Exploiting Half-wits: Smarter Storage for Low-Power Devices

Mastooreh Salajegheh*, Yue Wang†, Kevin Fu*, Anxiao (Andrew) Jiang†, Erik Learned-Miller*
*Department of Computer Science, University of Massachusetts Amherst
†Department of Computer Science and Engineering, Texas A&M University
{negin,kevinfu,elm}@cs.umass.edu, {yuewang,ajiang}@cse.tamu.edu

Abstract

This work analyzes the stochastic behavior of writing to embedded flash memory at voltages lower than recommended by a microcontroller's specifications to reduce energy consumption. Flash memory integrated *within* a microcontroller typically requires the entire chip to operate on common supply voltage almost double what the CPU portion requires. Our approach tolerates a lower supply voltage so that the CPU may operate in a more energy efficient manner. Energy efficient coding algorithms then cope with flash memory that behaves unpredictably.

Our software-only coding algorithms (*in-place writes*, *multiple-place writes*, *RS-Berger codes*) enable reliable storage at low voltages on unmodified hardware by exploiting the electrically cumulative nature of half-written data in write-once bits. For a sensor monitoring application using the MSP430, coding with in-place writes reduces the overall energy consumption by 34%. In-place writes are competitive when the time spent on computation is at least four times greater than the time spent on writes to flash memory. Our evaluation shows that tightly maintaining the digital abstraction for storage in embedded flash memory comes at a significant cost to energy consumption with minimal gain in reliability.

1 Introduction

Billions of microcontrollers appear in embedded systems ranging from thermostats and utility meters to tollway payment transponders and pacemakers[1]. Recent years have witnessed a proliferation of low-power embedded devices [2, 7, 17, 21], many of which use on-chip flash memory for storage.

While the reliability, low cost, and high storage density of flash memory make it a natural choice for embedded systems [15], its relatively *high voltage requirement* (Table 1) introduces challenges for energy-efficient designs aiming to maximize the system's effective lifetime (e.g., the run time on a typical battery whose voltage declines over time). Instrumenting the system to operate at a fixed low voltage v_l is one way to reduce power consumption; however, achieving *consistently correct* results for flash writes are guaranteed only if v_l is higher than a manufacturer-specified threshold. Moreover, in energy-limited devices that cannot provide a constant supply voltage, scenarios may arise in which the flash memory is the only part of the circuit whose operating requirements are not met. In such cases, applications can expect normal operation when they are not performing flash writes and unpredictable behavior when they are.

Microcontroller	CPU Min. voltage	Flash write Min. voltage
TI MSP430 [36]	1.8 V	2.2 or 2.7 V
PIC32M [24]	2.3 V	3.0 V
ATmega128L [3]	2.7 V	4.5 V

Table 1: Flash memory restricts choices for the CPU voltage supply on microcontrollers because the CPU shares the same power rail as the on-chip flash memory.

Because embedded flash memory typically shares a common voltage supply with the CPU (separate power rails are cost prohibitive), a single voltage must be chosen that satisfies different components with different minimum voltage requirements. Current embedded systems address the voltage limitations of flash memory in one of the following ways:

i) A system can choose a high supply voltage sufficient for both reliable writes to flash memory and reliable CPU operation. This is a common choice for embedded systems with on-chip flash memory, but causes the CPU to consume more energy than necessary. For example, the TI MSP430F2131 microcontroller [36] in active mode consumes almost double the power when operat-

[1] A single manufacturer claims to have shipped over 8 billion microcontrollers http://www.microchip.com/sec/annual/FY10/.

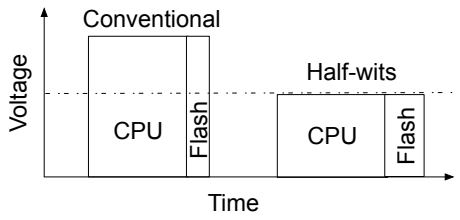


Figure 1: Operating at a lower voltage and tolerating errors instead of the conventional case of choosing the highest minimum voltage requirement may help decrease energy consumption. Considering that $Energy = voltage^2 \times time/resistance$, decreasing voltage decreases the energy consumption quadratically.

ing at 2.2 V instead of 1.8 V. Its onboard flash memory requires 2.2 V for reliable writes to flash memory.

ii) A system can choose a low supply voltage sufficient for CPU operation, but insufficient for reliable writes to flash memory. This choice allows the energy source to last longer and for the CPU to compute more efficiently. An example of such a system is the Intel WISP [33], a batteryless RFID tag that sets its operating voltage to 1.8 V—below its onboard flash memory's 2.2 V specified minimum—to save power. Flash memory cannot be written on this device. The microcontroller could use a low-power wireless interface (e.g., RF backscatter) to store data remotely. Such an approach, however, raises privacy as well as performance concerns [32].

iii) A system can modify hardware to enable dynamic voltage scaling. This approach requires additional analog circuitry such as voltage regulators and GPIO-controlled switches. Because many embedded systems are extremely cost sensitive, this choice is unattractive for high-volume manufacturing with low per-unit profit margins. An additional 50 cent part on a thermostat control can be cost prohibitive. Moreover, small changes may necessitate a new PCB layout—upsetting the delicate supply chain and invalidating stocked inventories of already fabricated PCBs.

Approach. Our approach reduces the operating voltage of the microcontroller to a point at which the resulting energy savings of the CPU portion of the workload exceeds the energy cost of the algorithms for ensuring reliable writes (Figure 1). The technique requires minimal or no hardware modification and also allows for RFID-scale devices to better exploit capacitors as power supplies. The capacitor provides finite energy and therefore the voltage decays exponentially. The long tail of the curve provides insufficient voltage for conventional writes to flash memory, but is sufficient for reliable storage with our techniques.

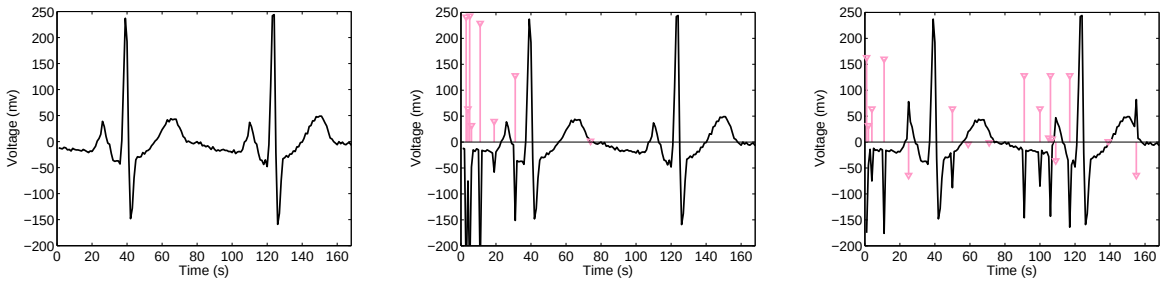
Of wits and half-wits. In 1982, Rivest and Shamir introduced the notion of write-once bits (wits) in the context of coding theory to make write-once storage behave like read-write storage [31]. Bits in flash memory behave like wits because a programmed bit cannot be reprogrammed without calling an energy-intensive erase operation to a block of memory much larger than a single write. We coin the term *half-wits* to refer to wits used in a manner inconsistent with a manufacturer's specifications, resulting in stochastic behavior. Half-wits in this work are wits of flash memory used below the recommended supply voltage.

In examining error rates at low voltage and constructing a system that provides reliable storage despite errors, our work suggests that it is appropriate to relax previously assumed constraints and reexamine the costly digital abstractions layered above on-chip flash memory.

Contributions. Our primary contributions include an empirical evaluation that characterizes the behavior of on-chip flash memory at voltages below minimum levels specified by manufacturers, and algorithms that enable reliable writes to flash memory while coping with low voltage. Our evaluation identifies three key factors affecting error rates: voltage, Hamming weight of the data, and the wear-out history of the flash memory.

The first algorithm, *in-place writes*, makes attempts at *write time* to store a value correctly in the given memory address. The *in-place writes* method repeatedly writes data to the same memory address. The intuition behind this approach is that repeating a write attempt in a consistent location accumulates the charge in the same cell, increasing the chance of storing a bit of information correctly. In addition, since flash writes only change bits in a single direction, a correctly written bit cannot be reversed to produce an error on a second write attempt. The second algorithm, *multiple-place writes*, tries to decrease the probability of error by making attempts at both write time and read time. This method stores data in more than one location aiming that the data (even partially) will be stored correctly in at least one of these locations. The third algorithm is a hybrid error-correcting code combining Reed-Solomon (RS) [29] and Berger [5] codes. The Berger code detects, but does not correct, asymmetric errors caused by the low write voltage. Given the approximate locations of errors, which are determined by the Berger code, the RS code efficiently recovers the originally stored data.

The paper compares all three methods in terms of energy consumption, execution time, and error correction rate. We also show that our methods are most effective for CPU-bound workloads. With respect to cost and energy, our techniques may enable already deployed embedded flash memory to remain competitive with emerging technology for low-power, non-volatile memory.



(a) Writes at 2.0 V (b) Writes at 1.9 V (c) Writes at 1.8 V

Figure 2: As operating voltage decreases, flash-write errors increase. (a) shows an original ECG signal correctly stored at 2.0 V (despite operating below the recommended threshold). As the voltage decreases in (b) and further in (c), erroneous writes (light-colored spikes, height varying according to the magnitude of the error) become more common. The black line shows the reconstructed signal that includes the errors.

2 Behavior of Storage on *Half-wits*

Before we can design effective coding algorithms, we must first understand the behavior of errors in half-wits. By tolerating a lower voltage, an energy-limited embedded device can decrease its power consumption and therefore extend its lifetime on a finite energy supply[2]. The minimum operating voltage of embedded devices that use nonvolatile on-chip storage is usually determined by the requirements of flash memory. For example, the TI MSP430 microcontroller can operate at 1.8 V, but its nominal minimum voltage for flash writing and erasure is 2.2 V (Table 1). Increasing operating voltage from 1.8 V to 2.2 V causes the CPU to draw about 50% more power without commensurate gain in clock speed because of the voltage squaring effect.

The drawback of lowering voltage below flash memory requirements in order to save power is the loss of flash memory reliability. Figure 2 shows the result of running a MSP430F2131 at three different voltages—all lower than the nominal minimum for flash writes—to store electrocardiogram (ECG) data samples from the PhysioNet database [13] in flash memory. Many medical sensor networks [20, 22, 34] that provide ECG measurements are energy limited and use on-chip flash memory as primary storage.

These graphs support the intuition that flash writes may not be error free at low voltages and that there exist voltage levels below the minimum recommended voltage at which flash writes function correctly[3]. To investigate the behavior of flash memory at low voltage and determine the factors influencing the error rate, we performed experiments on an automated testbed of our own design.

[2]Or because of relaxed requirements, eliminate the need for multiple batteries in series to achieve a high voltage.

[3]Moreover, a nonzero error rate may be tolerable by some applications. In the case of ECG data, the cardiac pulse interval can be recovered from noisy data stored at low voltage.

2.1 Experimental Methodology

We use a consistent experimental setup for all of the experiments in this work. Our choice of test platform is a TI MSP430 [36] microcontroller with on-chip flash memory. More specifically, we tested two types of TI microcontrollers: MSP430F2131 and MSP430F1232. The MSP430 is common in low-power embedded applications; we note especially its use in sensor motes [28] and RFID-scale batteryless devices [33]. In our setup, an MSP430 microcontroller runs a test program that involves both computation and flash operation. We power the microcontroller with an external power supply held steady at a voltage below the nominal minimum for flash writes. An external chip captures the contents of flash memory after each experiment.

To automate the testing of flash write behavior, we have developed a flash memory testbed. The two major components of the testbed are a test platform and a connected monitoring platform. The monitoring platform is based on an additional MSP430 microcontroller. The test platform runs a test program at low voltage. When the test program completes, the test platform sends the result of the experiment to the monitoring chip via GPIO pins. The test and monitoring platforms share 8+1 GPIO pins to carry one byte of data and a clock signal. Once the test platform puts data on its eight data pins, it raises the clock pin. The monitoring chip reads data from its GPIO pins whenever it detects a rising clock signal and logs the results in its own flash memory. The monitoring chip runs at a voltage above the nominal minimum for its own flash writes, and therefore stores reliably.

2.2 Unreliable, Low-Voltage Flash Memory Writes

The TI MSP430 datasheet [36] states that flash writes at any voltage lower than the nominal minimum voltage (which is 2.2 V in the case of MSP430F2131) are not guaranteed to succeed. However, as the graphs in

Figure 2 show, not all flash writes fail at low voltages. On the contrary, in this specific experiment, most of the writes (95.24% at 1.9 V and 89.88% at 1.8 V) succeed.

In a NOR flash memory, all cells are initialized to 1, and writing data to a byte of flash memory means setting an appropriate number of bits to 0 by applying electrical charge to the corresponding flash cells. At low voltage, there may be insufficient charge to effect a transition to 0, and a flash write may store fewer 0 bits than requested [27]. To be specific, we define errors as follows: when a byte of data d_1 is written in a flash memory address and then data d_2 is read from that address, there is an error if $d_1 \neq d_2$. An experiment, explained next, investigates the behavior of low-voltage flash memory and gives bit-level results.

Using the automated flash testbed explained in Section 2.1, the test platform runs a program that writes numbers $\{0, \cdots, 255\}$ to flash memory, then sends the contents of its flash memory to the monitoring platform via GPIO pins. Table 2 compares the written data and the intended data for cases in which errors occurred. It demonstrates that, when both are represented as integers, the absolute value of the stored data is always greater than or equal to the absolute value of the intended data.

2.3 Determining Factors That Affect Error Rates

We consider the following potential factors that may affect the error rate of setting a bit to 0 in a flash memory at low voltage: voltage level, Hamming weight of the data, wear-out history, permutation of 0s, and neighbor cells. The effects of each of these variables are evaluated by designing an experiment to test a hypothesis. All the experiments are performed on flash memories with minimal previous usage unless stated otherwise.

Voltage level: Our hypothesis is that the lower a chip's operating voltage (and that of its on-chip flash memory), the higher the error rate of flash writes. Figure 3 confirms this hypothesis; moreover, the graph shows that for different chips of exactly the same type, the error rate can be different even under equivalent voltage.

Experiment: Two MSP430F2131 and two MSP430F1232 microcontrollers run a program that writes zeros to the data segment of their flash memory. We increased the microcontroller's operating voltage in 10 mV steps, and used the monitoring platform to compute the byte error rates over 50 runs.

Hamming weight: In an erased (i.e., having value 1) flash cell, writing a 1 is always error free because no change to the cell is necessary. However, setting a cell to 0 might fail if there is not enough charge accumulated in that cell. Our hypothesis is that, the lower the Hamming weight (number of 1s in the binary representation) of a number, the higher the probability of error when writing that number to flash at low voltage.

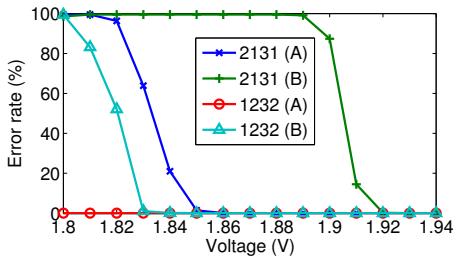


Figure 3: Flash write error rates decrease as voltage increases. This trend holds for all the chips (MSP430F2131 and MSP430F1232) we tested, though error rates differ even between chips of the same model.

Based on per-byte Hamming weight, there are nine equivalence classes of integers that can be represented in one byte. The weight-8 equivalence class has only one member, 255, which can always be written to an erased flash cell without error. The other extreme case is the weight-0 equivalence class, containing only 0s, that requires all eight bits to transition to 0. Figure 4 shows the byte error rate for all nine equivalence classes, measured via the following experiment.

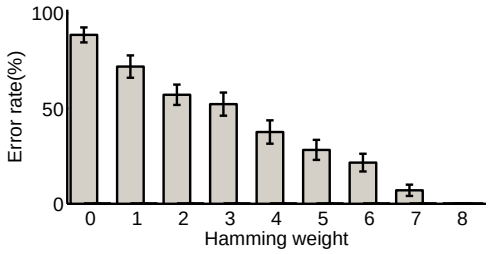


Figure 4: As the Hamming weight (number of 1s in the binary representation) of a number increases, the error rate of low-voltage flash write declines. The data corresponds to a MSP430F2131 running at 1.84 V.

Experiment: At 1.84 V, a MSP430F2131 runs a program that writes numbers from the same equivalence class to one block (64 bytes) of flash memory. We used the monitoring platform to compute the average byte error rate of flash writes for each of the nine equivalence classes over 50 runs.

Corollary: To exploit the fact that the Hamming weight of a number affects probability of error when it is written to flash, one can transform numbers into numbers with greater Hamming weights before writing them to flash memory.

Wear-out history: Flash memory has a limited lifetime (about 10^5 cycles of erasures) after which the erase operations fail to reliably reset the bits to 1. We suspect that the more flash memory is erased (worn-out), the

(Binary)	Intended	00001100	00001101	00001110	00010100	00100111	10100100
	Written	11101101	01011111	11111111	11111111	00101111	10101111
Hamming distance		4	3	5	6	1	3

Table 2: Erroneous flash writes at low voltage. Insufficient electrical charge may result in some bits failing to transition from 1 (the initial state) to 0.

lower its error rate of setting bits to 0 would become[4]. Figure 5 shows a heat map of bit error rate for three blocks of flash memory (192 bytes) on an MSP430F2131 microprocessor. Lighter colors in the heat map represent higher error rates. The disproportionately dark color of the middle block is due to more frequent erasure of that block compared to the other two blocks.

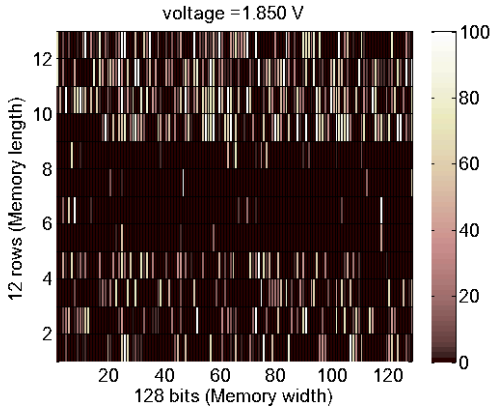


Figure 5: Worn-out flash memory blocks are biased toward ease of writing zeros. Lighter color represents higher average number of errors over 50 trials. The middle block has been write/erase cycled 6,000 times. The other two blocks are minimally used.

Experiment: A MSP430F2131 runs a program that writes zeros to all three blocks of its flash memory. The MSP430 is first worn out such that one block has 6,000 write/erase cycles and two blocks have minimal previous usage. We used the monitoring platform to compute the average error rate for all bits in the three blocks of memory over 50 trials.

Corollary: Wear-out history affects error rate, so storing data in more than one location might help decrease the error rate, especially if those locations are in different blocks of memory.

Permutation of 0s: Two numbers belonging to the same Hamming-weight equivalence class can have different permutations of 0 bits. We tested to see if the error rate depends on the permutation of 0s in one byte of data. For example, the numbers 240, 15, 170, and 71 all have four 0s in their binary representation but in different places (240 has 0s in the right nibble, and 15 has all of its 0s in its left nibble, etc.). The result of the experiment shows a similar byte error rate with mean of $39.85 \pm 4.29\%$ for numbers in the same equivalence class. The small standard deviation (4.29%) shows that the permutation of 0s does not significantly affect the error rate and therefore we do not consider this factor in our design directions.

Experiment: A MSP430F2131 runs a program that cycles through eight numbers from the same Hamming-weight equivalence class, writing them to 192 consecutive bytes of flash memory. We used the monitoring platform to compute the average error rates for each of the 192 bytes over 50 trials.

Neighbor cells: Another factor that might affect the error rate of storage in a flash cell at low voltage is the values of neighboring cells. However, our results suggest that a cell's error rate does not appear to depend on the values stored in neighboring cells (Figure 6).

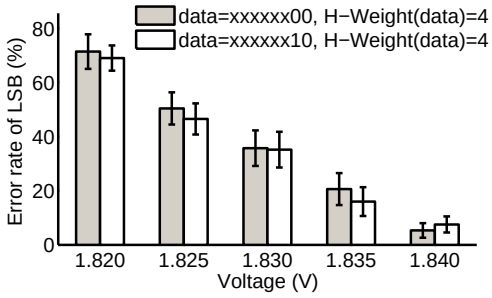


Figure 6: Error rate of a cell is not noticeably influenced by the value of its neighbor. The graph shows that the value of the second LSB does not greatly affect the error rate of the LSB. The bars show the error rate of the LSB for writing numbers from the same Hamming-weight equivalence class whose two LSBs are set to either 00 (dark bars) or to 10 (light bars).

Experiment: In order to determine if the error rate of a cell is affected by its neighbor, we consider all numbers from the same Hamming-weight equivalence class whose two Least Significant Bits (LSBs) are set to either 00 (case 1) or 10 (case 2). An example of case 1 is number 60 (0b00111100) and an example of case 2 is number 30 (0b00011110). This experiment fixes the Hamming weight variable and changes the neighbor value of the

[4]This counterintuitive hypothesis is consistent with the notion that flash erasures (settings bits to 1) become harder with wear out.

LSB to be 0 or 1. We deem a write erroneous if the LSB is not set to 0. The experiment was done for a Hamming weight of four and it was repeated for five voltage levels in the interval of 1.82 V to 1.84 V with steps of 5 mV. The error rate for any voltage above 1.84 V was close to 0% and for any voltage below 1.82 was close to 100%. We used the monitoring platform to compute the average error rates of case 1 and case 2 for each of the voltage levels over 50 trials.

2.4 Accumulative Memory Behavior

It is helpful to understand a few details of the electrical nature of flash memory in order to appreciate the expected behavior of conventional digital abstractions when layered above embedded flash memory. Each flash memory cell is a floating-gate (FG) transistor made up of a source, drain, control gate, and floating gate. The floating gate is separated from the source and drain by an insulating oxide layer that makes it difficult for electrons to travel into or out of the gate. Flash cells rely on this oxide to maintain logical state in the absence of power, making the memory non-volatile [27].

To write a memory cell (which has an erased value of 1), the control circuitry applies a high field to the source. The application of this field greatly increases the probability that electrons in the floating gate will tunnel to the source. If a sufficient number of electrons tunnel to the source, the cell is subsequently read as a 0. To erase a cell (restoring a 1), the control circuitry applies a high field to both the source and drain. This field energizes the electrons currently stored near the source, allowing them to jump the oxide barrier to the floating gate where they are once again trapped [27].

Not all electrons must transition in order for a write or erase operation to be successful. The operation only needs to change the state of some majority of the electrons so that subsequent read operations detect sufficient charge to discern the intended value. Lowering the applied voltage (and thus the field strength) lowers the probability of state change for each electron but, as noted earlier, electrons that do transition will remain in place.

A low-power storage scheme can benefit from this accumulative property by repeating writes to the same cell. Each write operation will increase the chance of success by forcing some number of state transitions. In other words, a failed write is still progress.

3 Design of a Low-Voltage Storage

This section presents our design for a software system that enables reliable flash memory writes at low voltage. We first present a model that captures the basic characteristics and behavior of flash memory. We then set design goals with that model under consideration. We introduce three methods for reliable flash storage, which we refer to as *in-place writes*, *multiple-place writes*, and *RS-Berger codes*. Each method aims to meet our design goals for reliable non-volatile storage.

3.1 Modeling Low-Voltage Flash Memory

A NOR flash memory has a set of n cells that are initially set to 1. We represent the state of the cells by $c_1, \ldots, c_n$; the value of c_i can be 0 or 1. A cell can be set to 0 using a write operation. The $1 \rightarrow 0$ transition might fail at low voltage while the $1 \rightarrow 1$ will obviously succeed. Flash memory at low voltage, where errors occur only in one direction, can be modeled as a Z-channel [19].

Flash memory is a write-once memory [31] and once a cell is set to 0 (i.e., once it is programmed), it cannot be changed back to 1 without using an erase operation. In flash memory, cells are organized by blocks, and an erase operation resets an entire block of cells. Block erasures are costly in terms of time and energy and they cause wear to flash cells.

Operations: There are two operations in this model: (1) An update operation that changes a subset of cells to 0 to represent a value, and (2) A decoding operation that maps cell states (i.e., memory state) to a value. Updating a variable means changing the values of $c_1, \ldots, c_n$ to $c'_1, \ldots, c'_n$. Assuming no erase operation occurs, and therefore no bits are reset to 1 after being set to 0, we have $\forall i \in \{1, \ldots, n\}, c_i \geq c'_i$ after an update. If the update operation is performed when operating voltage is below the nominal minimum required for flash memory, the update operation may not be error free.

3.2 Design Goals

Our storage techniques, which aim to provide reliable storage for low-power devices, are designed with the following metrics in mind:

- *Error rate:* The first and foremost design goal is to minimize the error rate to provide applications with reliable non-volatile storage.
- *Energy consumption:* The energy consumed to achieve an acceptably low error rate should not exceed the expected energy savings gained by running at a lower voltage.
- *Delay:* We define delay as the difference between the execution time to reliably store data at a low voltage and to store the same data at a high voltage. The delay caused by the storage technique should be reasonably small.

3.3 Proposed Methods

Toward the design goals discussed previously, we propose methods to deal with errors caused by using flash memory at low voltage.

3.3.1 In-Place Writes

Since the transition of a 1 to a 0 in a NOR flash memory at low voltage is stochastic rather than guaranteed, the *in-place writes* method repeats the write of each byte (to the same memory location) more than once if necessary, up to a *threshold* number of attempts. Algorithm 1 gives the details for ENCODE and DECODE procedures for in-place writes.

Algorithm 1 The encoding and decoding algorithms for *in-place writes* method to store *data* to *address* by repeating the writes up to *threshold* a number of attempts if necessary.

```
ENCODE(data, address, threshold)
1  WRITE_TO_FLASH(data,address)
2  result ← READ_FROM_FLASH(address)
3  repeat ← 1
4  while (result ≠ data) AND (repeat < threshold)
5        do WRITE_TO_FLASH(data,address)
6           result ← READ_FROM_FLASH(address)
7           repeat ← repeat+1

DECODE(address)
1  result ← READ_FROM_FLASH(address)
2  return result
```

The reason *in-place writes* decrease the error rate is that, as explained in Section 2.4, each write attempt in the same memory location increases the accumulated charge and therefore raises the probability of storing the intended bit sequence successfully.

3.3.2 Multiple-Place Writes

Another approach to increase the reliability of flash writes at low voltage is to write a value to more than one location in flash memory if necessary up to a *threshold* number of locations. Later, to retrieve the stored data, the *multiple-place writes* method reads the data from the specified address and several other addresses associated with it, then returns the bitwise AND of all of the stored values. Algorithm 2 details ENCODE and DECODE procedures of the *multiple-place writes* method. Writing a value to more than one memory location increases the probability of storing it successfully in the flash memory.

The reason the *multiple-place writes* approach can decrease the error rate is as follows: All cells of flash memory are initially set to 1. An error means that writing a 0 has failed and a bit cell c_i has remained untouched (logical 1) although it was intended to be set to 0. If the cell write in one of the locations has not failed, and cell c_i is 0 in at least one location, getting the AND of the read values from all locations will make cell $c_i = 0$ in the AND result. The case of writing a 1 to a cell does not cause an error since it means changing a cell from 1 to 1.

Algorithm 2 The encoding and decoding algorithms for *multiple-place writes* method to store *data* to *address* by repeating the writes up to a *threshold* number of locations if necessary. The distance between each of these associated locations is *offset*.

```
ENCODE(data, addr, threshold, offset)
1  WRITE_TO_FLASH(data,addr)
2  result ← READ_FROM_FLASH(addr)
3  repeat ← 1
4  while (result ≠ data) and (repeat < threshold)
5        do phy_addr ← addr + (repeat × offset)
6           WRITE_TO_FLASH(data,phy_addr)
7           n_result ← READ_FROM_FLASH(phy_addr)
8           result ← result & n_result
9           repeat ← repeat+1

DECODE(addr, threshold, offset)
1  for i ← 0 to (threshold−1)
2        do phy ← addr + (i × offset)
3           n_result ← READ_FROM_FLASH(phy)
4           result ← result & n_result
5  return result
```

3.3.3 RS-Berger Codes

Our third method to provide reliable flash memory at low voltage involves data coding. We use the concatenation of Reed-Solomon [29] and Berger [29] codes—which we call *RS-Berger codes*—to detect and correct errors at read time. Reed-Solomon is a widely used error-correcting code that can correct twice as many erasures as errors. Therefore, if the locations of errors are known, an RS code's error-correcting capacity is improved twofold.

To detect the location of errors and therefore improve the efficiency of the RS code, we use a Berger code, an error-detecting code for asymmetric channels. As previously mentioned (Section 3.1), flash memory at low voltage can be modeled as a Z-channel for which a Berger code is suitable. A Berger codeword consists of two parts: k information bits and $\lceil \log_2(k+1) \rceil$ check bits. The check bits of the Berger codeword represents the number of zeros in the k information bits. The Berger code can detect all zero-to-one errors, because the number of zeros in the information-bit component will always be less than the number represented by the check-bit component.

To represent RS-Berger codewords, we use a matrix in which each row is an RS codeword except for the last row which includes the Berger check bits of the RS codewords. In other words, each cell in the last row of the matrix is the sum of the number of zeros in the corresponding cells in all other rows.

When encoding the data, we first use RS code to generate n codewords (rows of the matrix) and then we apply a Berger code to compute the check bits for each symbol for all codewords (each column of the matrix).

When decoding data, we first use the Berger decoder to check whether or not each column is erroneous. If one entry in the column is erroneous, we consider all the symbols in the column erasures; otherwise, all the symbols in the column are considered correct. Then, once the error locations are known, we apply RS decoding to correct the erroneous sequences row by row.

Algorithm 3 The encoding and decoding algorithms for *RS-Berger codes* write method. t is the maximum number of errors RS-Berger code can correct.

```
ENCODE(data_{1,..,N}, n)
1  for i ← 1 to N
2       do CW_i ← RS_ENCODE(data_i, n)
3          WRITE_TO_FLASH(CW_i, address_i)
4  for i ← 1 to n
5       do for j ← 1 to N
6             do sym_{i,j} ← CW_j(i)
7          chk_i ← BERGER_ENCODE(sym_{i,(1,..,N)})
8          WRITE_TO_FLASH(chk_i, address_{N+1} + i-1)

DECODE(addr_{1,..,(N+1)}, n, t)
 1  for i ← 1 to N
 2       do chk_i ← READ_FROM_FLASH(addr_{N+1}+i-1)
 3  for i ← 1 to N
 4       do CW_i ← READ_FROM_FLASH(addr_i)
 5          for j ← 1 to n
 6             do sym_{i,j} ← CW_i(j)
 7  errors ← {}
 8  for i ← 1 to n
 9       do err ← BERGER_DECODE(sym_{i,(1,..,N)}, chk_i)
10          if err = 0
11            then errors ← errors ∪ {i}
12  if |errors| ≤ t
13     then for i ← 1 to N
14             do result_i ← RS_DECODE(CW_i, errors)
15          return result
16     else return "fail to correct errors"
```

4 Evaluation

Our storage techniques are designed for the resource limitations of low-power devices. In this section, we first evaluate the suitability of the three methods proposed in Section 3.3 for low-power devices; we then evaluate the hypothesis that for CPU-bound workloads, operating at low voltage and managing errors is more energy efficient than fixing the operating voltage to the maximum of all the components' nominal minimum voltages.

Summary of results: For a sensor monitoring application that reads 256 data samples from flash memory, aggregates data, and stores the results in flash memory, use of *in-place writes* at 1.8 V reduces the energy consumption up to 34% versus running the same application at 2.2 V (minimum voltage requirement for the flash memory). This sensing application models a common workload for both wireless sensor nodes and RFID-scale devices.

Experimental setup: We used a consistent experimental setup to measure the energy consumption and execution time of each program. Using an oscilloscope, we measured the voltage of a small resistor in series with a MSP430 microcontroller programmed with a task (e.g., a flash write). The integration of the current (voltage divided by the resistance) over the execution time of the task multiplied by the operating voltage of the device gives the energy consumption of that task ($Energy = \int I(t)\ dt \times V$). To facilitate precise identification of the task on the oscilloscope, the microcontroller toggled a GPIO pin immediately before and after the task.

4.1 Comparison of the Proposed Storage Methods

The workload used to measure the performance of each of the proposed methods is the storage of accelerometer traces—generated using the Intel WISP 4.1's 10-bit ADC sensor—to flash memory. The input trace is a series of three-dimensional 16-bit samples containing ten bits of information. We used a simple data compression method to store more data in the available flash memory. The compression method involved reading four samples of data, preparing the first byte of each sample to be stored in flash memory, then combining the remaining two bits of each sample into one byte of data. Using this compression scheme, we reduced every four samples (eight bytes) to five bytes.

The maximum number of write attempts for both *in-place writes* and *multiple-place* methods were set to two. The *RS-Berger codes* used three codewords of size 38 bytes (32 bytes data and 6 bytes parity). These settings enable all three methods to fit their data in 192 bytes of flash memory. Table 3 shows the energy consumption and time taken for the same workload under each method. Both *in-place writes* and *multiple-place writes* consume less energy and finish more quickly at 1.9 V

than at 1.8 V. Both of these methods are feedback based and repeat writes if they detect errors. Because there is a lower chance of error at 1.9 V, fewer rewrites are required than at 1.8 V, so less energy and time are required.

The *in-place writes* method slightly outperforms the *multiple-place writes* method at both voltage levels because its decoding procedure is less CPU intensive. *In-place writes* method has the best Error Correction Rate (ECR in Table 3) of all. The *multiple-place writes* method seems to be the most suitable when there are some memory cells that are hard to program and therefore rewriting in those cells is not helpful (Figure 5 gives an example of such case). Compared to *RS-Berger codes* that always guarantee that a certain number of errors can be corrected, the *in-place writes* and *multiple-place writes* methods are less reliable—they offer no such guarantees. Therefore, for applications with a hard reliability requirement, *RS-Berger codes* may be more suitable if the application knows the error rate in advance and is willing to incur extra computational costs for RS-Berger encoding and decoding.

Method	**V**	**Time (ms)**	**E (μJ)**	**ECR**
In-place	1.8	24.16	59	96%
M-place	1.8	25.00	63	84%
RS-B	1.8	334.45	160	0%
In-place	1.9	15.43	38	100%
M-place	1.9	16.85	40	100%
RS-B	1.9	334.73	180	100%

Table 3: Performance comparison of the proposed methods at 1.8 V and 1.9 V. Error Correction Rate (ECR) shows the effectiveness of the methods.

Error Correction Rate: As Table 3 illustrates, the two methods that do not use coding—in-place writes and multiple-place writes—incur similar energy consumption costs. We now compare the effectiveness of these two approaches with respect to the error correction rate.

Figure 7 and Figure 8 demonstrate that flash storage reliability improves as we increase the number of repeated writes/places at five different voltage levels (all below the nominal minimum voltage for flash writes).

Experiment: Using our automated testbed, the test platform runs a program that writes zeros to 192 consecutive bytes of flash memory (using *in-place writes* and *multiple-place writes* methods in two different experiments). We increase the maximum number of repeated writes from one to ten, one unit at a time. The monitoring platform counts the number of incorrectly stored bytes (those that are not set to zero after the experiment). The experiment was repeated for five different voltages (1.86 V–1.90 V).

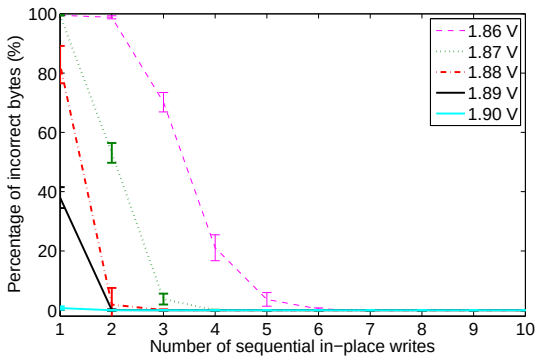


Figure 7: Reliability improvement using *in-place writes* over five different voltages.

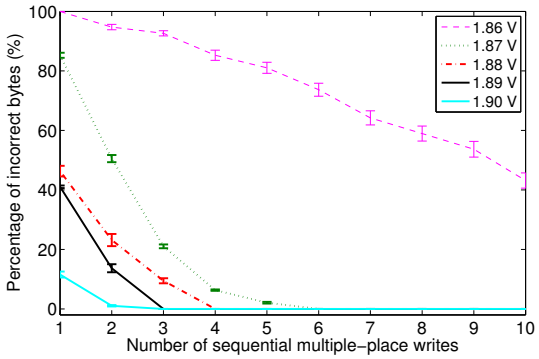


Figure 8: Reliability improvement using *multiple-place writes* over five different voltages.

Figure 9 compares the error rate of the *in-place* and *multiple-place* write methods. We choose the same maximum number of repeated writes for both approaches. As the graph shows, the *in-place writes* method improves the error rate more dramatically. We attribute this phenomenon to the fact that electrons accumulate in flash cells with each programming attempt. Figure 9 also allows us to evaluate hybrids of the *in-place writes* and *multiple-place writes* methods. For example, choosing one place to write the value and repeating the write up to three times (up to three writes in total) works better than repeating the write up to twice in two places (up to four writes in total). This graph offers evidence that a pure *in-place writes* approach works better than a hybrid approach or a pure *multiple-place writes* approach. However, we do not conclude that the *in-place writes* method always outperforms the *multiple-place writes*. A winning case for *multiple-place writes* is when a flash memory has unbalanced blocks (different error rates), for example, the chip shown in Figure 5. While *multiple-place writes* method requires more space, it could provide a more reliable storage compared to *in-place writes*.

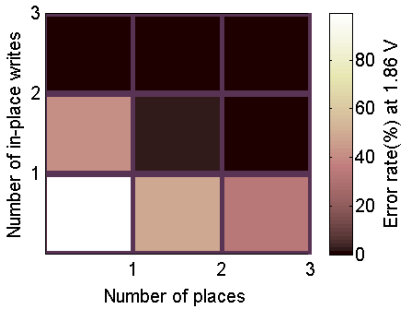


Figure 9: The *in-place writes* method reduces the error rate more effectively than *multiple-place writes* and a hybrid of both methods.

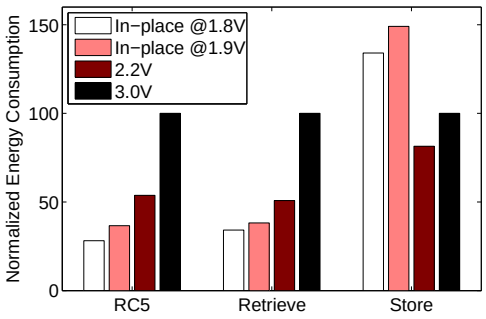


Figure 10: Micro-benchmarks: CPU (`RC5`), read (`retrieve`), and write (`store`) energy consumption measured at four different voltage levels. Although the `RC5` and `retrieve` test cases consume less energy at low voltage, this is not the case for the `store` test case (a write-intensive application) as the savings due to running the chip at low voltage does not compensate for the energy cost required to correct errors.

4.2 *Half-wits* Versus Wits in Practice

To evaluate our storage schemes, we consider three test cases representing CPU operations, flash read operations, and flash write operations.

The `RC5` [30] test case, a CPU-only workload, is a commonly used encryption algorithm that can cope with the resource limitations of low-power devices [8, 18]. `RC5` was implemented with a 32-bit word size, 18 rounds, and 16 bytes of secret key.

The `retrieve` and `store` test cases are both I/O-bound tasks. One reads and the other one writes 192 bytes of data from/to flash memory. CPU-bound operations in these test cases are minimal (essentially only a loop that calls a function to flash memory). The `store` program uses *in-place writes* with a maximum number of three (re)writes to deal with errors. Because flash read operations are fundamentally simpler than flash write operations, flash reads are reliable at low voltage.

We run each of the three test cases on a MSP430F2131 microcontroller at four different voltages that are all in the operating range of this microcontroller (1.8 V–3.5 V). Two voltage levels are below the recommended threshold for flash memory: 1.8 V and 1.9 V. Two voltage levels are at and above the recommended threshold: 2.2 V and 3.0 V. The microcontroller is set to work at its highest possible clock rate for each voltage level in order to gain the best energy performance. Figure 10 compares the average energy consumption over five trials of each test case at each voltage. By running at 1.8 V (below the nominal minimum voltage for flash writes on the MSP430F2131), the microcontroller consumes 48% and 33% less energy to finish the `RC5` and `retrieve` test cases respectively. However, our storage schemes do not seem to be beneficial for flash-write-intensive tasks (the `store` test case).

To evaluate the end-to-end performance of our storage methods, we have tested a sensor-monitoring application that is CPU-intensive and can benefit from a low-voltage storage. This application reads from flash memory 256 accelerometer samples (each ten bits), computes the maximum, minimum, mean, and standard deviation of the samples, and stores the aggregate information in flash memory. This monitoring application is a blend of CPU and I/O, but it is still a CPU-intensive workload. Table 4 shows that providing the system with a low-voltage storage mechanism via our methods helps to decrease the task's total energy consumption by 34%.

4.3 Finding a Crossover Point

We can empirically find the point at which the energy saved on computation compensates for the added cost of repeated flash writes. We compare a workload executed at 2.2 V to the same one running at 1.8 V using the *in-place writes* scheme with the threshold k set to 2. We make the worst-case assumption that all data must be written to flash twice (no bits change on the first attempt). The time spent on flash writes while running at 1.8 V is

Method	In-place 1.8 V	In-place 1.9 V	None 2.2 V	None 3.0 V
Clock rate	6 MHz	6 MHz	8 MHz	14 MHz
Energy(μJ)	270	300	410	760
Time(ms)	151.15	151.32	113.24	64.72

Table 4: Energy consumption and execution time for the accelerometer sensor application. At voltages below the recommended (1.8 V and 1.9 V), *in-place writes* method with a threshold of two is used.

then twice the time spent when operating at 2.2 V. We also assume that the clock rate of the system is set to the highest specified for the CPU at each voltage. Specifically, the clock rate would be set to 6 MHz at 1.8 V and to 8 MHz at 2.2 V.

We empirically determined the power consumption of CPU and flash writes with 1.8 V and 2.2 V voltage supplies. $P_{C\_1.8} = 1.8\ mW$, $P_{C\_2.2} = 3.4\ mW$, $P_{F\_1.8} = 3.7\ mW$, and $P_{F\_2.2} = 5.8\ mW$. The variables T_C and T_F are the time spent in computation and on flash memory respectively. With these assumptions, we can write the following inequality to determine whether a given workload is likely to result in reduced energy consumption:

$$\begin{gathered} Energy_{1.8} \leq Energy_{2.2} \Rightarrow \\ P_{C\_1.8} \times T_{C\_1.8} + P_{F\_1.8} \times k \times T_{F\_1.8} \leq \\ P_{C\_2.2} \times T_{C\_2.2} + P_{F\_2.2} \times T_{F\_2.2} \Rightarrow \\ P_{C\_1.8} \times \frac{8MHz}{6MHz} \times T_{C\_2.2} + P_{F\_1.8} \times k \times \frac{8MHz}{6MHz} \times T_{F\_2.2} \leq \\ P_{C\_2.2} \times T_{C\_2.2} + P_{F\_2.2} \times T_{F\_2.2} \end{gathered}$$

The solution with $k = 2$ is $T_{C\_2.2} \geq 4 \times T_{F\_2.2}$. Therefore, *in-place writes* are competitive over normal flash writes when the time spent on computation is at least four times greater than the time spent on flash writes.

5 Improvements and Alternatives

This section describes several complementary ways to further decrease the energy requirements of our schemes.

Hardware. One could add an adjustable voltage regulator and about a dozen other analog components such that software could toggle a GPIO for discrete dynamic voltage scaling. A feedback loop that dynamically adjusts a voltage supply could help identify the minimum voltage at which no write errors are detected, but such boundaries can vary with temperature and wear-out. Thus, our coding algorithms would remain helpful to cope with potential errors. Our work seeks to avoid hardware modification that would require additional components or design changes to a Printed Circuit Board (PCB) because embedded applications are often cost sensitive. Changing the PCB layout may require a manufacturer to flush its supply chain of parts typically manufactured in high volume. If an inexpensive, software-only approach with minimal disturbance to manufacturing can lead to significant savings in energy consumption, then it is hard to financially justify an expensive hardware approach that offers only comparable performance.

Sign bits and storing complements. As discussed in Section 2.3, one of the major factors influencing the error rate is the Hamming weight of a number. One way to improve the performance of the low-voltage storage methods is to store numbers with greater Hamming weights ($weight \geq 4$) in flash memory. If a number is *lightweight* ($weight < 4$), the complement of the number would be stored and a *sign bit* would be set for future data access. An array of sign bits can be stored separately from the data to avoid disturbing word alignment. A previous work [26] uses a similar technique for multi-level cell (MLC) flash memories with four levels; their techniques result in a significant decrease of energy consumption. Figure 11 shows that using the *sign-bit* scheme decreases the error rate at low voltage for the same ECG data used in Section 2. For this specific example, out of 168 bytes of ECG data, 160 bytes are *overweight* and therefore using the *sign-bit* scheme greatly decreased the error rate. The *sign-bit* approach involves very lightweight computation (counting the number of ones) and increases the number of writes by a factor of one-eighth. Therefore, the effect of this improvement on energy consumption and delay should be comparatively small.

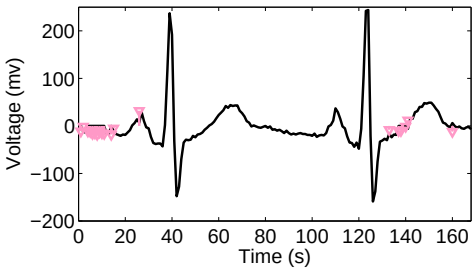


Figure 11: ECG data stored in flash memory at 1.89 V (the same chip from Figure 2) improved by using a sign bit. The light-colored bars show the difference between the ECG stored at low voltage and the original ECG data.

Memory mapping table. Another method to exploit the fact that numbers with greater Hamming weights have a lower probability of error is to map the most frequently used numbers in the user's data to the *heavier* numbers. The solution we suggest is to preprocess the data to sort numbers based on their frequency of use. A simple memory mapping table would map the most frequent numbers to the heaviest numbers. Such a table could be preloaded in flash memory so that storing the table would not consume energy at run time. Use of a memory mapping table would only increase the number of reads and would not increase the number of writes. Therefore, the energy consumption overhead and the delay should be smaller than the *sign bit* method.

An ideal, unrealizable scheme. We initially tried to set the voltage to a level lower than recommended but high enough to avoid errors. This method could not be realized for two reasons: finding a voltage that satisfies this condition requires a large number of experiments per chip—error rate varies chip by chip (Figure 3), and the error rate of flash writes varies depending on its lifespan

and its environment. We found that the byte error rate of MSP430F2131 that is 63% at 1.83 V at 25°C becomes negligible when the temperature goes up to 39°C.

6 Related Work

Storage for low-power embedded devices: Recent research focuses on optimizing use of off-chip flash memory. Off-chip memory allows for special features and larger memories than found on microcontrollers, but introduces additional costs for components. Microhash [38] is a memory index structure tailored for sensor devices with a large external flash memory. Mathur et al. [23] perform an extensive study of available flash memory candidates for sensor devices and demonstrate that an off-chip parallel NAND flash memory decreases the energy consumption of storage. Considering the off-chip NAND flash memory as the best candidate for sensor devices, Agrawal et al. [1] propose a method that allows sensor devices to exploit their flash memory while adapting to different amount of RAM. However, our storage schemes are designed for already deployed low-power devices that use on-chip flash memory. Moreover, while devices at the scale of sensor nodes might switch to block-grained, large off-chip flash memory, RFID-scale platforms might not benefit from this transition because of their challenging resource limitations to drive I/O.

Energy proportionality: Our approaches share the philosophy that energy consumption should scale proportionally to utilization or error rates rather than proportional to a worst-case scenario. Blaauw et al. [6] reduce power consumption by lowering the operating voltage of a pipelined CPU. Certain pipeline stages may produce incorrect computation that require recomputation, but the errors can be made rare to allow better scalability of power consumption. Misailovic et al. [25] demonstrate that the programs whose loops performs fewer iterations cause tolerable errors while their execution time becomes shorter. Weddle et al. [37] introduce PARAID, a scheme that scales power based on the user demand while maintaining the reliability of the system. Their present work also tries to scale power based on the utilization of flash memory without losing storage reliability. Our approaches share this philosophy of scaling performance with utilization. Our performance metric is energy consumption, writes to flash memory represent our utilization, and energy-efficient error correction is our coping mechanism.

Error correction codes for storage: Most previously published flash error correction codes [9, 11, 14] are designed for NAND flash memory. Chen et al. [10] mention that NOR flash normally does not require error correction. These techniques consider neither the asymmetry in low-voltage flash memory nor the resource limitations of low-power embedded devices. Many previous codes [4, 16, 40, 35] leverage the fact that each cell of MLC flash memory represents more than one bit of information. But the fact that single-level cells (SLC) are more suitable for embedded devices, in addition to the occurrence of errors in low-voltage conditions, requires a reconsideration of these codes for SLCs at low voltage. Zemor et al. [39] introduce error-correcting WOM codes for flash memory. They suggest codes that are able to correct up to one error when the flash memory is given enough voltage. This work does not account for errors that occur at low voltage. Godard et al. [12] propose hierarchical code correction and reliability management for NOR flash memory. This work considers on-chip ECCs such as Hamming and parity codes to correct the errors in NOR flash memory.

7 Conclusions and Future Work

The high voltage requirement of on-chip flash memory is a barrier to reducing the total energy consumption of low-power devices. This work examines the main factors affecting the behavior of flash memory at low voltage. Based on our observations of flash memory behavior at low voltage, we proposed three storage schemes—*in-place writes*, *multiple-place writes*, and *RS-Berger codes*—that aim to make flash memory available and reliable at low voltage while tolerating the resource limitations of low-power devices. Our evaluation shows that in-place writes can save 34% of energy consumption for a sensing workload on the MSP430 microcontroller.

Future work includes finding more energy-efficient coding schemes to combat flash write errors caused by low voltage. Currently, the system cannot take full advantage of dynamic voltage scaling. Another plan is to introduce benchmarks for the storage systems of low-power devices. The standard benchmarks used to evaluate the storage systems designed for desktop computers are not immediately applicable to the low-power domain.

Acknowledgments

This material is supported by a Sloan Research Fellowship and the NSF under CAREER Award CCF-0747415, CNS-0627476 (prime), CNS-0627529, CAREER Award CNS-0845874, CNS-0923313, and ECCS-0802107. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

We thank Shane Clark, Wendy Cooper, Marc Liberatore, and Benjamin Ransford for feedback on drafts; Joshua Smith and Alanson Sample at Intel Labs Seattle for providing the WISP over the last three years; and our shepherd Brian Noble and the anonymous reviewers for their detailed feedback and guidance. Portions of this work are patent pending in the United States.

References

[1] D. Agrawal, B. Li, Z. Cao, D. Ganesan, Y. Diao, and P. Shenoy. Exploiting the interplay between memory and flash storage in embedded sensor devices. In *Proceedings of the 16th IEEE Conference on Embedded and Real-time Computing Systems (RTCSA)*, pages 227–236, 2010.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] Atmel AVR Solutions. ATmega128L. http://www.atmel.com/atmel/acrobat/doc2467.pdf.

[4] A. Barg and A. Mazumdar. Codes in permutations and error correction for rank modulation. *IEEE Transactions on Information Theory*, 56(7):3158–3165, 2010.

[5] J. Berger. A note on error detection codes for asymmetric channels. *Information and Control*, 4(1):68–73, 1961.

[6] D. Blaauw and S. Das. CPU, heal thyself. *IEEE Spectrum*, 46(8):40–56, 2009.

[7] M. Buettner, B. Greenstein, A. Sample, J. R. Smith, and D. Wetherall. Revisiting smart dust with RFID sensor networks. In *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, October 2008.

[8] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist cryptography and computation on the WISP UHF RFID tag. In *Proceedings of the Conference on RFID Security*, July 2007.

[9] B. Chen, X. Zhang, and Z. Wang. Error correction for multi-level NAND flash memory using Reed-Solomon codes. In *IEEE Workshop on Signal Processing Systems (SiPS 2008)*, pages 94–99, Oct. 2008.

[10] S. Chen. What types of ECC should be used on flash memory? Application Note for SPANSION, 2007.

[11] M. Fujino and V. Moshnyaga. An efficient Hamming distance comparator for low-power applications. In *9th International Conference on Electronics, Circuits and Systems*, volume 2, pages 641–644, 2002.

[12] B. Godard, J.-M. Daga, L. Torres, and G. Sassatelli. Hierarchical code correction and reliability management in embedded NOR flash memories. In *Proceedings of the 2008 13th European Test Symposium*, pages 84–90, 2008.

[13] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000. Circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215.

[14] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli. On-chip error correcting techniques for new-generation flash memories. *Proceedings of the IEEE*, 91(4):602–616, April 2003.

[15] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1731–1735, 2008.

[16] A. Jiang, M. Schwartz, and J. Bruck. Correcting charge-constrained errors in the rank-modulation scheme. *IEEE Transactions on Information Theory*, 56(5):2112–2120, 2010.

[17] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for "Smart Dust". In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 271–278, 1999.

[18] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[19] T. Klove. Error correcting codes for the asymmetric channel. Technical report, Informatics, University of Bergen, 1995.

[20] B. P. L. Lo, S. Thiemjarus, R. King, and G. zhong Yang. Body sensor network - a wireless sensor platform for pervasive healthcare monitoring. In *Adjunct Proceedings of the 3rd International Conference on Pervasive Computing (PERVASIVE)*, pages 77–80, 2005.

[21] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[22] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[23] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *Proceedings of the 5th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 374–381, 2006.

[24] Microchip. 32-bit PIC MCUs. http://www.microchip.com/en_US/family/pic32/.

[25] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 25–34, 2010.

[26] V. Papirla and C. Chakrabarti. Energy-aware error control coding for flash memories. In *Proceedings of the 46th Annual Design Automation Conference (DAC)*, pages 658–663. ACM/EDAC/IEEE, 2009.

[27] P. Pavan, R. Bez, P. Olivo, and E. Zanoni. Flash memory cells-an overview. *Proceedings of the IEEE*, 85(8):1248–1271, Aug 1997.

[28] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[29] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[30] R. L. Rivest. The RC5 encryption algorithm. In B. Preneel, editor, *Fast Software Encryption*, pages 86–96. Springer, 1995. (Proceedings Second International Workshop, Dec. 1994, Leuven, Belgium).

[31] R. L. Rivest and A. Shamir. How to reuse a write-once memory. *Information and Control*, 55:1–19, 1982.

[32] M. Salajegheh, S. Clark, B. Ransford, K. Fu, and A. Juels. CCCP: Secure remote storage for computational RFIDs. In *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[33] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. Design of an RFID-based battery-free programmable sensing platform. In *IEEE Transactions on Instrumentation and Measurement*, 2008.

[34] V. Shnayder, B.-r. Chen, K. Lorincz, T. R. F. F. Jones, and M. Welsh. Sensor networks for medical care. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 314–314, 2005.

[35] I. Tamo and M. Schwartz. Correcting limited-magnitude errors in the rank-modulation scheme. *IEEE Transaction on Information Theory*, 56(6):2551–2560, 2010.

[36] Texas Instruments Incorporated. MSP430 Ultra-Low Power Microcontrollers. http://www.ti.com/msp430.

[37] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. *ACM Transactions on Storage (TOS)*, 3(3):Article 13:1–33, 2007.

[38] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar. Microhash: An efficient index structure for fash-based sensor devices. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, pages 31–44, 2005.

[39] G. Zemor and G. D. Cohen. Error-correcting WOM-codes. *IEEE Transactions on Information Theory*, 37(3):730–734, May 1991.

[40] F. Zhang, H. D. Pster, and A. Jiang. LDPC codes for rank modulation in flash memories. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2010.