Saving Time and Energy Using Partial Flash Memory Operations in Low-Power Microcontrollers

Prawar Poudel and Aleksandar Milenković
The University of Alabama in Huntsville, Huntsville, AL USA.
E-mail: {pp0030, milenka}@uah.edu

Abstract

This paper introduces a technique that reduces time and energy consumed by critical flash memory operations in ultra-low-power microcontrollers. The proposed technique utilizes partial or aborted flash memory erase and program operations that proved to have no negative impacts on accuracy and longevity of information stored in the flash memory. Our experimental evaluation performed on a family of microcontrollers shows that the proposed technique can save 98% of the energy consumed for flash erase operations and up to 75% for flash program operations.

Keywords

Flash memory, Low-power electronics, and Energy measurements.

1. Introduction

Energy efficiency is a key design requirement for many resource-constrained embedded systems commonly used in wireless sensor networks, wearable or implanted electronics, and IoTs. Increased energy efficiency enables the design of systems that have a smaller form factor, longer operating times, and reduced operating costs by eliminating the need for costly battery changes. For example, battery changes in implanted devices may require surgical procedures. Modern ultra-low-power microcontrollers are typically used in these systems to perform sensing, processing, communication, and actuation tasks.

Ultra-low-power microcontrollers typically integrate a processor core, flash memory, RAM memory, hardware accelerators, and a range of input/output peripheral interfaces (e.g., ADC, DAC, ports, timers, communication) on a single chip. The embedded flash memory serves as a non-volatile storage that contains system firmware and constants and behaves as a read-only memory during normal operation. However, to meet demands for frequent firmware updates or to prevent loss of critical application data in case of a power loss (e.g., in wearable health monitors), modern microcontrollers often include flash memory controllers that enable in-system flash erase and program operations. These operations can be initiated from within the system, rather than through external JTAG interfaces. Unfortunately, these operations are power hungry as they rely on internal charge pumps to generate high voltages needed to move charges to/from floating gates within flash memories. Thus, finding a way to minimize energy consumed by these operations is very beneficial for systems that are frequently updated or use internal flash memory for storing application critical data.

To address high energy demands of flash memory operations in ultra-low-power microcontrollers, Salajegheh et al. [1], [2] proposed an energy-saving technique. It utilizes reduced operating voltages that are at the level where the processor core will work, but standard flash operations may not work reliably. To remedy possible loss of information, they employ one of the following: (a) repeated in-place write operations, (b) multiple place write operation, or (c) RS-Berger coding of data. They report energy savings for in-place write operations of up to 34%. Similarly, a study by Tseng et al. [3] shows that up to 45% of energy consumed could be saved using dynamic voltage scaling controlled based on the flash operation being performed. Papirla et al. [5] find that energy required by flash write operations heavily depends on data patterns. Thus, they propose an encoding scheme that minimizes the frequency of power hungry bit patterns in codewords (‘10’ and ‘01’), reducing the total energy of flash write operations for up to 34%. Nath proposes a lazy amnesic compression based technique for storing data in flash memories [6]. The required energy for flash write operations is reduced by using a lossy compression; the compression ratio is adjusted based on age of data, i.e., fidelity of “old” data is lower than the fidelity of new data. Mathur et al. introduce Capsule [7], a log-structured object storage system for flash memories that supports fine-grain allocation of space for storage objects such as streams, files, arrays, queues. Though these techniques demonstrate significant potential in reducing the total energy consumed, they introduce extra overhead in time, compute resources, and/or memory space.

In this paper we propose an alternative technique for reducing both time and energy consumed by in-system flash operations at nominal voltage that does not require any additional memory overhead. First, we motivate the proposed technique by characterizing the behavior of an embedded NOR flash memory when subjected to partial erase and program operations. Here, the ongoing operations are prematurely aborted before their completion and the state of the erased/programmed segments is observed. We find that “there is a plenty room at the bottom” and that flash memory operations can be safely aborted prematurely without sacrificing accuracy of information in the flash memory.

The experimental evaluation is performed on a family of TI MSP430 microcontrollers. It involves comparing the time and energy consumed by reference or nominal flash memory operations and by the proposed partial flash operations that maintain accuracy requirements, while varying data patterns. The energy profiling is conducted using a setup for automated measurement of energy consumed by embedded computing systems [8]. The results of the experimental evaluation show that more than 98% of energy can be saved for costly flash erase operations, whereas up to 75% of energy can be saved for flash program operations.

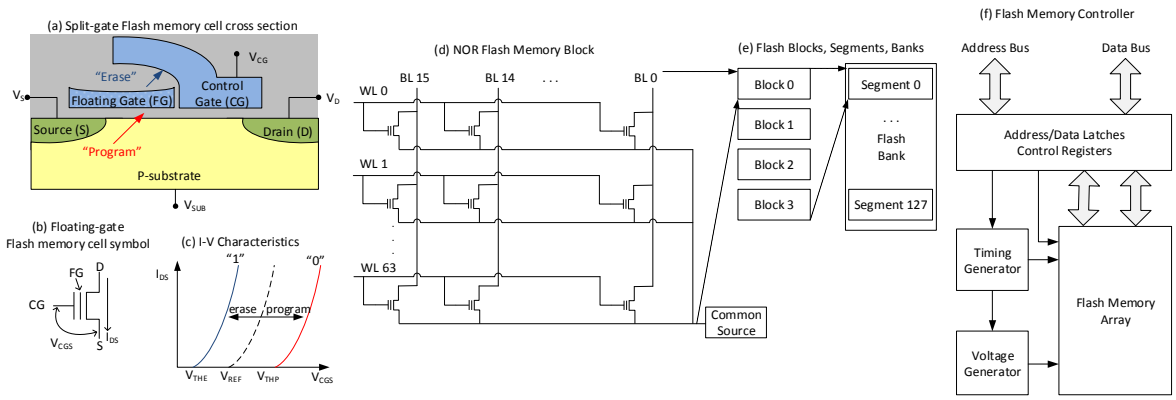

Figure 1. (a) Cross section of a split gate flash memory cell. (b) Floating gate transistor symbol. (c) I-V characteristic for a flash memory cell. (d) A NOR flash memory block organization. (e) Flash memory organized in blocks, segments and banks. (f) Block diagram of a flash memory controller.

The main contributions of this paper are as follows:

- It characterizes the flash memory behavior when subjected to partial erase and partial program operations and determines the partial erase (T_{PE}) and partial program times (T_{PP}) that offer error free flash operations;
- It evaluates the effectiveness of the partial erase and program operations in saving time and energy as a function of data written and aging induced changes in flash memory characteristics.

The rest of the paper is organized as follows. Section 2 gives a brief background discussing principles of NOR flash memory organization and operation. Section 3 discusses a flash memory characterization and how it motivates this work. Section 4 describes the experimental environment, including the platform, experiments, and measuring setup. Section 5 describes the results of the experimental evaluation and Section 6 concludes the paper.

2. Background

Flash Cell. Flash memory is composed of an array of flash memory cells. Each memory cell consists of a MOSFET (Metal Oxide Semiconductor Field Effect) transistor with an additional floating gate (FG). The floating gate is sandwiched between the conductive channel formed in the substrate and the control gate (CG) and is isolated from the transistor terminals by oxide layers. Figure 1(a) shows a cross section of a split-gate memory cells that is used as a building block in flash memories of interest for this study. Here a portion of the control gate lies directly on top of the substrate and the remaining part lies on top of the floating gate. A flash memory cell stores one bit of information in the form of charges placed on the floating gate. The presence of negative charges increases the transistor's threshold voltage (V_{THP}), which is equivalent to logic 0 (or programmed state), whereas their absence is equivalent to logic 1 (erased state), as shown in Figure 1(c).

Flash Organization and Operation. NOR flash memories are typically used in microcontrollers due to their low standby power, fast reads, high reliability, and random access through full address and data buses [9]. However, they have lower storage capacity, lower density, and longer erase and program times than NAND flash memories that are designed for high-capacity and low-cost storage solutions, but do not provide random access. Figure 1(d) shows organization of a NOR flash memory block organized in 64 addressable words, each 16-bit wide. A word line (WL) connects control gates of all flash cells in a row, whereas bit lines (BLs) connect all drain terminals in a column. The source terminals of all the cells in a block are connected together. Multiple flash blocks form a segment (4 in our case shown in Figure 1(e)), and multiple segments create a flash memory bank [9]. Thus, a segment in our example contains 256 words or 4096 bits.

NOR flash memories support three basic operations, read, program, and erase. Read and program operations are performed on a byte or a word level granularity, whereas flash erase operation takes place on a complete flash memory segment. To reprogram (write) new data into a word that requires a programmed bit be changed into erased bit (changing logic 0 to a logic 1), its corresponding segment needs to be erased.

Program and erase operations involve transport of electrons through the tunnel and blocking oxide layers as shown in Figure 1(a). They require high voltages and use channel hot electron injection for programming and Fowler-Nordheim tunneling for erasing. To program a flash cell, a high voltage is applied to its source terminal ($V_S \sim 10V$, $V_{CG} \sim 2V$, $V_D \sim 0.5V$) inducing hot carrier injection that places electrons on the floating gate ("Program" arrow). To erase a flash cell, a large positive voltage is applied on the control gate ($V_{CG} \sim 12V$, $V_S = V_D = 0V$) to remove electrons from the floating gate [9], [10].

A flash read operation involves bringing a read voltage on the selected word line and a sense voltage on the bit lines. The flash cells that are in the erased state will conduct the current and the cells in the programmed state will not (Figure 1(c)).

Flash Memory Controller. A flash memory embedded in an MCU typically contains code and constants and its default operating mode is read only. To utilize in-system flash program and erase operations, microcontrollers interact with the flash memory through its controller. Figure 1(f) shows a block diagram of a typical flash memory controller. In addition to the flash memory banks, it includes a voltage generator to generate voltages needed for program and erase

operations, timers to control their duration of flash operations, as well as address and data latches.

Programmers write into flash controller registers to initiate flash program and erase operations. During these operations the processor is halted because the flash memory is locked. Alternatively, the processor may continue program execution from the RAM memory, providing it does not access the currently locked flash memory bank. A typical program or erase cycle includes the time to generate required voltages, the time to perform the operation, and the time to remove required voltages. For the microcontroller used in this study the nominal word programming time is T_{PROG}=64-85 µs and the nominal segment erase time is T_{ERASE}=23-35 ms [9]. The flash operations can be aborted by setting an emergency exit bit (EMEX) in the control register by a program running from the RAM memory or a different flash memory bank.

3. Flash Memory Characterization

To extract physical characteristics of the embedded flash memory of interest, we design a set of experiments that utilize the emergency exit feature.

Code 1 describes steps carried out to characterize partial flash erase operations. A flash memory segment at the address *segaddr* is first erased and then fully programmed so that all bits are set to logic 0. Next, an erase operation is initiated, but it is prematurely aborted after a period of time called partial erase time or t_{PE}. The flash memory segment is then characterized as follows. It is repeatedly read *N* times and the state of individual flash cells is recorded. Flash cells that read as logic 0 *N* times are called *stable programmed cells* (stable0s), cells that read as logic 1 *N* times are called *stable erased cells* (stable1s), and cells that sometimes read as logic 0 and sometimes as logic 1 are called *unstable cells* (4,096 – stable1s – stable0s). The experiment is repeated by varying the partial erase time t_{PE} from 0 to the nominal erase time (T_{ERASE}) with resolution of a single clock cycle.

Code 1. Algorithm for partial erase characterization

Partial_Erase_Characterization (Segaddr, T_{ERASE})

```
1. for tPE from 0 to TERASE:
2.   Erase the entire segment (read as all 1s)
3.   Program all words in the segment (read as all 0s)
4.   Initiate the segment erase operation
5.   Wait for tPE
6.   Abort the erase operation
7.   Call Characterize(segaddr, N, stable1s, stable0s)
8. end for
```

Characterize(Segaddr, N, stable1s, stable0s)

```
1. stable1s = 0
2. stable0s = 0
3. for each word in the segment:
4.   Read the word N times
5.   Characterize each bit as always1, always0
6.   for each bit in the word:
7.     if (always1): stable1s += 1
8.     else (always0): stable0s+=1
9.   end for
10. end for
```

A similar experiment is designed to characterize partial program operations (write a word). However, to initiate a partial flash program operation, the flash segment is first fully erased. The program operation is aborted after a period of time called partial program time or t_{PP}. The experiment is repeated by varying t_{PP} from 0 to the nominal program time (T_{PROG}) with resolution of a single clock cycle.

Figure 2(a) and Figure 2(b) show the results of the partial erase characterization and the partial program characterization, respectively. The x-axes show the partial erase and program times (t_{PE} and t_{PP}). The red lines represent the number of stable erased cells (stable1s), the blue lines represent the number of stable programmed cells (stable0s), and the green lines represent the number of unstable cells.

Let us first analyze the results shown in Figure 2(a). For small partial erase times (t_{PE} ~ 0 µs) the erase operation is aborted promptly and expectedly all 4,096 cells in the segment are characterized as stable 0s. As we increase the partial erase time, the number of stable 0s starts decreasing and the number of stable 1s starts increasing. The plot shows that majority of cells change their state in a very narrow time window. Although the nominal erase time per specification ranges between 23-32 ms, we find that almost all flash cells are in the erased state after just 50 µs. Thus, the partial erase time of t_{PE}=50 µs appears to be sufficient to completely erase the flash segment.

One may argue that though flash cells appear to be erased, their threshold voltage may not quite correspond to the nominal V_{THE} as shown in Figure 1(c), that is, they may be in a weak erased state. Fortunately, the flash controller supports

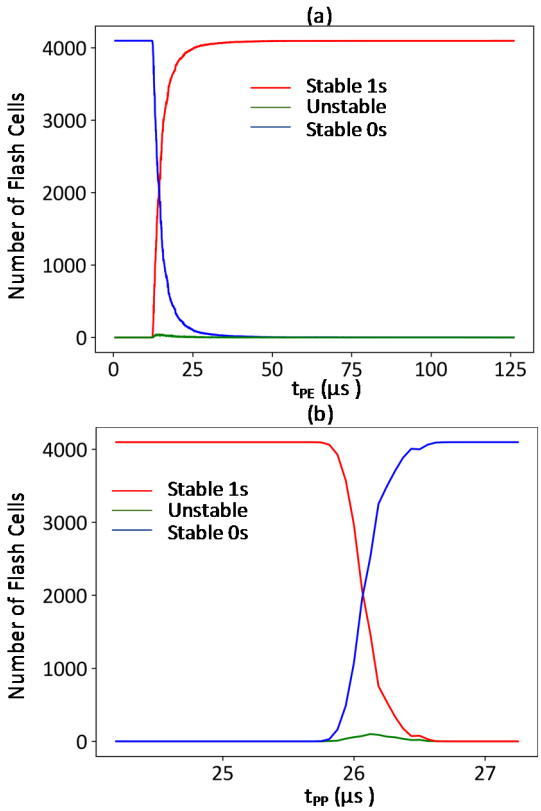

Figure 2. State of the flash segment cells as a function of the partial erase time (a) and the partial program time (b).

so-called marginal read operations that allows us to identify whether erased cells are in the weak or strong erased or programmed states. In spite of t_{PE} being significantly shorter than the nominal erase time, the marginal reads do not report weakly erased states (in this case when $t_{PE} \geq 50$ µs).

Similarly, Figure 2(b) shows that for small partial program times ($t_{PP} \sim 0$ µs) all segment bits remain in the erased state (the red lines is at 4,096). As we increase the partial program time, the number of erased cells starts decreasing and the number of programmed cells starts increasing. For the partial program time $t_{PP} \geq 27$ µs all cells are in the programmed state, though the nominal programming time ranges between 64 and 85 µs. To verify that transition of flash cells is complete marginal reads are used to identify potentially weakly programmed cells.

Based on these observations *we propose to use partial erase and partial program flash operations* instead of nominal erase and program flash operations, respectively. Using this approach, we can reduce the execution time of programs that require frequent in-system flash operations as well as reduce the energy consumed by these tasks. For this approach to be successful, we need to find partial erase times and partial program times that can be applied across different flash memory segments and different chips from the same family of microcontroller, without loss of accuracy of information stored.

4. Experimental Evaluation

4.1. Experimental Setup

Our experiments are based on the Texas Instruments' MSP430F5438 and MSP430F5529 family of ultra-low-power microcontrollers. They integrate a 16-bit processor core (20 bit with extended architecture), RAM memory, flash memory, a direct memory access controller, highly configurable clock subsystem, and a range of analog and digital peripherals. The in-system programmable embedded flash memory consists of multiple 64 KB banks and flash operations can be initiated from within a flash bank or from a RAM memory. Flash memory operations can be aborted by setting an emergency exit bit if the program is run from the RAM memory. Marginal read modes can help identify weekly programmed or erased flash memory cells [11].

To start a flash erase operation, an ERASE control bit in the control register is set. Any dummy write operation to an address space of the segment to be erased triggers the erase operation. The BUSY bit indicates that the flash memory is currently in use and it is lowered when the operation is finished. To start a program (write) operation, a control bit WRT is set in a flash controller register. A byte, word, or double word write at the desired address triggers a flash program operation. The BUSY bit is asserted and remain active until the flash program operation is finished.

4.2. Test Programs

To evaluate effectiveness of the proposed partial erase and program operations we develop test programs that are used for time and energy profiling as shown in Code 2 and Code 3.

Code 2 shows pseudo-code for the proposed segment partial erase procedure. The nominal segment erase includes steps 1, 2, 3, and 6. In the partial erase operation, instead of waiting for operation to be completed in step 6, we rather wait for the $T_{PE} < T_{ERASE}$ period and then set the EMEX bit in the flash control register to abort the current operation. This parameter T_{PE} is determined by the characterization procedure described in the previous section.

Code 3 shows pseudo-code for the proposed segment partial program procedure. Each word in the segment is partially programmed using steps 4-8. Again, unlike the nominal program procedure that waits for the operation to be completed, here we wait for the partial program time $T_{PP} < T_{PROG}$ before setting the EMEX bit to abort the current operation. Before proceeding to program the next word, we wait for the flash controller to come to the default state and update the current address.

4.3. Energy Profiling Setup

To profile energy consumed, the development board with the MSP430 microcontroller is connected to a setup for runtime energy profiling that consists an NI PXIe-4154 battery simulator, an NI PXIe-6361 data acquisition card, and a workstation [8]. The battery simulator supplies the power to the development board and samples the current drawn (I_S) with a sampling frequency of F_S = 100,000 Hz. The test program execution on the development board is synchronized with the collection of current samples at the workstation. The energy consumed as calculated as shown in Eq. (1), where M is the number of current samples collected during execution of the test procedures, V is power supply, and $\Delta t = 1/F_S$.

$$E = \sum_{i=0}^{M} V \cdot I_S \cdot \Delta t \quad (1)$$

Code 2. Segment partial erase procedure

```
Segment_Partial_Erase(Segaddr, TPE)
1.  Wait while BUSY
2.  Set ERASE bit
3.  Dummy write into the segment
4.  Wait for TPE time
5.  Set EMEX
6.  Wait while BUSY
```

Code 3. Segment partial program procedure.

```
Segment_Partial_Program(Segaddr, Data, TPP)
1.  Erase segment at Segaddr
2.  Caddr=Segaddr
3.  for each word in the segment:
4.    Set WRT bit
5.    Write Data at Caddr;
6.    Wait for TPP
7.    Set EMEX
8.    Wait while BUSY
9.    Caddr = Caddr + 2
10. end for
```

5. Results

This section describes the results of the experimental evaluation aimed at finding suitable partial erase and partial program times that will work across multiple chips and segments (5.1), quantifying energy savings due to proposed flash operations (5.2), and evaluating impact of flash memory aging on the proposed technique (5.3).

5.1. Partial Operations Characterization

The process described in Section 3 is used to characterize flash memory behavior in presence of partial erase and program operations. Table 1 shows the results of such characterization performed on four MSP430F5438 chips (Chip 0 – Chip 3) and five segments within each chip (Seg 0 – Seg 4). Each segment in each chip is profiled to determine its parameter t_{PP} that corresponds to the minimum partial program time when all bits within a word that need to be programmed are indeed programmed. Similarly, the column t_{PE} records the minimum partial erase time when all bits are indeed erased.

Table 1. Results of flash memory segment characterization.

MSP430F5438		t_{PP}	Optimal	Nomial	MSP430F5438		t_{PE}	Optimal	Nominal
Chip	Seg	(µs)	(T_{PP})	Time	Chip	Seg	(µs)	(T_{PE})	Time
0	0	27			2	0	113		
0	1	27			2	1	57		
0	2	26			2	2	57		
0	3	27			2	3	80		
0	4	27			2	4	57		
1	0	26	28 µs	64-85 µs	3	0	46	115 µs	23-32 ms
1	1	26			3	1	34		
1	2	26			3	2	34		
1	3	26			3	3	35		
1	4	26			3	4	35		

We find that the partial program times (t_{PP}) are fairly uniform across different segments and chips, ranging between 26 µs and 27 µs, which is significantly lower than the nominal program time that is 64-85 µs per specification and ~65 µs measured in our experiments. Thus, we find that roughly 1/3rd of the nominal time is sufficient to complete programming operation. However, we take a conservative approach and use 28 µs as optimal partial program time (T_{PP}) in further experiments.

The results in Table 1 show the partial erase times (t_{PE}) vary across different segments within a chip and across different chips. For example, they range from 34-46 µs in Chip 3 and from 57 to 115 µs in Chip 2. Still, these times that mark the earliest time when all bits within a segment are indeed erased are significantly lower than the nominal time that is 23-32 ms per specification and ~27 ms measured in our experiments. We choose an optimal partial erase time T_{PE} to be 115 µs in further experiments.

Similar characterization experiments are performed on a different microcontroller from the MSP430 family of microcontrollers. Using TI's MSP430F5529 microcontroller we find that the optimal partial programming time for a word, T_{PP}, is 16 µs (the nominal programming time T_{PROG} is 65-85 µs), whereas the optimal partial erase time for a segment, T_{PE}, is 73 µs (the nominal time is 23-32 ms). These results confirm that using partial program and erase operations can be used to replace the nominal flash operations.

5.2. Energy Profiling

Figure 3 shows the current profile for the nominal (blue lines) and partial flash operations (orange lines). The microcontroller draws ~1 mA when idle. The start of a flash erase operation is marked by a steep increase in the current that reaches a level slightly below 4 mA. The current remains high for the duration of the flash erase operation – in this experiment the measured nominal erase time T_{ERASE} ~ 27 ms. The red line shows the current profile for the equivalent partial erase time with T_{PE}=115 µs. The amplitude of the current reaches the level observed in the nominal operation. The total energy consumed by the nominal flash segment erase operation is on average ~258.6 µJ, whereas the total energy consumed for the partial erase operation is 3.3 µJ. Thus, the partial erase operation saves over 98% the energy consumed for the nominal erase operation.

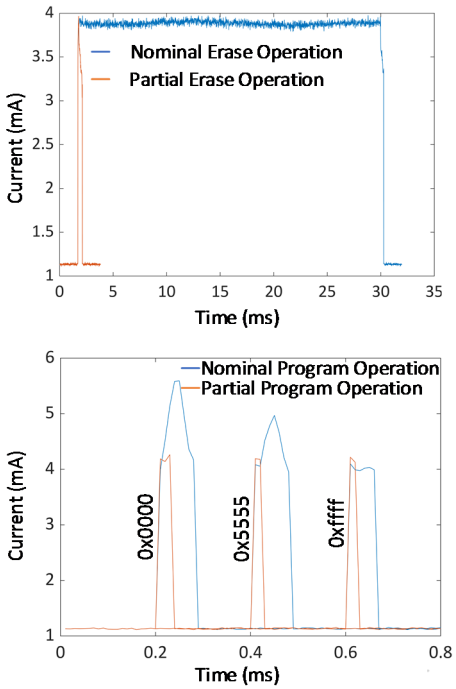

Figure 3. Current drawn by MSP430F5438 during nominal and partial erase and program operations.

Figure 3, bottom, shows the current drawn during nominal and partial program operation of a 16-bit word with three different data patterns: 0xFFFF means that no bit will actually get programmed to logic 0; 0x5555 means that a half bits in the word will get programmed, and 0x0000 means that all bits within a word will be programmed. The current profiles are data dependent, more so for the nominal word programming operations. Table 2 shows the energy consumed in µJ and percentage of energy savings achieved by using the partial program operations. The table shows the results for a single-word programming operation and when an entire segment is programmed. It should be noted that the nominal segment programming utilizes an optimized block-wide programming operation. In this mode the flash memory controller voltage generators remain active and data words are streamed one after the other. This mode cannot be used when utilizing emergency exit operation. However, in spite of that, the proposed partial program operations still result in energy savings that range from 24% when writing 0xFFFF to 64% when writing 0x0000.

Table 2. Energy consumed for word write and segment write with different data.

	Energy Consumed (µJ)		Savings (%)		Energy Consumed (µJ)		Savings (%)
Word Programmed	Nominal Program	Partial Program		Segment Programmed	Nominal Program	Partial Program	
0x0000	9	3	66.7	0x0000	127	45.5	64.2
0x5555	8	2	75	0x5555	92.3	43.6	52.8
0xffff	5	2	60	0xffff	53.9	40.8	24.3

5.3. Stress Analysis

We showed above that the flash cells transition into erased and programmed states much earlier than it is nominally expected. However, our experiments are conducted at room temperature and under nominal power supply of V_S=3.3 V. The flash memory segments used in characterization were relatively fresh, i.e. not exposed to wear-out.

One important concern is related to the robustness of the proposed technique in presence of segment aging. NOR flash memories can typically sustain up to 100,000 program-erase cycles (PE) before they may permanently fail. Will the optimal partial erase and program times found in our characterization study work on aged flash memory segments?

To explore robustness of the parameters found through characterization, we stress selected flash memory segments by performing repeated program-erase (PE) operations. We perform 10,000 PE cycles and then the characterization is conducted to determine the optimal partial program and partial erase times. The process is then repeated until the maximum endurance of the flash memory segment is reached.

Figure 4 (a, b) show the optimal partial erase time as a function of the segment stress levels (0 – 100 K PE cycles) for two different MSP430F5438 chips and multiple segments within each chip (each line plots T_{PE} for a particular segment). The results indicate that this parameter is indeed affected by the number of PE cycles and it increases with an increase in the segment stress level. Thus, on average the optimal partial erase time ranges from below 100 µs for unstressed flash

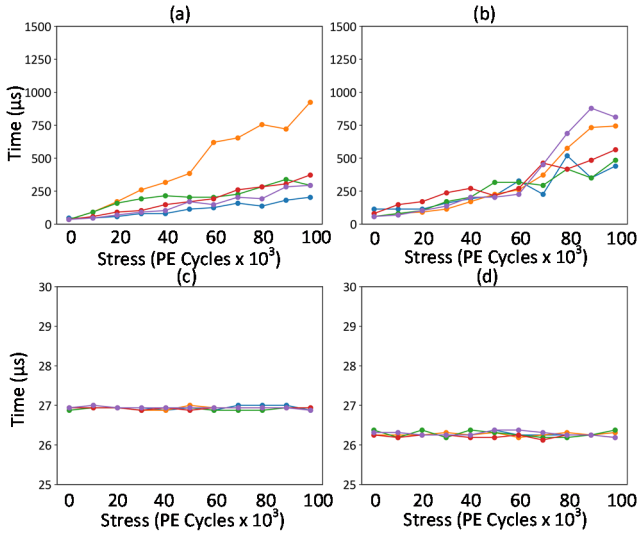

Figure 4. Partial erase (a, b) and partial program times (c, d) as a function of the stress level for MSP430F5438.

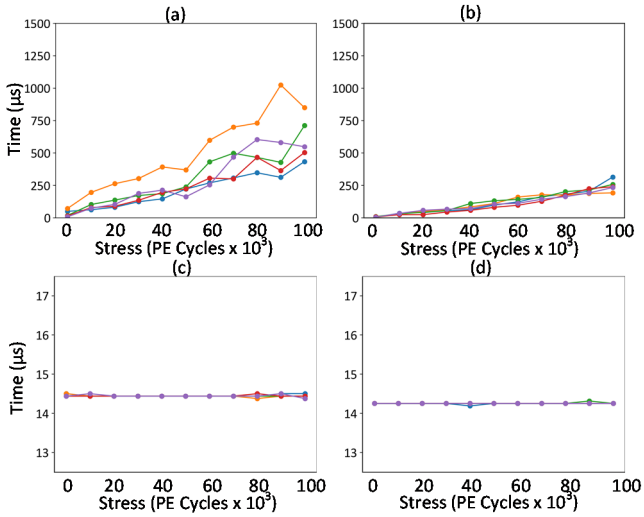

Figure 5. Partial erase (a, b) and partial program times (c, d) as a function of the stress level for MSP430F5529.

memory segments to below 925 µs for flash segments exposed to 100K stress cycles. However, even this worst-case partial erase time is a way smaller than the nominal segment erase time. Based on this analysis, the partial erase segment may take the segment age into account to adjust the partial erase time.

Figure 4 (c, d) show the optimal partial program time as a function of the segment stress levels for two different MSP430F5438 chips and multiple segments within each chip (each line plots T_{PP} for a particular segment). The plots here indicate that the optimal partial program time is not significantly affected by the stress level and it shows a slight decline as we increase the stress level. Using the optimal partial program time of 28 µs will work regardless of the segments' stress level.

This analysis is repeated for two MSP430F5529 chips. Figure 5 shows the plots for the optimal partial erase times (top) and the optimal partial program times (bottom). It shows similar trends – the worst-case partial erase time is ~850 µs and the worst-case partial program time is 16 µs.

6. Conclusions

Energy-efficiency have become a first-class design parameter in many emerging applications ranging from wearable electronics, mobile computing, to wireless sensor networks. In this paper we analyze an ultra-low-power microcontroller with an in-system programmable NOR flash memory that contains program and data. We characterized flash program and erase operations and found that these operations require less time than nominally required.

We introduce partial erase and program flash operations that abort them at an opportune time that guarantees no loss of information. The proposed flash operations are implemented and evaluated on a commercial microcontroller. We demonstrate that they can provide significant saving in energy of over 98% for segment erase operations and from 24-64% for segment program operations. We show how flash stress level impacts the parameters of interest for the proposed

technique. This technique does not require any hardware changes and can be solely implemented in firmware.

Future work will focus on further analysis on how power supply and environmental conditions may impact the optimal partial erase and program times and how the proposed technique can be implemented in other types of flash memories.

7. References

[1] M. Salajegheh, Y. Wang, K. Fu, A. Jiang, and E. Learned-Miller, "Exploiting Half-wits: Smarter Storage for Low-power Devices," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, Berkeley, CA, USA, 2011, pp. 55–68.

[2] M. Salajegheh, Y. Wang, A. (Andrew) Jiang, E. Learned-Miller, and K. Fu, "Half-Wits: Software Techniques for Low-Voltage Probabilistic Storage on Microcontrollers with NOR Flash Memory," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 1–25, May 2013, doi: 10.1145/2465787.2465793.

[3] H.-W. Tseng, L. M. Grupp, and S. Swanson, "Underpowering NAND flash: Profits and perils," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[4] H.-W. Tseng, L. M. Grupp, and S. Swanson, "Underpowering NAND flash: profits and perils," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, Austin, Texas, 2013, pp. 1–6, doi: 10.1145/2463209.2488935.

[5] V. Papirla and C. Chakrabarti, "Energy-aware error control coding for Flash memories," in *2009 46th ACM/IEEE Design Automation Conference*, 2009, pp. 658–663, doi: 10.1145/1629911.1630085.

[6] S. Nath, "Energy efficient sensor data logging with amnesic flash storage," in *2009 International Conference on Information Processing in Sensor Networks*, 2009, pp. 157–168.

[7] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Capsule: an energy-optimized object storage system for memory-constrained sensor devices," in *Proceedings of the 4th international conference on Embedded networked sensor systems - SenSys '06*, Boulder, Colorado, USA, 2006, p. 195, doi: 10.1145/1182807.1182827.

[8] A. Dzhagaryan, A. Milenković, M. Milosevic, and E. Jovanov, "An environment for automated measurement of energy consumed by mobile and embedded computing devices," *Measurement*, vol. 94, pp. 103–118, Dec. 2016, doi: 10.1016/j.measurement.2016.07.073.

[9] P. Poudel, B. Ray, and A. Milenkovic, "Microcontroller TRNGs Using Perturbed States of NOR Flash Memory Cells," *IEEE Transactions on Computers*, pp. 1–1, 2018, doi: 10.1109/TC.2018.2866459.

[10] A. R. Duncan, M. J. Gadlage, A. H. Roach, and M. J. Kay, "Characterizing Radiation and Stress-Induced Degradation in an Embedded Split-Gate NOR Flash Memory," *IEEE Transactions on Nuclear Science*, vol. 63, no. 2, pp. 1276–1283, Apr. 2016, doi: 10.1109/TNS.2016.2540803.

[11] "MSP430x5xx and MSP430x6xx Family User's Guide." Texas Instruments Incorporated, Jun-2008.