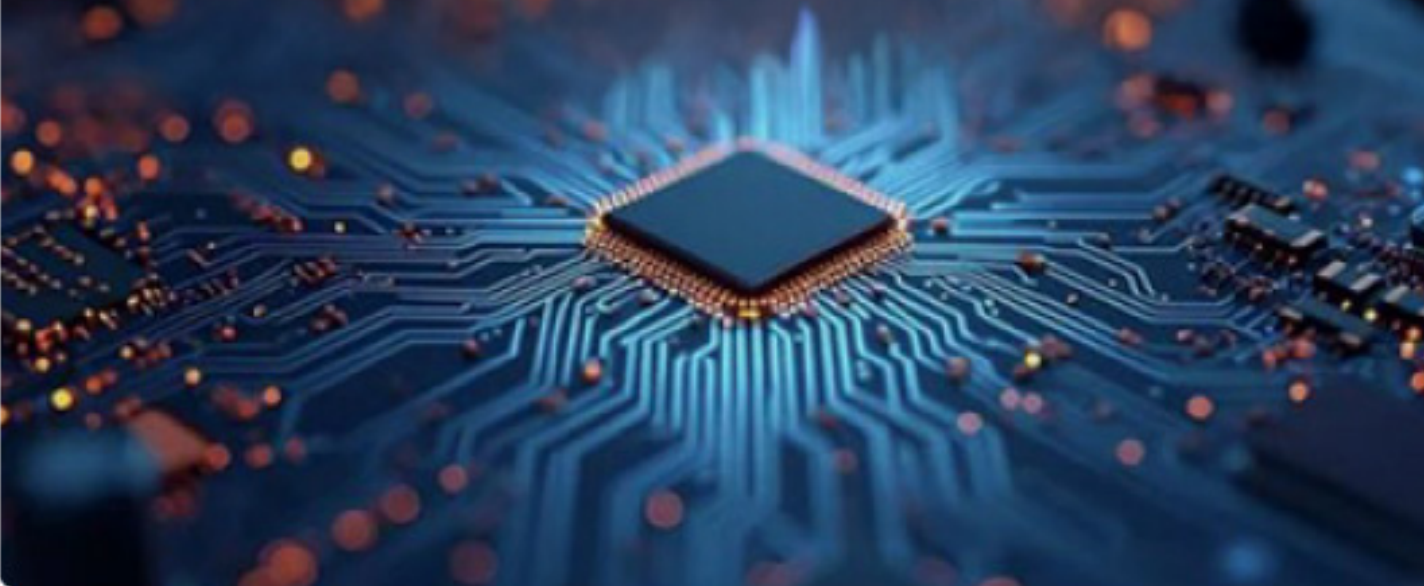


# Next-gen EEPROM wear levelling for embedded reliability excellence



# Table of contents

Overview	3
The challenges	3
The problem	4
The solution	4
Value delivered	9
Conclusion	9
Author information	11

# Overview

## Understanding EEPROM wear and its impact on embedded product reliability

In today's embedded systems, data integrity, reliability and device longevity are paramount. As devices operate in real-time environments across automotive, industrial, medical and consumer domains, efficient and sustainable memory usage becomes a critical design factor.

At the heart of this challenge lies Electrically Erasable Programmable Read-Only Memory (EEPROM) – a widely used non-volatile memory component valued for its ability to retain data across power cycles. However, EEPROMs are bound by finite endurance, defined as the maximum number of erase/ write cycles before data integrity is compromised. Typical endurance ranges from 1 million to 10 million cycles and frequent overwriting at the same address accelerates memory wear, leading to unreliable data retrieval and potential device failure. This limitation presents a significant barrier in long-duration embedded applications, particularly those requiring periodic data logging. Left unaddressed, it can erode system reliability and shorten product lifecycle.

HCLTech's engineering solution, a firmware-driven wear-levelling technique that leverages dynamic memory addressing, provides a sustainable answer to this challenge. By distributing write operations intelligently across memory cells, the technique extends EEPROM lifespan while safeguarding data integrity.

## The challenges

Finite endurance meets high-frequency demands: The EEPROM bottleneck in embedded systems

EEPROM endurance is finite. For embedded systems where key parameters (like temperature, pressure, etc.) are written every second, the risk of exceeding this endurance threshold becomes real.

Consider the example of the following parameter memory map:

Sr No	Record/parameter	Memory address (hex)	Typical value range
1	Room Temperature	0x0001	20-50
2	Air Pressure	0x0002	2-10

If such records are overwritten at these addresses every second, the memory endurance could degrade within weeks – leading to EEPROM failure, data corruption and system malfunction.

Furthermore, hardware constraints often necessitate saving values to predefined memory locations, complicating attempts to mitigate the degradation using traditional techniques.

# The problem

Preserving memory integrity without sacrificing logging frequency: The EEPROM longevity dilemma

Many embedded systems require periodic saving of operational records (e.g., sensor readings) to EEPROM for post-reset recovery. However, this frequent overwrite operation at fixed addresses drastically reduces the endurance of EEPROM.

Thus, the challenge lies in preserving EEPROM's lifespan while continuing to support the required frequency of data logging and retrieval.

# The solution

## Our dynamic addressing and wear levelling to maximize EEPROM longevity

To mitigate the inherent endurance limitations of EEPROM in embedded applications, our embedded engineering team implemented a firmware-driven dynamic wear-levelling mechanism. This approach directly addresses the root cause of premature EEPROM failure – frequent overwriting of records at static memory addresses – by adopting a strategy that systematically rotates write operations across available memory space.

By replacing traditional fixed memory mapping with dynamic memory addressing, the solution effectively prolongs EEPROM lifespan without compromising data integrity, retrieval accuracy or system performance.

### Key components of the solution:

<b>Dynamic write addressing</b>	Instead of saving parameter records at predefined, fixed memory locations, each new set of values is written sequentially to the next available memory address in a circular buffer. This rotating pattern prevents excessive wear at any single address and distributes write cycles evenly across the EEPROM.
<b>Auto-erase mechanism</b>	Prior to each new write operation, the EEPROM locations used during the previous cycle are explicitly erased. This ensures that previously stored data does not persist and reduces the risk of memory clutter or duplication. For example, if the previous cycle wrote values at 0x0018–0x0019, those addresses will be erased before the next values are written at 0x0020–0x0021.
<b>Power-cycle resilience and smart retrieval logic</b>	After a power reset or system restart, the firmware initiates a sequential scan across the EEPROM to detect valid data. Since erased EEPROM addresses return 0xFF, the firmware reads memory until it encounters the first non-0xFF value, which it interprets as the first parameter (e.g., room temperature). It continues reading the next values as per the expected memory map sequence (e.g., air pressure, humidity, etc.), successfully reconstructing the most recent state.

<b>Auto-tracking and write continuity</b>	The memory address where the last valid data set was retrieved becomes the new base address for the next write operation. This dynamic tracking ensures that the system writes from the correct point onward, maintaining continuity in data logging and retrieval operations without manual intervention.
---	--

### Illustrative example:

If the room temperature and air pressure were dynamically saved at addresses 0x0020 and 0x0021 respectively, the next write cycle (e.g., after 1 second) would store new records starting at 0x0022. Simultaneously, the older addresses (0x0018–0x0019) are erased. Upon power reset, the firmware would scan memory and skip over erased addresses (returning 0xFF) until it reaches the valid values at 0x0020, thereby ensuring complete data recovery.

### Outcome:

Through this dynamic wear-levelling technique, our solution extends the effective lifespan of EEPROM by orders of magnitude, while maintaining data reliability, logging frequency and operational continuity, crucial for long-duration embedded applications in real-time, safety-critical environments. Notably, this method requires no additional hardware and introduces minimal firmware overhead, making it a scalable, cost-efficient and highly reliable memory management strategy.

### Pseudocode / Software implementation:

```
#define TOTALPAGE_NUM 256//dummy size of Eeprom
#define PAGE_SIZE 24//dummy size of a page of
Eeprom
#define EEPROM_I2C_ADDRESS OXA4//dummy address of
Eeprom
#define BLANK_BYTE 0xFF

typedef struct
{
//Parameters to be saved from page 1 or onwards uint16_t P1;
//real time PCB temperature(used a multiplier of 10) uint8_t P2; //real time
state machine state uint8_t P3; //char

}EepromRecord;

I2C_HandleTypeDef hi2c1;
EepromRecord DataRecord_Eeprom;
uint16_t Record_Save_Page=PAGE_SIZE;

//To read the eeprom
Void ReadEeprom(void)
{
uint8_t ReadBuff[50], index=0,Read_Record_Size =4, i;
HAL_StatusTypeDef status=9; uint16_t MemAddress =
0;//EEPROM_PAGE_SIZE; //start add of
```

page 1

```
for(i=1;i<=TOTALPAGE_NUM;i++) { memset(ReadBuff,0,sizeof(ReadBuff));
//array initialization

MemAddress = EEPROM_PAGE_SIZE *i; if(MemAddress >
(TOTALPAGE_NUM*EEPROM_PAGE_SIZE) MemAddress =
EEPROM_PAGE_SIZE; //rollover to the 1st page if the page no exceeds the
max pages of eeprom status = HAL_I2C_Mem_Read(&hi2c1,
EEPROM_I2C_ADDRESS,
MemAddress, I2C_MEMADD_SIZE_16BIT, ReadBuff, 4, HAL_MAX_DELAY);
HAL_Delay(2);

if(status == HAL_OK)
{ if((ReadBuff[0]==BLANK_BYTE) && (ReadBuff[1]==BLANK_BYTE)
&& (ReadBuff[2]==BLANK_BYTE) && (ReadBuff[3]==BLANK_BYTE))

{
//Determine the add/page where the latest
record was saved
if(MemAddress != EEPROM_PAGE_SIZE) //
page 1 of eeprom
Record_Save_Page = MemAddress-
EEPROM_PAGE_SIZE; //previous page is where the latest records were
saved
else
{
Record_Save_Page = TOTALPAGE_
NUM*EEPROM_PAGE_SIZE; //if the 1st page is blank then it can be due to a
fresh eeprom or page rollover. Hence try reading the last page //Read record
from the retrieved page no where the last record was saved status =
9; //initialized to an unknown value so as to get a known value

memset(ReadBuff,0,sizeof(ReadBuff));
status = HAL_I2C_Mem_
Read(&hi2c1, EEPROM_I2C_ADDRESS, MemAddress, I2C_MEMADD_
SIZE_16BIT, ReadBuff,Read_Record_Size, HAL_MAX_DELAY);
HAL_Delay(2); if(status ==
HAL_OK)
{
//Load the values to the
respective parameters
DataRecord_Eeprom.P1 =
(ReadBuff[0]<<8) +
ReadBuff[1];
DataRecord_Eeprom.P2 =
ReadBuff[2];
DataRecord_Eeprom.P3 =
ReadBuff[3];
```

```

        }
        break;
    }
}

HAL_StatusTypeDef Page_Erase()
{
    uint8_t page_cntr=0;
    HAL_StatusTypeDef Eep_Stat =9;//initialized to an unknown value
    uint16_t page_erase_add = Record_Save_Page; //+
    EEPROM_PAGE_SIZE; uint8_t
    erase_arr[EEPROM_PAGE_SIZE];

    memset(erase_arr,BLANK_BYTE,EEPROM_PAGE_SIZE);

    if(page_erase_add > EEPROM_MEMORY_SIZE) //on last page
    page_erase_add = EEPROM_PAGE_SIZE; //rollover to 1st page else if
    (page_erase_add == EEPROM_MEMORY_SIZE) page_erase_add =
    EEPROM_PAGE_SIZE;//also erase the 1st page due to rollover else

    page_erase_add += EEPROM_PAGE_SIZE;

    Eep_Stat = HAL_I2C_Mem_Write(&hi2c1, EEPROM_I2C_ADDRESS,
    page_erase_add, I2C_MEMADD_SIZE_16BIT, erase_arr, EEPROM_PAGE_
    SIZE, HAL_MAX_DELAY);
    HAL_Delay(5); //5ms delay as like the wait period before starting the next
    eeprom write cycle

    return Eep_Stat;
}

Void SaveRecords()
{
    uint8_t Record_Buff[10]={0};Record_Size=4;i;
    HAL_StatusTypeDef eep_status = 9;
    uint16_t memadd=0;

    //Determine the eeprom page no to where the current records have to be
    saved
    //Increment the Record_Save_Page as this was the previous page no from
    where blank bytes were found only if it is not landing at the last page. In
    case of last page rollover to the 1st page
    Record_Save_Page += EEPROM_PAGE_SIZE; if(Record_Save_Page
    > EEPROM_MEMORY_SIZE) //if on last page Record_Save_Page =
    EEPROM_PAGE_SIZE; //rollover to 1st
    page
    eep_status = Page_Erase();
}

```

```

        if(eep_status == HAL_OK)
        {
            //Prepare the data to be saved //As temperature is saved
            //with a multiplier of 10, its lsb will
            //be saved lsb
            Record_Buff[0] = (uint8_t)(DataRecord_Eeprom.P1 >> 8); //
            //MSB of temperature
            Record_Buff[1] = (uint8_t)(DataRecord_Eeprom.P1 & 0xFF);
            //LSB of temperature
            Record_Buff[2] = DataRecord_Eeprom.P2;
            Record_Buff[3] = DataRecord_Eeprom.P3;

            stat = HAL_I2C_Mem_Write(&hi2c, EEPROM_
            I2C_ADDRESS, memadd, I2C_MEMADD_SIZE_16BIT, pData, Size,
            HAL_MAX_DELAY);
            HAL_Delay(5); //5ms delay as like the wait period before starting the
            //next eeprom write cycle
        }
    }

int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU
    Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    //Assume other initializations are done
    SystemClock_Config();

    uint32_t last_read_time = HAL_GetTick();

    ReadEeprom();

    While (1)
    {
        uint32_t current_time = HAL_GetTick(); if
        ((current_time - last_read_time) >= 1000) //1sec delay
            SaveRecords();
    }
} //end of main

```

# Value delivered

## Benefits of our dynamic EEPROM wear levelling solution

The implementation of our dynamic EEPROM wear levelling solution delivers measurable improvements across the entire embedded product lifecycle – enhancing reliability, reducing costs and enabling scalable performance across industries.

The following table outlines the key benefits and their direct impact on long-term system value.

Benefit	Value to product lifecycle
Extended EEPROM lifespan	Significantly reduces risk of premature wear and memory corruption by evenly distributing write cycles.
Improved system reliability	Ensures consistent, error-free parameter logging and retrieval – even after power cycles – with zero corruption incidents reported.
Cost-effective implementation	Firmware-only solution requires no additional hardware, reducing bill of materials and deployment complexity.
Lower maintenance costs	Minimizes need for memory replacements and field servicing, lowering Total Cost of Ownership (TCO).
Power-cycle resilience	Auto-detection of the most recent valid values post-reset without reliance on RAM backup.
Optimized firmware architecture	Simplifies memory access logic and supports modular, scalable firmware development.
Cross-domain applicability	Successfully deployed across IoT devices, industrial automation systems, automotive ECUs and other real-time embedded environments.

# Conclusion

## Maximizing EEPROM longevity through intelligent firmware engineering

EEPROM endurance has long been a limiting factor in embedded systems that require frequent data logging. This paper explored the critical trade-off between write frequency and memory longevity – an enduring challenge in high-reliability applications.

Our firmware-based dynamic memory addressing and wear levelling solution offers a powerful, scalable and hardware-agnostic answer. While the approach does not alter the intrinsic endurance rating of EEPROM, it strategically maximizes its usable lifespan by intelligently distributing write operations and simplifying post-reset data recovery. The result is a design that supports high-frequency operations without compromising data integrity or requiring hardware changes, delivering sustained product performance, lower maintenance costs and greater lifecycle efficiency.

This implementation reflects our engineering-first mindset and commitment to innovation, delivering solutions that are not only technically sound, but also optimized for cost, reliability and cross-domain scalability across automotive, industrial, medical and IoT applications.

In essence, this is an elegant resolution to a pervasive embedded systems problem, reinforcing our role as a trusted partner in next-generation product engineering.

# Author information



## **Gazala Mundra**

Gazala is a firmware evangelist and a solution architect working on various embedded products, including IoT. She has more than 18 years of experience in complete SDLC implementation, including requirement gathering and risk assessment, system design, firmware and middleware development and testing. She has worked in various domains, including manufacturing, industrial, off-highway, automotive, etc. She is richly experienced in bare metal programming in embedded systems.

# HCLTech | Supercharging Progress™

HCLTech is a global technology company, home to more than 223,000 people across 60 countries, delivering industry-leading capabilities centered around digital, engineering, cloud and AI, powered by a broad portfolio of technology services and products. We work with clients across all major verticals, providing industry solutions for Financial Services, Manufacturing, Life Sciences and Healthcare, High Tech, Semiconductor, Telecom and Media, Retail and CPG and Public Services. Consolidated revenues as of 12 months ending June 2025 totaled \$14 billion. To learn how we can supercharge progress for you, visit [hcltech.com](https://hcltech.com).

[hcltech.com](https://hcltech.com)

