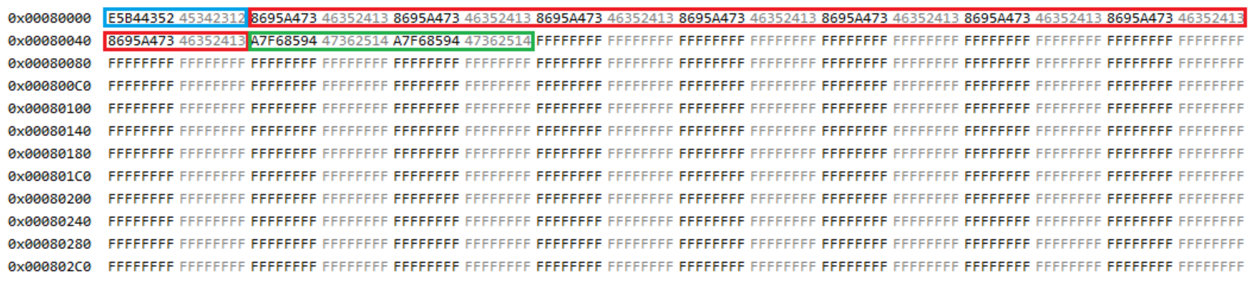
In the tests of adding data, firstly, an output was taken about how 3 different values are kept in memory without providing a fully efficient non-volatile memory usage structure. In this case, each data is written in different sectors and thus unused empty areas are formed between them as seen in Figure-5.



```
0x00080000  E5B44352 45342312 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080040  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100  8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413
0x00080140  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200  A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 5:Data Insertion Output Before the Full Efficient NVM Usage Study

On the other hand, judging by the results obtained after integrating the fully efficient memory usage structure, all data is appended to the previous one end as seen in Figure-6, thus avoiding lost areas.

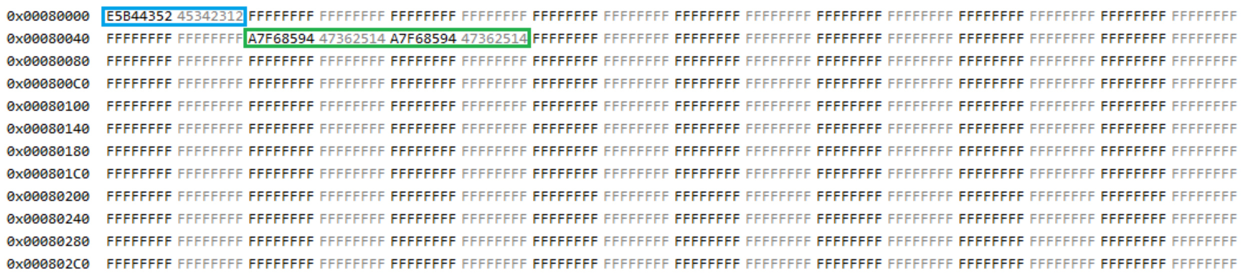


```
0x00080000  E5B44352 45342312 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413
0x00080040  8695A473 46352413 A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080140  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 6?Data Insertion Output After the Full Efficient NVM Usage Study

4.2 Data Deletion

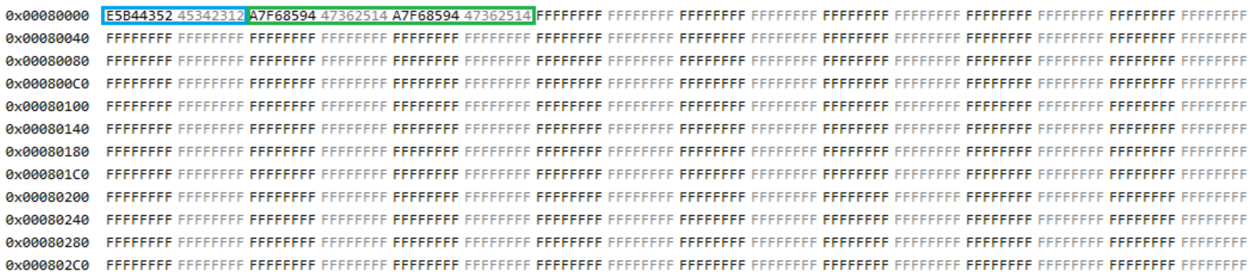
In the test of deleting data, first of all, the result has been observed when data is deleted without a fully efficient non-volatile memory structure. In this output, as seen in Figure-7, when the second data was deleted, it was found that the missing area was formed between the data.



```
0x00080000 E5B44352 45342312 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080040 FFFFFFFF FFFFFFFF A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080140 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 7:Data Deletion Output Before the Full Efficient NVM Usage Study

However, after integrating the fully efficient memory usage structure, as much as the size of the deleted data, the data after that address is shifted and thus the lost space is eluded as seen in Figure-8.

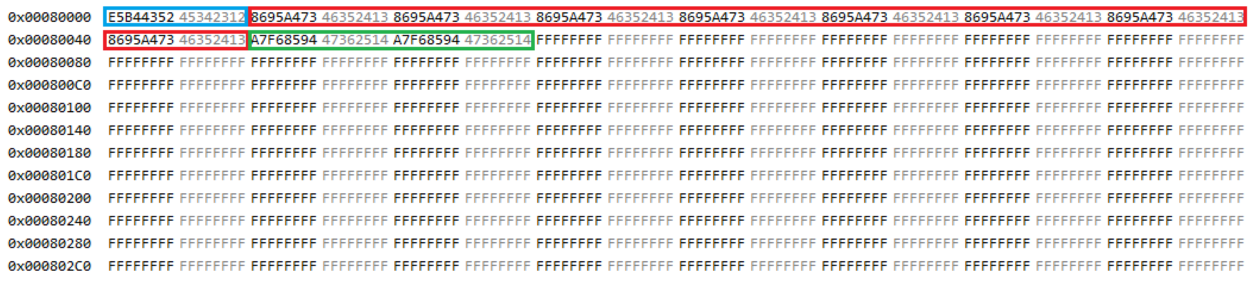


```
0x00080000 E5B44352 45342312 A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080040 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080140 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 8:Data Deletion Output After the Full Efficient NVM Usage Study

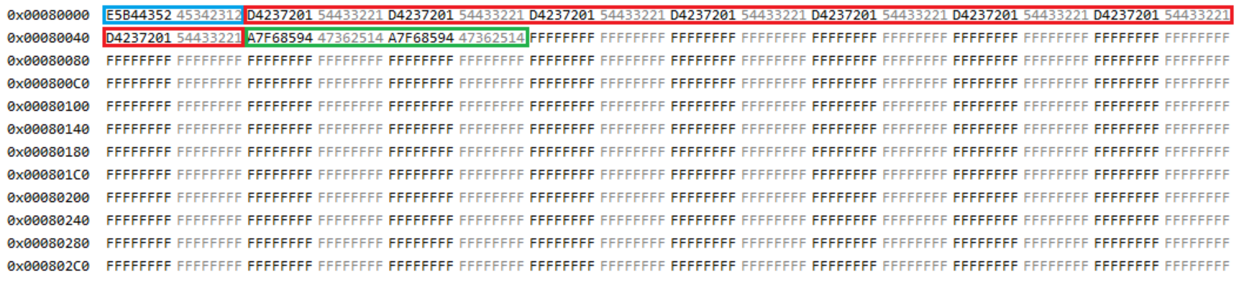
4.3 Data Updating

In the test of updating data, all data are represented with their older values as given in the Figure-9. After that, when the second data is updated, the output becomes as seen in the Figure-10.



```
0x00080000  E5B44352 45342312 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413 8695A473 46352413
0x00080040  8695A473 46352413 A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080140  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 9:Test Output with an Older Valued Data 2



```
0x00080000  E5B44352 45342312 D4237201 54433221 D4237201 54433221 D4237201 54433221 D4237201 54433221 D4237201 54433221 D4237201 54433221 D4237201 54433221
0x00080040  D4237201 54433221 A7F68594 47362514 A7F68594 47362514 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080080  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000800C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080100  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080140  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080180  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000801C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080200  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080240  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00080280  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000802C0  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

Figure 10:Test Output with a Newer Valued Data 2

5. Discussion and Conclusion

In summary, the full efficient NVM software modules developed for the MCUs have been developed to extend the life of NVM by optimizing the use of NVM and to increase the reliability of stored data by reducing the risk of data loss. Thanks to these features, it can be included in embedded and automotive systems. At the beginning of this study, the targeted and desired results were fully achieved.

On the other hand, by optimizing the software modules developed in addition to these features, the operations on the NVM can be made faster. This optimization is one of the aspects that can be improved for this project.

Finally, the power safe feature can be integrated into the system to ensure the security of the system against power cuts and to prevent data loss. In addition, the developed software modules can be encrypted within the scope of security, increasing the durability and reliability of the data.

Acknowledge

We would like to thank Huseyin Karacali, the Software Architect, for his expert guidance and inspiration as our mentor. We would also like to express our appreciation to TTTech Auto Turkey for their valuable support during the development of this project.

References

[1] C. Trick, "Volatile Memory vs. Nonvolatile Memory: What's the Difference?," *www.trentonsystems.com*. https://www.trentonsystems.com/blog/volatile-vs-nonvolatile-memory
[2] "NVM | Non-volatile memory - javatpoint," *www.javatpoint.com*. https://www.javatpoint.com/non-volatile-memory.
[3] "Non Volatile Memory," *Western Digital*. https://www.westerndigital.com/company/innovation/non-volatile-memory.
[4] "What is C? - Definition from WhatIs.com," *SearchWindowsServer*. https://www.techtarget.com/searchwindowsserver/definition/C
[5] B. Thompson, "What is C Programming Language? Basics, Introduction and History," *Guru99.com*, Oct. 14, 2019. https://www.guru99.com/c-programming-language.html
[6] R. Keim, "What Is a Microcontroller? The Defining Characteristics and Architecture of a Common Component - Technical Articles," *www.allaboutcircuits.com*, Mar. 25, 2019. https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/
[7] "What Is a Microcontroller? – Simply Explained," *All3DP*, Jan. 14, 2022. https://all3dp.com/2/what-is-a-microcontroller/

[8] "Microcontrollers - Overview - Tutorialspoint," *Tutorialspoint.com*, 2019. https://www.tutorialspoint.com/microprocessor/microcontrollers_overview.htm
[9] T. Wadhwa, "What is a Telematics Control Unit & How does it Work?," *LocoNav - Vehicle Tracking System | Telematics | Fleet Management System*, Apr. 22, 2022. https://loconav.com/blog/telematics-control-unit/.
[10] "Telematics control unit design resources | TI.com," *www.ti.com*. https://www.ti.com/solution/automotive-telematics-control-unit.
[11] I. T. AG, "Automotive telematics control unit (TCU) architecture - Infineon Technologies," *www.infineon.com*. https://www.infineon.com/cms/en/applications/automotive/automotive-security/telematics-control-unit/